aws

Guide du développeur

# AWS Lambda

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

# AWS Lambda: Guide du développeur

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Les marques commerciales et la présentation commerciale d'Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d'une manière susceptible d'entraîner une confusion chez les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d’Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

# Table of Contents

Qu'est-ce que c'est AWS Lambda ? ..... 1
Cas d'utilisation de Lambda ..... 1
Fonctions principales ..... 2
Créez votre première fonction ..... 4
Prérequis ..... 4
Créer une fonction Lambda à l'aide de la console ..... 6
Invoquer la fonction Lambda à l'aide de la console ..... 13
Nettoyage ..... 16
Ressources supplémentaires et prochaines étapes ..... 17
Créez votre première application ..... 19
Prérequis ..... 21
Téléchargement des exemples de fichiers d'application ..... 21
Déploiement de l'application ..... 29
Déployez les ressources manuellement ..... 29
Déployez les ressources à l'aide de AWS SAM ..... 38
Tester l'application ..... 39
Tester l'application manuellement ..... 40
Test de l'application avec le script automatique ..... 42
Étapes suivantes ..... 43
Concepts clés de Lambda ..... 45
Fonction ..... 45
Déclencheur ..... 45
Événement ..... 46
Environnement d’exécution ..... 46
Package de déploiement ..... 47
Environnement d’exécution ..... 47
Couche ..... 47
Concurrency ..... 48
Qualificateur ..... 48
Destination ..... 49
Infrastructure en tant que code (IaC) ..... 50
Outils de l’IaC pour Lambda ..... 50
Démarrage avec l’IaC pour Lambda ..... 51
Prérequis ..... 52

Création d'une fonction Lambda ..... 52
Afficher le AWS SAM modèle correspondant à votre fonction ..... 53
AWS Infrastructure Composer À utiliser pour concevoir une application sans serveur ..... 55
Déployez votre application sans serveur à l'aide de AWS SAM (facultatif) ..... 61
Testing de votre application déployée (facultatif) ..... 63
En utilisant le AWS CDK ..... 64
Prérequis ..... 64
Étape 1 : Configurer votre projet ..... 64
Étape 2 : définir la pile ..... 66
Étape 3 : Création de la fonction ..... 70
Étape 4 : Déployer la pile ..... 71
Étape 5 : tester la fonction ..... 73
Étape 6 : Nettoyer ..... 73
Étapes suivantes ..... 74
Modèle de programmation ..... 75
Environnements d'exécution (runtimes) Lambda ..... 77
Environnements d'exécution pris en charge ..... 78
Nouvelles versions d'exécution ..... 80
politique d'obsolescence de l'exécution ..... 80
Modèle de responsabilité partagée ..... 81
Utilisation à l'exécution après la dépréciation ..... 83
Réception de notifications d'obsolescence lors de l'exécution ..... 84
Environnements d'exécution obsolètes ..... 86
mises à jour des versions d'exécution ..... 89
Modes de mise à jour pendant ..... 90
Déploiement des versions de l'environnement d'exécution en deux phases ..... 91
Configuration de la gestion de l'exécution ..... 92
Annulation de la version d'exécution ..... 94
Identification des modifications apportées à la version d'exécution de Lambda ..... 95
Modèle de responsabilité partagée ..... 97
Autorisations ..... 99
Obtenir des données sur les fonctions par exécution ..... 101
Liste des versions de fonctions qui utilisent un environnement d'exécution particulier ..... 101
Identifier les fonctions les plus fréquemment et les plus récemment invoquées ..... 104
Modifications de l'environnement d'exécution ..... 108
Variables d'environnement spécifiques à un langage ..... 108

Scripts encapsuleurs ..... 108
Temps d'exécution API ..... 112
Invocation suivante ..... 112
Réponse d'invocation ..... 114
Erreur d'initialisation ..... 114
Erreur d'invocation ..... 116
Exécutions uniquement basées sur le système d'exploitation ..... 118
Création d’un environnement d’exécution personnalisé ..... 119
Environnement d’exécution personnalisé ..... 123
Environnement d’exécution ..... 133
Cycle de vie de l’environnement d’exécution ..... 134
Phase d’initialisation ..... 134
Échecs pendant la phase d’initialisation ..... 135
Phase de restauration (Lambda uniquement SnapStart ) ..... 136
Phase d’invocation ..... 136
Échecs pendant la phase d’invocation ..... 137
Phase d’arrêt ..... 140
Mettre en œuvre l'apatridie ..... 141
Configuration des fonctions ..... 142
Archives de fichiers .zip ..... 144
Création de la fonction ..... 144
Utilisation de l'éditeur de code de la console ..... 146
Mise à jour du code de fonction ..... 146
Modification de l'environnement d'exécution ..... 147
Modification de l'architecture ..... 148
Utilisation du Lambda API ..... 148
AWS CloudFormation ..... 149
Images de conteneur ..... 150
Prérequis ..... 152
Utilisation d'une image AWS de base ..... 152
Utilisation d'une image de base AWS uniquement pour le système d'exploitation ..... 153
Utilisation d'une image non AWS basique ..... 154
Clients d’interface d’exécution ..... 155
ECRAutorisations Amazon ..... 155
Cycle de vie des fonctions ..... 158
Mémoire ..... 160

Quand augmenter la mémoire ..... 160
Utilisation de la console ..... 161
À l'aide du AWS CLI ..... 161
En utilisant AWS SAM ..... 162
Acceptation des recommandations relatives à la mémoire d'une fonction (console) ..... 162
Stockage éphémère ..... 163
Cas d'utilisation ..... 163
Utilisation de la console ..... 164
À l'aide du AWS CLI ..... 164
En utilisant AWS SAM ..... 165
Jeux d'instructions (ARM/x86) ..... 166
Avantages liés à l'utilisation de l'architecture arm64 ..... 166
Exigences pour la migration vers l'architecture arm64 ..... 167
Compatibilité des codes de fonction avec l'architecture arm64 ..... 167
Comment migrer vers l'architecture arm64 ..... 168
Configuration de l'architecture de l'ensemble des instructions ..... 169
Expiration ..... 170
Quand augmenter le délai d'expiration ..... 170
Utilisation de la console ..... 171
À l'aide du AWS CLI ..... 171
En utilisant AWS SAM ..... 171
Configuration des variables d'environnement ..... 173
Variables d'environnement d'exécution définies ..... 177
Exemple de scénario pour les variables d'environnement ..... 179
Sécurisation des variables d'environnement ..... 180
Récupération de variables d'environnement ..... 183
Associer des fonctions à un VPC ..... 185
IAMAutorisations requises ..... 186
Associer des fonctions Lambda à un Amazon VPC dans votre Compte AWS ..... 187
Accès à Internet lorsqu'il est connecté à un VPC ..... 191
IPv6soutien ..... 191
Bonnes pratiques d'utilisation de Lambda avec Amazon VPCs ..... 192
Comprendre les interfaces réseau élastiques Hyperplane () ENIs ..... 194
Utilisation des clés de IAM condition pour les VPC paramètres ..... 195
Didacticiels VPC ..... 200
Associer des fonctions aux ressources d'un autre compte ..... 201

Prérequis ..... 201
Créez un Amazon VPC dans le compte de votre fonction ..... 202
Accordez VPC des autorisations au rôle d'exécution de votre fonction ..... 202
..... 203
Création d'une demande VPC de connexion peering ..... 203
Préparez le compte de votre ressource ..... 204
Mettre à jour VPC la configuration dans le compte de votre fonction ..... 205
Test de votre fonction ..... 207
Accès à Internet pour les VPC fonctions ..... 208
Réseaux entrants ..... 234
Considérations relatives aux points de terminaison d'interface Lambda ..... 234
Création d'un point de terminaison d'interface pour Lambda ..... 235
Création d'une stratégie de point de terminaison d'interface pour Lambda ..... 237
Système de fichiers ..... 239
Rôle d'exécution et autorisations utilisateur ..... 239
Configuration d'un système de fichiers et d'un point d'accès ..... 240
Connexion à un système de fichiers (console) ..... 241
Alias ..... 243
Utilisation des alias ..... 245
Configurations de routage ..... 246
Versions ..... 249
Création de versions de fonction ..... 250
Utilisation des versions ..... 251
Octroi d’autorisations ..... 252
Balises ..... 253
Autorisations requises pour l’utilisation des balises ..... 253
Utilisation des étiquettes avec la console ..... 253
Utilisation de balises avec AWS CLI ..... 255
Streaming des réponses ..... 257
Limitation de la bande passante pour la diffusion des réponses ..... 258
Fonctions d'écriture ..... 258
Invocation de fonctions ..... 260
Tutoriel : création d’une fonction de streaming de réponses avec une URL de la fonction .... 262
Invocation de fonctions ..... 266
Invoquer une fonction de manière synchrone ..... 268
Invocation asynchrone ..... 272

Gestion des erreurs ..... 273
Configuration ..... 274
Conservation des dossiers ..... 276
Mappages de source d'événement ..... 284
Mappages de sources d'événements et déclencheurs ..... 284
Comportement de traitement par lots ..... 285
Cartographie des sources d'événements API ..... 288
Balises de mappage des sources d'événements ..... 289
Filtrage des événements ..... 293
Comprendre les principes de base du filtrage des événements ..... 294
Traitement des enregistrements ne répondant pas aux critères de filtrage ..... 296
Syntaxe des règles de filtrage ..... 297
Attacher des critères de filtre à un mappage de sources d’événements (console) ..... 299
Attacher des critères de filtre à un mappage de sources d’événements (AWS CLI) ..... 300
Attacher des critères de filtre à un mappage de sources d’événements (AWS SAM) ..... 301
Chiffrement des critères de filtre ..... 302
Utilisation de filtres avec différents Services AWS ..... 308
Test dans la console ..... 310
Invocation de fonctions avec des événements de test ..... 310
Création d'événements de test privés ..... 311
Création d'événements de test partageables ..... 311
Suppression des schémas d'événements de test partageables ..... 313
États des fonctions ..... 314
États de la fonction lors de la mise à jour ..... 315
Nouvelle tentative ..... 317
Détection de boucle récursive ..... 319
Comprendre la détection de boucles récursives ..... 319
Pris en charge Services AWS et SDKs ..... 321
Notifications de boucles récursives ..... 324
Répondre aux notifications de détection de boucles récursives ..... 325
Autoriser l'exécution d'une fonction Lambda dans une boucle récursive ..... 326
Régions prises en charge pour la détection des boucles récursives Lambda ..... 328
Fonction URLs ..... 330
Création d'une fonction URL (console) ..... 331
Création d'une fonction URL (AWS CLI) ..... 333
Ajouter une fonction URL à un CloudFormation modèle ..... 333

Partage de ressources entre origines () CORS ..... 335
Fonction d'étranglement URLs ..... 337
Fonction de désactivation URLs ..... 337
Fonction de suppression URLs ..... 337
Contrôle d’accès ..... 338
Fonction d'appel URLs ..... 347
URL de fonctions de surveillance ..... 359
Didacticiel : création d’une fonction avec une URL de fonction ..... 361
Mise à l’échelle de fonction ..... 367
Comprendre et visualiser la simultanéité ..... 367
Calcul de la simultanéité d'une fonction ..... 372
Comprendre la simultanéité réservée et la simultanéité provisionnée ..... 374
Simultanéité réservée ..... 374
Simultanéité allouée ..... 377
Comment Lambda alloue la simultanéité provisionnée ..... 382
Comparaison de la simultanéité réservée et de la simultanéité provisionnée ..... 382
Comprendre la simultanéité et le nombre de demandes par seconde ..... 383
Quotas de simultanéité ..... 385
Configuration de la simultanéité réservée ..... 388
Configuration de la simultanéité réservée ..... 389
Estimation précise de la simultanéité réservée requise pour une fonction ..... 390
Configuration de la simultanéité provisionnée ..... 392
Configuration de la simultanéité provisionnée ..... 393
Estimation précise de la simultanéité provisionnée requise pour une fonction ..... 395
Optimisation du code de fonction lors de l'utilisation de la simultanéité provisionnée ..... 396
Utilisation de variables d'environnement pour visualiser et contrôler le comportement de simultanéité provisionné ..... 397
Comprendre le comportement de journalisation et de facturation grâce à la simultanéité provisionnée ..... 397
Utilisation d'Application Auto Scaling pour automatiser la gestion de la simultanéité provisionnée ..... 398
Comportement de mise à l’échelle. ..... 402
Taux de mise à l’échelle de la simultanéité ..... 402
Surveillance de la simultanéité ..... 404
Métriques de simultanéité générales ..... 404
Métriques de simultanéité provisionnée ..... 404

Travailler avec la métrique ClaimedAccountConcurrency ..... 407
Création avec Node.js ..... 411
Initialisation de Node.js ..... 413
Désignation d'un gestionnaire de fonctions en tant que module ES ..... 414
Versions incluses dans Runtime SDK ..... 414
Utilisation de keep-alive ..... 415
Chargement des certificats CA ..... 415
Handler (Gestionnaire) ..... 417
Notions de base sur le gestionnaire Node.js ..... 417
Dénomination ..... 418
Utilisation d'async/await ..... 419
Utilisation de callbacks ..... 421
Bonnes pratiques en matière de code pour les fonctions Lambda de Node.js ..... 424
Déploiement d'archives de fichiers .zip ..... 427
Dépendances d'exécution dans Node.js ..... 427
Création d'un package de déploiement .zip sans dépendances ..... 428
Création d'un package de déploiement .zip avec dépendances ..... 428
Création d'une couche Node.js pour vos dépendances ..... 430
Chemin de recherche des dépendances et bibliothèques incluses dans l'exécution ..... 431
Création et mise à jour de fonctions Lambda Node.js à l'aide de fichiers .zip ..... 432
Déploiement d'images de conteneur ..... 439
AWS images de base pour Node.js ..... 440
Utilisation d'une image AWS de base ..... 441
Utilisation d'une image non AWS basique ..... 447
Couches ..... 458
Prérequis ..... 458
Compatibilité de la couche Node.js avec l'environnement d'exécution Lambda ..... 459
Chemins de couche pour les environnements d'exécution de Node.js ..... 460
Empaqueter le contenu de la couche ..... 460
Création de la couche ..... 462
Ajouter la couche à votre fonction ..... 462
Contexte ..... 466
Journalisation ..... 468
Création d'une fonction qui renvoie des journaux ..... 468
Utilisation des contrôles de journalisation avancés de Lambda avec Node.js ..... 470
Afficher les journaux dans la console Lambda ..... 477

Afficher les journaux dans la CloudWatch console ..... 477
Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ..... 478
Suppression de journaux ..... 481
Tracing ..... 482
Utilisation ADOT pour instrumenter vos fonctions Node.js ..... 483
Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Node.js ..... 483
Activation du suivi avec la console Lambda ..... 484
Activation du traçage avec le Lambda API ..... 485
Activation du traçage avec AWS CloudFormation ..... 485
Interprétation d'un suivi X-Ray ..... 486
Stockage des dépendances d'exécution dans une couche (X-RaySDK) ..... 489
Construire avec TypeScript ..... 491
Environnement de développement ..... 492
Handler (Gestionnaire) ..... 494
Notions de base sur le gestionnaire Typescript ..... 494
Utilisation d’async/await ..... 495
Utilisation de callbacks ..... 496
Utilisation de types pour l'objet d'événement ..... 498
Meilleures pratiques en matière de code pour les fonctions Typescript Lambda ..... 499
Déployez des archives de fichiers .zip ..... 502
Utiliser AWS SAM ..... 502
Utilisation de la AWS CDK ..... 504
Utilisation de la AWS CLI et d’esbuild ..... 507
Déploiement d’images de conteneur ..... 510
Utilisation d'une image de base Node.js pour créer et empaqueter le code de TypeScript fonction ..... 510
Contexte ..... 518
Journalisation ..... 520
Outils et bibliothèques ..... 520
Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour la journalisation structurée ..... 521
Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour la journalisation structurée ..... 524
Afficher les journaux dans la console Lambda ..... 527
Afficher les journaux dans la CloudWatch console ..... 528
Tracing ..... 529

Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour le traçage ..... 530
Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour le traçage ..... 532
Interprétation d'un suivi X-Ray ..... 536
Création avec Python ..... 537
Versions incluses dans Runtime SDK ..... 539
Format de la réponse ..... 539
Arrêt progressif pour les extensions ..... 539
Handler (Gestionnaire) ..... 541
Dénomination ..... 541
Comment ça marche ..... 542
Renvoi d'une valeur ..... 542
Exemples ..... 543
Bonnes pratiques en matière de code pour les fonctions Lambda en Python ..... 545
Déploiement d’archives de fichiers .zip ..... 548
Dépendances d'exécution dans Python ..... 548
Création d'un package de déploiement .zip sans dépendances ..... 549
Création d'un package de déploiement .zip avec dépendances ..... 550
Chemin de recherche des dépendances et bibliothèques incluses dans l'exécution ..... 553
Utilisation des dossiers __pycache__ ..... 554
Création de packages de déploiement .zip avec des bibliothèques natives ..... 554
Création et mise à jour de fonctions Lambda Python à l'aide de fichiers .zip ..... 556
Déploiement d’images de conteneur ..... 564
AWS images de base pour Python ..... 565
Utilisation d'une image AWS de base ..... 567
Utilisation d'une image non AWS basique ..... 573
Couches ..... 583
Prérequis ..... 583
Compatibilité de la couche Python avec Amazon Linux ..... 584
Chemins de couche pour les environnements d'exécution Python ..... 585
Empaqueter le contenu de la couche ..... 586
Création de la couche ..... 587
Ajouter la couche à votre fonction ..... 588
Utilisation des distributions `manylinux` de roues ..... 590
Contexte ..... 595
Enregistrez et surveillez les fonctions Lambda en Python ..... 597
Impression dans le journal ..... 597

Utilisation d'une bibliothèque de journalisation ... 598
Utilisation des contrôles de journalisation avancés de Lambda avec Python ... 600
Affichage des journaux dans la console Lambda ... 605
Afficher les journaux dans CloudWatch la console ... 605
Afficher les journaux avec AWS CLI ... 606
Suppression de journaux ... 609
Outils et bibliothèques ... 609
Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour la journalisation structurée ... 610
Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour la journalisation structurée ... 614
Test ... 621
Test de vos applications sans serveur ... 622
Tracing ... 624
Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour le traçage ... 625
Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour le traçage ... 628
Utilisation ADOT pour instrumenter vos fonctions Python ... 633
Utiliser le X-Ray SDK pour instrumenter vos fonctions Python ... 633
Activation du suivi avec la console Lambda ... 634
Activation du traçage avec le Lambda API ... 634
Activation du traçage avec AWS CloudFormation ... 635
Interprétation d'un suivi X-Ray ... 635
Stockage des dépendances d'exécution dans une couche (X-RaySDK) ... 638
Création avec Ruby ... 640
Versions incluses dans Runtime SDK ... 642
Activer encore un autre Ruby JIT (YJIT) ... 642
Handler (Gestionnaire) ... 643
Notions de base sur Ruby Handler ... 643
Bonnes pratiques en matière de code pour les fonctions Ruby Lambda ... 644
Déploiement d'archives de fichiers .zip ... 647
Dépendances dans Ruby ... 648
Création d'un package de déploiement .zip sans dépendances ... 648
Création d'un package de déploiement .zip avec dépendances ... 648
Création d'une couche Ruby pour vos dépendances ... 650
Création de packages de déploiement .zip avec des bibliothèques natives ... 651
Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip ... 653

Déploiement d'images de conteneur ..... 660
AWS images de base pour Ruby ..... 661
Utilisation d'une image AWS de base ..... 661
Utilisation d'une image non AWS basique ..... 668
Contexte ..... 678
Journalisation ..... 679
Création d'une fonction qui renvoie des journaux ..... 679
Afficher les journaux dans la console Lambda ..... 681
Afficher les journaux dans la CloudWatch console ..... 681
Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ..... 681
Suppression de journaux ..... 685
Utilisation de la bibliothèque Ruby Logger ..... 685
Tracing ..... 687
Activer le traçage actif avec le Lambda API ..... 693
Activation du suivi actif avec AWS CloudFormation ..... 693
Stockage des dépendances d'exécution dans une couche ..... 694
Création avec Java ..... 695
Handler (Gestionnaire) ..... 698
Exemple de gestionnaire : exécutions Java 17 ..... 698
Exemple de gestionnaire : exécutions Java 11 et antérieures ..... 700
Code d'initialisation ..... 701
Choix des types d'entrée et de sortie ..... 702
Interfaces du gestionnaire ..... 703
Bonnes pratiques en matière de code pour les fonctions Java Lambda ..... 705
Exemple de code de gestionnaire ..... 707
Déploiement d'archives de fichiers .zip ..... 709
Prérequis ..... 709
Outils et bibliothèques ..... 709
Création d'un package de déploiement avec Gradle ..... 711
Création d'une couche Java pour vos dépendances ..... 712
Création d'un package de déploiement avec Maven ..... 713
Chargement d'un package de déploiement avec la console Lambda ..... 716
Téléchargement d'un package de déploiement à l'aide du AWS CLI ..... 717
Téléchargement d'un package de déploiement avec AWS SAM ..... 719
Déploiement d'images de conteneur ..... 722
AWS images de base pour Java ..... 723

Utilisation d'une image AWS de base ... 724
Utilisation d'une image non AWS basique ... 733
Couches ... 745
Prérequis ... 745
Compatibilité de la couche Java avec Amazon Linux ... 746
Chemins de couche pour les environnements d'exécution Java ... 747
Empaqueter le contenu de la couche ... 747
Création de la couche ... 749
Ajouter la couche à votre fonction ... 750
Lambda SnapStart ... 754
Fonctions prises en charge et limitations ... 755
Régions prises en charge ... 755
Considérations de compatibilité ... 756
Tarification ... 757
SnapStart et simultanéité provisionnée ... 757
Ressources supplémentaires ... 758
Activation SnapStart ... 759
Gestion de l'unicité ... 765
Hooks d'exécution ... 768
Surveillance ... 771
Modèle de sécurité ... 774
Bonnes pratiques ... 775
Résolution des problèmes ... 779
Personnalisation de Java ... 783
Comprendre la variable d'`JAVA_TOOL_OPTIONS`environnement ... 783
Contexte ... 786
Contexte dans des exemples d'applications ... 788
Journalisation ... 790
Création d’une fonction qui renvoie des journaux ... 790
Utilisation des contrôles de journalisation avancés de Lambda avec Java ... 792
Implémentation de la journalisation avancée avec Log4j2 et SLF4J ... 796
Outils et bibliothèques ... 799
Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour la journalisation structurée ... 800
Afficher les journaux dans la console Lambda ... 804
Afficher les journaux dans la CloudWatch console ... 805

Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ..... 805
Suppression de journaux ..... 808
Exemple de code de journalisation ..... 809
Tracing ..... 810
Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour le traçage ..... 811
Utilisation de Powertools pour AWS Lambda (Java) et AWS CDK pour le traçage ..... 813
Utilisation ADOT pour instrumenter vos fonctions Java ..... 825
Utiliser le X-Ray SDK pour instrumenter vos fonctions Java ..... 826
Activation du suivi avec la console Lambda ..... 826
Activation du traçage avec le Lambda API ..... 827
Activation du traçage avec AWS CloudFormation ..... 827
Interprétation d'un suivi X-Ray ..... 828
Stockage des dépendances d'exécution dans une couche (X-RaySDK) ..... 831
Traçage par rayons X dans les applications d'échantillonnage (X-RaySDK) ..... 832
Exemples d’application ..... 833
Création avec Go ..... 835
Prise en charge de l’exécution Go ..... 835
Outils et bibliothèques ..... 836
Handler (Gestionnaire) ..... 838
Notions de base sur Go Handler ..... 838
Dénomination ..... 840
Gestionnaire de fonctions Lambda utilisant des types structurés ..... 840
Utilisation de l'état global ..... 843
Meilleures pratiques en matière de code pour les fonctions Go Lambda ..... 844
Contexte ..... 846
Accès aux informations du contexte d'appel ..... 846
Déploiement d'archives de fichiers .zip ..... 849
Création d’un fichier .zip sur macOS et Linux ..... 849
Création d’un fichier .zip sous Windows ..... 851
Création et mise à jour de fonctions Lambda Go à l’aide de fichiers .zip ..... 854
Déploiement d’images de conteneur ..... 861
AWS images de base pour le déploiement des fonctions Go ..... 861
Client d’interface d’environnement d’exécution Go ..... 862
Utilisation d'une image de base AWS uniquement pour le système d'exploitation ..... 862
Utilisation d'une image non AWS basique ..... 870
Couches ..... 879

Journalisation ..... 880
Création d'une fonction qui renvoie des journaux ..... 880
Afficher les journaux dans la console Lambda ..... 882
Afficher les journaux dans la CloudWatch console ..... 882
Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ..... 883
Suppression de journaux ..... 886
Tracing ..... 887
Utilisation ADOT pour instrumenter les fonctions de votre Go ..... 888
Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Go ..... 888
Activation du suivi avec la console Lambda ..... 888
Activation du traçage avec le Lambda API ..... 889
Activation du traçage avec AWS CloudFormation ..... 889
Interprétation d'un suivi X-Ray ..... 890
Création avec C# ..... 894
Environnement de développement ..... 894
Installation du. NETmodèles de projets ..... 894
Installation et mise à jour des CLI outils ..... 895
Handler (Gestionnaire) ..... 896
. NETmodèles d'exécution pour Lambda ..... 897
Gestionnaires de bibliothèques de classes ..... 897
Gestionnaires d'assemblages exécutables ..... 898
Sérialisation dans les fonctions Lambda ..... 899
Simplifiez le code de la fonction à l'aide du cadre d'annotations Lambda ..... 902
Restrictions liées au gestionnaire de fonctions Lambda ..... 904
Meilleures pratiques en matière de code pour les fonctions Lambda C# ..... 905
Package de déploiement ..... 907
CLI globale NET Lambda ..... 908
AWS SAM ..... 914
AWS CDK ..... 918
ASP.NET ..... 921
Déploiement d'images de conteneur ..... 927
AWS images de base pour. NET ..... 928
Utilisation d'une image AWS de base ..... 928
Utilisation d'une image non AWS basique ..... 931
Compilation anticipée native ..... 936
Le fichier d'exécution Lambda ..... 936

Prérequis ... 937
Premiers pas ... 937
Sérialisation ... 941
Réduction ... 941
Résolution des problèmes ... 942
Contexte ... 943
Journalisation ... 945
Création d'une fonction qui renvoie des journaux ... 945
Outils et bibliothèques ... 946
Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour la journalisation structurée ... 946
Afficher les journaux dans la console Lambda ... 950
Afficher les journaux dans la CloudWatch console ... 950
Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ... 950
Suppression de journaux ... 954
Tracing ... 955
Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour le traçage ... 956
Utiliser le X-Ray SDK pour instrumenter votre. NETfonctions ... 959
Activation du suivi avec la console Lambda ... 960
Activation du traçage avec le Lambda API ... 961
Activation du traçage avec AWS CloudFormation ... 961
Interprétation d'un suivi X-Ray ... 962
Test ... 966
Test de vos applications sans serveur ... 967
Construire avec PowerShell ... 971
Environnement de développement ... 973
Package de déploiement ... 974
Création d'une fonction Lambda ... 974
Handler (Gestionnaire) ... 977
Renvoi de données ... 978
Contexte ... 979
Journalisation ... 980
Création d'une fonction qui renvoie des journaux ... 980
Afficher les journaux dans la console Lambda ... 982
Afficher les journaux dans la CloudWatch console ... 982
Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI) ... 983

Suppression de journaux ..... 986
Création avec Rust ..... 987
Handler (Gestionnaire) ..... 989
Notions de base sur le gestionnaire Rust ..... 989
Utilisation de l'état partagé ..... 990
Bonnes pratiques en matière de code pour les fonctions Rust Lambda ..... 991
Contexte ..... 994
Accès aux informations du contexte d'appel ..... 994
Événements HTTP ..... 996
Déploiement d’archives de fichiers .zip ..... 999
Prérequis ..... 999
Création de la fonction ..... 999
Déploiement de la fonction ..... 1000
Invoquer la fonction ..... 1002
Journalisation ..... 1004
Création d'une fonction qui écrit des journaux ..... 1004
Implémentation de la journalisation avancée avec la caisse de traçage ..... 1005
Bonnes pratiques ..... 1007
Code de fonction ..... 1007
Configuration de la fonction ..... 1009
Extensibilité des fonctions ..... 1010
Métriques et alarmes ..... 1010
Utilisation des flux ..... 1011
Bonnes pratiques de sécurité ..... 1012
Tester les fonctions sans serveur ..... 1014
Résultats commerciaux ciblés ..... 1015
Que faut-il tester ? ..... 1015
Comment tester le sans serveur ..... 1016
Techniques de test ..... 1017
Tests dans le cloud ..... 1018
Tester avec des simulations ..... 1021
Tester avec émulation ..... 1022
Bonnes pratiques ..... 1023
Prioriser les tests dans le cloud ..... 1024
Structurer votre code pour le rendre testable ..... 1024
Accélérer les boucles de rétroaction du développement ..... 1024

Se concentrer sur les tests d'intégration ..... 1025
Créer des environnements de test isolés ..... 1025
Utiliser des simulations pour isoler la logique métier ..... 1027
Utiliser les émulateurs avec parcimonie ..... 1027
Défis liés aux tests locaux ..... 1028
Exemple : la fonction Lambda crée un compartiment S3 ..... 1028
Exemple : la fonction Lambda traite les messages d'une file d'attente Amazon SQS ..... 1029
FAQ ..... 1029
Prochaines étapes et ressources ..... 1031
Intégration d’autres services ..... 1032
Création d'un déclencheur ..... 1032
Liste des services ..... 1033
Apache Kafka ..... 1036
Exemple d’évènement ..... 1037
Configuration de la source de l'événement ..... 1038
Messages de traitement ..... 1047
Filtrage des événements ..... 1055
Destinations en cas d’échec ..... 1061
Résolution des problèmes ..... 1065
APIPasserelle ..... 1069
Choix d'un API type ..... 1069
Ajout d’un point de terminaison public à votre fonction Lambda ..... 1072
Intégration de proxy ..... 1072
Format des événements ..... 1073
Format de la réponse ..... 1074
Autorisations ..... 1075
Exemple d'application ..... 1077
Didacticiel ..... 1077
Erreurs ..... 1100
Compositeur d'infrastructure ..... 1101
Exportation d'une fonction Lambda vers Infrastructure Composer ..... 1101
Autres ressources ..... 1104
CloudFormation ..... 1105
Amazon DocumentDB ..... 1109
Exemple d'événement Amazon DocumentDB ..... 1110
Conditions préalables et autorisations ..... 1111

Configuration réseau ..... 1113
Création d'un mappage des sources d'événements Amazon DocumentDB (console) ..... 1115
Création d'un mappage de source d'événements Amazon DocumentDB (ou) SDK CLI ..... 1117
Positions de départ des interrogations et des flux ..... 1120
Surveillance de votre source d'événements Amazon DocumentDB ..... 1120
Didacticiel ..... 1121
DynamoDB ..... 1156
Flux d’interrogation et de mise en lots ..... 1156
Positions de départ des interrogations et des flux ..... 1158
Lecteurs simultanés ..... 1158
Exemple d’évènement ..... 1158
Créer un mappage ..... 1160
Défaillances des articles du lot ..... 1162
Gestion des erreurs ..... 1175
Surveillance ..... 1178
Traitement dynamique ..... 1179
Paramètres ..... 1184
Filtrage des événements ..... 1187
Didacticiel ..... 1196
EC2 ..... 1213
Octroi d'autorisations à EventBridge (CloudWatch événements) ..... 1214
Elastic Load Balancing (Application Load Balancer) ..... 1215
Invoquer à l'aide d'un EventBridge planificateur ..... 1218
Configurer le rôle d'exécution ..... 1218
Créer une planification ..... 1218
Ressources connexes ..... 1223
IoT ..... 1224
Kinesis Data Streams ..... 1226
Flux d’interrogation et de mise en lots ..... 1226
Exemple d’évènement ..... 1228
Création d'un mappage ..... 1229
Défaillances des articles du lot ..... 1235
Gestion des erreurs ..... 1250
Traitement dynamique ..... 1253
Paramètres ..... 1257
Filtrage des événements ..... 1260

Didacticiel ..... 1264
MQ ..... 1282
Comprendre le groupe de consommateurs Lambda pour Amazon MQ ..... 1284
Création d'un mappage ..... 1288
Paramètres ..... 1295
Filtrage des événements ..... 1296
Dépannage ..... 1303
MSK ..... 1304
Exemple d’évènement ..... 1305
Configurer la source de l'événement ..... 1306
Messages de traitement ..... 1319
Filtrage des événements ..... 1329
Destinations en cas d’échec ..... 1334
Didacticiel ..... 1339
RDS ..... 1358
Configuration de votre fonction pour qu'elle fonctionne avec les RDS ressources ..... 1358
Connexion à une RDS base de données Amazon dans une fonction Lambda ..... 1361
Traitement des notifications d'événements provenant d'Amazon RDS ..... 1377
Tutoriel complet sur Lambda et Amazon RDS ..... 1379
S3 ..... 1380
Didacticiel : Utiliser un déclencheur S3 ..... 1382
Didacticiel : Utilisation d'un déclencheur Amazon S3 pour créer des images miniatures ..... 1409
SQS ..... 1439
Comprendre le comportement d'interrogation et de traitement par lots pour les mappages de sources SQS d'événements Amazon ..... 1439
Exemple d’événement de message de file d’attente standard ..... 1440
Exemple FIFO d'événement de message de file d'attente ..... 1442
Création d'un mappage ..... 1443
Comportement de mise à l’échelle. ..... 1447
Gestion des erreurs ..... 1449
Paramètres ..... 1462
Filtrage des événements ..... 1463
Didacticiel ..... 1468
Tutoriel SQS entre comptes ..... 1488
Lot S3 ..... 1496
Appel de fonctions Lambda à partir d'opérations par lot Amazon S3 ..... 1497

SNS ..... 1499
Ajout d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda à l'aide de la console ..... 1499
Ajout manuel d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda ..... 1500
Exemple de forme d'événement SNS ..... 1501
Didacticiel ..... 1502
Autorisations Lambda ..... 1524
Rôle d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources) ..... 1526
Création d'un rôle d'exécution dans la IAM console ..... 1526
Création et gestion des rôles à l'aide du AWS CLI ..... 1527
Accorder un accès assorti d'un privilège minimum à votre rôle d'exécution Lambda ..... 1529
Mettre à jour le rôle d'exécution ..... 1530
AWS politiques gérées ..... 1531
Fonction source ARN ..... 1535
Autorisations d'accès (autorisations permettant à d'autres entités d'accéder à vos fonctions) . 1540
Politiques basées sur l’identité ..... 1540
Politiques basées sur les ressources ..... 1547
Contrôle d'accès basé sur les attributs ..... 1555
Ressources et conditions ..... 1563
Sécurité, gouvernance et conformité ..... 1570
Protection des données ..... 1571
Chiffrement en transit ..... 1572
Chiffrement au repos ..... 1572
Gestion de l’identité et des accès ..... 1578
Public ciblé ..... 1578
Authentification par des identités ..... 1579
Gestion des accès à l’aide de politiques ..... 1583
Comment AWS Lambda fonctionne avec IAM ..... 1586
Exemples de politiques basées sur l’identité ..... 1593
Politiques gérées par AWS ..... 1596
Résolution des problèmes ..... 1602
Gouvernance ..... 1605
Contrôles proactifs avec Guard ..... 1607
Contrôles proactifs avec AWS Config ..... 1611
Detective contrôle avec AWS Config ..... 1619
Signature de code ..... 1624

Analyse du code ... 1627
Observabilité ... 1632
Validation de la conformité ... 1641
Résilience ... 1641
Sécurité de l'infrastructure ... 1642
Signature de code ... 1643
Validation de signature ... 1644
Configuration de la signature de code avec le Lambda API ... 1645
Créer une configuration ... 1646
Mettre à jour la configuration ... 1647
Autorisations ... 1648
Balises de configuration de signature de code ... 1649
Surveillance de fonctions ... 1654
Tarification ... 1654
Afficher les métriques des fonctions ... 1655
Afficher les métriques sur la CloudWatch console ... 1655
Types de métriques ... 1656
Journaux de fonctions ... 1661
IAMAutorisations requises ... 1661
Tarification ... 1662
Configuration des journaux de fonctions ... 1662
Afficher les journaux des fonctions ... 1678
CloudTrail journaux ... 1683
Événements relatifs aux données Lambda dans CloudTrail ... 1684
Événements de gestion Lambda dans CloudTrail ... 1686
Utilisation CloudTrail pour résoudre les problèmes liés aux sources d'événements Lambda désactivées ... 1688
Exemples d'événements Lambda ... 1689
AWS X-Ray ... 1692
Comprendre les suivis X-Ray ... 1693
Autorisations du rôle d'exécution ... 1698
Le AWS X-Ray daemon ... 1698
Activer le traçage actif avec le Lambda API ... 1699
Activation du suivi actif avec AWS CloudFormation ... 1699
Informations sur les fonctions ... 1701
Comment ça marche ... 1701

Tarification ..... 1702
Environnements d'exécution pris en charge ..... 1702
Activation de Lambda Insights dans la console ..... 1702
Activation de Lambda Insights par programme ..... 1703
Utilisation du tableau de bord Lambda Insights ..... 1703
Détection d'anomalies de fonction ..... 1705
Dépannage d'une fonction ..... 1707
Quelle est la prochaine étape ? ..... 1709
Couches Lambda ..... 1710
Utilisation des couches ..... 1712
Couches et versions de couches ..... 1712
Empaquetage des couches ..... 1714
Chemins d'accès de couche pour chaque exécution Lambda ..... 1714
Création et suppression de couches ..... 1718
Création d’une couche ..... 1718
Suppression d’une version de couche ..... 1720
Ajout de couches ..... 1721
Accès au contenu de la couche à partir de la fonction ..... 1723
Recherche d’informations sur la couche ..... 1723
Couches avec AWS CloudFormation ..... 1726
Couches avec AWS SAM ..... 1727
Extensions Lambda ..... 1728
Environnement d’exécution ..... 1729
Impact sur les performances et les ressources ..... 1730
Autorisations ..... 1731
Configuration des extensions ..... 1732
Configuration des extensions (archive de fichiers .zip) ..... 1732
Utilisation des extensions dans les images de conteneur ..... 1732
Étapes suivantes ..... 1733
Partenaires des extensions ..... 1734
AWS extensions gérées ..... 1735
Prolongations API ..... 1736
Cycle de vie d’un environnement d’exécution Lambda ..... 1737
APIRéférence des extensions ..... 1747
Télémétrie API ..... 1753
Création d'extensions à l'aide de la télémétrie API ..... 1754

Enregistrement de votre extension ... 1756
Création d'un écouteur de télémétrie ... 1757
Spécification d'un protocole de destination ... 1758
Configuration de l'utilisation de la mémoire et de la mise en mémoire tampon ... 1759
Envoi d'une demande d'abonnement à la télémétrie API ... 1761
Messages de télémétrie API entrants ... 1762
Référence d'API ... 1765
Référence du schéma `Event` ... 1769
Conversion d'événements en OTel spans ... 1790
Journaux API ... 1796
Résolution des problèmes ... 1810
Déploiement ... 1810
Général : autorisation refusée/impossible de charger ce fichier ... 1811
Général : Une erreur se produit lors de l'appel du UpdateFunctionCode ... 1812
Amazon S3 : code d'erreur PermanentRedirect. ... 1812
Général : impossible de trouver, impossible de charger, impossible d'importer, classe introuvable, aucun fichier ou répertoire de ce type ... 1812
Général : gestionnaire de méthode non défini ... 1813
Lambda : échec de la conversion de couche ... 1814
Lambda : ou InvalidParameterValueException RequestEntityTooLargeException ... 1814
Lambda : InvalidParameterValueException ... 1815
Lambda : quotas de simultanéité et de mémoire ... 1815
Invocation ... 1816
Lambda : expiration de la fonction pendant la phase d'initialisation (Sandbox.Timedout) ... 1816
IAM: lambda : InvokeFunction non autorisé ... 1817
Lambda : Impossible de trouver un bootstrap valide (Runtime). InvalidEntrypoint) ... 1817
Lambda : L'opération ne peut pas être effectuée ResourceConflictException ... 1818
Lambda : La fonction est bloquée à l'état Pending (En attente) ... 1818
Lambda : Une fonction utilise toute la simultanéité ... 1818
Général : Impossible d'invoquer la fonction avec d'autres comptes ou services ... 1819
Général : L'invocation de la fonction boucle ... 1819
Lambda : Routage d'alias avec une simultanéité approvisionnée ... 1819
Lambda : Démarrages à froid avec la simultanéité allouée ... 1819
Lambda : Démarrages à froid avec de nouvelles versions ... 1820
EFS: la fonction n'a pas pu monter le système de EFS fichiers ... 1821
EFS: la fonction n'a pas pu se connecter au système de EFS fichiers ... 1821

EFS: La fonction n'a pas pu monter le système de EFS fichiers en raison d'un délai d'attente ... 1821
Lambda : Lambda a détecté un processus d'E/S qui prenait trop de temps ... 1822
Exécution ... 1822
Lambda : l'exécution prend trop de temps ... 1822
Lambda : les journaux ou les traces n'apparaissent pas ... 1823
Lambda : certains journaux de ma fonction n'apparaissent pas ... 1823
Lambda : la fonction renvoie avant la fin de l'exécution ... 1824
AWS SDK: Versions et mises à jour ... 1825
Python : les bibliothèques se chargent de manière incorrecte ... 1826
Java : votre fonction met plus de temps à traiter les événements après la mise à jour de Java 11 vers Java 17 ... 1826
Réseaux ... 1826
VPC : la fonction perd l'accès à Internet ou expire ... 1827
VPC : la fonction doit accéder aux AWS services sans utiliser Internet ... 1827
VPC : limite d'interface réseau Elastic atteinte ... 1827
EC2 : interface réseau élastique de type « lambda » ... 1827
DNS : échec de la connexion aux hôtes avec UNKNOWNHOSTEXCEPTION ... 1828
Applications Lambda ... 1829
Surveillance des applications ... 1830
Déploiements propagés ... 1833
Exemple de AWS SAM modèle Lambda ... 1833
Kubernetes ... 1835
AWS Controllers for Kubernetes (ACK) ... 1835
Crossplane ... 1836
Exemples d’applications ... 1837
Travailler avec AWS SDKs ... 1840
Exemples de code ... 1842
Principes de base ... 1854
Hello Lambda ... 1855
Actions ... 1865
Scénarios ... 1983
Confirmez automatiquement les utilisateurs connus à l'aide d'une fonction Lambda ... 1984
Migrer automatiquement les utilisateurs connus à l'aide d'une fonction Lambda ... 2004
Créez un REST API pour suivre les données COVID -19 ... 2026
Création d'une bibliothèque de prêt REST API ... 2027

Créer une application de messagerie ..... 2028
Création d’une application sans serveur pour gérer des photos ..... 2029
Créer une application de chat WebSocket ..... 2033
Créez une application pour analyser les commentaires des clients ..... 2034
Mise en route avec des fonctions ..... 2040
Invoquer une fonction Lambda à partir d’un navigateur ..... 2153
Transformez les données avec S3 Object Lambda ..... 2154
Utiliser API Gateway pour appeler une fonction Lambda ..... 2155
Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda ..... 2157
Utilisent des événements planifiés pour invoquer une fonction Lambda ..... 2157
Rédigez des données d'activité personnalisées à l'aide d'une fonction Lambda après l'authentification de l'utilisateur Amazon Cognito ..... 2160
Exemples sans serveur ..... 2180
Connexion à une RDS base de données Amazon dans une fonction Lambda ..... 2181
Invoquer une fonction Lambda à partir d’un déclencheur Kinesis ..... 2198
Invoquer une fonction Lambda à partir d'un déclencheur DynamoDB ..... 2208
Invoquer une fonction Lambda depuis un déclencheur Amazon DocumentDB ..... 2218
Invoquer une fonction Lambda depuis un déclencheur Amazon MSK ..... 2225
Invoquer une fonction Lambda à partir d’un déclencheur Amazon S3 ..... 2233
Invoquer une fonction Lambda depuis un déclencheur Amazon SNS ..... 2244
Invoquer une fonction Lambda depuis un déclencheur Amazon SQS ..... 2254
Signalement des échecs d’articles par lots pour les fonctions Lambda à l’aide d’un déclencheur Kinesis ..... 2263
Signalement des défaillances d'éléments de lot pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB ..... 2276
Signalement des défaillances d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS ..... 2287
Quotas Lambda ..... 2298
calcul et stockage ..... 2298
Configuration, déploiement et exécution de fonction ..... 2299
Requêtes Lambda API ..... 2302
Autres services ..... 2303
Historique de la documentation ..... 2304
Mises à jour antérieures ..... 2334
..... mmcccxliii

# Qu'est-ce que c'est AWS Lambda ?

Vous pouvez l'utiliser AWS Lambda pour exécuter du code sans provisionner ni gérer de serveurs.

Lambda exécute le code sur une infrastructure informatique à haute disponibilité et effectue toute l'administration des ressources informatiques, y compris la maintenance des serveurs et du système d'exploitation, l'allocation et la mise à l'échelle automatique des capacités, ainsi que la mise à l'échelle automatique et la journalisation. Avec Lambda, il vous suffit de fournir votre code dans l'une des exécutions de langage prises en charge par Lambda.

Vous organisez votre code en fonctions Lambda. Le service Lambda n'exécute votre fonction qu'en cas de besoin et se met à l'échelle automatiquement. Vous payez uniquement le temps de calcul que vous utilisez. Vous n'exposez aucuns frais quand votre code n'est pas exécuté. Pour plus d'informations, consultez AWS Lambda Pricing (Tarification CTlong).

> **Tip**
>
> Pour apprendre à créer des solutions sans serveur, consultez le Guide du développeur sans serveur.

## Cas d'utilisation de Lambda

Lambda est un service de calcul idéal pour les scénarios d'application qui doivent augmenter la capacité rapidement, et la réduire à zéro lorsqu'elle n'est pas demandée. Par exemple, vous pouvez utiliser Lambda pour :

- Traitement de fichiers : utilisez Amazon Simple Storage Service (Amazon S3) pour déclencher le traitement des données Lambda en temps réel après un chargement.
- Traitement des flux : utilisez Lambda et Amazon Kinesis pour traiter des données de flux en temps réel pour le suivi de l'activité des applications, le traitement des ordres de transaction, l'analyse du flux de clics, le nettoyage des données, le filtrage des journaux, l'indexation, l'analyse des réseaux sociaux, la télémétrie des données des appareils de l'Internet des objets (IoT) et les métriques.
- Applications Web : associez Lambda à d'autres AWS services pour créer de puissantes applications Web qui évoluent automatiquement à la hausse ou à la baisse et s'exécutent dans une configuration hautement disponible sur plusieurs centres de données.

- Backends IoT : créez des backends sans serveur à l'aide de Lambda pour gérer les demandes Web, mobiles, IoT et tierces. API
- Backends mobiles : créez des backends à l'aide de Lambda et API Amazon Gateway pour authentifier et traiter les demandes. API AWS Amplify Utilisez-le pour intégrer facilement vos interfaces iOS, Android, Web et React Native.

Lorsque vous utilisez Lambda, vous n'êtes responsable que de votre code. Lambda gère le parc informatique qui offre un équilibre entre mémoireCPU, réseau et autres ressources pour exécuter votre code. Étant donné que Lambda gère ces ressources, vous ne pouvez ni vous connecter à des instances de calcul, ni personnaliser le système d'exploitation sur les runtimes fournis. Lambda accomplit des activités opérationnelles et d'administration en votre nom, dont la gestion de la capacité, la surveillance et la journalisation de vos fonctions Lambda.

# Fonctions principales

Les fonctions clés suivantes vous aident à développer des applications Lambda évolutives, sécurisées et facilement extensibles :

Variables d'environnement

Utilisez des variables d'environnement pour ajuster le comportement de votre fonction sans mettre à jour le code.

Versions

Gérez le déploiement de vos fonctions avec des versions, afin que, par exemple, une nouvelle fonction puisse être utilisée pour les tests bêta sans affecter les utilisateurs de la version de production stable.

Images de conteneur

Créez une image de conteneur pour une fonction Lambda en utilisant une image de base AWS fournie ou une image de base alternative afin de pouvoir réutiliser vos outils de conteneur existants ou déployer des charges de travail plus importantes qui reposent sur des dépendances importantes, telles que le machine learning.

Couches

Empaquetez les bibliothèques et autres dépendances pour réduire la taille des archives de déploiement et rendre plus rapide le déploiement de votre code.

### Extensions Lambda

Complétez vos fonctions Lambda avec des outils de surveillance, d'observabilité, de sécurité et de gouvernance.

### Fonction URLs

Ajoutez un point de terminaison HTTP (S) dédié à votre fonction Lambda.

### Streaming des réponses

Configurez votre fonction Lambda URLs pour qu'elle renvoie les charges utiles de réponse aux clients à partir des fonctions Node.js, afin d'améliorer les performances du délai jusqu'au premier octet (TTFB) ou de renvoyer des charges utiles plus importantes.

### Contrôles de simultanéité et de mise à l'échelle

Contrôlez avec précision la mise à l'échelle et la réactivité de vos applications de production.

### Signature de code

Vérifiez que seuls les développeurs approuvés publient du code non modifié et fiable dans vos fonctions Lambda

### Réseaux privés

Créez un réseau privé pour les ressources telles que les bases de données, les instances de mémoire cache ou les services internes.

### Accès au système de fichiers

Configurez une fonction pour monter un Amazon Elastic File System (AmazonEFS) dans un répertoire local, afin que votre code de fonction puisse accéder aux ressources partagées et les modifier en toute sécurité et avec une simultanéité élevée.

### Lambda SnapStart pour Java

Améliorez les performances de démarrage pour les exécutions Java jusqu'à 10 fois sans coût supplémentaire, généralement sans modification du code de votre fonction.

# Créez votre première fonction Lambda

Pour démarrer avec Lambda, utilisez la console Lambda pour créer une fonction. En quelques minutes, vous pouvez créer et déployer une fonction, mais aussi la tester dans la console.

Au cours du didacticiel, vous apprendrez certains concepts fondamentaux de Lambda, tels que la façon de transmettre des arguments à votre fonction à l'aide de l'objet d'événement Lambda. Vous apprendrez également comment renvoyer les résultats du journal de votre fonction et comment afficher les journaux d'invocation de votre fonction dans CloudWatch Logs.

Pour simplifier les choses, vous devez créer votre fonction à l'aide de l'exécution Python ou Node.js. Avec ces langages interprétés, vous pouvez modifier le code de fonction directement dans l'éditeur de code intégré à la console. Avec des langages compilés tels que Java et C#, vous devez créer un package de déploiement sur votre machine de génération locale et le télécharger sur Lambda. Pour en savoir plus sur le déploiement de fonctions sur Lambda à l'aide d'autres environnements d'exécution, consultez les liens figurant dans la section the section called “Ressources supplémentaires et prochaines étapes”.

> **Tip**
>
> Pour apprendre à créer des solutions sans serveur, consultez le Guide du développeur sans serveur.

## Prérequis

### Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

## Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu'utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d'un ensemble d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l'utilisateur AWS IAM Identity Center .

# Créer une fonction Lambda à l'aide de la console

Dans cet exemple, votre fonction prend un JSON objet contenant deux valeurs entières étiquetées `"length"` et`"width"`. La fonction multiplie ces valeurs pour calculer une surface et la renvoie sous forme de JSON chaîne.

Votre fonction imprime également la zone calculée, ainsi que le nom de son groupe de CloudWatch logs. Plus loin dans le didacticiel, vous apprendrez à utiliser les CloudWatch journaux pour afficher les enregistrements de l'invocation de vos fonctions.

Pour créer votre fonction, vous utilisez d'abord la console pour créer une fonction Hello world de base. À l'étape suivante, vous devez ensuite ajouter votre propre code de fonction.

Pour créer une fonction Lambda Hello world à l'aide de la console

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Sélectionnez Créer à partir de zéro.
4. Dans le volet Informations de base, pour Nom de la fonction, entrez **`myLambdaFunction`**.
5. Pour Environnement d'exécution, choisissez Node.js 20.x ou Python 3.12
6. Laissez architecture défini sur x86_64 et choisissez Créer une fonction.

Lambda crée une fonction qui renvoie le message`Hello from Lambda!`Lambda crée également unrôle d'exécutionpour votre événement. Un rôle d'exécution est un rôle AWS Identity and Access Management (IAM) qui accorde à une fonction Lambda l'autorisation d'accès Services AWS et de ressources. Pour votre fonction, le rôle créé par Lambda accorde des autorisations de base pour CloudWatch écrire dans Logs.

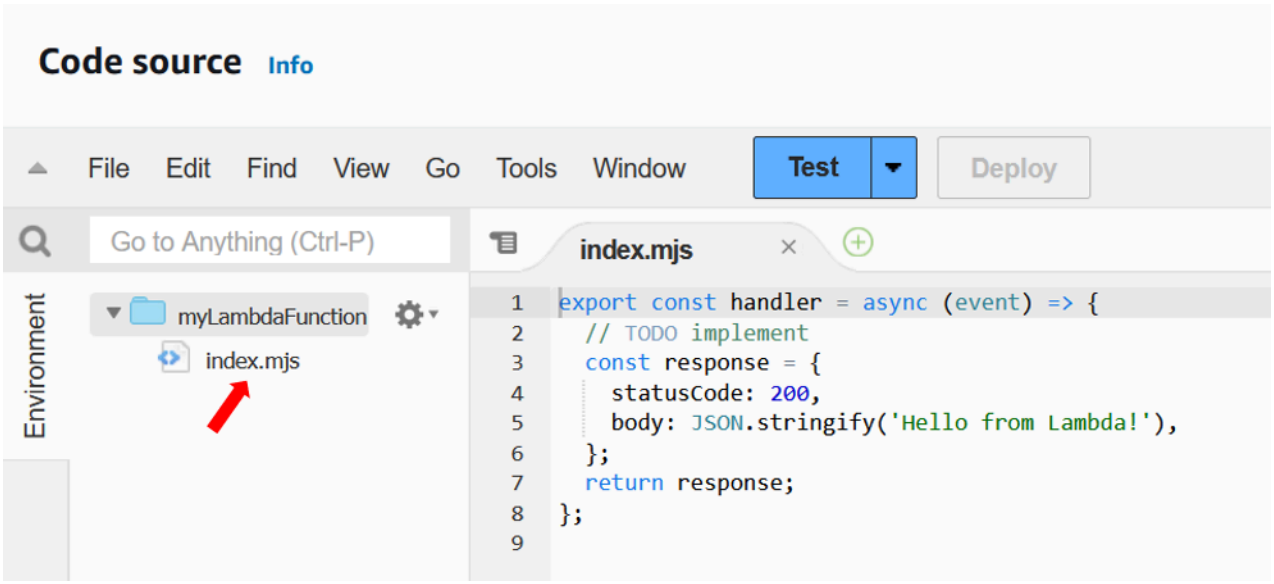
Vous utilisez désormais l'éditeur de code intégré de la console pour remplacer le code Hello world créé par Lambda avec votre propre code de fonction.

Node.js

Pour modifier le code dans la console

1. Cliquez sur l’onglet Code.

   Dans l'éditeur de code intégré de la console, vous devriez voir le code de fonction créé par Lambda. Si vous ne voyez pas l'onglet index.mjs dans l'éditeur de code, sélectionnez index.mjs dans l'explorateur de fichiers, comme le montre le schéma suivant.




2. Collez le code suivant dans l'onglet index.mjs, en remplaçant le code créé par Lambda.

```
export const handler = async (event, context) => {

  const length = event.length;
  const width = event.width;
  let area = calculateArea(length, width);
  console.log(`The area is ${area}`);

  console.log('CloudWatch log group: ', context.logGroupName);

  let data = {
    "area": area,
  };
    return JSON.stringify(data);

  function calculateArea(length, width) {
    return length * width;
  }
};
```

3. Choisissez Déployer afin de mettre à jour le code de votre fonction. Lorsque Lambda a déployé les modifications, la console affiche une bannière vous indiquant que votre fonction a bien été mise à jour.

## Comprendre le code de votre fonction

Avant de passer à l'étape suivante, examinons le code de la fonction pour comprendre certains concepts Lambda clés.

- Le gestionnaire Lambda :

  Votre fonction Lambda contient une fonction Node.js nommée `handler`. Une fonction Lambda en Node.js peut contenir plusieurs fonctions Node.js, mais la fonction gestionnaire est toujours le point d'entrée de votre code. Lorsque votre fonction est invoquée, Lambda exécute cette méthode.

  Lorsque vous avez créé votre fonction Hello world à l'aide de la console, Lambda a automatiquement défini le nom de la méthode du gestionnaire pour votre fonction sur `handler`. Veillez à ne pas modifier le nom de cette fonction Node.js. Dans ce cas, Lambda ne pourra pas exécuter votre code lorsque vous invoquerez votre fonction.

  Pour en savoir plus sur le gestionnaire Lambda en Node.js, voir the section called “Handler (Gestionnaire)”.

- L'objet de l'événement Lambda :

  La fonction `handler` accepte deux arguments, `event` et `context`. Dans Lambda, un événement est un document JSON formaté qui contient des données à traiter par votre fonction.

  Si votre fonction est invoquée par une autre personne Service AWS, l'objet d'événement contient des informations sur l'événement à l'origine de l'invocation. Par exemple, si un compartiment Amazon Simple Storage Service (Amazon S3) invoque votre fonction lorsqu'un objet est chargé, l'événement contiendra le nom du compartiment Amazon S3 et la clé de l'objet.

  Dans cet exemple, vous allez créer un événement dans la console en saisissant un document JSON formaté avec deux paires clé-valeur.

- L'objet de contexte Lambda :

  Le deuxième argument que votre fonction accepte est `context`. Lambda transmet automatiquement l'objet de contexte à votre fonction. L'objet de contexte contient des informations supplémentaires sur l'invocation de la fonction et l'environnement d'exécution.

Vous pouvez utiliser l'objet de contexte pour générer des informations sur l'invocation de votre fonction à des fins de surveillance. Dans cet exemple, votre fonction utilise le `logGroupName` paramètre pour afficher le nom de son groupe de CloudWatch journaux.

Pour en savoir plus sur l'objet de contexte Lambda en Node.js, voir the section called “Contexte”.

- Connexion à Lambda :

  Avec Node.js, vous pouvez utiliser des méthodes de la console telles que `console.log` et `console.error` pour envoyer des informations au journal de votre fonction. L'exemple de code utilise `console.log` des instructions pour afficher la zone calculée et le nom du groupe CloudWatch Logs de la fonction. Vous pouvez également utiliser n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`.

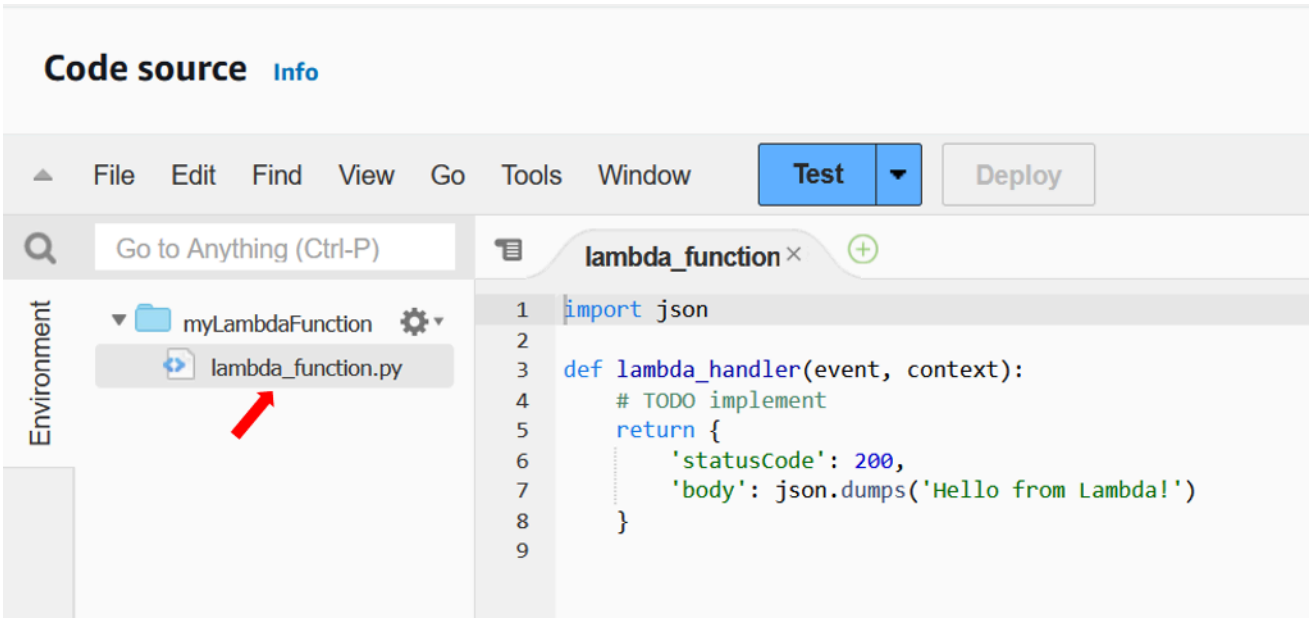
  Pour en savoir plus, consultez the section called “Journalisation”. Pour en savoir plus sur la connexion à d'autres environnements d'exécution, consultez les pages « Génération avec » correspondant aux environnements d'exécution qui vous intéressent.

Python

Pour modifier le code dans la console

1. Cliquez sur l’onglet Code.

   Dans l'éditeur de code intégré de la console, vous devriez voir le code de fonction créé par Lambda. Si vous ne voyez pas l'onglet lambda_function.py dans l'éditeur de code, sélectionnez lambda_function.py dans l'explorateur de fichiers, comme le montre le schéma suivant.




2. Collez le code suivant dans l'onglet lambda_function.py, en remplaçant le code créé par Lambda.

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

3. Choisissez Déployer afin de mettre à jour le code de votre fonction. Lorsque Lambda a déployé les modifications, la console affiche une bannière vous indiquant que votre fonction a bien été mise à jour.

### Comprendre le code de votre fonction

Avant de passer à l'étape suivante, examinons le code de la fonction pour comprendre certains concepts Lambda clés.

- Le gestionnaire Lambda :

  Votre fonction Lambda contient une fonction Python nommée `lambda_handler`. Une fonction Lambda en Python peut contenir plusieurs fonctions Python, mais la fonction gestionnaire est toujours le point d'entrée de votre code. Lorsque votre fonction est invoquée, Lambda exécute cette méthode.

  Lorsque vous avez créé votre fonction Hello world à l'aide de la console, Lambda a automatiquement défini le nom de la méthode du gestionnaire pour votre fonction sur `lambda_handler`. Veillez à ne pas modifier le nom de cette fonction Python. Dans ce cas, Lambda ne pourra pas exécuter votre code lorsque vous invoquerez votre fonction.

  Pour en savoir plus sur le gestionnaire Lambda en Python, voir the section called “Handler (Gestionnaire)”.

- L'objet de l'événement Lambda :

  La fonction `lambda_handler` accepte deux arguments, `event` et `context`. Dans Lambda, un événement est un document JSON formaté qui contient des données à traiter par votre fonction.

  Si votre fonction est invoquée par une autre personne Service AWS, l'objet d'événement contient des informations sur l'événement à l'origine de l'invocation. Par exemple, si un compartiment Amazon Simple Storage Service (Amazon S3) invoque votre fonction lorsqu'un objet est chargé, l'événement contiendra le nom du compartiment Amazon S3 et la clé de l'objet.

  Dans cet exemple, vous allez créer un événement dans la console en saisissant un document JSON formaté avec deux paires clé-valeur.

- L'objet de contexte Lambda :

Le deuxième argument que votre fonction accepte est `context`. Lambda transmet automatiquement l'objet de contexte à votre fonction. L'objet de contexte contient des informations supplémentaires sur l'invocation de la fonction et l'environnement d'exécution.

Vous pouvez utiliser l'objet de contexte pour générer des informations sur l'invocation de votre fonction à des fins de surveillance. Dans cet exemple, votre fonction utilise le `log_group_name` paramètre pour afficher le nom de son groupe de CloudWatch journaux.

Pour en savoir plus sur l'objet de contexte Lambda en Python, voir the section called “Contexte”.

- Connexion à Lambda :

  Avec Python, vous pouvez utiliser une instruction `print` ou une bibliothèque de journalisation Python pour envoyer des informations au journal de votre fonction. Pour illustrer la différence entre ce qui est capturé, l'exemple de code utilise les deux méthodes. Dans une application de production, nous vous recommandons d'utiliser une bibliothèque de journalisation.

  Pour en savoir plus, consultez the section called “Enregistrez et surveillez les fonctions Lambda en Python”. Pour en savoir plus sur la connexion à d'autres environnements d'exécution, consultez les pages « Génération avec » correspondant aux environnements d'exécution qui vous intéressent.

## Invoquer la fonction Lambda à l'aide de la console

Pour invoquer votre fonction à l'aide de la console Lambda, vous devez d'abord créer un événement de test à envoyer à votre fonction. L'événement est un document JSON formaté contenant deux paires clé-valeur avec les clés et. `"length"` `"width"`

Pour créer l'événement de test

1. Dans le volet Source du code, choisissez Test.
2. Sélectionnez Créer un événement.
3. Dans Nom de l'événement, saisissez **`myTestEvent`**.
4. Dans le JSON panneau Evénement, remplacez les valeurs par défaut en collant les valeurs suivantes :

```
{
```

```
  "length": 6,
  "width": 7
}
```

5. Choisissez Save (Enregistrer).

Vous testez maintenant votre fonction et utilisez la console Lambda et les CloudWatch journaux pour consulter les enregistrements de l'invocation de votre fonction.

Pour tester votre fonction et consulter les enregistrements d'invocation dans la console

- Dans le volet Source du code, choisissez Test. Lorsque votre fonction termine son exécution, les journaux des réponses et des fonctions s'afficheront dans l'onglet Résultats d'exécution. Vous devriez voir des résultats similaires à ce qui suit.

  Node.js

  ```
  Test Event Name
  myTestEvent

  Response
  "{\"area\":42}"

  Function Logs
  START RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Version: $LATEST
  2023-08-31T23:39:45.313Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO The area is
   42
  2023-08-31T23:39:45.331Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO CloudWatch
   log group:  /aws/lambda/myLambdaFunction
  END RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a
  REPORT RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Duration: 20.67 ms Billed
   Duration: 21 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration:
   163.87 ms

  Request ID
  5c012b0a-18f7-4805-b2f6-40912935034a
  ```

  Python

  ```
  Test Event Name
  myTestEvent
  ```

```
Response
"{\"area\": 42}"

Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO] 2023-08-31T23:43:26.428Z 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b CloudWatch
 logs group: /aws/lambda/myLambdaFunction
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Duration: 1.42 ms Billed
 Duration: 2 ms Memory Size: 128 MB Max Memory Used: 39 MB Init Duration: 123.74
 ms

Request ID
2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
```

Dans cet exemple, vous avez invoqué votre code à l'aide de la fonctionnalité de test de la console. Cela signifie que vous pouvez consulter les résultats d'exécution de votre fonction directement dans la console. Lorsque votre fonction est invoquée en dehors de la console, vous devez utiliser CloudWatch Logs.

Pour consulter les enregistrements d'invocation de votre fonction dans Logs CloudWatch

1. Ouvrez la page Groupes de journaux de la CloudWatch console.
2. Choisissez le groupe de journaux de votre fonction (`/aws/lambda/myLambdaFunction`). Il s'agit du nom du groupe de journaux que votre fonction a imprimé sur la console.
3. Dans l'onglet Flux de journaux, choisissez le flux de journaux pour l'invocation de votre fonction.

   Vous devez voir des résultats similaires à ce qui suit :

   Node.js

```
INIT_START Runtime Version: nodejs:20.v13    Runtime Version ARN:
 arn:aws:lambda:us-
west-2::runtime:e3aaabf6b92ef8755eaae2f4bfdcb7eb8c4536a5e044900570a42bdba7b869d9
START RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20 Version: $LATEST
2023-08-23T22:04:15.809Z    5c012b0a-18f7-4805-b2f6-40912935034a  INFO The area
 is 42
2023-08-23T22:04:15.810Z    aba6c0fc-cf99-49d7-a77d-26d805dacd20  INFO
 CloudWatch log group:  /aws/lambda/myLambdaFunction
END RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20
```

```
REPORT RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20    Duration: 17.77 ms
 Billed Duration: 18 ms    Memory Size: 128 MB    Max Memory Used: 67 MB    Init
 Duration: 178.85 ms
```

Python

```
INIT_START Runtime Version: python:3.12.v16    Runtime Version ARN:
 arn:aws:lambda:us-
west-2::runtime:ca202755c87b9ec2b58856efb7374b4f7b655a0ea3deb1d5acc9aee9e297b072
START RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e Version: $LATEST
The area is 42
[INFO] 2023-09-01T00:05:22.464Z 9315ab6b-354a-486e-884a-2fb2972b7d84 CloudWatch
 logs group: /aws/lambda/myLambdaFunction
END RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e
REPORT RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e    Duration: 1.15 ms
 Billed Duration: 2 ms    Memory Size: 128 MB    Max Memory Used: 40 MB
```

# Nettoyage

Lorsque vous avez terminé d'utiliser l'exemple de fonction, supprimez-le. Vous pouvez également supprimer le groupe de journaux qui stocke les journaux de la fonction, et le rôle d'exécution créé par la console.

Pour supprimer une fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Actions, Supprimer.
4. Dans la fenêtre de dialogue Delete function (Supprimer la fonction), saisissez delete, puis sélectionnez Delete (Supprimer).

Pour supprimer le groupe de journaux

1. Ouvrez la page Groupes de journaux de la CloudWatch console.
2. Sélectionnez le groupe de journaux de la fonction (`/aws/lambda/my-function`).
3. Sélectionnez Actions, Delete log group(s) (Supprimer le ou les groupes de journaux).

4. Dans la boîte de dialogue Delete log group(s) (Supprimer le ou les groupes de journaux), sélectionnez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la console AWS Identity and Access Management (IAM).
2. Sélectionnez le rôle d'exécution de la fonction (par exemple, `myLambdaFunction-role-31exxmpl`).
3. Sélectionnez Delete (Supprimer).
4. Dans la fenêtre de dialogue Delete role (Supprimer le rôle), saisissez le nom du rôle, puis sélectionnez Delete (Supprimer).

Vous pouvez automatiser la création et le nettoyage des fonctions, des groupes de journaux et des rôles avec AWS CloudFormation et le AWS Command Line Interface (AWS CLI).

## Ressources supplémentaires et prochaines étapes

Maintenant que vous avez créé et testé une fonction Lambda simple à l'aide de la console, procédez comme suit :

- Découvrez comment ajouter des dépendances à votre code et à les déployer à l'aide d'un package de déploiement .zip. Choisissez parmi les liens suivants les langages qui vous intéressent.

  Node.js

  Consultez the section called “Déploiement d'archives de fichiers .zip”.

  Typescript

  Voir the section called “Déployez des archives de fichiers .zip”

  Python

  Voir the section called “Déploiement d'archives de fichiers .zip”

  Ruby

  Voir the section called “Déploiement d'archives de fichiers .zip”

  Java

  Voir the section called “Déploiement d'archives de fichiers .zip”

Go

Voir the section called “Déploiement d'archives de fichiers .zip”

C#

Consultez the section called “Package de déploiement”

- Suivez le didacticiel Utilisation d'un déclencheur Amazon S3 pour invoquer une fonction Lambda pour savoir comment configurer une fonction Lambda afin qu'elle soit invoquée par un autre Service AWS.
- Choisissez l'un des didacticiels suivants pour un exemple plus complexe d'utilisation de Lambda avec d'autres Services AWS.
  - Utilisation de Lambda avec API Gateway : créez une Amazon API Gateway REST API qui invoque une fonction Lambda.
  - Utilisation d'une fonction Lambda pour accéder à une RDS base de données Amazon : utilisez une fonction Lambda pour écrire des données dans une base de données Amazon Relational Database Service (RDSAmazon) via un proxy. RDS
  - Utilisation d'un déclencheur Amazon S3 pour créer des images miniatures : utilisez une fonction Lambda pour créer une miniature chaque fois qu'un fichier image est chargé dans un compartiment Amazon S3.

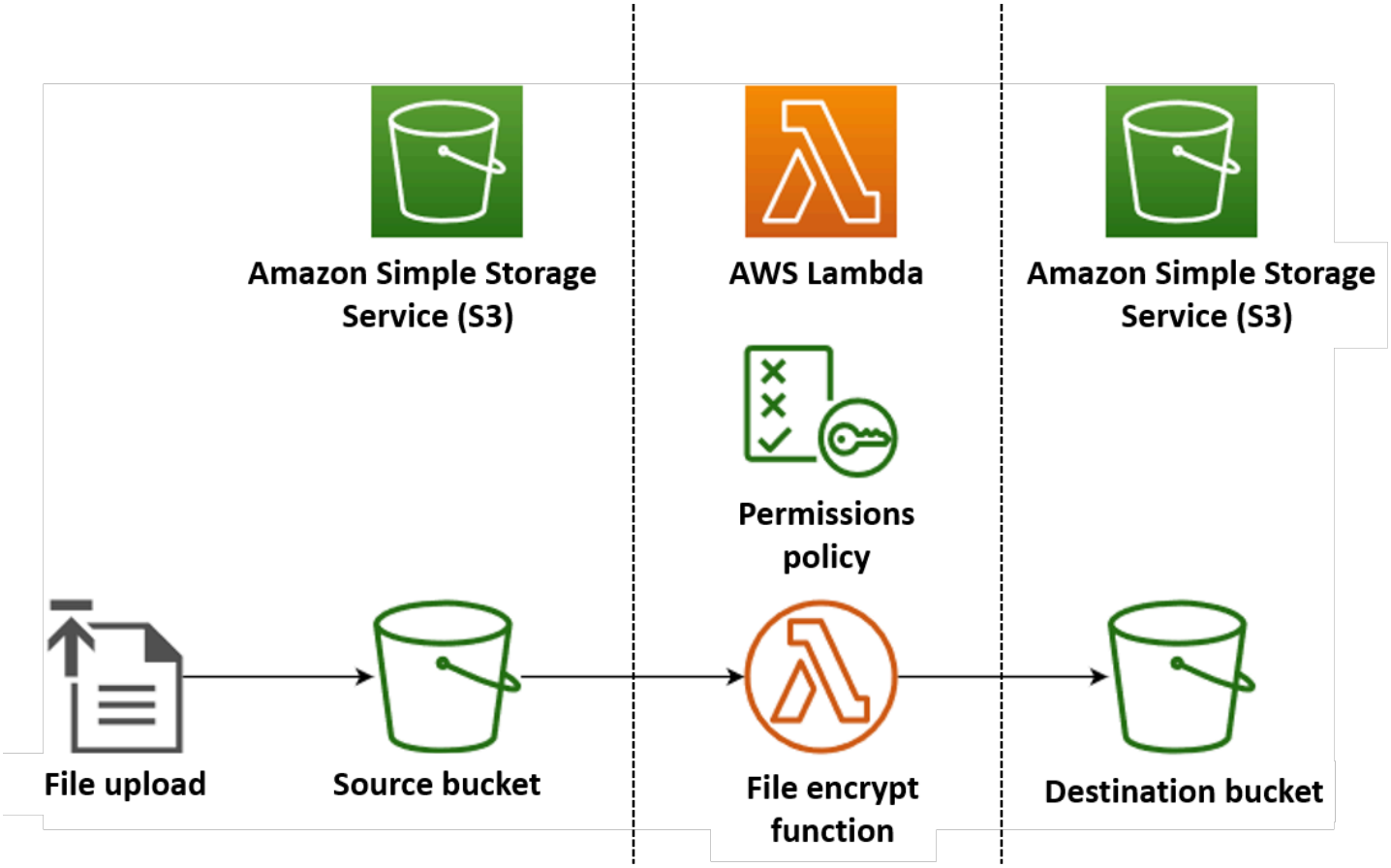
# Création d'une application de traitement de fichiers sans serveur

L'un des cas d'utilisation les plus courants de Lambda consiste à effectuer des tâches de traitement de fichiers. Par exemple, vous pouvez utiliser une fonction Lambda pour créer automatiquement des fichiers à partir de PDF HTML fichiers ou d'images, ou pour créer des miniatures lorsqu'un utilisateur télécharge une image.

Dans cet exemple, vous créez une application qui chiffre automatiquement les PDF fichiers lorsqu'ils sont chargés dans un bucket Amazon Simple Storage Service (Amazon S3). Pour implémenter cette application, vous devez créer les ressources suivantes :

- Un compartiment S3 dans lequel les utilisateurs peuvent télécharger PDF des fichiers
- Une fonction Lambda en Python qui lit le fichier téléchargé et en crée une version cryptée et protégée par mot de passe
- Un deuxième compartiment S3 dans lequel Lambda pourra enregistrer le fichier chiffré

Vous créez également une politique AWS Identity and Access Management (IAM) pour autoriser votre fonction Lambda à effectuer des opérations de lecture et d'écriture sur vos compartiments S3.




> **Tip**
>
> Si vous utilisez Lambda pour la première fois, nous vous recommandons de suivre le didacticiel Créez votre première fonction avant de créer cet exemple d'application.

Vous pouvez déployer votre application manuellement en créant et en configurant des ressources à l'aide du AWS Management Console ou du AWS Command Line Interface (AWS CLI). Vous pouvez également déployer l'application en utilisant le AWS Serverless Application Model (AWS SAM). AWS SAM est un outil d'infrastructure en tant que code (IaC). Avec IaC, vous ne créez pas de ressources manuellement, mais vous les définissez dans le code, puis vous les déployez automatiquement.

Si vous souhaitez en savoir plus sur l'utilisation de Lambda avec IaC avant de déployer cet exemple d'application, consultez. Infrastructure en tant que code (IaC)

## Prérequis

Avant de créer l'exemple d'application, assurez-vous que les outils de ligne de commande requis sont installés.

- AWS CLI

  Vous pouvez déployer manuellement les ressources de votre application à l'aide du AWS Management Console ou du AWS CLI. Pour l'utiliserCLI, installez-le en suivant les instructions d'installation du guide de l'AWS Command Line Interface utilisateur.
- AWS SAM CLI

  Si vous souhaitez déployer l'exemple d'application à l'aide de AWS SAM, vous devez installer à la fois le AWS CLI et le AWS SAM CLI. Pour l'installer AWS SAM CLI, suivez les instructions d'installation du guide de l'AWS SAM utilisateur.
- module pytest

  Après avoir déployé votre application, vous pouvez la tester à l'aide d'un script de test Python automatisé que nous fournissons. Pour utiliser ce script, installez le `pytest` package dans votre environnement de développement local en exécutant la commande suivante :

  ```
  pip install pytest
  ```

Pour déployer l'application à l'aide de Docker AWS SAM, vous devez également être installé sur votre machine de compilation.

## Téléchargement des exemples de fichiers d'application

Pour créer et tester l'exemple d'application, vous devez créer les fichiers suivants dans le répertoire de votre projet :

- `lambda_function.py`- le code de fonction Python pour la fonction Lambda qui effectue le chiffrement des fichiers
- `requirements.txt`- un fichier manifeste définissant les dépendances requises par le code de votre fonction Python
- `template.yaml`- un AWS SAM modèle que vous pouvez utiliser pour déployer l'application

- `test_pdf_encrypt.py`- un script de test que vous pouvez utiliser pour tester automatiquement votre application
- `pytest.ini`- un fichier de configuration pour le script de test

Développez les sections suivantes pour afficher le code et pour en savoir plus sur le rôle de chaque fichier dans la création et le test de votre application. Pour créer les fichiers sur votre machine locale, copiez et collez le code ci-dessous ou téléchargez les fichiers depuis le [aws-lambda-developer-guide GitHub dépôt.]

## Code de fonction Python

Copiez et collez le code suivant dans un fichier nommé`lambda_function.py`.

```
from pypdf import PdfReader, PdfWriter
import uuid
import os
from urllib.parse import unquote_plus
import boto3

# Create the S3 client to download and upload objects from S3
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Iterate over the S3 event object and get the key for all uploaded files
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key']) # Decode the S3 object key to
 remove any URL-encoded characters
        download_path = f'/tmp/{uuid.uuid4()}.pdf' # Create a path in the Lambda tmp
 directory to save the file to
        upload_path = f'/tmp/converted-{uuid.uuid4()}.pdf' # Create another path to
 save the encrypted file to

        # If the file is a PDF, encrypt it and upload it to the destination S3 bucket
        if key.lower().endswith('.pdf'):
            s3_client.download_file(bucket, key, download_path)
            encrypt_pdf(download_path, upload_path)
            encrypted_key = add_encrypted_suffix(key)
            s3_client.upload_file(upload_path, f'{bucket}-encrypted', encrypted_key)

# Define the function to encrypt the PDF file with a password
def encrypt_pdf(file_path, encrypted_file_path):
```

```
    reader = PdfReader(file_path)
    writer = PdfWriter()

    for page in reader.pages:
        writer.add_page(page)

    # Add a password to the new PDF
    writer.encrypt("my-secret-password")

    # Save the new PDF to a file
    with open(encrypted_file_path, "wb") as file:
        writer.write(file)

# Define a function to add a suffix to the original filename after encryption
def add_encrypted_suffix(original_key):
    filename, extension = original_key.rsplit('.', 1)
    return f'{filename}_encrypted.{extension}'
```

> **Note**
>
> Dans cet exemple de code, un mot de passe pour le fichier crypté (`my-secret-password`) est codé en dur dans le code de fonction. Dans une application de production, n'incluez pas d'informations sensibles telles que des mots de passe dans votre code de fonction. AWS Secrets Manager À utiliser pour stocker en toute sécurité les paramètres sensibles.

Le code de la fonction python contient trois fonctions : la fonction de gestion que Lambda exécute lorsque votre fonction est invoquée, et deux fonctions distinctes `add_encrypted_suffix` nommées `encrypt_pdf` et que le gestionnaire appelle pour effectuer le chiffrement. PDF

Lorsque votre fonction est invoquée par Amazon S3, Lambda transmet un argument d'événement JSON formaté à la fonction qui contient des détails sur l'événement à l'origine de l'invocation. Dans ce cas, les informations incluent le nom du compartiment S3 et les clés d'objet pour les fichiers téléchargés. Pour en savoir plus sur le format de l'objet d'événement pour Amazon S3, consultezthe section called “S3”.

Votre fonction utilise ensuite le AWS SDK for Python (Boto3) pour télécharger PDF les fichiers spécifiés dans l'objet d'événement vers son répertoire de stockage temporaire local, avant de les chiffrer à l'aide de la `pypdf`bibliothèque.

Enfin, la fonction utilise le Boto3 SDK pour stocker le fichier chiffré dans votre compartiment de destination S3.

### `requirements.txt`fichier manifeste

Copiez et collez le code suivant dans un fichier nommé`requirements.txt`.

```
boto3
pypdf
```

Dans cet exemple, le code de votre fonction ne comporte que deux dépendances qui ne font pas partie de la bibliothèque Python standard : celle SDK pour Python (Boto3) et le `pypdf` package utilisé par la fonction pour effectuer le PDF chiffrement.

> **Note**
>
> Une version de SDK for Python (Boto3) est incluse dans le runtime Lambda, de sorte que votre code s'exécute sans ajouter Boto3 au package de déploiement de votre fonction. Toutefois, pour garder le contrôle total des dépendances de votre fonction et éviter d'éventuels problèmes de désalignement des versions, la meilleure pratique pour Python consiste à inclure toutes les dépendances des fonctions dans le package de déploiement de votre fonction. Pour en savoir plus, veuillez consulter the section called “Dépendances d'exécution dans Python”.

### AWS SAM modèle

Copiez et collez le code suivant dans un fichier nommé`template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  EncryptPDFFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: EncryptPDF
      Architectures: [x86_64]
      CodeUri: ./
```

```
      Handler: lambda_function.lambda_handler
      Runtime: python3.12
      Timeout: 15
      MemorySize: 256
      LoggingConfig:
        LogFormat: JSON
      Policies:
        - AmazonS3FullAccess
      Events:
        S3Event:
          Type: S3
          Properties:
            Bucket: !Ref PDFSourceBucket
            Events: s3:ObjectCreated:*

  PDFSourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: EXAMPLE-BUCKET

  EncryptedPDFBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: EXAMPLE-BUCKET-encrypted
```

Le AWS SAM modèle définit les ressources que vous créez pour votre application. Dans cet exemple, le modèle définit une fonction Lambda utilisant le `AWS::Serverless::Function` type et deux compartiments S3 utilisant le type. `AWS::S3::Bucket` Les noms de compartiments spécifiés dans le modèle sont des espaces réservés. Avant de déployer l'application à l'aide de AWS SAM, vous devez modifier le modèle pour renommer les compartiments avec des noms uniques au niveau mondial conformes aux règles de dénomination des compartiments S3. Cette étape est expliquée plus loin dansthe section called “Déployez les ressources à l'aide de AWS SAM”.

La définition de la ressource de fonction Lambda configure un déclencheur pour la fonction à l'aide de la propriété event. `S3Event` Ce déclencheur entraîne l'appel de votre fonction chaque fois qu'un objet est créé dans votre compartiment source.

La définition de la fonction spécifie également une politique AWS Identity and Access Management (IAM) à associer au rôle d'exécution de la fonction. La politique AWS gérée `AmazonS3FullAccess` donne à votre fonction les autorisations dont elle a besoin pour lire et écrire des objets sur Amazon S3.

## Script de test automatisé

Copiez et collez le code suivant dans un fichier nommé`test_pdf_encrypt.py`.

```
import boto3
import json
import pytest
import time
import os

@pytest.fixture
def lambda_client():
    return boto3.client('lambda')

@pytest.fixture
def s3_client():
    return boto3.client('s3')

@pytest.fixture
def logs_client():
    return boto3.client('logs')

@pytest.fixture(scope='session')
def cleanup():
    # Create a new S3 client for cleanup
    s3_client = boto3.client('s3')

    yield
    # Cleanup code will be executed after all tests have finished

    # Delete test.pdf from the source bucket
    source_bucket = 'EXAMPLE-BUCKET'
    source_file_key = 'test.pdf'
    s3_client.delete_object(Bucket=source_bucket, Key=source_file_key)
    print(f"\nDeleted {source_file_key} from {source_bucket}")

    # Delete test_encrypted.pdf from the destination bucket
    destination_bucket = 'EXAMPLE-BUCKET-encrypted'
    destination_file_key = 'test_encrypted.pdf'
    s3_client.delete_object(Bucket=destination_bucket, Key=destination_file_key)
    print(f"Deleted {destination_file_key} from {destination_bucket}")


@pytest.mark.order(1)
```

```
def test_source_bucket_available(s3_client):
    s3_bucket_name = 'EXAMPLE-BUCKET'
    file_name = 'test.pdf'
    file_path = os.path.join(os.path.dirname(__file__), file_name)

    file_uploaded = False
    try:
        s3_client.upload_file(file_path, s3_bucket_name, file_name)
        file_uploaded = True
    except:
        print("Error: couldn't upload file")

    assert file_uploaded, "Could not upload file to S3 bucket"



@pytest.mark.order(2)
def test_lambda_invoked(logs_client):

    # Wait for a few seconds to make sure the logs are available
    time.sleep(5)

    # Get the latest log stream for the specified log group
    log_streams = logs_client.describe_log_streams(
        logGroupName='/aws/lambda/EncryptPDF',
        orderBy='LastEventTime',
        descending=True,
        limit=1
    )

    latest_log_stream_name = log_streams['logStreams'][0]['logStreamName']

    # Retrieve the log events from the latest log stream
    log_events = logs_client.get_log_events(
        logGroupName='/aws/lambda/EncryptPDF',
        logStreamName=latest_log_stream_name
    )

    success_found = False
    for event in log_events['events']:
        message = json.loads(event['message'])
        status = message.get('record', {}).get('status')
        if status == 'success':
            success_found = True
```

```
            break

    assert success_found, "Lambda function execution did not report 'success' status in
 logs."

@pytest.mark.order(3)
def test_encrypted_file_in_bucket(s3_client):
    # Specify the destination S3 bucket and the expected converted file key
    destination_bucket = 'EXAMPLE-BUCKET-encrypted'
    converted_file_key = 'test_encrypted.pdf'

    try:
        # Attempt to retrieve the metadata of the converted file from the destination
 S3 bucket
        s3_client.head_object(Bucket=destination_bucket, Key=converted_file_key)
    except s3_client.exceptions.ClientError as e:
        # If the file is not found, the test will fail
        pytest.fail(f"Converted file '{converted_file_key}' not found in the
 destination bucket: {str(e)}")

def test_cleanup(cleanup):
    # This test uses the cleanup fixture and will be executed last
    pass
```

Le script de test automatique exécute trois fonctions de test pour confirmer le bon fonctionnement de votre application :

- Le test `test_source_bucket_available` confirme que votre compartiment source a été créé avec succès en y téléchargeant un PDF fichier de test.
- Le test `test_lambda_invoked` interroge le dernier flux du journal CloudWatch Logs de votre fonction afin de confirmer que lorsque vous avez chargé le fichier de test, votre fonction Lambda s'est exécutée et a signalé un succès.
- Le test `test_encrypted_file_in_bucket` confirme que votre compartiment de destination contient le `test_encrypted.pdf` fichier chiffré.

Une fois tous ces tests exécutés, le script exécute une étape de nettoyage supplémentaire pour supprimer les `test_encrypted.pdf` fichiers `test.pdf` et de vos compartiments source et de destination.

Comme pour le AWS SAM modèle, les noms de compartiments spécifiés dans ce fichier sont des espaces réservés. Avant d'exécuter le test, vous devez modifier ce fichier avec les vrais noms de bucket de votre application. Cette étape est expliquée plus en détail dans the section called “Test de l'application avec le script automatique”

### Fichier de configuration du script de test

Copiez et collez le code suivant dans un fichier nommé`pytest.ini`.

```
[pytest]
markers =
    order: specify test execution order
```

Cela est nécessaire pour spécifier l'ordre dans lequel les tests du `test_pdf_encrypt.py` script s'exécutent.

## Déploiement de l'application

Vous pouvez créer et déployer les ressources pour cet exemple d'application manuellement ou à l'aide de AWS SAM. Dans un environnement de production, nous vous recommandons d'utiliser un outil IaC AWS SAM pour déployer rapidement et de manière répétitive des applications complètes sans serveur sans recourir à des processus manuels.

Pour cet exemple, suivez la console ou AWS CLI les instructions pour savoir comment configurer chaque AWS ressource séparément, ou passez directement the section called “Déployez les ressources à l'aide de AWS SAM” à pour déployer rapidement l'application à l'aide de quelques CLI commandes.

### Déployez les ressources manuellement

Pour déployer votre application manuellement, vous devez suivre les étapes suivantes :

- Création de compartiments S3 source et de destination
- Créez une fonction Lambda qui chiffre un PDF fichier et enregistre la version chiffrée dans un compartiment S3
- Configurez un déclencheur Lambda qui appelle votre fonction lorsque des objets sont téléchargés dans votre compartiment source

Suivez les instructions des paragraphes suivants pour créer et configurer vos ressources.

## Création de deux compartiments S3

Créez d'abord deux compartiments S3. Le premier compartiment est le compartiment source dans lequel vous allez télécharger vos PDF fichiers. Le deuxième compartiment est utilisé par Lambda pour enregistrer le fichier chiffré lorsque vous appelez votre fonction.

Console

Pour créer les compartiments S3 (console)

1. Ouvrez la page Compartiments de la console Amazon S3.
2. Choisissez Create bucket (Créer un compartiment).
3. Sous Configuration générale, procédez comme suit :
   a. Pour le nom du compartiment, saisissez un nom unique au monde qui respecte les règles de dénomination du compartiment Amazon S3. Les noms de compartiment peuvent contenir uniquement des lettres minuscules, des chiffres, de points (.) et des traits d'union (-).
   b. Pour Région AWS, choisissez la Région AWS la plus proche de votre emplacement géographique. Plus tard dans le processus de déploiement, vous devrez y créer votre fonction Lambda. Région AWS Notez donc la région que vous avez choisie.
4. Conservez les valeurs par défaut de toutes les autres options et choisissez Créer un compartiment.
5. Répétez les étapes 1 à 4 pour créer votre compartiment de destination. Pour Nom du compartiment, saisissez **`SOURCEBUCKET-encrypted`**, où **`SOURCEBUCKET`** est le nom du compartiment source que vous venez de créer.

AWS CLI

Pour créer les compartiments S3 ()AWS CLI

1. Exécutez la CLI commande suivante pour créer votre compartiment source. Le nom que vous choisissez pour votre compartiment doit être unique au monde et respecter les Règles de dénomination du compartiment Amazon S3. Les noms peuvent contenir uniquement des lettres minuscules, des chiffres, de points (.) et des traits d'union (-). Pour `region` et `LocationConstraint`, choisissez la Région AWS la plus proche de votre emplacement géographique.

```
aws s3api create-bucket --bucket SOURCEBUCKET --region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

Plus loin dans le didacticiel, vous devez créer votre fonction Lambda dans le même Région AWS emplacement que votre compartiment source. Notez donc la région que vous avez choisie.

2. Exécutez la commande suivante pour créer votre compartiment de destination. Pour le nom du compartiment, vous devez utiliser **SOURCEBUCKET-encrypted**, où **SOURCEBUCKET** est le nom du compartiment source que vous avez créé à l'étape 1. Pour `region` et`LocationConstraint`, choisissez le même Région AWS que celui que vous avez utilisé pour créer votre bucket source.

```
aws s3api create-bucket --bucket SOURCEBUCKET-encrypted --region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

## Création d'un rôle d'exécution (AWS CLI uniquement)

Un rôle d'exécution est un IAM rôle qui accorde à une fonction Lambda l'autorisation d'accès Services AWS et de ressources. Lorsque vous créez une fonction à l'aide de la console Lambda, Lambda crée automatiquement un rôle d'exécution. Vous devez uniquement créer un rôle manuellement si vous choisissez de déployer l'application à l'aide du AWS CLI. Pour donner à votre fonction un accès en lecture et en écriture à Amazon S3, vous devez joindre la politique AWS gérée`AmazonS3FullAccess`.

Console

Cette étape n'est requise que si vous choisissez de déployer votre application à l'aide du AWS CLI.

AWS CLI

Pour créer un rôle d'exécution et associer la politique **AmazonS3FullAccess** gérée (AWS CLI)

1. Enregistrez ce qui suit JSON dans un fichier nommé`trust-policy.json`. Cette politique de confiance permet à Lambda d'utiliser les autorisations du rôle en autorisant le principal `lambda.amazonaws.com` du service à appeler l'action AWS Security Token Service (AWS STS)`AssumeRole`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Dans le répertoire dans lequel vous avez enregistré le document de politique de JSON confiance, exécutez la CLI commande suivante pour créer le rôle d'exécution.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
 file://trust-policy.json
```

3. Pour associer la politique `AmazonS3FullAccess` gérée, exécutez la CLI commande suivante.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::aws:policy/AmazonS3FullAccess
```

## Créer le package de déploiement de la fonction

Pour créer votre fonction, vous créez un package de déploiement contenant le code de votre fonction et ses dépendances. Pour cette application, votre code de fonction utilise une bibliothèque séparée pour le PDF chiffrement.

Pour créer le package de déploiement

1. Accédez au répertoire du projet contenant les `requirements.txt` fichiers `lambda_function.py` et que vous avez créés ou téléchargés GitHub précédemment et créez un nouveau répertoire nommé`package`.
2. Installez les dépendances spécifiées dans le `requirements.txt` fichier de votre `package` répertoire en exécutant la commande suivante.

```
pip install -r requirements.txt --target ./package/
```

3. Créez un fichier .zip contenant le code de votre application et ses dépendances. Sous Linux ou macOS, exécutez les commandes suivantes depuis votre interface de ligne de commande.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

Sous Windows, utilisez l'outil zip de votre choix pour créer le fichier `lambda_function.zip`. Assurez-vous que votre fichier `lambda_function.py` et les dossiers contenant vos dépendances sont installés à la racine du fichier .zip.

Vous pouvez également créer votre package de déploiement à l'aide d'un environnement virtuel Python. Consultez Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python.

## Créer la fonction Lambda

Vous utilisez maintenant le package de déploiement que vous avez créé à l'étape précédente pour déployer votre fonction Lambda.

Console

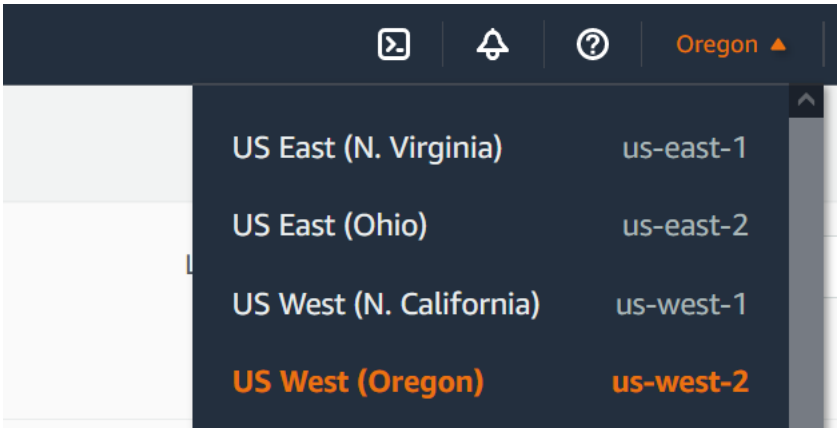
Pour créer une fonction (console)

Pour créer votre fonction Lambda à l'aide de la console, vous devez d'abord créer une fonction de base contenant du code « Hello world ». Vous remplacez ensuite ce code par votre propre code de fonction en téléchargeant le fichier the.zip que vous avez créé à l'étape précédente.

Pour garantir que votre fonction n'expire pas lors du chiffrement de PDF fichiers volumineux, configurez les paramètres de mémoire et de délai d'expiration de la fonction. Vous définissez également le format du journal de la fonction surJSON. La configuration de journaux JSON formatés est nécessaire lors de l'utilisation du script de test fourni afin que celui-ci puisse lire l'état d'invocation de la fonction dans CloudWatch Logs pour confirmer la réussite de l'appel.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Assurez-vous de travailler dans le même environnement que celui dans Région AWS lequel vous avez créé votre compartiment S3. Vous pouvez modifier votre région à l'aide de la liste déroulante en haut de l'écran.




3. Sélectionnez Create function (Créer une fonction).
4. Choisissez Créer à partir de zéro.
5. Sous Informations de base, procédez comme suit :
   a. Sous Nom de la fonction, saisissez **`EncryptPDF`**.
   b. Pour Runtime, choisissez Python 3.12.
   c. Pour Architecture, choisissez x86_64.
6. Sélectionnez Create function (Créer une fonction).

Pour charger le code de fonction (console)

1. Dans le volet Source du code, choisissez Charger à partir de.
2. Choisissez Fichier .zip.
3. Sélectionnez Charger.
4. Dans le sélecteur de fichiers, sélectionnez votre fichier .zip et choisissez Ouvrir.
5. Choisissez Save (Enregistrer).

Pour configurer la mémoire des fonctions et le délai d'attente (console)

1. Sélectionnez l'onglet Configuration correspondant à votre fonction.
2. Dans le volet Configuration générale, choisissez Modifier.
3. Réglez la mémoire sur 256 Mo et le délai d'expiration sur 15 secondes.

4. Choisissez Save (Enregistrer).

Pour configurer le format du journal (console)

1. Sélectionnez l'onglet Configuration correspondant à votre fonction.
2. Sélectionnez Outils de surveillance et d'exploitation.
3. Dans le volet de configuration de la journalisation, choisissez Modifier.
4. Pour la configuration de la journalisation, sélectionnez JSON.
5. Choisissez Save (Enregistrer).

AWS CLI

Pour créer la fonction (AWS CLI)

- Exécutez la commande suivante depuis le répertoire contenant votre `lambda_function.zip` fichier. Pour le `region` paramètre, remplacez-le par `us-west-2` la région dans laquelle vous avez créé vos compartiments S3.

```
aws lambda create-function --function-name EncryptPDF \
--zip-file fileb://lambda_function.zip --handler lambda_function.lambda_handler
 \
--runtime python3.12 --timeout 15 --memory-size 256 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-west-2 \
--logging-config LogFormat=JSON
```

## Configurer un déclencheur Amazon S3 pour appeler la fonction

Pour que votre fonction Lambda s'exécute lorsque vous téléchargez un fichier dans votre compartiment source, vous devez configurer un déclencheur pour votre fonction. Vous pouvez configurer le déclencheur Amazon S3 à l'aide de la console ou de la AWS CLI.

> **Important**
>
> Cette procédure configure le compartiment S3 pour qu'il invoque votre fonction chaque fois qu'un objet est créé dans le compartiment. Veillez à configurer cela uniquement pour le compartiment source. Si votre fonction Lambda crée des objets dans le même compartiment

que celui qui l'invoque, votre fonction peut être invoquée en continu dans une boucle. Cela peut entraîner la facturation de frais imprévus à votre Compte AWS charge.

Console

Pour configurer le déclencheur Amazon S3 (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction (`EncryptPDF`).
2. Choisissez Add trigger (Ajouter déclencheur).
3. Sélectionnez S3.
4. Sous Compartiment, sélectionnez votre compartiment source.
5. Sous Types d'événements, sélectionnez Tous les événements de création d'objets.
6. Sous Invocation récursive, cochez la case pour confirmer qu'il n'est pas recommandé d'utiliser le même compartiment S3 pour les entrées et les sorties. Vous pouvez en savoir plus sur les modèles d'invocation récursive dans Lambda en lisant la rubrique Modèles récursifs qui provoquent des fonctions Lambda incontrôlables dans Serverless Land.
7. Choisissez Ajouter.

   Lorsque vous créez un déclencheur à l'aide de la console Lambda, ce dernier crée automatiquement une politique basée sur une ressource pour donner au service que vous sélectionnez l'autorisation d'invoquer votre fonction.

AWS CLI

Pour configurer le déclencheur Amazon S3 (AWS CLI)

1. Pour que votre compartiment source Amazon S3 invoque votre fonction lorsque vous ajoutez un fichier, vous devez d'abord configurer les autorisations pour votre fonction à l'aide d'une politique basée sur les ressources. Une déclaration de politique basée sur les ressources donne à d'autres personnes Services AWS l'autorisation d'invoquer votre fonction. Pour autoriser Amazon S3 à appeler votre fonction, exécutez la CLI commande suivante. Assurez-vous de remplacer le `source-account` paramètre par votre propre Compte AWS identifiant et d'utiliser votre propre nom de compartiment source.

   ```
   aws lambda add-permission --function-name EncryptPDF \
   ```

```
--principal s3.amazonaws.com --statement-id s3invoke --action
 "lambda:InvokeFunction" \
--source-arn arn:aws:s3:::SOURCEBUCKET \
--source-account 123456789012
```

La politique que vous définissez à l'aide de cette commande permet à Amazon S3 d'invoquer votre fonction uniquement lorsqu'une action a lieu sur votre compartiment source.

> **Note**
>
> Bien que les noms des compartiments S3 soient uniques à l'échelle mondiale, lorsque vous utilisez des politiques basées sur les ressources, il est recommandé de spécifier que le compartiment doit appartenir à votre compte. En effet, si vous supprimez un bucket, il est possible qu'un autre Compte AWS en crée un avec le même Amazon Resource Name (ARN).

2. Enregistrez ce qui suit JSON dans un fichier nomméNOTIFICATION.JSON. Lorsqu'il est appliqué à votre compartiment source, cela JSON configure le compartiment pour qu'il envoie une notification à votre fonction Lambda chaque fois qu'un nouvel objet est ajouté. Remplacez le Compte AWS numéro et Région AWS dans la fonction Lambda par votre propre numéro ARN de compte et votre propre région.

```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "EncryptPDFEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:EncryptPDF",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Exécutez la CLI commande suivante pour appliquer les paramètres de notification du JSON fichier que vous avez créé à votre compartiment source. Remplacez SOURCEBUCKET par le nom de votre propre compartiment source.

```
aws s3api put-bucket-notification-configuration --bucket SOURCEBUCKET \
--notification-configuration file://notification.json
```

Pour en savoir plus sur la `put-bucket-notification-configuration` commande et l'`notification-configuration`option, consultez le put-bucket-notification-configurationmanuel de référence des AWS CLI commandes.

## Déployez les ressources à l'aide de AWS SAM

Pour déployer l'exemple d'application à l'aide du AWS SAM CLI, procédez comme suit.

Assurez-vous que vous avez installé la dernière version de Docker CLI et que Docker est installé sur votre machine de compilation.

1. Modifiez le `template.yaml` fichier pour spécifier le nom de vos compartiments S3. Les compartiments S3 doivent avoir des noms uniques au niveau mondial conformes aux règles de dénomination des compartiments S3.

   Remplacez le nom du compartiment `EXAMPLE-BUCKET` par le nom de votre choix composé de lettres minuscules, de chiffres, de points (.) et de tirets (-). Pour le compartiment de destination, remplacez `EXAMPLE-BUCKET-encrypted` par`<source-bucket-name>-encrypted`, où `<source-bucket>` est le nom que vous avez choisi pour votre compartiment source.
2. Exécutez la commande suivante depuis le répertoire dans lequel vous avez enregistré vos `requirements.txt` fichiers `template.yaml``lambda_function.py`,, et.

   ```
   sam build --use-container
   ```

   Cette commande rassemble les artefacts de compilation pour votre application et les place dans le format et l'emplacement appropriés pour les déployer. La spécification de l'`--use-container`option crée votre fonction dans un conteneur Docker de type Lambda. Nous l'utilisons ici, vous n'avez donc pas besoin d'installer Python 3.12 sur votre machine locale pour que la compilation fonctionne.

   Pendant le processus de génération, AWS SAM recherche le code de la fonction Lambda à l'emplacement que vous avez spécifié avec la `CodeUri` propriété dans le modèle. Dans ce cas, nous avons spécifié le répertoire actuel comme emplacement (`./`).

   Si un `requirements.txt` fichier est présent, AWS SAM utilisez-le pour rassembler les dépendances spécifiées. Par défaut, AWS SAM crée un package de déploiement .zip avec le

code de votre fonction et ses dépendances. Vous pouvez également choisir de déployer votre fonction sous forme d'image de conteneur à l'aide de la PackageTypepropriété.

3. Pour déployer votre application et créer les ressources Lambda et Amazon S3 spécifiées dans votre AWS SAM modèle, exécutez la commande suivante.

```
sam deploy --guided
```

L'utilisation du `--guided` drapeau signifie que des instructions vous AWS SAM seront affichées pour vous guider tout au long du processus de déploiement. Pour ce déploiement, acceptez les options par défaut en appuyant sur Entrée.

Au cours du processus de déploiement, AWS SAM crée les ressources suivantes dans votre Compte AWS :

- Une AWS CloudFormation pile nommée `sam-app`
- Une fonction Lambda portant le nom `EncryptPDF`
- Deux compartiments S3 portant les noms que vous avez choisis lorsque vous avez modifié le `template.yaml` AWS SAM fichier modèle
- Un rôle IAM d'exécution pour votre fonction avec le format de nom `sam-app-EncryptPDFFunctionRole-`*`2qGaapHFWOQ8`*

Lorsque vous avez AWS SAM terminé de créer vos ressources, le message suivant devrait s'afficher :

```
Successfully created/updated stack - sam-app in us-west-2
```

## Tester l'application

Pour tester votre application, vous téléchargez un PDF fichier dans votre compartiment source et vous confirmez que Lambda crée une version cryptée du fichier dans votre compartiment de destination. Dans cet exemple, vous pouvez le tester manuellement à l'aide de la AWS CLI console ou du script de test fourni.

Pour les applications de production, vous pouvez utiliser des méthodes et techniques de test traditionnelles, telles que les tests unitaires, pour confirmer le bon fonctionnement de votre code de

fonction Lambda. La meilleure pratique consiste également à effectuer des tests tels que ceux du script de test fourni, qui effectuent des tests d'intégration avec de vraies ressources basées sur le cloud. Les tests d'intégration dans le cloud confirment que votre infrastructure a été correctement déployée et que les événements circulent entre les différents services comme prévu. Pour en savoir plus, consultez Tester les fonctions sans serveur.

## Tester l'application manuellement

Vous pouvez tester votre fonction manuellement en ajoutant un PDF fichier à votre compartiment source Amazon S3. Lorsque vous ajoutez votre fichier au compartiment source, votre fonction Lambda doit être automatiquement invoquée et doit stocker une version chiffrée du fichier dans votre compartiment cible.

Console

Pour tester votre application en chargeant un fichier (console)

1. Pour charger un PDF fichier dans votre compartiment S3, procédez comme suit :
   a. Ouvrez la page Compartiments de la console Amazon S3 et choisissez votre compartiment source.
   b. Sélectionnez Charger.
   c. Choisissez Ajouter des fichiers et utilisez le sélecteur de fichiers pour choisir le PDF fichier que vous souhaitez télécharger.
   d. Choisissez Ouvrir, puis Charger.
2. Vérifiez que Lambda a enregistré une version chiffrée de votre PDF fichier dans votre compartiment cible en procédant comme suit :
   a. Retournez à la page Compartiments de la console Amazon S3 et choisissez votre compartiment de destination.
   b. Dans le volet Objets, vous devriez maintenant voir un fichier au format de nom `filename_encrypted.pdf` (où `filename.pdf` était le nom du fichier que vous avez chargé dans votre compartiment source). Pour télécharger votre fichier cryptéPDF, sélectionnez le fichier, puis choisissez Télécharger.
   c. Vérifiez que vous pouvez ouvrir le fichier téléchargé avec le mot de passe avec lequel votre fonction Lambda l'a protégé ()`my-secret-password`.

AWS CLI

Pour tester votre application en téléchargeant un fichier ()AWS CLI

1. Dans le répertoire contenant le PDF fichier que vous souhaitez télécharger, exécutez la CLI commande suivante. Remplacez le paramètre `--bucket` par le nom de votre compartiment source. Pour les `--body` paramètres `--key` et, utilisez le nom de fichier de votre fichier de test.

```
aws s3api put-object --bucket SOURCEBUCKET --key test.pdf --body ./test.pdf
```

2. Vérifiez que votre fonction a créé une version cryptée de votre fichier et l'a enregistrée dans votre compartiment S3 cible. Exécutez la CLI commande suivante en la `SOURCEBUCKET-encrypted` remplaçant par le nom de votre propre compartiment de destination.

```
aws s3api list-objects-v2 --bucket SOURCEBUCKET-encrypted
```

   Si votre fonction s'exécute correctement, vous obtiendrez un résultat similaire à celui qui suit. Votre compartiment cible doit contenir un fichier au format de nom`<your_test_file>_encrypted.pdf`, où `<your_test_file>` est le nom du fichier que vous avez chargé.

```
{
    "Contents": [
        {
            "Key": "test_encrypted.pdf",
            "LastModified": "2023-06-07T00:15:50+00:00",
            "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
            "Size": 2763,
            "StorageClass": "STANDARD"
        }
    ]
}
```

3. Pour télécharger le fichier que Lambda a enregistré dans votre compartiment de destination, exécutez la commande suivanteCLI. Remplacez le `--bucket` paramètre par le nom de votre compartiment de destination. Pour le `--key` paramètre, utilisez le nom de fichier`<your_test_file>_encrypted.pdf`, qui `<your_test_file>` est le nom du fichier de test que vous avez chargé.

```
aws s3api get-object --bucket SOURCEBUCKET-encrypted --key test_encrypted.pdf
 my_encrypted_file.pdf
```

Cette commande télécharge le fichier dans votre répertoire actuel et l'enregistre sous le nom`my_encrypted_file.pdf`.

4. Vérifiez que vous pouvez ouvrir le fichier téléchargé avec le mot de passe avec lequel votre fonction Lambda l'a protégé ()`my-secret-password`.

## Test de l'application avec le script automatique

Pour tester votre application à l'aide du script de test fourni, assurez-vous d'abord que le `pytest` module est installé dans votre environnement local. Vous pouvez effectuer l'installation en `pytest` exécutant la commande suivante :

```
pip install pytest
```

Vous devez également modifier le code du `test_pdf_encrypt.py` fichier pour remplacer les noms de compartiment réservés par les noms de vos compartiments source et de destination Amazon S3. Apportez les modifications suivantes à `test_pdf_encrypt.py` :

- Dans la `test_source_bucket_available` fonction, remplacez `EXAMPLE-BUCKET` par le nom de votre compartiment source.
- Dans la `test_encrypted_file_in_bucket` fonction, remplacez `EXAMPLE-BUCKET-encrypted` par`<source-bucket>-encrypted`, où se `<source-bucket>` trouve le nom de votre compartiment source.
- Dans la `cleanup` fonction, remplacez `EXAMPLE-BUCKET` par le nom de votre compartiment source et remplacez `EXAMPLE-BUCKET-encrypted` par`#source-bucket>-encrypted`, où se `<source-bucket>` trouve le nom de votre compartiment source.

Pour exécuter les tests, procédez comme suit :

- Enregistrez un PDF fichier nommé `test.pdf` dans le répertoire contenant les `pytest.ini` fichiers `test_pdf_encrypt.py` et.
- Ouvrez un terminal ou un programme shell et exécutez la commande suivante depuis le répertoire contenant les fichiers de test.

```
pytest -s -v
```

Une fois le test terminé, vous devriez voir un résultat comme celui-ci :

```
============================================================= test session starts
 ==============================================================
platform linux -- Python 3.12.2, pytest-7.2.2, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/
pdf_encrypt_app/.hypothesis/examples')
Test order randomisation NOT enabled. Enable with --random-order or --random-order-
bucket=<bucket_type>
rootdir: /home/pdf_encrypt_app, configfile: pytest.ini
plugins: anyio-3.7.1, hypothesis-6.70.0, localserver-0.7.1, random-order-1.1.0
collected 4 items

test_pdf_encrypt.py::test_source_bucket_available PASSED
test_pdf_encrypt.py::test_lambda_invoked PASSED
test_pdf_encrypt.py::test_encrypted_file_in_bucket PASSED
test_pdf_encrypt.py::test_cleanup PASSED
Deleted test.pdf from EXAMPLE-BUCKET
Deleted test_encrypted.pdf from EXAMPLE-BUCKET-encrypted


============================================================== 4 passed in 7.32s
 ==============================================================
```

# Étapes suivantes

Maintenant que vous avez créé cet exemple d'application, vous pouvez utiliser le code fourni comme base pour créer d'autres types d'applications de traitement de fichiers. Modifiez le code du `lambda_function.py` fichier pour implémenter la logique de traitement de fichiers adaptée à votre cas d'utilisation.

De nombreux cas d'utilisation typiques du traitement de fichiers impliquent le traitement d'image. Lorsque vous utilisez Python, les bibliothèques de traitement d'image les plus populaires telles que pillow contiennent généralement des composants C ou C++. Afin de garantir que le package

de déploiement de votre fonction est compatible avec l'environnement d'exécution Lambda, il est important d'utiliser le bon binaire de distribution source.

Lorsque vous déployez vos ressources avec AWS SAM, vous devez prendre des mesures supplémentaires pour inclure la distribution source appropriée dans votre package de déploiement. Étant donné que les dépendances AWS SAM ne seront pas installées pour une plate-forme différente de celle de votre machine de génération, la spécification de la distribution source (`.whl`fichier) correcte dans votre `requirements.txt` fichier ne fonctionnera pas si votre machine de génération utilise un système d'exploitation ou une architecture différent de l'environnement d'exécution Lambda. Vous devez plutôt effectuer l'une des opérations suivantes :

- Utilisez l'`--use-container`option lors de l'exécution`sam build`. Lorsque vous spécifiez cette option, AWS SAM télécharge une image de base de conteneur compatible avec l'environnement d'exécution Lambda et crée le package de déploiement de votre fonction dans un conteneur Docker à l'aide de cette image. Pour en savoir plus, voir Création d'une fonction Lambda à l'intérieur d'un conteneur fourni.
- Créez vous-même le package de déploiement .zip de votre fonction en utilisant le binaire de distribution source approprié et enregistrez le fichier .zip dans le répertoire que vous spécifiez `CodeUri` dans le AWS SAM modèle. Pour en savoir plus sur la création de packages de déploiement .zip pour Python à l'aide de distributions binaires, consultez the section called “Création d'un package de déploiement .zip avec dépendances” etthe section called “Création de packages de déploiement .zip avec des bibliothèques natives”.

# Comprendre les concepts clés de Lambda

Lambda est un service de calcul piloté par les événements qui exécute des instances de votre fonction pour traiter les événements. Vous pouvez appeler votre fonction directement à l'aide du LambdaAPI, ou vous pouvez configurer un AWS service ou une ressource pour appeler votre fonction.

Les sections suivantes décrivent certains concepts clés des fonctions Lambda, du modèle de programmation piloté par les événements et de l'environnement dans lequel votre fonction s'exécute.

Concepts

- Fonction
- Déclencheur
- Événement
- Environnement d’exécution
- Package de déploiement
- Environnement d’exécution
- Couche
- Concurrency
- Qualificateur
- Destination

## Fonction

Une fonction est une ressource que vous pouvez appeler pour exécuter votre code dans Lambda. Une fonction possède un code pour traiter les événements que vous transmettez à la fonction ou que d'autres AWS services lui envoient.

## Déclencheur

Un déclencheur est une ressource ou une configuration qui appelle une fonction Lambda. Les déclencheurs incluent les services AWS que vous pouvez configurer pour appeler une fonction et les mappages de source d’événement. Un mappage de source d’événement est une ressource dans Lambda qui lit les éléments d’un flux ou d’une file d’attente et appelle une fonction. Pour plus

d'informations, consultez Comprendre les méthodes d'appel de fonctions Lambda et Invoquer Lambda avec des événements provenant d'autres services AWS.

# Événement

Un événement est un document JSON formaté qui contient des données à traiter par une fonction Lambda. Le runtime convertit l'événement en objet et le transmet au code de votre fonction. Lorsque vous appelez une fonction, vous déterminez la structure et le contenu de l'événement.

Example événement personnalisé – données météorologiques

```
{
  "TemperatureK": 281,
  "WindKmh": -3,
  "HumidityPct": 0.55,
  "PressureHPa": 1020
}
```

Lorsqu'un AWS service invoque votre fonction, il définit la forme de l'événement.

Example événement de service — SNS notification Amazon

```
{
  "Records": [
    {
      "Sns": {
        "Timestamp": "2019-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        ...
```

Pour plus d'informations sur les événements liés AWS aux services, consultezInvoquer Lambda avec des événements provenant d'autres services AWS.

# Environnement d'exécution

Un environnement d'exécution fournit un environnement de runtime sécurisé et isolé pour votre fonction Lambda. Un environnement d'exécution gère les processus et les ressources nécessaires

à l'exécution de la fonction. L'environnement d'exécution prend en charge le cycle de vie pour la fonction et pour toutes les extensions associées à votre fonction.

Pour de plus amples informations, veuillez consulter Comprendre le cycle de vie de l'environnement d'exécution Lambda.

## Package de déploiement

Vous déployez votre code de fonction Lambda à l'aide d'un package de déploiement. Lambda prend en charge deux types de packages de déploiement :

- Une archive de fichiers .zip qui contient le code et les dépendances de votre fonction. Lambda fournit le système d'exploitation et le runtime pour votre fonction. Pour de plus amples informations, veuillez consulter Déploiement des fonctions Lambda comme des archives de fichiers .zip.
- Une image de conteneur compatible avec la spécification Open Container Initiative (OCI). Vous ajoutez votre code de fonction et ses dépendances à l'image. Vous devez également inclure le système d'exploitation et un runtime Lambda. Pour de plus amples informations, veuillez consulter Création d'une fonction Lambda à l'aide d'une image de conteneur.

## Environnement d'exécution

Le runtime fournit un environnement spécifique au langage qui s'exécute dans un environnement d'exécution. Le runtime relaie les événements d'appel, les informations de contexte et les réponses entre Lambda et la fonction. Vous pouvez utiliser les runtimes que Lambda fournit, ou créer vos propres runtimes. Si vous conditionnez votre code sous la forme d'une archive de fichiers .zip, vous devez configurer votre fonction pour utiliser un runtime correspondant à votre langage de programmation. Pour une image de conteneur, vous incluez le runtime lorsque vous créez l'image.

Pour de plus amples informations, veuillez consulter Environnements d'exécution (runtimes) Lambda.

## Couche

Une couche Lambda est une archive de fichiers .zip qui peut contenir du code supplémentaire ou autre chose. Une couche peut contenir des bibliothèques, un environnement d'exécution personnalisé, des données ou des fichiers de configuration.

Les couches sont un moyen pratique de conditionner en package des bibliothèques et d'autres dépendances que vous pouvez utiliser avec vos fonctions Lambda. L'utilisation de couches réduit la

taille des archives de déploiement chargées et accélère le déploiement de votre code. Les couches favorisent également le partage de code et la séparation des responsabilités afin que vous puissiez effectuer plus rapidement des itérations lors de l'écriture de la logique métier.

Vous pouvez inclure jusqu'à cinq couches par fonction. Les couches sont comptabilisées dans les quotas de taille de déploiement Lambda standard. Lorsque vous incluez une couche dans une fonction, le contenu est extrait dans le `/opt` répertoire de l'environnement d'exécution.

Par défaut, les couches que vous créez sont privées pour votre AWS compte. Vous pouvez choisir de partager une couche avec d'autres comptes ou de la rendre publique. Si vos fonctions consomment une couche publiée par un autre compte, vos fonctions peuvent continuer à utiliser la version de la couche après qu'elle a été supprimée ou après la révocation de votre autorisation d'accès à la couche. Toutefois, vous ne pouvez pas créer de nouvelle fonction ou mettre à jour des fonctions à l'aide d'une version de couche supprimée.

Les fonctions déployées en tant qu'image de conteneur n'utilisent pas de couches. En lieu et place, vous conditionnez vos runtimes, bibliothèques et autres dépendances préférés en image de conteneur, lorsque vous créez l'image.

Pour de plus amples informations, veuillez consulter Couches Lambda.

## Concurrency

La simultanéité est le nombre de demandes auxquelles votre fonction répond à un moment donné. Lors de l'appel de votre fonction, Lambda approvisionne une instance de celle-ci pour traiter l'événement. Lorsque le code de la fonction a fini de s'exécuter, il peut gérer une autre demande. Si la fonction est appelée à nouveau alors qu'une demande est toujours en cours de traitement, une autre instance est fournie, ce qui augmente la simultanéité de la fonction.

La simultanéité est soumise à des quotas au niveau de la AWS région. Vous pouvez configurer des fonctions individuelles pour limiter leur simultanéité ou leur permettre d'atteindre un niveau spécifique de simultanéité. Pour de plus amples informations, veuillez consulter Configuration de la simultanéité réservée pour une fonction.

## Qualificateur

Lorsque vous appelez ou affichez une fonction, vous pouvez inclure un qualificateur pour spécifier une version ou un alias. Une version est un instantané immuable du code et de la configuration d'une

fonction ayant un qualificateur numérique. Par exemple, `my-function:1`. Un alias est un pointeur vers une version que vous pouvez mettre à jour pour un mappage avec une version différente, ou pour le fractionnement du trafic entre deux versions. Par exemple, `my-function:BLUE`. Vous pouvez utiliser des versions et des alias ensemble pour fournir une interface stable permettant aux clients d'appeler votre fonction.

Pour de plus amples informations, veuillez consulter Gérer les versions des fonctions Lambda.

## Destination

Une destination est une AWS ressource dans laquelle Lambda peut envoyer des événements à partir d'un appel asynchrone. Vous pouvez configurer une destination pour les événements dont le traitement échoue. Certains services prennent également en charge une destination pour les événements dont le traitement aboutit.

Pour de plus amples informations, veuillez consulter Ajouter une destination.

# Utilisation de Lambda avec infrastructure en tant que code (IaC)

Les fonctions Lambda s'exécutent rarement de manière isolée. Au lieu de cela, ils font souvent partie d'une application sans serveur avec d'autres ressources telles que les bases de données, les files d'attente et le stockage. Grâce à l'infrastructure sous forme de code (IaC), vous pouvez automatiser vos processus de déploiement afin de déployer et de mettre à jour rapidement et de manière répétée des applications sans serveur complètes impliquant de nombreuses ressources distinctes. AWS Cette approche accélère votre cycle de développement, facilite la gestion des configurations et garantit que vos ressources sont déployées de la même manière à chaque fois.

## Outils de l'IaC pour Lambda

AWS CloudFormation

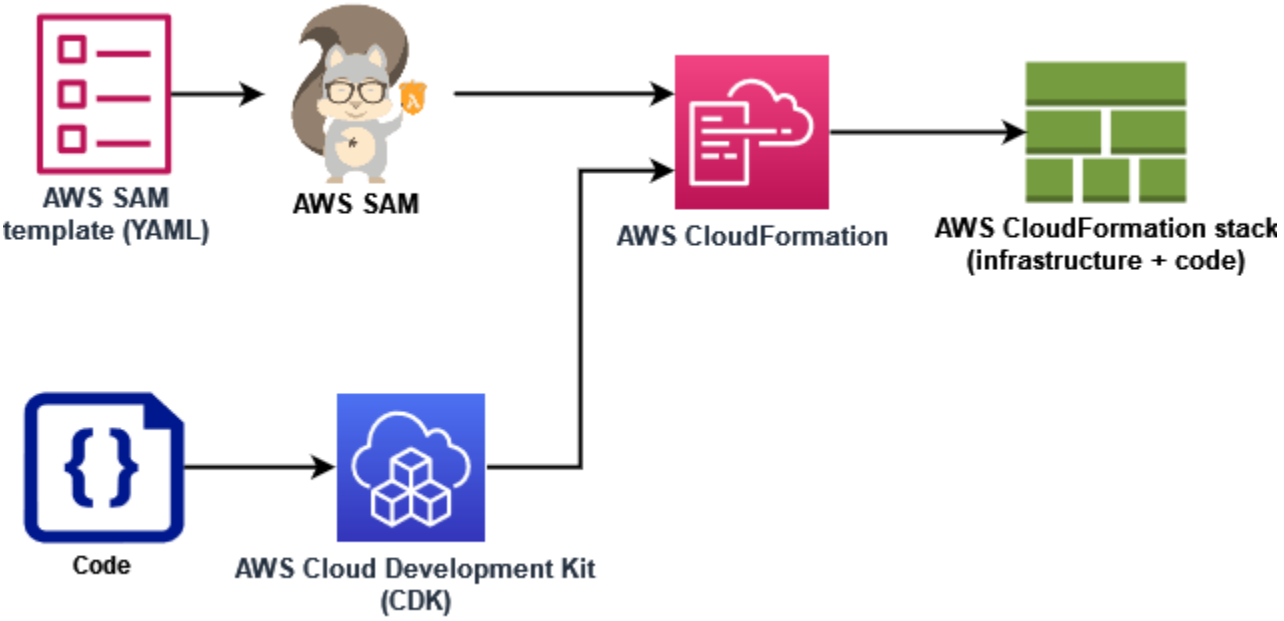
CloudFormation est le service IaC de base de AWS. Vous pouvez utiliser nos YAMLJSONmodèles pour modéliser et provisionner l'ensemble de votre AWS infrastructure, y compris les fonctions Lambda. CloudFormation gère les complexités liées à la création, à la mise à jour et à la suppression de vos AWS ressources.

AWS Serverless Application Model (AWS SAM)

AWS SAM est un framework open source construit sur. CloudFormation Il fournit une syntaxe simplifiée pour définir les applications sans serveur. Utilisez AWS SAM des modèles pour configurer rapidement les fonctions LambdaAPIs, les bases de données et les sources d'événements en quelques lignes seulement. YAML

AWS Cloud Development Kit (AWS CDK)

CDKIl s'agit d'une approche d'IaC axée sur le code. Vous pouvez définir votre architecture basée sur Lambda à l'aide de JavaScript Python TypeScript, Java, C#/.NET ou Go. Choisissez votre langage préféré et utilisez des éléments de programmation tels que les paramètres, les conditions, les boucles, la composition et l'héritage pour définir le résultat souhaité pour votre infrastructure. Génère CDK ensuite les CloudFormation modèles sous-jacents pour le déploiement. Pour un exemple d'utilisation de Lambda avecCDK, voir. Déploiement de fonctions Lambda avec le AWS CDK




AWS fournit également un service appelé AWS Infrastructure Composer à développer des modèles IaC à l'aide d'une interface graphique simple. Avec Infrastructure Composer, vous concevez une architecture d'application en faisant glisser, en regroupant et Services AWS en connectant dans un canevas visuel. Infrastructure Composer crée ensuite un AWS SAM modèle ou un AWS CloudFormation modèle à partir de votre conception que vous pouvez utiliser pour déployer votre application.

Dans la the section called “Démarrage avec l’IaC pour Lambda” section ci-dessous, vous utilisez Infrastructure Composer pour développer un modèle pour une application sans serveur basée sur une fonction Lambda existante.

## Démarrage avec l’IaC pour Lambda

Dans ce didacticiel, vous pouvez commencer à utiliser IaC avec Lambda en créant un AWS SAM modèle à partir d'une fonction Lambda existante, puis en développant une application sans serveur dans Infrastructure Composer en ajoutant d'autres ressources. AWS

Au cours de ce didacticiel, vous apprendrez certains concepts fondamentaux, tels que la manière dont les AWS ressources sont spécifiées dans AWS SAM. Vous apprendrez également à utiliser Infrastructure Composer pour créer une application sans serveur que vous pouvez déployer à l'aide de AWS SAM ou AWS CloudFormation.

Pour compléter ce didacticiel, effectuez les tâches suivantes :

- Créer un exemple de fonction Lambda
- Utilisez la console Lambda pour afficher le AWS SAM modèle de la fonction
- Exportez la configuration de votre fonction vers une application sans serveur simple AWS Infrastructure Composer et concevez une application sans serveur basée sur la configuration de votre fonction
- Enregistrez un AWS SAM modèle mis à jour que vous pouvez utiliser comme base pour déployer votre application sans serveur

## Prérequis

Dans ce didacticiel, vous utiliserez la fonction de synchronisation locale d'Infrastructure Composer pour enregistrer vos modèles et vos fichiers de code sur votre machine de génération locale. Pour utiliser cette fonctionnalité, vous devez disposer d'un navigateur prenant en charge l'accès au système de fichiersAPI, qui permet aux applications Web de lire, d'écrire et d'enregistrer des fichiers dans votre système de fichiers local. Nous vous recommandons d'utiliser Google Chrome ou Microsoft Edge. Pour plus d'informations sur l'accès au système de fichiersAPI, voir Qu'est-ce que l'accès au système de fichiers API ?

## Création d'une fonction Lambda

Dans cette première étape, vous allez créer une fonction Lambda que vous pouvez utiliser pour terminer le reste du didacticiel. Pour simplifier les choses, vous utilisez la console Lambda pour créer une fonction de base « Hello world » à l'aide du moteur d'exécution Python 3.11.

Pour créer une fonction Lambda « Hello world » à l'aide de la console

1. Ouvrez la console Lambda.
2. Sélectionnez Créer une fonction.
3. Ne désélectionnez pas l'option Créer à partir de zéro puis, sous Informations de base, saisissez **`LambdaIaCDemo`** pour le nom de la fonction.
4. Pour l'environnement d'exécution, sélectionnez Python 3.11.
5. Sélectionnez Create function (Créer une fonction).

# Afficher le AWS SAM modèle correspondant à votre fonction

Avant d'exporter la configuration de votre fonction vers Infrastructure Composer, utilisez la console Lambda pour afficher la configuration actuelle de votre fonction sous forme de modèle. AWS SAM En suivant les étapes décrites dans cette section, vous découvrirez l'anatomie d'un AWS SAM modèle et comment définir des ressources telles que les fonctions Lambda pour commencer à spécifier une application sans serveur.

Pour afficher le AWS SAM modèle de votre fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez la fonction que vous venez de créer (`LambdaIaCDemo`).
3. Dans le volet de Présentation de la fonction, choisissez Modèle.

   À la place du schéma représentant la configuration de votre fonction, vous verrez un AWS SAM modèle pour votre fonction. Le modèle doit ressembler à ce qui suit.

```
# This AWS SAM template has been generated from your function's
# configuration. If your function has one or more triggers, note
# that the AWS resources associated with these triggers aren't fully
# specified in this template and include placeholder values.Open this template
# in AWS Application Composer or your favorite IDE and modify
# it to specify a serverless application with other AWS resources.
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
```

```
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogGroup
            Resource: arn:aws:logs:us-east-1:123456789012:*
          - Effect: Allow
            Action:
              - logs:CreateLogStream
              - logs:PutLogEvents
            Resource:
              - >-
                arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
LambdaIaCDemo:*
```

Prenons un moment pour examiner le YAML modèle de votre fonction et comprendre certains concepts clés.

Le modèle commence par la déclaration `Transform: AWS::Serverless-2016-10-31`. Cette déclaration est obligatoire car, dans les coulisses, les AWS SAM modèles sont déployés via AWS CloudFormation. L'utilisation de l'instruction `Transform` identifie le modèle en tant que fichier modèle AWS SAM .

Après la déclaration `Transform` vient la section `Resources`. C'est ici que sont définies les AWS ressources que vous souhaitez déployer avec votre AWS SAM modèle. AWS SAM les modèles peuvent contenir une combinaison de AWS SAM ressources et de AWS CloudFormation ressources. Cela est dû au fait que pendant le déploiement, les AWS SAM AWS CloudFormation modèles se transforment en modèles, de sorte que toute AWS CloudFormation syntaxe valide peut être ajoutée à un AWS SAM modèle.

Pour le moment, une seule ressource est définie dans la section `Resources` du modèle, votre fonction Lambda `LambdaIaCDemo`. Pour ajouter une fonction Lambda à un AWS SAM modèle, vous devez utiliser le type de `AWS::Serverless::Function` ressource. La ressource `Properties` d'une fonction Lambda définit l'exécution de la fonction, le gestionnaire de fonctions et les autres

options de configuration. Le chemin d'accès au code source de votre fonction qui AWS SAM doit être utilisé pour déployer la fonction est également défini ici. Pour en savoir plus sur les ressources des fonctions Lambda dans AWS SAM, consultez AWS: :Serverless : :Function dans le manuel du développeur.AWS SAM

Outre les propriétés et les configurations de la fonction, le modèle spécifie également une politique AWS Identity and Access Management (IAM) pour votre fonction. Cette politique autorise votre fonction à écrire des journaux sur Amazon CloudWatch Logs. Lorsque vous créez une fonction dans la console Lambda, Lambda associe automatiquement cette politique à votre fonction. Pour en savoir plus sur la spécification d'une IAM politique pour une fonction dans un AWS SAM modèle, consultez la `policies` propriété sur la AWS page ::Serverless : :Function du manuel du AWS SAM développeur.

Pour en savoir plus sur la structure des AWS SAM modèles, consultez la section Anatomie des AWS SAM modèles.

## AWS Infrastructure Composer À utiliser pour concevoir une application sans serveur

Pour commencer à créer une application sans serveur simple en utilisant le AWS SAM modèle de votre fonction comme point de départ, vous exportez la configuration de votre fonction vers Infrastructure Composer et activez le mode de synchronisation local d'Infrastructure Composer. La synchronisation locale enregistre automatiquement le code de votre fonction et votre AWS SAM modèle sur votre machine de génération locale et synchronise votre modèle enregistré lorsque vous ajoutez d'autres AWS ressources dans Infrastructure Composer.

Pour exporter votre fonction vers Infrastructure Composer

1. Dans le volet Vue d’ensemble des fonctions, choisissez Exporter vers Application Composer.

   Pour exporter la configuration et le code de votre fonction vers Infrastructure Composer, Lambda crée un compartiment Amazon S3 dans votre compte pour stocker temporairement ces données.
2. Dans la boîte de dialogue, choisissez Confirmer et créez un projet pour accepter le nom par défaut de ce compartiment et exporter la configuration et le code de votre fonction vers Infrastructure Composer.
3. (Facultatif) Pour choisir un autre nom pour le compartiment Amazon S3 créé par Lambda, entrez un nouveau nom et choisissez Confirmer et créer un projet. Les noms de compartiment Amazon S3 doivent être uniques et respecter les règles de dénomination de compartiment.

Sélectionnez Confirmer et créer un projet pour ouvrir la console Infrastructure Composer. Sur le canevas, votre fonction Lambda apparaît.

4. Dans le menu déroulant, choisissez Activer la synchronisation locale.
5. Dans la boîte de dialogue qui apparaît, choisissez Sélectionner un dossier et un dossier sur votre machine de construction locale.
6. Choisissez Activer pour activer la synchronisation locale.

Pour exporter votre fonction vers Infrastructure Composer, vous devez être autorisé à utiliser certaines API actions. Si vous ne parvenez pas à exporter votre fonction, vérifiez the section called “Autorisations nécessaires” et assurez-vous que vous disposez des autorisations nécessaires.

> **Note**
>
> La tarification standard d'Amazon S3 s'applique au bucket créé par Lambda lorsque vous exportez une fonction vers Infrastructure Composer. Les objets que Lambda place dans le compartiment sont automatiquement supprimés au bout de 10 jours, mais Lambda ne supprime pas le compartiment lui-même.
> Pour éviter que des frais supplémentaires ne soient ajoutés à votre Compte AWS compte, suivez les instructions de la section Supprimer un bucket après avoir exporté votre fonction vers Infrastructure Composer. Pour en savoir plus sur le compartiment Amazon S3 créé par Lambda, consultez the section called “Compositeur d'infrastructure”.

Pour concevoir votre application sans serveur dans Infrastructure Composer

Après avoir activé la synchronisation locale, les modifications que vous apportez dans Infrastructure Composer seront reflétées dans le AWS SAM modèle enregistré sur votre machine de génération locale. Vous pouvez désormais glisser-déposer des AWS ressources supplémentaires sur le canevas Infrastructure Composer pour créer votre application. Dans cet exemple, vous ajoutez une file d'attente Amazon SQS simple comme déclencheur pour votre fonction Lambda et une table DynamoDB pour la fonction dans laquelle écrire des données.

1. Ajoutez un SQS déclencheur Amazon à votre fonction Lambda en procédant comme suit :

   a. Dans le champ de recherche de la palette Ressources, entrez **SQS**.

b. Faites glisser la ressource SQSQueue sur votre canevas et positionnez-la à gauche de votre fonction Lambda.

c. Choisissez Détails, puis, pour Logical ID, entrez **`LambdaIaCQueue`**.

d. Choisissez Save (Enregistrer).

e. Connectez vos ressources Amazon SQS et Lambda en cliquant sur le port d'abonnement sur la carte de SQS file d'attente et en le faisant glisser vers le port gauche de la carte de fonction Lambda. L'apparition d'une ligne entre les deux ressources indique que la connexion a abouti. Infrastructure Composer affiche également un message au bas du canevas indiquant que les deux ressources sont correctement connectées.

2. Ajoutez une table Amazon DynamoDB dans laquelle votre fonction Lambda pourra y écrire des données en procédant comme suit :

a. Dans le champ de recherche de la palette Ressources, entrez **`DynamoDB`**.

b. Faites glisser la ressource Table DynamoDB sur votre canevas et positionnez-la à droite de votre fonction Lambda.

c. Choisissez Détails, puis, pour Logical ID, entrez **`LambdaIaCTable`**.

d. Choisissez Save (Enregistrer).

e. Connectez la table DynamoDB à votre fonction Lambda en cliquant sur le port droit de la carte de fonction Lambda et en la faisant glisser vers le port gauche de la carte DynamoDB.

Maintenant que vous avez ajouté ces ressources supplémentaires, examinons le AWS SAM modèle mis à jour créé par Infrastructure Composer.

Pour afficher votre AWS SAM modèle mis à jour

- Sur le canevas Infrastructure Composer, choisissez Template pour passer de la vue canevas à la vue modèle.

Votre AWS SAM modèle doit désormais contenir les ressources et propriétés supplémentaires suivantes :

- Une SQS file d'attente Amazon avec l'identifiant `LambdaIaCQueue`

```
LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
```

```
      MessageRetentionPeriod: 345600
```

Lorsque vous ajoutez une SQS file d'attente Amazon à l'aide d'Infrastructure Composer, Infrastructure Composer définit la `MessageRetentionPeriod` propriété. Vous pouvez également définir la `FifoQueue` propriété en sélectionnant Détails sur la carte de SQS file d'attente et en cochant ou décochant la file d'attente Fifo.

Pour définir d'autres propriétés pour votre file d'attente, vous pouvez modifier manuellement le modèle et les y ajouter. Pour en savoir plus sur la `AWS::SQS::Queue` ressource et ses propriétés disponibles, voir AWS: SQS : :Queue dans le guide de l'AWS CloudFormation utilisateur.

- Une `Events` propriété dans la définition de votre fonction Lambda qui spécifie la SQS file d'attente Amazon comme déclencheur de la fonction

```
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1
```

La propriété `Events` est constituée d'un type d'événement et d'un ensemble de propriétés qui dépendent du type. Pour en savoir plus sur les différentes options Services AWS que vous pouvez configurer pour déclencher une fonction Lambda et sur les propriétés que vous pouvez définir, consultez le EventSourceguide du AWS SAM développeur.

- Table DynamoDB avec l'identifiant `LambdaIaCTable`

```
LambdaIaCTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      StreamSpecification:
        StreamViewType: NEW_AND_OLD_IMAGES
```

Lorsque vous ajoutez une table DynamoDB à l'aide d'Infrastructure Composer, vous pouvez définir les clés de votre table en choisissant Détails sur la fiche de table DynamoDB et en modifiant les valeurs clés. Infrastructure Composer définit également des valeurs par défaut pour un certain nombre d'autres propriétés, notamment `BillingMode` et`StreamViewType`.

Pour en savoir plus sur ces propriétés et les autres propriétés que vous pouvez ajouter à votre AWS SAM modèle, consultez AWS: :DynamoDB : :Table dans le guide de l'utilisateur.AWS CloudFormation

- Nouvelle IAM politique qui autorise votre fonction à effectuer des CRUD opérations sur la table DynamoDB que vous avez ajoutée.

```
Policies:
...
  - DynamoDBCrudPolicy:
    TableName: !Ref LambdaIaCTable
```

Le AWS SAM modèle final complet doit ressembler à ce qui suit.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
```

```
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        - Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
              Resource: arn:aws:logs:us-east-1:594035263019:*
            - Effect: Allow
              Action:
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource:
                - arn:aws:logs:us-east-1:594035263019:log-group:/aws/lambda/
LambdaIaCDemo:*
        - DynamoDBCrudPolicy:
            TableName: !Ref LambdaIaCTable
      Events:
        LambdaIaCQueue:
          Type: SQS
          Properties:
            Queue: !GetAtt LambdaIaCQueue.Arn
            BatchSize: 1
      Environment:
        Variables:
          LAMBDAIACTABLE_TABLE_NAME: !Ref LambdaIaCTable
          LAMBDAIACTABLE_TABLE_ARN: !GetAtt LambdaIaCTable.Arn
  LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 345600
  LambdaIaCTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      StreamSpecification:
```

```
        StreamViewType: NEW_AND_OLD_IMAGES
```

## Déployez votre application sans serveur à l'aide de AWS SAM (facultatif)

Si vous souhaitez déployer une application sans serveur AWS SAM à l'aide du modèle que vous venez de créer dans Infrastructure Composer, vous devez d'abord installer le AWS SAM CLI. Pour ce faire, suivez les instructions de la section Installation du AWS SAM CLI.

Avant de déployer votre application, vous devez également mettre à jour le code de fonction enregistré par Infrastructure Composer avec votre modèle. Pour le moment, le `lambda_function.py` fichier enregistré par Infrastructure Composer ne contient que le code de base « Hello world » fourni par Lambda lors de la création de la fonction.

Pour mettre à jour votre code de fonction, copiez le code suivant et collez-le dans le `lambda_function.py` fichier Infrastructure Composer enregistré sur votre machine de construction locale. Vous avez spécifié le répertoire dans lequel Infrastructure Composer doit enregistrer ce fichier lorsque vous avez activé le mode de synchronisation locale.

Ce code accepte une paire clé-valeur dans un message provenant de la SQS file d'attente Amazon que vous avez créée dans Infrastructure Composer. Si la clé et la valeur sont des chaînes, le code les utilise ensuite pour écrire un élément dans la table DynamoDB définie dans votre modèle.

Mise à jour du code de la fonction Python

```
import boto3
import os
import json

# define the DynamoDB table that Lambda will connect to
tablename = os.environ['LAMBDAIACTABLE_TABLE_NAME']

# create the DynamoDB resource
dynamo = boto3.client('dynamodb')

def lambda_handler(event, context):
    # get the message out of the SQS event
    message = event['Records'][0]['body']
    data = json.loads(message)
    # write event data to DDB table
    if check_message_format(data):
        key = next(iter(data))
```

```
        value = data[key]
        dynamo.put_item(
            TableName=tablename,
            Item={
                'id': {'S': key},
                'Value': {'S': value}
            }
        )
    else:
        raise ValueError("Input data not in the correct format")

# check that the event object contains a single key value
# pair that can be written to the database
def check_message_format(message):
    if len(message) != 1:
        return False

    key, value = next(iter(message.items()))

    if not (isinstance(key, str) and isinstance(value, str)):
        return False

    else:
        return True
```

Pour déployer votre application sans serveur

Pour déployer votre application à l'aide du AWS SAM CLI, effectuez les étapes suivantes. Pour que votre fonction soit correctement construite et déployée, la version 3.11 de Python doit être installée sur votre machine de compilation et sur votre PATH.

1. Exécutez la commande suivante depuis le répertoire dans lequel Infrastructure Composer a enregistré vos `lambda_function.py` fichiers `template.yaml` et.

   ```
   sam build
   ```

   Cette commande rassemble les artefacts de compilation pour votre application et les place dans le format et l'emplacement appropriés pour les déployer.
2. Pour déployer votre application et créer les ressources LambdaSQS, Amazon et DynamoDB spécifiées dans votre AWS SAM modèle, exécutez la commande suivante.

```
sam deploy --guided
```

L'utilisation du `--guided` drapeau signifie que des instructions vous AWS SAM seront affichées pour vous guider tout au long du processus de déploiement. Pour ce déploiement, acceptez les options par défaut en appuyant sur Entrée.

Au cours du processus de déploiement, AWS SAM crée les ressources suivantes dans votre Compte AWS :

- Une AWS CloudFormation pile nommée `sam-app`
- Une fonction Lambda avec le format de nom `sam-app-LambdaIaCDemo-`*`99VXPpYQVv1M`*
- Une SQS file d'attente Amazon au format de nom `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*
- Une table DynamoDB au format de nom `sam-app-LambdaIaCTable-`*`CN0S66C0VLNV`*

AWS SAM crée également les IAM rôles et les politiques nécessaires pour que votre fonction Lambda puisse lire les messages de la SQS file d'attente Amazon et effectuer des CRUD opérations sur la table DynamoDB.

## Testing de votre application déployée (facultatif)

Pour vérifier que votre application sans serveur s'est correctement déployée, envoyez un message à votre SQS file d'attente Amazon contenant une paire clé-valeur et vérifiez que Lambda écrit un élément dans votre table DynamoDB en utilisant ces valeurs.

Tester votre application sans serveur

1. Ouvrez la page Queues de la SQS console Amazon et sélectionnez la file d'attente AWS SAM créée à partir de votre modèle. Le nom est au format `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*.
2. Choisissez Envoyer et recevoir des messages et collez le texte suivant JSON dans le corps du message de la section Envoyer un message.

```
{
    "myKey": "myValue"
}
```

3. Choisissez Send Message (Envoyer un message).

   L'envoi de votre message à la file d'attente amène Lambda à invoquer votre fonction par le biais du mappage de la source d'événement défini dans votre modèle AWS SAM . Pour confirmer que Lambda a invoqué votre fonction comme prévu, indiquez qu'un élément a été ajouté à votre table DynamoDB.
4. Ouvrez la page Tables de la console DynamoDB et choisissez votre table. Le nom est au format `sam-app-LambdaIaCTable-CN0S66C0VLNV`.
5. Sélectionnez Explore table items (Explorer les éléments de la table). Dans le volet Items returned (Éléments retournés), vous devriez voir un élément avec l'id `myKey` et la Valeur `myValue`.

# Déploiement de fonctions Lambda avec le AWS CDK

AWS Cloud Development Kit (AWS CDK) Il s'agit d'un framework d'infrastructure en tant que code (IaC) que vous pouvez utiliser pour définir une infrastructure AWS cloud en utilisant le langage de programmation de votre choix. Pour définir votre propre infrastructure cloud, vous devez d'abord écrire une application (dans CDK l'une des langues prises en charge) contenant une ou plusieurs piles. Ensuite, vous le synthétisez dans un AWS CloudFormation modèle et déployez vos ressources sur votre Compte AWS. Suivez les étapes décrites dans cette rubrique pour déployer une fonction Lambda qui renvoie un événement depuis un point de terminaison Amazon API Gateway.

La bibliothèque AWS Construct, incluse dans leCDK, fournit des modules que vous pouvez utiliser pour modéliser les ressources qui Services AWS fournissent. Pour les services populaires, la bibliothèque fournit des constructions organisées avec des valeurs par défaut intelligentes et les bonnes pratiques. Vous pouvez utiliser le module aws_lambda pour définir votre fonction et les ressources de support en quelques lignes de code.

## Prérequis

Avant de commencer ce didacticiel, installez le AWS CDK en exécutant la commande suivante.

```
npm install -g aws-cdk
```

## Étape 1 : Configurez votre AWS CDK projet

Créez un répertoire pour votre nouvelle AWS CDK application et initialisez le projet.

JavaScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language javascript
```

TypeScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language typescript
```

Python

```
mkdir hello-lambda
cd hello-lambda
cdk init --language python
```

Une fois le projet démarré, activez l'environnement virtuel du projet et installez les dépendances de base pour AWS CDK.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-lambda
cd hello-lambda
cdk init --language java
```

Importez ce projet Maven dans votre environnement de développement intégré Java (IDE). Par exemple, dans Eclipse, choisissez Fichier, Import, Maven, Existing Maven Projects.

C#

```
mkdir hello-lambda
cd hello-lambda
cdk init --language csharp
```

> **Note**
>
> Le modèle AWS CDK d'application utilise le nom du répertoire du projet pour générer des noms pour les classes et les fichiers sources. Dans cet exemple, le répertoire est nommé `hello-lambda`. Si vous utilisez un nom répertoire différent, votre application ne correspondra pas à ces instructions.

AWS CDK v2 inclut des constructions stables pour tous Services AWS dans un seul package appelé`aws-cdk-lib`. Ce package est installé en tant que dépendance lorsque vous initialisez le projet. Lorsque vous travaillez avec certains langages de programmation, le package est installé lorsque vous créez le projet pour la première fois.

## Étape 2 : définir la AWS CDK pile

Une CDK pile est un ensemble d'une ou plusieurs constructions qui définissent les AWS ressources. Chaque CDK pile représente une AWS CloudFormation pile dans votre CDK application.

Pour définir votreCDK, Stack, suivez les instructions de votre langage de programmation préféré. Cette pile définit les éléments suivants :

- Nom logique de la fonction : `MyFunction`
- Emplacement du code de fonction, spécifié dans la `code` propriété. Pour plus d'informations, consultez la section Code du gestionnaire dans la AWS Cloud Development Kit (AWS CDK) APIréférence.
- Le nom logique REST API est le suivant : `HelloApi`
- Nom logique du point de terminaison API Gateway : `ApiGwEndpoint`

Notez que toutes les CDK piles de ce didacticiel utilisent le runtime Node.js pour la fonction Lambda. Vous pouvez utiliser différents langages de programmation pour la CDK pile et la fonction Lambda afin de tirer parti des points forts de chaque langage. Par exemple, vous pouvez utiliser TypeScript for the CDK stack pour tirer parti des avantages du typage statique pour votre code d'infrastructure. Vous pouvez utiliser la fonction Lambda JavaScript pour tirer parti de la flexibilité et du développement rapide d'un langage typé dynamiquement.

JavaScript

Ouvrez le fichier `lib/hello-lambda-stack.js` et remplacez le contenu par ce qui suit.

```
const { Stack } = require('aws-cdk-lib');
const lambda = require('aws-cdk-lib/aws-lambda');
const apigw = require('aws-cdk-lib/aws-apigateway');

class HelloLambdaStack extends Stack {
  /**
   *
   * @param {Construct} scope
   * @param {string} id
   * @param {StackProps=} props
   */
  constructor(scope, id, props) {
    super(scope, id, props);
    const fn = new lambda.Function(this, 'MyFunction', {
      code: lambda.Code.asset('lib/lambda-handler'),
      runtime: lambda.Runtime.NODEJS_LATEST,
      handler: 'index.handler'
    });

    const endpoint = new apigw.LambdaRestApi(this, 'MyEndpoint', {
      handler: fn,
      restApiName: "HelloApi"
    });

  }
}

module.exports = { HelloLambdaStack }
```

TypeScript

Ouvrez le fichier `lib/hello-lambda-stack.ts` et remplacez le contenu par ce qui suit.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as apigw from "aws-cdk-lib/aws-apigateway";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as path from 'node:path';

export class HelloLambdaStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps){
    super(scope, id, props)
    const fn = new lambda.Function(this, 'MyFunction', {
```

```
      runtime: lambda.Runtime.NODEJS_LATEST,
      handler: 'index.handler',
      code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
    });

    const endpoint = new apigw.LambdaRestApi(this, `ApiGwEndpoint`, {
      handler: fn,
      restApiName: `HelloApi`,
    });

  }
}
```

Python

Ouvrez le fichier `/hello-lambda/hello_lambda/hello_lambda_stack.py` et remplacez le contenu par ce qui suit.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigw,
    aws_lambda as _lambda
)
from constructs import Construct

class HelloLambdaStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        fn = _lambda.Function(
            self,
            "MyFunction",
            runtime=_lambda.Runtime.NODEJS_LATEST,
            handler="index.handler",
            code=_lambda.Code.from_asset("lib/lambda-handler")
        )

        endpoint = apigw.LambdaRestApi(
            self,
            "ApiGwEndpoint",
            handler=fn,
            rest_api_name="HelloApi"
```

```
        )
```

Java

Ouvrez le fichier `/hello-lambda/src/main/java/com/myorg/HelloLambdaStack.java` et remplacez le contenu par ce qui suit.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.apigateway.LambdaRestApi;
import software.amazon.awscdk.services.lambda.Function;

public class HelloLambdaStack extends Stack {
    public HelloLambdaStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloLambdaStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Function hello = Function.Builder.create(this, "MyFunction")

 .runtime(software.amazon.awscdk.services.lambda.Runtime.NODEJS_LATEST)

 .code(software.amazon.awscdk.services.lambda.Code.fromAsset("lib/lambda-handler"))
                            .handler("index.handler")
                            .build();

        LambdaRestApi api = LambdaRestApi.Builder.create(this, "ApiGwEndpoint")
                                .restApiName("HelloApi")
                                .handler(hello)
                                .build();

    }
}
```

C#

Ouvrez le fichier `src/HelloLambda/HelloLambdaStack.cs` et remplacez le contenu par ce qui suit.

```
using Amazon.CDK;
using Amazon.CDK.AWS.APIGateway;
using Amazon.CDK.AWS.Lambda;
using Constructs;

namespace HelloLambda
{
    public class HelloLambdaStack : Stack
    {
        internal HelloLambdaStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var fn = new Function(this, "MyFunction", new FunctionProps
            {
                Runtime = Runtime.NODEJS_LATEST,
                Code = Code.FromAsset("lib/lambda-handler"),
                Handler = "index.handler"
            });

            var api = new LambdaRestApi(this, "ApiGwEndpoint", new
 LambdaRestApiProps
            {
                Handler = fn
            });
        }
    }
}
```

## Étape 3 : Création du code de fonction Lambda

1. À partir de la racine de votre projet (`hello-lambda`), créez le `/lib/lambda-handler` répertoire pour le code de la fonction Lambda. Ce répertoire est spécifié dans la `code` propriété de votre AWS CDK pile.
2. Créez un nouveau fichier appelé `index.js` dans le `/lib/lambda-handler` répertoire. Collez le code suivant dans le fichier. La fonction extrait des propriétés spécifiques de la API demande et les renvoie sous forme de JSON réponse.

```
exports.handler = async (event) => {
  // Extract specific properties from the event object
  const { resource, path, httpMethod, headers, queryStringParameters, body } =
 event;
  const response = {
    resource,
    path,
    httpMethod,
    headers,
    queryStringParameters,
    body,
  };
  return {
    body: JSON.stringify(response, null, 2),
    statusCode: 200,
  };
};
```

## Étape 4 : Déployer la AWS CDK pile

1. À la racine de votre projet, exécutez la commande cdk synth :

```
cdk synth
```

Cette commande synthétise un AWS CloudFormation modèle à partir de votre CDK pile. Le modèle est un YAML fichier d'environ 400 lignes, similaire au suivant.

> **Note**
>
> Si le message d'erreur suivant s'affiche, assurez-vous que vous êtes bien à la racine du répertoire de votre projet.
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

Example AWS CloudFormation modèle

```
Resources:
```

```
  MyFunctionServiceRole3C357FF2:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Metadata:
      aws:cdk:path: HelloLambdaStack/MyFunction/ServiceRole/Resource
  MyFunction1BAA52E7:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        S3Bucket:
          Fn::Sub: cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}
        S3Key:
ab1111111cd32708dc4b83e81a21c296d607ff2cdef00f1d7f48338782f92l3901.zip
      Handler: index.handler
      Role:
        Fn::GetAtt:
          - MyFunctionServiceRole3C357FF2
          - Arn
      Runtime: nodejs20.x
      ...
```

2. Exécutez la commande cdk deploy :

```
cdk deploy
```

Patientez pendant que vos ressources sont créées. Le résultat final inclut le point URL de terminaison de votre API passerelle. Exemple :

```
Outputs:
```

```
HelloLambdaStack.ApiGwEndpoint77F417B1 = https://abcd1234.execute-api.us-
east-1.amazonaws.com/prod/
```

## Étape 5 : tester la fonction

Pour appeler la fonction Lambda, copiez le point de terminaison API Gateway et collez-le dans un navigateur Web ou exécutez une `curl` commande :

```
curl -s https://abcd1234.execute-api.us-east-1.amazonaws.com/prod/
```

La réponse est une JSON représentation des propriétés sélectionnées à partir de l'objet d'événement d'origine, qui contient des informations sur la demande envoyée au point de terminaison de la API passerelle. Exemple :

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "GET",
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding": "gzip, deflate, br, zstd",
    "Accept-Language": "en-US,en;q=0.9",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-ASN": "16509",
    "CloudFront-Viewer-Country": "US",
    "Host": "abcd1234.execute-api.us-east-1.amazonaws.com",
     ...
```

## Étape 6 : Nettoyer vos ressources

Le point de terminaison API Gateway est accessible au public. Pour éviter des frais inattendus, exécutez la commande cdk destroy pour supprimer la pile et toutes les ressources associées.

```
cdk destroy
```

# Étapes suivantes

Pour plus d'informations sur l'écriture d' AWS CDK applications dans la langue de votre choix, consultez les rubriques suivantes :

TypeScript

- Travailler avec le AWS CDK in TypeScript

JavaScript

- Travailler avec le AWS CDK in JavaScript

Python

- Travailler avec le AWS CDK en Python

Java

- Travailler avec le AWS CDK en Java

C#

- Travailler avec le AWS CDK en C#

Go

- Travailler avec AWS CDK In Go

# Comprendre le modèle de programmation Lambda

Lambda fournit un modèle de programmation commun à tous les runtimes. Le modèle de programmation définit l'interface entre votre code et le système Lambda. Vous indiquez à Lambda le point d'entrée de votre fonction en définissant un gestionnaire dans la configuration de la fonction. Le runtime transmet les objets au gestionnaire qui contiennent l'événement d'appel et le contexte, tels que le nom de la fonction et l'ID de la demande.

Lorsque le gestionnaire termine le traitement du premier événement, le runtime lui en envoie un autre. La classe de la fonction reste en mémoire, de sorte que les clients et variables déclarés en dehors de la méthode du gestionnaire dans le code d'initialisation peuvent être réutilisés. Pour réduire le temps de traitement lors des événements suivants, créez des ressources réutilisables telles que des AWS SDK clients lors de l'initialisation. Une fois initialisée, chaque instance de votre fonction peut traiter des milliers de demandes.

Votre fonction a également accès au stockage local dans le répertoire `/tmp`. Le contenu du répertoire est conservé lorsque l'environnement d'exécution est gelé, fournissant ainsi un cache temporaire qui peut servir à plusieurs invocations. Pour plus d'informations, consultez Environnement d'exécution Lambda.

Lorsque le suivi AWS X-Ray est activé, le runtime enregistre des sous-segments distincts pour l'initialisation et l'exécution.

Le moteur d'exécution capture les résultats de journalisation de votre fonction et les envoie à Amazon CloudWatch Logs. En plus de consigner la sortie de votre fonction, le runtime consigne également les entrées lorsque la fonction d'appel démarre et se termine. Cela inclut un journal de rapport avec l'ID de demande, la durée facturée, la durée d'initialisation et d'autres détails. Si votre fonction génère une erreur, le runtime renvoie cette erreur au mécanisme d'appel.

> **Note**
>
> La journalisation est soumise à des quotas de CloudWatch journalisation. Les données des journaux peuvent être perdues en raison de la limitation ou, dans certains cas, lorsqu'une instance de votre fonction est arrêtée.

Lambda met à l'échelle votre fonction en exécutant des instances supplémentaires à mesure que la demande augmente et en arrêtant des instances à mesure que la demande diminue. Ce modèle entraîne des variations dans l'architecture des applications, telles que :

- Sauf indication contraire, les demandes entrantes peuvent être traitées dans le désordre ou simultanément.
- Ne comptez pas sur la longévité des instances de votre fonction, stockez plutôt l'état de votre application ailleurs.
- Utilisez le stockage local et les objets de niveau classe afin d'améliorer les performances, mais réduisez au minimum la taille de votre package de déploiement et la quantité de données que vous transférez vers l'environnement d'exécution.

Pour une introduction pratique au modèle de programmation dans le langage de programmation de votre choix, consultez les sections suivantes.

- Création de fonctions Lambda avec Node.js
- Création de fonctions Lambda avec Python
- Création de fonctions Lambda avec Ruby
- Création de fonctions Lambda avec Java
- Création de fonctions Lambda avec Go
- Création de fonctions Lambda avec C#
- Création de fonctions Lambda avec PowerShell

# Environnements d’exécution (runtimes) Lambda

Lambda prend en charge plusieurs langages via l’utilisation d’environnements d’exécution. Une exécution fournit un environnement spécifique au langage qui relaie les événements d’invocation, les informations de contexte et les réponses entre Lambda et la fonction. Vous pouvez utiliser les runtimes que Lambda fournit, ou créer vos propres runtimes.

Chaque version majeure du langage de programmation possède un environnement d’exécution distinct, avec un identifiant de l’environnement d’exécution unique, tel que `nodejs20.x` ou `python3.12`. Pour configurer une fonction afin d’utiliser une nouvelle version majeure du langage, vous devez modifier l’identifiant d’exécution. Comme il AWS Lambda n'est pas possible de garantir la rétrocompatibilité entre les versions principales, il s'agit d'une opération pilotée par le client.

Pour une fonction définie en tant qu’image de conteneur, vous choisissez une exécution et la distribution Linux lorsque vous créez l’image de conteneur. Pour modifier l’environnement d’exécution, vous créez une image de conteneur.

Lorsque vous utilisez une archive de fichiers .zip pour le package de déploiement, vous choisissez un environnement d’exécution lorsque vous créez la fonction. Pour modifier l’environnement d’exécution, vous pouvez mettre à jour la configuration de votre fonction. L’environnement d’exécution est associé à l’une des distributions Amazon Linux. L’environnement d’exécution sous-jacent fournit des bibliothèques et des variables d’environnement supplémentaires auxquelles vous pouvez accéder depuis le code de votre fonction.

Lambda invoque votre fonction dans un environnement d’exécution. L’environnement d’exécution fournit un environnement d’environnement d’exécution sécurisé et isolé qui gère les ressources nécessaires à l’exécution de votre fonction. Lambda réutilise l’environnement d’exécution à partir d’une invocation antérieure dans le cas où il y en a un disponible, ou il peut en créer un nouveau.

Pour utiliser d'autres langages dans Lambda, tels que Go ou Rust, utilisez un environnement d'exécution uniquement pour le système d'exploitation. L’environnement d’exécution de Lambda fournit une interface d’environnement d’exécution pour obtenir des événements d’invocations et envoyer des réponses. Vous pouvez déployer les autres langages en implémentant un environnement d’exécution en association avec le code de votre fonction, ou dans une couche.

# Environnements d'exécution pris en charge

Le tableau suivant répertorie les durées d'exécution Lambda prises en charge et les dates d'obsolescence prévues. Une fois qu'un environnement d'exécution est obsolète, vous pouvez toujours créer et mettre à jour des fonctions pendant une période limitée. Pour de plus amples informations, veuillez consulter the section called “Utilisation à l'exécution après la dépréciation”. Le tableau fournit les dates actuellement prévues pour la dépréciation de l'exécution. Ces dates sont fournies à des fins de planification et sont susceptibles d'être modifiées.

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Node.js 20 | `nodejs20.x` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Node.js 18 | `nodejs18.x` | Amazon Linux 2 | 31 juillet 2025 | 1 sept. 2025 | 1 octobre 2025 |
| Python 3.12 | `python3.12` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.11 | `python3.11` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.10 | `python3.10` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.9 | `python3.9` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.8 | `python3.8` | Amazon Linux 2 | 14 octobre 2024 | 28 février 2025 | 31 mars 2025 |
| Java 21 | `java21` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Java 17 | `java17` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Java 11 | `java11` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Java 8 | `java8.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| . NET8 | `dotnet8` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| . NET6 | `dotnet6` | Amazon Linux 2 | 20 déc. 2024 | 28 février 2025 | 31 mars 2025 |
| Ruby 3.3 | `ruby3.3` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Ruby 3.2 | `ruby3.2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Exécution réservée au système d'exploitation | `provided.al2023` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Exécution réservée au système d'exploitation | `provided.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

> **Note**
>
> Pour les nouvelles régions, Lambda ne prendra pas en charge les environnements d'exécution qui devraient devenir obsolètes dans les six prochains mois.

Lambda tient à jour les environnements d'exécution gérés et de leurs images de conteneur correspondantes à jour grâce à des correctifs et à la prise en charge des versions mineures. Pour plus d'informations, consultez Mises à jour de l'exécution Lambda.

Lambda continue de prendre en charge le langage de programmation Go après l'obsolescence de l'environnement d'exécution Go 1.x. Pour plus d'informations, consultez la section Migration des AWS Lambda fonctions de l'environnement d'exécution Go1.x vers le runtime personnalisé sur Amazon Linux 2 sur le AWS blog Compute.

Tous les environnements d'exécution Lambda pris en charge prennent en charge les architectures x86_64 et arm64.

## Nouvelles versions d'exécution

Lambda fournit des environnements d'exécution gérés pour les nouvelles versions linguistiques uniquement lorsque la version atteint la phase de support à long terme (LTS) du cycle de publication du langage. Par exemple, pour le cycle de publication de Node.js, lorsque la version atteint la LTS phase active.

Avant que la version n'atteigne la phase de support à long terme, elle reste en développement et peut encore faire l'objet de modifications majeures. Lambda applique automatiquement les mises à jour d'environnements d'exécution par défaut, de sorte que l'interruption des modifications apportées à une version d'environnement d'exécution peut empêcher vos fonctions de fonctionner comme prévu.

Lambda ne fournit pas d'environnements d'exécution gérés pour les versions linguistiques dont la publication n'est pas prévue. LTS

La liste suivante montre le mois de lancement cible pour les environnements d'exécution Lambda à venir. Ces dates ne sont données qu'à titre indicatif et sont susceptibles d'être modifiées.

- Python 3.13 - novembre 2024
- Node.js 22 - novembre 2024

## politique d'obsolescence de l'exécution

Environnements d'exécution (runtimes) Lambda pour les archives de fichiers .zip combinent un système d'exploitation, un langage de programmation et des bibliothèques de logiciels qui font l'objet d'une maintenance et de mises à jour de sécurité. La politique d'obsolescence standard de Lambda consiste à déprécier un environnement d'exécution lorsqu'un composant majeur de l'environnement

d'exécution atteint la fin du support à long terme de la communauté (LTS) et que les mises à jour de sécurité ne sont plus disponibles. Le plus souvent, il s'agit de l'environnement d'exécution du langage, bien que dans certains cas, un environnement d'exécution puisse être déconseillé car le système d'exploitation (OS) arrive à la fin de. LTS

Une fois qu'un environnement d'exécution est obsolète, il ne AWS peut plus y appliquer de correctifs ou de mises à jour de sécurité, et les fonctions utilisant cet environnement d'exécution ne sont plus éligibles au support technique. Ces environnements d'exécution obsolètes sont fournis « tels quels », sans aucune garantie, et peuvent contenir des bogues, des erreurs, des défauts ou d'autres vulnérabilités.

Pour en savoir plus sur la gestion des mises à niveau d'exécution et de la dépréciation, consultez les sections suivantes et la gestion des mises à niveau AWS Lambda d'exécution sur le blog AWS Compute.

> **Important**
>
> Lambda retarde parfois l'obsolescence d'un environnement d'exécution Lambda pendant une période limitée au-delà de la date de fin du support de la version du langage prise en charge par l'environnement d'exécution. Pendant cette période, Lambda applique uniquement des correctifs de sécurité au système d'exploitation de l'environnement d'exécution. Lambda n'applique pas de correctifs de sécurité aux environnements d'exécution des langages de programmation une fois leur date de fin de support atteinte.

## Modèle de responsabilité partagée

Lambda est responsable de l'organisation et de la publication des mises à jour de sécurité pour tous les environnements d'exécution gérés et les images de base de conteneurs pris en charge. Par défaut, Lambda appliquera ces mises à jour automatiquement aux fonctions utilisant des environnements d'exécution gérés. Lorsque le paramètre de mise à jour automatique de l'exécution par défaut a été modifié, consultez le modèle de responsabilité partagée des contrôles de gestion de l'exécution. Pour les fonctions déployées à l'aide d'images de conteneur, vous êtes chargé de reconstruire l'image de conteneur de votre fonction à partir de la dernière image de base et de redéployer l'image de conteneur.

Lorsqu'un environnement d'exécution est obsolète, la responsabilité de Lambda en matière de mise à jour de l'environnement d'exécution géré et des images de base du conteneur cesse. Vous êtes

responsable de la mise à niveau de vos fonctions afin d'utiliser un environnement d'exécution ou une image de base compatible.

Dans tous les cas, vous êtes responsable de l'application des mises à jour de votre code de fonction, y compris de ses dépendances. Vos responsabilités dans le cadre du modèle de responsabilité partagée sont résumées dans le tableau suivant.

| Phase du cycle de vie d'exécution | Responsabilités de Lambda | Vos responsabilités |
| --- | --- | --- |
| Runtime géré pris en charge | Fournissez des mises à jour d'exécution régulières avec des correctifs de sécurité et d'autres mises à jour.<br><br>Appliquez les mises à jour d'exécution automatiquement par défaut (voir the section called “Modes de mise à jour pendant” pour les comportements autres que ceux par défaut). | Mettez à jour le code de votre fonction, y compris les dépendances, pour corriger les éventuelles failles de sécurité. |
| Image de conteneur prise en charge | Fournissez des mises à jour régulières de l'image de base du conteneur avec des correctifs de sécurité et d'autres mises à jour. | Mettez à jour le code de votre fonction, y compris les dépendances, pour corriger les éventuelles failles de sécurité.<br><br>Reconstruisez et redéployez régulièrement votre image de conteneur à l'aide de la dernière image de base. |
| Runtime géré proche de la dépréciation | Avertissez les clients avant la dépréciation de l'exécution via la documentation AWS Health Dashboard, le courrier | Surveillez la documentation Lambda AWS Health Dashboard, les e-mails ou les informations relatives à la |

| Phase du cycle de vie d'exécution | Responsabilités de Lambda | Vos responsabilités |
| --- | --- | --- |
| | électronique et. Trusted Advisor<br><br>La responsabilité des mises à jour d'exécution prend fin en cas de dépréciation. | Trusted Advisor dépréciation de l'exécution.<br><br>Mettez à niveau les fonctions vers un environnement d'exécution compatible avant que le précédent ne soit déconseillé. |
| L'image du conteneur approche de la dépréciation | Les notifications d'obsoles cence ne sont pas disponibles pour les fonctions utilisant des images de conteneur.<br><br>La responsabilité des mises à jour des images de base des conteneurs prend fin à la dépréciation. | Tenez compte des programme s de dépréciation et des fonctions de mise à niveau vers une image de base prise en charge avant que l'image précédente ne soit déconseil lée. |

## Utilisation à l'exécution après la dépréciation

Une fois qu'un environnement d'exécution est obsolète, il ne AWS peut plus y appliquer de correctifs ou de mises à jour de sécurité, et les fonctions utilisant cet environnement d'exécution ne sont plus éligibles au support technique. Ces environnements d'exécution obsolètes sont fournis « tels quels », sans aucune garantie, et peuvent contenir des bogues, des erreurs, des défauts ou d'autres vulnérabilités. Les fonctions qui utilisent un environnement d'exécution obsolète peuvent également connaître une dégradation des performances ou d'autres problèmes, tels que l'expiration d'un certificat, susceptibles de les empêcher de fonctionner correctement.

Pendant au moins 30 jours après l’obsolescence d'un environnement d'exécution, vous pouvez toujours créer de nouvelles fonctions Lambda à l'aide de cet environnement d'exécution. À partir de 30 jours après l’obsolescence, Lambda commence à bloquer la création de nouvelles fonctions.

Pendant au moins 60 jours après la désapprobation d'un environnement d'exécution, vous pouvez toujours mettre à jour le code de fonction et la configuration des fonctions existantes. À partir de 60

jours après la dépréciation, Lambda commence à bloquer la mise à jour du code de fonction et de la configuration des fonctions existantes.

> **Note**
>
> Pour certains environnements d'exécution, AWS cela retarde les block-function-update dates block-function-create et au-delà des 30 et 60 jours habituels après la dépréciation. AWS a apporté cette modification en réponse aux commentaires des clients afin de vous donner plus de temps pour améliorer vos fonctionnalités. Reportez-vous aux tableaux figurant dans the section called “Environnements d'exécution pris en charge” et the section called “Environnements d’exécution obsolètes” pour connaître les dates de votre exécution.

Vous pouvez mettre à jour une fonction pour utiliser indéfiniment un environnement d'exécution compatible plus récent une fois qu'un environnement d'exécution est devenu obsolète. Vous devez vérifier que votre fonction fonctionne avec le nouveau moteur d'exécution avant d'appliquer le changement d'exécution dans les environnements de production, car vous ne pourrez pas revenir à l'environnement d'exécution obsolète une fois la période de 60 jours écoulée. Nous vous recommandons d'utiliser des versions de fonctions et des alias pour permettre un déploiement sécurisé avec restauration.

Notez que la durée exacte pendant laquelle vous pouvez continuer à créer et à mettre à jour des fonctions n'est pas fixée. Cette période peut varier en fonction de chaque dépréciation et selon les besoins. Régions AWS Les dates nominales pour le blocage des créations et des mises à jour de fonctions sont fournies dans le tableau des environnements d’exécution pris en charge dans la première section de cette page. Lambda ne commencera pas à bloquer les créations ou les mises à jour de fonctions avant les dates indiquées dans ce tableau.

Vous pouvez continuer à invoquer vos fonctions indéfiniment une fois que l’environnement d’exécution est devenu obsolète. Toutefois, il AWS est vivement recommandé de migrer les fonctions vers un environnement d'exécution compatible afin que celles-ci continuent de recevoir des correctifs de sécurité et restent éligibles au support technique.

## Réception de notifications d'obsolescence lors de l'exécution

Lorsqu'un environnement d'exécution approche de sa date d'obsolescence, Lambda vous envoie une alerte par e-mail si l'une de vos Compte AWS fonctions utilise cet environnement d'exécution. Les notifications sont également affichées dans AWS Health Dashboard et dans AWS Trusted Advisor.

- Réception de notifications par e-mail :

  Lambda vous envoie une alerte par e-mail au moins 180 jours avant qu'un environnement d'exécution ne soit obsolète. Cet e-mail répertorie les LATEST versions $ de toutes les fonctions utilisant le moteur d'exécution. Pour consulter la liste complète des versions de fonctions concernées, utilisez Trusted Advisor ou consultezthe section called “Obtenir des données sur les fonctions par exécution”.

  Lambda envoie une notification par e-mail au contact principal Compte AWS de votre compte. Pour plus d'informations sur l'affichage ou la mise à jour des adresses e-mail de votre compte, consultez la section Mise à jour des informations de contact dans la Référence générale AWS .

- Recevoir des notifications par le biais de AWS Health Dashboard :

  AWS Health Dashboard Affiche une notification au moins 180 jours avant qu'un environnement d'exécution ne soit obsolète. Les notifications apparaissent sur la page État de votre compte sous Autres notifications. L'onglet Ressources affectées de la notification répertorie les LATEST versions $ de toutes les fonctions utilisant le runtime.

  > **Note**
  >
  > Pour voir l'intégralité et la up-to-date liste des versions des fonctions concernées, utilisez Trusted Advisor ou consultezthe section called “Obtenir des données sur les fonctions par exécution”.

  AWS Health Dashboard les notifications expirent 90 jours après que le runtime concerné soit devenu obsolète.

- En utilisant AWS Trusted Advisor

  Trusted Advisor affiche une notification 180 jours avant qu'un environnement d'exécution ne soit obsolète. Les notifications apparaissent sur la page Sécurité. La liste des fonctions concernées s'affiche sous Fonctions AWS Lambda utilisant des environnements d'exécution obsolètes. Cette liste de fonctions affiche à la fois les versions $ LATEST et publiées et les mises à jour sont automatiquement mises à jour pour refléter l'état actuel de vos fonctions.

  Vous pouvez activer les notifications hebdomadaires par e-mail depuis Trusted Advisor la page des préférences de la Trusted Advisor console.

# Environnements d’exécution obsolètes

Les environnements d’exécution suivants ont atteint la fin de prise en charge :

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Node.js 16 | `nodejs16.x` | Amazon Linux 2 | 12 juin 2024 | 28 février 2025 | 31 mars 2025 |
| . NET7 (contenant uniquement) | `dotnet7` | Amazon Linux 2 | 14 mai 2024 | N/A | N/A |
| Java 8 | `java8` | Amazon Linux | 8 janvier 2024 | 8 février 2024 | 28 février 2025 |
| Go 1.x | `go1.x` | Amazon Linux | 8 janvier 2024 | 8 février 2024 | 28 février 2025 |
| Exécution réservée au système d'exploitation | `provided` | Amazon Linux | 8 janvier 2024 | 8 février 2024 | 28 février 2025 |
| Ruby 2.7 | `ruby2.7` | Amazon Linux 2 | 7 décembre 20 | 9 janvier 2024 | 28 février 2025 |
| Node.js 14 | `nodejs14.x` | Amazon Linux 2 | 4 décembre 20 | 9 janvier 2024 | 28 février 2025 |
| Python 3.7 | `python3.7` | Amazon Linux | 4 décembre 20 | 9 janvier 2024 | 28 février 2025 |
| . NETNoyau 3.1 | `dotnetcore3.1` | Amazon Linux 2 | 3 avril 2023 | 3 avril 2023 | 3 mai 2023 |

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Node.js 12 | `nodejs12.x` | Amazon Linux 2 | 31 mars 2023 | 31 mars 2023 | 30 avril 2023 |
| Python 3.6 | `python3.6` | Amazon Linux | 18 juillet 2022 | 18 juillet 2022 | 29 août 2022 |
| . NET5 (contenant uniquement) | `dotnet5.0` | Amazon Linux 2 | 10 mai 2022 | N/A | N/A |
| . NETNoyau 2.1 | `dotnetcore2.1` | Amazon Linux | 5 janvier 2022 | 5 janvier 2022 | 13 avril 2022 |
| Node.js 10 | `nodejs10.x` | Amazon Linux 2 | 30 juillet 2021 | 30 juillet 2021 | 14 février 2022 |
| Ruby 2.5 | `ruby2.5` | Amazon Linux | 30 juillet 2021 | 30 juillet 2021 | 31 mars 2022 |
| Python 2.7 | `python2.7` | Amazon Linux | 15 juillet 2021 | 15 juillet 2021 | 30 mai 2022 |
| Node.js 8.10 | `nodejs8.10` | Amazon Linux | 6 mars 2020 | N/A | 6 mars 2020 |
| Node.js 4.3 | `nodejs4.3` | Amazon Linux | 5 mars 2020 | N/A | 5 mars 2020 |
| Node.js 4.3 edge | `nodejs4.3-edge` | Amazon Linux | 5 mars 2020 | N/A | 30 avril 2019 |
| Node.js 6.10 | `nodejs6.10` | Amazon Linux | 12 août 2019 | 12 août 2019 | N/A |

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| . NETNoyau 1.0 | `dotnetcore1.0` | Amazon Linux | 27 juin 2019 | N/A | 30 juillet 2019 |
| . NETNoyau 2.0 | `dotnetcore2.0` | Amazon Linux | 30 mai 2019 | N/A | 30 mai 2019 |
| Node.js 0.10 | `nodejs` | Amazon Linux | N/A | N/A | 31 octobre 2016 |

Dans presque tous les cas, la end-of-life date d'une version linguistique ou d'un système d'exploitation est connue bien à l'avance. Les liens suivants fournissent des end-of-life plannings pour chaque langage pris en charge par Lambda en tant qu'environnement d'exécution géré.

Stratégies de prise en charge des langages et des infrastructures

- Node.js – github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java — www.oracle.com et Corretto FAQs
- Go – golang.org
- . NET— dotnet.microsoft.com

# Comprendre comment Lambda gère les mises à jour des versions d'exécution

Lambda maintient à jour chaque environnement d'exécution géré avec des mises à jour de sécurité, des corrections de bogues, de nouvelles fonctionnalités, des améliorations de performance et la prise en charge des versions mineures. Ces mises à jour d'environnement d'exécution sont publiées sous forme de versions d'environnement d'exécution. Lambda applique les mises à jour d'environnement d'exécution aux fonctions en faisant migrer la fonction d'une version antérieure vers une nouvelle version.

Par défaut, pour les fonctions utilisant des exécutions gérées, Lambda applique des mises à jour automatiquement. Avec les mises à jour de l'environnement d'exécution automatiques, Lambda prend en charge la charge opérationnelle de l'application de correctifs aux versions de l'environnement d'exécution. Pour la plupart des clients, les mises à jour automatiques sont le bon choix. Vous pouvez modifier ce comportement par défaut en configurant les paramètres de gestion de l'exécution.

Lambda publie également chaque nouvelle version de l'environnement d'exécution sous forme d'image de conteneur. Pour mettre à jour les versions des environnements d'exécution pour les fonctions basées sur des conteneurs, vous devez créer une nouvelle image de conteneur à partir de l'image de base mise à jour et redéployer votre fonction.

Chaque version d'exécution est associée à un numéro de version et à un ARN (Amazon Resource Name). Les numéros de version de l'environnement d'exécution utilisent un schéma de numérotation que Lambda définit, indépendamment des numéros de version utilisés par le langage de programmation. La version d'exécution ARN est un identifiant unique pour chaque version d'exécution. Vous pouvez consulter la version ARN d'exécution actuelle de votre fonction dans la console Lambda ou dans la `INIT_START`ligne de vos journaux de fonctions.

Les versions des environnements d'exécution ne doivent pas être confondues avec les identifiants des environnements d'exécution. Chaque environnement d'exécution possède un identifiant unique, tel que `python3.12` ou `nodejs20.x`. Ceux-ci correspondent à chaque version majeure du langage de programmation. Les versions de l'environnement d'exécution décrivent la version corrective d'un environnement d'exécution individuel.

> **Note**
>
> Le numéro de version d'exécution ARN pour un même environnement d'exécution peut varier selon Régions AWS les CPU architectures.

Rubriques


- Modes de mise à jour pendant
- Déploiement des versions de l’environnement d’exécution en deux phases
- Configuration des paramètres de gestion du runtime Lambda
- Annulation d'une version d'exécution Lambda
- Identification des modifications apportées à la version d'exécution de Lambda
- Comprendre le modèle de responsabilité partagée pour la gestion des environnements d'exécution Lambda
- Contrôle des autorisations de mise à jour du runtime Lambda pour les applications hautement conformes


## Modes de mise à jour pendant

Lambda s’efforce de fournir des mises à jour d’environnement d’exécution qui sont rétrocompatibles avec les fonctions existantes. Cependant, comme pour les correctifs logiciels, il existe de rares cas dans lesquels une mise à jour de l’environnement d’exécution peut avoir un impact négatif sur une fonction existante. Par exemple, les correctifs de sécurité peuvent exposer un problème sous-jacent à une fonction existante qui dépend du comportement précédent, non sécurisé. Les contrôles de gestion de l’environnement d’exécution Lambda permettent de réduire le risque d’impact sur vos charges de travail dans le cas rare d’une incompatibilité de version. Pour chaque version de fonction (`$LATEST` ou version publiée), vous pouvez choisir l’un des modes de mise à jour de l’environnement d’exécution suivants :

- Auto (par défaut) – Mettre automatiquement à jour vers la version de l’environnement d’exécution la plus sécurisée et la plus sûre à l’aide de Déploiement des versions de l’environnement d’exécution en deux phases. Nous recommandons ce mode à la plupart des clients afin que vous puissiez toujours bénéficier des mises à jour de l’environnement d’exécution.
- Mise à jour de fonction – Mise à jour vers la version d’exécution la plus récente et la plus sécurisée lorsque vous mettez à jour votre fonction. Lorsque vous mettez à jour votre fonction, Lambda

met à jour l'exécution de votre fonction vers la version la plus récente et la plus sécurisée. Cette approche synchronise les mises à jour de l'environnement d'exécution avec les déploiements de fonctions, ce qui vous permet de contrôler le moment où Lambda applique les mises à jour de l'environnement d'exécution. Avec ce mode, vous pouvez détecter et atténuer rapidement les rares incompatibilités de mise à jour de l'environnement d'exécution. Lorsque vous utilisez ce mode, vous devez régulièrement mettre à jour vos fonctions pour maintenir leur environnement d'exécution à jour.

- Manuel – Mise à jour manuelle de la version de votre exécution. Vous spécifiez une version de l'exécution dans la configuration de votre fonction. La fonction utilise cette version de l'environnement d'exécution indéfiniment. Dans les rares cas où une nouvelle version de l'environnement d'exécution est incompatible avec une fonction existante, vous pouvez utiliser ce mode pour ramener votre fonction à une version antérieure. Nous vous déconseillons d'utiliser le mode Manual (Manuel) pour tenter d'obtenir une cohérence de l'environnement d'exécution entre les déploiements. Pour de plus amples informations, veuillez consulter Annulation d'une version d'exécution Lambda.

La responsabilité de l'application des mises à jour de l'environnement d'exécution à vos fonctions varie en fonction du mode de mise à jour que vous choisissez. Pour de plus amples informations, veuillez consulter Comprendre le modèle de responsabilité partagée pour la gestion des environnements d'exécution Lambda.

## Déploiement des versions de l'environnement d'exécution en deux phases

Lambda introduit de nouvelles gestions des versions d'exécution dans l'ordre suivant :

1. Dans la première phase, Lambda applique la nouvelle version d'exécution chaque fois que vous créez ou mettez à jour une fonction. Une fonction est mise à jour lorsque vous appelez les UpdateFunctionConfigurationAPIopérations UpdateFunctionCodeor.
2. Dans la deuxième phase, Lambda met à jour toute fonction qui utilise le mode de mise à jour Auto de l'environnement d'exécution et qui n'a pas encore été mise à jour vers la nouvelle version.

La durée totale du processus de déploiement varie en fonction de plusieurs facteurs, notamment la gravité de tout correctif de sécurité inclus dans la mise à jour de l'environnement d'exécution.

Si vous développez et déployez activement vos fonctions, vous obtiendrez très probablement de nouvelles versions de l'environnement d'exécution au cours de la première phase. Cela permet de synchroniser les mises à jour de l'environnement d'exécution avec les mises à jour des fonctions.

Dans le cas rare où la dernière version d'exécution aurait un impact négatif sur votre application, cette approche vous permet de prendre rapidement des mesures correctives. Les fonctions qui ne sont pas en cours de développement bénéficient toujours des avantages opérationnels des mises à jour automatiques de l'environnement d'exécution au cours de la deuxième phase.

Cette approche n'affecte pas les fonctions définies en mode Function update (Mise à jour de fonction) ou en mode Manual (Manuel). Les fonctions utilisant le mode Function update (Mise à jour de fonction) reçoivent les dernières mises à jour de l'environnement d'exécution uniquement lorsque vous les créez ou les mettez à jour. Les fonctions utilisant le mode Manual (Manuel) ne reçoivent pas de mises à jour de l'environnement d'exécution.

Lambda publie les nouvelles versions de l'environnement d'exécution de manière progressive et continue à travers Régions AWS. Si vos fonctions sont configurées en mode Auto ou Function update (Mise à jour de fonction), il est possible que les fonctions déployées au même moment dans différentes régions, ou à différents moments dans la même région, reçoivent des versions de l'environnement d'exécution différentes. Les clients qui exigent une cohérence garantie des versions de l'environnement d'exécution dans leurs environnements doivent utiliser des images de conteneurs pour déployer leurs fonctions Lambda. Le mode Manuel est conçu comme une mesure d'atténuation temporaire pour permettre de restaurer la version d'exécution dans le rare cas où une version d'exécution est incompatible avec votre fonction.

## Configuration des paramètres de gestion du runtime Lambda

Vous pouvez configurer les paramètres de gestion de l'environnement d'exécution à l'aide de la console Lambda ou de AWS Command Line Interface (AWS CLI).

> **Note**
>
> Vous pouvez configurer les paramètres de gestion de l'environnement d'exécution séparément pour chaque version de fonction.

Pour configurer la façon dont Lambda met à jour la version de votre environnement d'exécution (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.

3. Dans l'onglet Code, sous Runtime settings (Paramètres de l'environnement d'exécution), choisissez Edit runtime management configuration (Modifier la configuration de gestion de l'environnement d'exécution).
4. Sous Runtime management configuration (Configuration de la gestion de l'environnement d'exécution), choisissez l'une des options suivantes :

   - Pour que votre fonction se mette automatiquement à jour vers la dernière version de l'environnement d'exécution, sélectionnez Auto.
   - Pour que votre fonction se mette à jour vers la dernière version de l'environnement d'exécution lorsque vous modifiez la fonction, sélectionnez Function update (Mise à jour de fonction).
   - Pour que votre fonction soit mise à jour vers la dernière version d'exécution uniquement lorsque vous modifiez la version d'exécutionARN, choisissez Manuel. Vous trouverez la version d'exécution ARN sous Configuration de la gestion du temps d'exécution. Vous pouvez également les trouver ARN dans la `INIT_START` ligne de vos journaux de fonctions.

   Pour plus d'informations sur ces options, consultez la section Modes de mise à jour d'exécution.
5. Choisissez Save (Enregistrer).

Pour configurer la façon dont Lambda met à jour votre version de l'environnement d'exécution (AWS CLI)

Pour configurer la gestion de l'exécution d'une fonction, exécutez la put-runtime-management-config AWS CLI commande. Lorsque vous utilisez `Manual` le mode, vous devez également fournir la version d'exécutionARN.

```
aws lambda put-runtime-management-config \
  --function-name my-function \
  --update-runtime-on Manual \
  --runtime-version-arn arn:aws:lambda:us-
east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1
```

Vous devez voir des résultats similaires à ce qui suit :

```
{
  "UpdateRuntimeOn": "Manual",
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
  "RuntimeVersionArn": "arn:aws:lambda:us-
east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
```

```
}
```

## Annulation d'une version d'exécution Lambda

Dans le cas rare où une nouvelle version de l'environnement d'exécution est incompatible avec votre fonction existante, vous pouvez rétablir sa version à une version antérieure. Cela permet à votre application de continuer à fonctionner et de minimiser les perturbations, en laissant le temps de remédier à l'incompatibilité avant de revenir à la dernière version de l'environnement d'exécution.

Lambda n'impose pas de limite de temps à l'utilisation d'une version de l'environnement d'exécution particulière. Cependant, nous vous recommandons vivement de passer à la dernière version de l'environnement d'exécution dès que possible pour bénéficier des derniers correctifs de sécurité, des améliorations de performances et des fonctionnalités. Lambda offre la possibilité de revenir à une version antérieure de l'environnement d'exécution uniquement à titre d'atténuation temporaire dans le cas rare d'un problème de compatibilité de mise à jour de l'environnement d'exécution. Les fonctions qui utilisent une version d'exécution antérieure pendant une période prolongée peuvent subir une dégradation des performances ou des problèmes, tels que l'expiration d'un certificat, qui peuvent les empêcher de fonctionner correctement.

Vous pouvez rétablir une version de l'environnement d'exécution de l'une des manières suivantes :

- Utilisation du mode de mise à jour manuelle
- En utilisant les versions de la fonction publiée

Pour plus d'informations, consultez la section Présentation des contrôles de gestion du temps AWS Lambda d'exécution sur le blog AWS Compute.

### Rétablissement d'une version de l'environnement d'exécution à l'aide du mode de mise à jour Manual (Manuel)

Si vous utilisez le mode de mise à jour Auto de la version de l'environnement d'exécution, ou si vous utilisez la version `$LATEST`, vous pouvez rétablir la version de votre environnement d'exécution en utilisant le mode Manual (Manuel). Pour la version de fonction que vous souhaitez restaurer, remplacez le mode de mise à jour de la version d'exécution par Manuel et spécifiez ARN celle de la version d'exécution précédente. Pour plus d'informations sur la recherche ARN de la version d'exécution précédente, consultezIdentification des modifications apportées à la version d'exécution de Lambda.

> **Note**
>
> Si la `$LATEST` version de votre fonction est configurée pour utiliser le mode manuel, vous ne pouvez pas modifier l'CPUarchitecture ou la version d'exécution utilisée par votre fonction. Pour effectuer ces modifications, vous devez passer en mode Auto ou Function update (Mise à jour de fonction).

### Rétablissement d’une version de l’environnement d’exécution à l’aide des versions de fonction publiées

Les versions de fonction publiées sont un instantané immuable du code et de la configuration de la fonction `$LATEST` au moment où vous les avez créées. En mode Auto, Lambda met automatiquement à jour la version de l’environnement d’exécution des versions de fonctions publiées pendant la phase deux du déploiement de la version de l’environnement d’exécution. En mode Function update (Mise à jour de fonction), Lambda ne met pas à jour la version de l’environnement d’exécution des versions de fonctions publiées.

Les versions de fonctions publiées utilisant le mode Function update (Mise à jour de fonction) créent donc un instantané statique du code de la fonction, de la configuration et de la version de l’environnement d’exécution. En utilisant le mode Function update (Mise à jour de fonction) avec les versions de fonctions, vous pouvez synchroniser les mises à jour de l’environnement d’exécution avec vos déploiements. Vous pouvez également coordonner le rétablissement des versions de code, de configuration et de l’environnement d’exécution en redirigeant le trafic vers une version de fonction publiée antérieurement. Vous pouvez intégrer cette approche à votre système d’intégration et de livraison continues (CI/CD) afin de rétablir automatiquement la situation dans les rares cas d’incompatibilité des mises à jour de l’environnement d’exécution. Lorsque vous utilisez cette approche, vous devez mettre à jour votre fonction régulièrement et publier de nouvelles versions de la fonction pour récupérer les dernières mises à jour de l’exécution. Pour de plus amples informations, veuillez consulter Comprendre le modèle de responsabilité partagée pour la gestion des environnements d'exécution Lambda.

## Identification des modifications apportées à la version d'exécution de Lambda

Le numéro de version d'exécution et ARN sont enregistrés dans la ligne de `INIT_START` journal, que Lambda envoie à CloudWatch Logs chaque fois qu'il crée un nouvel environnement d'exécution.

Étant donné que l'environnement d'exécution utilise la même exécution pour tous les invocations de fonction, Lambda émet la ligne de journal `INIT_START` uniquement lorsqu'il exécute la phase init. Lambda n'émet pas cette ligne de journal pour chaque invocation de fonction. Lambda envoie la ligne de journal à CloudWatch Logs, mais elle n'est pas visible dans la console.

Example Exemple INIT _ START log line

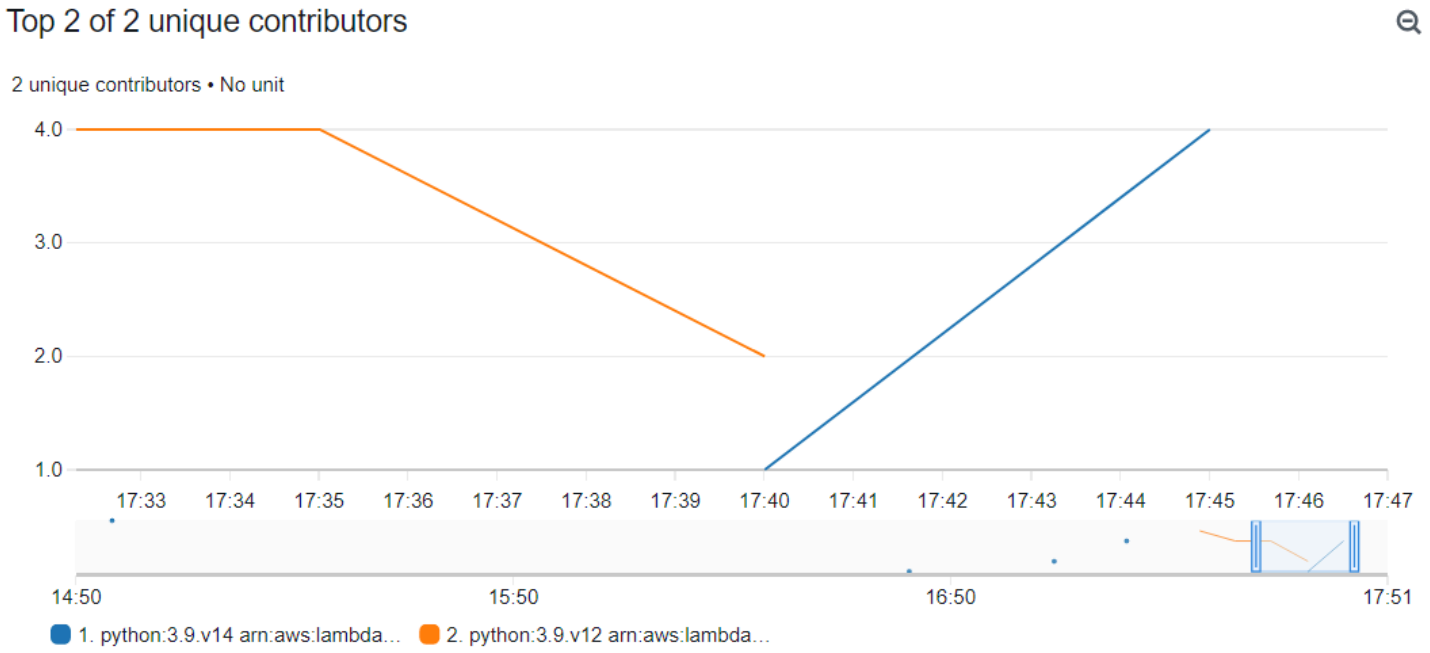
```
INIT_START Runtime Version: python:3.9.v14    Runtime Version ARN: arn:aws:lambda:eu-
south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Plutôt que de travailler directement avec les journaux, vous pouvez utiliser Amazon CloudWatch Contributor Insights pour identifier les transitions entre les versions d'exécution. La règle suivante compte les versions de l'environnement d'exécution distinctes de chaque ligne de journal `INIT_START`. Pour utiliser la règle, remplacez l'exemple de nom de groupe de journaux `/aws/lambda/*` par le préfixe approprié pour votre fonction ou groupe de fonctions.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "eventType",
        "In": [
          "INIT_START"
        ]
      }
    ],
    "Keys": [
      "runtimeVersion",
      "runtimeVersionArn"
    ]
  },
  "LogFormat": "CLF",
  "LogGroupNames": [
    "/aws/lambda/*"
  ],
  "Fields": {
    "1": "eventType",
```

```
    "4": "runtimeVersion",
    "8": "runtimeVersionArn"
  }
}
```

Le rapport CloudWatch Contributor Insights suivant montre un exemple de transition de version d'exécution tel qu'illustré par la règle précédente. La ligne orange montre l'initialisation de l'environnement d'exécution pour la version précédente (python:3.9.v12), et la ligne bleue montre l'initialisation de l'environnement d'exécution pour la nouvelle version (python:3.9.v14).




## Comprendre le modèle de responsabilité partagée pour la gestion des environnements d'exécution Lambda

Lambda est responsable de la conservation et de la publication des mises à jour de sécurité pour tous les environnements d'exécution gérés et les images de conteneurs pris en charge. La responsabilité de la mise à jour des fonctions existantes pour utiliser la dernière version de l'environnement d'exécution varie en fonction du mode de mise à jour que vous utilisez.

Lambda est responsable de l'application des mises à jour de l'environnement d'exécution à toutes les fonctions configurées pour utiliser le mode de mise à jour Auto.

Pour les fonctions configurées avec le mode de mise à jour d'exécution Function update (Mise à jour de fonction), vous êtes responsable de la mise à jour régulière de votre fonction. Lambda est responsable de l'application des mises à jour de l'environnement d'exécution lorsque vous

effectuez ces mises à jour. Si vous ne mettez pas à jour votre fonction, Lambda ne met pas à jour l'environnement d'exécution. Si vous ne mettez pas régulièrement à jour votre fonction, nous vous recommandons vivement de la configurer pour les mises à jour automatiques de l'environnement d'exécution afin qu'elle continue à recevoir les mises à jour de sécurité.

Pour les fonctions configurées pour utiliser le mode de mise à jour Manual (Manuel), vous êtes responsable de la mise à jour de votre fonction pour utiliser la dernière version de l'environnement d'exécution. Nous vous recommandons vivement d'utiliser ce mode uniquement pour rétablir la version de l'environnement d'exécution à titre de mesure d'atténuation temporaire dans le cas rare d'une incompatibilité de mise à jour. Nous vous recommandons également de passer en mode Auto le plus rapidement possible afin de minimiser le temps pendant lequel vos fonctions ne sont pas corrigées.

Si vous utilisez des images de conteneur pour déployer vos fonctions, Lambda est responsable de la publication des images de base mises à jour. Dans ce cas, vous êtes responsable de la recréation de l'image de conteneur de votre fonction à partir de la dernière image de base et du redéploiement de l'image de conteneur.

Ceci est résumé dans le tableau suivant :

| Mode de déploiement | Responsabilité de Lambda | Responsabilité du client |
|---|---|---|
| Environnement d'exécution géré, mode Auto | Publiez de nouvelles versions de l'environnement d'exécution contenant les derniers correctifs.<br><br>Appliquez les correctifs de l'environnement d'exécution aux fonctions existantes. | Rétablissez une version de l'environnement d'exécution précédente dans le cas rare d'un problème de compatibilité de mise à jour de l'environnement d'exécution. |
| Environnement d'exécution géré, mode Function update (Mise | Publiez de nouvelles versions de l'environnement d'exécution contenant les derniers correctifs. | Mettez régulièrement à jour les fonctions afin de bénéficier de la dernière version de l'environnement d'exécution.<br><br>Passez une fonction en mode Auto lorsque vous ne la mettez pas régulièrement à jour. |

| Mode de déploiement | Responsabilité de Lambda | Responsabilité du client |
| --- | --- | --- |
| à jour de fonction) | | Rétablissez une version de l'environnement d'exécution précédente dans le cas rare d'un problème de compatibilité de mise à jour de l'environnement d'exécution. |
| Environnement d'exécution géré, mode Manual (Manuel) | Publiez de nouvelles versions de l'environnement d'exécution contenant les derniers correctifs. | Utilisez ce mode uniquement pour restaurer temporairement l'environnement d'exécution dans le cas rare d'un problème de compatibilité de mise à jour.<br><br>Passez les fonctions en mode Auto ou Function update (Mise à jour de fonction) et la dernière version de l'environnement d'exécution dès que possible. |
| Image de conteneur | Publiez de nouvelles images de conteneur contenant les derniers correctifs. | Redéployez régulièrement les fonctions en utilisant la dernière image de base du conteneur pour bénéficier des derniers correctifs. |

Pour plus d'informations sur le partage des responsabilités avec AWS, voir Modèle de responsabilité partagée.

## Contrôle des autorisations de mise à jour du runtime Lambda pour les applications hautement conformes

Pour répondre aux exigences de correctifs, les clients Lambda s'appuient généralement sur des mises à jour automatiques de l'environnement d'exécution. Si votre application est soumise à des exigences strictes en matière de nouveauté des correctifs, vous voudrez peut-être limiter l'utilisation de versions de l'environnement d'exécution antérieures. Vous pouvez restreindre les contrôles de gestion d'exécution de Lambda en utilisant AWS Identity and Access Management (IAM) pour refuser aux utilisateurs de votre AWS compte l'accès à l'PutRuntimeManagementConfigAPIopération. Cette opération est utilisée pour choisir le mode de mise à jour de l'environnement d'exécution d'une fonction. En refusant l'accès à cette opération, toutes les fonctions passent par défaut au mode Auto.

Vous pouvez appliquer cette restriction à l'ensemble de votre organisation à l'aide d'une politique de contrôle des services (SCP). Si vous devez rétablir une version d'exécution antérieure d'une fonction, vous pouvez accorder une exception de politique sur une case-by-case base déterminée.

# Récupérer des données sur les fonctions Lambda qui utilisent un environnement d'exécution obsolète

Lorsqu'un environnement d'exécution Lambda est sur le point de devenir obsolète, Lambda vous avertit par e-mail et vous envoie des notifications dans le et. AWS Health Dashboard Trusted Advisor Ces e-mails et notifications répertorient les LATEST versions $ des fonctions utilisant le moteur d'exécution. Pour répertorier toutes les versions de vos fonctions qui utilisent un environnement d'exécution particulier, vous pouvez utiliser le AWS Command Line Interface (AWS CLI) ou l'un des AWS SDKs.

Si un grand nombre de fonctions utilisent un environnement d'exécution destiné à être obsolète, vous pouvez également utiliser le AWS CLI ou pour vous aider AWS SDKs à prioriser les mises à jour de vos fonctions les plus fréquemment invoquées.

Reportez-vous aux sections suivantes pour savoir comment utiliser AWS CLI et AWS SDKs pour collecter des données sur les fonctions qui utilisent un environnement d'exécution particulier.

## Liste des versions de fonctions qui utilisent un environnement d'exécution particulier

Pour utiliser le AWS CLI pour répertorier toutes les versions de vos fonctions qui utilisent un environnement d'exécution particulier, exécutez la commande suivante. `RUNTIME_IDENTIFIER`Remplacez-le par le nom du moteur d'exécution qui est obsolète et choisissez le vôtre. Région AWS Pour répertorier uniquement les versions de LATEST fonction $, omettez cette option `--function-version ALL` dans la commande.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --
query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

> **Tip**
>
> L'exemple de commande répertorie les fonctions d'une `us-east-1` région en particulier. Compte AWS Vous devrez répéter cette commande pour chaque région dans laquelle votre compte possède des fonctions et pour chacune de vos Comptes AWS.

Vous pouvez également répertorier les fonctions qui utilisent un environnement d'exécution particulier à l'aide de l'un des AWS SDKs. L'exemple de code suivant utilise le V3 AWS SDK for JavaScript et le

AWS SDK for Python (Boto3) pour renvoyer une liste des fonctions ARNs utilisant un environnement d'exécution particulier. L'exemple de code renvoie également le groupe de CloudWatch journaux pour chacune des fonctions répertoriées. Vous pouvez utiliser ce groupe de journaux pour trouver la date du dernier appel de la fonction. Consultez la section suivante the section called “Identifier les fonctions les plus fréquemment et les plus récemment invoquées” pour plus d'informations.

Node.js

Example JavaScript code pour répertorier les fonctions utilisant un environnement d'exécution particulier

```
import { LambdaClient, ListFunctionsCommand } from "@aws-sdk/client-lambda";
const lambdaClient = new LambdaClient();

const command = new ListFunctionsCommand({
    FunctionVersion: "ALL",
    MaxItems: 50
});
const response = await lambdaClient.send(command);

for (const f of response.Functions){
    if (f.Runtime == '<your_runtime>'){ // Use the runtime id, e.g. 'nodejs18.x' or
 'python3.9'
        console.log(f.FunctionArn);
        // get the CloudWatch log group of the function to
        // use later for finding the last invocation date
        console.log(f.LoggingConfig.LogGroup);
    }
}
// If your account has more functions than the specified
// MaxItems, use the returned pagination token in the
// next request with the 'Marker' parameter
if ('NextMarker' in response){
    let paginationToken = response.NextMarker;
  }
```

Python

Example Code Python pour répertorier les fonctions utilisant un environnement d'exécution particulier

```
import boto3
```

```
from botocore.exceptions import ClientError

def list_lambda_functions(target_runtime):

    lambda_client = boto3.client('lambda')

    response = lambda_client.list_functions(
        FunctionVersion='ALL',
        MaxItems=50
    )
    if not response['Functions']:
            print("No Lambda functions found")
    else:
        for function in response['Functions']:
            if function['PackageType']=='Zip' and function['Runtime'] ==
 target_runtime:
                print(function['FunctionArn'])
                # Print the CloudWatch log group of the function
                # to use later for finding last invocation date
                print(function['LoggingConfig']['LogGroup'])

    if 'NextMarker' in response:
       pagination_token = response['NextMarker']

if __name__ == "__main__":
    # Replace python3.12 with the appropriate runtime ID for your Lambda functions
    list_lambda_functions('python3.12')
```

Pour en savoir plus sur l'utilisation de an AWS SDK pour répertorier vos fonctions à l'aide de l'ListFunctionsaction, consultez la SDKdocumentation de votre langage de programmation préféré.

Vous pouvez également utiliser la fonctionnalité de requêtes AWS Config avancées pour répertorier toutes les fonctions qui utilisent un environnement d'exécution affecté. Cette requête renvoie uniquement LATEST les versions de la fonction $, mais vous pouvez agréger les requêtes pour répertorier les fonctions dans toutes les régions et plusieurs à l' Comptes AWS aide d'une seule commande. Pour en savoir plus, consultez la section Interrogation de l'état de configuration actuel des AWS Auto Scaling ressources dans le guide du AWS Config développeur.

# Identifier les fonctions les plus fréquemment et les plus récemment invoquées

Si vous utilisez Compte AWS des fonctions qui utilisent un environnement d'exécution destiné à être obsolète, vous souhaiterez peut-être donner la priorité à la mise à jour des fonctions fréquemment invoquées ou des fonctions qui ont été invoquées récemment.

Si vous ne disposez que de quelques fonctions, vous pouvez utiliser la console CloudWatch Logs pour recueillir ces informations en consultant les flux de journaux de vos fonctions. Voir Afficher les données de journal envoyées à CloudWatch Logs pour plus d'informations.

Pour connaître le nombre d'appels de fonctions récents, vous pouvez également utiliser les informations de CloudWatch métriques affichées dans la console Lambda. Pour consulter ces informations, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction pour laquelle vous souhaitez consulter les statistiques d'invocation.
3. Choisissez l'onglet Monitor (Surveiller).
4. Définissez la période pendant laquelle vous souhaitez consulter les statistiques à l'aide du sélecteur de plage de dates. Les appels récents sont affichés dans le volet Invocations.

Pour les comptes dotés d'un plus grand nombre de fonctions, il peut être plus efficace de collecter ces données par programmation en utilisant l' AWS CLI action DescribeLogStreamset GetMetricStatisticsAPIou l' AWS SDKsune des actions.

Les exemples suivants fournissent des extraits de code utilisant la V3 AWS SDK for JavaScript et le AWS SDK for Python (Boto3) pour identifier la date du dernier appel pour une fonction particulière et pour déterminer le nombre d'appels pour une fonction particulière au cours des 14 derniers jours.

Node.js

Example JavaScript code pour trouver l'heure du dernier appel d'une fonction

```
import { CloudWatchLogsClient, DescribeLogStreamsCommand } from "@aws-sdk/client-
cloudwatch-logs";
const cloudWatchLogsClient = new CloudWatchLogsClient();
const command = new DescribeLogStreamsCommand({
    logGroupName: '<your_log_group_name>',
```

```
    orderBy: 'LastEventTime',
    descending: true,
    limit: 1
});
try {
    const response = await cloudWatchLogsClient.send(command);
    const lastEventTimestamp = response.logStreams.length > 0 ?
        response.logStreams[0].lastEventTimestamp : null;
    // Convert the UNIX timestamp to a human-readable format for display
    const date = new Date(lastEventTimestamp).toLocaleDateString();
    const time = new Date(lastEventTimestamp).toLocaleTimeString();
    console.log(`${date} ${time}`);

} catch (e){
    console.error('Log group not found.')
}
```

### Python

#### Example Code Python pour trouver l'heure du dernier appel d'une fonction

```
import boto3
from datetime import datetime

cloudwatch_logs_client  = boto3.client('logs')

response = cloudwatch_logs_client.describe_log_streams(
    logGroupName='<your_log_group_name>',
    orderBy='LastEventTime',
    descending=True,
    limit=1
)

try:
    if len(response['logStreams']) > 0:
        last_event_timestamp = response['logStreams'][0]['lastEventTimestamp']
        print(datetime.fromtimestamp(last_event_timestamp/1000)) # Convert timestamp
 from ms to seconds
    else:
        last_event_timestamp = None
except:
    print('Log group not found')
```

> **Tip**
>
> Vous pouvez trouver le nom du groupe de journaux de votre fonction à l'aide de l'ListFunctionsAPIopération. Consultez le code ci-dessous the section called “Liste des versions de fonctions qui utilisent un environnement d'exécution particulier” pour un exemple de la procédure à suivre.

### Node.js

Example JavaScript code pour trouver le nombre d'invocations au cours des 14 derniers jours

```
import { CloudWatchClient, GetMetricStatisticsCommand } from "@aws-sdk/client-
cloudwatch";
const cloudWatchClient = new CloudWatchClient();
const command = new GetMetricStatisticsCommand({
    Namespace: 'AWS/Lambda',
    MetricName: 'Invocations',
    StartTime: new Date(Date.now()-86400*1000*14), // 14 days ago
    EndTime: new Date(Date.now()),
    Period: 86400 * 14, // 14 days.
    Statistics: ['Sum'],
    Dimensions: [{
        Name: 'FunctionName',
        Value: '<your_function_name>'
    }]
});
const response = await cloudWatchClient.send(command);
const invokesInLast14Days = response.Datapoints.length > 0 ?
    response.Datapoints[0].Sum : 0;

console.log('Number of invocations: ' + invokesInLast14Days);
```

### Python

Example Code Python pour trouver le nombre d'invocations au cours des 14 derniers jours

```
import boto3
from datetime import datetime, timedelta

cloudwatch_client = boto3.client('cloudwatch')
```

```
response = cloudwatch_client.get_metric_statistics(
    Namespace='AWS/Lambda',
    MetricName='Invocations',
    Dimensions=[
        {
            'Name': 'FunctionName',
            'Value': '<your_function_name>'
        },
    ],
    StartTime=datetime.now() - timedelta(days=14),
    EndTime=datetime.now(),
    Period=86400 * 14, # 14 days
    Statistics=[
        'Sum'
    ]
)

if len(response['Datapoints']) > 0:
    invokes_in_last_14_days = int(response['Datapoints'][0]['Sum'])
else:
    invokes_in_last_14_days = 0

print(f'Number of invocations: {invokes_in_last_14_days}')
```

# Modification de l’environnement d’exécution

Vous pouvez utiliser des extensions internes pour modifier le processus de runtime. Les extensions internes ne sont pas des processus séparés. Elles s’exécutent dans le cadre du processus du runtime.

Lambda fournit des variables d’environnement spécifiques du langage, que vous pouvez définir pour ajouter des options et des outils au runtime. Lambda intègre également des scripts encapsuleurs qui lui permettent de déléguer le démarrage du runtime à votre script. Vous pouvez créer un script encapsuleur pour personnaliser le comportement de démarrage du runtime.

## Variables d’environnement spécifiques à un langage

Lambda prend en charge des méthodes de configuration pour permettre le pré-chargement de code lors de l’initialisation de la fonction via les variables d’environnement spécifiques du langage suivantes :

- `JAVA_TOOL_OPTIONS` : sur Java, Lambda prend en charge cette variable d’environnement pour définir des variables de ligne de commande supplémentaires dans Lambda. Cette variable d’environnement vous permet de spécifier l’initialisation des outils, en particulier le lancement d’agents de langage de programmation natifs ou Java à l’aide des options `agentlib` ou `javaagent`. Pour plus d’informations, veuillez consulter la rubrique Variables d’environnement `JAVA_TOOL_OPTIONS`.
- `NODE_OPTIONS`— Disponible dans les environnements d’exécution de Node.js.
- `DOTNET_STARTUP_HOOKS` – Sur .NET Core 3.1 et versions ultérieures, cette variable d’environnement précise le chemin d’accès d’un assembly (dll) que Lambda peut utiliser.

L’utilisation de variables d’environnement spécifiques à un langage est la méthode privilégiée pour définir les propriétés de démarrage.

## Scripts encapsuleurs

Vous pouvez créer un script encapsuleur pour personnaliser le comportement de démarrage du runtime de votre fonction Lambda. Un script encapsuleur vous permet de définir des paramètres de configuration qui ne peuvent pas être définis via des variables d’environnement spécifiques à un langage.

> **Note**
>
> Les invocations peuvent échouer si le script encapsuleur ne démarre pas correctement le processus de runtime.

Les scripts encapsulateurs sont pris en charge sur tous les environnements d'exécution Lambda natifs. Les scripts Wrapper ne sont pas pris en charge sur Exécutions uniquement basées sur le système d'exploitation (famille d'environnements d'exécution `provided`).

Lorsque vous utilisez un script encapsuleur pour votre fonction, Lambda démarre le runtime en utilisant votre script. Lambda envoie à votre script le chemin d'accès de l'interpréteur et tous les arguments d'origine pour le démarrage du runtime standard. Votre script peut étendre ou transformer le comportement de démarrage du programme. Par exemple, le script peut injecter et modifier des arguments, définir des variables d'environnement ou capturer des métriques, des erreurs et d'autres informations de diagnostic.

Vous spécifiez le script en définissant la valeur de la variable d'environnement `AWS_LAMBDA_EXEC_WRAPPER` comme étant le chemin vers le système de fichiers d'un fichier binaire ou d'un script exécutable.

## Exemple : Créer et utiliser un script encapsuleur avec Python 3.8

Dans l'exemple suivant, vous créez un script encapsuleur pour démarrer l'interpréteur Python avec l'option `-X importtime`. Lorsque vous exécutez la fonction, Lambda génère une entrée de journal pour indiquer la durée de chaque importation.

Pour créer et utiliser un script encapsuleur avec Python 3.8

1. Pour créer le script encapsuleur, collez le code suivant dans un fichier nommé `importtime_wrapper`:

```
#!/bin/bash

# the path to the interpreter and all of the originally intended arguments
args=("$@")

# the extra options to pass to the interpreter
extra_args=("-X" "importtime")
```

```
# insert the extra options
args=("${args[@]:0:$#-1}" "${extra_args[@]}" "${args[@]: -1}")

# start the runtime with the extra options
exec "${args[@]}"
```

2. Pour donner au script des autorisations d'exécution, entrez `chmod +x importtime_wrapper` à partir de la ligne de commande.
3. Déployez le script en tant que couche Lambda.
4. Créez une fonction à l'aide de la console Lambda.
   a. Ouvrez la console Lambda.
   b. Sélectionnez Create function (Créer une fonction).
   c. Sous Informations de base, pour Nom de la fonction, entrez **`wrapper-test-function`**.
   d. Pour Environnement d'exécution, sélectionnez Python 3.8.
   e. Sélectionnez Create function (Créer une fonction).
5. Ajoutez la couche à votre fonction.
   a. Choisissez votre fonction, puis choisissez l'option Code si elle n'est pas déjà sélectionnée.
   b. Choisissez Add a layer (Ajouter une couche).
   c. Sous Choose a layer (Choisir une couche), choisissez le nom et la version de la couche compatible que vous avez créée précédemment.
   d. Choisissez Ajouter.
6. Ajoutez le code et la variable d'environnement à votre fonction.
   a. Dans l'éditeur de code, de fonction, collez le code de fonction suivant :

```
import json

  def lambda_handler(event, context):
      # TODO implement
      return {
          'statusCode': 200,
          'body': json.dumps('Hello from Lambda!')
      }
```

b. Choisissez Enregistrer.

c. Sous Variables d'environnement, choisissez Modifier.

d. Choisissez Ajouter une variable d'environnement.

e. Pour Clé, entrez `AWS_LAMBDA_EXEC_WRAPPER`.

f. Pour le champ Value (Valeur), entrez `/opt/importtime_wrapper`.

g. Choisissez Enregistrer.

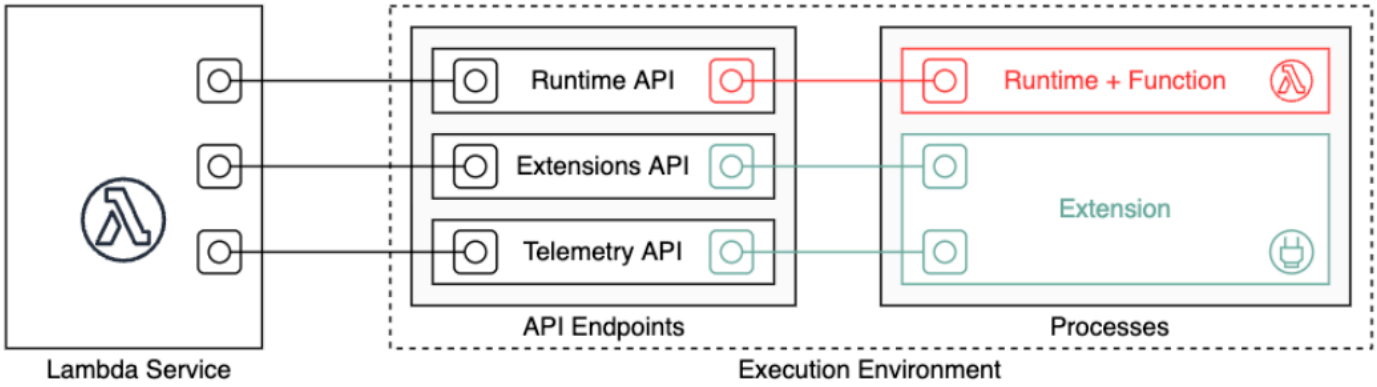
7. Pour exécuter la fonction, choisissez Tester.

   Votre script encapsuleur ayant lancé l'interpréteur Python avec l'option `-X importtime`, les journaux indiquent le temps requis pour chaque importation. Exemples :

```
...
2020-06-30T18:48:46.780+01:00 import time: 213 | 213 | simplejson
2020-06-30T18:48:46.780+01:00 import time: 50 | 263 | simplejson.raw_json
...
```

# Utilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés

AWS Lambda fournit un environnement d'exécution personnalisé HTTP API pour recevoir des événements d'appel de Lambda et renvoyer des données de réponse dans l'environnement d'exécution Lambda. Cette section contient la API référence pour le runtime Lambda. API




La API spécification Open pour la API version d'exécution 2018-06-01 est disponible dans le fichier runtime-api.zip

Pour créer une API demandeURL, les environnements d'exécution obtiennent le API point de terminaison à partir de la variable d'`AWS_LAMBDA_RUNTIME_API`environnement, ajoutent la API version et ajoutent le chemin de ressource souhaité.

Example Demande

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

APIméthodes

- Invocation suivante
- Réponse d'invocation
- Erreur d'initialisation
- Erreur d'invocation

## Invocation suivante

Chemin – `/runtime/invocation/next`

Méthode — GET

L'exécution envoie ce message à Lambda pour demander un événement d'invocation. Le corps de la réponse contient la charge utile de l'invocation, qui est un JSON document contenant les données d'événements du déclencheur de fonction. Les en-têtes de la réponse contiennent des données supplémentaires sur l'invocation.

En-têtes de réponse

- `Lambda-Runtime-Aws-Request-Id` – ID de demande qui identifie la demande ayant déclenché l'invocation de la fonction.

  Par exemple, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.
- `Lambda-Runtime-Deadline-Ms` – Date à laquelle la fonction expire, exprimée en millisecondes au format horaire Unix.

  Par exemple, `1542409706888`.
- `Lambda-Runtime-Invoked-Function-Arn`— De la fonction, ARN de la version ou de l'alias Lambda spécifiée lors de l'appel.

  Par exemple, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.
- `Lambda-Runtime-Trace-Id` – En-tête de suivi AWS X-Ray.

  Par exemple, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.
- `Lambda-Runtime-Client-Context`— Pour les appels depuis le AWS mobileSDK, les données relatives à l'application client et à l'appareil.
- `Lambda-Runtime-Cognito-Identity`— Pour les appels depuis le AWS mobileSDK, données relatives au fournisseur d'identité Amazon Cognito.

Ne définissez pas de délai d'expiration pour la demande `GET`, car la réponse peut être retardée. Entre le moment où Lambda amorce le runtime et celui où le runtime a un événement à renvoyer, le processus d'exécution peut être bloqué pendant plusieurs secondes.

L'ID de demande suit l'invocation au sein de Lambda. Utilisez-le pour spécifier l'invocation lorsque vous envoyez la réponse.

L'en-tête de suivi contient l'ID de suivi, l'ID parent et la décision d'échantillonnage. Si la demande est échantillonnée, elle a été échantillonnée par Lambda ou un service en amont. Le runtime doit définir

l'_X_AMZN_TRACE_ID avec la valeur de l'en-tête. Le X-Ray SDK lit ceci pour obtenir la demande IDs et déterminer s'il faut suivre la demande.

## Réponse d'invocation

Chemin – /runtime/invocation/*AwsRequestId*/response

Méthode — POST

Une fois l'exécution de la fonction terminée, le runtime envoie une réponse à l'invocation à Lambda. Pour les invocations synchrones, Lambda envoie la réponse au client.

Example demande d'opération réussie

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "SUCCESS"
```

## Erreur d'initialisation

Si la fonction renvoie une erreur ou si le runtime rencontre une erreur lors de l'initialisation, le runtime utilise cette méthode pour signaler l'erreur à Lambda.

Chemin – /runtime/init/error

Méthode — POST

En-têtes

Lambda-Runtime-Function-Error-Type – Type d'erreur que l'environnement d'exécution a rencontré. Requis : non.

Cet en-tête se compose d'une valeur de chaîne. Lambda accepte n’importe quelle chaîne, mais nous recommandons le format <category.reason>. Par exemple :

- Durée d'exécution. NoSuchHandler
- Durée d'exécution. APIKeyNotFound
- Durée d'exécution. ConfigInvalid
- Durée d'exécution. UnknownReason

Paramètres de corps

`ErrorRequest` : informations sur l'erreur. Requis : non.

Ce champ est un JSON objet dont la structure est la suivante :

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Notez que Lambda accepte n’importe quelle valeur pour `errorType`.

L’exemple suivant montre un message d’erreur de fonction Lambda indiquant que la fonction n’a pas pu analyser les données d’événement fournies dans l’invocation.

Example Erreur de fonction

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Paramètres du corps de la réponse

- `StatusResponse` – String. Informations d'état, envoyées avec les codes de réponse 202.
- `ErrorResponse` – Informations supplémentaires sur l'erreur, envoyées avec les codes de réponse d'erreur. ErrorResponse contient un type d'erreur et un message d'erreur.

Codes de réponse

- 202 – Accepté
- 403 – Interdit
- 500 – Erreur de conteneur. État non récupérable. L'environnement d'exécution doit se terminer rapidement.

Example demande d'erreur d'initialisation

```
ERROR="{\"errorMessage\" : \"Failed to load function.\", \"errorType\" :
 \"InvalidFunctionException\"}"
```

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR" --
header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

## Erreur d'invocation

Si la fonction renvoie une erreur ou si le runtime rencontre une erreur, le runtime utilise cette méthode pour signaler l'erreur à Lambda.

Chemin – `/runtime/invocation/AwsRequestId/error`

Méthode — POST

En-têtes

`Lambda-Runtime-Function-Error-Type` – Type d'erreur que l'environnement d'exécution a rencontré. Requis : non.

Cet en-tête se compose d'une valeur de chaîne. Lambda accepte n'importe quelle chaîne, mais nous recommandons le format <category.reason>. Par exemple :

- Durée d'exécution. NoSuchHandler
- Durée d'exécution. APIKeyNotFound
- Durée d'exécution. ConfigInvalid
- Durée d'exécution. UnknownReason

Paramètres de corps

`ErrorRequest` : informations sur l'erreur. Requis : non.

Ce champ est un JSON objet dont la structure est la suivante :

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Notez que Lambda accepte n'importe quelle valeur pour `errorType`.

L'exemple suivant montre un message d'erreur de fonction Lambda indiquant que la fonction n'a pas pu analyser les données d'événement fournies dans l'invocation.

Example Erreur de fonction

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Paramètres du corps de la réponse

- `StatusResponse` – String. Informations d'état, envoyées avec les codes de réponse 202.
- `ErrorResponse` – Informations supplémentaires sur l'erreur, envoyées avec les codes de réponse d'erreur. ErrorResponse contient un type d'erreur et un message d'erreur.

Codes de réponse

- 202 – Accepté
- 400 – Demande erronée.
- 403 – Interdit
- 500 – Erreur de conteneur. État non récupérable. L'environnement d'exécution doit se terminer rapidement.

Example demande d'erreur

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
ERROR="{\"errorMessage\" : \"Error parsing event data.\", \"errorType\" :
 \"InvalidEventDataException\"}"
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/error"
 -d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

# Quand utiliser les environnements d'exécution uniquement pour le système d'exploitation de Lambda

Lambda fournit des environnements d'exécution gérés pour Java, Python, Node.js,. NET, et Ruby. Pour créer des fonctions Lambda dans un langage de programmation qui n'est pas disponible en tant qu'environnement d'exécution géré, utilisez un environnement d'exécution uniquement basé sur le système d'exploitation (famille des environnements d'exécution `provided`). Il existe trois principaux cas d'utilisation pour les environnements d'exécution uniquement basés sur le système d'exploitation :

- Compilation native ahead-of-time (AOT) : des langages tels que Go, Rust et C++ se compilent nativement en un binaire exécutable, ce qui ne nécessite pas d'exécution de langage dédié. Ces langages ont uniquement besoin d'un environnement de système d'exploitation dans lequel le binaire compilé peut s'exécuter. Vous pouvez également utiliser des environnements d'exécution uniquement pour le système d'exploitation Lambda pour déployer des fichiers binaires compilés avec. NETNatif AOT et Java GraalVM natif.

  Vous devez inclure un client d'interface d'exécution. Le client de l'interface d'exécution appelle l'Utilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés pour récupérer les invocations de fonction, puis appelle votre gestionnaire de fonctions. Lambda fournit des clients d'interface d'exécution pour Go,. NETNative AOT, C++ et Rust (expérimental).

  Vous devez compiler votre binaire pour un environnement Linux et pour la même architecture de jeu d'instructions que celle que vous prévoyez d'utiliser pour la fonction (x86_64 ou arm64).
- Runtimes tiers : vous pouvez exécuter des fonctions Lambda à off-the-shelf l'aide d'environnements d'exécution tels que Bref PHP for ou Swift Runtime for AWS Lambda Swift.
- Exécutions personnalisées : vous pouvez créer votre propre environnement d'exécution pour une langue ou une version de langue pour laquelle Lambda ne fournit pas d'environnement d'exécution géré, comme Node.js 19. Pour de plus amples informations, veuillez consulter Création d'un environnement d'exécution personnalisé pour AWS Lambda. Il s'agit du cas d'utilisation le moins courant pour les environnements d'exécution uniquement basés sur le système d'exploitation.

Lambda prend en charge les environnements d'exécution uniquement basés sur le système d'exploitation suivants :

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Exécution réservée au système d'exploitation | `provided.al2023` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Exécution réservée au système d'exploitation | `provided.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

L'environnement d'exécution Amazon Linux 2023 (`provided.al2023`) offre plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que `glibc`.

L'environnement d'exécution `provided.al2023` utilise `dnf` comme gestionnaire de packages au lieu de `yum`, qui est le gestionnaire de packages par défaut dans Amazon Linux 2. Pour plus d'informations sur les différences entre `provided.al2023` et`provided.al2`, consultez Présentation du runtime Amazon Linux 2023 AWS Lambda sur le blog AWS Compute.

## Création d'un environnement d'exécution personnalisé pour AWS Lambda

Vous pouvez implémenter un AWS Lambda environnement d'exécution dans n'importe quel langage de programmation. Un environnement d'exécution est un programme qui exécute la méthode de gestionnaire d'une fonction Lambda lors de l'invocation de la fonction. Vous pouvez inclure le runtime dans le package de déploiement de votre fonction ou le distribuer dans une couche. Lorsque vous créez la fonction Lambda, choisissez un environnement d'exécution uniquement pour le système d'exploitation (la famille d'exécution `provided`).

> **Note**
>
> La création d'un environnement d'exécution personnalisé est un cas d'utilisation avancé. Si vous recherchez des informations sur la compilation vers un binaire natif ou sur

l'utilisation d'un off-the-shelf environnement d'exécution tiers, consultezQuand utiliser les environnements d'exécution uniquement pour le système d'exploitation de Lambda.

Pour une présentation détaillée du processus de déploiement d'un environnement d'exécution personnalisé, voir Tutoriel : création d'un environnement d'exécution personnalisé. Vous pouvez également explorer un environnement d'exécution personnalisé implémenté en C++ sur awslabs/aws-lambda-cpp on. GitHub

Rubriques

- Prérequis
- Implémentation du flux de réponses dans un environnement d'exécution personnalisé

## Prérequis

Les environnements d'exécution personnalisés doivent effectuer certaines tâches d'initialisation et de traitement. Une exécution exécute le code d'installation de la fonction, lit le nom du gestionnaire à partir d'une variable d'environnement et lit les événements d'invocation à partir de l'API d'exécution Lambda. L'environnement d'exécution transmet les données d'événements au gestionnaire de la fonction, et renvoie la réponse du gestionnaire à Lambda.

Tâches d'initialisation

Les tâches d'initialisation sont exécutées une seule fois par instance de la fonction pour préparer l'environnement à la gestion des invocations.

- Récupérer les paramètres – Lecture des variables d'environnement pour obtenir des détails sur la fonction et l'environnement.
  - `_HANDLER` – Emplacement du gestionnaire, issu de la configuration de la fonction. Le format standard est *`file`*`.`*`method`*, où `file` est le nom du fichier sans extension et `method` est le nom d'une méthode ou fonction qui est définie dans le fichier.
  - `LAMBDA_TASK_ROOT` – Répertoire contenant le code de la fonction.
  - `AWS_LAMBDA_RUNTIME_API` – Hôte et port de l'API d'exécution.

  Pour obtenir la liste complète des variables disponibles, consultez Variables d'environnement d'exécution définies.

- Initialiser la fonction – Charger le fichier de gestionnaire et exécuter tout code global ou statique qu'il contient. Les fonctions doivent créer des ressources statiques telles que des clients de kit SDK et des connexions de base de données une seule fois, puis les réutiliser pour plusieurs invocations.
- Gérer les erreurs : si une erreur se produit, appelez l'API erreur de l'initialisation et quittez immédiatement.

L'initialisation est comptabilisée dans le délai d'attente et la durée d'exécution facturés. Lorsqu'une exécution déclenche l'initialisation d'une nouvelle instance de votre fonction, vous pouvez voir le temps d'initialisation dans les journaux et le suivi AWS X-Ray.

Example journal

```
REPORT RequestId: f8ac1208... Init Duration: 48.26 ms   Duration: 237.17 ms   Billed
 Duration: 300 ms   Memory Size: 128 MB   Max Memory Used: 26 MB
```

Traitement des tâches

Pendant son exécution, l'environnement d'exécution utilise l'interface d'environnement d'exécution Lambda pour gérer les événements entrants et signaler des erreurs. Après avoir terminé les tâches d'initialisation, l'environnement d'exécution traite les événements entrants dans une boucle. Dans votre code d'exécution, effectuez les étapes suivantes dans l'ordre.

- Obtenir un événement – Appeler l'API d'invocation suivante pour obtenir l'événement suivant. Le corps de la réponse contient les données de l'événement. Les en-têtes de la réponse contiennent l'ID de la demande et d'autres informations.
- Propager l'en-tête de suivi – Obtenir l'en-tête de suivi X-Ray à partir de l'en-tête `Lambda-Runtime-Trace-Id` dans la réponse de l'API. Définissez la variable d'environnement `_X_AMZN_TRACE_ID` localement avec la même valeur. Le kit SDK X-Ray utilise cette valeur pour connecter les données de suivi entre les services.
- Créer un objet de contexte – Créer un objet avec des informations de contexte à partir des variables d'environnement et des en-têtes de la réponse de l'API.
- Invoquer le gestionnaire de fonctions – Transmettre l'événement et l'objet contexte au gestionnaire.
- Gérer la réponse – Appeler l'API réponse d'invocation pour afficher la réponse du gestionnaire.
- Gérer les erreurs : si une erreur se produit, appelez l'API erreur de l'invocation.
- Nettoyage : libérer les ressources inutilisées, envoyer des données à d'autres services ou accomplir des tâches supplémentaires avant de passer à l'événement suivant.

Entrypoint

Le point d'entrée d'un runtime personnalisé est un fichier exécutable nommé `bootstrap`. Le fichier d'amorçage peut être l'environnement d'exécution, ou il peut invoquer un autre fichier qui crée le runtime. Si la racine de votre package de déploiement ne contient pas de fichier nommé `bootstrap`, Lambda recherche le fichier dans les couches de la fonction. Si le fichier `bootstrap` n'existe pas ou n'est pas exécutable, votre fonction renvoie une erreur `Runtime.InvalidEntrypoint` au moment de l'invocation.

Voici un exemple de `bootstrap` fichier qui utilise une version groupée de Node.js pour exécuter un JavaScript environnement d'exécution dans un fichier distinct nommé`runtime.js`.

Example amorçage

```
#!/bin/sh
    cd $LAMBDA_TASK_ROOT
    ./node-v11.1.0-linux-x64/bin/node runtime.js
```

## Implémentation du flux de réponses dans un environnement d'exécution personnalisé

Pour les fonctions de streaming de réponses, les points de terminaison `response` et `error` ont un comportement légèrement modifié qui permet à l'exécution de diffuser des réponses partielles au client et de renvoyer les charges utiles par morceaux. Pour plus d'informations sur le comportement spécifique, consultez ce qui suit :

- `/runtime/invocation/AwsRequestId/response` : propage l'en-tête `Content-Type` de l'environnement d'exécution pour l'envoyer au client. Lambda renvoie la charge utile de la réponse en morceaux via l'encodage de transfert en bloc HTTP/1.1. Le flux de réponse peut avoir une taille maximale de 20 Mio. Pour envoyer le flux de réponse à Lambda, l'environnement d'exécution doit :
  - Définir l'en-tête HTTP `Lambda-Runtime-Function-Response-Mode` sur `streaming`.
  - Définissez l'en-tête `Transfer-Encoding` sur `chunked`.
  - Écrire la réponse conformément à la spécification d'encodage de transfert en bloc HTTP/1.1.
  - Fermer la connexion sous-jacente après avoir écrit la réponse avec succès.
- `/runtime/invocation/AwsRequestId/error` : l'exécution peut utiliser ce point de terminaison pour signaler des erreurs de fonction ou d'exécution à Lambda, qui accepte également l'en-tête `Transfer-Encoding`. Ce point de terminaison ne peut être appelé qu'avant que l'environnement d'exécution ne commence à envoyer une réponse à l'invocation.

- Signaler les erreurs en cours de diffusion à l'aide des en-têtes de suivi d'erreur dans `/runtime/invocation/AwsRequestId/response`. Pour signaler les erreurs qui se produisent après que l'exécution commence à écrire la réponse à l'invocation, l'exécution peut optionnellement attacher des en-têtes de fin HTTP nommés `Lambda-Runtime-Function-Error-Type` et `Lambda-Runtime-Function-Error-Body`. Lambda considère cette réponse comme une réponse réussie et transmet les métadonnées d'erreur que l'exécution fournit au client.

> **Note**
>
> Pour joindre des en-têtes de fin, l'environnement d'exécution doit définir la valeur de l'en-tête `Trailer` au début de la demande HTTP. C'est une exigence de la spécification de l'encodage de transfert en bloc HTTP/1.1.

  - `Lambda-Runtime-Function-Error-Type` : le type d'erreur que l'exécution a rencontré. Cet en-tête se compose d'une valeur de chaîne. Lambda accepte n'importe quelle chaîne, mais nous recommandons le format *`<category.reason>`*. Par exemple, `Runtime.APIKeyNotFound`.
  - `Lambda-Runtime-Function-Error-Body` : informations encodées en Base64 sur l'erreur.

## Tutoriel : création d'un environnement d'exécution personnalisé

Dans ce didacticiel, vous allez créer une fonction Lambda avec un runtime personnalisé. Vous commencez par inclure le runtime dans le package de déploiement de la fonction. Ensuite, vous le migrez vers une couche que vous gérez indépendamment de la fonction. Enfin, vous partagez la couche du runtime en mettant à jour sa stratégie d'autorisations basée sur les ressources.

### Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. CLILes exemples de commandes présentés dans ce guide utilisent le formatage Linux. Les commandes qui incluent JSON des documents intégrés doivent être reformatées si vous utilisez Windows. CLI

Vous avez besoin d'un IAM rôle pour créer une fonction Lambda. Le rôle doit être autorisé à envoyer des CloudWatch journaux à Logs et à accéder aux AWS services utilisés par votre fonction. Si vous ne possédez pas de rôle pour le développement de fonction, créez-en un.

Pour créer un rôle d'exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :

   - Entité de confiance – Lambda.
   - Autorisations — AWSLambdaBasicExecutionRole.
   - Nom de rôle – **`lambda-role`**.

   La AWSLambdaBasicExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux dans Logs.

## Créer une fonction

Créez une fonction Lambda avec un runtime personnalisé. Cet exemple comprend deux fichiers : un fichier `bootstrap` d'exécution et un gestionnaire de fonctions. Tous deux sont mis en œuvre en Bash.

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Créez un nouveau fichier appelé `bootstrap`. Il s'agit de l'environnement d'exécution personnalisé.

   Example amorçage

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event. The HTTP request will block until one is received
  EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

  # Extract request ID by scraping response headers received above
  REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
done
```

Le runtime charge un script de fonction à partir du package de déploiement. Il utilise deux variables pour localiser le script. `LAMBDA_TASK_ROOT` lui indique où le package a été extrait et `_HANDLER` inclut le nom du script.

Une fois que le moteur d'exécution charge le script de fonction, il l'utilise API pour récupérer un événement d'invocation depuis Lambda, le transmet au gestionnaire et renvoie la réponse à Lambda. Pour obtenir l'ID de demande, le moteur d'exécution enregistre les en-têtes de la API réponse dans un fichier temporaire et lit l'`Lambda-Runtime-Aws-Request-Id`en-tête du fichier.

> **Note**
>
> Les runtimes ont d'autres responsabilités, notamment la gestion des erreurs et la fourniture d'informations de contexte au gestionnaire. Pour plus de détails, consultez Prérequis.

3. Créez un script pour la fonction. L'exemple de script suivant définit une fonction de gestionnaire qui accepte les données des événements, la consigne dans `stderr`, puis la renvoie.

   Example function.sh

   ```
   function handler () {
     EVENT_DATA=$1
     echo "$EVENT_DATA" 1>&2;
     RESPONSE="Echoing request: '$EVENT_DATA'"

     echo $RESPONSE
   }
   ```

   Le répertoire `runtime-tutorial` devrait maintenant ressembler à ceci :

   ```
   runtime-tutorial
   # bootstrap
   # function.sh
   ```

4. Rendez les fichiers exécutables et ajoutez-les dans une archive ZIP. Vous obtiendrez alors le package de déploiement.

   ```
   chmod 755 function.sh bootstrap
   ```

```
zip function.zip function.sh bootstrap
```

5. Créez une fonction nommée `bash-runtime`. Pour`--role`, entrez le nom ARN de votre rôle d'exécution Lambda.

```
aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime
 provided.al2023 \
--role arn:aws:iam::123456789012:role/lambda-role
```

6. Invoquer la fonction.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

   L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

   Vous devriez obtenir une réponse comme celle-ci :

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

7. Vérifiez la réponse.

```
cat response.txt
```

   Vous devriez obtenir une réponse comme celle-ci :

```
Echoing request: '{"text":"Hello"}'
```

## Créer une couche

Pour séparer le code du runtime du code de la fonction, créez une couche qui contient uniquement le runtime. Les couches vous permettent de développer les dépendances de votre fonction de manière indépendante et peuvent réduire l'utilisation du stockage lorsque vous utilisez la même couche avec plusieurs fonctions. Pour de plus amples informations, veuillez consulter Gestion des dépendances Lambda à l'aide de couches.

1. Créez un fichier .zip contient le fichier `bootstrap`.

   ```
   zip runtime.zip bootstrap
   ```

2. Créez une couche à l'aide de la publish-layer-versioncommande.

   ```
   aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
   runtime.zip
   ```

   Cela crée la première version de la couche.

## Mettre à jour la fonction

Pour utiliser la couche d'exécution dans la fonction, configurez la fonction pour utiliser la couche et supprimez le code de l'environnement d'exécution de la fonction.

1. Mettez à jour la configuration de la fonction pour extraire la couche.

   ```
   aws lambda update-function-configuration --function-name bash-runtime \
   --layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
   ```

   Cela ajoute le runtime à la fonction dans le répertoire `/opt`. Pour garantir que Lambda utilise l'environnement d'exécution dans la couche, vous devez supprimer `boostrap` du package de déploiement de la fonction, comme indiqué dans les deux étapes suivantes.

2. Créez un fichier .zip reprenant le code de la fonction.

   ```
   zip function-only.zip function.sh
   ```

3. Mettez à jour le code de la fonction de façon à inclure uniquement le script du gestionnaire.

```
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://
function-only.zip
```

4. Invoquez la fonction pour confirmer qu'elle fonctionne avec la couche de l'environnement d'exécution.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez obtenir une réponse comme celle-ci :

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

5. Vérifiez la réponse.

```
cat response.txt
```

Vous devriez obtenir une réponse comme celle-ci :

```
Echoing request: '{"text":"Hello"}'
```

## Mettre à jour le runtime

1. Pour enregistrer des informations sur l'environnement d'exécution, mettez à jour le script du runtime pour générer les variables d'environnement.

Example amorçage

```
#!/bin/sh
```

```
set -euo pipefail

# Configure runtime to output environment variables
echo "##  Environment variables:"
env

# Load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event. The HTTP request will block until one is received
  EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

  # Extract request ID by scraping response headers received above
  REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
done
```

2. Créez un fichier .zip reprenant la nouvelle version du fichier `bootstrap`.

```
zip runtime.zip bootstrap
```

3. Créer une nouvelle version de la couche `bash-runtime`.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

4. Configurez la fonction pour utiliser la nouvelle version de la couche.

```
aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

## Partager la couche

Pour partager une couche avec une autre Compte AWS, ajoutez une déclaration d'autorisations entre comptes à la politique basée sur les ressources de la couche. Exécutez la add-layer-version-permissioncommande et spécifiez l'ID du compte sous la formeprincipal. Dans chaque relevé, vous pouvez autoriser un seul compte, tous les comptes ou une organisation dans AWS Organizations.

L'exemple suivant autorise le compte 111122223333 à accéder à la version 2 de la couche bash-runtime.

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Vous devez voir des résultats similaires à ce qui suit :

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Les autorisations ne s'appliquent qu'à une seule version de couche. Répétez le processus chaque fois que vous créez une nouvelle version de la couche.

## Nettoyage

Supprimez chaque version de la couche.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Étant donné que la fonction contient une référence à la version 2 de la couche, elle existe toujours dans Lambda. La fonction continue de fonctionner, mais les fonctions ne peuvent plus être configurées pour utiliser la version supprimée. Si vous modifiez la liste des couches sur la fonction, vous devez spécifier une nouvelle version ou omettre la couche supprimée.

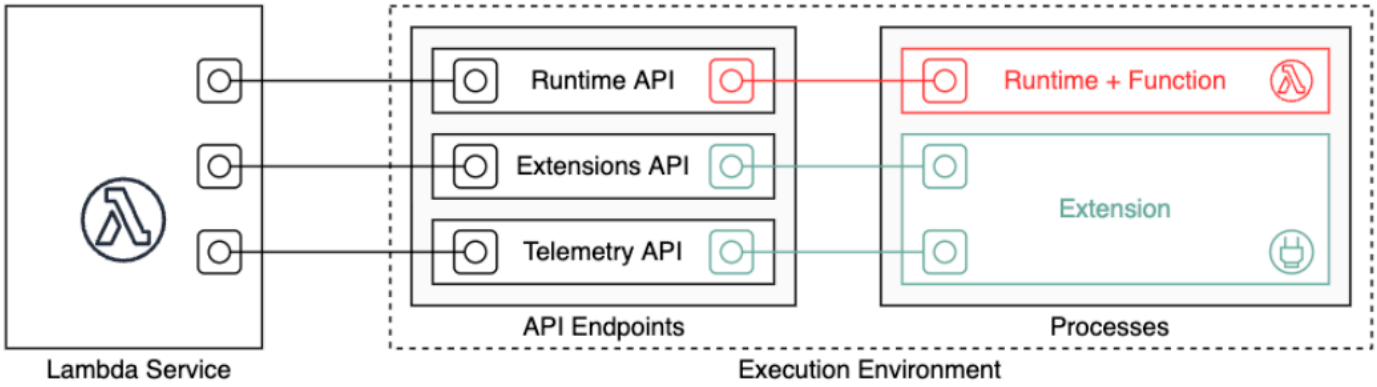
Supprimez la fonction à l'aide de la commande delete-function.

```
aws lambda delete-function --function-name bash-runtime
```

# Comprendre le cycle de vie de l'environnement d'exécution Lambda

Lambda invoque votre fonction dans un environnement d'exécution, qui fournit un environnement d'exécution sécurisé et isolé. L'environnement d'exécution gère les ressources nécessaires à l'exécution de votre fonction. L'environnement d'exécution prend également en charge le cycle de vie pour l'exécution de la fonction et pour toutes les extensions externes associées à votre fonction.

Le runtime de la fonction communique avec Lambda à l'aide du Runtime. API Les extensions communiquent avec Lambda à l'aide des extensions. API Les extensions peuvent également recevoir des messages de journal et d'autres données télémétriques de la fonction à l'aide de la télémétrie. API



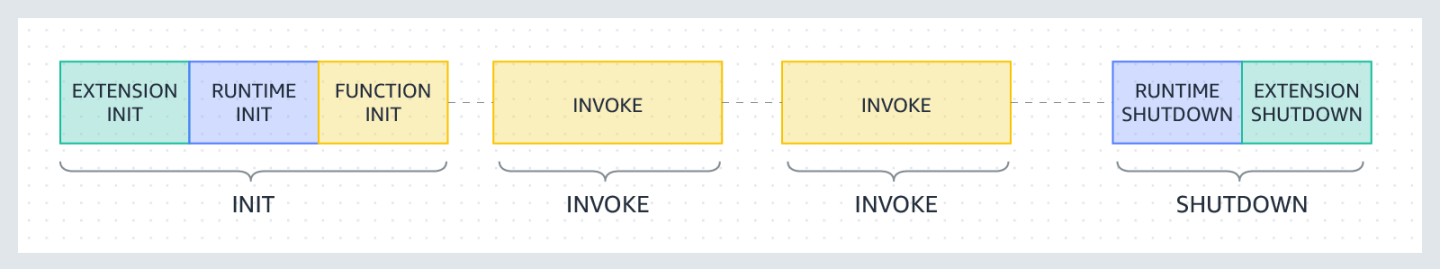

Lorsque vous créez votre fonction Lambda, vous précisez des informations de configuration, telles que la quantité de mémoire disponible et le temps d'exécution maximum autorisé pour votre fonction. Lambda utilise ces informations pour configurer l'environnement d'exécution.

L'exécution de la fonction et chaque extension externe sont des processus qui s'exécutent dans l'environnement d'exécution. Les autorisations, les ressources, les informations d'identification et les variables d'environnement sont partagées entre la fonction et les extensions.

Rubriques

- Cycle de vie d'un environnement d'exécution Lambda
- Intégrer l'apatridie dans les fonctions

# Cycle de vie d'un environnement d'exécution Lambda




Chaque phase commence par un événement que Lambda envoie à l'exécution et à toutes les extensions enregistrées. Le runtime et chaque extension indiquent l'achèvement en envoyant une `Next` API demande. Lambda gèle l'environnement d'exécution lorsque l'exécution de chaque extension est terminée, et qu'il n'y a pas d'événement en attente.

Rubriques


- Phase d'initialisation
- Échecs pendant la phase d'initialisation
- Phase de restauration (Lambda uniquement SnapStart )
- Phase d'invocation
- Échecs pendant la phase d'invocation
- Phase d'arrêt


## Phase d'initialisation

Dans la phase `Init`, Lambda effectue trois tâches :

- Démarrage de toutes les extensions (`Extension init`)
- Amorçage de l'exécution (`Runtime init`)
- Exécution du code statique de la fonction (`Function init`)
- Exécutez tous les hooks `beforeCheckpoint` d'exécution (Lambda uniquement SnapStart )

La `Init` phase se termine lorsque le moteur d'exécution et toutes les extensions signalent qu'elles sont prêtes en envoyant une `Next` API demande. La phase `Init` est limitée à 10 secondes. Si les trois tâches ne se terminent pas dans les 10 secondes, Lambda relance la phase `Init` au moment de la première invocation de fonction avec le délai d'attente de fonction configuré.

Lorsque Lambda SnapStart est activé, la phase `Init` se produit lorsque vous publiez une version de la fonction. Lambda enregistre un instantané de l'état de la mémoire et du disque de l'environnement d'exécution initialisé, fait persister l'instantané chiffré et le met en cache pour un accès à faible latence. Si vous avez un hook d'exécution `beforeCheckpoint`, le code s'exécute à la fin de la phase `Init`.

> **Note**
>
> Le délai de 10 secondes ne s'applique pas aux fonctions qui utilisent la simultanéité provisionnée ou. SnapStart Pour la simultanéité et les SnapStart fonctions configurées, votre code d'initialisation peut s'exécuter pendant 15 minutes maximum. Le délai d'attente est de 130 secondes ou le délai d'expiration de la fonction configurée (900 secondes au maximum), la valeur la plus élevée étant retenue.

Lorsque vous utilisez la simultanéité provisionnée, Lambda initialise l'environnement d'exécution lorsque vous configurez les paramètres du PC pour une fonction. Lambda garantit également que les environnements d'exécution initialisés sont toujours disponibles avant les invocations. Vous pouvez constater des écarts entre les phases d'initialisation et d'invocation de votre fonction. En fonction de la configuration d'environnement d'exécution et de mémoire de votre fonction, vous pouvez également constater une variabilité de latence est possible lors de la première invocation dans un environnement d'exécution initialisé.

Pour les fonctions utilisant la simultanéité à la demande, Lambda peut occasionnellement initialiser les environnements d'exécution avant les demandes d'invocation. Lorsque cela se produit, vous pouvez également observer un intervalle de temps inattendu entre les phases d'initialisation et d'invocation de votre fonction. Nous vous recommandons de ne pas dépendre de ce comportement.

## Échecs pendant la phase d'initialisation

Si une fonction se bloque ou expire pendant la phase `Init`, Lambda émet des informations d'erreur dans le journal `INIT_REPORT`.

Example — INIT _ REPORT enregistre le délai d'expiration

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: timeout
```

Example — INIT _ REPORT log en cas d'échec de l'extension

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: error Error Type:
 Extension.Crash
```

Si la `Init` phase est réussie, Lambda n'émet pas le `INIT_REPORT` journal, sauf s'il est activé. SnapStart SnapStart les fonctions émettent toujours`INIT_REPORT`. Pour de plus amples informations, veuillez consulter Surveillance pour Lambda SnapStart.

## Phase de restauration (Lambda uniquement SnapStart )

Lorsque vous appelez une SnapStartfonction pour la première fois et que celle-ci évolue, Lambda reprend les nouveaux environnements d'exécution à partir de l'instantané persistant au lieu d'initialiser la fonction à partir de zéro. Si vous avez un hook d’exécution `afterRestore()`, le code s’exécute à la fin de la phase `Restore`. Vous êtes facturé pour la durée des hooks d’exécution `afterRestore()`. Le runtime (JVM) doit se charger et les hooks `afterRestore()` d'exécution doivent se terminer dans le délai imparti (10 secondes). Sinon, vous obtiendrez un SnapStartTimeoutException. Lorsque la phase `Restore` se termine, Lambda invoque le gestionnaire de fonction (Phase d’invocation).

### Échecs pendant la phase de restauration

Si la phase `Restore` échoue, Lambda émet des informations d’erreur dans le journal `RESTORE_REPORT`.

Example — RESTORE _ REPORT enregistre le délai d'expiration

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: timeout
```

Example — RESTORE _ REPORT log en cas d'échec du hook d'exécution

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: error Error Type: Runtime.ExitError
```

Pour plus d’informations sur le journal `RESTORE_REPORT`, consultez Surveillance pour Lambda SnapStart.

## Phase d’invocation

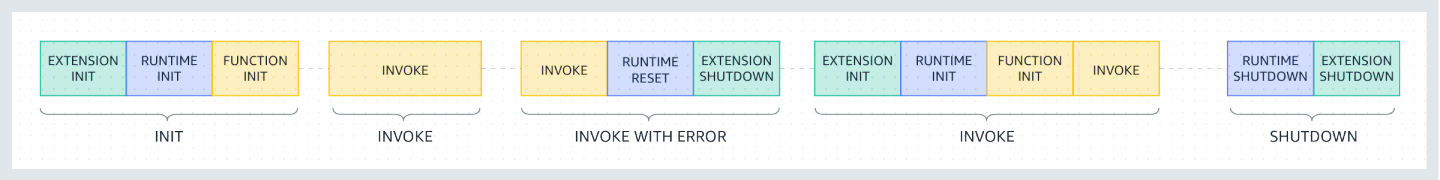
Lorsqu'une fonction Lambda est invoquée en réponse à une `Next` API demande, Lambda envoie un `Invoke` événement au moteur d'exécution et à chaque extension.

Le paramètre d'expiration de la fonction limite la durée de l'ensemble de la phase `Invoke`. Par exemple, si vous définissez le délai d'expiration de la fonction sur 360 secondes, la fonction et toutes les extensions doivent être terminées dans un délai de 360 secondes. Notez qu'il n'y a pas de phase post-invocation indépendante. La durée correspond à la somme de tous les temps d'invocation (exécution + extensions) et n'est calculée que lorsque l'exécution de la fonction et de toutes les extensions est terminée.

La phase d'appel se termine après l'exécution et toutes les extensions signalent qu'elles sont terminées en envoyant une `Next` API demande.

## Échecs pendant la phase d'invocation

Si la fonction Lambda se bloque ou expire pendant la phase `Invoke`, Lambda réinitialise l'environnement d'exécution. Le diagramme suivant illustre le comportement de l'environnement d'exécution Lambda en cas d'échec de l'invocation :




Dans le diagramme précédent :

- La première phase est la INITphase qui s'exécute sans erreur.
- La deuxième phase est la INVOKEphase qui s'exécute sans erreur.
- À un moment donné, supposons que votre fonction rencontre un échec d'invocation (tel qu'un dépassement de délai de la fonction ou une erreur d'exécution). La troisième phase, intitulée INVOKEWITHERROR, illustre ce scénario. Lorsque cela se produit, le service Lambda effectue une réinitialisation. La réinitialisation se comporte comme un événement `Shutdown`. Lambda commence par arrêter l'exécution, puis envoie un événement `Shutdown` à chaque extension externe enregistrée. L'événement comprend le motif de l'arrêt. Si cet environnement est utilisé pour une nouvelle invocation, Lambda réinitialise l'extension et l'exécution en même temps que l'invocation suivante.

  Notez que la réinitialisation Lambda n'efface pas le contenu du `/tmp` répertoire avant la phase d'initialisation suivante. Ce comportement est cohérent avec la phase d'arrêt normale.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> Si la configuration du journal système de votre fonction est définie sur du texte brut, cette modification affecte les messages de journal capturés dans CloudWatch les journaux lorsque votre fonction rencontre un échec d'appel. Les exemples suivants montrent les sorties des journaux dans les anciens et les nouveaux formats.
> Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Example CloudWatch Journalise la sortie du journal (exécution ou crash de l'extension) - ancien style

```
START RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Version: $LATEST
RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Error: Runtime exited without
 providing a reason
Runtime.ExitError
END RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1
REPORT RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Duration: 933.59 ms Billed
 Duration: 934 ms Memory Size: 128 MB Max Memory Used: 9 MB
```

Example CloudWatch Enregistre la sortie du journal (délai d'expiration de la fonction) - ancien style

```
START RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Version: $LATEST
2024-03-04T17:22:38.033Z b70435cc-261c-4438-b9b6-efe4c8f04b21 Task timed out after
 3.00 seconds
END RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21
REPORT RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Duration: 3004.92 ms Billed
 Duration: 3000 ms Memory Size: 128 MB Max Memory Used: 33 MB Init Duration: 111.23
 ms
```

Le nouveau format des CloudWatch journaux inclut un `status` champ supplémentaire dans la `REPORT` ligne. Dans le cas d'un crash d'exécution ou d'une extension, la `REPORT` ligne inclut également un champ`ErrorType`.

Example CloudWatch Journalise la sortie du journal (exécution ou crash de l'extension) - nouveau style

```
START RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Version: $LATEST
END RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd
REPORT RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Duration: 133.61 ms Billed
 Duration: 133 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 80.00
 ms Status: error Error Type: Runtime.ExitError
```

Example CloudWatch Enregistre la sortie du journal (délai d'expiration de la fonction) - nouveau style

```
START RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Version: $LATEST
END RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda
REPORT RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Duration: 3016.78 ms Billed
 Duration: 3016 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 84.00
 ms Status: timeout
```

- La quatrième phase représente la INVOKEphase qui suit immédiatement un échec d'appel. Ici, Lambda initialise à nouveau l'environnement en relançant la phase. INIT Cela s’appelle une init supprimée. Lorsque des initialisations sont supprimées, Lambda ne signale pas explicitement de phase INIT CloudWatch supplémentaire dans Logs. Au lieu de cela, vous remarquerez peut-être que la durée indiquée dans la REPORT ligne inclut une INITdurée supplémentaire + la INVOKEdurée. Supposons, par exemple, que les connexions suivantes s'affichent CloudWatch :

```
2022-12-20T01:00:00.000-08:00 START RequestId: XXX Version: $LATEST
2022-12-20T01:00:02.500-08:00 END RequestId: XXX
2022-12-20T01:00:02.500-08:00 REPORT RequestId: XXX Duration: 3022.91 ms
Billed Duration: 3000 ms Memory Size: 512 MB Max Memory Used: 157 MB
```

Dans cet exemple, la différence entre les START horodatages REPORT et est de 2,5 secondes. Cela ne correspond pas à la durée déclarée de 3022,91 millisecondes, car cela ne prend pas en compte le surcroît INIT(initialisation supprimée) effectué par Lambda. Dans cet exemple, vous pouvez déduire que la INVOKEphase réelle a duré 2,5 secondes.

Pour plus d'informations sur ce comportement, vous pouvez utiliser la Accès aux données de télémétrie en temps réel pour les extensions à l'aide de la télémétrie API. La télémétrie API émet `INIT_START` des `INIT_REPORT` événements `phase=invoke` chaque fois que `INIT_RUNTIME_DONE` des initialisations supprimées se produisent entre les phases d'appel.

- La cinquième phase représente la SHUTDOWNphase qui s'exécute sans erreur.

## Phase d'arrêt

Quand Lambda est sur le point d'arrêter l'exécution, il envoie un événement `Shutdown` à chaque extension externe enregistrée. Les extensions peuvent utiliser ce temps pour les tâches de nettoyage final. L'`Shutdown`événement est une réponse à une `Next` API demande.

Durée : l'ensemble de la phase `Shutdown` est limitée à 2 secondes. Si l'exécution ou une extension ne répondent pas, Lambda y met fin via un signal (`SIGKILL`).

Lorsque la fonction et toutes les extensions ont pris fin, Lambda conserve l'environnement d'exécution pendant un certain temps en prévision d'une autre invocation de fonction. Cependant, Lambda met fin aux environnements d'exécution toutes les quelques heures pour permettre les mises à jour et la maintenance de l'exécution, même pour les fonctions invoquées en continu. Vous ne devez pas partir du principe que l'environnement d'exécution persistera indéfiniment. Pour de plus amples informations, veuillez consulter Intégrer l'apatridie dans les fonctions.

Lorsque la fonction est invoquée à nouveau, Lambda réactive l'environnement pour la réutilisation. La réutilisation de l'environnement d'exécution a les conséquences suivantes :

- Les objets déclarés en dehors de la méthode de gestionnaire de la fonction restent initialisés, ce qui fournit une optimisation supplémentaire lorsque la fonction est invoquée à nouveau. Par exemple, si votre fonction Lambda établit une connexion de base de données, au lieu de rétablir la connexion, la connexion d'origine est utilisée dans les invocations suivantes. Nous vous recommandons d'ajouter une logique dans votre code pour vérifier s'il existe une connexion avant d'en créer une nouvelle.
- Chaque environnement d'exécution fournit entre 512 Mo et 10 240 Mo, par incréments de 1 Mo d'espace disque dans le répertoire `/tmp`. Le contenu du répertoire est conservé lorsque l'environnement d'exécution est gelé, fournissant ainsi un cache temporaire qui peut servir à plusieurs invocations. Vous pouvez ajouter du code pour vérifier si le cache contient les données que vous avez stockées. Pour de plus amples informations sur les limites de taille de déploiement, veuillez consulter Quotas Lambda.

- Les processus en arrière-plan ou les rappels qui ont été initiés par votre fonction Lambda et qui ne sont pas terminés à la fin de l’exécution de la fonction reprennent si Lambda réutilise l’environnement d’exécution. Assurez-vous que les processus d’arrière-plan ou les rappels dans votre code se terminent avant que l’exécution du code ne prenne fin.

## Intégrer l'apatridie dans les fonctions

Lorsque vous écrivez le code de votre fonction Lambda, considérez l'environnement d'exécution comme étant apatride, en supposant qu'il n'existe que pour un seul appel. Lambda met fin aux environnements d'exécution toutes les quelques heures pour permettre les mises à jour et la maintenance de l'exécution, même pour les fonctions invoquées en continu. Initialisez tout état requis (par exemple, récupérer un panier d'achat depuis une table Amazon DynamoDB) lorsque votre fonction démarre. Avant de quitter, apportez des modifications permanentes aux données dans des magasins durables tels qu'Amazon Simple Storage Service (Amazon S3), DynamoDB ou Amazon Simple Queue Service (Amazon). SQS Évitez de vous fier à des structures de données existantes, à des fichiers temporaires ou à des états couvrant plusieurs invocations, tels que des compteurs ou des agrégats. Cela garantit que votre fonction gère chaque appel indépendamment.

# Configuration des AWS Lambda fonctions

Découvrez comment configurer les fonctionnalités et options principales de votre fonction Lambda à l'aide de la Lambda ou de la console. API

Mémoire

Découvrez comment et quand augmenter la mémoire des fonctions.

Stockage éphémère

Découvrez comment et quand augmenter la capacité de stockage temporaire de votre fonction.

Expiration

Découvrez comment et quand augmenter le délai d'expiration de votre fonction.

Variables d'environnement

Vous pouvez rendre le code de votre fonction portable et garder les secrets de votre code en les stockant dans la configuration de votre fonction à l'aide de variables d'environnement.

Réseaux sortants

Vous pouvez utiliser votre fonction Lambda avec les AWS ressources d'un Amazon. VPC La connexion de votre fonction à un vous VPC permet d'accéder aux ressources d'un sous-réseau privé, telles que les bases de données relationnelles et les caches.

Réseaux entrants

Vous pouvez utiliser un point de VPC terminaison d'interface pour appeler vos fonctions Lambda sans passer par l'Internet public.

Système de fichiers

Vous pouvez utiliser votre fonction Lambda pour monter un Amazon dans un EFS répertoire local. Un système de fichiers permet à votre code de fonction d'accéder à des ressources partagées et de les modifier en toute sécurité et avec une simultanéité élevée.

Alias

Vous pouvez configurer vos clients pour qu'ils invoquent une version spécifique de la fonction Lambda en utilisant un alias, au lieu de mettre à jour le client.

### Versions

En publiant une version de votre fonction, vous pouvez stocker votre code et votre configuration comme une ressource séparée qui ne peut pas être modifiée.

### Balises

Utilisez des balises pour activer le contrôle d'accès basé sur les attributs (ABAC), pour organiser vos fonctions Lambda et pour filtrer et générer des rapports sur vos fonctions à l'aide des services AWS Cost Explorer ou AWS Billing and Cost Management.

### Streaming des réponses

Vous pouvez configurer votre fonction Lambda URLs pour transmettre les charges utiles de réponse aux clients. Le streaming des réponses peut profiter aux applications sensibles à la latence en améliorant les performances du délai jusqu'au premier octet (TTFB). En effet, vous pouvez renvoyer des réponses partielles au client dès qu'elles sont disponibles. En outre, vous pouvez utiliser le streaming de réponses pour créer des fonctions qui renvoient des charges utiles plus importantes.

# Déploiement des fonctions Lambda comme des archives de fichiers .zip

Lorsque vous créez une fonction Lambda, vous empaquetez le code de votre fonction dans un package de déploiement. Lambda prend en charge deux types de packages de déploiement : les images conteneurs et les archives de fichier .zip. Le flux de travail de création d'une fonction dépend du type de package de déploiement. Pour configurer une fonction définie en tant qu'image de conteneur, consultez the section called “Images de conteneur”.

Vous pouvez utiliser la console Lambda et le Lambda API pour créer une fonction définie avec une archive de fichier .zip. Vous pouvez également charger un fichier .zip mis à jour pour modifier le code de la fonction.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Rubriques


- Création de la fonction
- Utilisation de l'éditeur de code de la console
- Mise à jour du code de fonction
- Modification de l'environnement d'exécution
- Modification de l'architecture
- Utilisation du Lambda API
- AWS CloudFormation


## Création de la fonction

Lorsque vous créez une fonction définie avec une archive de fichiers .zip, vous choisissez un modèle de code, la version du langage et le rôle d'exécution pour la fonction. Vous ajoutez votre code de fonction après que Lambda a créé la fonction.

Pour créer la fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Choisissez Créer à partir de zéro ou Utiliser un plan pour créer votre fonction.
4. Sous Informations de base, procédez comme suit :
   a. Pour Function name (Nom de la fonction), saisissez le nom de la fonction. Les noms de fonctions sont limités à 64 caractères.
   b. Pour Runtime, sélectionnez la version du langage à utiliser pour votre fonction.
   c. (Facultatif) Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour utiliser la fonction. L'architecture par défaut est x86_64. Lorsque vous créez le package de déploiement de la fonction, assurez-vous qu'il est compatible avec cette architecture de l'ensemble des instructions.
5. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou utiliser un rôle existant.
6. (Facultatif) Développez Advanced settings (Paramètres avancés). Vous pouvez choisir une Configuration de signature de code pour la fonction. Vous pouvez également configurer un (AmazonVPC) pour que la fonction puisse accéder.
7. Sélectionnez Créer une fonction.

Lambda crée la nouvelle fonction. Vous pouvez désormais utiliser la console pour ajouter le code de fonction et configurer d'autres paramètres et caractéristiques de la fonction. Pour obtenir des instructions de déploiement de code, consultez la page du gestionnaire pour l'exécution utilisée par votre fonction.

Node.js

Déployer des fonctions Lambda en Node.js avec des archives de fichiers .zip

Python

Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python

Ruby

Déployer des fonctions Lambda en Ruby avec des archives de fichiers .zip

Java

- Déployez des fonctions Java Lambda avec des archives .zip ou de fichiers JAR

Go

- Déployer des fonctions Lambda Go avec des archives de fichiers .zip

C#

- Créez et déployez des fonctions Lambda C# à l'aide des archives de fichiers .zip

PowerShell

- Déployer des fonctions PowerShell Lambda avec des archives de fichiers .zip

## Utilisation de l'éditeur de code de la console

La console crée une fonction Lambda avec un seul fichier source. Pour le scripting, vous pouvez modifier ce fichier et ajouter des fichiers à l'aide de l'éditeur de code intégré. Choisissez Save pour enregistrer les changements. Ensuite, pour exécuter votre code, choisissez Test.

Lorsque vous enregistrez votre code de fonction, la console Lambda crée un package de déploiement d'archive de fichiers .zip. Lorsque vous développez votre code de fonction en dehors de la console (à l'aide d'unIDE), vous devez créer un package de déploiement pour télécharger votre code dans la fonction Lambda.

## Mise à jour du code de fonction

Pour le scripting (Node.js, Python et Ruby), vous pouvez modifier votre code de fonction dans l'éditeur intégré. Si ce code est d'une taille supérieure à 3 Mo, si vous devez ajouter des bibliothèques ou dans le cas des langages que l'éditeur ne prend pas en charge (Java, Go, C#), vous devez charger le code de votre fonction en tant qu'archive .zip. Si l'archive de fichier .zip est d'une taille inférieure à 50 Mo, vous pouvez charger l'archive de fichier .zip à partir de votre machine locale. Si le fichier est d'une taille supérieure à 50 Mo, chargez le fichier dans la fonction à partir d'un compartiment Amazon S3.

Pour charger le code de fonction en tant qu'archive .zip

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction à mettre à jour, puis choisissez l'onglet Code.

3. Sous Code source (Source du code), sélectionnez Upload from (Charger depuis).
4. Choisissez .zip file (fichier .zip), puis Upload (Charger).
   - Dans le sélecteur de fichiers, sélectionnez la nouvelle version de l'image et choisissez Open (Ouvrir), puis Save (Enregistrer).
5. (Alternative à l'étape 4) Choisissez Emplacement Amazon S3.
   - Dans la zone de texte, entrez le lien S3 URL de l'archive de fichiers .zip, puis choisissez Enregistrer.

## Modification de l'environnement d'exécution

Si vous mettez à jour la configuration de la fonction pour utiliser un nouvel environnement d'exécution, vous devrez peut-être mettre à jour le code de la fonction pour qu'il soit compatible avec la nouvelle version. Si vous mettez à jour la configuration de la fonction pour utiliser une autre exécution, vous devez fournir un nouveau code de fonction compatible avec l'exécution et l'architecture. Pour obtenir des instructions sur comment créer un package de déploiement pour le code de la fonction, consultez la page du gestionnaire de l'exécution utilisé par la fonction.

Le Node.js 20, Python 3.12, Java 21,. NETLes images de base 8, Ruby 3.3 et versions ultérieures sont basées sur l'image de conteneur minimale Amazon Linux 2023. Les images de base antérieures utilisaient Amazon Linux 2. AL2Le 023 offre plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que`glibc`. Pour plus d'informations, consultez Présentation de l'environnement d'exécution Amazon Linux 2023 pour AWS Lambda sur le blog AWS Compute.

Pour modifier l'environnement d'exécution

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction à mettre à jour, puis choisissez l'onglet Code.
3. Faites défiler jusqu'à la section Runtime settings (Paramètres d'exécution), qui se trouve sous l'éditeur de code.
4. Choisissez Modifier.
   a. Pour Runtime (Environnement d'exécution), sélectionnez l'identifiant de l'environnement d'exécution.
   b. Pour Gestionnaire, spécifiez le nom de fichier et gestionnaire de votre fonction.

c. Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour utiliser votre fonction.

5. Choisissez Save (Enregistrer).

## Modification de l'architecture

Avant de pouvoir modifier l'architecture de l'ensemble des instructions, vous devez vous assurer que le code de la fonction est compatible avec l'architecture cible.

Si vous utilisez Node.js, Python ou Ruby et que vous modifiez le code de fonction dans l'éditeur, le code existant peut s'exécuter sans modification.

Toutefois, si vous fournissez le code de fonction en utilisant un package de déploiement d'archives de fichiers .zip, vous devez préparer une nouvelle archive de fichiers .zip compilée et créée correctement pour l'exécution cible et l'architecture de l'ensemble des instructions. Pour obtenir des instructions, veuillez consulter la page du gestionnaire d'exécution de la fonction.

Pour modifier l'architecture de l'ensemble des instructions

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction à mettre à jour, puis choisissez l'onglet Code.
3. Sous Paramètres du runtime, choisissez Modifier.
4. Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour utiliser votre fonction.
5. Choisissez Save (Enregistrer).

## Utilisation du Lambda API

Pour créer et configurer une fonction qui utilise une archive de fichier .zip, effectuez les API opérations suivantes :

- CreateFunction
- UpdateFunctionCode
- UpdateFunctionConfiguration

# AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une fonction Lambda qui utilise une archive de fichiers .zip. Dans votre AWS CloudFormation modèle, la `AWS::Lambda::Function` ressource spécifie la fonction Lambda. Pour une description des propriétés de la *`AWS::Lambda::Function`* ressource, voir AWS: :Lambda : :Function dans le guide de l'AWS CloudFormation utilisateur.

Dans la ressource `AWS::Lambda::Function`, définissez les propriétés suivantes pour créer une fonction définie en tant qu'archive de fichiers .zip :

- AWS: :Lambda : :Function
  - PackageType — Réglé sur`Zip`.
  - Code – Saisissez le nom du compartiment Amazon S3 et le nom du fichier .zip dans les champs `S3Bucket` et `S3Key`. Pour Node.js ou Python, vous pouvez fournir le code source inclus de votre fonction Lambda.
  - Runtime – Définissez la valeur du runtime.
  - Architecture — Définissez la valeur de l'architecture `arm64` pour utiliser le processeur AWS Graviton2. La valeur de l'architecture par défaut est `x86_64`.

# Création d'une fonction Lambda à l'aide d'une image de conteneur

Le code de votre AWS Lambda fonction se compose de scripts ou de programmes compilés et de leurs dépendances. Pour déployer votre code de fonction vers Lambda, vous utilisez un package de déploiement. Lambda prend en charge deux types de packages de déploiement : les images conteneurs et les archives de fichiers .zip.

Il existe trois méthodes pour créer une image de conteneur pour une fonction Lambda :

- Utilisation d'une image AWS de base pour Lambda

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.
- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure un client d'interface d'exécution pour votre langage dans l'image.
- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure un client d'interface d'exécution pour votre langage dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Pour créer une fonction Lambda à partir d'une image de conteneur, créez votre image localement et chargez-la dans un référentiel Amazon Elastic Container Registry ECR (Amazon). Spécifiez ensuite le référentiel URI lorsque vous créez la fonction. Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda. Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

> **Note**
>
> Lambda ne prend pas en charge les ECR FIPS points de terminaison Amazon pour les images de conteneurs. Si votre référentiel URI inclut`ecr-fips`, vous utilisez un FIPS point de terminaison. Exemple: `111122223333.dkr.ecr-fips.us-east-1.amazonaws.com`.

Cette page explique les types d'images de base et les exigences requises pour créer des images de conteneur compatibles avec Lambda.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Rubriques


- Prérequis
- Utilisation d'une image AWS de base pour Lambda
- Utilisation d'une image de base AWS uniquement pour le système d'exploitation
- Utilisation d'une image non AWS basique
- Clients d'interface d'exécution
- ECRAutorisations Amazon
- Cycle de vie des fonctions

## Prérequis

Installez la version 2 AWS Command Line Interface (AWS CLI) et le Docker CLI. En outre, notez les exigences suivantes :

- L'image de conteneur doit implémenter l'Utilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés. Les clients de l'interface d'exécution AWS open source implémentent leAPI. Vous pouvez ajouter un client d'interface de runtime à votre image de base préférée pour la rendre compatible avec Lambda.
- L'image de conteneur doit pouvoir s'exécuter sur un système de fichiers en lecture seule. Votre code de fonction peut accéder à un répertoire `/tmp` inscriptible entre 512 Mo et 10 240 Mo de stockage, par incréments de 1 Mo.
- L'utilisateur Lambda par défaut doit pouvoir lire tous les fichiers requis pour exécuter votre code de fonction. Lambda suit les bonnes pratiques de sécurité en définissant un utilisateur Linux par défaut avec des autorisations assortie d'un privilège minimum. Cela signifie que vous n'avez pas besoin de spécifier un USERdans votre Dockerfile. Vérifiez que votre code d'application ne repose pas sur des fichiers dont l'exécution n'est pas autorisée à d'autres utilisateurs Linux.
- Lambda prend uniquement en charge les images de conteneur basées sur Linux.
- Lambda fournit des images de base multi-architecture. Toutefois, l'image que vous créez pour la fonction ne doit avoir que l'une des architectures pour cible. Lambda ne prend pas en charge les fonctions qui utilisent les images de conteneur multi-architecture.

## Utilisation d'une image AWS de base pour Lambda

Vous pouvez utiliser l'une des images de base AWS pour Lambda afin de créer l'image de conteneur pour votre code de fonction. Les images de base sont préchargées avec une exécution de langage et d'autres composants requis pour exécuter une image conteneur sur Lambda. Vous ajoutez votre code de fonction et vos dépendances à l'image de base, puis vous l'empaquetez en tant qu'image de conteneur.

AWS fournit régulièrement des mises à jour des images AWS de base pour Lambda. Si votre Dockerfile inclut le nom de l'image dans la FROM propriété, votre client Docker extrait la dernière version de l'image du référentiel Amazon. ECR Afin d'utiliser l'image de base mise à jour, vous devez reconstruire votre image de conteneur et mettre à jour le code de la fonction.

Le Node.js 20, Python 3.12, Java 21,. NETLes images de base 8, Ruby 3.3 et versions ultérieures sont basées sur l'image de conteneur minimale Amazon Linux 2023. Les images de base antérieures

utilisaient Amazon Linux 2. AL2Le 023 offre plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que`glibc`.

AL2Les images basées sur le format 023 utilisent `microdnf` (lien symbolique sous forme`dnf`) comme gestionnaire de packages au lieu de`yum`, qui est le gestionnaire de packages par défaut dans Amazon Linux 2. `microdnf`est une implémentation autonome de. `dnf` Pour obtenir la liste des packages inclus dans les images AL2 basées sur la version 023, reportez-vous aux colonnes Conteneur minimal de la section Comparaison des packages installés sur les images de conteneurs Amazon Linux 2023. Pour plus d'informations sur les différences entre AL2 023 et Amazon Linux 2, consultez Présentation du runtime Amazon Linux 2023 AWS Lambda sur le blog AWS Compute.

> **Note**
>
> Pour exécuter des images AL2 basées sur 023 localement, y compris avec AWS Serverless Application Model (AWS SAM), vous devez utiliser Docker version 20.10.10 ou ultérieure.

Pour créer une image de conteneur à l'aide d'une image de AWS base, choisissez les instructions correspondant à votre langue préférée :

- Node.js
- TypeScript(utilise une image de base Node.js)
- Python
- Java
- Go
- .NET
- Ruby

## Utilisation d'une image de base AWS uniquement pour le système d'exploitation

AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous

pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure un client d'interface d'exécution pour votre langage dans l'image.

| Balises | Environnement d'exécution | Système d'exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| al2023 | Exécution uniquement basée sur le système d'exploitation | Amazon Linux | Dockerfile pour OS uniquement Runtime sur GitHub | Non prévu |
| al2 | Exécution uniquement basée sur le système d'exploitation | Amazon Linux 2 | Dockerfile pour OS uniquement Runtime sur GitHub | Non prévu |

Galerie publique Amazon Elastic Container Registry : gallery.ecr.aws/lambda/fourni

## Utilisation d'une image non AWS basique

Lambda prend en charge toute image conforme à l'un des formats de manifeste d'image suivants :

- Manifeste d'images Docker V2, schéma 2 (utilisé avec la version 1.10 de Docker et les versions les plus récentes)
- Spécifications de l'Open Container Initiative (OCI) (v1.0.0 et versions ultérieures)

Lambda prend en charge une taille maximale d'image non compressée de 10 Go, comprenant toutes les couches.

> **Note**
>
> Pour rendre l'image compatible avec Lambda, vous devez inclure un client d'interface d'exécution pour votre langage dans l'image.

## Clients d'interface d'exécution

Si vous utilisez une image de base uniquement pour le système d'exploitation ou une autre image de base, vous devez inclure un client d'interface d'exécution dans votre image. Le client de l'interface d'exécution doit étendre leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l'interaction entre Lambda et votre code de fonction. AWS fournit des clients d'interface d'exécution open source pour les langages suivants :

- Node.js
- Python
- Java
- .NET
- Go
- Ruby
- Rust – Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Si vous utilisez un langage qui ne possède pas de client d'interface d'exécution AWS fourni, vous devez créer le vôtre.

## ECRAutorisations Amazon

Avant de créer une fonction Lambda à partir d'une image de conteneur, vous devez créer l'image localement et la télécharger dans un référentiel AmazonECR. Lorsque vous créez la fonction, spécifiez le ECR référentiel AmazonURI.

Assurez-vous que les autorisations accordées à l'utilisateur ou au rôle qui crée la fonction incluent `GetRepositoryPolicy` et`SetRepositoryPolicy`.

Par exemple, utilisez la IAM console pour créer un rôle avec la politique suivante :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
  ]
}
```

## Politiques relatives aux ECR référentiels Amazon

Pour une fonction associée au même compte que l'image du conteneur dans AmazonECR, vous pouvez ajouter `ecr:BatchGetImage` des `ecr:GetDownloadUrlForLayer` autorisations à votre politique de ECR dépôt Amazon. L'exemple suivant présente la stratégie minimum :

```
{
        "Sid": "LambdaECRImageRetrievalPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": [
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ]
    }
```

Pour plus d'informations sur les autorisations relatives aux ECR référentiels Amazon, consultez les politiques relatives aux référentiels privés dans le guide de l'utilisateur d'Amazon Elastic Container Registry.

Si le ECR référentiel Amazon n'inclut pas ces autorisations, Lambda ajoute `ecr:BatchGetImage` des autorisations `ecr:GetDownloadUrlForLayer` au référentiel d'images du conteneur. Lambda ne peut ajouter ces autorisations que si le principal appelant Lambda dispose des autorisations `ecr:getRepositoryPolicy` et `ecr:setRepositoryPolicy`.

Pour consulter ou modifier les autorisations de votre ECR référentiel Amazon, suivez les instructions de la section Définition d'une déclaration de politique de dépôt privé dans le guide de l'utilisateur d'Amazon Elastic Container Registry.

Autorisations ECR multi-comptes Amazon

Un autre compte dans la même région peut créer une fonction qui utilise une image conteneur appartenant à votre compte. Dans l'exemple suivant, votre politique d'autorisation de ECR dépôt Amazon nécessite les instructions suivantes pour autoriser l'accès au numéro de compte 123456789012.

- CrossAccountPermission— Permet au compte 123456789012 de créer et de mettre à jour des fonctions Lambda qui utilisent des images de ce référentiel. ECR
- L ambdaECRImage CrossAccountRetrievalPolicy — Lambda finira par définir l'état d'une fonction sur inactif si elle n'est pas invoquée pendant une période prolongée. Cette instruction est requise pour que Lambda puisse récupérer l'image conteneur à des fins d'optimisation et de mise en cache pour le compte de la fonction détenue par 123456789012.

Example – Ajouter une autorisation entre comptes à votre référentiel

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPermission",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      }
    },
    {
      "Sid": "LambdaECRImageCrossAccountRetrievalPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
```

```
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "StringLike": {
          "aws:sourceARN": "arn:aws:lambda:us-east-1:123456789012:function:*"
        }
      }
    }
  ]
}
```

Pour donner accès à plusieurs comptes, vous ajoutez le compte IDs à la liste Principal de la `CrossAccountPermission` politique et à la liste d'évaluation des conditions dans le`LambdaECRImageCrossAccountRetrievalPolicy`.

Si vous travaillez avec plusieurs comptes au sein d'une AWS organisation, nous vous recommandons d'énumérer chaque ID de compte dans la politique ECR d'autorisation. Cette approche est conforme aux meilleures pratiques de AWS sécurité qui consistent à définir des autorisations restreintes dans les IAM politiques.

Outre les autorisations Lambda, l'utilisateur ou le rôle qui crée la fonction doit également disposer `BatchGetImage` d'`GetDownloadUrlForLayer`autorisations.

## Cycle de vie des fonctions

Après avoir chargé une nouvelle image de conteneur ou une image de conteneur mise à jour, Lambda optimise l'image avant que la fonction ne puisse traiter les appels. Le processus d'optimisation peut prendre quelques secondes. La fonction reste à l'état `Pending` jusqu'à la fin du processus. La fonction passe ensuite à l'état `Active`. Lorsque la fonction est à l'état `Pending`, vous pouvez l'appeler, mais les autres opérations sur la fonction échouent. Les appels survenant lors de la mise à jour d'une image exécutent le code de l'image précédente.

Si une fonction n'est pas appelée pendant plusieurs semaines, Lambda récupère sa version optimisée, et la fonction passe à l'état `Inactive`. Pour réactiver la fonction, vous devez l'appeler. Lambda rejette la première invocation et la fonction passe à l'état `Pending` jusqu'à ce que Lambda réoptimise l'image. La fonction revient ensuite à l'état `Active`.

Lambda récupère régulièrement l'image du conteneur associée dans le référentiel Amazon. ECR Si l'image de conteneur correspondante n'existe plus sur Amazon ECR ou si les autorisations sont

révoquées, la fonction passe à l'`Failed`état et Lambda renvoie un échec pour toute invocation de fonction.

Vous pouvez utiliser le Lambda API pour obtenir des informations sur l'état d'une fonction. Pour de plus amples informations, veuillez consulter États de la fonction Lambda.

# Configuration de la mémoire des fonctions Lambda

Lambda alloue de la puissance d'UC en fonction de la quantité de mémoire configurée. La mémoire est la quantité de mémoire disponible pour une fonction Lambda lors de l'exécution. Vous pouvez augmenter ou diminuer la mémoire et la puissance du processeur allouées à votre fonction à l'aide du paramètre Mémoire. Vous pouvez configurer la mémoire entre 128 Mo et 10 240 Mo par incréments de 1 Mo. A 1 769 Mo, une fonction dispose de l'équivalent d'1 vCPU (un vCPU-seconde de crédits par seconde).

Cette page décrit comment et quand mettre à jour le paramètre de mémoire d'une fonction Lambda.

Sections


- Déterminer le paramètre de mémoire approprié pour une fonction Lambda
- Configuration de la mémoire d'une fonction (console)
- Configuration de la mémoire des fonctions (AWS CLI)
- Configuration de la mémoire des fonctions (AWS SAM)
- Acceptation des recommandations relatives à la mémoire d'une fonction (console)


## Déterminer le paramètre de mémoire approprié pour une fonction Lambda

La mémoire est le principal levier permettant de contrôler les performances d'une fonction. Le paramètre par défaut, 128 Mo, est le paramètre le plus bas possible. Nous vous recommandons de n'utiliser que 128 Mo pour les fonctions Lambda simples, telles que celles qui transforment et acheminent des événements vers d'autres AWS services. Une allocation de mémoire plus élevée peut améliorer les performances des fonctions qui utilisent des bibliothèques importées, des couches Lambda, Amazon Simple Storage Service (Amazon S3) ou Amazon Elastic File System (Amazon EFS). L'ajout de mémoire augmente proportionnellement la quantité de processeur, augmentant ainsi la puissance de calcul globale disponible. Si une fonction est liée au processeur, au réseau ou à la mémoire, l'augmentation du paramètre de mémoire peut améliorer considérablement ses performances.

Pour trouver la configuration de mémoire adaptée à vos fonctions, nous vous recommandons d'utiliser l'outil open source AWS Lambda Power Tuning. Cet outil permet AWS Step Functions d'exécuter plusieurs versions simultanées d'une fonction Lambda avec différentes allocations de mémoire et de mesurer les performances. La fonction de saisie s'exécute dans votre AWS compte et effectue des appels HTTP en direct et une interaction avec le SDK, afin de mesurer les performances

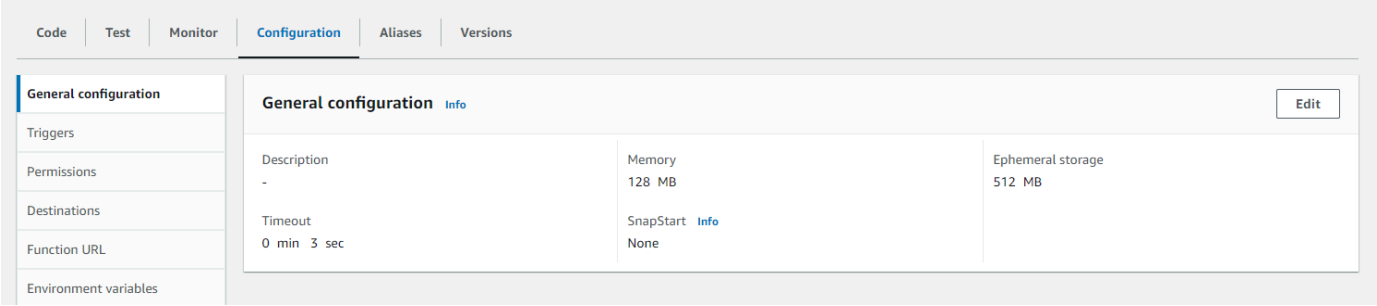
probables dans un scénario de production en direct. Vous pouvez également implémenter un processus CI/CD pour utiliser cet outil afin de mesurer automatiquement les performances des nouvelles fonctions que vous déployez.

## Configuration de la mémoire d'une fonction (console)

Vous pouvez configurer la mémoire de votre fonction dans la console Lambda.

Pour mettre à jour la mémoire d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez l'onglet Configuration, puis sélectionnez Configuration générale.




4. Sous Configuration générale, choisissez Modifier.
5. Pour la mémoire, définissez une valeur comprise entre 128 Mo et 10 240 Mo.
6. Choisissez Enregistrer.

## Configuration de la mémoire des fonctions (AWS CLI)

Vous pouvez utiliser la update-function-configurationcommande pour configurer la mémoire de votre fonction.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --memory-size 1024
```

## Configuration de la mémoire des fonctions (AWS SAM)

Vous pouvez utiliser le AWS Serverless Application Modelpour configurer la mémoire de votre fonction. Mettez à jour la MemorySizepropriété dans votre `template.yaml` fichier, puis exécutez sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 1024
      # Other function properties...
```

## Acceptation des recommandations relatives à la mémoire d'une fonction (console)

Si vous disposez d'autorisations d'administrateur dans AWS Identity and Access Management (IAM), vous pouvez choisir de recevoir des recommandations de configuration de la mémoire des fonctions Lambda de la part de. AWS Compute Optimizer Pour obtenir des instructions sur la manière de choisir de recevoir des recommandations relatives à la mémoire pour votre compte ou votre organisation, consultez Inscription à votre compte dans le Guide de l'utilisateur AWS Compute Optimizer .

> **Note**
>
> Compute Optimizer prend en charge uniquement les fonctions qui utilisent l'architecture x86_64.

Si vous avez choisi cette option et si votre fonction Lambda répond aux exigences de Compute Optimizer, vous pouvez afficher et accepter les recommandations de Compute Optimizer relatives à la mémoire de la fonction dans la console Lambda dans Configuration générale.

# Configuration du stockage éphémère pour les fonctions Lambda

Lambda fournit un stockage éphémère pour les fonctions du répertoire. `/tmp` Ce stockage est temporaire et propre à chaque environnement d'exécution. Vous pouvez contrôler la quantité de stockage éphémère allouée à votre fonction à l'aide du paramètre Stockage éphémère. Vous pouvez configurer le stockage éphémère entre 512 Mo et 10 240 Mo, par incréments de 1 Mo. Toutes les données qui y sont stockées sont cryptées au repos à l'aide d'une clé gérée par AWS. `/tmp`

Cette page décrit les cas d'utilisation courants et explique comment mettre à jour le stockage éphémère pour une fonction Lambda.

Sections


- Cas d'utilisation courants pour un stockage éphémère accru
- Configuration du magasin éphémère (console)
- Configuration du stockage éphémère ()AWS CLI
- Configuration du stockage éphémère ()AWS SAM


## Cas d'utilisation courants pour un stockage éphémère accru

Voici plusieurs cas d'utilisation courants qui bénéficient d'un stockage éphémère accru :

- xtract-transform-load Tâches E (ETL) : augmentez le stockage éphémère lorsque votre code effectue des calculs intermédiaires ou télécharge d'autres ressources pour terminer le traitement. Plus d'espace temporaire permet à des tâches ETL plus complexes de s'exécuter dans les fonctions Lambda.
- Inférence par apprentissage automatique (ML) : de nombreuses tâches d'inférence reposent sur des fichiers de données de référence volumineux, notamment des bibliothèques et des modèles. Grâce à un stockage éphémère accru, vous pouvez télécharger des modèles plus grands depuis Amazon Simple Storage Service (Amazon S3) `/tmp` et les utiliser dans le cadre de votre traitement.
- Traitement des données : pour les charges de travail qui téléchargent des objets depuis Amazon S3 en réponse à des événements S3, plus d'`/tmp`espace permet de gérer des objets plus volumineux sans recourir au traitement en mémoire. Les charges de travail qui créent des PDF ou traitent des médias bénéficient également d'un stockage plus éphémère.
- Traitement graphique : le traitement d'image est un cas d'utilisation courant pour les applications basées sur Lambda. Pour les charges de travail qui traitent de gros fichiers TIFF ou des images

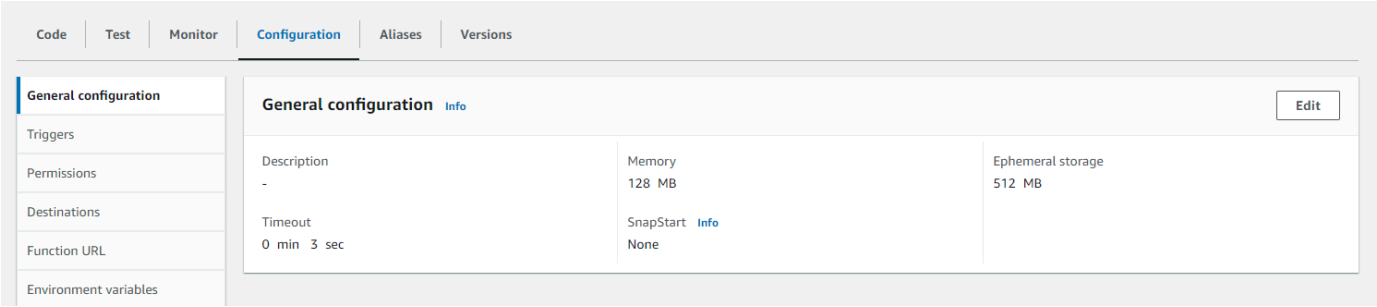
satellites, un stockage plus éphémère facilite l'utilisation des bibliothèques et l'exécution des calculs dans Lambda.

## Configuration du magasin éphémère (console)

Vous pouvez configurer le stockage éphémère dans la console Lambda.

Pour modifier le stockage éphémère d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez l'onglet Configuration, puis sélectionnez Configuration générale.




4. Sous Configuration générale, choisissez Modifier.
5. Pour le stockage éphémère, définissez une valeur comprise entre 512 Mo et 10 240 Mo, par incréments de 1 Mo.
6. Choisissez Enregistrer.

## Configuration du stockage éphémère ()AWS CLI

Vous pouvez utiliser la update-function-configurationcommande pour configurer le stockage éphémère.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --ephemeral-storage '{"Size": 1024}'
```

# Configuration du stockage éphémère ()AWS SAM

Vous pouvez utiliser le AWS Serverless Application Modelpour configurer le stockage éphémère pour votre fonction. Mettez à jour la EphemeralStoragepropriété dans votre `template.yaml` fichier, puis exécutez sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      # Other function properties...
```

# Sélection et configuration d'une architecture de jeu d'instructions pour votre fonction Lambda

L'architecture de l'ensemble des instructions d'une fonction Lambda détermine le type de processeur d'ordinateur que Lambda utilise pour exécuter la fonction. Lambda permet de choisir parmi plusieurs architectures de l'ensemble des instructions :

- arm64 — ARM architecture 64 bits, pour le processeur AWS Graviton2.
- x86_64 : architecture 64 bits x86, pour les processeurs x86.

> **Note**
>
> L'architecture arm64 est disponible dans la plupart des Régions AWS cas. Pour plus d'informations, consultez AWS Lambda Pricing (Tarification CTlong). Dans le tableau des prix de la mémoire, choisissez l'onglet Arm Price, puis ouvrez la liste déroulante des régions pour voir qui Régions AWS prend en charge arm64 avec Lambda.
> Pour un exemple de création d'une fonction avec l'architecture arm64, voir AWS Lambda Fonctions alimentées par le processeur AWS Graviton2.

Rubriques


- Avantages liés à l'utilisation de l'architecture arm64
- Exigences pour la migration vers l'architecture arm64
- Compatibilité des codes de fonction avec l'architecture arm64
- Comment migrer vers l'architecture arm64
- Configuration de l'architecture de l'ensemble des instructions


## Avantages liés à l'utilisation de l'architecture arm64

Les fonctions Lambda qui utilisent l'architecture arm64 (processeur AWS Graviton2) peuvent atteindre un prix et des performances nettement supérieurs à ceux des fonctions équivalentes exécutées sur l'architecture x86_64. Envisagez d'utiliser arm64 pour les applications de calcul intensif telles que le calcul haute performance, le codage vidéo et les applications de simulation.

Le Graviton2 CPU utilise le noyau Neoverse N1 et supporte Armv8.2 (y compris les extensions cryptographiques) CRC ainsi que plusieurs autres extensions architecturales.

Graviton2 réduit le temps de lecture de la mémoire en fournissant un cache L2 plus important par vCPU, ce qui améliore les performances de latence des backends Web et mobiles, des microservices et des systèmes de traitement de données. Graviton2 fournit également des performances de chiffrement améliorées et prend en charge des ensembles d'instructions qui améliorent la latence des inférences CPU basées sur l'apprentissage automatique.

Pour plus d'informations sur AWS Graviton2, consultez la section Processeur AWS Graviton.

## Exigences pour la migration vers l'architecture arm64

Lorsque vous sélectionnez une fonction Lambda à migrer vers l'architecture arm64, pour que la migration soit fluide, assurez-vous que la fonction répond aux exigences suivantes :

- Le package de déploiement ne contient que des composants open source et le code source que vous contrôlez, ce qui vous permet d'effectuer les mises à jour nécessaires à la migration.
- Si le code de la fonction inclut des dépendances tierces, chaque bibliothèque ou package fournit une version arm64.

## Compatibilité des codes de fonction avec l'architecture arm64

Le code de la fonction Lambda doit être compatible avec l'architecture de l'ensemble des instructions de la fonction. Avant de migrer une fonction vers l'architecture arm64, veuillez noter les points suivants concernant le code de fonction actuel :

- Si vous avez ajouté le code de la fonction à l'aide de l'éditeur de code intégré, le code s'exécute probablement sur l'une des architectures sans modification.
- Si vous avez chargé le code de la fonction, vous devez charger un nouveau code compatible avec l'architecture cible.
- Si la fonction utilise des couches, vous devez vérifier chaque couche afin de vous assurer qu'elle est compatible avec la nouvelle architecture. Si une couche n'est pas compatible, modifiez la fonction, afin de remplacer la version actuelle de la couche par une version de couche compatible.
- Si la fonction utilise des extensions Lambda, vous devez vérifier chaque extension, afin de vous assurer de sa compatibilité avec la nouvelle architecture.

- Si votre fonction utilise un type de package de déploiement de l'image de conteneur, vous devez créer une nouvelle image de conteneur compatible avec l'architecture de la fonction.

## Comment migrer vers l'architecture arm64

Pour migrer une fonction Lambda vers l'architecture arm64, nous recommandons de procéder comme suit :

1. Créez la liste des dépendances de l'application ou de la charge de travail. Les dépendances courantes comprennent :
   - Toutes les bibliothèques et les packages utilisés par la fonction.
   - Les outils que vous utilisez pour créer, déployer et tester la fonction, tels que les compilateurs, les suites de tests, les pipelines d'intégration continue et de livraison continue (CI/CD), les outils d'approvisionnement et les scripts.
   - Les extensions Lambda et les outils tiers que vous utilisez pour contrôler la fonction en production.
2. Pour chacune des dépendances, vérifiez la version, puis vérifiez si les versions arm64 sont disponibles.
3. Créez un environnement pour migrer l'application.
4. Amorcez l'application.
5. Testez et déboguez l'application.
6. Testez la performance de la fonction arm64. Comparez la performance avec celle de la version x86_64.
7. Mettez à jour le pipeline d'infrastructure pour prendre en charge les fonctions Lambda arm64.
8. Établissez le déploiement vers la production.

   Par exemple, utilisez la configuration du routage d'alias pour fractionner le trafic entre les versions x86 et arm64 de la fonction et comparez les performances et la latence.

Pour plus d'informations sur la création d'un environnement de code pour l'architecture arm64, y compris des informations spécifiques au langage pour Java, Go,. NET, et Python, consultez le GitHub référentiel Getting started with AWS Graviton.

# Configuration de l'architecture de l'ensemble des instructions

Vous pouvez configurer l'architecture du jeu d'instructions pour les fonctions Lambda nouvelles et existantes à l'aide de la console Lambda, AWS SDKs, AWS Command Line Interface ()AWS CLI ou. AWS CloudFormation Procédez comme suit pour modifier l'architecture de l'ensemble des instructions d'une fonction Lambda existante à partir de la console.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction pour laquelle vous voulez configurer l'architecture de l'ensemble des instructions.
3. Dans l'onglet principal Code, pour la section Paramètres d'exécution, choisissez Modifier.
4. Sous Architecture, choisissez l'architecture de l'ensemble des instructions que votre fonction doit utiliser.
5. Choisissez Save (Enregistrer).

# Configurer le délai d'expiration de la fonction Lambda

Lambda exécute votre code pendant une durée définie avant l’expiration du délai. Le délai d’expiration est la durée maximale, en secondes, pendant laquelle une fonction Lambda peut être exécutée. La valeur par défaut de ce paramètre est de 3 secondes, mais vous pouvez l'ajuster par incréments d'une seconde jusqu'à une valeur maximale de 900 secondes (15 minutes).

Cette page décrit comment et quand mettre à jour le paramètre de délai d'expiration d'une fonction Lambda.

Sections


- Déterminer la valeur de délai d'expiration appropriée pour une fonction Lambda
- Configuration du délai d'expiration (console)
- Configuration du délai d'expiration ()AWS CLI
- Configuration du délai d'expiration ()AWS SAM


## Déterminer la valeur de délai d'expiration appropriée pour une fonction Lambda

Si la valeur du délai d'expiration est proche de la durée moyenne d'une fonction, le risque que la fonction expire de façon inattendue est plus élevé. La durée d'une fonction peut varier en fonction de la quantité de données transférées et traitées, ainsi que de la latence des services avec lesquels la fonction interagit. Parmi les causes courantes de délai d'attente, citons :

- Les téléchargements depuis Amazon Simple Storage Service (Amazon S3) sont plus importants ou prennent plus de temps que la moyenne.
- Une fonction envoie une demande à un autre service, qui met plus de temps à répondre.
- Les paramètres fournis à une fonction nécessitent une complexité de calcul accrue dans la fonction, ce qui entraîne un allongement de l'invocation.

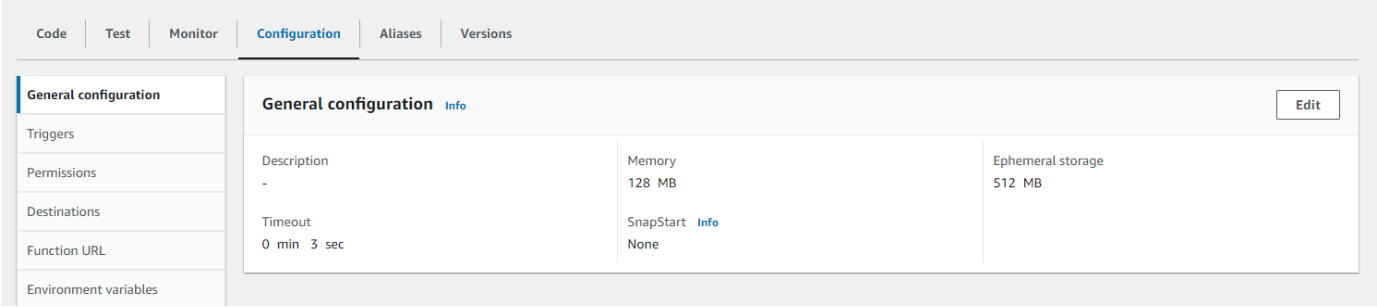
Lorsque vous testez votre application, assurez-vous que vos tests reflètent avec précision la taille et la quantité de données et que les valeurs des paramètres sont réalistes. Les tests utilisent souvent de petits échantillons pour des raisons de commodité, mais vous devez utiliser des ensembles de données se situant dans les limites supérieures de ce que l'on peut raisonnablement attendre de votre charge de travail.

## Configuration du délai d'expiration (console)

Vous pouvez configurer le délai d'expiration de la fonction dans la console Lambda.

Pour modifier le délai d'expiration d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez l'onglet Configuration, puis sélectionnez Configuration générale.




4. Sous Configuration générale, choisissez Modifier.
5. Pour Timeout, définissez une valeur comprise entre 1 et 900 secondes (15 minutes).
6. Choisissez Enregistrer.

## Configuration du délai d'expiration ()AWS CLI

Vous pouvez utiliser la update-function-configurationcommande pour configurer la valeur du délai d'expiration, en secondes. L'exemple de commande suivant augmente le délai d'expiration de la fonction à 120 secondes (2 minutes).

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --timeout 120
```

## Configuration du délai d'expiration ()AWS SAM

Vous pouvez utiliser le AWS Serverless Application Modelpour configurer la valeur du délai d'expiration de votre fonction. Mettez à jour la propriété Timeout dans votre `template.yaml` fichier, puis exécutez sam deploy.

### Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      # Other function properties...
```

# Utiliser les variables d'environnement Lambda pour configurer les valeurs dans le code

Vous pouvez utiliser des variables d’environnement pour ajuster le comportement de votre fonction sans mettre à jour le code. Une variable d’environnement est une paire de chaînes stockées dans la configuration spécifique à la version d’une fonction. L’environnement d’exécution de Lambda met les variables d’environnement à la disposition de votre code et définit les variables d’environnement supplémentaires qui contiennent des informations sur la fonction et la demande d’invocation.

> **Note**
>
> Pour renforcer la sécurité, nous vous recommandons d'utiliser à la AWS Secrets Manager place des variables d'environnement pour stocker les informations d'identification de la base de données et d'autres informations sensibles telles que API les clés ou les jetons d'autorisation. Pour plus d'informations, voir Créer et gérer des secrets avec AWS Secrets Manager.

Les variables d’environnement ne sont pas évaluées avant l’invocation de la fonction. Toute valeur que vous définissez est considérée comme une chaîne littérale et non développée. Effectuez l’évaluation de la variable dans le code de votre fonction.

Vous pouvez configurer des variables d'environnement dans Lambda à l'aide de la console Lambda, du AWS Command Line Interface (AWS CLI), AWS Serverless Application Model (AWS SAM) ou d'un. AWS SDK

Console

Vous définissez des variables d’environnement sur la version non publiée de votre fonction. Lorsque vous publiez une version, les variables d'environnement sont verrouillées pour cette version ainsi que pour les autres paramètres de configuration spécifiques à la version.

Vous créez une variable d’environnement pour votre fonction en définissant une clé et une valeur. Votre fonction utilise le nom de la clé pour récupérer la valeur de la variable d’environnement.

Pour définir des variables d’environnement dans la console Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez une fonction.
3. Choisissez Configuration, puis Environment variables (Variables d'environnement).
4. Sous Variables d'environnement, choisissez Modifier.
5. Choisissez Ajouter une variable d'environnement.
6. Entrez une clé et une valeur.

   Prérequis

   - Les clés commencent par une lettre et comportent au moins deux caractères.
   - Les clés contiennent uniquement des lettres, des chiffres et le caractère de soulignement (_).
   - Les clés ne sont pas réservées par Lambda.
   - La taille totale de toutes les variables d'environnement ne dépasse pas 4 Ko.
7. Choisissez Save (Enregistrer).

Pour générer une liste de variables d'environnement dans l'éditeur de code de la console

Vous pouvez générer une liste de variables d'environnement dans l'éditeur de code Lambda. Il s'agit d'un moyen rapide de référencer vos variables d'environnement pendant que vous codez.

1. Cliquez sur l'onglet Code.
2. Choisissez l'onglet Variables d'environnement.
3. Choisissez Outils, puis Afficher les variables d'environnement.

Les variables d'environnement restent chiffrées lorsqu'elles sont répertoriées dans l'éditeur de code de la console. Si vous avez activé les assistants de chiffrement pour le chiffrement en transit, ces paramètres restent inchangés. Pour de plus amples informations, veuillez consulter Sécurisation des variables d'environnement Lambda.

La liste des variables d'environnement est en lecture seule et n'est disponible que sur la console Lambda. Ce fichier n'est pas inclus lorsque vous téléchargez l'archive .zip de la fonction et vous ne pouvez pas ajouter de variables d'environnement en chargeant ce fichier.

AWS CLI

L'exemple suivant définit deux variables d'environnement sur une fonction nommée `my-function`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --environment "Variables={BUCKET=amzn-s3-demo-bucket,KEY=file.txt}"
```

Lorsque vous appliquez des variables d'environnement avec la commande `update-function-configuration`, l'ensemble du contenu de la structure `Variables` est remplacé. Pour conserver les variables d'environnement existantes lorsque vous en ajoutez une nouvelle, incluez toutes les valeurs existantes dans votre demande.

Pour obtenir la configuration actuelle, utilisez la commande `get-function-configuration`.

```
aws lambda get-function-configuration \
  --function-name my-function
```

Vous devriez voir la sortie suivante :

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
    "Runtime": "nodejs20.x",
    "Role": "arn:aws:iam::111122223333:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "amzn-s3-demo-bucket",
            "KEY": "file.txt"
        }
    },
    "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",
    ...
}
```

Vous pouvez transmettre l'identifiant de révision depuis la sortie de `get-function-configuration` en tant que paramètre à `update-function-configuration`. Cela garantit que les valeurs ne changent pas entre le moment où vous lisez la configuration et celui où vous la mettez à jour.

Pour configurer la clé de chiffrement d'une fonction, définissez l'option `KMSKeyARN`.

```
aws lambda update-function-configuration \
  --function-name my-function \
```

```
  --kms-key-arn arn:aws:kms:us-east-2:111122223333:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

### AWS SAM

Vous pouvez utiliser le AWS Serverless Application Modelpour configurer les variables d'environnement de votre fonction. Mettez à jour les propriétés de l'environnement et des variables dans votre `template.yaml` fichier, puis exécutez sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      Environment:
        Variables:
          BUCKET: amzn-s3-demo-bucket
          KEY: file.txt
      # Other function properties...
```

### AWS SDKs

Pour gérer les variables d'environnement à l'aide d'un AWS SDK, utilisez les API opérations suivantes.

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Pour en savoir plus, consultez la AWS SDKdocumentation de votre langage de programmation préféré.

## Variables d'environnement d'exécution définies

Les runtimes Lambda définissent plusieurs variables d'environnement lors de l'initialisation. La plupart des variables d'environnement fournissent des informations sur la fonction ou l'exécution. Les clés de ces variables d'environnement sont réservées et ne peuvent pas être définies dans la configuration de la fonction.

Variables d'environnement réservées

- `_HANDLER` – Emplacement de gestionnaire configuré sur la fonction.
- `_X_AMZN_TRACE_ID` – En-tête de suivi X-Ray. Cette variable d'environnement change à chaque invocation.
  - Cette variable d'environnement n'est pas définie pour les environnements d'exécution uniquement basés sur le système d'exploitation (famille des environnements d'exécution `provided`). Vous pouvez définir `_X_AMZN_TRACE_ID` pour les environnements d'exécution personnalisés à l'aide de l'en-tête de réponse `Lambda-Runtime-Trace-Id` de l'Invocation suivante.
  - Pour les versions 17 et ultérieures du runtime Java, cette variable d'environnement n'est pas utilisée. Lambda stocke plutôt les informations de suivi dans la propriété du système `com.amazonaws.xray.traceHeader`.
- `AWS_DEFAULT_REGION`— L' Région AWS endroit par défaut où la fonction Lambda est exécutée.
- `AWS_REGION`— L' Région AWS endroit où la fonction Lambda est exécutée. Si elle est définie, cette valeur remplace `AWS_DEFAULT_REGION`.
  - Pour plus d'informations sur l'utilisation des variables d' Région AWS environnement avec AWS SDKs, consultez AWS la section Région dans le guide de référence des outils AWS SDKs et.
- `AWS_EXECUTION_ENV` – Identifiant d'exécution, doté du préfixe `AWS_Lambda_` (par exemple, `AWS_Lambda_java8`). Cette variable d'environnement n'est pas définie pour les environnements d'exécution uniquement basés sur le système d'exploitation (famille des environnements d'exécution `provided`).
- `AWS_LAMBDA_FUNCTION_NAME` – Nom de la fonction.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE` – Quantité de mémoire disponible pour la fonction en Mo.

- `AWS_LAMBDA_FUNCTION_VERSION` – Version de la fonction en cours d'exécution.
- `AWS_LAMBDA_INITIALIZATION_TYPE` – Type d'initialisation de la fonction, à savoir `on-demand`, `provisioned-concurrency` ou `snap-start`. Pour plus d'informations, consultez Configuration de la simultanéité provisionnée ou Améliorer les performances de démarrage avec Lambda SnapStart.
- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME` — Le nom du groupe Amazon CloudWatch Logs et du flux associés à la fonction. Les variables `AWS_LAMBDA_LOG_GROUP_NAME` d'`AWS_LAMBDA_LOG_STREAM_NAME`environnement et ne sont pas disponibles dans les fonctions Lambda SnapStart .
- `AWS_ACCESS_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` – Clés d'accès obtenues du rôle d'exécution de la fonction.
- `AWS_LAMBDA_RUNTIME_API`— (Runtime personnalisé) L'hôte et le port du runtime API.
- `LAMBDA_TASK_ROOT` – Chemin d'accès du code de la fonction Lambda.
- `LAMBDA_RUNTIME_DIR` – Chemin d'accès des bibliothèques de l'environnement d'exécution.

Les variables d'environnement supplémentaires suivantes ne sont pas réservées et peuvent être étendues dans la configuration de la fonction.

Variables d'environnement non réservées

- `LANG` – Paramètres régionaux de l'environnement d'exécution (`en_US.UTF-8`).
- `PATH` – Chemin d'exécution (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).
- `LD_LIBRARY_PATH` – Chemin de la bibliothèque système (`/var/lang/lib:/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- `NODE_PATH` – (Node.js) Chemin d'accès de la bibliothèque Node.js (`/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `PYTHONPATH` – (Python 2.7, 3.6, 3.8) Chemin d'accès de la bibliothèque Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH` – (Ruby) Chemin d'accès de la bibliothèque Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).
- `AWS_XRAY_CONTEXT_MISSING`— Pour le traçage par rayons X, Lambda le définit de manière `LOG_ERROR` à éviter de générer des erreurs d'exécution depuis le X-Ray. SDK

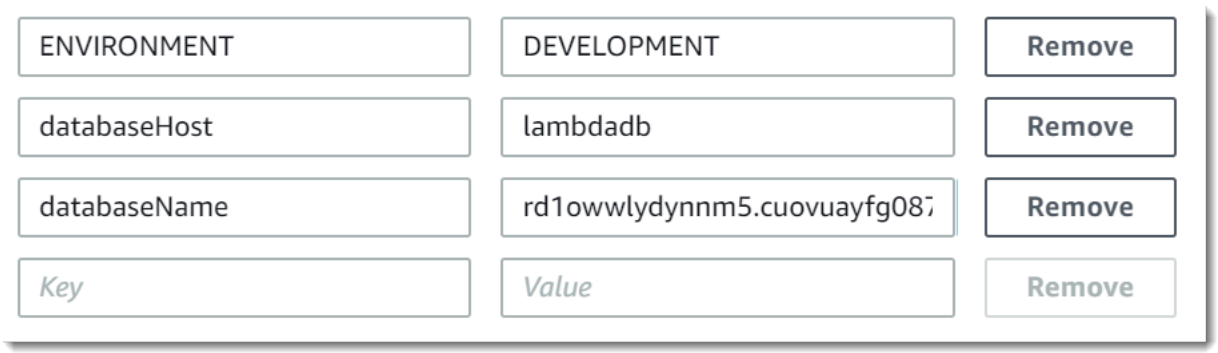
- `AWS_XRAY_DAEMON_ADDRESS` – Pour le suivi X-Ray, adresse IP et le port du démon X-Ray.
- `AWS_LAMBDA_DOTNET_PREJIT`— Pour le. NET6 et. NET7 environnements d'exécution, définissez cette variable pour qu'elle soit activée ou désactivée. NEToptimisations d'exécution spécifiques. Les valeurs incluent `always`, `never` et `provisioned-concurrency`. Pour de plus amples informations, veuillez consulter Configuration de la simultanéité provisionnée pour une fonction.
- `TZ` – Fuseau horaire de l'environnement (`:UTC`). L'environnement d'exécution permet NTP de synchroniser l'horloge du système.

Les valeurs d'exemple affichées reflètent les derniers moteurs d'exécution. La présence de variables spécifiques ou leurs valeurs peuvent varier dans les anciens environnements d'exécution.

## Exemple de scénario pour les variables d'environnement

Vous pouvez utiliser des variables d'environnement pour personnaliser le comportement des fonctions dans vos environnements de test et de production. Vous pouvez par exemple créer deux fonctions avec le même code mais une configuration différente. Une fonction se connecte à une base de données de test et l'autre à une base de données de production. Dans ce cas, vous utilisez des variables d'environnement pour transmettre à la fonction le nom d'hôte, ainsi que d'autres détails de connexion pour la base de données.

L'exemple suivant montre comment définir l'hôte de la base de données et le nom de base de données en tant que variables d'environnement.




Si vous souhaitez que votre environnement de test génère plus d'informations de débogage que l'environnement de production, vous pouvez définir une variable d'environnement pour configurer votre environnement de test de manière à utiliser une journalisation plus approfondie ou un suivi plus détaillé.

# Sécurisation des variables d'environnement Lambda

Pour sécuriser vos variables d’environnement, vous pouvez utiliser le chiffrement côté serveur pour protéger vos données au repos, et le chiffrement côté client pour protéger vos données en transit.

> **Note**
>
> Pour renforcer la sécurité de la base de données, nous vous recommandons d'utiliser à la AWS Secrets Manager place des variables d'environnement pour stocker les informations d'identification de la base de données. Pour plus d’informations, consultez Utilisation AWS Lambda avec Amazon RDS.

## Sécurité au repos

Lambda fournit toujours le chiffrement côté serveur au repos grâce à une AWS KMS key. Par défaut, Lambda utilise une Clé gérée par AWS. Si ce comportement par défaut convient à votre flux de travail, vous n’avez pas besoin de configurer quoi que ce soit d’autre. Lambda les crée Clé gérée par AWS dans votre compte et gère les autorisations pour vous. AWS l'utilisation de cette clé ne vous est pas facturée.

Si vous préférez, vous pouvez plutôt fournir une clé gérée par le AWS KMS client. Vous pouvez procéder ainsi pour contrôler la rotation de la clé KMS ou pour répondre aux exigences de votre organisation en matière de gestion des clés KMS. Si vous utilisez une clé gérée par le client, seuls les utilisateurs de votre compte ayant accès à la clé KMS peuvent afficher ou gérer les variables d’environnement sur la fonction.

Les clés gérées par le client entraînent des AWS KMS frais standard. Pour en savoir plus, consultez AWS Key Management Service Tarification.

## Sécurité pendant le transit

Pour plus de sécurité, vous pouvez activer les assistants pour le chiffrement en transit, ce qui garantit que vos variables d’environnement sont chiffrées côté client pour une meilleure protection pendant le transit.

Pour configurer le chiffrement de vos variables d'environnement

1. Utilisez le AWS Key Management Service (AWS KMS) pour créer des clés gérées par le client que Lambda utilisera pour le chiffrement côté serveur et côté client. Pour en savoir plus, consultez Création des clés dans le AWS Key Management Service Guide du développeur .
2. À l'aide de la console Lambda, accédez à la page Modifier les variables d'environnement.
   a. Ouvrez la page Functions (Fonctions) de la console Lambda.
   b. Choisissez une fonction.
   c. Choisissez Configuration, puis Variables d'environnement dans la barre de navigation de gauche.
   d. Dans la section Variables d'environnement), choisissez Modifier.
   e. Développez Configuration du chiffrement.
3. (Facultatif) Activez les assistants de chiffrement de la console afin qu'ils utilisent le chiffrement côté client pour protéger vos données en transit.
   a. Sous Encryption in transit (Chiffrement pendant le transit), choisissez Enable helpers for encryption in transit (Activer les assistants pour le chiffrement pendant le transit).
   b. Pour chaque variable d'environnement pour laquelle vous souhaitez activer les assistants de chiffrement de la console, choisissez Encrypt (Chiffrer) en regard de la variable d'environnement.
   c. Sous AWS KMS key Pour chiffrer en cours de transfert, choisissez une clé gérée par le client que vous avez créée au début de cette procédure.
   d. Choisissez Politique de rôle d'exécution et copiez la politique. Cette politique octroie au rôle d'exécution de votre fonction l'autorisation de déchiffrer les variables d'environnement.

      Enregistrez cette politique, car vous l'utiliserez à la dernière étape de cette procédure.
   e. Ajoutez le code à votre fonction qui déchiffre les variables d'environnement . Pour obtenir un exemple, choisissez Déchiffrer l'extrait de secrets.
4. (Facultatif) Spécifiez votre clé gérée par le client pour le chiffrement au repos.
   a. Choisissez Utiliser une clé principale client.
   b. Choisissez une clé gérée par le client que vous avez créée au début de cette procédure.
5. Choisissez Enregistrer.
6. Configurez des autorisations.

Si vous utilisez une clé gérée par le client avec chiffrement côté serveur, octroyez des autorisations à tous les utilisateurs ou rôles que vous voulez pouvoir afficher ou pour gérer des variables d'environnement sur la fonction. Pour plus d'informations, consultez Gestion des autorisations sur votre clé de chiffrement KMS côté serveur.

Si vous activez le chiffrement côté client pour la sécurité pendant le transit, votre fonction doit disposer d'une autorisation pour appeler l'opération d'API `kms:Decrypt`. Ajoutez la politique que vous avez précédemment enregistrée dans cette procédure au rôle d'exécution de la fonction.

## Gestion des autorisations sur votre clé de chiffrement KMS côté serveur

Aucune AWS KMS autorisation n'est requise pour que votre utilisateur ou le rôle d'exécution de la fonction utilise la clé de chiffrement par défaut. Pour utiliser une clé gérée par le client, vous devez disposer des autorisations appropriées pour utiliser la clé. Lambda utilise vos autorisations pour créer un octroi sur la clé. Cela permet à Lambda de l'utiliser pour le chiffrement.

- `kms:ListAliases` – Pour afficher les clés dans la console Lambda.
- `kms:CreateGrant`, `kms:Encrypt` – Pour configurer une clé gérée par le client sur une fonction.
- `kms:Decrypt` – Pour afficher et gérer des variables d'environnement chiffrées à l'aide d'une clé gérée par le client.

Vous pouvez obtenir ces autorisations à partir de votre politique d'autorisation basée sur les ressources Compte AWS ou de celle d'une clé. `ListAliases`est fourni par les politiques gérées pour Lambda. Les stratégies de clé accordent les autorisations restantes aux utilisateurs du groupe Key users (Utilisateurs de clés).

Les utilisateurs sans autorisation `Decrypt` peuvent gérer les fonctions, mais ils ne peuvent pas afficher les variables d'environnement ni les gérer dans la console Lambda. Pour empêcher un utilisateur d'afficher des variables d'environnement, ajoutez une instruction aux autorisations de cet utilisateur afin de refuser l'accès à la clé par défaut, à une clé gérée par le client ou à toutes les clés.

Example Politique IAM – Refuser l'accès en fonction de l'ARN de la clé

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/3be10e2d-xmpl-4be4-
bc9d-0405a71945cc"
        }
    ]
}
```

Pour en savoir plus sur la gestion des autorisations de clés, consultez Stratégies de clé dans AWS KMS que vous trouverez dans le Guide du développeur AWS Key Management Service .

## Récupération des variables d'environnement Lambda

Pour récupérer les variables d’environnement dans le code de votre fonction, utilisez la méthode standard pour votre langage de programmation.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
  region = os.environ['AWS_REGION']
```

> **Note**
>
> Dans certains cas, vous devrez peut-être utiliser le format suivant :
>
> ```
> region = os.environ.get('AWS_REGION')
> ```

Ruby

```
region = ENV["AWS_REGION"]
```

Java

```
String region = System.getenv("AWS_REGION");
```

Go

```
var region = os.Getenv("AWS_REGION")
```

C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

PowerShell

```
$region = $env:AWS_REGION
```

Lambda stocke les variables d'environnement en toute sécurité en les chiffrant au repos. Vous pouvez configurer Lambda pour utiliser une clé de chiffrement différente, chiffrer les valeurs des variables d'environnement côté client ou définir des variables d'environnement dans un AWS CloudFormation modèle avec. AWS Secrets Manager

# Permettre aux fonctions Lambda d'accéder aux ressources d'un Amazon VPC

Avec Amazon Virtual Private Cloud (AmazonVPC), vous pouvez créer des réseaux privés Compte AWS pour héberger des ressources telles que des instances Amazon Elastic Compute Cloud (AmazonEC2), des instances Amazon Relational Database Service (RDSAmazon) et des instances ElastiCache Amazon. Vous pouvez donner à votre fonction Lambda l'accès aux ressources hébergées sur un Amazon en VPC attachant votre fonction VPC via les sous-réseaux privés qui contiennent les ressources. Suivez les instructions des sections suivantes pour associer une fonction Lambda à un Amazon à VPC l'aide de la console Lambda, du AWS Command Line Interface ()AWS CLI ou. AWS SAM

> **Note**
>
> Chaque fonction Lambda s'exécute dans une VPC fonction détenue et gérée par le service Lambda. Ils VPCs sont gérés automatiquement par Lambda et ne sont pas visibles pour les clients. La configuration de votre fonction pour accéder à d'autres AWS ressources d'un Amazon n'VPCa aucun effet sur le système géré par Lambda dans lequel VPC votre fonction est exécutée.

Sections


- IAMAutorisations requises
- Associer des fonctions Lambda à un Amazon VPC dans votre Compte AWS
- Accès à Internet lorsqu'il est connecté à un VPC
- IPv6soutien
- Bonnes pratiques d'utilisation de Lambda avec Amazon VPCs
- Comprendre les interfaces réseau élastiques Hyperplane () ENIs
- Utilisation des clés de IAM condition pour les VPC paramètres
- Didacticiels VPC

# IAMAutorisations requises

Pour associer une fonction Lambda à un Amazon VPC dans votre environnement, Compte AWS Lambda a besoin d'autorisations pour créer et gérer les interfaces réseau qu'elle utilise pour permettre à votre fonction d'accéder aux ressources du. VPC

Les interfaces réseau créées par Lambda sont appelées Hyperplane Elastic Network Interfaces ou Hyperplane. ENIs Pour en savoir plus sur ces interfaces réseau, consultezthe section called “Comprendre les interfaces réseau élastiques Hyperplane () ENIs”.

Vous pouvez accorder à votre fonction les autorisations dont elle a besoin en associant la politique AWS gérée AWSLambdaVPCAccessExecutionRoleau rôle d'exécution de votre fonction. Lorsque vous créez une nouvelle fonction dans la console Lambda et que vous l'associez à une, VPC Lambda ajoute automatiquement cette politique d'autorisation pour vous.

Si vous préférez créer votre propre politique d'IAMautorisations, veillez à ajouter toutes les autorisations suivantes :

- EC2 : CreateNetworkInterface
- ec2 : DescribeNetworkInterfaces — Cette action ne fonctionne que si elle est autorisée sur toutes les ressources (`"Resource": "*"`).
- EC2 : DescribeSubnets
- ec2 : DeleteNetworkInterface — Si vous ne spécifiez pas d'ID de ressource pour DeleteNetworkInterfacele rôle d'exécution, votre fonction ne pourra peut-être pas accéder auVPC. Spécifiez un ID de ressource unique ou incluez toutes les ressourcesIDs, par exemple`"Resource": "arn:aws:ec2:us-west-2:123456789012:*/*"`.
- EC2 : AssignPrivateIpAddresses
- EC2 : UnassignPrivateIpAddresses

Notez que le rôle de votre fonction n'a besoin de ces autorisations que pour créer les interfaces réseau, et non pour appeler votre fonction. Vous pouvez toujours invoquer votre fonction avec succès lorsqu'elle est attachée à un AmazonVPC, même si vous supprimez ces autorisations du rôle d'exécution de votre fonction.

Pour associer votre fonction à unVPC, Lambda doit également vérifier les ressources réseau à l'aide de votre rôle IAM d'utilisateur. Assurez-vous que votre rôle d'utilisateur dispose des IAM autorisations suivantes :

- EC2 : DescribeSecurityGroups
- EC2 : DescribeSubnets
- EC2 : DescribeVpcs

> **Note**
>
> Les EC2 autorisations Amazon que vous accordez au rôle d'exécution de votre fonction sont utilisées par le service Lambda pour associer votre fonction à un. VPC Cependant, vous accordez également implicitement ces autorisations au code de votre fonction. Cela signifie que votre code de fonction est capable de passer ces EC2 API appels Amazon. Pour obtenir des conseils sur le respect des meilleures pratiques en matière de sécurité, consultezthe section called “Bonnes pratiques de sécurité”.

## Associer des fonctions Lambda à un Amazon VPC dans votre Compte AWS

Associez votre fonction à un Amazon VPC dans votre ordinateur à Compte AWS l'aide de la console Lambda, du AWS CLI ou. AWS SAM Si vous utilisez le AWS CLI ou AWS SAM, ou si vous associez une fonction existante à une console VPC utilisant la console Lambda, assurez-vous que le rôle d'exécution de votre fonction dispose des autorisations nécessaires répertoriées dans la section précédente.

Les fonctions Lambda ne peuvent pas se connecter directement à une instance VPC dotée d'une location d'instance dédiée. Pour vous connecter aux ressources d'un espace dédiéVPC, connectez-le à une seconde VPC avec une location par défaut.

Lambda console

Pour associer une fonction à un Amazon VPC lorsque vous le créez

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Sous Informations de base, dans Nom de fonction, entrez un nom pour votre fonction.
3. Configurez les VPC paramètres de la fonction en procédant comme suit :

    a. Développez Advanced settings (Paramètres avancés).

    b. Sélectionnez Activer VPC, puis sélectionnez celui VPC auquel vous souhaitez associer la fonction.

c. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.

d. Choisissez les sous-réseaux et les groupes de sécurité pour lesquels créer l'interface réseau. Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR

> **Note**
>
> Pour accéder aux ressources privées, connectez votre fonction à des sous-réseaux privés. Si votre fonction nécessite un accès à Internet, consultezthe section called “Accès à Internet pour les VPC fonctions”. La connexion d’une fonction à un sous-réseau public ne lui donne pas accès à Internet ou à une adresse IP publique.

4. Sélectionnez Create function (Créer une fonction).

Pour associer une fonction existante à un Amazon VPC

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Choisissez l'onglet Configuration, puis choisissez VPC.
3. Choisissez Modifier.
4. Sous VPC, sélectionnez l'Amazon VPC auquel vous souhaitez associer votre fonction.
5. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.
6. Choisissez les sous-réseaux et les groupes de sécurité pour lesquels créer l'interface réseau. Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR

> **Note**
>
> Pour accéder aux ressources privées, connectez votre fonction à des sous-réseaux privés. Si votre fonction nécessite un accès à Internet, consultezthe section called “Accès à Internet pour les VPC fonctions”. La connexion d’une fonction à un sous-réseau public ne lui donne pas accès à Internet ou à une adresse IP publique.

7. Choisissez Save (Enregistrer).

### AWS CLI

Pour associer une fonction à un Amazon VPC lorsque vous le créez

- Pour créer une fonction Lambda et l'associer à unVPC, exécutez la commande suivante CLI`create-function`.

```
aws lambda create-function --function-name my-function \
--runtime nodejs20.x --handler index.js --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/lambda-role \
--vpc-config
 Ipv6AllowedForDualStack=true,SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a03
```

Spécifiez vos propres sous-réseaux et groupes de sécurité et configurez-les sur `true` ou `Ipv6AllowedForDualStack` en `false` fonction de votre cas d'utilisation.

Pour associer une fonction existante à un Amazon VPC

- Pour associer une fonction existante à unVPC, exécutez la CLI `update-function-configuration` commande suivante.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config Ipv6AllowedForDualStack=true,
 SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912
```

Pour dissocier votre fonction d'un VPC

- Pour dissocier votre fonction d'unVPC, exécutez la `update-function-configuration` CLI commande suivante avec une liste vide de VPC sous-réseaux et de groupes de sécurité.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

AWS SAM

Pour associer votre fonction à un VPC

- Pour associer une fonction Lambda à un AmazonVPC, ajoutez la `VpcConfig` propriété à votre définition de fonction, comme indiqué dans l'exemple de modèle suivant. Pour plus d'informations sur cette propriété, voir AWS: :Lambda : :Function VpcConfig dans le guide de AWS CloudFormation l'utilisateur (la AWS SAM `VpcConfig` propriété est transmise directement à la `VpcConfig` propriété d'une AWS CloudFormation `AWS::Lambda::Function` ressource).

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./lambda_function/
      Handler: lambda_function.handler
      Runtime: python3.12
      VpcConfig:
        SecurityGroupIds:
          - !Ref MySecurityGroup
        SubnetIds:
          - !Ref MySubnet1
          - !Ref MySubnet2
      Policies:
        - AWSLambdaVPCAccessExecutionRole

  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for Lambda function
      VpcId: !Ref MyVPC

  MySubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.1.0/24
```

```
  MySubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.2.0/24

  MyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
```

Pour plus d'informations sur la configuration VPC de votre AWS SAM entrée, consultez AWS: : EC2 : : VPC dans le guide de AWS CloudFormation l'utilisateur.

## Accès à Internet lorsqu'il est connecté à un VPC

Par défaut, les fonctions Lambda ont accès à l'Internet public. Lorsque vous associez votre fonction à unVPC, elle ne peut accéder qu'aux ressources disponibles dans celui-ciVPC. Pour permettre à votre fonction d'accéder à Internet, vous devez également configurer le pour VPC qu'elle ait accès à Internet. Pour en savoir plus, consultez the section called “Accès à Internet pour les VPC fonctions”.

## IPv6soutien

Votre fonction peut se connecter aux ressources des VPC sous-réseaux à double pile. IPv6 Cette option est désactivée par défaut. Pour autoriser le IPv6 trafic sortant, utilisez la console ou l'`--vpc-config Ipv6AllowedForDualStack=true`option associée à la fonction ou à la commande de création. update-function-configuration

> **Note**
>
> Pour autoriser le IPv6 trafic sortant dans aVPC, tous les sous-réseaux connectés à la fonction doivent être des sous-réseaux à double pile. Lambda ne prend pas en charge les IPv6 connexions IPv6 sortantes pour les sous-réseaux uniquement dans aVPC, les connexions sortantes pour les fonctions qui ne sont pas connectées à aVPC, ni les IPv6 connexions entrantes utilisant des points de terminaison (). IPv6 VPC AWS PrivateLink

Vous pouvez mettre à jour votre code de fonction pour vous connecter explicitement aux ressources du sous-réseau. IPv6 L'exemple Python suivant ouvre un socket et se connecte à un IPv6 serveur.

Example — Se connecter au IPv6 serveur

```
def connect_to_server(event, context):
    server_address = event['host']
    server_port = event['port']
    message = event['message']
    run_connect_to_server(server_address, server_port, message)

def run_connect_to_server(server_address, server_port, message):
    sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        # Send data
        sock.connect((server_address, int(server_port), 0, 0))
        sock.sendall(message.encode())
        BUFF_SIZE = 4096
        data = b''
        while True:
            segment = sock.recv(BUFF_SIZE)
            data += segment
            # Either 0 or end of data
            if len(segment) < BUFF_SIZE:
                break
        return data
    finally:
        sock.close()
```

## Bonnes pratiques d'utilisation de Lambda avec Amazon VPCs

Pour vous assurer que votre VPC configuration Lambda est conforme aux meilleures pratiques, suivez les conseils des sections suivantes.

### Bonnes pratiques de sécurité

Pour associer votre fonction Lambda à unVPC, vous devez attribuer un certain nombre d'autorisations Amazon EC2 au rôle d'exécution de votre fonction. Ces autorisations sont nécessaires pour créer les interfaces réseau que votre fonction utilise pour accéder aux ressources duVPC. Toutefois, ces autorisations sont également implicitement accordées au code de votre fonction. Cela signifie que votre code de fonction est autorisé à effectuer ces EC2 API appels Amazon.

Pour suivre le principe du moindre privilège d'accès, ajoutez une politique de refus, comme dans l'exemple suivant, au rôle d'exécution de votre fonction. Cette politique empêche votre fonction de

passer des appels à l'Amazon EC2 APIs que le service Lambda utilise pour associer votre fonction à un. VPC

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                 "ec2:CreateNetworkInterface",
                 "ec2:DeleteNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
                 "ec2:DetachNetworkInterface",
                 "ec2:AssignPrivateIpAddresses",
                 "ec2:UnassignPrivateIpAddresses",
            ],
            "Resource": [ "*" ],
            "Condition": {
                "ArnEquals": {
                    "lambda:SourceFunctionArn": [
                        "arn:aws:lambda:us-west-2:123456789012:function:my_function"
                    ]
                }
            }
        }
    ]
}
```

AWS fournit des groupes de sécurité et des listes de contrôle d'accès réseau (ACLs) pour renforcer la sécurité de votreVPC. Les groupes de sécurité contrôlent le trafic entrant et sortant pour vos ressources, et le réseau ACLs contrôlent le trafic entrant et sortant pour vos sous-réseaux. Les groupes de sécurité offrent un contrôle d'accès suffisant pour la plupart des sous-réseaux. Vous pouvez utiliser le réseau ACLs si vous souhaitez une couche de sécurité supplémentaire pour votreVPC. Pour obtenir des directives générales sur les meilleures pratiques de sécurité lors de l'utilisation d'AmazonVPCs, consultez les meilleures pratiques de sécurité pour vous VPC dans le guide de l'utilisateur d'Amazon Virtual Private Cloud.

## Bonnes pratiques en matière de performances

Lorsque vous attachez votre fonction à unVPC, Lambda vérifie s'il existe une ressource réseau disponible (HyperplaneENI) à laquelle elle peut se connecter. Les hyperplans ENIs sont associés

à une combinaison particulière de groupes de sécurité et de VPC sous-réseaux. Si vous avez déjà attaché une fonction à unVPC, le fait de spécifier les mêmes sous-réseaux et groupes de sécurité lorsque vous associez une autre fonction signifie que Lambda peut partager les ressources du réseau et éviter d'avoir à créer un nouvel hyperplan. ENI Pour plus d'informations sur Hyperplane ENIs et son cycle de vie, consultezthe section called “Comprendre les interfaces réseau élastiques Hyperplane () ENIs”.

## Comprendre les interfaces réseau élastiques Hyperplane () ENIs

Un hyperplan ENI est une ressource gérée qui agit comme une interface réseau entre votre fonction Lambda et les ressources auxquelles vous souhaitez que votre fonction se connecte. Le service Lambda les crée et les gère ENIs automatiquement lorsque vous associez votre fonction à un. VPC

Les hyperplans ne ENIs sont pas directement visibles pour vous et vous n'avez pas besoin de les configurer ou de les gérer. Cependant, le fait de connaître leur fonctionnement peut vous aider à comprendre le comportement de votre fonction lorsque vous l'associez à unVPC.

La première fois que vous attachez une fonction à une combinaison VPC de sous-réseau et de groupe de sécurité en particulier, Lambda crée un hyperplan. ENI Les autres fonctions de votre compte qui utilisent la même combinaison de sous-réseau et de groupe de sécurité peuvent également l'utiliser. ENI Dans la mesure du possible, Lambda réutilise les ressources existantes ENIs pour optimiser l'utilisation des ressources et minimiser la création de nouvelles ressources. ENIs Chaque Hyperplane ENI prend en charge jusqu'à 65 000 connexions/ports. Si le nombre de connexions dépasse cette limite, Lambda adapte le nombre de connexions ENIs automatiquement en fonction du trafic réseau et des exigences de simultanéité.

Pour les nouvelles fonctions, pendant que Lambda crée un hyperplanENI, votre fonction reste à l'état En attente et vous ne pouvez pas l'invoquer. Votre fonction passe à l'état actif uniquement lorsque l'hyperplan ENI est prêt, ce qui peut prendre plusieurs minutes. Pour les fonctions existantes, vous ne pouvez pas effectuer d'opérations supplémentaires ciblant la fonction, telles que la création de versions ou la mise à jour du code de la fonction, mais vous pouvez continuer à invoquer les versions précédentes de la fonction.

> **Note**
>
> Si une fonction Lambda reste inactive pendant 30 jours, Lambda récupère tout hyperplan inutilisé ENIs et définit l'état de la fonction sur inactif. La prochaine tentative d'appel échouera et la fonction reviendra à l'état En attente jusqu'à ce que Lambda ait terminé la création ou

l'allocation d'un hyperplan. ENI Pour plus d'informations sur les états des fonctions Lambda, consultez. the section called “États des fonctions”

## Utilisation des clés de IAM condition pour les VPC paramètres

Vous pouvez utiliser des clés de condition spécifiques à Lambda pour les VPC paramètres afin de fournir des contrôles d'autorisation supplémentaires pour vos fonctions Lambda. Par exemple, vous pouvez exiger que toutes les fonctions de votre organisation soient connectées à unVPC. Vous pouvez également spécifier les sous-réseaux et les groupes de sécurité que les utilisateurs de la fonction peuvent et ne peuvent pas utiliser.

Lambda prend en charge les clés de condition suivantes dans IAM les politiques :

- lambda : VpcIds — Autoriser ou refuser un ou plusieursVPCs.
- lambda : SubnetIds — Autoriser ou refuser un ou plusieurs sous-réseaux.
- lambda : SecurityGroupIds — Autoriser ou refuser un ou plusieurs groupes de sécurité.

Les API opérations CreateFunctionLambda prennent en UpdateFunctionConfigurationcharge ces clés de condition. Pour plus d'informations sur l'utilisation des clés de condition dans IAM les politiques, voir Éléments de IAM JSON politique : condition dans le guide de IAM l'utilisateur.

> **Tip**
>
> Si votre fonction inclut déjà une VPC configuration issue d'une API demande précédente, vous pouvez envoyer une `UpdateFunctionConfiguration` demande sans cette VPC configuration.

### Exemples de politiques avec des clés de condition pour VPC les paramètres

Les exemples suivants montrent comment utiliser les clés de condition pour VPC les paramètres. Après avoir créé une instruction de politique avec les restrictions souhaitées, ajoutez l’instruction de politique pour l’utilisateur ou le rôle cible.

Assurez-vous que les utilisateurs déploient uniquement les VPC fonctions connectées

Pour garantir que tous les utilisateurs déploient uniquement les fonctions VPC connectées, vous pouvez refuser les opérations de création et de mise à jour de fonctions qui n'incluent pas d'VPCidentifiant valide.

Notez que VPC l'ID n'est pas un paramètre d'entrée pour la `UpdateFunctionConfiguration` demande `CreateFunction` or. Lambda récupère la valeur de l'VPCID en fonction des paramètres du sous-réseau et du groupe de sécurité.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
           "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Refuser aux utilisateurs l'accès à VPCs des sous-réseaux ou à des groupes de sécurité spécifiques

Pour refuser aux utilisateurs l'accès à une condition spécifiqueVPCs, utilisez cette option `StringEquals` pour vérifier la valeur de la `lambda:VpcIds` condition. L'exemple suivant refuse aux utilisateurs l'accès à `vpc-1` et `vpc-2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceOutOfVPC",
```

```
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
            "lambda:VpcIds": ["vpc-1", "vpc-2"]
        }
      }
    }
```

Pour refuser aux utilisateurs l'accès à des sous-réseaux spécifiques, utilisez `StringEquals` pour vérifier la valeur de la condition `lambda:SubnetIds`. L'exemple suivant refuse aux utilisateurs l'accès à `subnet-1` et `subnet-2`.

```
{
      "Sid": "EnforceOutOfSubnet",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Pour refuser aux utilisateurs l'accès à des groupes de sécurité spécifiques, utilisez `StringEquals` pour vérifier la valeur de la condition `lambda:SecurityGroupIds`. L'exemple suivant refuse aux utilisateurs l'accès à `sg-1` et `sg-2`.

```
{
      "Sid": "EnforceOutOfSecurityGroups",
      "Action": [
          "lambda:CreateFunction",
```

```
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Permettre aux utilisateurs de créer et de mettre à jour des fonctions avec des VPC paramètres spécifiques

Pour autoriser les utilisateurs à accéder à une condition spécifiqueVPCs, utilisez cette option `StringEquals` pour vérifier la valeur de la `lambda:VpcIds` condition. L'exemple suivant permet aux utilisateurs d'accéder à `vpc-1` et `vpc-2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceStayInSpecificVpc",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
            "lambda:VpcIds": ["vpc-1", "vpc-2"]
        }
      }
    }
```

Pour permettre aux utilisateurs d'accéder à des sous-réseaux spécifiques, utilisez `StringEquals` pour vérifier la valeur de la condition `lambda:SubnetIds`. L'exemple suivant permet aux utilisateurs d'accéder à `subnet-1` et `subnet-2`.

```
{
      "Sid": "EnforceStayInSpecificSubnets",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Pour permettre aux utilisateurs d'accéder à des groupes de sécurité spécifiques, utilisez `StringEquals` pour vérifier la valeur de la condition `lambda:SecurityGroupIds`. L'exemple suivant permet aux utilisateurs d'accéder à `sg-1` et `sg-2`.

```
{
      "Sid": "EnforceStayInSpecificSecurityGroup",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

# Didacticiels VPC

Dans les didacticiels suivants, vous allez connecter une fonction Lambda aux ressources de votre. VPC

- Tutoriel : Utilisation d'une fonction Lambda pour accéder RDS à Amazon depuis un Amazon VPC
- Tutoriel : Configuration d'une fonction Lambda pour accéder ElastiCache à Amazon depuis un Amazon VPC

# Donner aux fonctions Lambda l'accès à une ressource d'un Amazon via un autre VPC compte

Vous pouvez donner à votre AWS Lambda fonction l'accès à une ressource d'Amazon VPC dans Amazon Virtual Private Cloud géré par un autre compte, sans exposer l'un ou l'autre VPC à Internet. Ce modèle d'accès vous permet de partager des données avec d'autres organisations utilisant AWS. Grâce à ce modèle d'accès, vous pouvez partager des données VPCs avec un niveau de sécurité et de performance supérieur à celui d'Internet. Configurez votre fonction Lambda pour utiliser une connexion de VPC peering Amazon pour accéder à ces ressources.

> **Warning**
>
> Lorsque vous autorisez l'accès entre les comptes ou VPCs que vous vérifiez que votre plan répond aux exigences de sécurité des organisations respectives qui gèrent ces comptes. Le respect des instructions de ce document aura une incidence sur le niveau de sécurité de vos ressources.

Dans ce didacticiel, vous connectez deux comptes à l'aide d'une connexion d'appairage à l'aide IPv4 de. Vous configurez une fonction Lambda qui n'est pas déjà connectée à un Amazon. VPC Vous configurez DNS la résolution pour connecter votre fonction à des ressources qui ne fournissent pas de données statiquesIPs. Pour adapter ces instructions à d'autres scénarios de peering, consultez le guide de VPCpeering.

## Prérequis

Pour permettre à une fonction Lambda d'accéder à une ressource d'un autre compte, vous devez avoir :

- Une fonction Lambda, configurée pour s'authentifier auprès de votre ressource puis lire à partir de celle-ci.
- Une ressource d'un autre compte, tel qu'un RDS cluster Amazon, disponible via AmazonVPC.
- Informations d'identification du compte de votre fonction Lambda et du compte de votre ressource. Si vous n'êtes pas autorisé à utiliser le compte de votre ressource, contactez un utilisateur autorisé pour préparer ce compte.
- Autorisation de créer et de mettre à jour une VPC (et les VPC ressources Amazon de support) à associer à votre fonction Lambda.

- Autorisation de mettre à jour le rôle d'exécution et VPC la configuration de votre fonction Lambda.
- Autorisation de créer une connexion de VPC peering dans le compte de votre fonction Lambda.
- Autorisation d'accepter une connexion de VPC peering sur le compte de votre ressource.
- Autorisation de mettre à jour la configuration de vos ressources VPC (et des VPC ressources Amazon de support).
- Autorisation d'invoquer votre fonction Lambda.

## Créez un Amazon VPC dans le compte de votre fonction

Créez un AmazonVPC, des sous-réseaux, des tables de routage et un groupe de sécurité dans le compte de votre fonction Lambda.

Pour créer unVPC, des sous-réseaux et d'autres VPC ressources à l'aide de la console

1. Ouvrez la VPC console Amazon à l'adresse https://console.aws.amazon.com/vpc/.
2. Sur le tableau de bord, choisissez Create VPC.
3. Pour le IPv4CIDRbloc, fournissez un CIDR bloc privé. Votre CIDR bloc ne doit pas chevaucher les blocs utilisés dans celui de votre ressourceVPC. Ne choisissez pas un bloc que vos ressources VPC utilisent à attribuer IPs aux ressources ou un bloc déjà défini dans les tables de routage de vos ressources. VPC Pour plus d'informations sur la définition des CIDR blocs appropriés, consultez la section VPCCIDRblocs.
4. Choisissez PersonnaliserAZs.
5. Sélectionnez la même ressource AZs que votre ressource.
6. Pour Nombre de sous-réseaux publics, choisissez 0.
7. Pour les VPCpoints de terminaison, choisissez Aucun.
8. Choisissez Créer VPC.

## Accordez VPC des autorisations au rôle d'exécution de votre fonction

AWSLambdaVPCAccessExecutionRoleAttachez-vous au rôle d'exécution de votre fonction pour lui permettre de se connecter àVPCs.

Pour accorder VPC des autorisations au rôle d'exécution de votre fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez le nom de votre fonction .
3. Choisissez Configuration.
4. Choisissez Autorisations.
5. Sous Nom du rôle, choisissez le rôle d'exécution.
6. Dans la section Politiques d'autorisations, choisissez Ajouter des autorisations.
7. Dans la liste déroulante, choisissez Joindre des politiques.
8. Dans la zone de recherche, saisissez `AWSLambdaVPCAccessExecutionRole`.
9. À gauche du nom de la politique, cochez la case.
10. Choisissez Add permissions (Ajouter des autorisations).

Pour associer votre fonction à votre Amazon VPC

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction .
3. Choisissez l'onglet Configuration, puis choisissez VPC.
4. Choisissez Modifier.
5. Sous VPC, sélectionnez votre VPC
6. Sous Sous-réseaux, choisissez vos sous-réseaux.
7. Sous Groupes de sécurité, choisissez le groupe de sécurité par défaut pour votreVPC.
8. Choisissez Save (Enregistrer).

## Création d'une demande VPC de connexion peering

Créez une demande de connexion VPC peering depuis celle de votre fonction VPC (le demandeurVPC) vers celle de votre ressource VPC (l'accepteur). VPC

Pour demander une connexion VPC de peering depuis votre fonction VPC

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Peering connections (Connexions d'appairage).
3. Choisissez Create peering connection (Créer une connexion d'appairage).
4. Pour VPCID (demandeur), sélectionnez celui de votre fonction. VPC

5. Pour ID de compte, entrez l'ID du compte de votre ressource.
6. Pour VPCID (Accepter), entrez celui de votre ressourceVPC.

## Préparez le compte de votre ressource

Pour créer votre connexion d'appairage et préparer celle de votre ressource VPC à utiliser la connexion, connectez-vous au compte de votre ressource avec un rôle qui détient les autorisations répertoriées dans les prérequis. Les étapes de connexion peuvent être différentes en fonction de la manière dont le compte est sécurisé. Pour plus d'informations sur la façon de se connecter à un AWS compte, consultez le Guide de l'utilisateur de AWS connexion. Dans le compte de votre ressource, effectuez les procédures suivantes.

Pour accepter la demande de connexion VPC d'appairage

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Peering connections (Connexions d'appairage).
3. Sélectionnez la connexion d'VPCappairage en attente (le statut est en attente d'acceptation).
4. Choisissez Actions.
5. Dans la liste déroulante, choisissez Accepter la demande.
6. Lorsque vous êtes invité à confirmer l'opération, choisissez Accepter la demande.
7. Choisissez Modifier mes tables de routage maintenant pour ajouter un itinéraire à la table de routage principale pour vous VPC afin de pouvoir envoyer et recevoir du trafic via la connexion d'appairage.

Inspectez les tables de routage pour les ressourcesVPC. L'itinéraire généré par Amazon VPC peut ne pas établir de connectivité, selon la configuration VPC de votre ressource. Vérifiez l'absence de conflits entre le nouvel itinéraire et la configuration existante duVPC. Pour plus d'informations sur le dépannage, consultez la section Résoudre les problèmes liés à une connexion de VPC peering dans le guide de VPCpeering Amazon Virtual Private Cloud.

Pour mettre à jour le groupe de sécurité de votre ressource

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le panneau de navigation, choisissez Groupes de sécurité.
3. Sélectionnez le groupe de sécurité pour votre ressource.

4. Choisissez Actions.
5. Dans la liste déroulante, choisissez Modifier les règles entrantes.
6. Choisissez Ajouter une règle.
7. Pour Source, entrez l'ID de compte et l'ID de groupe de sécurité de votre fonction, séparés par une barre oblique (par exemple, 111122223333/sg-1a2b3c4d).
8. Choisissez Edit outbound rules (Modifier les règles sortantes).
9. Vérifiez si le trafic sortant est restreint. VPCLes paramètres par défaut autorisent tout le trafic sortant. Si le trafic sortant est restreint, passez à l'étape suivante.
10. Choisissez Ajouter une règle.
11. Pour Destination, entrez l'ID de compte et l'ID de groupe de sécurité de votre fonction, séparés par une barre oblique (par exemple, 111122223333/sg-1a2b3c4d).
12. Sélectionnez Enregistrer les règles.

Pour activer DNS la résolution de votre connexion de peering

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Peering connections (Connexions d'appairage).
3. Sélectionnez votre connexion de peering.
4. Choisissez Actions.
5. Choisissez Modifier DNS les paramètres.
6. Sous DNSRésolution de l'acceptation, sélectionnez Autoriser le demandeur VPC à convertir les VPC hôtes accepteurs en adresse IP privée. DNS
7. Sélectionnez Enregistrer les modifications.

## Mettre à jour VPC la configuration dans le compte de votre fonction

Connectez-vous au compte de votre fonction, puis mettez à jour la VPC configuration.

Pour ajouter un itinéraire pour votre connexion de VPC peering

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Route tables (Tables de routage).
3. Cochez la case à côté du nom de la table de routage pour le sous-réseau que vous avez associé à votre fonction.

4. Choisissez Actions.
5. Choisissez Edit routes (Modifier des routes).
6. Choisissez Ajouter une route.
7. Dans Destination, entrez le CIDR bloc correspondant à votre ressourceVPC.
8. Pour Target, sélectionnez votre connexion d'VPCappairage.
9. Sélectionnez Enregistrer les modifications.

Pour plus d'informations sur les considérations que vous pouvez rencontrer lors de la mise à jour de vos tables de routage, consultez Mettre à jour vos tables de routage pour une connexion d'VPCappairage.

Pour mettre à jour le groupe de sécurité de votre fonction Lambda

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le panneau de navigation, choisissez Groupes de sécurité.
3. Choisissez Actions.
4. Choisissez Modifier les règles entrantes.
5. Choisissez Ajouter une règle.
6. Pour Source, entrez l'ID de compte et l'ID du groupe de sécurité de votre ressource, séparés par une barre oblique (par exemple, 111122223333/sg-1a2b3c4d).
7. Sélectionnez Enregistrer les règles.

Pour activer DNS la résolution de votre connexion de peering

1. Ouvrez le https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Peering connections (Connexions d'appairage).
3. Sélectionnez votre connexion de peering.
4. Choisissez Actions.
5. Choisissez Modifier DNS les paramètres.
6. Sous DNSRésolution du demandeur, sélectionnez Autoriser l'accepteur VPC à résoudre les VPC hôtes DNS du demandeur en adresse IP privée.
7. Sélectionnez Enregistrer les modifications.

# Test de votre fonction

Pour créer un événement de test et inspecter le résultat de votre fonction

1. Dans le volet Source du code, choisissez Test.
2. Sélectionnez Créer un événement.
3. Dans le JSON panneau Evénement, remplacez les valeurs par défaut par une entrée adaptée à votre fonction Lambda.
4. Sélectionnez Invoquer .
5. Dans l'onglet Résultats de l'exécution, vérifiez que Response contient le résultat attendu.

En outre, vous pouvez consulter les journaux de votre fonction pour vérifier qu'ils correspondent à vos attentes.

Pour consulter les enregistrements d'invocation de votre fonction dans Logs CloudWatch

1. Choisissez l’onglet Monitor (Surveiller).
2. Choisissez Afficher CloudWatch les journaux.
3. Dans l'onglet Flux de journaux, choisissez le flux de journaux pour l'invocation de votre fonction.
4. Vérifiez que vos journaux sont conformes à vos attentes.

# Activer l'accès à Internet pour les VPC fonctions Lambda connectées

Par défaut, les fonctions Lambda s'exécutent dans un environnement géré par Lambda disposant d'un accès à InternetVPC. Pour accéder aux ressources VPC d'un compte, vous pouvez ajouter une VPC configuration à une fonction. Cela limite la fonction aux ressources qui s'y trouventVPC, sauf si elles VPC ont accès à Internet. Cette page explique comment fournir un accès Internet aux fonctions Lambda VPC connectées.

## Je n'en ai pas VPC encore

### Créez le VPC

Le VPCflux de travail Create crée toutes les VPC ressources nécessaires à une fonction Lambda pour accéder à l'Internet public à partir d'un sous-réseau privé, y compris les sous-réseaux, la NAT passerelle, la passerelle Internet et les entrées de table de routage.

Pour créer le VPC

1. Ouvrez la VPC console Amazon à l'adresse https://console.aws.amazon.com/vpc/.
2. Sur le tableau de bord, choisissez Create VPC.
3. Pour que Resources crée, choisissez VPCet plus encore.
4. Configurez le VPC
    a. Pour la génération automatique du tag Name, entrez un nom pour leVPC.
    b. Pour le IPv4CIDRblocage, vous pouvez conserver la suggestion par défaut ou saisir le CIDR bloc requis par votre application ou votre réseau.
    c. Si votre application communique à l'aide d'IPv6adresses, choisissez IPv6CIDRBloquer, bloc fourni par Amazon IPv6 CIDR.
5. Configurer les sous-réseaux
    a. Pour Nombre de zones de disponibilité, choisissez 2. Nous en recommandons au moins deux AZs pour une haute disponibilité.
    b. Pour Number of public subnets (Nombre de sous-réseaux publics), choisissez 2.
    c. Pour Number of private subnets (Nombre de sous-réseaux privés), choisissez 2.

   d. Vous pouvez conserver le CIDR bloc par défaut pour le sous-réseau public, ou bien vous pouvez développer Personnaliser les CIDR blocs de sous-réseau et entrer un CIDR bloc. Pour plus d'informations, consultez la section CIDRBlocs de sous-réseaux.
6. Pour les NATpasserelles, choisissez-en une par AZ pour améliorer la résilience.
7. Pour la passerelle Internet de sortie uniquement, choisissez Oui si vous avez choisi d'inclure un IPv6 CIDR bloc.
8. Pour les VPCpoints de terminaison, conservez la valeur par défaut (S3 Gateway). Cette option est gratuite. Pour plus d'informations, consultez la section Types de VPC points de terminaison pour Amazon S3.
9. Pour les DNSoptions, conservez les paramètres par défaut.
10. Choisissez Create VPC.

Configurer la fonction Lambda

Pour configurer une fonction VPC lors de la création d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Sous Informations de base, dans Nom de fonction, entrez un nom pour votre fonction.
4. Développez Advanced settings (Paramètres avancés).
5. Sélectionnez Activer VPC, puis choisissez unVPC.
6. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.
7. Pour les sous-réseaux, sélectionnez tous les sous-réseaux privés. Les sous-réseaux privés peuvent accéder à Internet via la NAT passerelle. La connexion d'une fonction à un sous-réseau public ne lui donne pas accès à Internet.

   > **Note**
   >
   > Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR
8. Pour les groupes de sécurité, sélectionnez un groupe de sécurité qui autorise le trafic sortant.
9. Sélectionnez Create function (Créer une fonction).

Lambda crée automatiquement un rôle d'exécution avec la politique AWSLambdaVPCAccessExecutionRole AWS gérée. Les autorisations définies dans cette politique ne sont requises que pour créer des interfaces réseau élastiques pour la VPC configuration, et non pour invoquer votre fonction. Pour appliquer les autorisations de moindre privilège, vous pouvez supprimer la AWSLambdaVPCAccessExecutionRolepolitique de votre rôle d'exécution après avoir créé la fonction et VPC la configuration. Pour de plus amples informations, veuillez consulter IAMAutorisations requises.

Pour configurer un VPC pour une fonction existante

Pour ajouter une VPC configuration à une fonction existante, le rôle d'exécution de la fonction doit être autorisé à créer et à gérer des interfaces réseau élastiques. La politique AWSLambdaVPCAccessExecutionRole AWS gérée inclut les autorisations requises. Pour appliquer les autorisations de moindre privilège, vous pouvez supprimer la AWSLambdaVPCAccessExecutionRolepolitique de votre rôle d'exécution après avoir créé la VPC configuration.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez l'onglet Configuration, puis choisissez VPC.
4. Sous VPC, choisissez Modifier.
5. Sélectionnez leVPC.
6. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.
7. Pour les sous-réseaux, sélectionnez tous les sous-réseaux privés. Les sous-réseaux privés peuvent accéder à Internet via la NAT passerelle. La connexion d'une fonction à un sous-réseau public ne lui donne pas accès à Internet.

   > **Note**
   >
   > Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR

8. Pour les groupes de sécurité, sélectionnez un groupe de sécurité qui autorise le trafic sortant.
9. Choisissez Save (Enregistrer).

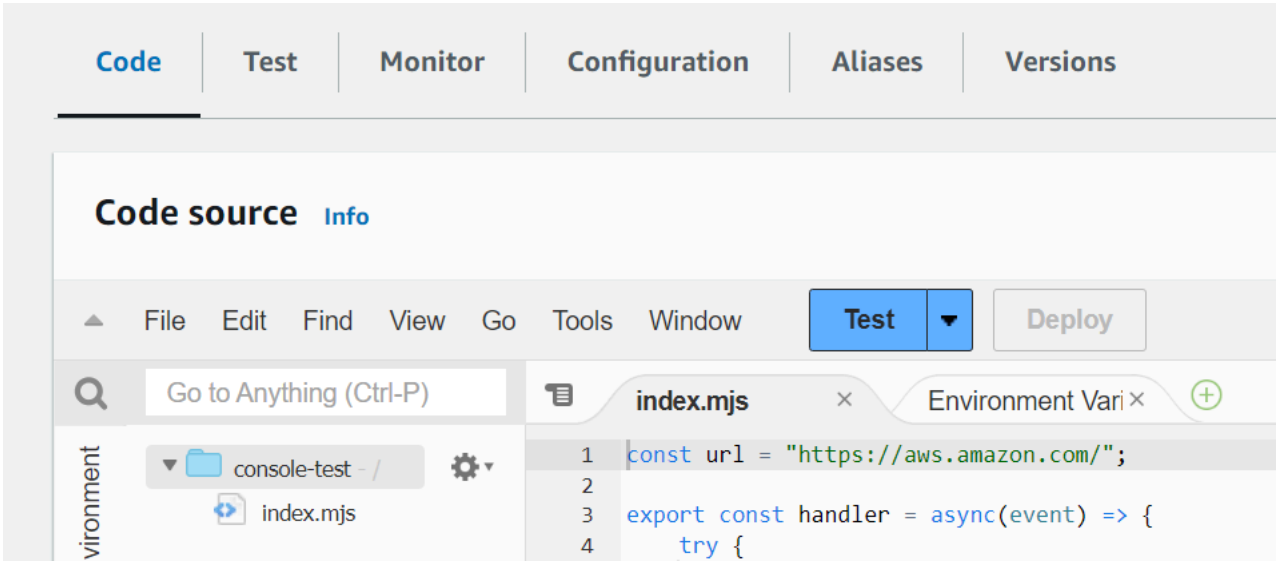
### Tester la fonction

Utilisez l'exemple de code suivant pour vérifier que votre fonction VPC connectée peut accéder à l'Internet public. En cas de succès, le code renvoie un code d'`200`état. En cas d'échec, la fonction expire.

#### Node.js

Cet exemple utilise`fetch`, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

1. Dans le volet Source du code de la console Lambda, collez le code suivant dans le fichier index.mjs. La fonction envoie une HTTP GET demande à un point de terminaison public et renvoie le code de HTTP réponse pour tester si la fonction a accès à l'Internet public.

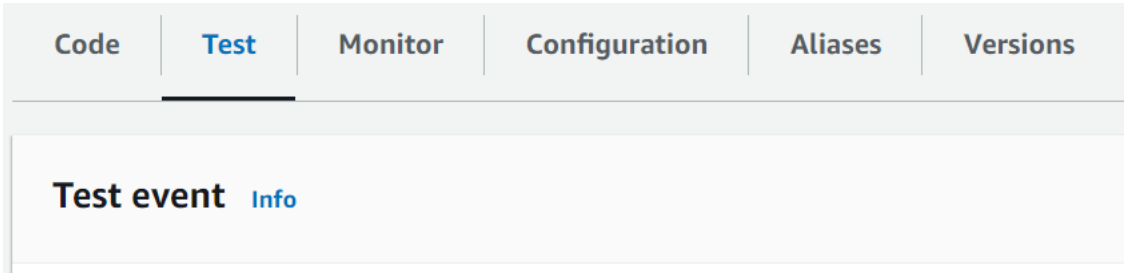



Example — HTTP demande avec async/await

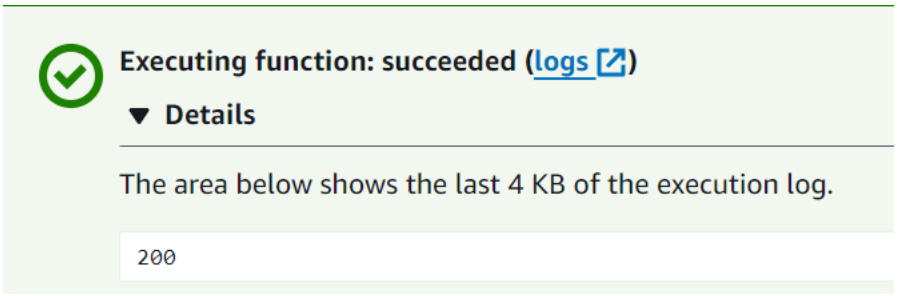
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        // fetch is available with Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
```

```
        return 500;
    }
};
```

2. Choisissez Deploy (Déployer).
3. Choisissez l'onglet Test.




4. Sélectionnez Tester).
5. La fonction renvoie un code d'`200`état. Cela signifie que la fonction dispose d'un accès Internet sortant.

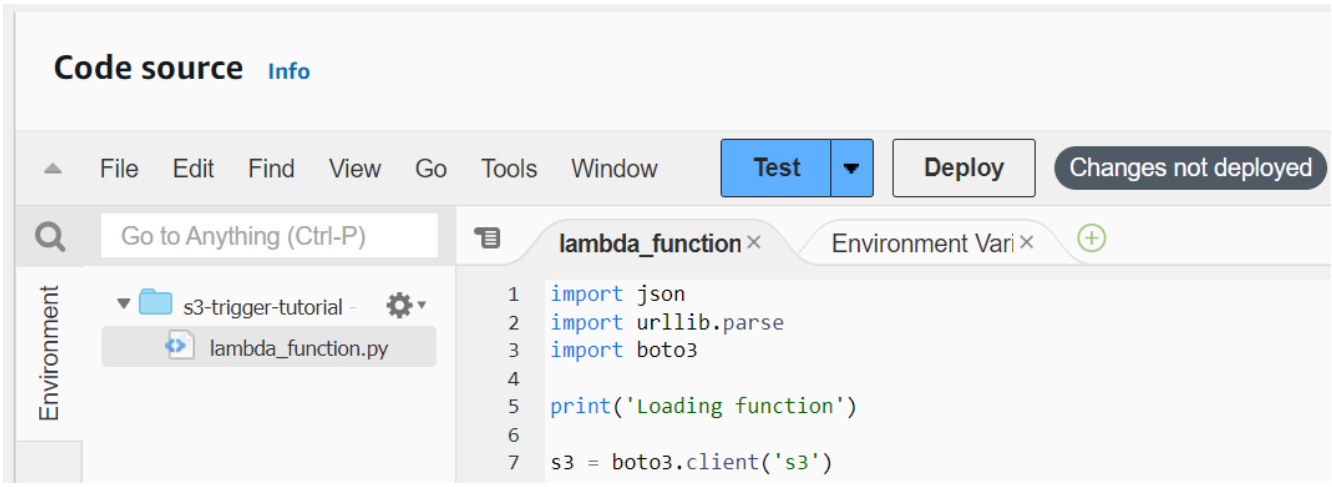



Si la fonction ne parvient pas à accéder à l'Internet public, le message d'erreur suivant s'affiche :

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

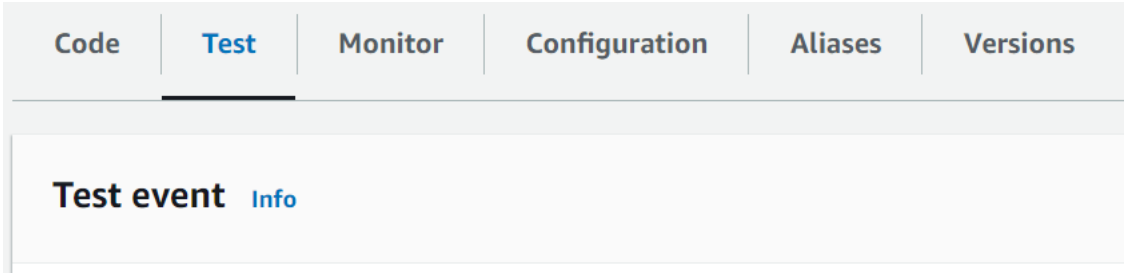
Python

1. Dans le volet Source du code de la console Lambda, collez le code suivant dans le fichier lambda_function.py. La fonction envoie une HTTP GET demande à un point de terminaison public et renvoie le code de HTTP réponse pour tester si la fonction a accès à l'Internet public.

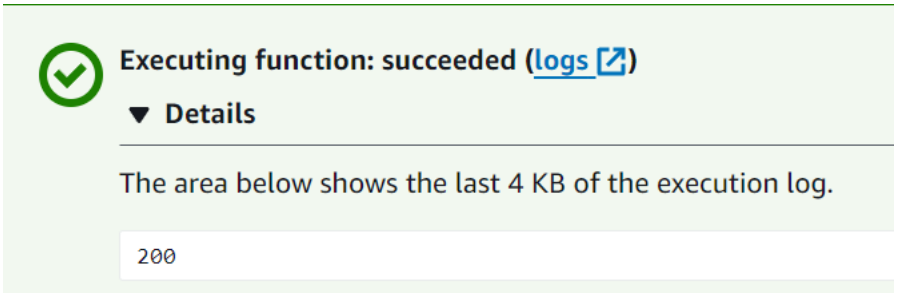



```
import urllib.request

def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

2. Choisissez Deploy (Déployer).
3. Choisissez l'onglet Test.




4. Sélectionnez Tester).
5. La fonction renvoie un code d'`200`état. Cela signifie que la fonction dispose d'un accès Internet sortant.




Si la fonction ne parvient pas à accéder à l'Internet public, le message d'erreur suivant s'affiche :

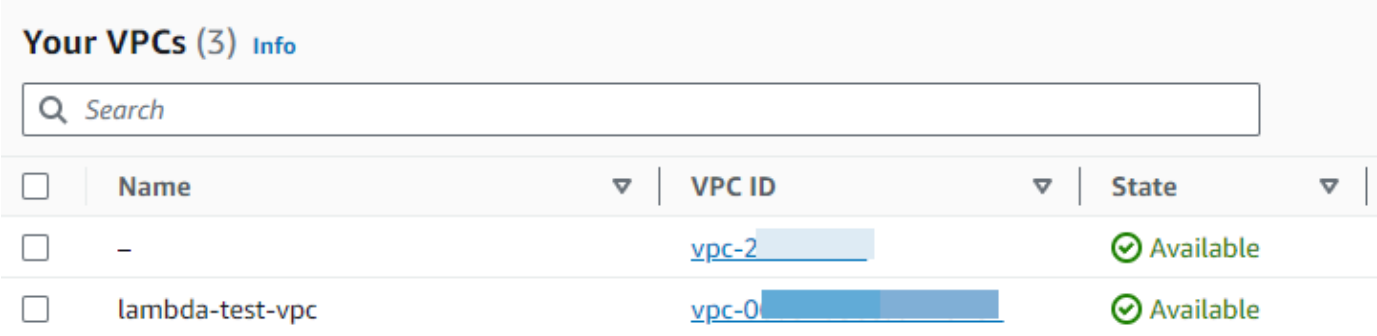
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

## J'ai déjà un VPC

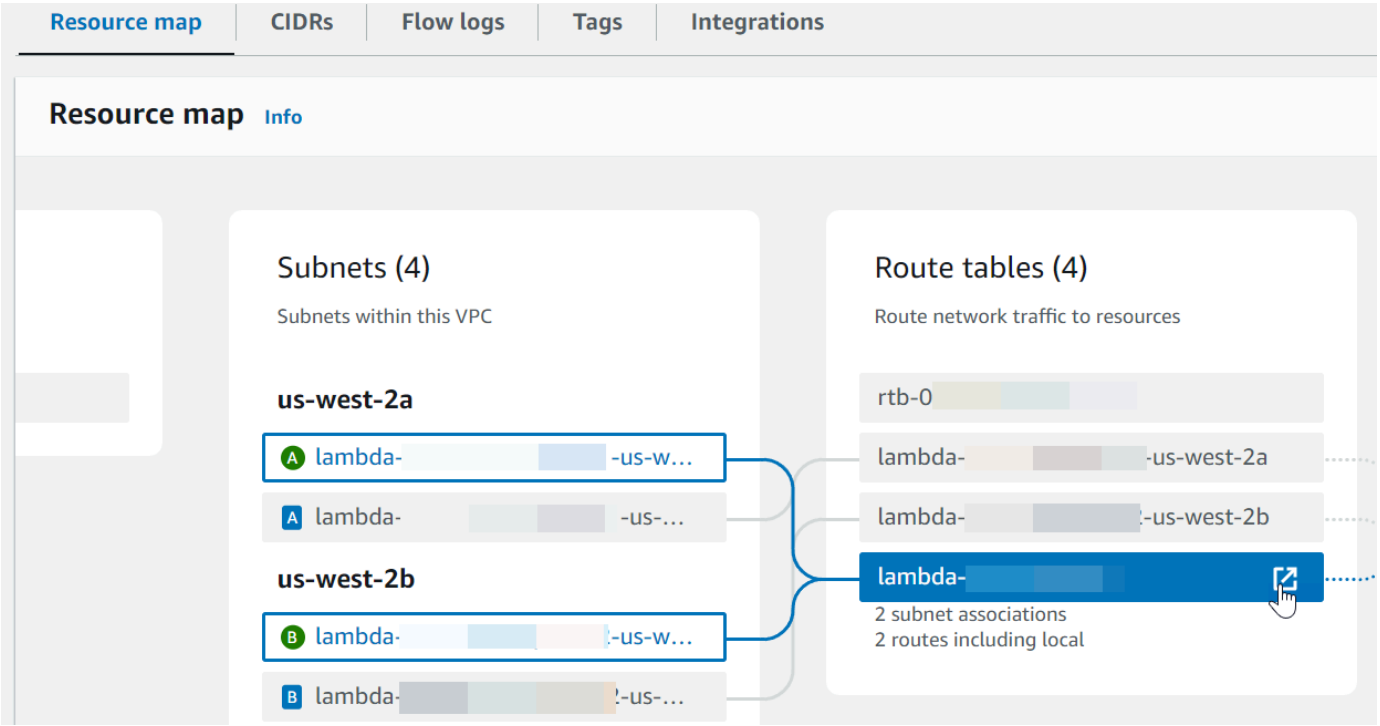
Si vous en avez déjà un VPC mais que vous devez configurer l'accès public à Internet pour une fonction Lambda, procédez comme suit. Cette procédure suppose que vous disposez VPC d'au moins deux sous-réseaux. Si vous n'avez pas deux sous-réseaux, consultez la section Créer un sous-réseau dans le guide de VPCl'utilisateur Amazon.

Vérifiez la configuration de la table de routage

1. Ouvrez la VPC console Amazon à l'adresse https://console.aws.amazon.com/vpc/.
2. Choisissez l'VPCidentifiant.




3. Faites défiler la page jusqu'à la section Carte des ressources. Notez les mappages des tables de routage. Ouvrez chaque table de routage mappée à un sous-réseau.

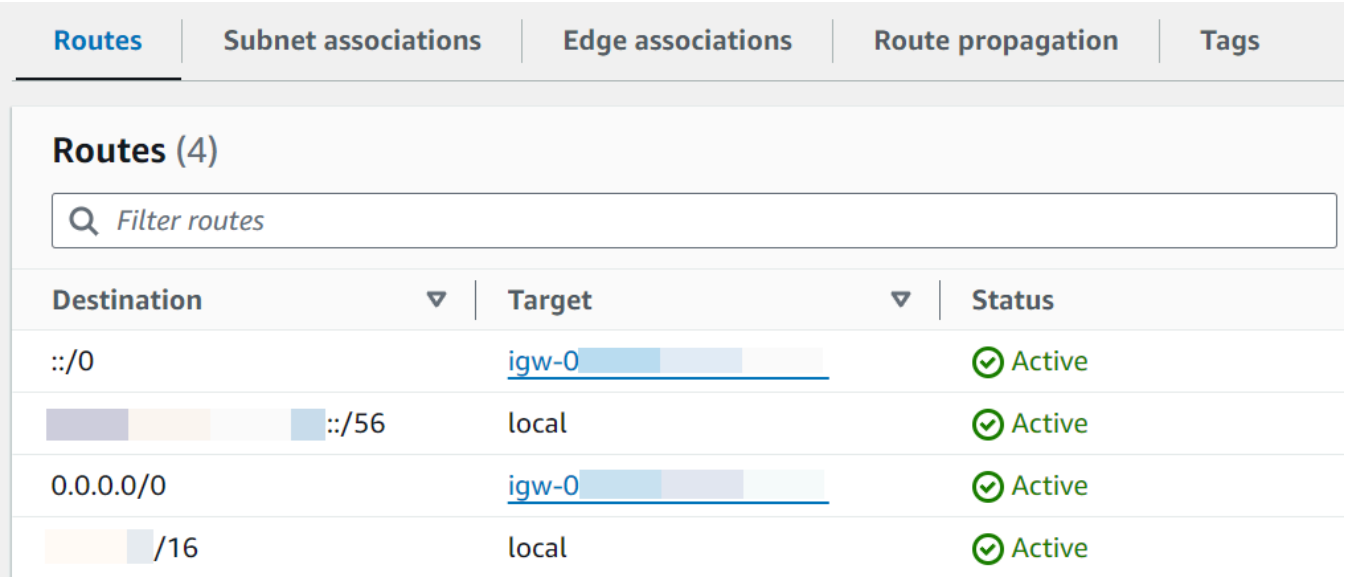



4. Faites défiler la page jusqu'à l'onglet Routes. Passez en revue les itinéraires pour déterminer si l'une des conditions suivantes est vraie. Chacune de ces exigences doit être satisfaite par une table de routage distincte.

   - Le trafic Internet (`0.0.0.0/0`pourIPv4, `::/0` pourIPv6) est acheminé vers une passerelle Internet (). `igw-xxxxxxxxxx` Cela signifie que le sous-réseau associé à la table de routage est un sous-réseau public.

     > **Note**
     >
     > Si votre sous-réseau n'a pas de IPv6 CIDR bloc, vous ne verrez que l'IPv4itinéraire (`0.0.0.0/0`).

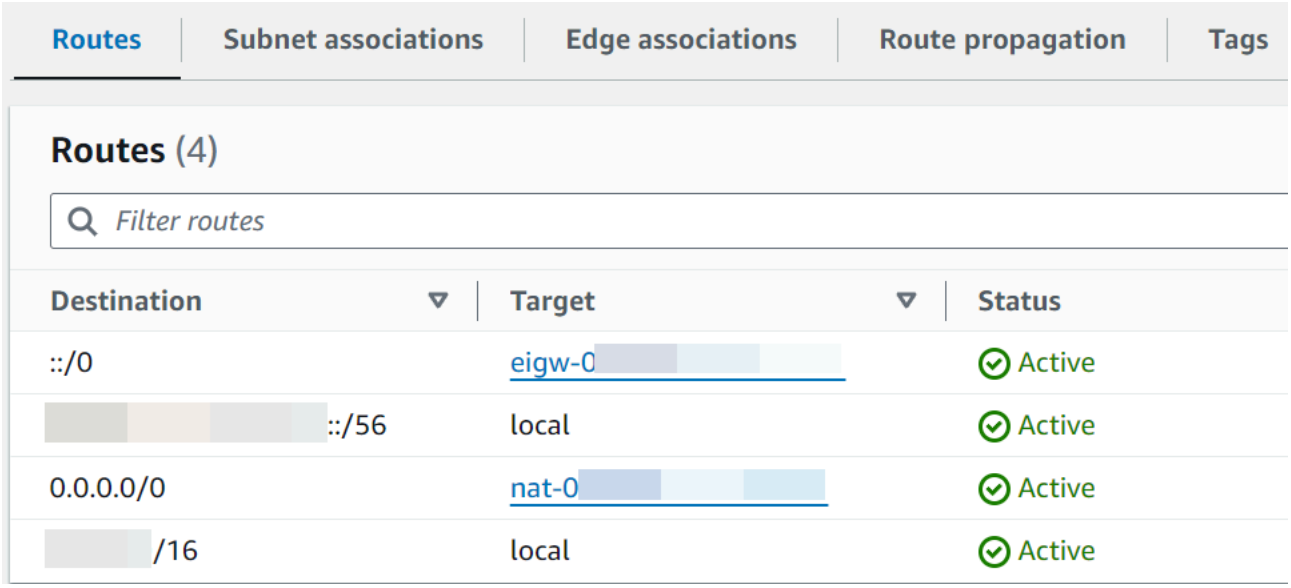
Example table de routage du sous-réseau public




- Le trafic Internet pour IPv4 (`0.0.0.0/0`) est acheminé vers une NAT passerelle (`nat-xxxxxxxxxx`) associée à un sous-réseau public. Cela signifie que le sous-réseau est un sous-réseau privé qui peut accéder à Internet via la NAT passerelle.

> **Note**
>
> Si votre sous-réseau comporte un IPv6 CIDR bloc, la table de routage doit également acheminer le IPv6 trafic Internet (`::/0`) vers une passerelle Internet de sortie uniquement (). `eigw-xxxxxxxxxx` Si votre sous-réseau n'a pas de IPv6 CIDR bloc, vous ne verrez que l'IPv4itinéraire (`0.0.0.0/0`).

Example table de routage de sous-réseau privé




5. Répétez l'étape précédente jusqu'à ce que vous ayez examiné chaque table de routage associée à un sous-réseau dans votre répertoire VPC et confirmé que vous disposez d'une table de routage avec une passerelle Internet et d'une table de routage avec une NAT passerelle.

   Si vous n'avez pas deux tables de routage, l'une avec un itinéraire vers une passerelle Internet et l'autre avec un itinéraire vers une NAT passerelle, procédez comme suit pour créer les ressources manquantes et les entrées de table de routage.

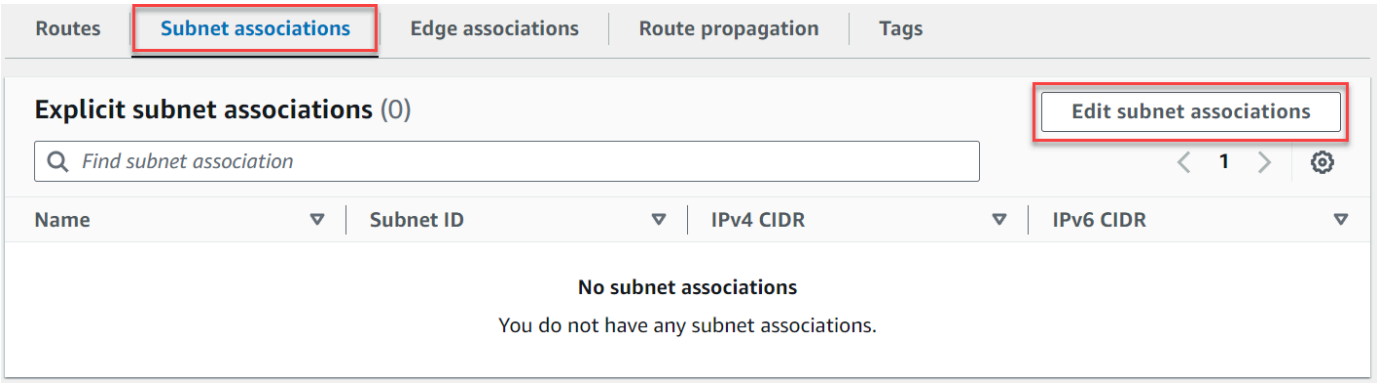
## Création d’une table de routage

Procédez comme suit pour créer une table de routage et l'associer à un sous-réseau.

Pour créer une table de routage personnalisée à l'aide de la VPC console Amazon

1. Ouvrez la VPC console Amazon à l'adresse https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation, choisissez Route tables (Tables de routage).
3. Choisissez Créer une table de routage.
4. (Facultatif) Pour Nom, entrez un nom pour votre table de routage.
5. Pour VPC, choisissez votreVPC.
6. (Facultatif) Pour ajouter une identification, choisissez Add new tag (Ajouter une identification) et saisissez la clé et la valeur de l'identification.

7. Choisissez Créer une table de routage.
8. Sur l'onglet Associations de sous-réseau, choisissez Modifier les associations de sous-réseau.



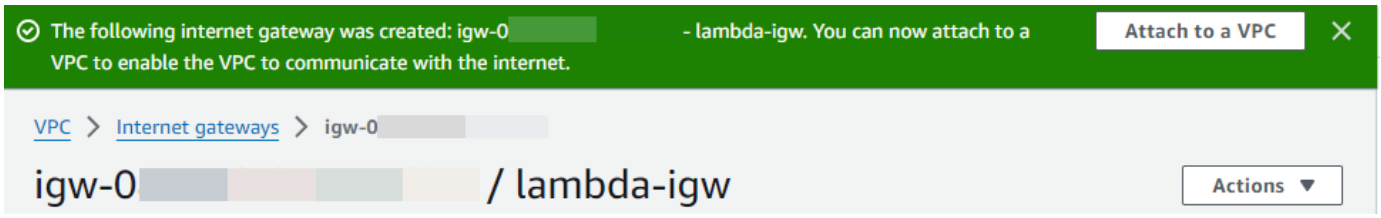

9. Sélectionnez la case à cocher pour le sous-réseau à associer à la table de routage.
10. Choisissez Save associations (Enregistrer les associations).

## Création d'une passerelle Internet

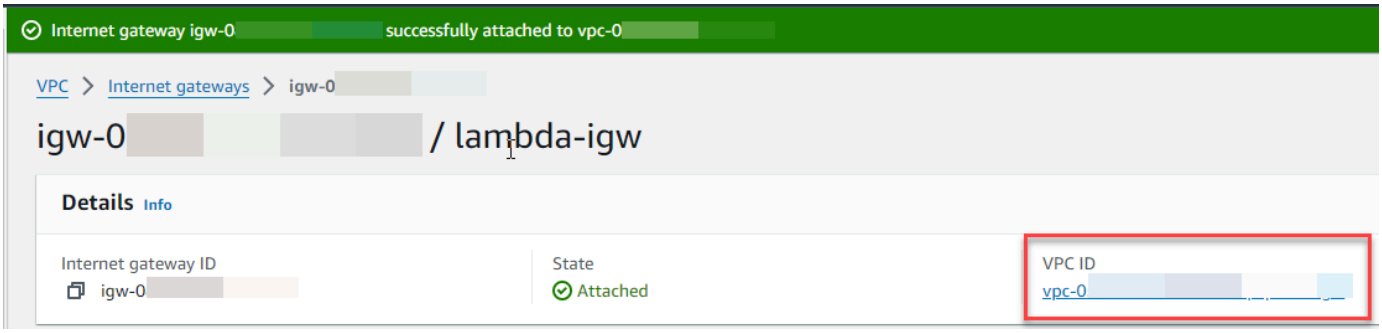
Procédez comme suit pour créer une passerelle Internet, l'associer à votre VPC et l'ajouter à la table de routage de votre sous-réseau public.

Pour créer une passerelle Internet

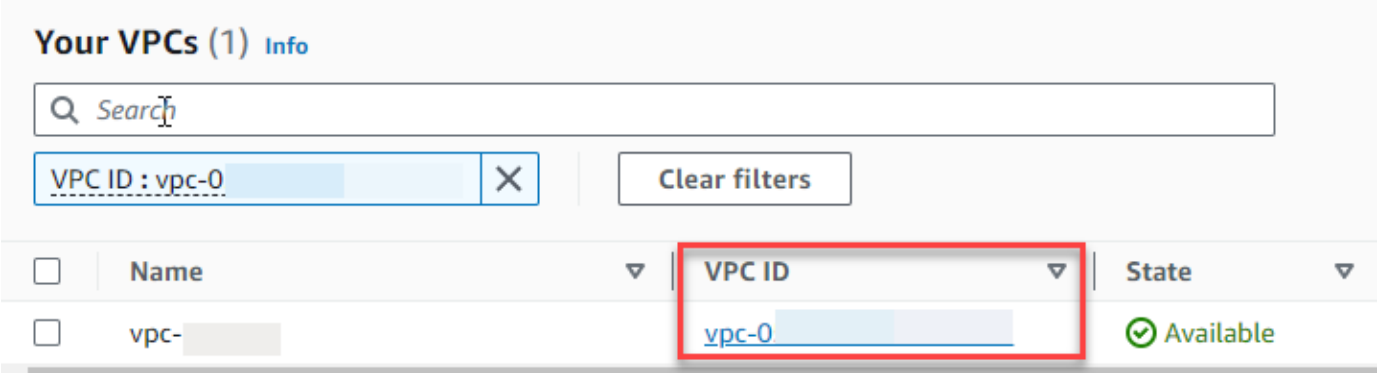
1. Ouvrez la VPC console Amazon à l'adresse https://console.aws.amazon.com/vpc/.
2. Dans le panneau de navigation, choisissez Passerelles Internet.
3. Choisissez Créer une passerelle Internet.
4. (Facultatif) Saisissez un nom pour votre passerelle Internet.
5. (Facultatif) Pour ajouter une identification, choisissez Add new tag (Ajouter une identification) et saisissez la clé et la valeur de l'identification.
6. Choisissez Créer une passerelle Internet.
7. Choisissez Joindre VPC à une dans la bannière en haut de l'écran, sélectionnez une option disponibleVPC, puis choisissez Joindre une passerelle Internet.



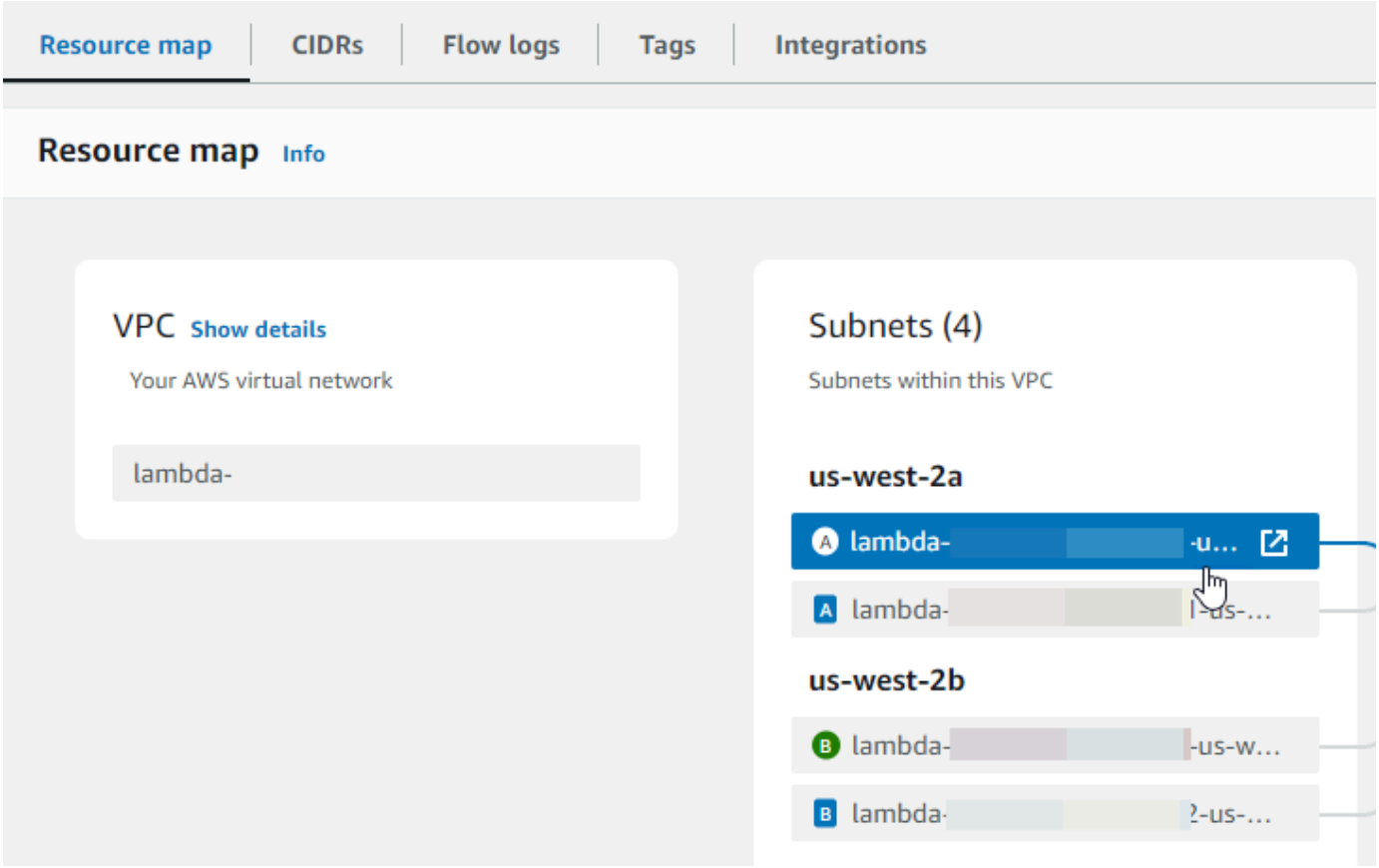

8. Choisissez l'VPCidentifiant.



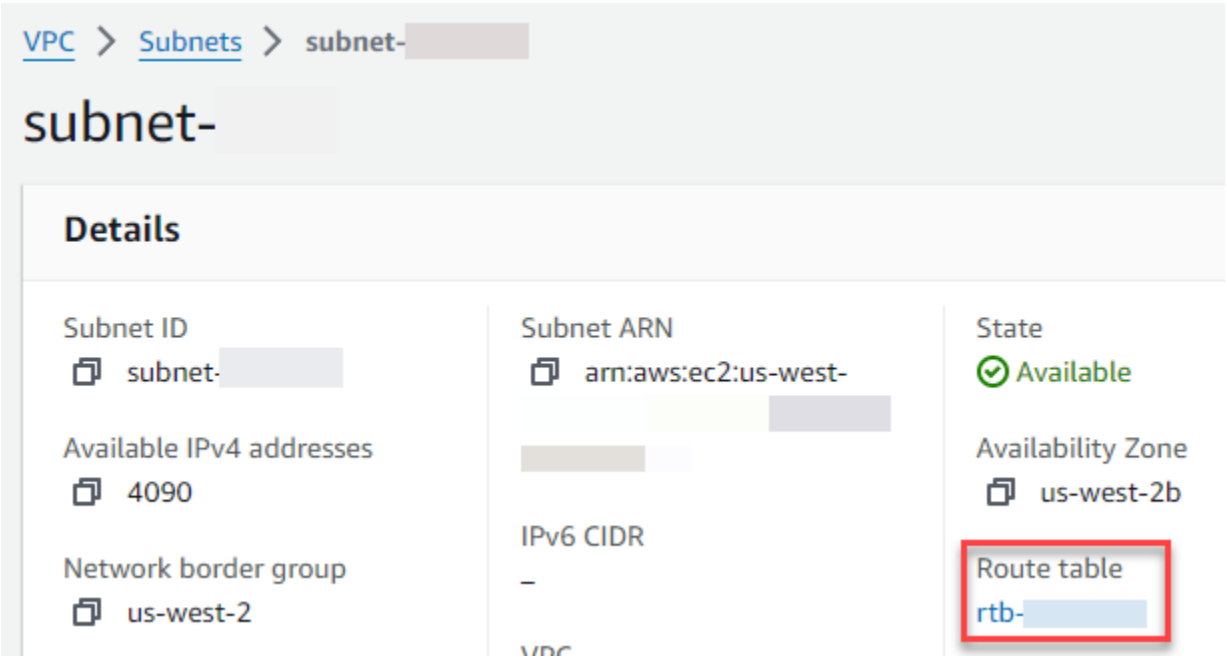

9. Choisissez à nouveau l'VPCID pour ouvrir la page de VPC détails.



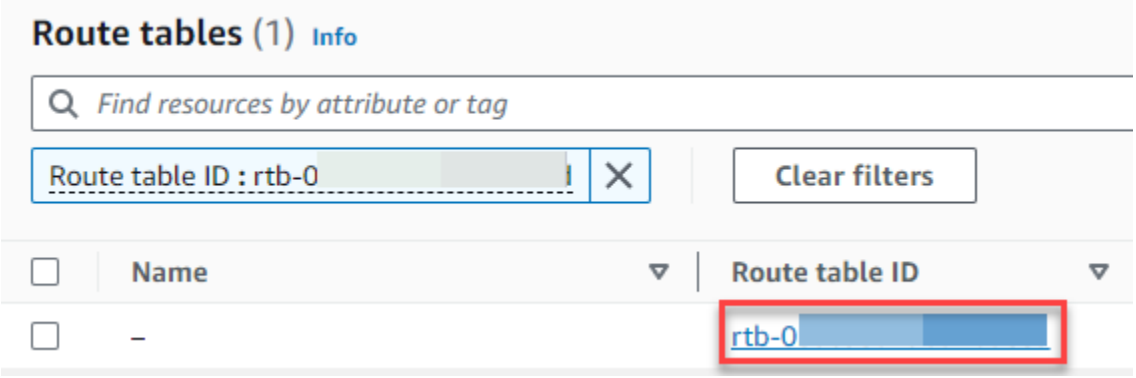

10. Faites défiler la page jusqu'à la section Carte des ressources, puis choisissez un sous-réseau. Les détails du sous-réseau sont affichés dans un nouvel onglet.



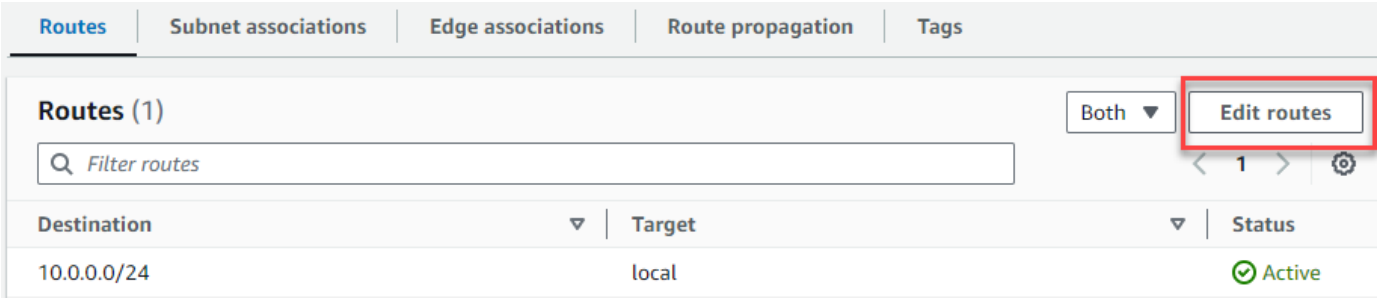

11. Cliquez sur le lien sous Table de routage.



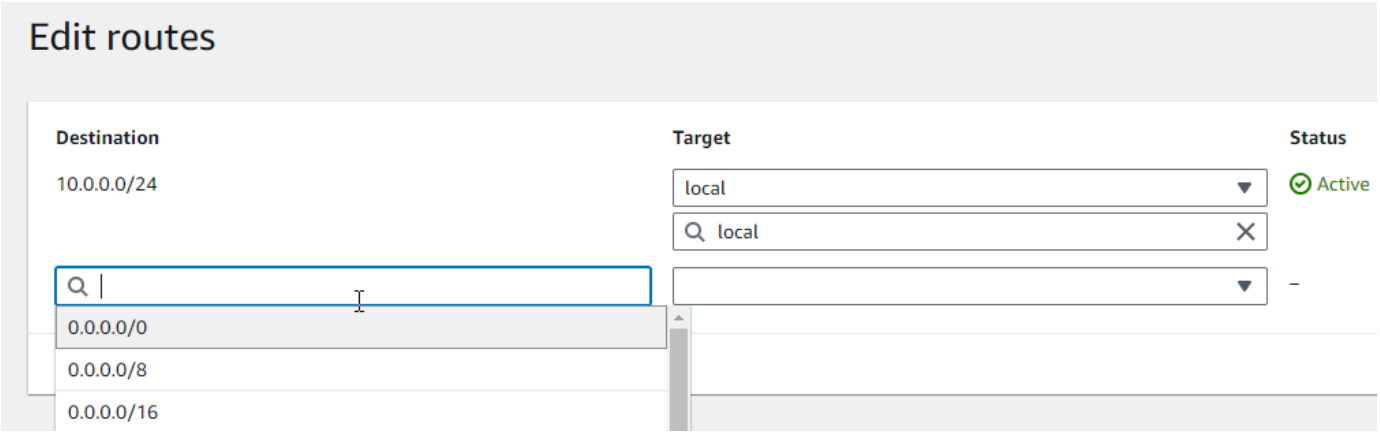

12. Choisissez l'ID de la table de routage pour ouvrir la page des détails de la table de routage.



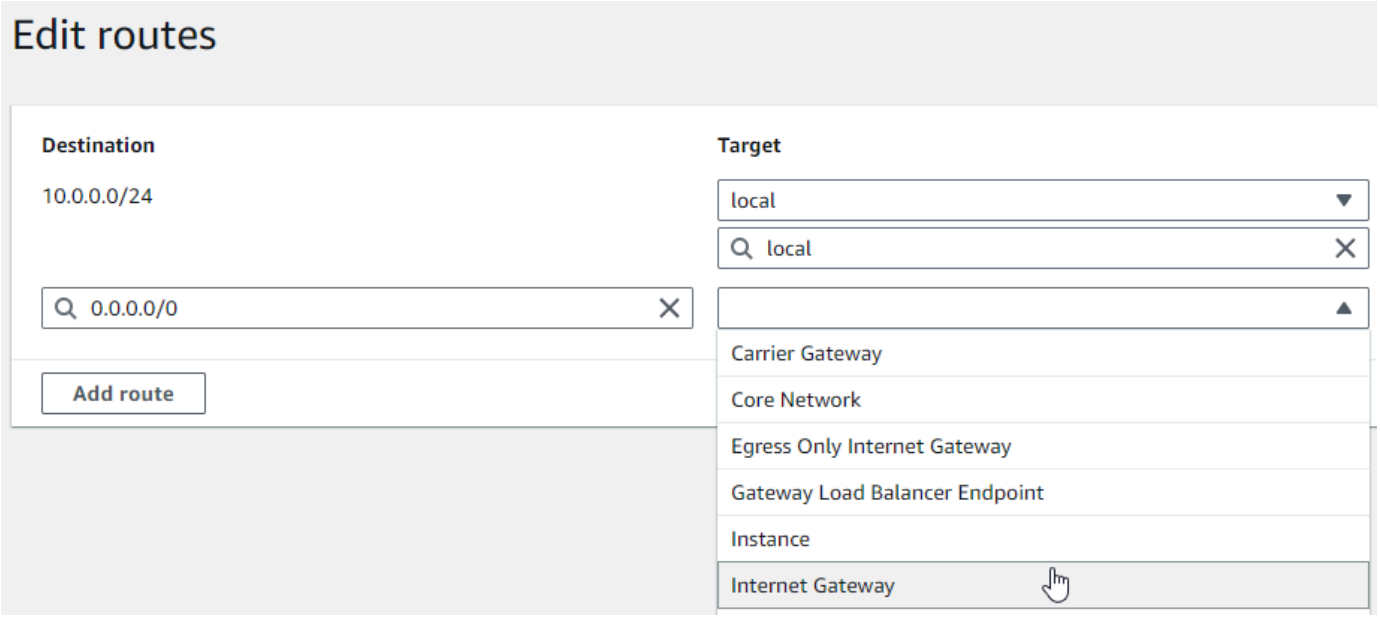

13. Sous Itinéraires, choisissez Modifier les itinéraires.




14. Choisissez Ajouter un itinéraire, puis entrez `0.0.0.0/0` dans le champ Destination.




15. Pour Target, sélectionnez Passerelle Internet, puis choisissez la passerelle Internet que vous avez créée précédemment. Si votre sous-réseau est IPv6 CIDR bloqué, vous devez également ajouter une route `::/0` vers la même passerelle Internet.




16. Sélectionnez Enregistrer les modifications.

### Création d'une NAT passerelle

Procédez comme suit pour créer une NAT passerelle, l'associer à un sous-réseau public, puis l'ajouter à la table de routage de votre sous-réseau privé.

Pour créer une NAT passerelle et l'associer à un sous-réseau public

1. Dans le volet de navigation, choisissez les NATpasserelles.
2. Choisissez Create NAT gateway.
3. (Facultatif) Entrez le nom de votre NAT passerelle.
4. Pour Sous-réseau, sélectionnez un sous-réseau public dans votre. VPC (Un sous-réseau public est un sous-réseau qui possède une route directe vers une passerelle Internet dans sa table de routage.)

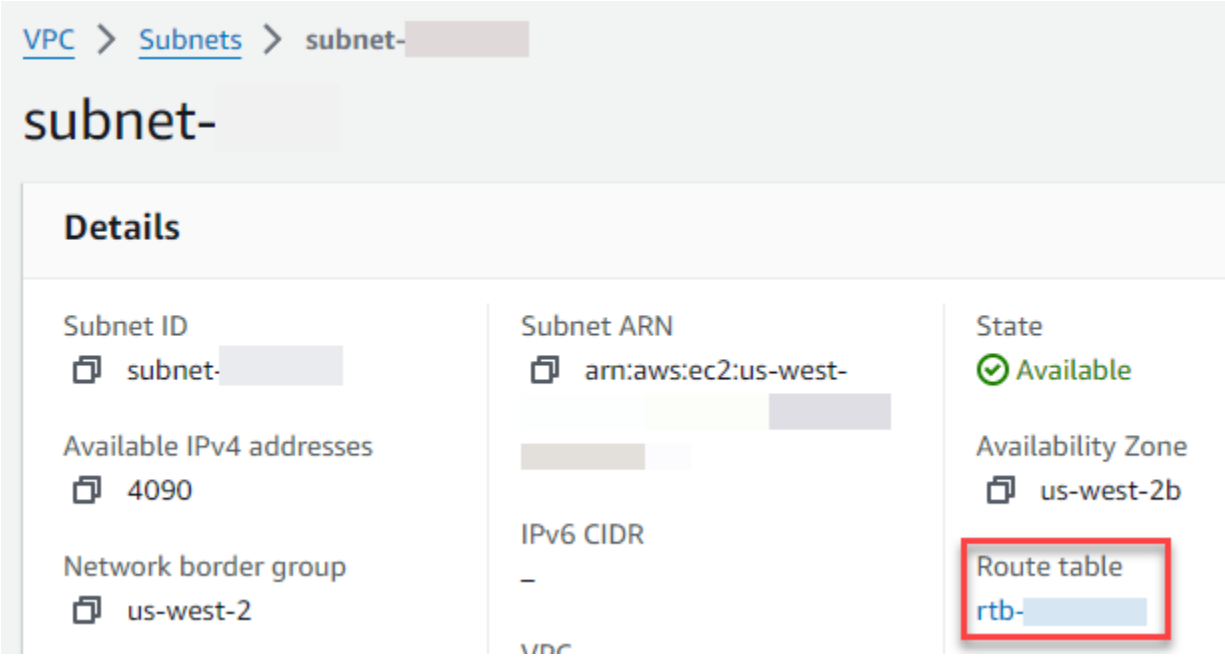
> **Note**
>
> NATles passerelles sont associées à un sous-réseau public, mais l'entrée de la table de routage se trouve dans le sous-réseau privé.

5. Pour l'ID d'allocation IP élastique, sélectionnez une adresse IP élastique ou choisissez Allouer une adresse IP élastique.

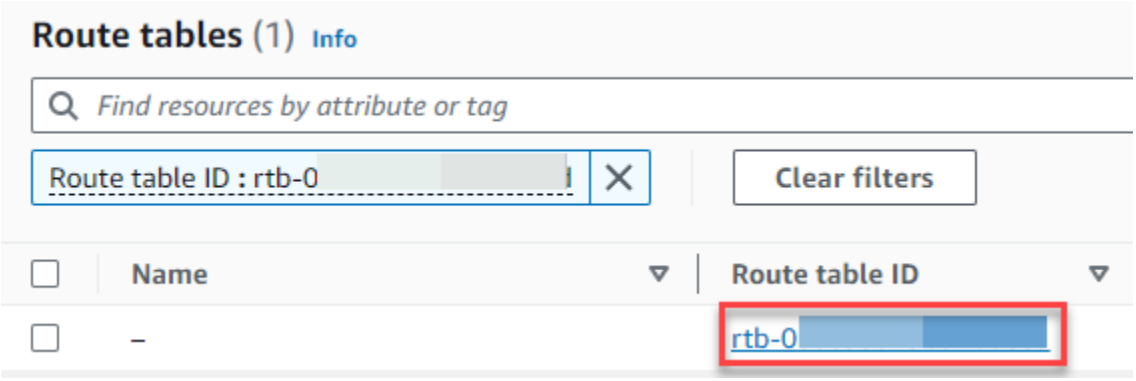
6. Choisissez Create NAT gateway.

Pour ajouter un itinéraire vers la NAT passerelle dans la table de routage du sous-réseau privé

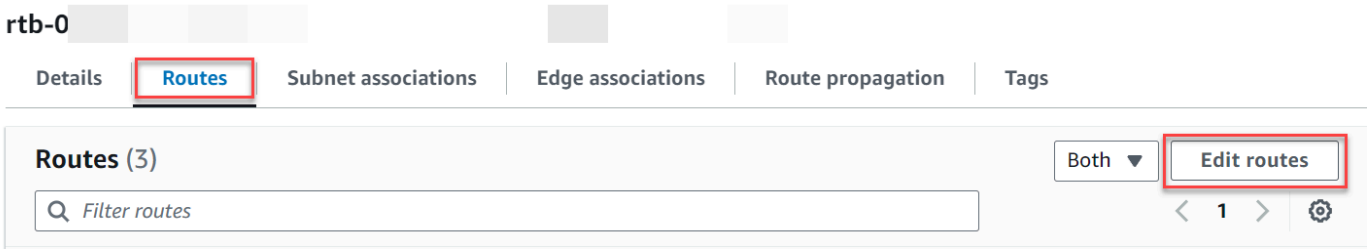
1. Dans le panneau de navigation, choisissez Subnets (Sous-réseaux).
2. Sélectionnez un sous-réseau privé dans votre. VPC (Un sous-réseau privé est un sous-réseau dont la table de routage ne contient pas de route vers une passerelle Internet.)
3. Cliquez sur le lien sous Table de routage.



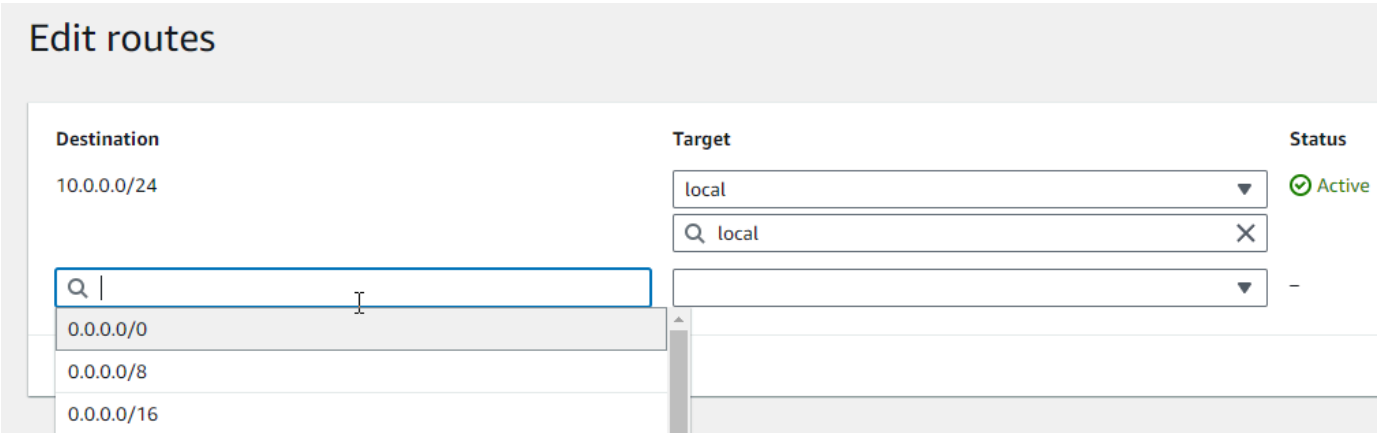

4. Choisissez l'ID de la table de routage pour ouvrir la page des détails de la table de routage.



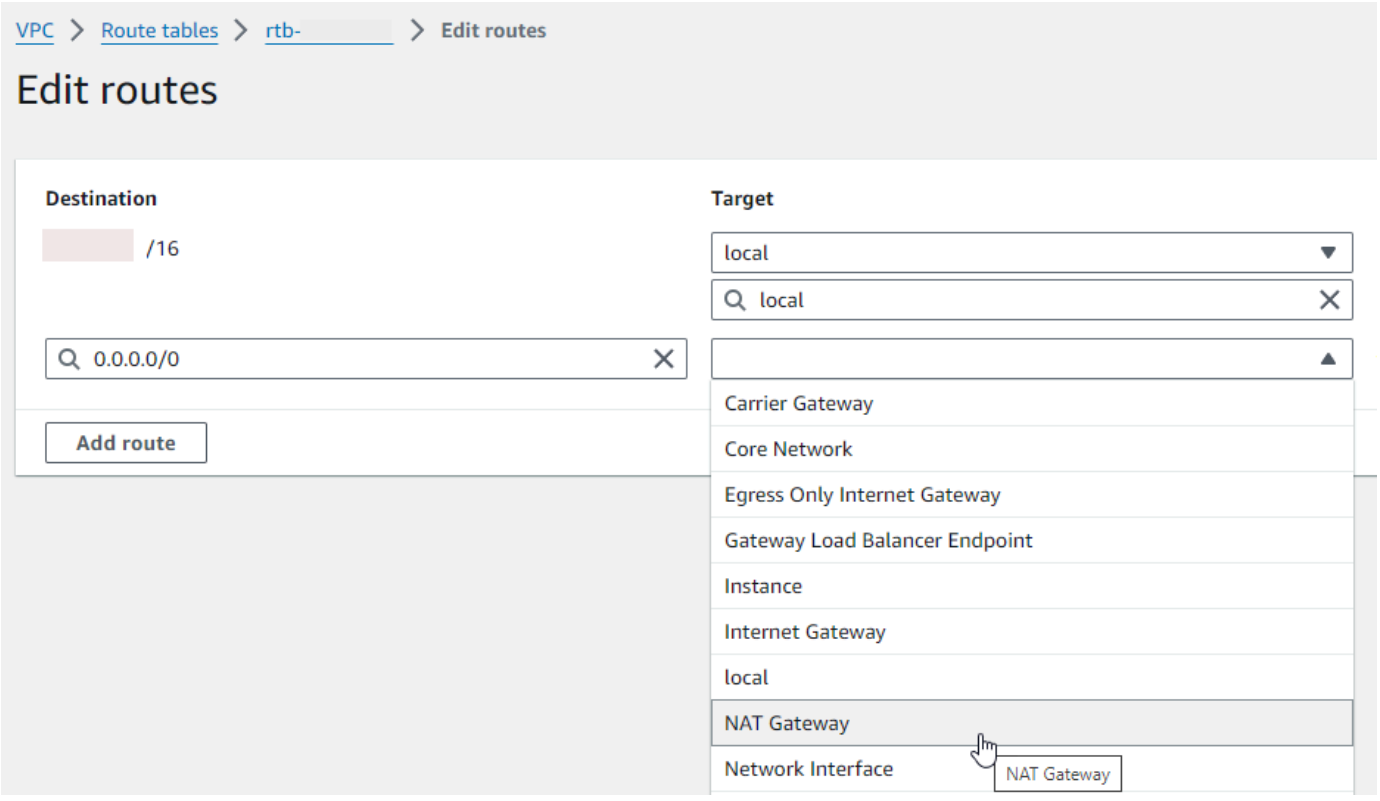

5. Faites défiler l'écran vers le bas et choisissez l'onglet Routes, puis choisissez Modifier les itinéraires




6. Choisissez Ajouter un itinéraire, puis entrez `0.0.0.0/0` dans le champ Destination.




7. Pour Target, sélectionnez NATGateway, puis choisissez la NAT passerelle que vous avez créée précédemment.



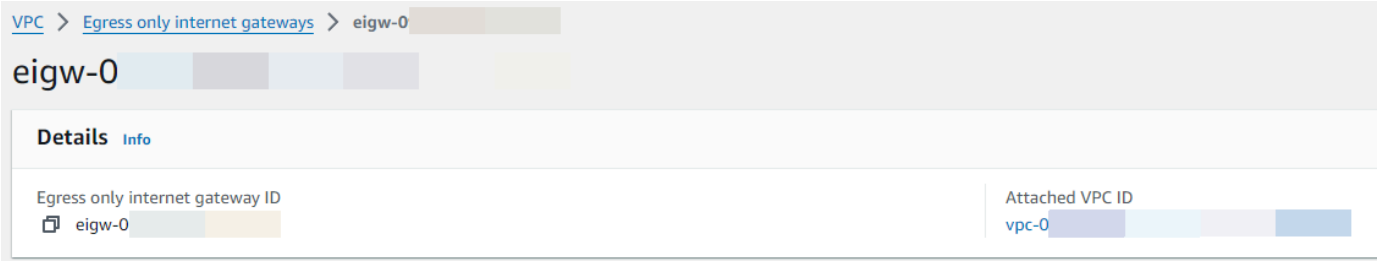

8. Sélectionnez Enregistrer les modifications.

### Création d'une passerelle Internet de sortie uniquement (uniquement) IPv6

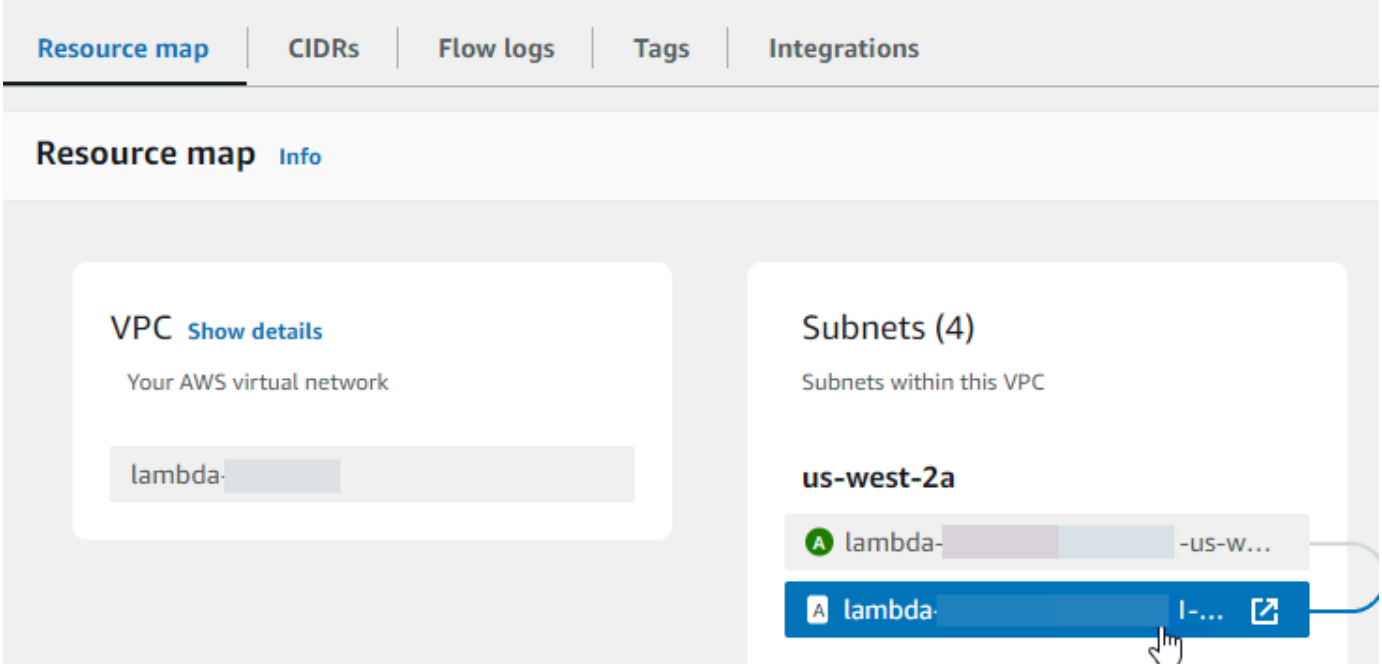
Procédez comme suit pour créer une passerelle Internet de sortie uniquement et l'ajouter à la table de routage de votre sous-réseau privé.

Pour créer une passerelle Internet de sortie uniquement

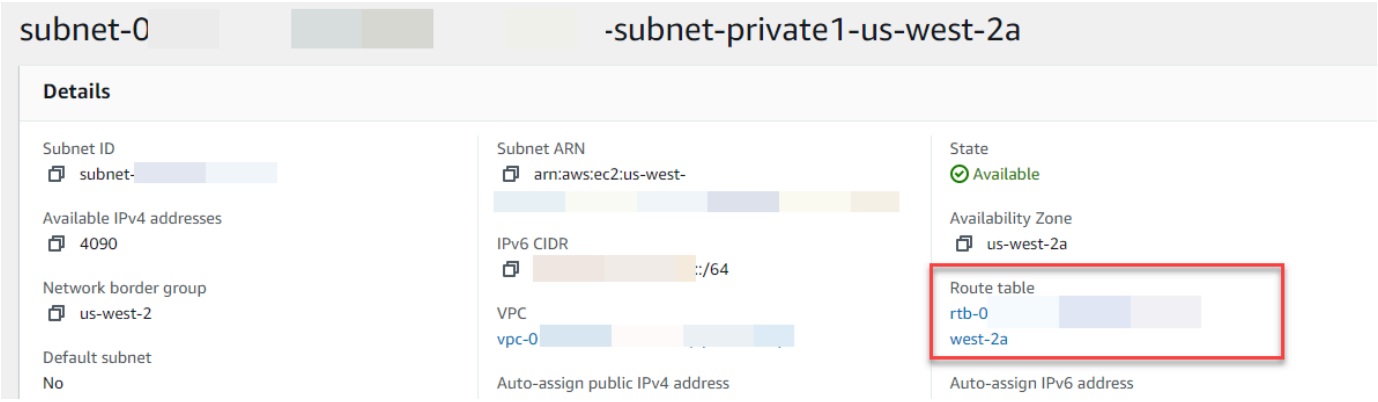
1. Dans le volet de navigation, choisissez Passerelles Internet de sortie uniquement.
2. Choisissez Créer une passerelle Internet de sortie uniquement.
3. (Facultatif) Entrez un nom.
4. Sélectionnez celui VPC dans lequel vous souhaitez créer la passerelle Internet de sortie uniquement.
5. Choisissez Créer une passerelle Internet de sortie uniquement.
6. Cliquez sur le lien sous VPCIdentifiant joint.



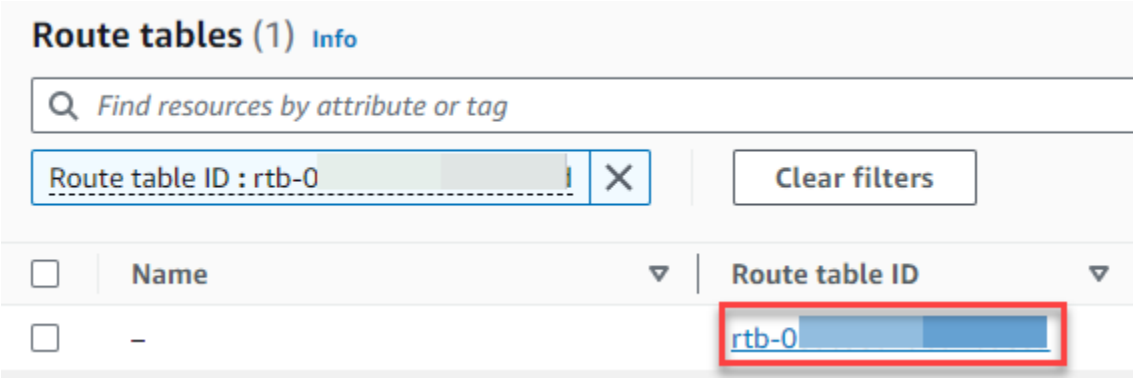

7. Cliquez sur le lien sous VPCID pour ouvrir la page de VPC détails.
8. Faites défiler la page jusqu'à la section Carte des ressources, puis choisissez un sous-réseau privé. (Un sous-réseau privé est un sous-réseau dont la table de routage ne contient pas de route vers une passerelle Internet.) Les détails du sous-réseau sont affichés dans un nouvel onglet.



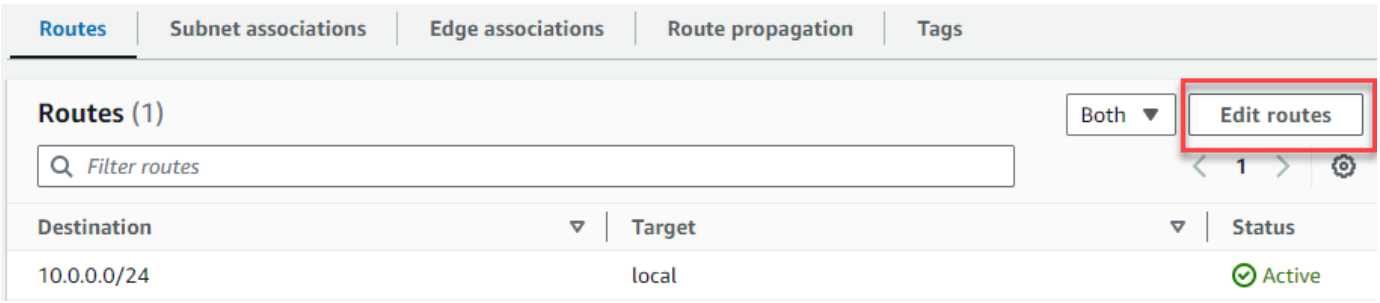

9. Cliquez sur le lien sous Table de routage.



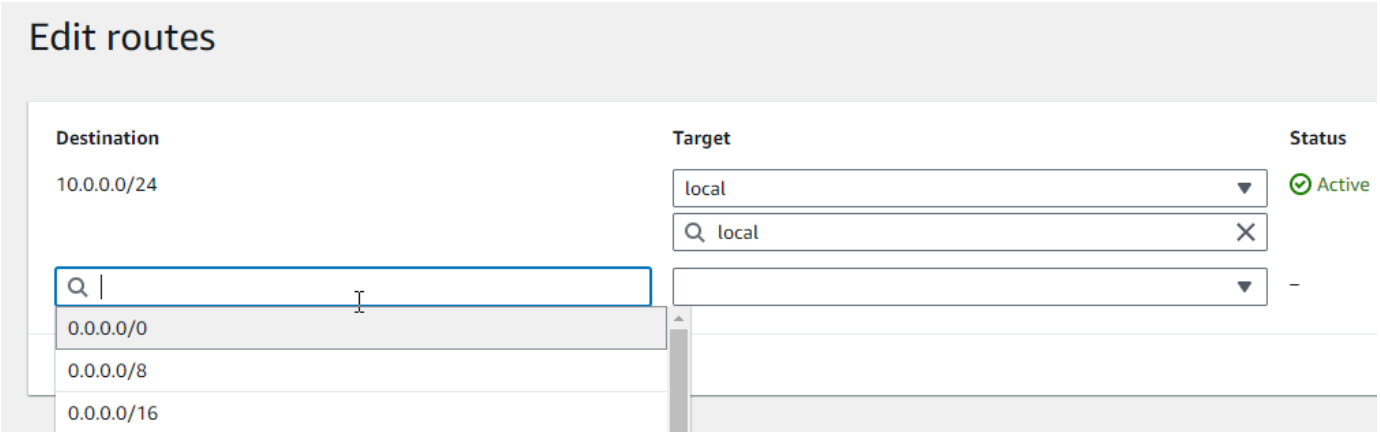

10. Choisissez l'ID de la table de routage pour ouvrir la page des détails de la table de routage.



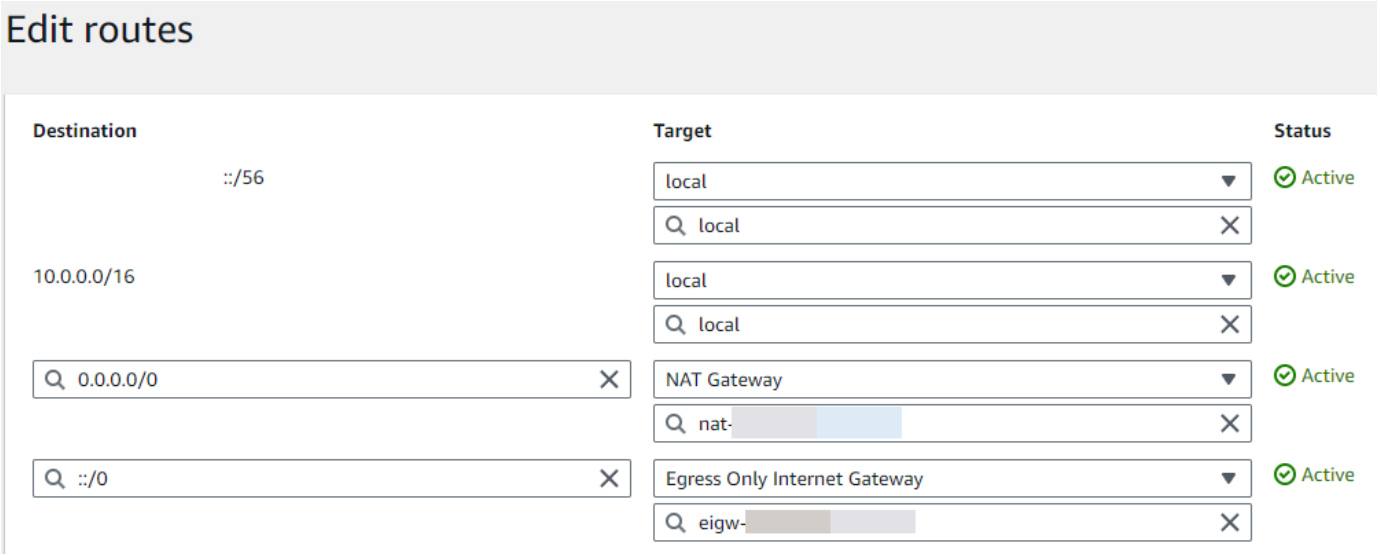

11. Sous Itinéraires, choisissez Modifier les itinéraires.




12. Choisissez Ajouter un itinéraire, puis entrez `::/0` dans le champ Destination.




13. Pour Target, sélectionnez Egress Only Internet Gateway, puis choisissez la passerelle que vous avez créée précédemment.




14. Sélectionnez Enregistrer les modifications.

Configurer la fonction Lambda

Pour configurer une fonction VPC lors de la création d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Sous Informations de base, dans Nom de fonction, entrez un nom pour votre fonction.
4. Développez Advanced settings (Paramètres avancés).
5. Sélectionnez Activer VPC, puis choisissez unVPC.
6. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.
7. Pour les sous-réseaux, sélectionnez tous les sous-réseaux privés. Les sous-réseaux privés peuvent accéder à Internet via la NAT passerelle. La connexion d'une fonction à un sous-réseau public ne lui donne pas accès à Internet.

   > **Note**
   >
   > Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR

8. Pour les groupes de sécurité, sélectionnez un groupe de sécurité qui autorise le trafic sortant.
9. Sélectionnez Create function (Créer une fonction).

Lambda crée automatiquement un rôle d'exécution avec la politique AWSLambdaVPCAccessExecutionRole AWS gérée. Les autorisations définies dans cette politique ne sont requises que pour créer des interfaces réseau élastiques pour la VPC configuration, et non pour invoquer votre fonction. Pour appliquer les autorisations de moindre privilège, vous pouvez supprimer la AWSLambdaVPCAccessExecutionRolepolitique de votre rôle d'exécution après avoir créé la fonction et VPC la configuration. Pour de plus amples informations, veuillez consulter IAMAutorisations requises.

Pour configurer un VPC pour une fonction existante

Pour ajouter une VPC configuration à une fonction existante, le rôle d'exécution de la fonction doit être autorisé à créer et à gérer des interfaces réseau élastiques. La politique AWSLambdaVPCAccessExecutionRole AWS gérée inclut les autorisations

requises. Pour appliquer les autorisations de moindre privilège, vous pouvez supprimer la AWSLambdaVPCAccessExecutionRolepolitique de votre rôle d'exécution après avoir créé la VPC configuration.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez l'onglet Configuration, puis choisissez VPC.
4. Sous VPC, choisissez Modifier.
5. Sélectionnez leVPC.
6. (Facultatif) Pour autoriser le IPv6trafic sortant, sélectionnez Autoriser le IPv6 trafic pour les sous-réseaux à double pile.
7. Pour les sous-réseaux, sélectionnez tous les sous-réseaux privés. Les sous-réseaux privés peuvent accéder à Internet via la NAT passerelle. La connexion d'une fonction à un sous-réseau public ne lui donne pas accès à Internet.

   > **Note**
   >
   > Si vous avez sélectionné Autoriser IPv6 le trafic pour les sous-réseaux à double pile, tous les sous-réseaux sélectionnés doivent comporter un IPv4 CIDR bloc et un bloc. IPv6 CIDR

8. Pour les groupes de sécurité, sélectionnez un groupe de sécurité qui autorise le trafic sortant.
9. Choisissez Save (Enregistrer).

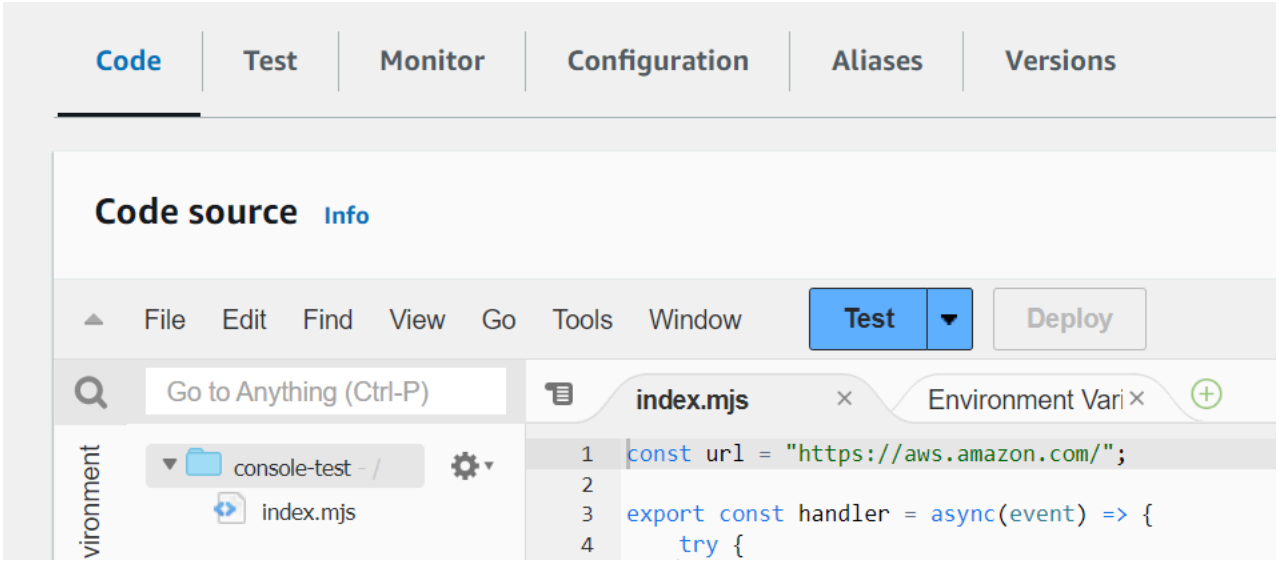
### Tester la fonction

Utilisez l'exemple de code suivant pour vérifier que votre fonction VPC connectée peut accéder à l'Internet public. En cas de succès, le code renvoie un code d'`200`état. En cas d'échec, la fonction expire.

Node.js

Cet exemple utilise`fetch`, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

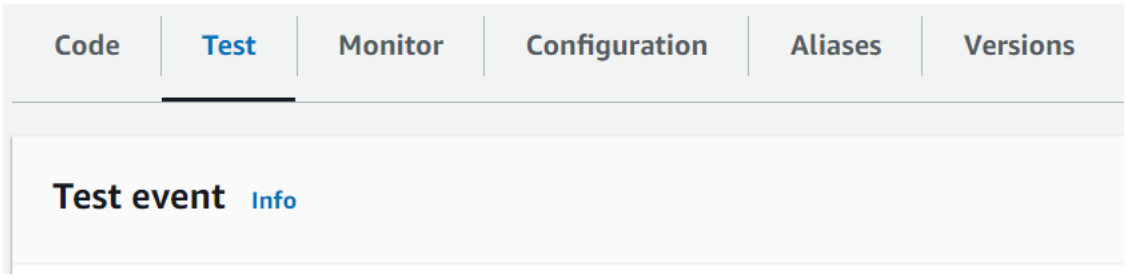
1. Dans le volet Source du code de la console Lambda, collez le code suivant dans le fichier index.mjs. La fonction envoie une HTTP GET demande à un point de terminaison public et renvoie le code de HTTP réponse pour tester si la fonction a accès à l'Internet public.



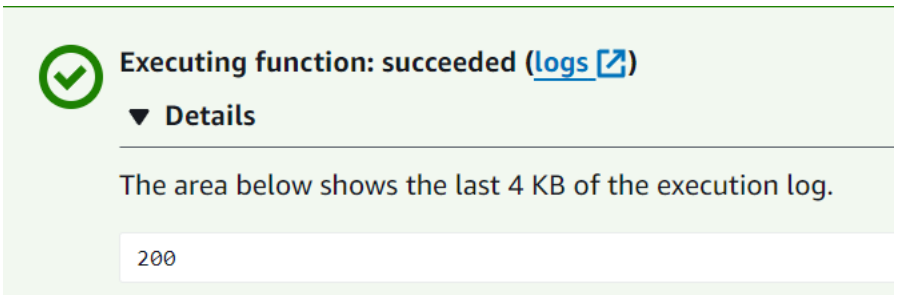

Example — HTTP demande avec async/await

```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        // fetch is available with Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

2. Choisissez Deploy (Déployer).
3. Choisissez l’onglet Test.




4. Sélectionnez Tester).
5. La fonction renvoie un code d'`200`état. Cela signifie que la fonction dispose d'un accès Internet sortant.

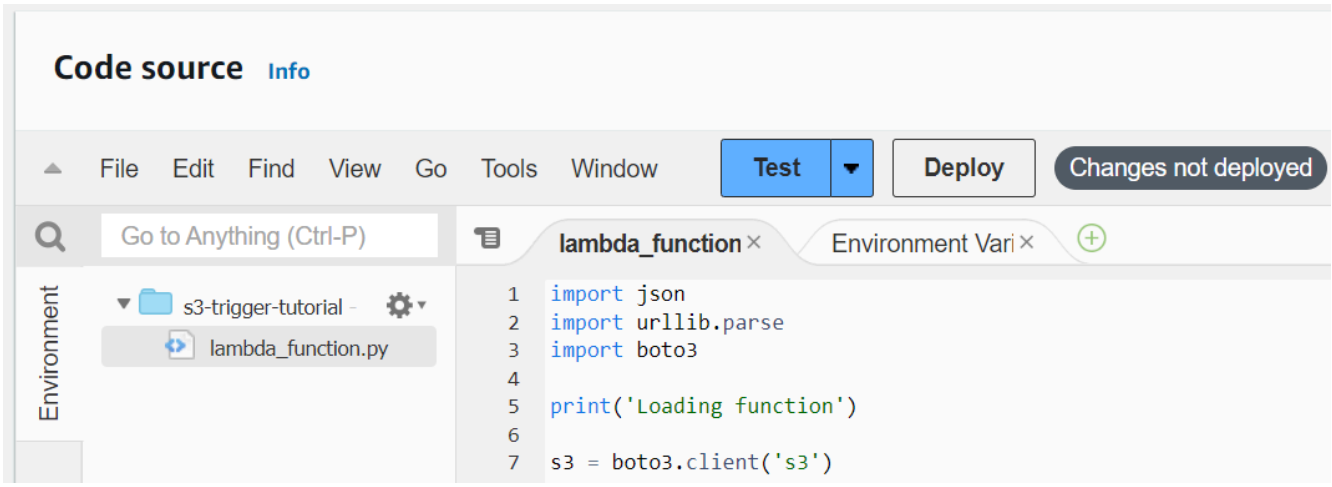



Si la fonction ne parvient pas à accéder à l'Internet public, le message d'erreur suivant s'affiche :

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

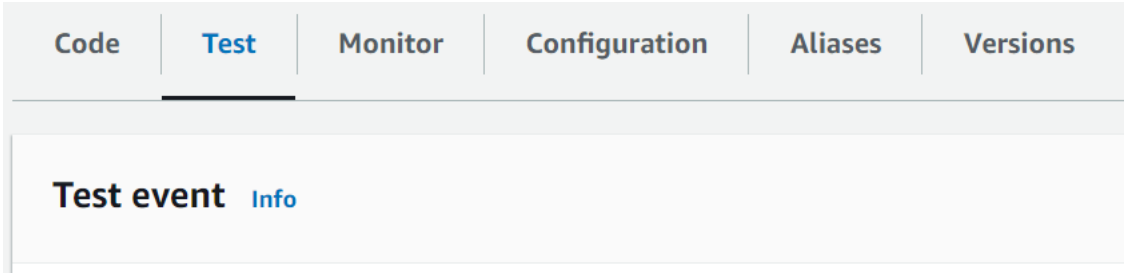
Python

1. Dans le volet Code source de la console Lambda, collez le code suivant dans le fichier lambda_function.py. La fonction envoie une HTTP GET demande à un point de terminaison public et renvoie le code de HTTP réponse pour tester si la fonction a accès à l'Internet public.

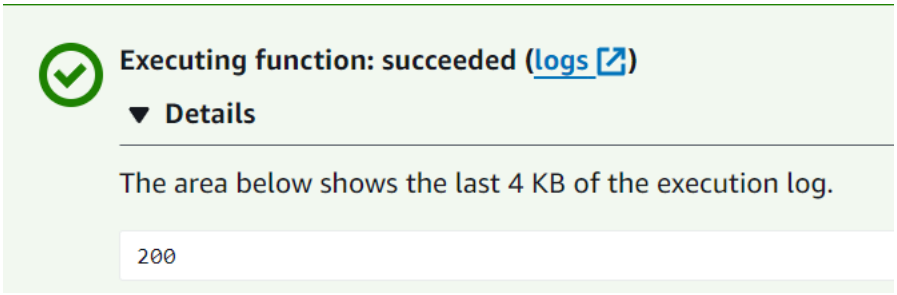



```
import urllib.request

def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

2. Choisissez Deploy (Déployer).
3. Choisissez l'onglet Test.




4. Sélectionnez Tester).
5. La fonction renvoie un code d'`200`état. Cela signifie que la fonction dispose d'un accès Internet sortant.




Si la fonction ne parvient pas à accéder à l'Internet public, le message d'erreur suivant s'affiche :

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

# Connexion des VPC points de terminaison de l'interface entrante pour Lambda

Si vous utilisez Amazon Virtual Private Cloud (AmazonVPC) pour héberger vos AWS ressources, vous pouvez établir une connexion entre vous VPC et Lambda. Vous pouvez utiliser cette connexion pour invoquer votre fonction Lambda sans traverser l'Internet public.

Pour établir une connexion privée entre vous VPC et Lambda, créez un point de terminaison d'interface VPC. Les points de terminaison de l'interface sont alimentés par AWS PrivateLinkce qui vous permet d'accéder à APIs Lambda en privé sans passerelle InternetNAT, appareilVPN, AWS Direct Connect connexion ou connexion. Les instances de votre VPC ordinateur n'ont pas besoin d'adresses IP publiques pour communiquer avec LambdaAPIs. Le trafic entre vous VPC et Lambda ne quitte pas le AWS réseau.

Chaque point de terminaison d'interface est représenté par une ou plusieurs interfaces réseau Elastic dans vos sous-réseaux. Une interface réseau fournit une adresse IP privée qui sert de point d'entrée au trafic vers Lambda.

Sections


- Considérations relatives aux points de terminaison d'interface Lambda
- Création d'un point de terminaison d'interface pour Lambda
- Création d'une stratégie de point de terminaison d'interface pour Lambda


## Considérations relatives aux points de terminaison d'interface Lambda

Avant de configurer un point de terminaison d'interface pour Lambda, assurez-vous de consulter les propriétés et les limites du point de terminaison d'interface dans le guide de VPCl'utilisateur Amazon.

Vous pouvez appeler n'importe laquelle des API opérations Lambda depuis votre. VPC Par exemple, vous pouvez appeler la fonction Lambda en appelant la fonction from within `Invoke` API your. VPC Pour la liste complète de LambdaAPIs, voir Actions dans la référence Lambda. API

`use1-az3`est une région à capacité limitée pour les fonctions LambdaVPC. Vous ne devez pas utiliser de sous-réseaux dans cette zone de disponibilité avec vos fonctions Lambda, car cela peut réduire la redondance zonale en cas de panne.

### Keep-alive pour les connexions persistantes

Lambda purgeant les connexions inactives au fil du temps, vous devez utiliser une directive keep-alive pour maintenir les connexions persistantes. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, voir Réutilisation des connexions avec Keep-Alive dans Node.js dans le Guide du développeur AWS SDK for JavaScript .

### Considérations sur la facturation

L'accès à une fonction Lambda via un point de terminaison d'interface n'occasionne aucun coût supplémentaire. Pour plus d'informations sur la tarification de Lambda, consultez Tarification AWS Lambda.

La tarification standard AWS PrivateLink s'applique aux points de terminaison d'interface pour Lambda. Votre AWS compte est facturé pour chaque heure pendant laquelle un point de terminaison d'interface est configuré dans chaque zone de disponibilité et pour les données traitées via le point de terminaison d'interface. Pour plus d'informations sur la tarification des points de terminaison d'interface, consultez Tarification AWS PrivateLink.

### VPCConsidérations relatives au peering

Vous pouvez connecter d'autres points de terminaison VPCs à l'interface VPC with en utilisant le VPCpeering. VPCle peering est une connexion réseau entre deuxVPCs. Vous pouvez établir une connexion de VPC peering entre vos deux VPCs comptes ou avec une connexion sur un VPC autre AWS compte. Ils VPCs peuvent également se trouver dans deux AWS régions différentes.

Le trafic entre pairs VPCs reste sur le AWS réseau et ne traverse pas l'Internet public. Une fois peered, VPCs les ressources telles que les instances Amazon Elastic Compute Cloud (AmazonEC2), les instances Amazon Relational Database Service (RDSAmazon) ou les fonctions Lambda VPC activées dans les deux VPCs peuvent accéder au API Lambda via des points de terminaison d'interface créés dans l'un des. VPCs

## Création d'un point de terminaison d'interface pour Lambda

Vous pouvez créer un point de terminaison d'interface pour Lambda à l'aide de VPC la console Amazon ou du AWS Command Line Interface ()AWS CLI. Pour plus d'informations, consultez la section Création d'un point de terminaison d'interface dans le guide de VPC l'utilisateur Amazon.

Pour créer un point de terminaison d'interface pour Lambda (console)

1. Ouvrez la page Endpoints de la VPC console Amazon.
2. Choisissez Créer un point de terminaison.
3. Pour Catégorie de service, assurez-vous que Services AWS est sélectionné.
4. Pour Nom du service, choisissez com.amazonaws.***region***.lambda. Vérifiez que le Type est Interface.
5. Choisissez un VPC et des sous-réseaux.
6. Pour activer le mode privé DNS pour le point de terminaison de l'interface, cochez la case Activer le DNS nom. Nous vous recommandons d'activer les DNS noms privés pour vos VPC points de terminaison pour Services AWS. Cela garantit que les demandes qui utilisent les points de terminaison du service public, telles que les demandes effectuées via un AWS SDK, sont résolues vers votre VPC point de terminaison.
7. Pour Groupes de sécurité, choisissez un ou plusieurs groupes de sécurité.
8. Choisissez Créer un point de terminaison.

Pour utiliser l'DNSoption privée, vous devez définir la `enableDnsHostnames` fin `enableDnsSupportattributes` de votreVPC. Pour plus d'informations, consultez la section Affichage et mise à jour du DNS support correspondant VPC dans le guide de VPC l'utilisateur Amazon. Si vous activez le mode privé DNS pour le point de terminaison de l'interface, vous pouvez envoyer des API demandes à Lambda en utilisant son DNS nom par défaut pour la région, par exemple,. `lambda.us-east-1.amazonaws.com` Pour plus de points de terminaison de service, consultez Points de terminaison de service et quotas dans le Références générales AWS.

Pour plus d'informations, consultez la section Accès à un service via un point de terminaison d'interface dans le guide de VPC l'utilisateur Amazon.

Pour plus d'informations sur la création et la configuration d'un point de terminaison à l'aide de AWS CloudFormation, consultez la VPCEndpoint ressource AWSEC2: : : : dans le guide de AWS CloudFormation l'utilisateur.

Pour créer un point de terminaison d'interface pour Lambda (AWS CLI)

Utilisez la create-vpc-endpointcommande et spécifiez l'VPCID, le type de point de VPC terminaison (interface), le nom du service, les sous-réseaux qui utiliseront le point de terminaison et les groupes de sécurité à associer aux interfaces réseau du point de terminaison. Par exemple :

```
aws ec2 create-vpc-endpoint
  --vpc-id vpc-ec43eb89
  --vpc-endpoint-type Interface
  --service-name com.amazonaws.us-east-1.lambda
  --subnet-id subnet-abababab
  --security-group-id sg-1a2b3c4d
```

## Création d'une stratégie de point de terminaison d'interface pour Lambda

Pour contrôler qui peut utiliser votre point de terminaison d'interface, ainsi que les fonctions Lambda auxquelles l'utilisateur peut accéder, vous pouvez attacher une stratégie de point de terminaison à votre point de terminaison. La stratégie spécifie les informations suivantes :

- Le principal qui peut exécuter des actions.
- Les actions que le principal peut effectuer.
- Ressources sur lesquelles le principal peut effectuer des actions.

Pour plus d'informations, consultez la section Contrôle de l'accès aux services avec des VPC points de terminaison dans le guide de VPC l'utilisateur Amazon.

Exemple : stratégie de point de terminaison d'interface pour des actions Lambda

Voici un exemple de stratégie de point de terminaison pour Lambda. Lorsqu'elle est attachée à un point de terminaison, cette stratégie permet à l'utilisateur `MyUser` d'invoquer la fonction `my-function`.

> **Note**
>
> Vous devez inclure à la fois la fonction qualifiée et la fonction non qualifiée ARN dans la ressource.

```
{
   "Statement":[
      {
         "Principal":
         {
             "AWS": "arn:aws:iam::111122223333:user/MyUser"
         },
```

```
         "Effect":"Allow",
         "Action":[
            "lambda:InvokeFunction"
         ],
         "Resource": [
               "arn:aws:lambda:us-east-2:123456789012:function:my-function",
               "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
            ]
      }
   ]
}
```

# Configuration de l'accès au système de fichiers pour les fonctions Lambda

Vous pouvez configurer une fonction pour monter un système de fichiers Amazon Elastic File System (AmazonEFS) dans un répertoire local. Avec AmazonEFS, votre code de fonction peut accéder aux ressources partagées et les modifier en toute sécurité et à un rythme élevé.

Sections


- Rôle d'exécution et autorisations utilisateur
- Configuration d'un système de fichiers et d'un point d'accès
- Connexion à un système de fichiers (console)


## Rôle d'exécution et autorisations utilisateur

Si le système de fichiers n'a pas de stratégie configurée par l'utilisateur AWS Identity and Access Management (IAM), EFS utilise une stratégie par défaut qui accorde un accès complet à tout client qui peut se connecter au système de fichiers à l'aide d'une cible de montage du système de fichiers. Si le système de fichiers dispose d'une IAM politique configurée par l'utilisateur, le rôle d'exécution de votre fonction doit disposer des `elasticfilesystem` autorisations appropriées.

Autorisations du rôle d’exécution

- système de fichiers élastique : ClientMount
- elasticfilesystem : ClientWrite (non requis pour les connexions en lecture seule)

Ces autorisations sont incluses dans la politique AmazonElasticFileSystemClientReadWriteAccessgérée. En outre, votre rôle d'exécution doit disposer des autorisations requises pour se connecter au système de fichiers VPC.

Lorsque vous configurez un système de fichiers, Lambda utilise vos autorisations pour vérifier les cibles de montage. Pour configurer une fonction pour vous connecter à un système de fichiers, votre utilisateur a besoin des autorisations suivantes :

Autorisations des utilisateurs

- système de fichiers élastique : DescribeMountTargets

# Configuration d'un système de fichiers et d'un point d'accès

Créez un système de fichiers dans Amazon EFS avec une cible de montage dans chaque zone de disponibilité à laquelle votre fonction se connecte. Pour les performances et une résilience, utilisez au moins deux zones de disponibilité. Par exemple, dans une configuration simple, vous pourriez avoir deux sous-réseaux privés dans des zones de disponibilité distinctes. VPC La fonction se connecte aux deux sous-réseaux et une cible de montage est disponible dans chacun. Assurez-vous que NFS le trafic (port 2049) est autorisé par les groupes de sécurité utilisés par la fonction et les cibles de montage.

> **Note**
>
> Lorsque vous créez un système de fichiers, vous choisissez un mode de performances qui ne peut pas être modifié ultérieurement. Le mode à usage général a une latence plus faible, et le mode Max I/O prend en charge un débit maximal plus élevé et. IOPS Pour vous aider à choisir, consultez Amazon EFS Performance dans le guide de l'utilisateur Amazon Elastic File System.

Un point d'accès relie chaque instance de la fonction à la cible de montage droite pour la zone de disponibilité à laquelle elle se connecte. Pour des performances optimales, créez un point d'accès avec un chemin non racine et limitez le nombre de fichiers que vous créez dans chaque répertoire. L'exemple suivant crée un répertoire nommé `my-function` sur le système de fichiers et définit l'ID propriétaire sur 1001 avec les autorisations d'annuaire standard (755).

Example configuration du point d'accès

- Nom – `files`
- ID de l'utilisateur – `1001`
- ID du groupe – `1001`
- Chemin – `/my-function`
- Permissions (Autorisations – `755`
- ID d'utilisateur du propriétaire – `1001`
- ID d'utilisateur du groupe – `1001`

Lorsqu'une fonction utilise le point d'accès, elle reçoit l'ID utilisateur 1001 et dispose d'un accès complet au répertoire.

Pour plus d'informations, consultez les rubriques suivantes dans le Guide de l'utilisateur Amazon Elastic File System User.

- Création de ressources pour Amazon EFS
- Collaboration avec des utilisateurs, des groupes et des autorisations

## Connexion à un système de fichiers (console)

Une fonction se connecte à un système de fichiers via le réseau local dans un fichierVPC. Les sous-réseaux auxquels votre fonction se connecte peuvent être les mêmes sous-réseaux qui contiennent des points de montage pour votre système de fichiers, ou des sous-réseaux de la même zone de disponibilité qui peuvent acheminer le NFS trafic (port 2049) vers le système de fichiers.

> **Note**
>
> Si votre fonction n'est pas déjà connectée à unVPC, consultezPermettre aux fonctions Lambda d'accéder aux ressources d'un Amazon VPC.

Pour configurer l'accès au système de fichiers

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Configuration, puis File systems (Systèmes de fichiers).
4. Sous Système de fichiers, choisissez Ajouter un système de fichiers.
5. Configurez les propriétés suivantes :
   - EFSsystème de fichiers — Point d'accès à un système de fichiers dans le même systèmeVPC.
   - Chemin de montage local – Emplacement où le système de fichiers est monté sur la fonction Lambda, commençant par `/mnt/`.

**Tarification**

Amazon EFS facture le stockage et le débit, avec des tarifs qui varient en fonction de la classe de stockage. Pour plus de détails, consultez EFSles tarifs Amazon.
Lambda facture le transfert de données entre. VPCs Cela ne s'applique que si votre fonction VPC est appairée à une autre VPC avec un système de fichiers. Les tarifs sont les mêmes que pour le transfert de EC2 données Amazon entre VPCs une même région. Pour plus d'informations, consultez Tarification ambda.

# Création d'un alias pour une fonction Lambda

Vous pouvez créer des alias pour votre fonction Lambda. Un alias Lambda est un pointeur vers une version de la fonction que vous pouvez mettre à jour. Les utilisateurs de la fonction peuvent accéder à la version de la fonction en utilisant l'alias Amazon Resource Name (ARN). Lorsque vous déployez une nouvelle version, vous pouvez mettre à jour l'alias pour utiliser la nouvelle version, ou diviser le trafic entre deux versions.

Console

Pour créer un alias à l'aide de la console

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Alias, puis Create alias (Créer un alias).
4. Sur la page Create alias (Créer un alias), procédez de la manière suivante :
   a. Renseignez le champ Name (Nom) pour l'alias.
   b. (Facultatif) Renseignez le champ Description de l'alias.
   c. Pour Version, choisissez une version de fonction vers laquelle vous souhaitez que l'alias pointe.
   d. (Facultatif) Pour configurer le routage sur l'alias, développez Weighted alias (Alias pondéré). Pour de plus amples informations, veuillez consulter Création d'une configuration de routage pour un alias Lambda.
   e. Choisissez Save (Enregistrer).

AWS CLI

Pour créer un alias à l'aide de AWS Command Line Interface (AWS CLI), utilisez la commande create-alias.

```
aws lambda create-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number \
  --description " "
```

Pour modifier un alias afin de pointer vers une nouvelle version de la fonction, utilisez la commande update-alias.

```
aws lambda update-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number
```

Pour supprimer un alias, utilisez la commande delete-alias.

```
aws lambda delete-alias \
  --function-name my-function \
  --name alias-name
```

Les AWS CLI commandes des étapes précédentes correspondent aux opérations Lambda API suivantes :

- CreateAlias
- UpdateAlias
- DeleteAlias

# Utilisation d'alias Lambda dans les sources d'événements et les politiques d'autorisation

Chaque alias possède un identifiant uniqueARN. Un alias peut uniquement pointer vers une version de fonction, et non vers un autre alias. Vous pouvez mettre à jour un alias de sorte qu'il pointe vers une nouvelle version de la fonction.

Des sources d'événements telles qu'Amazon Simple Storage Service (Amazon S3) appellent votre fonction Lambda. Ces sources d'événements gèrent un mappage qui identifie la fonction à appeler lorsque des événements se produisent. Si vous spécifiez un alias de fonction Lambda dans la configuration du mappage, vous n'avez pas besoin de mettre à jour le mappage lorsque la version de la fonction change. Pour de plus amples informations, veuillez consulter Comment Lambda traite les enregistrements provenant de sources d'événements basées sur des flux et des files d'attente.

Dans une stratégie de ressources, vous pouvez accorder des autorisations en lien avec des sources d'événements à utiliser votre fonction Lambda. Si vous spécifiez un alias ARN dans la politique, il n'est pas nécessaire de mettre à jour la politique lorsque la version de la fonction change.

## Stratégies basées sur une ressource

Vous pouvez utiliser une stratégie basée sur les ressources pour accorder l'accès à votre fonction à un service, à une ressource ou à un compte. La portée de cette autorisation est dépendante du fait que vous appliquiez l'autorisation à un alias, à une version ou à la fonction entière. Par exemple, si vous utilisez un nom d'alias (tel que`helloworld:PROD`), l'autorisation vous permet d'appeler la `helloworld` fonction à l'aide de l'alias ARN (`helloworld:PROD`).

Si vous essayez d'appeler la fonction sans alias ou version spécifique, vous obtenez une erreur d'autorisation. Cette erreur d'autorisation continue de se produire même si vous tentez d'appeler directement la version de fonction associée à l'alias.

Par exemple, la AWS CLI commande suivante autorise Amazon S3 à invoquer l'PRODalias de la `helloworld` fonction lorsqu'Amazon S3 agit pour le compte de`amzn-s3-demo-bucket`.

```
aws lambda add-permission \
  --function-name helloworld \
  --qualifier PROD \
  --statement-id 1 \
  --principal s3.amazonaws.com \
  --action lambda:InvokeFunction \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
```

```
  --source-account 123456789012
```

Pour de plus amples informations sur l'utilisation des noms de ressources dans les stratégies, veuillez consulter Affiner les sections Ressources et Conditions des politiques.

## Création d'une configuration de routage pour un alias Lambda

Utilisez une configuration de routage sur un alias pour envoyer une partie du trafic vers une deuxième version de fonction. Par exemple, vous pouvez réduire le risque de déploiement d'une nouvelle version en configurant l'alias de sorte qu'il envoie la majeure partie du trafic vers la version existante, et seulement un faible pourcentage du trafic vers la nouvelle version.

Lambda utilise un modèle probabiliste simple pour distribuer le trafic entre les deux versions de la fonction. Quand le niveau de trafic est faible, il se peut que vous observiez une variance élevée entre les pourcentages de trafic configuré et réel sur chaque version. Si votre fonction utilise une simultanéité approvisionnée, vous pouvez éviter des appels de débordement en configurant un plus grand nombre d'instances de simultanéité approvisionnées pendant que le routage d'alias est actif.

Vous pouvez faire pointer un alias vers au maximum deux versions de fonction Lambda. Les versions doivent répondre aux critères suivants :

- Les deux versions doivent avoir le même rôle d'exécution.
- Les deux versions doivent avoir la même configuration de file d'attente de lettres mortes ou aucune configuration de file d'attente de lettres mortes.
- Les deux versions doivent être publiées. L'alias ne peut pas pointer vers `$LATEST`.

Console

Pour configurer le routage sur un alias à l'aide de la console

> **Note**
>
> Vérifiez que la fonction a au moins deux versions publiées. Pour créer des versions supplémentaires, suivez les instructions de la section Gérer les versions des fonctions Lambda.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez une fonction.
3. Sélectionnez Alias, puis Create alias (Créer un alias).
4. Sur la page Create alias (Créer un alias), procédez de la manière suivante :
   a. Renseignez le champ Name (Nom) pour l'alias.
   b. (Facultatif) Renseignez le champ Description de l'alias.
   c. Pour Version, choisissez la première version de fonction vers laquelle vous souhaitez que l'alias pointe.
   d. Développez Weighted alias (Alias pondéré).
   e. Pour Additional version (Version supplémentaire), choisissez la deuxième version de fonction vers laquelle vous souhaitez que l'alias pointe.
   f. Pour Weight (%) (Pondération (%)), entrez une valeur de pondération pour la fonction. La pondération est le pourcentage de trafic qui est affecté à cette version lorsque l'alias est appelé. La première version reçoit le reste du trafic. Par exemple, si vous spécifiez 10 % pour Additional version (Version supplémentaire), 90 % du trafic est automatiquement affecté à la première version.
   g. Choisissez Save (Enregistrer).

AWS CLI

Utilisez les AWS CLI commandes create-alias et update-alias pour configurer les pondérations du trafic entre deux versions de fonction. Lorsque vous créez ou mettez à jour l'alias, vous spécifiez la pondération du trafic dans le paramètre `routing-config`.

L'exemple suivant crée un alias de fonction Lambda nommé routing-alias qui pointe vers la version 1 de la fonction. La version 2 de la fonction reçoit 3 % du trafic. Les 97 % restants du trafic sont acheminés vers la version 1.

```
aws lambda create-alias \
  --name routing-alias \
  --function-name my-function \
  --function-version 1  \
  --routing-config AdditionalVersionWeights={"2"=0.03}
```

Utilisez la commande `update-alias` pour augmenter le pourcentage de trafic entrant vers la version 2. Dans l'exemple suivant, vous augmentez le trafic à 5 %.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function \
  --routing-config AdditionalVersionWeights={"2"=0.05}
```

Pour acheminer l'ensemble du trafic vers la version 2, utilisez la commande `update-alias` pour modifier la propriété `function-version` afin que l'alias pointe vers la version 2. La commande réinitialise également la configuration de routage.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function  \
  --function-version 2 \
  --routing-config AdditionalVersionWeights={}
```

Les AWS CLI commandes des étapes précédentes correspondent aux opérations Lambda API suivantes :

- CreateAlias
- UpdateAlias

## Déterminer quelle version a été invoquée

Lorsque vous configurez des pondérations de trafic entre deux versions de fonction, vous pouvez déterminer la version de fonction Lambda appelée de deux manières :

- CloudWatch Journaux : Lambda envoie automatiquement une entrée de `START` journal contenant l'ID de version invoqué à Amazon CloudWatch Logs pour chaque appel de fonction. Voici un exemple :

  `19:44:37 START RequestId: request id Version: $version`

  Pour les appels d'alias, Lambda utilise la dimension `Executed Version` pour filtrer les métriques en fonction de la version appelée. Pour de plus amples informations, veuillez consulter Afficher les métriques des fonctions Lambda.
- Charge utile de réponse (appels synchrones) – Les réponses aux appels de fonction synchrones incluent un en-tête `x-amz-executed-version` indiquant la version de fonction appelée.

# Gérer les versions des fonctions Lambda

Vous pouvez utiliser des versions pour gérer le déploiement de vos fonctions. Par exemple, vous pouvez publier une nouvelle version d'une fonction à des fins de test bêta sans affecter les utilisateurs de la version de production stable. Lambda crée une nouvelle version de votre fonction chaque fois que vous la publiez. La nouvelle version est une copie de la version non publiée de la fonction. La version inédite est nommée`$LATEST`.

> **Note**
>
> Pour créer une nouvelle version de votre fonction, vous devez d'abord apporter des modifications à la version non publiée (`$LATEST`). Ces modifications peuvent inclure la mise à jour du code ou la modification des paramètres de configuration. Si elle `$LATEST` est identique à une version publiée précédemment, vous ne pourrez pas créer de nouvelle version tant que vous n'aurez pas déployé les modifications apportées à`$LATEST`.

Une fois que vous avez publié une version de fonction, son code, son environnement d'exécution, son architecture, sa mémoire, ses couches et la plupart des autres paramètres de configuration sont immuables. Cela signifie que vous ne pouvez pas modifier ces paramètres sans publier une nouvelle version à partir de`$LATEST`. Vous pouvez configurer les éléments suivants pour une version de fonction publiée :

- Déclencheurs
- Destinations
- Simultanéité allouée
- Appel asynchrone
- Connexions aux bases de données et proxys

> **Note**
>
> Lorsque vous utilisez les contrôles de gestion de l'exécution en mode automatique, la version d'exécution utilisée par la version de la fonction est automatiquement mise à jour. Lorsque vous utilisez le mode Function update (Mise à jour de fonction) ou Manual (Manuel), la version de l'environnement d'exécution n'est pas mise à jour. Pour de plus amples informations, veuillez consulter the section called “mises à jour des versions d'exécution”.

Sections


- Création de versions de fonction
- Utilisation des versions
- Octroi d'autorisations


## Création de versions de fonction

Vous pouvez modifier le code et les paramètres de la fonction uniquement sur la version non publiée d'une fonction. Lorsque vous publiez une version, Lambda verrouille le code et la plupart des paramètres afin de préserver une expérience cohérente pour les utilisateurs de cette version.

Vous pouvez créer une version de fonction à l'aide de la console Lambda.

Pour créer une version de fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez une fonction, puis choisissez Versions.
3. Sur la page de configuration des versions, sélectionnez Publish new version (Publier une nouvelle version).
4. (Facultatif) Entrez une description de la version.
5. Choisissez Publish.

Vous pouvez également publier une version d'une fonction à l'aide de l'PublishVersionAPIopération.

La AWS CLI commande suivante publie une nouvelle version d'une fonction. La réponse renvoie des informations de configuration concernant la nouvelle version, notamment le numéro de version et la fonction ARN avec le suffixe de version.

```
aws lambda publish-version --function-name my-function
```

Vous devriez voir la sortie suivante :

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
  "Version": "1",
```

```
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs20.x",
  ...
}
```

> **Note**
>
> Lambda attribue des numéros de séquence à croissance monotone pour la gestion des versions. Lambda ne réutilise jamais les numéros de version, même après avoir supprimé et recréé une fonction.

## Utilisation des versions

Vous pouvez référencer votre fonction Lambda en utilisant une fonction qualifiée ARN ou non qualifiée. ARN

- ARNQualifiée : fonction ARN dotée d'un suffixe de version. L'exemple suivant fait référence à la version 42 de la fonction `helloworld`.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld:42
  ```

- Non qualifiée ARN : fonction ARN sans suffixe de version.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld
  ```

Vous pouvez utiliser un qualifié ou un non qualifié ARN dans toutes les API opérations pertinentes. Cependant, vous ne pouvez pas utiliser un nom non qualifié ARN pour créer un alias.

Si vous décidez de ne pas publier les versions des fonctions, vous pouvez invoquer la fonction qualifiée ou non qualifiée ARN dans le mappage des sources de votre événement. Lorsque vous appelez une fonction à l'aide d'une fonction non qualifiéeARN, Lambda l'invoque implicitement. `$LATEST`

Lambda publie une nouvelle version de fonction uniquement si le code n'a jamais été publié ou s'il a changé par rapport à la dernière version publiée. S'il n'y a pas de changement, la version de fonction publiée la plus récente reste active.

Le ARN qualificatif pour chaque version de fonction Lambda est unique. Une fois que vous avez publié une version, vous ne pouvez pas modifier le code ARN ou le code de fonction.

## Octroi d’autorisations

Vous pouvez utiliser une stratégie basée sur les ressources ou une stratégie basée sur l’identité pour accorder l’accès à votre fonction. La portée de l’autorisation est dépendante du fait que vous appliquiez la stratégie à une fonction ou à une version d’une fonction. Pour de plus amples informations sur les noms de ressource de fonction dans les stratégies, veuillez consulter Affiner les sections Ressources et Conditions des politiques.

Vous pouvez simplifier la gestion des sources d'événements et AWS Identity and Access Management (IAM) des politiques en utilisant des alias de fonction. Pour de plus amples informations, veuillez consulter Création d'un alias pour une fonction Lambda.

# Utilisation de balises sur les fonctions Lambda

Vous pouvez étiqueter des fonctions pour organiser et gérer vos ressources. Les balises sont des paires clé-valeur de forme libre associées à vos ressources et prises en charge par l'ensemble des services. AWS Pour plus d'informations sur les cas d'utilisation des balises, consultez la section Stratégies de balisage courantes dans le guide des AWS ressources de balisage et de l'éditeur de balises.

Les balises s’appliquent au niveau de la fonction, pas aux versions ni aux alias. Les balises ne font pas partie de la configuration spécifique à la version qui AWS Lambda crée un instantané du moment où vous publiez une version. Vous pouvez utiliser le Lambda API pour afficher et mettre à jour les balises. Vous pouvez également afficher et mettre à jour les balises tout en gérant une fonction spécifique dans la console Lambda.

Sections


- Autorisations requises pour l’utilisation des balises
- Utilisation des balises avec la console Lambda
- Utilisation de balises avec AWS CLI


## Autorisations requises pour l’utilisation des balises

Pour autoriser une AWS Identity and Access Management (IAM) identité (utilisateur, groupe ou rôle) à lire ou à définir des balises sur une ressource, accordez-lui les autorisations correspondantes :

- lambda : ListTags —Lorsqu' une ressource possède des balises, accordez cette autorisation à tous ceux qui ont besoin de `ListTags` l'utiliser. Pour les fonctions balisées, cette autorisation est également nécessaire pour`GetFunction`.
- lambda : TagResource —Accordez cette autorisation à toute personne ayant besoin d'appeler `TagResource` ou d'exécuter un tag lors de la création.

Pour de plus amples informations, veuillez consulter IAMPolitiques basées sur l'identité pour Lambda.

## Utilisation des balises avec la console Lambda

Vous pouvez utiliser la console Lambda pour créer des fonctions qui comportent des balises, pour ajouter des balises aux fonctions existantes et pour filtrer des fonctions selon les balises ajoutées.

Ajout de balises lors de la création d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Choisissez Author from scratch (Créer à partir de zéro) ou Container image (Image de conteneur).
4. Sous Informations de base, configurez votre fonction. Pour plus d'informations sur la configuration des fonctions, consultezConfiguration des fonctions.
5. Développez Advanced settings (Paramètres avancés) et sélectionnez Enable tags (Activer les balises).
6. Pour cela, choisissez Ajouter une balise, puis saisissez une clé et éventuellement une valeur. Répétez cette étape pour ajouter d'autres balises.
7. Sélectionnez Create function (Créer une fonction).

Pour ajouter des balises à une fonction existante

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Sélectionnez Configuration, puis Tags (Balises).
4. Sous Balises, choisissez Gérer les balises.
5. Pour cela, choisissez Ajouter une balise, puis saisissez une clé et éventuellement une valeur. Répétez cette étape pour ajouter d'autres balises.
6. Choisissez Save (Enregistrer).

Pour filtrer des fonctions avec des balises

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Cliquez sur le champ de recherche pour afficher la liste des propriétés des fonctions et des clés de balise.
3. Choisissez une clé de balise pour afficher la liste des valeurs utilisées dans la AWS région actuelle.
4. Sélectionnez Utiliser : « tag-name » pour voir toutes les fonctions étiquetées avec cette touche, ou choisissez un opérateur pour filtrer davantage par valeur.

5. Sélectionnez la valeur de votre balise pour filtrer par une combinaison de clé et de valeur de balise.

La barre de recherche prend également en charge la recherche de clés de balise. Saisissez `tag` pour afficher uniquement une liste de clés de balise ou entrez le nom d'une clé pour la rechercher dans la liste.

## Utilisation de balises avec AWS CLI

Vous pouvez ajouter et supprimer des balises sur des ressources Lambda existantes, y compris des fonctions, avec le Lambda. API Vous pouvez également ajouter des balises lors de la création d'une fonction, ce qui vous permet de conserver une ressource étiquetée tout au long de son cycle de vie.

### Mise à jour des balises avec la balise Lambda APIs

Vous pouvez ajouter et supprimer des balises pour les ressources Lambda prises en charge par le biais des opérations TagResourceet UntagResourceAPI.

Vous pouvez appeler ces opérations à l'aide du AWS CLI. Pour ajouter des balises à une ressource existante, utilisez la `tag-resource` commande. Cet exemple ajoute deux balises, dont une avec la clé *`Department`* et un avec la clé *`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pour supprimer des balises, utilisez la commande `untag-resource`. Cet exemple supprime le tag contenant la clé *`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

### Ajouter des balises lors de la création d'une fonction

Pour créer une nouvelle fonction Lambda avec des balises, utilisez l'CreateFunctionAPIopération. Spécifiez le paramètre `Tags`. Vous pouvez appeler cette opération à l'aide de la `create-function` CLI commande et de l'option --tags. Avant d'utiliser le paramètre tags with`CreateFunction`, assurez-vous que votre rôle est autorisé à étiqueter les ressources en plus des autorisations

habituelles nécessaires pour cette opération. Pour plus d'informations sur les autorisations relatives au balisage, consultezthe section called “Autorisations requises pour l’utilisation des balises”. Cet exemple ajoute deux balises, dont une avec la clé *Department* et un avec la clé *CostCenter*.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs20.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

## Affichage des balises d’une fonction

Pour afficher les balises appliquées à une ressource Lambda spécifique, utilisez l'`ListTags`APIopération. Pour plus d'informations, consultez ListTags.

Vous pouvez appeler cette opération à l'aide de la `list-tags` AWS CLI commande en fournissant un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier
```

Vous pouvez afficher les balises appliquées à une ressource spécifique lors de l'GetFunctionAPIopération. Aucune fonctionnalité comparable n'est disponible pour les autres types de ressources.

Vous pouvez appeler cette opération à l'aide de la `get-function` CLI commande :

```
aws lambda get-function --function-name my-function
```

## Filtrer les ressources par tag

Vous pouvez utiliser cette AWS Resource Groups Tagging API GetResourcesAPIopération pour filtrer vos ressources par balises. L’opération `GetResources` reçoit jusqu’à 10 filtres, chaque filtre contenant une clé de balise et jusqu’à 10 valeurs de balise. Vous fournissez `GetResources`avec un `ResourceType` pour filtrer par certains types de ressources.

Vous pouvez appeler cette opération à l'aide de la `get-resources` AWS CLI commande. Pour des exemples d'utilisation`get-resources`, voir get-resources dans la référence des AWS CLIcommandes.

# Streaming de réponses pour les fonctions Lambda

Vous pouvez configurer votre fonction Lambda URLs pour transmettre les charges utiles de réponse aux clients. Le streaming des réponses peut profiter aux applications sensibles à la latence en améliorant les performances du délai jusqu'au premier octet (TTFB). En effet, vous pouvez renvoyer des réponses partielles au client dès qu'elles sont disponibles. En outre, vous pouvez utiliser le streaming de réponses pour créer des fonctions qui renvoient des charges utiles plus importantes. Les charges utiles de la diffusion des réponses ont une limite souple de 20 Mo par rapport à la limite de 6 Mo pour les réponses mises en mémoire tampon. La diffusion d'une réponse signifie également que votre fonction n'a pas besoin de stocker l'intégralité de la réponse en mémoire. Pour les réponses très volumineuses, cela peut réduire la quantité de mémoire que vous devez configurer pour votre fonction.

La vitesse à laquelle Lambda diffuse vos réponses dépend de la taille de la réponse. Le débit de diffusion pour les 6 premiers Mo de réponse de votre fonction n'est pas plafonné. Pour les réponses supérieures à 6 Mo, le reste de la réponse est soumis à une limite de bande passante. Pour plus d'informations sur la bande passante de diffusion, consultez Limitation de la bande passante pour la diffusion des réponses.

Le streaming des réponses entraîne un coût. Pour plus d'informations, consultez AWS Lambda Pricing (Tarification CTlong).

Lambda prend en charge le streaming des réponses sur les exécutions gérées par Node.js. Pour les autres langages, vous pouvez utiliser un environnement d'exécution personnalisé avec une API intégration d'environnement d'exécution personnalisée pour diffuser les réponses ou utiliser l'adaptateur Web Lambda. Vous pouvez diffuser les réponses via la fonction Lambda URLs, le AWS SDK, ou en utilisant le Lambda. InvokeWithResponseStreamAPI

> **Note**
>
> Lorsque vous testez votre fonction via la console Lambda, vous verrez toujours les réponses comme étant mises en mémoire tampon.

Rubriques

- Limitation de la bande passante pour la diffusion des réponses
- Écrire des fonctions Lambda compatibles avec le streaming des réponses

- Invocation d'une fonction activée pour le streaming de réponses à l'aide de la fonction Lambda URLs
- Tutoriel : création d'une fonction Lambda de streaming de réponses avec une URL de la fonction

## Limitation de la bande passante pour la diffusion des réponses

Les 6 premiers Mo de la charge utile de la réponse de votre fonction disposent d'une bande passante non plafonnée. Après cette première rafale, Lambda diffuse votre réponse à un débit maximum de 2. MBps Si les réponses de votre fonction ne dépassent jamais 6 Mo, cette limite de bande passante ne s'applique pas.

> **Note**
>
> Les limites de bande passante s'appliquent uniquement à la charge utile de la réponse de votre fonction, et non à l'accès au réseau par votre fonction.

Le débit de bande passante non plafonnée varie en fonction d'un certain nombre de facteurs, notamment la vitesse de traitement de votre fonction. Vous pouvez normalement vous attendre à un débit supérieur à 2 MBps pour les 6 premiers Mo de réponse de votre fonction. Si votre fonction diffuse une réponse vers une destination située à l'extérieur AWS, le débit de diffusion dépend également de la vitesse de la connexion Internet externe.

## Écrire des fonctions Lambda compatibles avec le streaming des réponses

L'écriture du gestionnaire pour les fonctions de streaming de réponses est différente des modèles de gestionnaire typiques. Lorsque vous écrivez des fonctions de streaming, assurez-vous de faire ce qui suit :

- Enveloppez votre fonction avec le décorateur `awslambda.streamifyResponse()` fourni par les exécutions natives de Node.js.
- Terminez la diffusion de manière élégante afin de vous assurer que le traitement des données est terminé.

## Configuration d'une fonction gestionnaire pour diffuser les réponses

Pour indiquer à l'exécution que Lambda doit diffuser les réponses de votre fonction, vous devez envelopper votre fonction avec le décorateur `streamifyResponse()`. Cela indique à l'exécution d'utiliser le chemin logique approprié pour le streaming des réponses et permet à la fonction de diffuser les réponses.

Le décorateur `streamifyResponse()` accepte une fonction qui accepte les paramètres suivants :

- `event`— Fournit des informations sur l'événement URL d'appel de la fonction, telles que la HTTP méthode, les paramètres de requête et le corps de la demande.
- `responseStream` : fournit un flux inscriptible.
- `context` : fournit des méthodes et des propriétés avec des informations sur l'invocation, la fonction et l'environnement d'exécution.

L'objet `responseStream` est un `writableStream` Node.js. Comme pour tout flux de ce type, vous devez utiliser la méthode `pipeline()`.

Example gestionnaire compatible avec le streaming de réponses

```
const pipeline = require("util").promisify(require("stream").pipeline);
const { Readable } = require('stream');

exports.echo = awslambda.streamifyResponse(async (event, responseStream, _context) => {
  // As an example, convert event to a readable stream.
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));

  await pipeline(requestStream, responseStream);
});
```

Bien que `responseStream` propose la méthode `write()` pour écrire dans le flux, nous vous recommandons d'utiliser `pipeline()` dans la mesure du possible. L'utilisation de `pipeline()` permet de s'assurer que le flux accessible en écriture n'est pas submergé par un flux accessible en lecture plus rapide.

## Fin du flux

Assurez-vous de terminer correctement le flux avant le retour du gestionnaire. La méthode `pipeline()` s'en charge automatiquement.

Pour les autres cas d'utilisation, appelez la méthode `responseStream.end()` pour terminer correctement un flux. Cette méthode signale que plus aucune donnée ne doit être écrite dans le flux. Cette méthode n'est pas nécessaire si vous écrivez dans le flux avec `pipeline()` ou `pipe()`.

Example Exemple de fin d'un flux avec pipeline()

```
const pipeline = require("util").promisify(require("stream").pipeline);

exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
 => {
  await pipeline(requestStream, responseStream);
});
```

Example Exemple de fin d'un flux sans pipeline()

```
exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
 => {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

## Invocation d'une fonction activée pour le streaming de réponses à l'aide de la fonction Lambda URLs

> **Note**
>
> Vous devez appeler votre fonction à l'aide d'une fonction URL pour diffuser les réponses.

Vous pouvez invoquer des fonctions activées par le streaming de réponses en modifiant le mode d'appel de votre fonctionURL. Le mode d'appel détermine API l'opération que Lambda utilise pour appeler votre fonction. Les modes d'invocation disponibles sont les suivants :

- `BUFFERED` : il s'agit de l'option par défaut. Lambda appelle votre fonction à l'aide de l'opération. `Invoke` API Les résultats de l’invocation sont disponibles lorsque la charge utile est complète. La taille de la charge utile maximale est de 6 Mo.

- `RESPONSE_STREAM` : permet à votre fonction de diffuser les résultats de la charge utile au fur et à mesure qu'ils sont disponibles. Lambda appelle votre fonction à l'aide de l'opération. `InvokeWithResponseStream` API La taille maximale des données utiles est de 20 Mo. Toutefois, vous pouvez demander une augmentation du quota.

Vous pouvez toujours appeler votre fonction sans diffusion de réponse en continu en appelant directement l'`Invoke`APIopération. Cependant, Lambda diffuse toutes les charges utiles de réponse pour les invocations qui passent par la fonction URL jusqu'à ce que vous changiez le mode d'appel en. `BUFFERED`

Console

Pour définir le mode d'appel d'une fonction URL (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez le nom de la fonction pour laquelle vous voulez définir le mode d'invocation.
3. Choisissez l'onglet Configuration, puis choisissez Function URL.
4. Sélectionnez Modifier, puis Paramètres supplémentaires.
5. Sous Mode d'invocation, sélectionnez le mode d'invocation de votre choix.
6. Choisissez Save (Enregistrer).

AWS CLI

Pour définir le mode d'appel de URL (AWS CLI) d'une fonction

```
aws lambda update-function-url-config \
  --function-name my-function \
  --invoke-mode RESPONSE_STREAM
```

AWS CloudFormation

Pour définir le mode d'appel de URL (AWS CloudFormation) d'une fonction

```
MyFunctionUrl:
  Type: AWS::Lambda::Url
  Properties:
    AuthType: AWS_IAM
    InvokeMode: RESPONSE_STREAM
```

Pour plus d'informations sur la configuration de la fonctionURLs, consultez la section Fonction Lambda. URLs

## Tutoriel : création d'une fonction Lambda de streaming de réponses avec une URL de la fonction

Dans ce didacticiel, vous créez une fonction Lambda définie comme archive de fichier ZIP avec un point de terminaison d'URL de la fonction qui renvoie un flux de réponses. Pour de plus amples informations sur la configuration des URL de fonctions, consultez Fonction URLs.

### Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines commandes CLI Bash que vous utilisez couramment avec Lambda (par exemple `zip`) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. Les exemples de commandes CLI de ce guide utilisent le formatage Linux. Les commandes qui incluent des documents JSON en ligne doivent être reformatées si vous utilisez la CLI Windows.

## Créer un rôle d'exécution

Créez le rôle d'exécution qui donne à votre fonction Lambda l'autorisation d'accéder aux ressources AWS .

Pour créer un rôle d'exécution

1. Accédez à la page Roles (Rôles) de la AWS Identity and Access Management console (IAM).
2. Sélectionnez Créer un rôle.
3. Créez un rôle avec les propriétés suivantes :
   - Type d'entité sécurisée – Service AWS
   - Cas d'utilisation – Lambda
   - Autorisations — AWSLambdaBasicExecutionRole
   - Nom de rôle – **`response-streaming-role`**

La AWSLambdaBasicExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux sur Amazon Logs. Après avoir créé le rôle, notez son Amazon Resource Name (ARN). Vous en aurez besoin à l'étape suivante.

## Création d'une fonction de streaming de réponses (AWS CLI)

Créez une fonction Lambda de streaming de réponses avec un point de terminaison d'URL de la fonction à l'aide de l' AWS Command Line Interface (AWS CLI).

Pour créer une fonction capable de diffuser des réponses

1. Copiez l'exemple de code suivant dans un fichier nommé `index.mjs`.

```
import util from 'util';
import stream from 'stream';
const { Readable } = stream;
const pipeline = util.promisify(stream.pipeline);

/* global awslambda */
export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));
  await pipeline(requestStream, responseStream);
```

```
});
```

2. Créez un package de déploiement.

```
zip function.zip index.mjs
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`. Remplacez la valeur de `--role` par l'ARN de rôle de l'étape précédente.

```
aws lambda create-function \
  --function-name my-streaming-function \
  --runtime nodejs16.x \
  --zip-file fileb://function.zip \
  --handler index.handler \
  --role arn:aws:iam::123456789012:role/response-streaming-role
```

Pour créer une URL de la fonction

1. Ajoutez une stratégie basée sur les ressources à votre fonction pour autoriser l'accès à l'URL de votre fonction. Remplacez la valeur de `--principal` par votre Compte AWS identifiant.

```
aws lambda add-permission \
  --function-name my-streaming-function \
  --action lambda:InvokeFunctionUrl \
  --statement-id 12345 \
  --principal 123456789012 \
  --function-url-auth-type AWS_IAM \
  --statement-id url
```

2. Créez un point de terminaison d'URL pour la fonction avec la commande `create-function-url-config`.

```
aws lambda create-function-url-config \
  --function-name my-streaming-function \
  --auth-type AWS_IAM \
  --invoke-mode RESPONSE_STREAM
```

## Test du point de terminaison d'URL de fonction

Testez votre intégration en invoquant votre fonction. Vous pouvez ouvrir l'URL de votre fonction dans un navigateur ou utiliser curl.

```
curl --request GET "<function_url>" --user "<key:token>" --aws-sigv4 "aws:amz:us-
east-1:lambda" --no-buffer
```

Notre URL de la fonction utilise le type d'authentification `IAM_AUTH`. Cela signifie que vous devez signer les demandes avec votre clé AWS d'accès et votre clé secrète. Dans la commande précédente, remplacez par `<key:token>` l'ID de la clé d' AWS accès. Entrez votre clé AWS secrète lorsque vous y êtes invité. Si vous n'avez pas votre clé AWS secrète, vous pouvez utiliser des AWS informations d'identification temporaires à la place.

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer le rôle d'exécution

1. Ouvrez la page Roles (Rôles) de la console IAM.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

# Comprendre les méthodes d'appel de fonctions Lambda

Après avoir déployé votre fonction Lambda, vous pouvez l'invoquer de plusieurs manières :

- La console Lambda : utilisez la console Lambda pour créer rapidement un événement de test afin d'appeler votre fonction.
- Le AWS SDK— Utilisez le AWS SDK pour appeler votre fonction par programmation.
- The Invoke API — Utilisez le Lambda Invoke API pour appeler directement votre fonction.
- The AWS Command Line Interface (AWS CLI) — Utilisez la `aws lambda invoke` AWS CLI commande pour appeler directement votre fonction depuis la ligne de commande.
- Un point de terminaison de fonction URL HTTP (S) — Utilisez une fonction URLs pour créer un point de terminaison HTTP (S) dédié que vous pouvez utiliser pour appeler votre fonction.

Toutes ces méthodes sont des moyens directs d'invoquer votre fonction. Dans Lambda, un cas d'utilisation courant consiste à invoquer votre fonction en fonction d'un événement qui se produit ailleurs dans votre application. Certains services peuvent invoquer une fonction Lambda à chaque nouvel événement. C'est ce qu'on appelle un déclencheur. Pour les services basés sur les flux et les files d'attente, Lambda invoque la fonction avec des lots d'enregistrements. C'est ce qu'on appelle un mappage des sources d'événements.

Lorsque vous invoquez une fonction, vous pouvez choisir de le faire de façon synchrone ou asynchrone. Avec une invocation synchrone, vous attendez de la fonction qu'elle traite l'événement et renvoie une réponse. Avec l'invocation asynchrone, Lambda place en file d'attente l'événement à traiter, et renvoie une réponse immédiatement. Le paramètre de `InvocationType` requête dans Invoke API détermine la manière dont Lambda appelle votre fonction. Une valeur de `RequestResponse` indique un appel synchrone et une valeur de `Event` indique un appel asynchrone.

Pour appeler votre fonctionIPv6, utilisez les points de terminaison publics à double pile de Lambda. Les points de terminaison à double pile prennent en charge les deux etIPv4. IPv6 Les points de terminaison à double pile de Lambda utilisent la syntaxe suivante :

```
protocol://lambda.us-east-1.api.aws
```

Vous pouvez également utiliser la fonction Lambda URLs pour appeler des fonctions. IPv6 Les URL points de terminaison des fonctions ont le format suivant :

```
https://url-id.lambda-url.us-east-1.on.aws
```

Si l'invocation de la fonction entraîne une erreur, pour les invocations synchrones, consultez le message d'erreur dans la réponse et réessayez l'appel manuellement. Pour les appels asynchrones, Lambda gère automatiquement les nouvelles tentatives et peut envoyer les enregistrements d'appel à une destination.

# Invoquer une fonction Lambda de manière synchrone

Lorsque vous invoquez une fonction de façon synchrone, Lambda l'exécute et attend une réponse. Une fois l'exécution de la fonction terminée, Lambda renvoie la réponse du code de la fonction avec des données supplémentaires, telles que la version de la fonction qui a été invoquée. Pour appeler une fonction de manière synchrone avec le AWS CLI, utilisez la `invoke` commande.

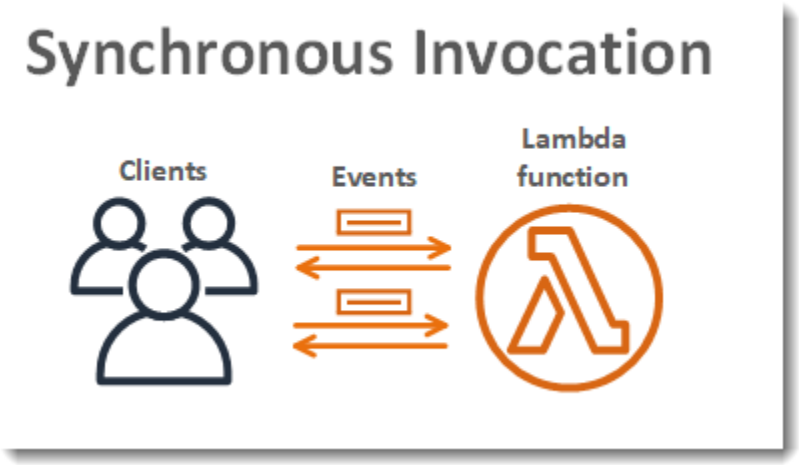
```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Le diagramme suivant montre des clients invoquant une fonction Lambda de manière synchrone. Lambda envoie les événements directement à la fonction et renvoie la réponse de la fonction à l'appelant.



`payload`Il s'agit d'une chaîne qui contient un événement au JSON format. Le nom du fichier où l' AWS CLI écrit la réponse de la fonction est `response.json`. Si la fonction renvoie un objet ou une erreur, le corps de la réponse est l'objet ou l'erreur de JSON format. Si la fonction s'arrête sans erreur, le corps de la réponse est`null`.

> **Note**
>
> Lambda n'attend pas que les extensions externes se terminent avant d'envoyer la réponse. Les extensions externes s'exécutent comme des processus indépendants dans l'environnement d'exécution et continuent de s'exécuter après l'invocation de la fonction. Pour plus d'informations, consultez Augmentez les fonctions Lambda à l'aide des extensions Lambda.

La sortie de la commande qui s'affiche dans le terminal inclut des informations extraites d'en-têtes figurant dans la réponse de Lambda. Ces informations incluent la version qui a traité l'événement (utile lorsque vous utilisez des alias) et le code d'état renvoyé par Lambda. Si Lambda a pu exécuter la fonction, le code d'état est 200, même si la fonction a renvoyé une erreur.

> **Note**
>
> Pour les fonctions avec un long délai d'attente, votre client peut être déconnecté lors d'une invocation synchrone, pendant qu'il attend une réponse. Configurez votre HTTP client, votre pare-feuSDK, votre proxy ou votre système d'exploitation pour autoriser les connexions de longue durée avec des paramètres de temporisation ou de maintien en vie.

Si Lambda n'a pas pu exécuter la fonction, l'erreur s'affiche dans la sortie.

```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload value response.json
```

Vous devriez voir la sortie suivante :

```
An error occurred (InvalidRequestContentException) when calling the Invoke operation:
 Could not parse request body into json: Unrecognized token 'value': was expecting
 ('true', 'false' or 'null')
```

```
 at [Source: (byte[])"value"; line: 1, column: 11]
```

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-`

`in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

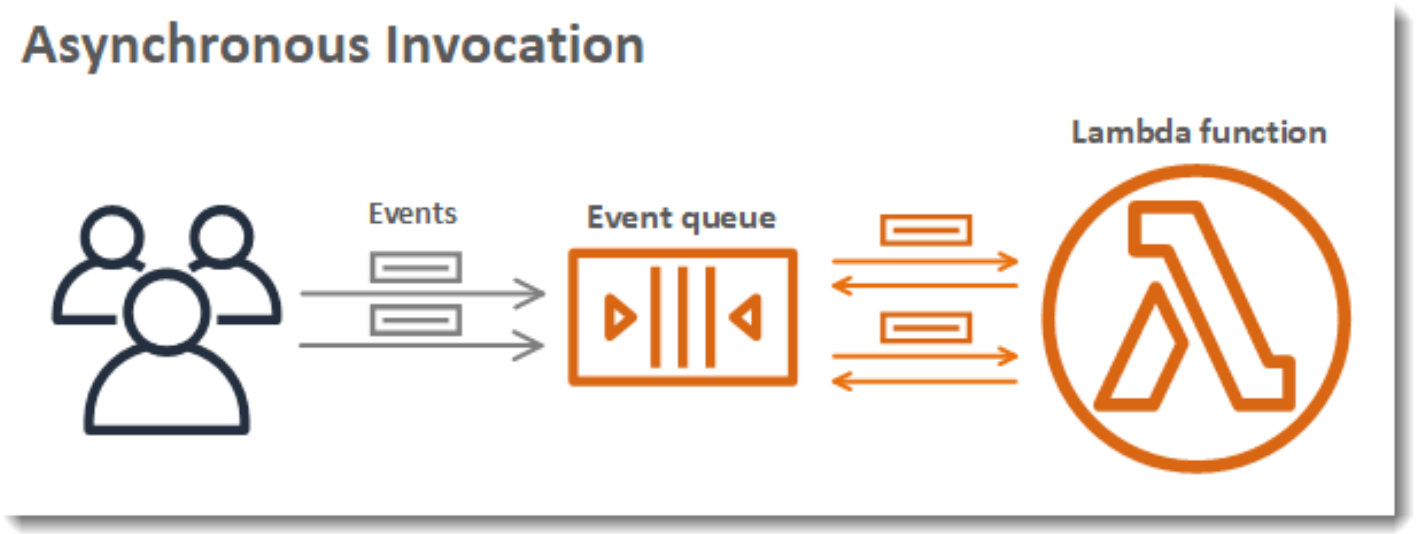
Pour plus d'informations sur le `Invoke`API, y compris une liste complète des paramètres, des en-têtes et des erreurs, consultez Invoke.

Lorsque vous invoquez une fonction directement, vous pouvez vérifier la réponse pour les erreurs et réessayer. Ils AWS CLI peuvent AWS SDK également réessayer automatiquement en cas d'expiration des délais, de limitation et d'erreurs de service du client. Pour de plus amples informations, veuillez consulter Comprendre le comportement des nouvelles tentatives dans Lambda.

# Invocation d'une fonction Lambda de manière asynchrone

Plusieurs d' Services AWS entre eux, tels qu'Amazon Simple Storage Service (Amazon S3) et Amazon Simple Notification Service (SNSAmazon), invoquent des fonctions de manière asynchrone pour traiter les événements. Vous pouvez également appeler une fonction Lambda de manière asynchrone en utilisant le AWS Command Line Interface (AWS CLI) ou l'un des. AWS SDKs Lorsque vous invoquez une fonction de manière asynchrone, vous n'attendez pas de réponse de la part du code de la fonction. Vous remettez l'événement à Lambda qui se charge du reste. Vous pouvez configurer la manière dont Lambda gère les erreurs et envoyer des enregistrements d'invocation à une ressource en aval telle qu'Amazon Simple Queue Service (Amazon) SQS ou Amazon EventBridge (EventBridge) pour enchaîner les composants de votre application.

Le diagramme suivant montre des clients invoquant une fonction Lambda de manière asynchrone. Lambda place les événements en file d'attente avant de les envoyer à la fonction.




Pour une invocation asynchrone, Lambda place l'événement dans une file d'attente et renvoie une réponse de succès sans plus informations. Un processus distinct lit les événements à partir de la file d'attente et les envoie à votre fonction.

Pour appeler une fonction Lambda de manière asynchrone à l'aide de AWS Command Line Interface (AWS CLI) ou de l'un des AWS SDKs, définissez le paramètre sur. [InvocationType](#)`Event` L'exemple suivant montre une AWS CLI commande pour appeler une fonction.

```
aws lambda invoke \
  --function-name my-function  \
  --invocation-type Event \
```

```
  --cli-binary-format raw-in-base64-out \
  --payload '{ "key": "value" }' response.json
```

Vous devriez voir la sortie suivante :

```
{
    "StatusCode": 202
}
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Le fichier de sortie (`response.json`) ne contient pas d'informations, mais il est tout de même créé lorsque vous exécutez cette commande. Si Lambda n'est pas en mesure d'ajouter l'événement à la file d'attente, un message d'erreur s'affiche dans la sortie de la commande.

## Comment Lambda gère les erreurs et les nouvelles tentatives avec un appel asynchrone

Lambda gère la file d'attente d'événements asynchrone de votre fonction et tente de réessayer en cas d'erreur. Si la fonction renvoie une erreur, Lambda tente par défaut de l'exécuter deux fois de plus, avec une minute d'attente entre les deux premières tentatives et deux minutes entre la deuxième et la troisième tentative. Les erreurs de fonction comprennent des erreurs renvoyées par le code de la fonction et des erreurs renvoyées par l'environnement d'exécution de la fonction, comme les délais d'expiration.

Si la fonction n'a pas suffisamment de simultanéité disponible pour traiter tous les événements, des demandes supplémentaires sont bloquées. Pour les erreurs de régulation (429) et les erreurs système (série 500), Lambda renvoie l'événement dans la file d'attente et tente de réexécuter la fonction pendant 6 heures maximum par défaut. L'intervalle de nouvelle tentative augmente de manière exponentielle de 1 seconde après la première tentative jusqu'à un maximum de 5 minutes. Si la file d'attente contient de nombreuses entrées, Lambda augmente l'intervalle entre les tentatives et réduit la fréquence des événements de lecture à partir de la file d'attente.

Même si votre fonction ne renvoie pas d'erreur, elle peut recevoir plusieurs fois le même événement de Lambda, car la file d'attente elle-même est cohérente à terme. Si la fonction ne peut pas à prendre en charge les événements entrants, ils peuvent également être supprimés de la file d'attente sans être envoyés à la fonction. Assurez-vous que le code de votre fonction gère correctement les événements dupliqués et que vous avez suffisamment de simultanéité disponible pour gérer toutes les invocations.

Lorsque la file d'attente est très longue, il peut arriver que de nouveaux événements expirent avant que Lambda ait la possibilité de les envoyer à votre fonction. Quand un événement expire ou échoue à toutes les tentatives de traitement, Lambda le supprime. Vous pouvez configurer la gestion des erreurs pour une fonction de façon à réduire le nombre de nouvelles tentatives que Lambda effectue, ou à supprimer les événements non traités plus rapidement.

Vous pouvez désormais configurer Lambda pour envoyer un enregistrement d'invocation à un autre service. Pour en savoir plus, consultez Capture d'enregistrements d'appels asynchrones Lambda.

## Configuration des paramètres de gestion des erreurs pour les appels asynchrones Lambda

Utilisez les paramètres suivants pour configurer la manière dont Lambda gère les erreurs et les nouvelles tentatives pour les appels de fonctions asynchrones :

- MaximumEventAgeInSeconds: durée maximale, en secondes, pendant laquelle Lambda conserve un événement dans la file d'attente d'événements asynchrones avant de le supprimer.
- MaximumRetryAttempts: le nombre maximum de fois que Lambda réessaie des événements lorsque la fonction renvoie une erreur.

Utilisez la console Lambda ou AWS CLI pour configurer les paramètres de gestion des erreurs sur une fonction, une version ou un alias.

Console

Pour configurer la gestion des erreurs

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Configuration, puis Asynchronous invocation (Invocation asynchrone).
4. Sous Asynchronous invocation (Invocation asynchrone), choisissez Edit (Modifier).

5. Configurez les paramètres suivants.
   - Maximum age of event (Âge maximal de l'événement) – Durée maximale, jusqu'à 6 heures, pendant laquelle Lambda conserve un événement dans la file d'attente d'événements asynchrones.
   - Retry attempts (Nouvelles tentatives) – Nombre de nouvelles tentatives, entre 0 et 2, que Lambda effectue lorsque la fonction renvoie une erreur.
6. Choisissez Save (Enregistrer).

AWS CLI

Pour configurer l'invocation asynchrone avec le AWS CLI, utilisez la put-function-event-invoke commande -config. L'exemple suivant montre comment configurer une fonction avec un âge d'événement maximal de 1 heure et aucune nouvelle tentative.

```
aws lambda put-function-event-invoke-config \
  --function-name error \
  --maximum-event-age-in-seconds 3600 \
  --maximum-retry-attempts 0
```

La commande `put-function-event-invoke-config` remplace toute configuration existante sur la fonction, la version ou l'alias. Pour configurer une option sans en réinitialiser d'autres, utilisez update-function-event-invoke-config. L'exemple suivant configure Lambda pour envoyer un enregistrement à une file d'attente SQS standard `destination` nommée lorsqu'un événement ne peut pas être traité.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Vous devriez voir la sortie suivante :

```
{
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
    "MaximumRetryAttempts": 0,
```

```
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

Quand un événement d'invocation dépasse l'âge maximum ou échoue à toutes les tentatives, Lambda le supprime. Pour conserver une copie des événements ignorés, configurez une destination pour les événements ayant échoué.

## Capture d'enregistrements d'appels asynchrones Lambda

Lambda peut envoyer des enregistrements d'appels asynchrones à l'une des entités suivantes. Services AWS

- Amazon SQS — Une SQS file d'attente standard.
- Amazon SNS — Un SNS sujet standard.
- AWS Lambda – Fonction Lambda.
- Amazon EventBridge — Un bus EventBridge événementiel.

L'enregistrement d'invocation contient les détails de la demande et de la réponse au JSON format. Vous pouvez configurer des destinations distinctes pour les événements qui ont été traités avec succès et ceux dont le traitement a échoué. Vous pouvez également configurer une file d'SQSattente Amazon standard ou une SNS rubrique Amazon standard sous forme de file d'attente de lettres mortes pour les événements ignorés. Pour les files d'attente de lettres mortes, Lambda envoie uniquement le contenu de l'événement, sans plus d'informations sur la réponse.

Si Lambda ne parvient pas à envoyer un enregistrement vers une destination que vous avez configurée, il envoie une `DestinationDeliveryFailures` métrique à Amazon. CloudWatch Cela peut se produire si votre configuration inclut un type de destination non pris en charge, tel qu'une SQS FIFO file d'attente Amazon ou un SNS FIFO sujet Amazon. Des erreurs de remise peuvent également se produire en raison d'erreurs d'autorisations et de limites de taille. Pour plus d'informations sur les métriques d'invocation Lambda, consultez Métriques d'invocation.

> **Note**
>
> Pour empêcher une fonction de se déclencher, vous pouvez définir la simultanéité réservée de la fonction sur zéro. Lorsque vous définissez la simultanéité réservée sur zéro pour une fonction invoquée de manière asynchrone, Lambda commence à envoyer les nouveaux événements à la file d'attente de lettres mortes ou à la destination d'événement en situation d'échec, sans aucune nouvelle tentative. Pour traiter les événements qui ont été envoyés alors que la simultanéité réservée était définie sur zéro, vous devez consommer les événements de la file d'attente de lettres mortes ou de la destination des événements en situation d'échec.

## Ajouter une destination

Pour retenir les enregistrements des invocations asynchrones, ajoutez une destination à votre fonction. Vous pouvez choisir d'envoyer les invocations réussies ou non à une destination. Chaque fonction peut avoir plusieurs destinations. Vous pouvez donc configurer des destinations distinctes pour les événements réussis et échoués. Chaque enregistrement envoyé à la destination est un JSON document contenant les détails de l'invocation. Comme les paramètres de gestion des erreurs, vous pouvez configurer des destinations au niveau d'une fonction, d'une version de fonction ou d'un alias.

> **Note**
>
> Vous pouvez également conserver les enregistrements des appels ayant échoué pour les types de mappage des sources d'événements suivants : Amazon Kinesis, AmazonDynamoDB, Apache Kafka autogéré et Amazon. MSK

Le tableau suivant répertorie les destinations prises en charge pour les enregistrements d'invocation asynchrones. Pour que Lambda envoie correctement les enregistrements vers la destination que vous avez choisie, assurez-vous que le rôle d'exécution de votre fonction contient également les autorisations appropriées. Le tableau décrit également comment chaque type de destination reçoit l'enregistrement JSON d'appel.

| Type de destination | Autorisation obligatoire | Format spécifique à la destination JSON |
| --- | --- | --- |
| SQSFile d'attente Amazon | sqs : SendMessage | Lambda transmet l'enregis trement d'invocation en tant que `Message` à la destination. |
| SNSRubrique Amazon | sns:Publish | Lambda transmet l'enregis trement d'invocation en tant que `Message` à la destination. |
| Fonction Lambda | InvokeFunction | Lambda transmet l'enregis trement d'invocation comme charge utile à la fonction. |
| EventBridge | événements : PutEvents | • Lambda transmet l'enregis trement d'invocation tel qu'il figure `detail` dans l'appel. PutEvents<br>• La valeur du champ événement `source` est `lambda`.<br>• La valeur du champ événement `detail-type` est soit Résultat de l'invocat ion de la fonction lambda – Succès, soit Résultat de l'invocation de la fonction lambda – Échec.<br>• Le champ `resource` d'événement contient la fonction et la destination Amazon Resource Names (ARNs).<br>• Pour les autres champs relatifs aux événement |

| Type de destination | Autorisation obligatoire | Format spécifique à la destination JSON |
| --- | --- | --- |
| | | s, consultez Amazon EventBridge events. |

Les étapes suivantes décrivent comment configurer une destination pour une fonction à l'aide de la console Lambda et du. AWS CLI

Console

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Asynchronous invocation (Invocation asynchrone).
5. Pour Condition choisissez l'une des options suivantes :
   - On failure (En cas d'échec) – Envoyer un enregistrement quand l'événement échoue à toutes les tentatives de traitement ou dépasse l'âge maximal.
   - On success (En cas de succès) – Envoyer un enregistrement quand la fonction traite avec succès une invocation asynchrone.
6. Pour Type de destination, choisissez le type de ressource qui reçoit l'enregistrement d'invocation.
7. Pour Destination, choisissez une ressource.
8. Choisissez Save (Enregistrer).

AWS CLI

Pour configurer une destination à l'aide de AWS CLI, exécutez la commande update-function-event-invoke-config. L'exemple suivant configure Lambda pour envoyer un enregistrement à une file d'attente SQS standard `destination` nommée lorsqu'un événement ne peut pas être traité.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
```

```
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Lorsqu'un appel répond à la condition, Lambda envoie JSON un document contenant les détails de l'appel à la destination. L'exemple suivant illustre un enregistrement d'invocation pour un événement dont le traitement a échoué à trois reprises en raison d'une erreur de fonction.

Example enregistrement d'invocation

```
{
    "version": "1.0",
    "timestamp": "2019-11-14T18:16:05.568Z",
    "requestContext": {
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
        "condition": "RetriesExhausted",
        "approximateInvokeCount": 3
    },
    "requestPayload": {
        "ORDER_IDS": [
            "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",
            "637de236-e7b2-464e-xmpl-baf57f86bb53",
            "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"
        ]
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "responsePayload": {
        "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited
 before completing request"
    }
}
```

L'enregistrement d'invocation contient des détails sur l'événement, la réponse et la raison pour laquelle l'enregistrement a été envoyé.

Suivi des demandes vers les destinations

Vous pouvez utiliser AWS X-Ray pour visualiser une vue connectée de chaque demande lorsqu'elle est placée en file d'attente, traitée par une fonction Lambda et transmise au service de destination. Lorsque vous activez le suivi X-Ray pour une fonction ou un service qui invoque une fonction, Lambda ajoute un en-tête X-Ray à la demande et transmet l'en-tête au service de destination. Les traces des services en amont sont automatiquement liées aux traces des fonctions Lambda en aval et des services de destination, créant ainsi une end-to-end vue de l'ensemble de l'application. Pour plus d'informations sur le suivi, consultez Visualisez les invocations de fonctions Lambda à l'aide de AWS X-Ray.

## Ajouter une file d'attente de lettres mortes

Comme alternative à une destination réservée aux échecs, vous pouvez configurer votre fonction avec une file d'attente de lettres mortes pour enregistrer les événements ignorés en vue d'un traitement ultérieur. Une file d'attente de lettres mortes agit de la même manière qu'une destination réservée aux échecs en ce sens qu'elle est utilisée lorsqu'un événement échoue à toutes les tentatives de traitement ou expire sans être traité. Toutefois, vous ne pouvez ajouter ou supprimer une file d'attente de lettres mortes qu'au niveau de la fonction. Les versions de fonction utilisent les mêmes paramètres de file d'attente de lettres mortes que la version non publiée ($). LATEST Les destinations réservées aux échecs prennent également en charge des cibles supplémentaires et incluent des détails sur la réponse de la fonction dans l'enregistrement d'invocation.

Pour retraiter les événements d'une file d'attente contenant des lettres mortes, vous pouvez la définir comme source d'événements pour votre fonction Lambda. Vous pouvez également récupérer manuellement les événements.

Vous pouvez choisir une file d'attente SQS standard Amazon ou un sujet SNS standard Amazon pour votre file d'attente de lettres mortes. FIFOles files d'attente et les SNS FIFO sujets Amazon ne sont pas pris en charge.

- SQSFile d'attente Amazon : une file d'attente contient les événements ayant échoué jusqu'à ce qu'ils soient récupérés. Choisissez une file d'attente SQS standard Amazon si vous vous attendez à ce qu'une seule entité, telle qu'une fonction Lambda ou une CloudWatch alarme, traite l'événement défaillant. Pour de plus amples informations, veuillez consulter Utilisation de Lambda avec Amazon SQS.
- SNSRubrique Amazon : une rubrique transmet les événements ayant échoué à une ou plusieurs destinations. Choisissez un sujet SNS standard d'Amazon si vous vous attendez à ce que plusieurs entités agissent en cas d'échec d'un événement. Par exemple, vous pouvez configurer une

rubrique pour envoyer des événements à une adresse e-mail, à une fonction Lambda et/ou à un point de terminaison. HTTP Pour de plus amples informations, veuillez consulter Invocation de fonctions Lambda avec les notifications Amazon SNS.

Pour envoyer des événements à une file d'attente ou une rubrique, votre fonction a besoin d'autorisations supplémentaires. Ajoutez une politique avec les autorisations requises au rôle d'exécution de votre fonction.

Si la file d'attente ou la rubrique cible est chiffrée avec une clé gérée par le client, le rôle d'exécution doit également être un utilisateur dans la stratégie basée sur les ressources de la clé.

Après avoir créé la cible et mis à jour votre rôle d'exécution de la fonction, ajoutez la file d'attente de lettres mortes à votre fonction. Vous pouvez configurer plusieurs fonctions pour envoyer des événements à la même cible.

Console

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Configuration, puis Asynchronous invocation (Invocation asynchrone).
4. Sous Asynchronous invocation (Invocation asynchrone), choisissez Edit (Modifier).
5. Définissez le service de file d'attente des lettres mortes sur Amazon ou SQS Amazon. SNS
6. Choisissez la file d'attente ou la rubrique cible.
7. Choisissez Save (Enregistrer).

AWS CLI

Pour configurer une file d'attente contenant des lettres mortes avec le AWS CLI, utilisez la update-function-configurationcommande.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --dead-letter-config TargetArn=arn:aws:sns:us-east-1:123456789012:my-topic
```

Lambda envoie l'événement à la file d'attente de lettres mortes en l'état, avec des informations supplémentaires dans les attributs. Vous pouvez utiliser ces informations pour identifier l'erreur que la

fonction a renvoyée ou établir une corrélation entre l'événement et des journaux ou une trace AWS X-Ray .

Attributs de message de file d'attente de lettres mortes

- RequestID (Chaîne) – ID de la demande d'invocation. IDsLes demandes apparaissent dans les journaux des fonctions. Vous pouvez également utiliser le X-Ray SDK pour enregistrer l'ID de demande sur un attribut de la trace. Vous pouvez ensuite rechercher des suivis par ID de demande dans la console X-Ray.
- ErrorCode(Numéro) — Le code HTTP d'état.
- ErrorMessage(String) — Les 1 premiers Ko du message d'erreur.

Si Lambda ne parvient pas à envoyer de message à la file d'attente contenant des lettres mortes, il supprime l'événement et émet la métrique. DeadLetterErrors Cela peut se produire en raison d'un manque d'autorisations ou si la taille totale du message est supérieure à la limite de la file d'attente cible ou rubrique. Supposons, par exemple, qu'une SNS notification Amazon dont le corps est proche de 256 Ko déclenche une fonction qui génère une erreur. Dans ce cas, les données d'événement ajoutées par AmazonSNS, combinées aux attributs ajoutés par Lambda, peuvent faire en sorte que le message dépasse la taille maximale autorisée dans la file d'attente de lettres mortes.

Si vous utilisez Amazon SQS comme source d'événements, configurez une file d'attente de lettres mortes sur la file d'attente Amazon SQS elle-même et non sur la fonction Lambda. Pour de plus amples informations, veuillez consulter Utilisation de Lambda avec Amazon SQS.

# Comment Lambda traite les enregistrements provenant de sources d'événements basées sur des flux et des files d'attente

Un mappage de source d'événements est une ressource Lambda qui lit des éléments provenant de services basés sur des flux et des files d'attente et invoque une fonction avec des lots d'enregistrements. Les services suivants utilisent des mappages de sources d'événements pour appeler des fonctions Lambda :

- Amazon DocumentDB (compatible avec MongoDB) (Amazon DocumentDB)
- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming pour Apache Kafka (Amazon) MSK
- Self-managed Apache Kafka
- Service de file d'attente Amazon Simple (AmazonSQS)

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

## En quoi les mappages de sources d'événements diffèrent des déclencheurs directs

Certains AWS services peuvent appeler directement des fonctions Lambda à l'aide de déclencheurs. Ces services transmettent les événements à Lambda, et la fonction est invoquée immédiatement lorsque l'événement spécifié se produit. Les déclencheurs sont adaptés aux événements discrets et au traitement en temps réel. Lorsque vous créez un déclencheur à l'aide de la console Lambda, celle-ci interagit avec le AWS service correspondant pour configurer la notification d'événement sur ce service. Le déclencheur est en fait stocké et géré par le service qui génère les événements, et non

par Lambda. Voici quelques exemples de services qui utilisent des déclencheurs pour appeler des fonctions Lambda :

- Amazon Simple Storage Service (Amazon S3) : invoque une fonction lorsqu'un objet est créé, supprimé ou modifié dans un compartiment. Pour de plus amples informations, veuillez consulter Didacticiel : utilisation d'un déclencheur Amazon S3 pour invoquer une fonction Lambda.
- Amazon Simple Notification Service (AmazonSNS) : invoque une fonction lorsqu'un message est publié dans un SNS sujet. Pour de plus amples informations, veuillez consulter Tutoriel : Utilisation AWS Lambda avec Amazon Simple Notification Service.
- Amazon API Gateway : invoque une fonction lorsqu'une API demande est envoyée à un point de terminaison spécifique. Pour de plus amples informations, veuillez consulter Invocation d'une fonction Lambda à l'aide d'un point de terminaison Amazon Gateway API.

Les mappages de sources d'événements sont des ressources Lambda créées et gérées au sein du service Lambda. Les mappages de sources d'événements sont conçus pour traiter de gros volumes de données en streaming ou de messages provenant de files d'attente. Le traitement par lots des enregistrements d'un flux ou d'une file d'attente est plus efficace que le traitement individuel des enregistrements.

## Comportement de traitement par lots

Par défaut, un mappage de source d'événements regroupe des enregistrements dans une même charge utile que Lambda envoie à votre fonction. Pour affiner le comportement du traitement par lots, vous pouvez configurer une fenêtre de traitement par lots (MaximumBatchingWindowInSeconds) et une taille de lot (). BatchSize Une fenêtre de traitement par lots représente l'intervalle de temps maximal pour collecter des enregistrements dans une même charge utile. La taille d'un lot est le nombre maximal d'enregistrements dans un même lot. Lambda invoque votre fonction en présence de l'un des trois critères suivants :

- La fenêtre de traitement par lots atteint sa valeur maximale. Le comportement par défaut de la fenêtre de traitement par lots varie en fonction de la source d'événement spécifique.
  - Pour les sources d'événements Kinesis, DynamoDB et SQS Amazon : la fenêtre de traitement par lots par défaut est de 0 seconde. Cela signifie que Lambda appelle votre fonction dès que les enregistrements sont disponibles. Pour définir une fenêtre de traitement par lots, configurez`MaximumBatchingWindowInSeconds`. Vous pouvez définir ce paramètre sur n'importe quelle valeur comprise entre 0 et 300 secondes par incréments de 1 seconde. Si vous

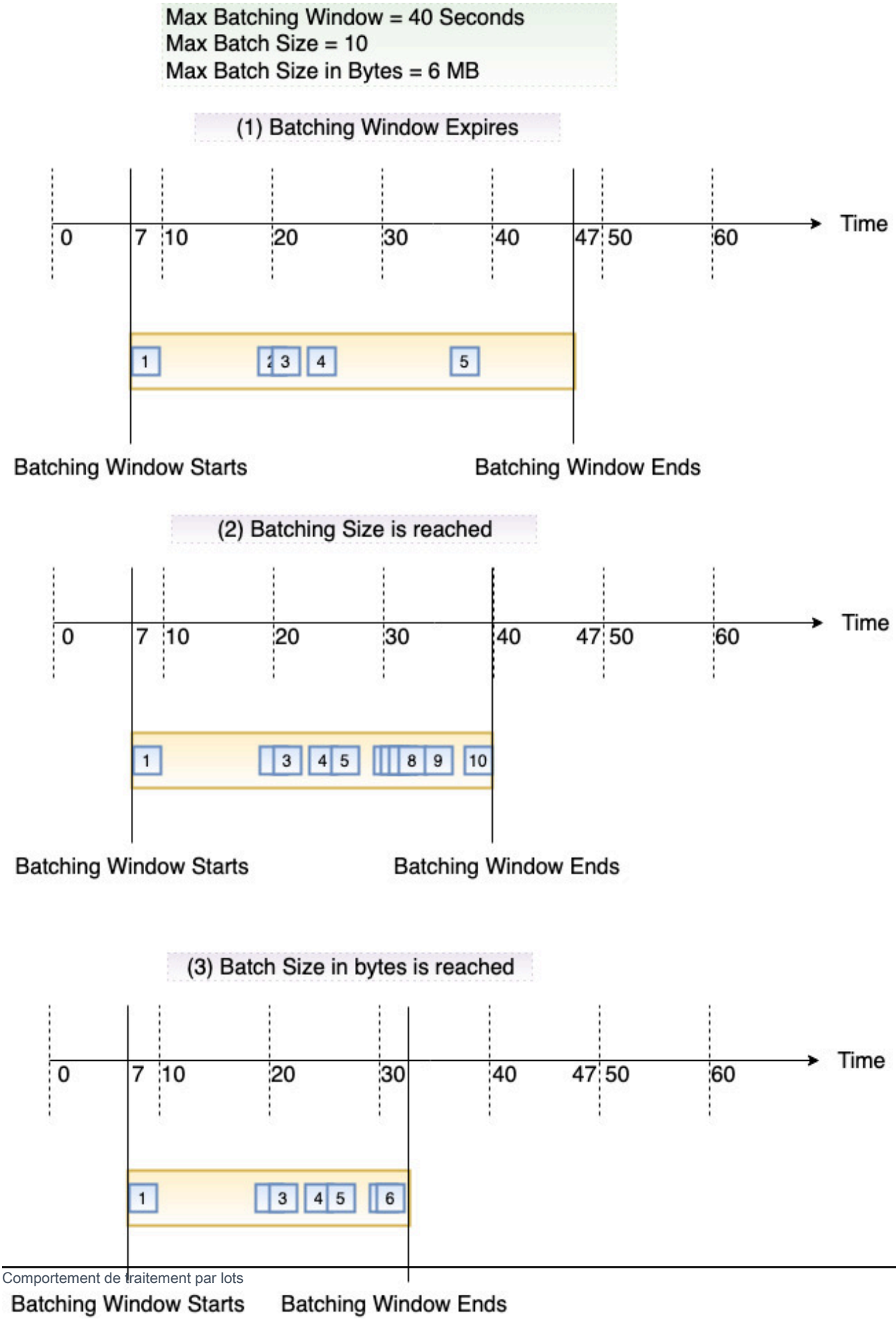
configurez une fenêtre de traitement par lots, la fenêtre suivante commence dès que l'appel de fonction précédent est terminé.

- Pour AmazonMSK, les sources d'événements autogérées Apache Kafka, Amazon MQ et Amazon DocumentDB : la fenêtre de traitement par lots par défaut est de 500 ms. Vous pouvez configurer `MaximumBatchingWindowInSeconds` à n'importe quelle valeur comprise entre 0 et 300 secondes par incréments de secondes. Une fenêtre de traitement par lots commence dès l'arrivée du premier registre.

  > **Note**
  >
  > Comme vous ne pouvez modifier `MaximumBatchingWindowInSeconds` que par incréments de quelques secondes, vous ne pouvez pas revenir à la fenêtre de traitement par lots par défaut de 500 ms après l'avoir modifiée. Pour restaurer la fenêtre de traitement par lots par défaut, vous devez créer un mappage de source d'événement.

- La taille du lot est atteinte. La taille minimale du lot est de 1. La taille par défaut et la taille maximale du lot dépendent de la source d'événement. Pour plus de détails sur ces valeurs, consultez les BatchSizespécifications de l'`CreateEventSourceMapping`APIopération.
- La taille de la charge utile atteint 6 Mo. Vous ne pouvez pas modifier cette limite.

Le diagramme suivant illustre ces trois conditions. Supposons qu'une fenêtre de traitement par lots commence à `t = 7` secondes. Dans le premier scénario, la fenêtre de traitement par lots atteint son maximum de 40 secondes à `t = 47` secondes après avoir accumulé 5 enregistrements. Dans le second scénario, la taille du lot atteint 10 avant l'expiration de la fenêtre de traitement par lots, de sorte que la fenêtre de traitement par lots se termine plus tôt. Dans le troisième scénario, la taille maximale de la charge utile est atteinte avant l'expiration de la fenêtre de traitement par lots, de sorte que la fenêtre de traitement par lots se termine plus tôt.


Max Batching Window = 40 Seconds
Max Batch Size = 10
Max Batch Size in Bytes = 6 MB
(1) Batching Window Expires
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
Batching Window Starts
Batching Window Ends
(2) Batching Size is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
8
9
10
Batching Window Starts
Batching Window Ends
(3) Batch Size in bytes is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
6
Batching Window Starts
Batching Window Ends

Nous vous recommandons de tester avec différentes tailles de lot et d'enregistrement afin que la fréquence d'interrogation de chaque source d'événements soit adaptée à la rapidité avec laquelle votre fonction est capable d'accomplir sa tâche. Le CreateEventSourceMapping BatchSize paramètre contrôle le nombre maximum d'enregistrements qui peuvent être envoyés à votre fonction à chaque appel. Une taille de lot plus grande peut souvent absorber plus efficacement l'invocation sur un plus large ensemble d'enregistrements, ce qui accroît votre débit.

Lambda n'attend pas que les extensions configurées soient terminées avant d'envoyer le lot suivant pour traitement. En d'autres termes, vos extensions peuvent continuer à s'exécuter pendant que Lambda traite le prochain lot d'enregistrements. Cela peut causer des problèmes de limitation si vous enfreignez l'un des paramètres ou l'une des limites de simultanéité de votre compte. Pour détecter s'il s'agit d'un problème éventuel, surveillez vos fonctions et vérifiez si vous observez des métriques de simultanéité plus élevées que prévu pour votre mappage des sources d'événements. En raison de la brièveté des intervalles entre les invocations, Lambda peut brièvement signaler une utilisation simultanée supérieure au nombre de partitions. Cela peut être vrai même pour les fonctions Lambda sans extensions.

Par défaut, si votre fonction renvoie une erreur, le mappage de la source d'événements retraite l'ensemble du lot jusqu'à ce que la fonction réussisse ou que les éléments du lot arrivent à expiration. Pour s'assurer d'un traitement dans l'ordre, le mappage de la source d'événement met en pause le traitement dans la partition concernée jusqu'à la résolution de l'erreur. Pour les sources de flux (DynamoDB et Kinesis), vous pouvez configurer le nombre maximal de tentatives que Lambda effectue lorsque votre fonction renvoie une erreur. Les erreurs de service ou les limitations où le lot n'atteint pas votre fonction ne comptent pas dans les tentatives de réessai. Vous pouvez également configurer le mappage de la source d'événements pour envoyer un enregistrement d'appel vers une destination lorsqu'il supprime un lot d'événements.

## Cartographie des sources d'événements API

Pour gérer une source d'événement avec le AWS Command Line Interface (AWS CLI) ou un AWS SDK, vous pouvez utiliser les API opérations suivantes :

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

# Utilisation de balises sur les mappages de sources d'événements

Vous pouvez baliser les mappages de sources d'événements pour organiser et gérer vos ressources. Les balises sont des paires clé-valeur de forme libre associées à vos ressources et prises en charge par l'ensemble des services. AWS Pour plus d'informations sur les cas d'utilisation des balises, consultez la section Stratégies de balisage courantes dans le guide des AWS ressources de balisage et de l'éditeur de balises.

Les mappages de sources d'événements sont associés à des fonctions, qui peuvent avoir leurs propres balises. Les mappages de sources d'événements n'héritent pas automatiquement des balises des fonctions. Vous pouvez utiliser le AWS Lambda API pour afficher et mettre à jour les balises. Vous pouvez également afficher et mettre à jour les balises tout en gérant un mappage de source d'événement spécifique dans la console Lambda.

## Autorisations requises pour l’utilisation des balises

Pour autoriser une AWS Identity and Access Management (IAM) identité (utilisateur, groupe ou rôle) à lire ou à définir des balises sur une ressource, accordez-lui les autorisations correspondantes :

- lambda : ListTags —Lorsqu' une ressource possède des balises, accordez cette autorisation à tous ceux qui ont besoin de `ListTags` l'utiliser. Pour les fonctions balisées, cette autorisation est également nécessaire pour`GetFunction`.
- lambda : TagResource —Accordez cette autorisation à toute personne ayant besoin d'appeler `TagResource` ou d'exécuter un tag lors de la création.

Pour de plus amples informations, veuillez consulter IAMPolitiques basées sur l'identité pour Lambda.

## Utilisation des balises avec la console Lambda

Vous pouvez utiliser la console Lambda pour créer des mappages de sources d'événements comportant des balises, ajouter des balises aux mappages de sources d'événements existants et filtrer les mappages de sources d'événements par balise.

Lorsque vous ajoutez un déclencheur pour les services basés sur les flux et les files d'attente pris en charge à l'aide de la console Lambda, Lambda crée automatiquement un mappage de source d'événements. Pour plus d'informations sur ces sources d'événements, consultezthe section called “Mappages de source d'événement”. Pour créer un mappage des sources d'événements dans la console, vous devez remplir les conditions préalables suivantes :

- Une fonction .
- Une source d'événement provenant d'un service concerné.

Vous pouvez ajouter les balises dans le cadre de l'interface utilisateur que vous utilisez pour créer ou mettre à jour des déclencheurs.

Pour ajouter une balise lors de la création d'un mappage de sources d'événements

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction .
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Sous Configuration du déclencheur, dans la liste déroulante, choisissez le nom du service dont provient la source de votre événement.
5. Fournissez la configuration de base de votre source d'événements. Pour plus d'informations sur la configuration de votre source d'événements, consultez la section relative au service correspondant dansIntégration d’autres services.
6. Sous Configuration du mappage des sources d'événements, sélectionnez Paramètres supplémentaires.
7. Sous Balises, choisissez Ajouter une nouvelle étiquette
8. Dans le champ Clé, entrez la clé de votre tag. Pour plus d'informations sur les restrictions relatives au balisage, consultez la section Limites et exigences relatives au nommage des balises dans le Guide des  AWS ressources de balisage et de l'éditeur de balises.
9. Choisissez Ajouter.

Pour ajouter des balises à un mappage de source d'événement existant

1. Ouvrez les mappages de sources d'événements dans la console Lambda.
2. Dans la liste des ressources, choisissez le UUIDmappage de source d'événement correspondant à votre fonction et à votre source d'événement ARN.
3. Dans la liste d'onglets située sous le volet de configuration générale, choisissez Tags.
4. Choisissez Gérer les balises.
5. Choisissez Add new tag (Ajouter une nouvelle balise).

6. Dans le champ Clé, entrez la clé de votre tag. Pour plus d'informations sur les restrictions relatives au balisage, consultez la section Limites et exigences relatives au nommage des balises dans le Guide des  AWS ressources de balisage et de l'éditeur de balises.
7. Choisissez Save (Enregistrer).

Pour filtrer les mappages de sources d'événements par balise

1. Ouvrez les mappages de sources d'événements dans la console Lambda.
2. Choisissez le champ de recherche.
3. Dans la liste déroulante, sélectionnez votre clé de tag sous le sous-titre Tags.
4. Sélectionnez Utiliser : « tag-name » pour voir tous les mappages de sources d'événements marqués avec cette clé, ou choisissez un opérateur pour filtrer davantage par valeur.
5. Sélectionnez la valeur de votre balise pour filtrer par une combinaison de clé et de valeur de balise.

Le champ de recherche permet également de rechercher des clés de balise. Entrez le nom d'une clé pour la retrouver dans la liste.

## Utilisation de balises avec AWS CLI

Vous pouvez ajouter et supprimer des balises sur les ressources Lambda existantes, y compris les mappages de sources d'événements, avec le Lambda. API Vous pouvez également ajouter des balises lors de la création d'un mappage de sources d'événements, ce qui vous permet de conserver une ressource étiquetée tout au long de son cycle de vie.

Mise à jour des balises avec la balise Lambda APIs

Vous pouvez ajouter et supprimer des balises pour les ressources Lambda prises en charge par le biais des opérations TagResourceet UntagResourceAPI.

Vous pouvez appeler ces opérations à l'aide du AWS CLI. Pour ajouter des balises à une ressource existante, utilisez la `tag-resource` commande. Cet exemple ajoute deux balises, dont une avec la clé *`Department`* et un avec la clé *`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pour supprimer des balises, utilisez la commande `untag-resource`. Cet exemple supprime le tag contenant la clé *`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier \
--tag-keys Department
```

Ajout de balises lors de la création d'un mappage de sources d'événements

Pour créer un nouveau mappage de source d'événements Lambda avec des balises, utilisez l'CreateEventSourceMappingAPIopération. Spécifiez le paramètre `Tags`. Vous pouvez appeler cette opération à l'aide de la `create-event-source-mapping` AWS CLI commande et de l'`--tags`option. Pour plus d'informations sur la CLI commande, reportez-vous create-event-source-mappingà la référence des AWS CLI commandes.

Avant d'utiliser le `Tags` paramètre with`CreateEventSourceMapping`, assurez-vous que votre rôle est autorisé à étiqueter les ressources en plus des autorisations habituelles nécessaires pour cette opération. Pour plus d'informations sur les autorisations relatives au balisage, consultezthe section called “Autorisations requises pour l’utilisation des balises”.

Afficher les tags avec le tag Lambda APIs

Pour afficher les balises appliquées à une ressource Lambda spécifique, utilisez l'`ListTags`APIopération. Pour plus d'informations, consultez ListTags.

Vous pouvez appeler cette opération à l'aide de la `list-tags` AWS CLI commande en fournissant un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Filtrer les ressources par tag

Vous pouvez utiliser cette AWS Resource Groups Tagging API GetResourcesAPIopération pour filtrer vos ressources par balises. L’opération `GetResources` reçoit jusqu’à 10 filtres, chaque filtre contenant une clé de balise et jusqu’à 10 valeurs de balise. Vous fournissez `GetResources`avec un `ResourceType` pour filtrer par certains types de ressources.

Vous pouvez appeler cette opération à l'aide de la `get-resources` AWS CLI commande. Pour des exemples d'utilisation`get-resources`, voir get-resources dans la référence des AWS CLIcommandes.

# Contrôlez les événements que Lambda envoie à votre fonction

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Par exemple, vous pouvez ajouter un filtre afin que votre fonction traite uniquement les SQS messages Amazon contenant certains paramètres de données. Le filtrage des événements ne fonctionne qu'avec certains mappages de sources d'événements. Vous pouvez ajouter des filtres aux mappages de sources d'événements pour les AWS services suivants :

- Amazon DynamoDB
- Amazon Kinesis Data Streams
- Amazon MQ
- Amazon Managed Streaming pour Apache Kafka (Amazon) MSK
- Self-managed Apache Kafka
- Service de file d'attente Amazon Simple (AmazonSQS)

Pour obtenir des informations spécifiques sur le filtrage avec des sources d'événements spécifiques, consultezthe section called "Utilisation de filtres avec différents Services AWS". Lambda ne prend pas en charge le filtrage d'événements pour Amazon DocumentDB.

Par défaut, vous pouvez définir jusqu'à cinq filtres différents pour un seul mappage des sources d'événements. Vos filtres sont logiquement ORed combinés. Si un enregistrement de votre source d'événement satisfait un ou plusieurs de vos filtres, Lambda inclut l'enregistrement dans le prochain événement qu'il envoie à votre fonction. Si aucun de vos filtres n'est satisfait, Lambda rejette l'enregistrement.

> **Note**
>
> Si vous devez définir plus de cinq filtres pour une source d'événement, vous pouvez demander une augmentation du quota pour un maximum de 10 filtres pour chaque source d'événement. Si vous essayez d'ajouter plus de filtres que ne le permet votre quota actuel, Lambda renvoie un message d'erreur lorsque vous essayez de créer la source d'événement.

Rubriques

- Comprendre les principes de base du filtrage des événements

- Traitement des enregistrements ne répondant pas aux critères de filtrage
- Syntaxe des règles de filtrage
- Attacher des critères de filtre à un mappage de sources d'événements (console)
- Attacher des critères de filtre à un mappage de sources d'événements (AWS CLI)
- Attacher des critères de filtre à un mappage de sources d'événements (AWS SAM)
- Chiffrement des critères de filtre
- Utilisation de filtres avec différents Services AWS

## Comprendre les principes de base du filtrage des événements

Un objet (`FilterCriteria`) de critères de filtre est une structure composée d'une liste de filtres (`Filters`). Chaque filtre est une structure qui définit un modèle de filtrage d'événements (`Pattern`). Un modèle est une représentation sous forme de chaîne d'une règle de JSON filtre. La structure d'un objet `FilterCriteria` est la suivante.

```
{
   "Filters": [
        {
            "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\":
 [ rule2 ] }}"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "Metadata1": [ rule1 ],
    "data": {
        "Data1": [ rule2 ]
    }
}
```

Votre modèle de filtrage peut inclure des propriétés de métadonnées, des propriétés de données ou les deux. Les paramètres de métadonnées disponibles et le format des paramètres de données varient en fonction de l'objet Service AWS qui sert de source d'événements. Supposons, par exemple, que le mappage de la source de votre événement reçoive l'enregistrement suivant d'une SQS file d'attente Amazon :

```
{
    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
    "body": "{\n "City": "Seattle",\n "State": "WA",\n "Temperature": "46"\n}",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
}
```

- Les propriétés de métadonnées sont les champs contenant des informations sur l'événement qui a créé l'enregistrement. Dans l'exemple d'SQSenregistrement Amazon, les propriétés des métadonnées incluent des champs tels que `messageIDeventSourceArn`, et`awsRegion`.
- Les propriétés de données sont les champs de l'enregistrement contenant les données de votre flux ou file d'attente. Dans l'exemple d'SQSévénement Amazon, la clé du champ de données est`body`, et les propriétés des données sont les champs `CityState`, et`Temperature`.

Les différents types de sources d'événements utilisent des valeurs clés différentes pour leurs champs de données. Pour filtrer sur les propriétés de données, assurez-vous d'utiliser la bonne clé dans le modèle de votre filtre. Pour obtenir la liste des clés de filtrage des données et des exemples de modèles de filtre pris en charge pour chacune d'entre elles Service AWS, reportez-vous àUtilisation de filtres avec différents Services AWS.

Le filtrage des événements peut gérer le JSON filtrage à plusieurs niveaux. Par exemple, considérez le fragment suivant d'un enregistrement provenant d'un flux DynamoDB :

```
"dynamodb": {
    "Keys": {
        "ID": {
            "S": "ABCD"
        }
        "Number": {
            "N": "1234"
```

```
    },
    ...
}
```

Supposons que vous vouliez traiter uniquement les enregistrements dont la valeur de la clé de tri `Number` est 4 567. Dans ce cas, votre objet `FilterCriteria` se présente comme suit :

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\":
 [ "4567" ] } } } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "dynamodb": {
        "Keys": {
            "Number": {
                "N": [ "4567" ]
                }
            }
        }
}
```

## Traitement des enregistrements ne répondant pas aux critères de filtrage

La façon dont Lambda gère les enregistrements qui ne répondent pas à vos critères de filtre dépend de la source de l'événement.

- Pour Amazon SQS, si un message ne répond pas à vos critères de filtrage, Lambda le supprime automatiquement de la file d'attente. Il n'est pas nécessaire de supprimer manuellement ces messages sur AmazonSQS.
- Pour Kinesis et DynamoDB, une fois que vos critères de filtre ont évalué un enregistrement, l'itérateur de flux passe au-delà de cet enregistrement. Si l’enregistrement ne répond pas à vos critères de filtre, vous n’avez pas besoin de supprimer manuellement l’enregistrement de la source de votre événement. Après la période de conservation, Kinesis et DynamoDB suppriment

automatiquement ces anciens enregistrements. Si vous souhaitez que les enregistrements soient supprimés plus tôt, consultez Modification de la période de conservation des données.

- Pour les messages Amazon MSK, Apache Kafka autogérés et Amazon MQ, Lambda supprime les messages qui ne correspondent pas à tous les champs inclus dans le filtre. Pour Amazon MSK et Apache Kafka autogéré, Lambda valide les décalages pour les messages correspondants et non correspondants après avoir invoqué la fonction avec succès. Pour Amazon MQ, Lambda accuse réception des messages correspondants après avoir invoqué la fonction avec succès, et reconnaît les messages non correspondants lors de leur filtrage.

## Syntaxe des règles de filtrage

Pour les règles de filtrage, Lambda prend en charge EventBridge les règles Amazon et utilise la même syntaxe que. EventBridge Pour plus d'informations, consultez les modèles EventBridge d'événements Amazon dans le guide de EventBridge l'utilisateur Amazon.

Voici un récapitulatif de tous les opérateurs de comparaison disponibles pour le filtrage d’événement Lambda.

| Opérateur de comparaison | Exemple | Syntaxe des règles |
|---|---|---|
| Null | UserID est null | "UserID": [ null ] |
| Vide | LastName est vide | "LastName": [""] |
| Égal à | Le nom est « Alice » | "Name": [ "Alice" ] |
| Est égal à (ignorer les majuscules) | Le nom est « Alice » | « Nom » : [{"equals-ignore-case« : « alice »}] |
| And | Le lieu est « New York » et le jour est « Monday » | "Location": [ "New York" ], "Day": ["Monday"] |
| Ou | PaymentType est « Crédit » ou « Débit » | PaymentType« : [« Crédit », « Débit"] |
| Ou (plusieurs champs) | Location est « New York », ou Day est « Monday ». | "$or": [ { "Location": [ "New York" ] }, { "Day": [ "Monday" ] } ] |

| Opérateur de comparaison | Exemple | Syntaxe des règles |
|---|---|---|
| Pas | La météo est tout sauf « Raining » | "Weather": [ { "anything-but": [ "Raining" ] } ] |
| Numérique (égal) | Le prix est de 100 | "Price": [ { "numeric": [ "=", 100 ] } ] |
| Numérique (plage) | Le prix est supérieur à 10 et inférieur ou égal à 20 | "Price": [ { "numeric": [ ">", 10, "<=", 20 ] } ] |
| Existe | ProductName existe | ProductName« : [{« existe » : vrai}] |
| N'existe pas | ProductName n'existe pas | ProductName« : [{« existe » : faux}] |
| Commence par | La région se trouve aux États-Unis | "Region": [ {"prefix": "us-" } ] |
| Se termine par | FileName se termine par une extension .png. | FileName« : [{« suffixe » : « .png »}] |

> **Note**
>
> Comme pour les chaînes EventBridge, Lambda utilise une character-by-character correspondance exacte sans pliage en majuscules ni aucune autre normalisation des chaînes. Pour les nombres, Lambda utilise également une représentation par chaîne. Par exemple, 300, 300,0 et 3.0e2 ne sont pas considérés égaux.

Notez que l'opérateur Exists ne fonctionne que sur les nœuds foliaires de votre source d'événementsJSON. Il ne correspond pas aux nœuds intermédiaires. Par exemple, avec ce qui suitJSON, le modèle de filtre `{ "person": { "address": [ { "exists": true } ] } }` ne trouverait aucune correspondance car il `"address"` s'agit d'un nœud intermédiaire.

```
{
  "person": {
```

```
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
  }
}
```

## Attacher des critères de filtre à un mappage de sources d'événements (console)

Suivez ces étapes pour créer un nouveau mappage de source d'événement avec des critères de filtre à l'aide de la console Lambda.

Pour créer un nouveau mappage de sources d'événements avec des critères de filtre (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction pour laquelle créer un mappage de source d'événements.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Pour Trigger configuration (Configuration du déclencheur), choisissez un type de déclencheur qui prend en charge le filtrage des événements. Pour obtenir la liste des services pris en charge, reportez-vous à la liste figurant au début de cette page.
5. Développer Additional settings (Paramètres supplémentaires).
6. Sous Filter criteria (Critères de filtre), choisissez Add (Ajouter) puis définissez et saisissez vos filtres. Par exemple, vous pouvez saisir ce qui suit.

   ```
   { "Metadata" : [ 1, 2 ] }
   ```

   Cela indique à Lambda de traiter uniquement les enregistrements où le champ `Metadata` est égal à 1 ou 2. Vous pouvez continuer à sélectionner Ajouter pour ajouter d'autres filtres jusqu'à la quantité maximale autorisée.
7. Lorsque vous avez terminé d'ajouter vos filtres, sélectionnez Enregistrer.

Lorsque vous saisissez des critères de filtrage à l'aide de la console, vous n'entrez que le modèle de filtrage et n'avez pas besoin d'indiquer la touche `Pattern` ni d'échapper aux guillemets. À l'étape 6 des instructions précédentes, `{ "Metadata" : [ 1, 2 ] }` correspond à ce qui suit `FilterCriteria`.

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Après avoir créé le mappage de la source d'événements dans la console, vous pouvez voir les `FilterCriteria` formatés dans les détails du déclencheur. Pour plus d'exemples de création de filtres d'événements à l'aide de la console, consultez Utilisation de filtres avec différents Services AWS.

## Attacher des critères de filtre à un mappage de sources d'événements (AWS CLI)

Supposons que vous souhaitiez que le mappage d'une source d'événements comporte les suivants `FilterCriteria`:

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Cette create-event-source-mappingcommande crée un nouveau mappage de source d'SQSévénements Amazon pour la fonction `my-function` avec le paramètre spécifié`FilterCriteria`.

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Notez que pour mettre à jour le mappage d'une source d'événement, vous en avez besoinUUID. Vous pouvez les obtenir grâce UUID à un list-event-source-mappingsappel. Lambda renvoie également le UUID dans la create-event-source-mappingCLIréponse.

Pour supprimer les critères de filtre d'une source d'événement, vous pouvez exécuter la update-event-source-mappingcommande suivante avec un `FilterCriteria` objet vide.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria "{}"
```

Pour plus d'exemples de création de filtres d'événements à l'aide du AWS CLI, voirUtilisation de filtres avec différents Services AWS.

## Attacher des critères de filtre à un mappage de sources d'événements (AWS SAM)

Supposons que vous souhaitiez configurer une source d'événements AWS SAM pour utiliser les critères de filtre suivants :

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Pour ajouter ces critères de filtre au mappage de votre source d'événement, insérez l'extrait suivant dans le YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{"Metadata": [1, 2]}'
```

Pour plus d'informations sur la création et la configuration d'un AWS SAM modèle pour un mappage de sources d'événements, consultez la EventSourcesection du guide du AWS SAM développeur. Pour plus d'exemples de création de filtres d'événements à l'aide AWS SAM de modèles, consultezUtilisation de filtres avec différents Services AWS.

## Chiffrement des critères de filtre

Par défaut, Lambda ne chiffre pas votre objet de critères de filtre. Dans les cas d'utilisation où vous pouvez inclure des informations sensibles dans votre objet de critères de filtre, vous pouvez utiliser votre propre KMSclé pour les chiffrer.

Une fois que vous avez chiffré votre objet de critères de filtre, vous pouvez consulter sa version en texte brut à l'aide d'un GetEventSourceMappingAPIappel. Vous devez disposer `kms:Decrypt` des autorisations nécessaires pour afficher correctement les critères de filtre en texte brut.

> **Note**
>
> Si votre objet de critères de filtre est chiffré, Lambda expédie la valeur du `FilterCriteria` champ dans la réponse aux appels. ListEventSourceMappings Ce champ s'affiche plutôt sous la forme`null`. Pour connaître la vraie valeur de`FilterCriteria`, utilisez le GetEventSourceMappingAPI.
> Pour afficher la valeur déchiffrée de `FilterCriteria` dans la console, assurez-vous que votre IAM rôle contient des autorisations pour. GetEventSourceMapping

Vous pouvez spécifier votre propre KMS clé via la console,API/CLI, ou AWS CloudFormation.

Pour chiffrer les critères de filtre à l'aide d'une KMS clé appartenant au client (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez Add trigger (Ajouter déclencheur). Si vous possédez déjà un déclencheur, cliquez sur l'onglet Configuration, puis sur Déclencheurs. Sélectionnez le déclencheur existant, puis choisissez Modifier.
3. Cochez la case à côté de Chiffrer avec une clé gérée KMS par le client.
4. Pour Choisir une clé de KMS chiffrement gérée par le client, sélectionnez une clé activée existante ou créez-en une nouvelle. En fonction de l'opération, vous avez besoin de certaines ou de toutes les autorisations suivantes : `kms:DescribeKey``kms:GenerateDataKey`, et`kms:Decrypt`. Utilisez la politique KMS clé pour accorder ces autorisations.

Si vous utilisez votre propre KMS clé, les API opérations suivantes doivent être autorisées dans la politique des clés :

- `kms:Decrypt`— Doit être accordé au directeur du service Lambda régional ()`lambda.AWS_region.amazonaws.com`. Cela permet à Lambda de déchiffrer les données avec cette clé. KMS
  - Pour éviter un problème de confusion entre les services adjoints, la politique clé utilise la clé de condition `aws:SourceArn`globale. La valeur correcte de la `aws:SourceArn` clé est celle ARN de votre ressource de mappage des sources d'événements. Vous ne pouvez donc l'ajouter à votre politique qu'une fois que vous en avez pris connaissanceARN. Lambda transmet également les `aws:lambda:EventSourceArn` clés `aws:lambda:FunctionArn` and et leurs valeurs respectives dans le contexte de chiffrement lorsqu'il envoie une demande de déchiffrement à. KMS Ces valeurs doivent correspondre aux conditions spécifiées dans la politique clé pour que la demande de déchiffrement aboutisse. Vous n'avez pas besoin d'inclure EventSourceArn les sources d'événements Kafka autogérées car elles n'ont pas de. EventSourceArn
- `kms:Decrypt`— Doit également être accordé au principal qui a l'intention d'utiliser la clé pour afficher les critères de filtrage en texte brut dans les DeleteEventSourceMappingAPIappels GetEventSourceMappingou les appels.
- `kms:DescribeKey`— Fournit les informations clés gérées par le client pour permettre au principal spécifié d'utiliser la clé.
- `kms:GenerateDataKey`— Permet à Lambda de générer une clé de données pour chiffrer les critères de filtre, au nom du principal spécifié (chiffrement de l'enveloppe).

Vous pouvez l'utiliser AWS CloudTrail pour suivre les AWS KMS demandes que Lambda fait en votre nom. Pour des exemples CloudTrail d'événements, voir???.

Nous recommandons également d'utiliser la clé de kms:ViaServicecondition pour limiter l'utilisation de la KMS clé aux demandes provenant de Lambda uniquement. La valeur de cette clé est le principal de service Lambda régional ()`lambda.`*`AWS_region`*`.amazonaws.com`. Voici un exemple de politique clé qui accorde toutes les autorisations pertinentes :

Example AWS KMS politique clé

```
{
    "Version": "2012-10-17",
    "Id": "example-key-policy-1",
    "Statement": [
        {
            "Sid": "Allow Lambda to decrypt using the key",
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.us-east-1.amazonaws.com"
            },
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals" : {
                    "aws:SourceArn": [
                        "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:<esm_uuid>"
                    ]
                },
                "StringEquals": {
                    "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:test-function",
                    "kms:EncryptionContext:aws:lambda:EventSourceArn": "arn:aws:sqs:us-
east-1:123456789012:test-queue"
                }
            }
        },
        {
            "Sid": "Allow actions by an AWS account on the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "kms:*",
```

```
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key to specific roles",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/ExampleRole"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals" : {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        }
    ]
}
```

Pour utiliser votre propre KMS clé afin de chiffrer les critères de filtre, vous pouvez également utiliser la CreateEventSourceMapping AWS CLI commande suivante. Spécifiez la KMS clé ARN avec le `--kms-key-arn` drapeau.

```
aws lambda create-event-source-mapping --function-name my-function \
    --maximum-batching-window-in-seconds 60 \
    --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
    --filter-criteria "{\"filters\": [{\"pattern\": \"{\"a\": [\"1\", \"2\"]}\" }]}" \
    --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Si vous disposez déjà d'un mappage de source d'événements, utilisez plutôt la UpdateEventSourceMapping AWS CLI commande. Spécifiez la KMS clé ARN avec le `--kms-key-arn` drapeau.

```
aws lambda update-event-source-mapping --function-name my-function \
    --maximum-batching-window-in-seconds 60 \
    --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
    --filter-criteria "{\"filters\": [{\"pattern\": \"{\"a\": [\"1\", \"2\"]}\" }]}" \
```

```
    --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Cette opération remplace toute KMS clé précédemment spécifiée. Si vous spécifiez l'`--kms-key-arn`indicateur avec un argument vide, Lambda cesse d'utiliser votre KMS clé pour chiffrer les critères de filtre. Lambda utilise à nouveau par défaut une clé appartenant à Amazon.

Pour spécifier votre propre KMS clé dans un AWS CloudFormation modèle, utilisez la `KMSKeyArn` propriété du type de `AWS::Lambda::EventSourceMapping` ressource. Par exemple, vous pouvez insérer l'extrait suivant dans le YAML modèle de votre source d'événement.

```
MyEventSourceMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    ...
    FilterCriteria:
      Filters:
        - Pattern: '{"a": [1, 2]}'
    KMSKeyArn: "arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599"
    ...
```

Pour pouvoir afficher vos critères de filtre chiffrés en texte clair dans un DeleteEventSourceMappingAPIappel GetEventSourceMappingou, vous devez disposer d'`kms:Decrypt`autorisations.

À compter du 6 août 2024, le `FilterCriteria` champ n'apparaît plus dans AWS CloudTrail les journaux de CreateEventSourceMappinget dans UpdateEventSourceMappingles DeleteEventSourceMappingAPIappels si votre fonction n'utilise pas le filtrage des événements. Si votre fonction utilise le filtrage des événements, le `FilterCriteria` champ apparaît vide (`{}`). Vous pouvez toujours afficher vos critères de filtre en texte clair en réponse aux GetEventSourceMappingAPIappels si vous êtes autorisé à `kms:Decrypt` utiliser la bonne KMS clé.

Exemple d'entrée de CloudTrail journal pour les appels de création/mise à jour/d'appel DeleteEventSourceMapping

Dans l' AWS CloudTrail exemple d'entrée de journal suivant pour un CreateEventSourceMapping appel, `FilterCriteria` apparaît comme vide (`{}`) car la fonction utilise le filtrage des événements. C'est le cas même si `FilterCriteria` l'objet contient des critères de filtre valides

que votre fonction utilise activement. Si la fonction n'utilise pas le filtrage des événements, le `FilterCriteria` champ CloudTrail n'apparaîtra pas du tout dans les entrées du journal.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:userid1",
        "arn": "arn:aws:sts::123456789012:assumed-role/Example/example-role",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA987654321EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/User1",
                "accountId": "123456789012",
                "userName": "User1"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-05-09T20:35:01Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2024-05-09T21:05:41Z",
    "eventSource": "lambda.amazonaws.com",
    "eventName": "CreateEventSourceMapping20150331",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
        "functionName": "example-function",
        "enabled": true,
        "batchSize": 10,
        "filterCriteria": {},
        "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "scalingConfig": {},
        "maximumBatchingWindowInSeconds": 0,
```

```
        "sourceAccessConfigurations": []
    },
    "responseElements": {
        "uUID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "batchSize": 10,
        "maximumBatchingWindowInSeconds": 0,
        "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
        "filterCriteria": {},
        "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:example-
function",
        "lastModified": "May 9, 2024, 9:05:41 PM",
        "state": "Creating",
        "stateTransitionReason": "USER_INITIATED",
        "functionResponseTypes": [],
        "eventSourceMappingArn": "arn:aws:lambda:us-east-2:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```


## Utilisation de filtres avec différents Services AWS

Les différents types de sources d'événements utilisent des valeurs clés différentes pour leurs champs de données. Pour filtrer sur les propriétés de données, assurez-vous d'utiliser la bonne clé dans le modèle de votre filtre. Le tableau suivant indique les clés de filtrage pour chacune des clés prises en charge Service AWS.

| Service AWS | Clé de filtrage |
| --- | --- |
| DynamoDB | `dynamodb` |
| Kinesis | `data` |

| Service AWS | Clé de filtrage |
| --- | --- |
| Amazon MQ | `data` |
| Amazon MSK | `value` |
| Self-managed Apache Kafka | `value` |
| Amazon SQS | `body` |

Les sections suivantes présentent des exemples de modèles de filtres pour différents types de sources d'événements. Elles fournissent également des définitions des formats de données entrantes pris en charge et des formats de corps de modèle de filtre pour chaque service pris en charge.

- the section called “Filtrage des événements”
- the section called “Filtrage des événements”
- the section called “Filtrage des événements”
- the section called “Filtrage des événements”
- the section called “Filtrage des événements”
- the section called “Filtrage des événements”

# Test des fonctions Lambda dans la console

Vous pouvez tester votre fonction Lambda dans la console en invoquant votre fonction avec un événement de test. Un événement de test est une entrée JSON pour votre fonction. Si votre fonction ne nécessite pas d'entrée, l'événement peut être un document vide (`{}`).

Lorsque vous exécutez un test dans la console, Lambda invoque votre fonction de manière synchrone avec l'événement de test. L'exécution de la fonction convertit le JSON de l'événement en un objet et le transmet à la méthode du gestionnaire de votre code afin qu'il soit traité.

> **Créer un événement de test**
>
> Avant de pouvoir tester dans la console, vous devez créer un événement de test privé ou partageable.

## Invocation de fonctions avec des événements de test

Pour tester une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction que vous voulez tester.
3. Choisissez l'onglet Test.
4. Sous Test event (Événement de test), sélectionnez Create new event (Créer un nouvel événement) ou Edit saved event (Modifier un événement sauvegardé), puis sélectionnez l'événement sauvegardé que vous voulez utiliser.
5. En option, choisissez un modèle pour le JSON de l'événement.
6. Sélectionnez Tester).
7. Pour examiner les résultats du test, sous Execution result (Résultat de l'exécution), développez Details (Détails).

Pour invoquer votre fonction sans enregistrer votre événement de test, choisissez Tester avant d'enregistrer. Cela crée un événement de test non sauvegardé que Lambda ne conserve que pour la durée de la session.

Vous pouvez également accéder à vos événements de test enregistrés et non enregistrés dans l'onglet Code. De là, choisissez Test (Tester), puis choisissez votre événement de test.

## Création d'événements de test privés

Les événements de test privés sont disponibles uniquement pour le créateur de l'événement et ils ne nécessitent aucune autorisation supplémentaire pour être utilisés. Vous pouvez créer et enregistrer jusqu'à 10 événements de test privés par fonction.

Pour créer un événement de test privé

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction que vous voulez tester.
3. Choisissez l'onglet Test.
4. Sous Test event (Événement de test), procédez comme suit :
    a. Choisissez un Template (Modèle).
    b. Saisissez un Name (Nom) pour le test.
    c. Dans la zone de saisie de texte, entrez l'événement de test JSON.
    d. Sous Event sharing settings (Paramètres de partage de l'événement), choisissez Private (Privé).
5. Sélectionnez Enregistrer les modifications.

Vous pouvez également créer de nouveaux événements de test dans l'onglet Code. De là, choisissez Test, Configure test event (Configurer l'événement de test).

## Création d'événements de test partageables

Les événements de test partageables sont des événements de test que vous pouvez partager avec d'autres utilisateurs dans le même compte AWS. Vous pouvez modifier les événements de test partageables des autres utilisateurs et invoquer votre fonction avec eux.

Lambda enregistre les événements de test partageables sous forme de schémas dans un registre de schémas Amazon EventBridge (CloudWatch Events) nommé. `lambda-testevent-schemas` Comme Lambda utilise ce registre pour stocker et appeler les événements de test partageables que vous créez, nous vous recommandons de ne pas modifier ce registre ou de créer un registre utilisant le nom `lambda-testevent-schemas`.

Pour consulter, partager et modifier des événements de test partageables, vous devez disposer d'autorisations pour toutes les opérations d'API de registre de schéma EventBridge (CloudWatch événements) suivantes :

- schemas.CreateRegistry
- schemas.CreateSchema
- schemas.DeleteSchema
- schemas.DeleteSchemaVersion
- schemas.DescribeRegistry
- schemas.DescribeSchema
- schemas.GetDiscoveredSchema
- schemas.ListSchemaVersions
- schemas.UpdateSchema

Notez que l'enregistrement des modifications apportées à un événement de test partageable remplace cet événement.

Si vous ne pouvez pas créer, modifier ou consulter des événements de test partageables, vérifiez que votre compte dispose des autorisations requises pour ces opérations. Si vous disposez des autorisations requises mais que vous ne pouvez toujours pas accéder aux événements de test partageables, vérifiez s'il existe des politiques basées sur les ressources susceptibles de limiter l'accès au registre EventBridge (des CloudWatch événements).

Pour créer un événement de test partageable

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction que vous voulez tester.
3. Choisissez l'onglet Test.
4. Sous Test event (Événement de test), procédez comme suit :
    a. Choisissez un Template (Modèle).
    b. Saisissez un Name (Nom) pour le test.
    c. Dans la zone de saisie de texte, entrez l'événement de test JSON.
    d. Sous Event sharing settings (Paramètres de partage de l'événement), choisissez Shareable (Partageable).

5. Choisissez Save changes (Enregistrer les modifications).

> **Utilisez des événements de test partageables avec AWS Serverless Application Model.**
>
> Vous pouvez utiliser AWS SAM pour invoquer des événements de test partageables. Pour en savoir plus, consultez `sam remote test-event` le Guide du développeur de AWS Serverless Application Model

## Suppression des schémas d'événements de test partageables

Lorsque vous supprimez des événements de test partageables, Lambda les supprime du registre `lambda-testevent-schemas`. Si vous supprimez le dernier événement de test partageable du registre, Lambda supprime le registre.

Si vous supprimez la fonction, Lambda ne supprime pas les schémas d'événements de test partageables associés. Vous devez nettoyer ces ressources manuellement depuis la console EventBridge (CloudWatch Events).

# États de la fonction Lambda

Lambda inclut un champ d'état dans la configuration de la fonction pour toutes les fonctions, afin d'indiquer quand la fonction est prête à être appelée. `State` fournit des informations sur l'état actuel de la fonction, y compris si vous pouvez appeler la fonction avec succès. Les états de la fonction ne modifient pas le comportement des appels de la fonction ou la façon dont votre fonction exécute le code. Les états de la fonction incluent :

- `Pending` : une fois que Lambda crée la fonction, il définit l'état comme « en attente ». Lorsqu'elle est en attente, Lambda tente de créer ou de configurer des ressources pour la fonction, telles que des ressources VPC ou EFS. Lambda n'appelle pas de fonction pendant que l'état est en attente. Les appels ou autres actions d'API qui agissent sur la fonction échoueront.
- `Active` – La fonction passera à l'état actif lorsque Lambda aura terminé la configuration et l'approvisionnement des ressources. Les fonctions peuvent uniquement être appelées avec succès lorsqu'elles sont actives.
- `Failed` : indique que la configuration ou l'approvisionnement des ressources ont rencontré une erreur.
- `Inactive` : une fonction devient inactive lorsqu'elle a été inactive suffisamment longtemps afin que Lambda puisse récupérer les ressources externes qui ont été configurées pour elle. Lorsque vous essayez d'appeler une fonction qui est inactive, l'appel échoue et Lambda définit la fonction à l'état « en attente » jusqu'à ce que les ressources de la fonction soient recréées. Si Lambda ne parvient pas à recréer les ressources, la fonction revient à l'état inactif. Si votre fonction est bloquée à l'état inactif, reportez-vous à la fonction `StatusCode` et à ses `StatusCodeReason` attributs pour un dépannage plus approfondi. Vous devrez peut-être résoudre les erreurs et redéployer votre fonction pour la rétablir à l'état actif.

Si vous utilisez des flux de travail d'automatisation basés sur un kit SDK ou si vous appelez directement les API de service Lambda, vérifiez l'état d'une fonction avant l'appel pour vous assurer qu'elle est active. Vous pouvez le faire à l'aide de l'action GetFunctionde l'API Lambda ou en configurant un serveur à l'aide du SDK pour AWS Java 2.0.

```
aws lambda get-function --function-name my-function --query 'Configuration.[State,
 LastUpdateStatus]'
```

Vous devriez voir la sortie suivante :

```
[
 "Active",
 "Successful"
]
```

Les opérations suivantes échouent lorsque la création de fonction est en attente :

- Invoke
- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

## États de la fonction lors de la mise à jour

Lambda fournit un contexte supplémentaire pour les fonctions faisant l'objet de mises à jour à l'aide de l'attribut `LastUpdateStatus` qui peut avoir les états suivants :

- `InProgress` – Une mise à jour est en cours sur une fonction existante. Pendant qu'une mise à jour de fonction est en cours, les appels sont dirigés vers l'ancien code et l'ancienne configuration de la fonction.
- `Successful` – La mise à jour est terminée. Lorsque Lambda a terminé la mise à jour, celle-ci demeure définie jusqu'à une nouvelle mise à jour.
- `Failed` – La mise à jour de la fonction a échoué. Lambda abandonne la mise à jour et l'ancien code et l'ancienne configuration de la fonction demeurent disponibles.

Example

Vous trouverez ci-dessous le résultat de `get-function-configuration` sur une fonction faisant l'objet d'une mise à jour.

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs20.x",
    "VpcConfig": {
        "SubnetIds": [
            "subnet-071f712345678e7c8",
```

```
            "subnet-07fd123456788a036",
            "subnet-0804f77612345cacf"
        ],
        "SecurityGroupIds": [
            "sg-085912345678492fb"
        ],
        "VpcId": "vpc-08e1234569e011e83"
    },
    "State": "Active",
    "LastUpdateStatus": "InProgress",
    ...
}
```

FunctionConfiguration dispose de deux autres attributs, `LastUpdateStatusReason` et `LastUpdateStatusReasonCode` destinés à résoudre les problèmes survenant lors d'une mise à jour.

Les opérations suivantes échouent lorsqu'une mise à jour asynchrone est en cours :

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion
- TagResource

# Comprendre le comportement des nouvelles tentatives dans Lambda

Lorsque vous invoquez une fonction directement, vous déterminez la stratégie de gestion des erreurs liées au code de votre fonction. Lambda ne réessaie pas automatiquement ce type d'erreur en votre nom. Pour réessayer, vous pouvez manuellement invoquer à nouveau votre fonction, envoyer l'événement échoué à une file d'attente pour le débogage, ou ignorer l'erreur. Le code de votre fonction peut s'exécuter entièrement, partiellement ou pas du tout. Si vous recommencez, assurez-vous que le code de votre fonction peut gérer le même événement plusieurs fois, sans provoquer des transactions en double ou d'autres effets secondaires.

Lorsque vous appelez une fonction indirectement, vous devez connaître le comportement de la nouvelle tentative de l'appelant et tout service que la demande rencontre sur le chemin. Cela inclut les scénarios suivants.

- Appel asynchrone – Lambda réessaie deux fois les erreurs de fonction. Si la fonction ne dispose pas d'une capacité suffisante pour traiter toutes les demandes entrantes, les événements peuvent attendre des heures dans la file d'attente avant d'être envoyés à la fonction. Vous pouvez configurer une file d'attente de lettres mortes sur la fonction pour capturer les événements qui n'ont pas été traitées avec succès. Pour de plus amples informations, veuillez consulter Invocation d'une fonction Lambda de manière asynchrone.
- Mappages de source d'événement – Les mappages de source d'événement qui lisent à partir de flux réessaient la totalité du lot d'événements. Les erreurs répétées bloquent le traitement de la partition concernée jusqu'à la résolution de l'erreur ou l'expiration des éléments. Pour détecter des partitions bloquées, vous pouvez surveiller la métrique Âge de l'itérateur.

  Pour les mappages de source d'événement qui lisent à partir d'une file d'attente, vous devez déterminer l'intervalle entre les nouvelles tentatives et la destination pour les événements ayant échoué, en configurant le délai de visibilité et la stratégie de redirection dans la file d'attente source. Pour plus d'informations, consultez Comment Lambda traite les enregistrements provenant de sources d'événements basées sur des flux et des files d'attente ainsi que les rubriques spécifiques du service dans Invoquer Lambda avec des événements provenant d'autres services AWS.
- AWS services — AWS les services peuvent appeler votre fonction de manière synchrone ou asynchrone. Dans le cas des appels synchrones, le service décide s'il convient d'effectuer une nouvelle tentative. Par exemple, les opérations par lot Amazon S3 retentent l'opération si la fonction Lambda renvoie un code de réponse `TemporaryFailure`. Les services qui demandent

par proxy à un utilisateur ou à un client en amont peuvent avoir une stratégie de nouvelle tentative ou peuvent transmettre la réponse d'erreur au demandeur. Par exemple, API Gateway transmet toujours la réponse d'erreur au demandeur.

Pour un appel asynchrone, la logique de nouvelle tentative est la même quelle que soit la source d'appel. Par défaut, Lambda réessaie un appel asynchrone qui a échoué jusqu'à deux fois. Pour de plus amples informations, veuillez consulter Comment Lambda gère les erreurs et les nouvelles tentatives avec un appel asynchrone.

- Autres comptes et clients – Lorsque vous accordez l'accès à d'autres comptes, vous pouvez utiliser des stratégies basées sur une ressource afin de restreindre les services ou ressources qu'ils peuvent configurer pour appeler votre fonction. Pour éviter que votre fonction ne soit surchargée, pensez à placer une API couche devant votre fonction avec Amazon API Gateway.

Pour vous aider à gérer les erreurs dans les applications Lambda, Lambda s'intègre à des services tels qu'Amazon et. CloudWatch AWS X-Ray Vous pouvez utiliser une combinaison de journaux, de mesures, d'alarmes et de suivi pour détecter et identifier rapidement les problèmes liés à votre code de fonction ou à d'autres ressources prenant en charge votre application. API Pour de plus amples informations, veuillez consulter Surveillance et dépannage des fonctions Lambda.

# Utilisez la détection des boucles récursives Lambda pour empêcher les boucles infinies

Lorsque vous configurez une fonction Lambda pour qu'elle envoie une sortie au même service ou à la même ressource que celui ou celle qui l'invoque, il est possible de créer une boucle récursive infinie. Par exemple, une fonction Lambda peut écrire un message dans une file d'attente Amazon Simple Queue Service SQS (Amazon), qui invoque ensuite la même fonction. Cette invocation amène la fonction à écrire un autre message dans la file d'attente, qui à son tour invoque à nouveau la fonction.

Les boucles récursives involontaires peuvent entraîner la facturation de frais imprévus à votre charge. Compte AWS Les boucles peuvent également entraîner la mise à l'échelle de Lambda et l'utilisation de toute la simultanéité disponible de votre compte. Pour aider à réduire l'impact des boucles involontaires, Lambda détecte certains types de boucles récursives peu après leur apparition. Par défaut, lorsque Lambda détecte une boucle récursive, il arrête l'appel de votre fonction et vous en informe. Si votre conception utilise intentionnellement des modèles récursifs, vous pouvez modifier la configuration par défaut d'une fonction pour lui permettre d'être invoquée de manière récursive. Pour plus d'informations, consultez the section called “Autoriser l'exécution d'une fonction Lambda dans une boucle récursive”.

Sections


- Comprendre la détection de boucles récursives
- Pris en charge Services AWS et SDKs
- Notifications de boucles récursives
- Répondre aux notifications de détection de boucles récursives
- Autoriser l'exécution d'une fonction Lambda dans une boucle récursive
- Régions prises en charge pour la détection des boucles récursives Lambda


## Comprendre la détection de boucles récursives

La détection de boucles récursives dans Lambda fonctionne en suivant les événements. Lambda est un service de calcul piloté par les événements qui exécute le code de votre fonction lorsque certains événements se produisent. Par exemple, lorsqu'un article est ajouté à une SQS file d'attente Amazon ou à une rubrique Amazon Simple Notification Service (AmazonSNS). Lambda transmet les événements à votre fonction sous forme JSON d'objets contenant des informations sur la

modification de l'état du système. Lorsqu'un événement provoque l'exécution de votre fonction, cela s'appelle une invocation.

Pour détecter les boucles récursives, Lambda utilise des en-têtes de suivi AWS X-Ray. Lorsque les Services AWS qui prennent en charge cette la détection des boucles récursives envoient des événements à Lambda, ces événements sont automatiquement annotés avec des métadonnées. Lorsque votre fonction Lambda écrit l'un de ces événements dans une autre fonction prise en charge à Service AWS l'aide d'une version prise en charge de an AWS SDK, elle met à jour ces métadonnées. Les métadonnées mises à jour comprennent un décompte du nombre de fois où l'événement a invoqué la fonction.

> **Note**
>
> Il n'est pas nécessaire d'activer le suivi actif de X-Ray pour que cette fonctionnalité soit active. La détection des boucles récursives est activée par défaut pour tous les clients AWS . L'utilisation de cette fonctionnalité est gratuite.

Une chaîne de requêtes est une séquence d'invocations Lambda provoquée par le même événement déclencheur. Par exemple, imaginez qu'une SQS file d'attente Amazon appelle votre fonction Lambda. Votre fonction Lambda renvoie ensuite l'événement traité à la même SQS file d'attente Amazon, qui invoque à nouveau votre fonction. Dans cet exemple, chaque invocation de votre fonction appartient à la même chaîne de requêtes.

Si votre fonction est invoquée environ 16 fois dans la même chaîne de requêtes, Lambda arrête automatiquement l'appel de fonction suivant dans cette chaîne de demandes et vous en informe. Si votre fonction est configurée avec plusieurs déclencheurs, les invocations provenant d'autres déclencheurs ne sont pas affectées.

> **Note**
>
> Même lorsque le `maxReceiveCount` paramètre de la politique de redrive de la file d'attente source est supérieur à 16, la protection récursive Lambda n'empêche pas SQS Amazon de réessayer le message après la détection et l'arrêt d'une boucle récursive. Lorsque Lambda détecte une boucle récursive et interrompt les invocations suivantes, il renvoie une `RecursiveInvocationException` au mappage des sources d'événements. Cela augmente la `receiveCount` valeur du message. Lambda continue de réessayer le message et de bloquer les appels de fonctions jusqu'à ce qu'SQSAmazon détermine que

`maxReceiveCount` le délai est dépassé et envoie le message à la file d'attente de lettres mortes configurée.

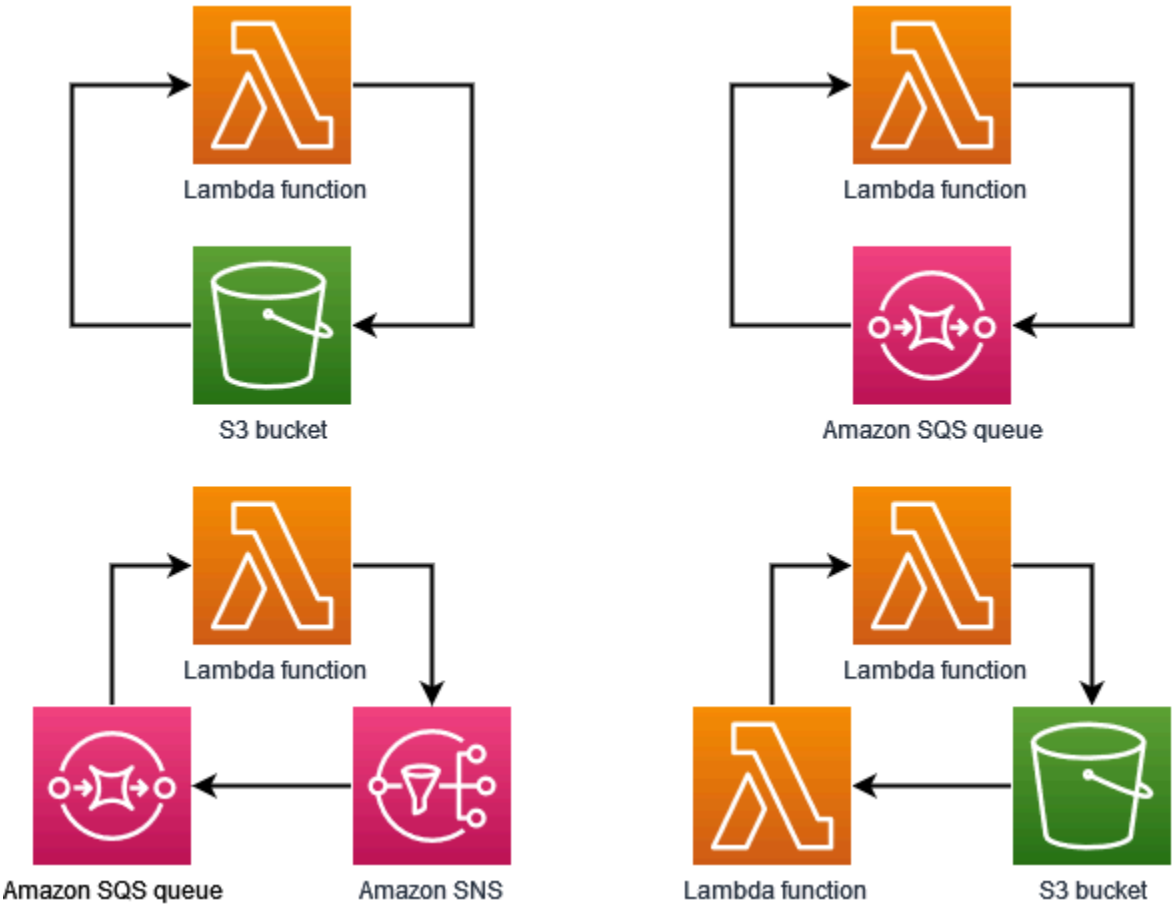
Si vous avez configuré une destination en cas d'échec ou une file d'attente de lettres mortes pour votre fonction, Lambda envoie également l'événement de l'invocation arrêtée vers votre destination ou votre file d'attente de lettres mortes. Lorsque vous configurez une file d'attente de destination ou de lettre morte pour votre fonction, veillez à ne pas utiliser de SNS rubrique Amazon ou de SQS file d'attente Amazon que votre fonction utilise également comme déclencheur d'événement ou comme mappage de source d'événement. Si vous envoyez des événements à la même ressource qui invoque votre fonction, vous pouvez créer une autre boucle récursive.

## Pris en charge Services AWS et SDKs

Lambda ne peut détecter que les boucles récursives dont certaines sont prises en charge. Services AWS Pour que les boucles récursives soient détectées, votre fonction doit également utiliser l'une des boucles prises en charge AWS SDKs.

### Soutenu Services AWS

Lambda détecte actuellement les boucles récursives entre vos fonctions, Amazon, Amazon SQS S3 et Amazon. SNS Lambda détecte également les boucles composées uniquement de fonctions Lambda, qui peuvent s'invoquer de manière synchrone ou asynchrone. Les schémas suivants montrent quelques exemples de boucles que Lambda peut détecter :




Lorsqu'une autre Service AWS , telle qu'Amazon DynamoDB, fait partie de la boucle, Lambda ne peut actuellement pas la détecter ni l'arrêter.

Comme Lambda ne détecte actuellement que les boucles récursives impliquant Amazon, Amazon SQS S3 et AmazonSNS, il est toujours possible que les boucles impliquant d'autres Services AWS boucles entraînent une utilisation involontaire de vos fonctions Lambda.

Pour éviter que des frais imprévus ne vous soient facturés Compte AWS, nous vous recommandons de configurer les CloudWatchalarmes Amazon pour vous avertir des habitudes d'utilisation inhabituelles. Par exemple, vous pouvez configurer pour vous CloudWatch avertir des pics de simultanéité ou d'invocation des fonctions Lambda. Vous pouvez également configurer une alarme de facturation pour vous avertir lorsque les dépenses sur votre compte dépassent un seuil que vous avez spécifié. Vous pouvez également utiliser AWS Cost Anomaly Detectionpour vous alerter en cas de schémas de facturation inhabituels.

## Soutenu AWS SDKs

Pour que Lambda détecte les boucles récursives, votre fonction doit utiliser l'une des SDK versions suivantes ou supérieures :

| Environnement d’exécution | AWS SDKVersion minimale requise |
|---|---|
| Node.js | 2.1147.0 (SDKversion 2)<br>3.105.0 (SDKversion 3) |
| Python | 1.24.46 (boto3)<br>1.27.46 (botocore) |
| Java 8 et Java 11 | 2,17135 |
| Java 17 | 2,20,81 |
| Java 21 | 2,21,24 |
| .NET | 3,7,293,0 |
| Ruby | 3,134,0 |
| PHP | 3,232,0 |
| Go | SDKV2 (utiliser la dernière version) |

Certains environnements d'exécution Lambda tels que Python et Node.js incluent une version du. AWS SDK Si la SDK version incluse dans le runtime de votre fonction est inférieure à la version minimale requise, vous pouvez ajouter une version prise en charge SDK au package de déploiement de votre fonction. Vous pouvez également ajouter une SDK version prise en charge à votre fonction à l'aide d'une couche Lambda. Pour obtenir la liste des composants SDKs inclus dans chaque environnement d'exécution Lambda, consultez. Environnements d’exécution (runtimes) Lambda

# Notifications de boucles récursives

Lorsque Lambda arrête une boucle récursive, vous recevez des notifications le AWS Health Dashboard et par e-mail. Vous pouvez également utiliser CloudWatch des métriques pour surveiller le nombre d'appels récursifs interrompus par Lambda.

## AWS Health Dashboard notifications

Lorsque Lambda arrête un appel récursif, une notification s' AWS Health Dashboard affiche sur la page de santé de votre compte, sous Problèmes ouverts et récents. Notez qu'il peut s'écouler jusqu'à trois heures après l'arrêt d'une invocation récursive par Lambda avant que cette notification ne s'affiche. Pour plus d'informations sur l'affichage des événements du compte dans le AWS Health Dashboard, consultez Getting started with your AWS Health Dashboard — Your account health dans le AWS Health User Guide.

## Alertes par e-mail

Lorsque Lambda arrête pour la première fois une invocation récursive de votre fonction, il vous envoie une alerte par e-mail. Lambda envoie au maximum un e-mail toutes les 24 heures pour chaque fonction de votre Compte AWS. Une fois que Lambda a envoyé une notification par e-mail, vous ne recevrez plus d'e-mails pour cette fonction pendant 24 heures, même si Lambda arrête d'autres invocations récursives de la fonction. Notez qu'il peut s'écouler jusqu'à trois heures après l'arrêt d'une invocation récursive par Lambda avant que vous ne receviez cette alerte par e-mail.

Lambda envoie des alertes par e-mail en boucle récursive au contact principal Compte AWS de votre compte et au contact opérationnel alternatif. Pour plus d'informations sur l'affichage ou la mise à jour des adresses e-mail de votre compte, consultez la section Mise à jour des informations de contact dans la Référence générale AWS .

## CloudWatch Métriques Amazon

La CloudWatch métrique `RecursiveInvocationsDropped` enregistre le nombre d'invocations de fonctions que Lambda a arrêtées parce que votre fonction a été invoquée plus de 16 fois environ au cours d'une même chaîne de requêtes. Lambda émet cette métrique dès qu'il arrête une invocation récursive. Pour afficher cette métrique, suivez les instructions relatives à l'affichage des métriques sur la CloudWatch console et choisissez la métrique`RecursiveInvocationsDropped`.

# Répondre aux notifications de détection de boucles récursives

Lorsque votre fonction est invoquée plus de 16 fois environ par le même événement déclencheur, Lambda arrête l'appel de fonction suivant pour cet événement afin de rompre la boucle récursive. Pour éviter qu'une boucle récursive interrompue par Lambda ne se reproduise, procédez comme suit :

- Réduisez la simultanéité disponible de votre fonction à zéro, ce qui limite toutes les invocations futures.
- Supprimez ou désactivez le déclencheur ou le mappage des sources d'événements qui invoque votre fonction.
- Identifiez et corrigez les défauts de code qui réécrivent les événements dans la AWS ressource qui appelle votre fonction. Une source fréquente de défauts survient lorsque vous utilisez des variables pour définir la source et la cible des événements d'une fonction. Vérifiez que vous n'utilisez pas la même valeur pour les deux variables.

En outre, si la source d'événements de votre fonction Lambda est une SQS file d'attente Amazon, envisagez de configurer une file d'attente de lettres mortes dans la file d'attente source.

> **Note**
>
> Veillez à configurer la file d’attente de lettres mortes sur la file d’attente source, non sur la fonction Lambda. La file d’attente de lettres mortes que vous configurez dans une fonction est utilisée pour la file d’attente d’invocation asynchrone et non pour les files d’attente source d’événement.

Si la source de l'événement est un SNS sujet Amazon, pensez à ajouter une destination en cas d'échec pour votre fonction.

Pour réduire à zéro la simultanéité disponible de votre fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction .
3. Choisissez Limiter.
4. Dans la boîte de dialogue Limiter votre fonction, choisissez Confirmer.

Pour supprimer un déclencheur ou un mappage des sources d'événements pour votre fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction .
3. Choisissez l'onglet Configuration, puis Déclencheurs.
4. Sous Déclencheurs, sélectionnez le déclencheur ou le mappage des sources d'événements que vous souhaitez supprimer, puis choisissez Supprimer.
5. Dans la boîte de dialogue Supprimer des déclencheurs, choisissez Supprimer.

Pour désactiver un mappage des sources d'événements pour votre fonction (AWS CLI)

1. Pour trouver le UUID mappage des sources d'événements que vous souhaitez désactiver, exécutez la list-event-source-mappingscommande AWS Command Line Interface (AWS CLI).

   ```
   aws lambda list-event-source-mappings
   ```

2. Pour désactiver le mappage des sources d'événements, exécutez la AWS CLI update-event-source-mappingcommande suivante.

   ```
   aws lambda update-event-source-mapping --function-name MyFunction \
   --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
   ```

## Autoriser l'exécution d'une fonction Lambda dans une boucle récursive

Si votre conception utilise intentionnellement une boucle récursive, vous pouvez configurer une fonction Lambda pour permettre son appel récursif. Nous vous recommandons d'éviter d'utiliser des boucles récursives dans votre conception. Les erreurs de mise en œuvre peuvent entraîner des appels récursifs utilisant toutes les données disponibles simultanément et des frais imprévus facturés sur votre Compte AWS compte.

> **Important**
>
> Si vous utilisez des boucles récursives, traitez-les avec prudence. Mettez en œuvre des garde-fous conformes aux meilleures pratiques afin de minimiser les risques d'erreurs de mise en œuvre. Pour en savoir plus sur les meilleures pratiques relatives à l'utilisation de

modèles récursifs, consultez la section Modèles récursifs qui provoquent des fonctions Lambda incontrôlables dans Serverless Land.

Vous pouvez configurer des fonctions pour autoriser les boucles récursives à l'aide de la console Lambda, AWS Command Line Interface du AWS CLI() et du. PutFunctionRecursionConfigAPI Vous pouvez également configurer le paramètre de détection de boucle récursive d'une fonction dans AWS SAM et AWS CloudFormation.

Par défaut, Lambda détecte et arrête les boucles récursives. À moins que votre conception n'utilise intentionnellement une boucle récursive, nous vous recommandons de ne pas modifier la configuration par défaut de vos fonctions.

Notez que lorsque vous configurez une fonction pour autoriser les boucles récursives, la CloudWatch métrique `RecursiveInvocationsDropped` n'est pas émise.

Console

Pour autoriser l'exécution d'une fonction dans une boucle récursive (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction pour ouvrir la page détaillée de la fonction.
3. Choisissez l'onglet Configuration, puis sélectionnez Concurrence et détection de récursivité.
4. À côté de Détection de boucle récursive, choisissez Modifier.
5. Sélectionnez Autoriser les boucles récursives.
6. Choisissez Save (Enregistrer).

AWS CLI

Vous pouvez utiliser le PutFunctionRecursionConfigAPIpour permettre à votre fonction d'être invoquée dans une boucle récursive. Spécifiez `Allow` le paramètre de boucle récursive. Par exemple, vous pouvez appeler cela API avec la `put-function-recursion-config` AWS CLI commande :

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Allow
```

Vous pouvez rétablir la configuration par défaut de votre fonction afin que Lambda mette fin aux boucles récursives lorsqu'il les détecte. Modifiez la configuration de votre fonction à l'aide de la console Lambda ou du. AWS CLI

Console

Pour configurer une fonction de manière à ce que les boucles récursives soient interrompues (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction pour ouvrir la page détaillée de la fonction.
3. Choisissez l'onglet Configuration, puis sélectionnez Concurrence et détection de récursivité.
4. À côté de Détection de boucle récursive, choisissez Modifier.
5. Sélectionnez Terminer les boucles récursives.
6. Choisissez Save (Enregistrer).

AWS CLI

Vous pouvez utiliser le PutFunctionRecursionConfigAPIpour configurer votre fonction afin que Lambda mette fin aux boucles récursives lorsqu'il les détecte. Spécifiez `Terminate` le paramètre de boucle récursive. Par exemple, vous pouvez appeler cela API avec la `put-function-recursion-config` AWS CLI commande :

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Terminate
```

## Régions prises en charge pour la détection des boucles récursives Lambda

La détection de boucle récursive Lambda est prise en charge dans ce qui suit. Régions AWS

- USA Est (Virginie du Nord)
- USA Est (Ohio)
- USA Ouest (Californie du Nord)
- USA Ouest (Oregon)
- Afrique (Le Cap)
- Asie-Pacifique (Hong Kong)

- Asia Pacific (Mumbai)
- Asie-Pacifique (Osaka)
- Asia Pacific (Seoul)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Sydney)
- Asie-Pacifique (Tokyo)
- Canada (Centre)
- Europe (Francfort)
- Europe (Irlande)
- Europe (Londres)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Moyen-Orient (Bahreïn)
- Amérique du Sud (Sao Paulo)

# Création et gestion de la fonction Lambda URLs

Une fonction URL est un point de terminaison HTTP (S) dédié à votre fonction Lambda. Vous pouvez créer et configurer une fonction URL par le biais de la console Lambda ou du Lambda. API Lorsque vous créez une fonctionURL, Lambda génère automatiquement un point de URL terminaison unique pour vous. Une fois que vous avez créé une fonctionURL, son URL point de terminaison ne change jamais. Les URL points de terminaison des fonctions ont le format suivant :

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> URLsLes fonctions ne sont pas prises en charge dans les pays suivants Régions AWS : Asie-Pacifique (Hyderabad`ap-south-2`) (), Asie-Pacifique (Melbourne`ap-southeast-4`) (), Asie-Pacifique (Malaisie`ap-southeast-5`) (), Canada Ouest (Calgary) (`ca-west-1`), Europe (Espagne) (`eu-south-2`), Europe (Zurich) (`eu-central-2`), Israël (Tel Aviv) (`il-central-1`) et Moyen-Orient (UAE) (`me-central-1`).

URLsLes fonctions sont compatibles avec le double empilage, prenant en charge et. IPv4 IPv6 Après avoir configuré une fonction URL pour votre fonction, vous pouvez appeler votre fonction via son point de terminaison HTTP (S) via un navigateur Web, curl, Postman ou n'importe quel HTTP client.

> **Note**
>
> Vous ne pouvez accéder à votre fonction que URL par le biais de l'Internet public. Bien que les fonctions Lambda soient compatibles AWS PrivateLink, les fonctions ne le sont URLs pas.

La fonction Lambda URLs utilise des politiques basées sur les ressources pour la sécurité et le contrôle d'accès. La fonction prend URLs également en charge les options de configuration du partage de ressources entre origines (CORS).

Vous pouvez appliquer une fonction URLs à n'importe quel alias de fonction ou à la version de fonction `$LATEST` non publiée. Vous ne pouvez pas ajouter de fonction URL à une autre version de fonction.

La section suivante explique comment créer et gérer une fonction à URL l'aide de la console Lambda et du AWS CLI modèle AWS CloudFormation

Rubriques


- Création d'une fonction URL (console)
- Création d'une fonction URL (AWS CLI)
- Ajouter une fonction URL à un CloudFormation modèle
- Partage de ressources entre origines () CORS
- Fonction d'étranglement URLs
- Fonction de désactivation URLs
- Fonction de suppression URLs
- Contrôler l'accès aux URL des fonctions Lambda
- Invocation de la fonction Lambda URLs
- Surveillance des URL de fonctions Lambda
- Didacticiel : Creating a Lambda function with a function URL (Création d’une fonction Lambda avec une URL de fonction)


## Création d'une fonction URL (console)

Procédez comme suit pour créer une fonction à URL l'aide de la console.

Pour créer une fonction URL pour une fonction existante (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction URL pour laquelle vous souhaitez créer la fonction.
3. Choisissez l'onglet Configuration, puis choisissez Function URL.
4. Sélectionnez Create functionURL (Créer une fonction).
5. Pour le type d'authentification, choisissez AWS_IAMou NONE. Pour plus d'informations sur URL l'authentification fonctionnelle, consultezContrôle d’accès.
6. (Facultatif) Sélectionnez Configurer le partage de ressources entre origines (CORS), puis configurez les CORS paramètres de votre fonctionURL. Pour plus d'informations surCORS, voirPartage de ressources entre origines () CORS.
7. Choisissez Save (Enregistrer).

Cela crée une fonction URL pour la version `$LATEST` non publiée de votre fonction. La fonction URL apparaît dans la section Vue d'ensemble des fonctions de la console.

Pour créer une fonction URL pour un alias existant (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction avec l'alias URL pour lequel vous souhaitez créer la fonction.
3. Cliquez sur l'onglet Alias, puis choisissez le nom de l'alias URL pour lequel vous souhaitez créer la fonction.
4. Choisissez l'onglet Configuration, puis choisissez Function URL.
5. Sélectionnez Create functionURL (Créer une fonction).
6. Pour le type d'authentification, choisissez AWS_IAMou NONE. Pour plus d'informations sur URL l'authentification fonctionnelle, consultezContrôle d’accès.
7. (Facultatif) Sélectionnez Configurer le partage de ressources entre origines (CORS), puis configurez les CORS paramètres de votre fonctionURL. Pour plus d'informations surCORS, voirPartage de ressources entre origines () CORS.
8. Choisissez Save (Enregistrer).

Cela crée une fonction URL pour votre alias de fonction. La fonction URL apparaît dans la section Vue d'ensemble des fonctions de la console pour votre alias.

Pour créer une nouvelle fonction avec une fonction URL (console)

Pour créer une nouvelle fonction avec une fonction URL (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez Create function (Créer une fonction).
3. Sous Basic information (Informations de base), procédez comme suit :

    a. Dans Function name (nom de la fonction), saisissez un nom pour votre fonction, par exemple **`my-function`**.

    b. Pour Runtime (Exécution), choisissez l’environnement d’exécution du langage que vous préférez, comme Node.js 18.x.

    c. Pour Architecture, choisissez x86_64 ou arm64.

    d. Développez Permissions (Autorisations), puis choisissez si vous souhaitez créer un nouveau rôle d’exécution ou utiliser un rôle existant.

4. Développez les paramètres avancés, puis sélectionnez Fonction URL.
5. Pour le type d'authentification, choisissez AWS_IAMou NONE. Pour plus d'informations sur URL l'authentification fonctionnelle, consultezContrôle d’accès.
6. (Facultatif) Sélectionnez Configurer le partage de ressources entre origines (CORS). En sélectionnant cette option lors de la création de la fonction, celle-ci URL accepte par défaut les demandes provenant de toutes les origines. Vous pouvez modifier les CORS paramètres de votre fonction URL après l'avoir créée. Pour plus d'informations surCORS, voirPartage de ressources entre origines () CORS.
7. Sélectionnez Create function (Créer une fonction).

Cela crée une nouvelle fonction avec une fonction URL pour la version `$LATEST` non publiée de la fonction. La fonction URL apparaît dans la section Vue d'ensemble des fonctions de la console.

## Création d'une fonction URL (AWS CLI)

Pour créer une fonction URL pour une fonction Lambda existante à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante :

```
aws lambda create-function-url-config \
    --function-name my-function \
    --qualifier prod \ // optional
    --auth-type AWS_IAM
    --cors-config {AllowOrigins="https://example.com"} // optional
```

Cela ajoute une fonction URL au **`prod`** qualificatif de la fonction**`my-function`**. Pour plus d'informations sur ces paramètres de configuration, reportez-vous CreateFunctionUrlConfigà la API référence.

> **Note**
>
> Pour créer une fonction URL via le AWS CLI, la fonction doit déjà exister.

## Ajouter une fonction URL à un CloudFormation modèle

Pour ajouter une `AWS::Lambda::Url` ressource à votre AWS CloudFormation modèle, utilisez la syntaxe suivante :

### JSON

```
{
  "Type" : "AWS::Lambda::Url",
  "Properties" : {
      "AuthType" : String,
      "Cors" : Cors,
      "Qualifier" : String,
      "TargetFunctionArn" : String
    }
}
```

### YAML

```
Type: AWS::Lambda::Url
Properties:
  AuthType: String
  Cors:
    Cors
  Qualifier: String
  TargetFunctionArn: String
```

## Paramètres

- (Obligatoire) `AuthType` — Définit le type d'authentification pour votre fonctionURL. Les valeurs possibles sont soit `AWS_IAM`, soit `NONE`. Pour limiter l'accès aux utilisateurs authentifiés uniquement, définissez la valeur sur `AWS_IAM`. Pour contourner IAM l'authentification et autoriser n'importe quel utilisateur à envoyer des demandes à votre fonction, définissez sur`NONE`.
- (Facultatif) `Cors` — Définit les CORSparamètres de votre fonctionURL. Pour ajouter `Cors` des éléments à votre `AWS::Lambda::Url` ressource CloudFormation, utilisez la syntaxe suivante.

  Example AWS: :Lambda : :Url.Cors () JSON

  ```
  {
    "AllowCredentials" : Boolean,
    "AllowHeaders" : [ String, ... ],
    "AllowMethods" : [ String, ... ],
    "AllowOrigins" : [ String, ... ],
    "ExposeHeaders" : [ String, ... ],
  ```

```
  "MaxAge" : Integer
}
```

Example AWS: :Lambda : :Url.Cors () YAML

```
  AllowCredentials: Boolean
  AllowHeaders:
    - String
  AllowMethods:
    - String
  AllowOrigins:
    - String
  ExposeHeaders:
    - String
  MaxAge: Integer
```

- (Facultatif) `Qualifier` – Le nom de l'alias.
- (Obligatoire) `TargetFunctionArn` — Le nom ou Amazon Resource Name (ARN) de la fonction Lambda. Les formats de nom valides sont notamment les suivants :
  - Nom de fonction – `my-function`
  - Fonction ARN — `arn:aws:lambda:us-west-2:123456789012:function:my-function`
  - Partiel ARN — `123456789012:function:my-function`

## Partage de ressources entre origines () CORS

Pour définir comment les différentes origines peuvent accéder à votre fonctionURL, utilisez le partage de ressources entre origines (CORS). Nous vous recommandons de configurer CORS si vous avez l'intention d'appeler votre fonction URL depuis un autre domaine. Lambda prend en charge les CORS en-têtes de fonction suivants. URLs

| CORSen-tête | CORSpropriété de configuration | Exemples de valeur |
| --- | --- | --- |
| Access-Control-Allow-Origine | `AllowOrigins` | * (autoriser toutes les origines)<br>`https://www.example.com` |

| CORSen-tête | CORSpropriété de configuration | Exemples de valeur |
| --- | --- | --- |
| | | `http://localhost:60905` |
| Access-Control-Allow-Méthodes | `AllowMethods` | `GET`, `POST`, `DELETE`, `*` |
| Access-Control-Allow-En-têtes | `AllowHeaders` | `Date`, `Keep-Alive` , `X-Custom-Header` |
| Access-Control-Expose-En-têtes | `ExposeHeaders` | `Date`, `Keep-Alive` , `X-Custom-Header` |
| Access-Control-Allow-Informations d'identification | `AllowCredentials` | `TRUE` |
| Access-Control-Max-Âge | `MaxAge` | `5` (par défaut), `300` |

Lorsque vous configurez une fonction CORS à URL l'aide de la console Lambda ou du, AWS CLI Lambda ajoute automatiquement CORS les en-têtes à toutes les réponses via la fonction. URL Vous pouvez également ajouter manuellement des CORS en-têtes à la réponse de votre fonction. En cas de conflit entre les en-têtes, le comportement attendu dépend du type de demande :

- Pour les demandes préalables au vol, telles que les OPTIONS demandes, les CORS en-têtes configurés sur la fonction ont prioritéURL. Lambda renvoie uniquement ces CORS en-têtes dans la réponse.
- Pour les requêtes autres que celles effectuées avant le vol, telles que les POST requêtes GET ou, Lambda renvoie à la fois les CORS en-têtes configurés sur la fonctionURL, ainsi que les CORS en-têtes renvoyés par la fonction. Cela peut entraîner des doublons d'CORSen-têtes dans la réponse. Un message d'erreur similaire au suivant peut s'afficher : `The 'Access-Control-Allow-Origin' header contains multiple values '*, *', but only one is allowed`

En général, nous recommandons de configurer tous les CORS paramètres de la fonctionURL, plutôt que d'envoyer CORS les en-têtes manuellement dans la réponse de la fonction.

## Fonction d'étranglement URLs

La limitation limite le débit auquel la fonction traite les demandes. Cette fonction est utile dans de nombreuses situations, notamment pour empêcher votre fonction de surcharger les ressources en aval ou pour gérer une augmentation soudaine des demandes.

Vous pouvez limiter le taux de demandes traitées par votre fonction Lambda par le biais d'une URL fonction en configurant la simultanéité réservée. La simultanéité réservée limite le nombre maximal d'invocations simultanées pour votre fonction. Le taux de requêtes maximal par seconde (RPS) de votre fonction est équivalent à 10 fois la simultanéité réservée configurée. Par exemple, si vous configurez votre fonction avec une simultanéité réservée de 100, le maximum RPS est de 1 000.

Chaque fois que la simultanéité de votre fonction dépasse la simultanéité réservée, votre fonction URL renvoie un code d'HTTP`429`état. Si votre fonction reçoit une demande qui dépasse le RPS maximum de 10 fois sur la base de la simultanéité réservée que vous avez configurée, vous recevez également une HTTP `429` erreur. Pour de plus amples informations sur la simultanéité réservée, consultez Configuration de la simultanéité réservée pour une fonction.

## Fonction de désactivation URLs

En cas d'urgence, vous souhaiterez peut-être rejeter tout le trafic vers votre fonctionURL. Pour désactiver votre fonctionURL, réglez la simultanéité réservée sur zéro. Cela limite toutes les demandes adressées à votre fonctionURL, ce qui entraîne des réponses de HTTP `429` statut. Pour réactiver votre fonctionURL, supprimez la configuration de simultanéité réservée ou définissez la configuration sur un montant supérieur à zéro.

## Fonction de suppression URLs

Lorsque vous supprimez une fonctionURL, vous ne pouvez pas la récupérer. La création d'une nouvelle fonction URL se traduira par une URL adresse différente.

> **Note**
>
> Si vous supprimez une fonction URL avec un type d'authentification`NONE`, Lambda ne supprime pas automatiquement la politique basée sur les ressources associée. Si vous souhaitez supprimer cette politique, vous devez le faire manuellement.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez le nom de la fonction.
3. Choisissez l'onglet Configuration, puis choisissez Function URL.
4. Sélectionnez Delete (Supprimer).
5. Entrez le mot delete dans le champ pour confirmer la suppression.
6. Sélectionnez Delete (Supprimer).

> **Note**
>
> Lorsque vous supprimez une fonction qui possède une fonctionURL, Lambda supprime la fonction de manière asynchrone. URL Si vous créez immédiatement une nouvelle fonction portant le même nom dans le même compte, il est possible que la fonction d'origine URL soit mappée à la nouvelle fonction au lieu d'être supprimée.

## Contrôler l'accès aux URL des fonctions Lambda

Vous pouvez contrôler l'accès aux URL de votre fonctions Lambda à l'aide du paramètre `AuthType` combiné avec les stratégies basées sur les ressources attachées à votre fonction spécifique. La configuration de ces deux composants détermine qui peut invoquer ou exécuter d'autres actions administratives sur l'URL de votre fonction.

Le paramètre `AuthType` détermine comment Lambda authentifie ou autorise les demandes vers l'URL de votre fonction. Lorsque vous configurez l'URL de votre fonction, vous devez spécifier l'une des options `AuthType` suivantes :

- `AWS_IAM`— Lambda utilise AWS Identity and Access Management (IAM) pour authentifier et autoriser les demandes en fonction de la politique d'identité du principal IAM et de la politique basée sur les ressources de la fonction. Choisissez cette option si vous souhaitez que seuls les utilisateurs et les rôles authentifiés invoquent votre fonction via l'URL de la fonction.

- `NONE` – Lambda n'effectue aucune authentification avant d'invoquer votre fonction. Toutefois, la stratégie basée sur les ressources de votre fonction est toujours en vigueur et doit accorder un accès public avant que l'URL de votre fonction puisse recevoir des demandes. Choisissez cette option pour autoriser un accès public et non authentifié à l'URL de votre fonction.

En plus du paramètre `AuthType`, vous pouvez également utiliser des stratégies basées sur les ressources pour accorder des autorisations à d'autres Comptes AWS pour invoquer votre fonction. Pour plus d'informations, consultez Utilisation de IAM politiques basées sur les ressources dans Lambda.

Pour obtenir des informations supplémentaires sur la sécurité, vous pouvez AWS Identity and Access Management Access Analyzer obtenir une analyse complète de l'accès externe à l'URL de votre fonction. IAM Access Analyzer surveille également les autorisations nouvelles ou mises à jour sur vos fonctions Lambda afin de vous aider à identifier les autorisations accordant un accès public et entre comptes. L'utilisation d'IAM Access Analyzer est gratuite pour tous AWS les clients. Pour commencer à utiliser IAM Access Analyzer, consultez la section Utilisation d' AWS IAM Access Analyzer.

Cette page contient des exemples de politiques basées sur les ressources pour les deux types d'authentification, ainsi que la façon de créer ces politiques à l'aide de l'opération AddPermissionAPI ou de la console Lambda. Pour plus d'informations sur la procédure d'invocation de l'URL de votre fonction après avoir configuré les autorisations, consultez Invocation de la fonction Lambda URLs.

Rubriques

- Utilisation du type d'authentification AWS_IAM
- Utilisation du type d'authentification NONE
- Gouvernance et contrôle d'accès

## Utilisation du type d'authentification **`AWS_IAM`**

Si vous choisissez le type d'authentification `AWS_IAM`, les utilisateurs qui ont besoin d'invoquer l'URL de votre fonction Lambda doivent avoir l'autorisation `lambda:InvokeFunctionUrl`. Selon la personne qui effectue la demande d'invocation, vous devrez peut-être accorder cette autorisation à l'aide d'une stratégie basée sur les ressources.

Si le principal qui fait la demande se trouve dans la même URL Compte AWS que l'URL de la fonction, le principal doit soit disposer d'`lambda:InvokeFunctionUrl`autorisations dans sa politique basée sur l'identité, soit avoir des autorisations qui lui sont accordées dans le cadre de la politique basée sur les ressources de la fonction. En d'autres termes, une stratégie basée sur les ressources est facultative si l'utilisateur possède déjà des autorisations `lambda:InvokeFunctionUrl` dans sa stratégie basée sur l'identité. L'évaluation des stratégies suit les règles décrites dans Identification d'une demande autorisée ou refusée dans un compte.

Si le principal qui fait la demande se trouve dans un autre compte, il doit avoir à la fois une stratégie basée sur l'identité qui lui donne des autorisations `lambda:InvokeFunctionUrl` et des autorisations qui lui sont accordées dans une stratégie basée sur les ressources de la fonction qu'il essaie d'invoquer. Dans ces cas de comptes croisés, l'évaluation de la stratégie suit les règles décrites dans Comment déterminer si une demande d'accès entre comptes est autorisée.

Pour un exemple d'interaction entre comptes, la politique basée sur les ressources suivante permet au `example` rôle in d'invoquer l'URL de la fonction associée Compte AWS `444455556666` à la fonction : `my-function`

Example exemple de stratégie d'invocation entre comptes d'URL de fonction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/example"
            },
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

Vous pouvez créer cette déclaration de stratégie via la console en procédant comme suit :

Comment accorder des autorisations d'invocation d'URL à un autre compte (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction pour laquelle vous souhaitez accorder des autorisations d'invocation d'URL.
3. Choisissez l'onglet Configuration, puis Permissions (Autorisations).
4. Sous Resource-based policy (stratégie basée sur une ressource), choisissez Add permissions (Ajouter des autorisations).

5. Choisissez Function URL (URL de fonction).
6. Pour Auth type (type d'authentification), choisissez AWS_IAM (IAM_AWS).
7. (Facultatif) Pour Statement ID (ID de déclaration), saisissez un ID de déclaration pour votre déclaration de stratégie.
8. Pour Principal, saisissez l'ID de compte ou l'Amazon Resource Name (ARN) de l'utilisateur ou du rôle auquel vous souhaitez accorder des autorisations. Par exemple : **444455556666**.
9. Choisissez Enregistrer.

Vous pouvez également créer cette déclaration de politique à l'aide de la commande add permission AWS Command Line Interface (AWS CLI) suivante :

```
aws lambda add-permission --function-name my-function \
    --statement-id example0-cross-account-statement \
    --action lambda:InvokeFunctionUrl \
    --principal 444455556666 \
    --function-url-auth-type AWS_IAM
```

Dans l'exemple précédent, la valeur de la clé de condition `lambda:FunctionUrlAuthType` est `AWS_IAM`. Cette stratégie n'autorise l'accès que lorsque le type d'authentification de l'URL de votre fonction est également `AWS_IAM`.

## Utilisation du type d'authentification **NONE**

> **Important**
>
> Lorsque le type d'authentification de votre URL de fonction est `NONE` et que vous disposez d'une stratégie basée sur une ressource qui accorde un accès public, tout utilisateur non authentifié ayant de votre URL de fonction peut invoquer votre fonction.

Dans certains cas, vous pouvez souhaiter que votre URL de fonction soit publique. Vus pourriez souhaiter répondre aux demandes envoyées directement à partir d'un navigateur Web. Pour autoriser l'accès public à votre URL de fonction, choisissez le type d'authentification `NONE`.

Si vous choisissez le `NONE` type d'authentification, Lambda n'utilise pas IAM pour authentifier les demandes vers votre URL de fonction. Toutefois, les utilisateurs doivent toujours avoir des autorisations `lambda:InvokeFunctionUrl` afin d'invoquer correctement l'URL de votre fonction.

Vous pouvez accorder des autorisations `lambda:InvokeFunctionUrl` à l'aide de la stratégie basée sur une ressource suivante :

Example exemple de stratégie d'invocation d'URL de fonction pour tous les principaux non authentifiés

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

> **Note**
>
> Lorsque vous créez une URL de fonction avec un type d'authentification `NONE` via la console ou AWS Serverless Application Model (AWS SAM), Lambda crée automatiquement pour vous la déclaration de politique basée sur les ressources précédente. Si la stratégie existe déjà, ou si l'utilisateur ou le rôle qui crée l'application ne dispose pas des autorisations appropriées, Lambda ne la créera pas pour vous. Si vous utilisez directement l'API AWS CLI AWS CloudFormation, ou l'API Lambda, vous devez ajouter vous-même `lambda:InvokeFunctionUrl` des autorisations. Cela rend votre fonction publique. En outre, si vous supprimez l'URL de votre fonction avec un type d'authentification `NONE`, Lambda ne supprime pas automatiquement la politique basée sur les ressources associée. Si vous souhaitez supprimer cette politique, vous devez le faire manuellement.

Dans cette déclaration, la valeur de la clé de condition `lambda:FunctionUrlAuthType` est `NONE`. Cette déclaration de stratégie n'autorise l'accès que lorsque le type d'authentification de votre URL de fonction est également `NONE`.

Si la stratégie basée sur les ressources d'une fonction n'accorde pas d'autorisations `lambda:invokeFunctionUrl`, les utilisateurs obtiendront alors un code d'erreur 403 Forbidden lorsqu'ils essaient d'invoquer l'URL de votre fonction, même si l'URL de la fonction utilise le type d'autorisation `NONE`.

## Gouvernance et contrôle d'accès

Outre les autorisations d'invocation d'URL de fonction, vous pouvez également contrôler l'accès aux actions utilisées pour configurer les URL de fonctions. Lambda prend en charge les actions de stratégie IAM suivantes pour les URL de fonctions :

- `lambda:InvokeFunctionUrl` – Invoquer une fonction Lambda à l'aide de l'URL de fonction.
- `lambda:CreateFunctionUrlConfig` – Créer une URL de fonction et définir son `AuthType`.
- `lambda:UpdateFunctionUrlConfig` – Mettre à jour la configuration d'une URL de fonction et son `AuthType`.
- `lambda:GetFunctionUrlConfig` – Affichez les détails d'une URL de fonction.
- `lambda:ListFunctionUrlConfigs` – Répertorier les configurations d'URL de fonction.
- `lambda:DeleteFunctionUrlConfig` – Supprimer une URL de fonction.

> **Note**
>
> La console Lambda prend en charge l'ajout d'autorisations uniquement pour `lambda:InvokeFunctionUrl`. Pour toutes les autres actions, vous devez ajouter des autorisations à l'aide de l'API Lambda ou AWS CLI.

Pour autoriser ou refuser l'accès à l'URL des fonctions à d'autres AWS entités, incluez ces actions dans les politiques IAM. Par exemple, la politique suivante accorde au `example` rôle dans les Compte AWS `444455556666` autorisations de mettre à jour l'URL de la fonction pour la fonction **`my-function`** dans le compte`123456789012`.

Example exemple de stratégie d'URL de fonction inter-comptes

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/example"
            },
            "Action": "lambda:UpdateFunctionUrlConfig",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
        }
    ]
}
```

Clés de condition

Pour un contrôle d'accès précis sur vos URL de fonctions, utilisez une clé de condition. Lambda prend en charge une clé de condition supplémentaire pour les URL de fonctions : `FunctionUrlAuthType`. La clé `FunctionUrlAuthType` définit une valeur d'énumération décrivant le type d'authentification utilisé par l'URL de votre fonction. La valeur peut être `AWS_IAM` ou `NONE`.

Vous pouvez utiliser cette clé de condition dans les stratégies associées à votre fonction. Vous pourriez vouloir limiter les personnes pouvant apporter des modifications de configuration à vos URL de fonctions. Pour rejeter toutes les demandes `UpdateFunctionUrlConfig` à n'importe quelle fonction avec un type d'authentification URL `NONE`, vous pouvez définir la stratégie suivante :

Example exemple de stratégie d'URL de fonction avec refus explicite

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action":[
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
```

```
}
```

Pour accorder le `example` rôle dans Compte AWS `444455556666` les autorisations de création `CreateFunctionUrlConfig` et de `UpdateFunctionUrlConfig` requêtes sur les fonctions avec un type d'authentification URL`AWS_IAM`, vous pouvez définir la politique suivante :

Example exemple de stratégie d'URL de fonction avec autorisation explicite

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/example"
            },
            "Action":[
                "lambda:CreateFunctionUrlConfig",
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

Vous pouvez également utiliser cette clé de condition dans une stratégie de contrôle de service (SCP). Utilisez les SCP pour gérer les autorisations dans toute une organisation dans AWS Organizations. Pour empêcher les utilisateurs de créer ou de mettre à jour des URL de fonctions qui utilisent autre chose que le type d'authentification `AWS_IAM`, utilisez la stratégie de contrôle de service suivante :

Example exemple d'URL de fonction SCP avec refus explicite

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Deny",
            "Action":[
                "lambda:CreateFunctionUrlConfig",
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:*:123456789012:function:*",
            "Condition": {
                "StringNotEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

# Invocation de la fonction Lambda URLs

Une fonction URL est un point de terminaison HTTP (S) dédié à votre fonction Lambda. Vous pouvez créer et configurer une fonction URL via la console Lambda ou Lambda. API Lorsque vous créez une fonctionURL, Lambda génère automatiquement un point de URL terminaison unique pour vous. Une fois que vous avez créé une fonctionURL, son URL point de terminaison ne change jamais. Les URL points de terminaison des fonctions ont le format suivant :

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> URLsLes fonctions ne sont pas prises en charge dans les pays suivants Régions AWS : Asie-Pacifique (Hyderabad`ap-south-2`) (), Asie-Pacifique (Melbourne`ap-southeast-4`) (), Asie-Pacifique (Malaisie`ap-southeast-5`) (), Canada Ouest (Calgary) (`ca-west-1`), Europe (Espagne) (`eu-south-2`), Europe (Zurich) (`eu-central-2`), Israël (Tel Aviv) (`il-central-1`) et Moyen-Orient (UAE) (`me-central-1`).

URLsLes fonctions sont compatibles avec le double empilage, supportant et. IPv4 IPv6 Après avoir configuré votre fonctionURL, vous pouvez appeler votre fonction via son point de terminaison HTTP (S) via un navigateur Web, curl, Postman ou n'importe quel HTTP client. Pour appeler une fonctionURL, vous devez disposer d'`lambda:InvokeFunctionUrl`autorisations. Pour de plus amples informations, veuillez consulter Contrôle d'accès.

Rubriques

- Principes de base de URL l'invocation de fonctions
- Charges utiles de demandes et de réponses

## Principes de base de URL l'invocation de fonctions

Si votre fonction URL utilise le type d'`AWS_IAM`authentification, vous devez signer chaque HTTP demande à l'aide de AWS Signature Version 4 (SigV4). Des outils tels qu'awscurl, Postman et AWS SigV4 Proxy offrent des moyens intégrés de signer vos demandes avec SigV4.

Si vous n'utilisez aucun outil pour signer les HTTP demandes adressées à votre fonctionURL, vous devez signer manuellement chaque demande à l'aide de SigV4. Lorsque votre fonction URL

reçoit une demande, Lambda calcule également la signature SigV4. Lambda traite la demande uniquement si les signatures correspondent. Pour obtenir des instructions sur la façon de signer manuellement vos demandes avec SigV4, voir Signature des AWS demandes avec signature version 4 du Référence générale d'Amazon Web Services guide.

Si votre fonction URL utilise le type d'`NONE`authentification, vous n'êtes pas obligé de signer vos demandes à l'aide de SigV4. Vous pouvez appeler votre fonction à l'aide d'un navigateur Web, de curl, de Postman ou de n'importe quel HTTP client.

Pour tester des demandes `GET` simples vers votre fonction, utilisez un navigateur web. Par exemple, si votre fonction URL est `https://abcdefg.lambda-url.us-east-1.on.aws` et qu'elle prend un paramètre de chaîne`message`, votre requête URL pourrait ressembler à ceci :

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Pour tester d'autres HTTP requêtes, comme une `POST` requête, vous pouvez utiliser un outil tel que curl. Par exemple, si vous souhaitez inclure des JSON données dans une `POST` requête adressée à votre fonctionURL, vous pouvez utiliser la commande curl suivante :

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \
-H 'content-type: application/json' \
-d '{ "example": "test" }'
```

## Charges utiles de demandes et de réponses

Lorsqu'un client appelle votre fonctionURL, Lambda fait correspondre la demande à un objet d'événement avant de la transmettre à votre fonction. La réponse de votre fonction est ensuite mappée à une HTTP réponse que Lambda renvoie au client via la fonction. URL

Les formats des événements de demande et de réponse suivent le même schéma que le format de charge utile Amazon API Gateway version 2.0.

Format de la charge utile de demande

Une charge utile de demande a la structure suivante :

```
{
  "version": "2.0",
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
```

```
  "cookies": [
    "cookie1",
    "cookie2"
  ],
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "<urlid>",
    "authentication": null,
    "authorizer": {
        "iam": {
                "accessKey": "AKIA...",
                "accountId": "111122223333",
                "callerId": "AIDA...",
                "cognitoIdentity": null,
                "principalOrgId": null,
                "userArn": "arn:aws:iam::111122223333:user/example-user",
                "userId": "AIDA..."
        }
    },
    "domainName": "<url-id>.lambda-url.us-west-2.on.aws",
    "domainPrefix": "<url-id>",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "123.123.123.123",
      "userAgent": "agent"
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
    "timeEpoch": 1583348638390
  },
  "body": "Hello from client!",
  "pathParameters": null,
```

```
  "isBase64Encoded": false,
  "stageVariables": null
}
```

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `version` | La version du format de la charge utile pour cet événement. La fonction Lambda prend URLs actuellement en charge le format de charge utile version 2.0. | `2.0` |
| `routeKey` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit sur `$default` comme espace réservé. | `$default` |
| `rawPath` | Chemin d'accès de la demande. Par exemple, si la demande l'URLest`https://{url-id}.lambda-url.{region}.on.aws/example/test/demo` , la valeur brute du chemin est`/example/test/demo` . | `/example/test/demo` |
| `rawQueryString` | La chaîne brute contenant les paramètres de la chaîne de la requête de la demande. Les caractères pris en charge sont les suivants : `a-z`, `A-Z`, `0-9`, `.`, `_`, `-`, `%`, &, =, et +. | `"?parameter1=value1&parameter2=value2"` |

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `cookies` | Un tableau contenant tous les cookies envoyés dans le cadre de la demande. | `["Cookie_1=Value_1", "Cookie_2=Value_2"]` |
| `headers` | La liste des en-têtes de demande, présentée sous forme de paires valeur-clé. | `{"header1": "value1", "header2": "value2"}` |
| `queryStringParameters` | Paramètres de requête pour la demande. Par exemple, si la demande URL est le cas`https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane` , la `queryStringParameters` valeur est un JSON objet avec une clé `name` et une valeur de`Jane`. | `{"name": "Jane"}` |
| `requestContext` | Un objet qui contient des informations supplémentaires sur la demande, telles que l'`requestId` heure de la demande et l'identité de l'appelant s'il est autorisé via AWS Identity and Access Management (IAM). | |
| `requestContext.accountId` | Compte AWS ID du propriétaire de la fonction. | `"123456789012"` |
| `requestContext.apiId` | ID de la fonctionURL. | `"33anwqw8fj"` |
| `requestContext.authentication` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit sur `null`. | `null` |

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `requestContext.authorizer` | Objet contenant des informations sur l'identité de l'appelant, si la fonction URL utilise le type d'`AWS_IAM`authentification. Sinon, Lambda définit cette valeur sur `null`. | |
| `requestContext.authorizer.iam.accessKey` | La clé d'accès de l'identité de l'appelant. | `"AKIAIOSFODNN7EXAMPLE"` |
| `requestContext.authorizer.iam.accountId` | Compte AWS Identifiant de l'identité de l'appelant. | `"111122223333"` |
| `requestContext.authorizer.iam.callerId` | L'ID (ID d'utilisateur) de l'appelant. | `"AIDACKCEVSQ6C2EXAMPLE"` |
| `requestContext.authorizer.iam.cognitoIdentity` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit `null` ou l'exclut du. JSON | `null` |
| `requestContext.authorizer.iam.principalOrgId` | L'ID de l'organisation du principal associé à l'ID de l'appelant. | `"AIDACKCEVSQORGEXAMPLE"` |
| `requestContext.authorizer.iam.userArn` | L'utilisateur Amazon Resource Name (ARN) de l'identité de l'appelant. | `"arn:aws:iam::111122223333:user/example-user"` |
| `requestContext.authorizer.iam.userId` | L’identifiant utilisateur de l’identité de l'appelant. | `"AIDACOSFODNN7EXAMPLE2"` |

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `requestContext.domainName` | Le nom de domaine de la fonctionURL. | `"<url-id>.lambda-url.us-west-2.on.aws"` |
| `requestContext.domainPrefix` | Le préfixe de domaine de la fonction. URL | `"<url-id>"` |
| `requestContext.http` | Objet contenant des informations sur la HTTP demande. | |
| `requestContext.http.method` | HTTPMéthode utilisée dans cette demande. Les valeurs valides sont les suivantes : `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`, `PATCH` et `DELETE`. | `GET` |
| `requestContext.http.path` | Chemin d'accès de la demande. Par exemple, si la demande l'URLest`https://{url-id}.lambda-url.{region}.on.aws/example/test/demo` , la valeur du chemin est/`example/test/demo` . | `/example/test/demo` |
| `requestContext.http.protocol` | Le protocole de la demande. | `HTTP/1.1` |
| `requestContext.http.sourceIp` | Adresse IP source de la TCP connexion immédiate à l'origine de la demande. | `123.123.123.123` |
| `requestContext.http.userAgent` | La valeur de l'en-tête de la demande Utilisateur-Agent. | `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0` |

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `requestContext.requestId` | L'identifiant de la demande d'invocation. Vous pouvez utiliser cet ID pour suivre les journaux d'invocation liés à votre fonction. | `e1506fd5-9e7b-434f-bd42-4f8fa224b599` |
| `requestContext.routeKey` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit sur `$default` comme espace réservé. | `$default` |
| `requestContext.stage` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit sur `$default` comme espace réservé. | `$default` |
| `requestContext.time` | Horodatage de la demande. | `"07/Sep/2021:22:50:22 +0000"` |
| `requestContext.timeEpoch` | L'horodatage de la demande, en heure d'époque Unix. | `"1631055022677"` |
| body | Le corps de la demande. Si le type de contenu de la demande est binaire, le corps est codé en base64. | `{"key1": "value1", "key2": "value2"}` |
| `pathParameters` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit `null` ou l'exclut du. JSON | `null` |
| `isBase64Encoded` | TRUE si le corps est une charge utile binaire et codé en base64. Sinon `FALSE`. | `FALSE` |

| Paramètre | Description | Exemple |
| --- | --- | --- |
| `stageVariables` | Fonction : URLs n'utilisez pas ce paramètre. Lambda le définit `null` ou l'exclut du. JSON | `null` |

Format de la charge utile de la réponse

Lorsque votre fonction renvoie une réponse, Lambda l'analyse et la convertit en réponse. HTTP Les charges utiles de la réponse de fonction ont le format suivant :

```
{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/json",
        "My-Custom-Header": "Custom Value"
    },
    "body": "{ \"message\": \"Hello, world!\" }",
    "cookies": [
        "Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",
        "Cookie_2=Value2; Max-Age=78000"
    ],
    "isBase64Encoded": false
}
```

Lambda déduit le format de réponse pour vous. Si votre fonction renvoie une valeur valide JSON et ne renvoie pas `astatusCode`, Lambda part du principe suivant :

- `statusCode` est `200`.
- `content-type` est `application/json`.
- `body` est la réponse de la fonction.
- `isBase64Encoded` est `false`.

Les exemples suivants montrent comment la sortie de votre fonction Lambda correspond à la charge utile de réponse, et comment la charge utile de réponse correspond à la réponse finale. HTTP Lorsque le client appelle votre fonctionURL, il voit la HTTP réponse.

Exemple de sortie pour une réponse sous forme de chaîne

| Sortie de la fonction Lambda | Sortie de réponse interprétée | HTTPréponse (ce que voit le client) |
|---|---|---|
| `"Hello, world!"` | `{ "statusCode": 200, "body": "Hello, world!", "headers": { "content-type": "application/json" }, "isBase64Encoded": false }` | `HTTP/2 200`<br>`date: Wed, 08 Sep 2021 18:02:24 GMT`<br>`content-type: application/json`<br>`content-length: 15`<br><br>`"Hello, world!"` |

Exemple de sortie pour une JSON réponse

| Sortie de la fonction Lambda | Sortie de réponse interprétée | HTTPréponse (ce que voit le client) |
|---|---|---|
| `{ "message": "Hello, world!" }` | `{ "statusCode": 200, "body": { "message": "Hello, world!" }, "headers": { "content-type": "application/json" }, "isBase64Encoded": false }` | `HTTP/2 200`<br>`date: Wed, 08 Sep 2021 18:02:24 GMT`<br>`content-type: application/json`<br>`content-length: 34`<br><br>`{ "message": "Hello, world!" }` |

Exemple de sortie pour une réponse personnalisée

| Sortie de la fonction Lambda | Sortie de réponse interprétée | HTTPréponse (ce que voit le client) |
| --- | --- | --- |
| <pre>{<br>   "statusCode": 201,<br>    "headers": {<br>        "Content-Type":<br> "application/json",<br>        "My-Custom-<br>Header": "Custom Value"<br>    },<br>    "body": JSON.stri<br>ngify({<br>        "message":<br> "Hello, world!"<br>    }),<br>    "isBase64Encoded":<br> false<br>}</pre> | <pre>{<br>   "statusCode": 201,<br>    "headers": {<br>        "Content-Type":<br> "application/json",<br>        "My-Custom-<br>Header": "Custom Value"<br>    },<br>    "body": JSON.stri<br>ngify({<br>        "message":<br> "Hello, world!"<br>    }),<br>    "isBase64Encoded":<br> false<br>}</pre> | <pre>HTTP/2 201<br>date: Wed, 08 Sep 2021<br> 18:02:24 GMT<br>content-type: applicati<br>on/json<br>content-length: 27<br>my-custom-header:<br> Custom Value<br><br>{<br>  "message": "Hello,<br> world!"<br>}</pre> |

### Cookies

Pour renvoyer des cookies depuis votre fonction, n'ajoutez pas manuellement des en-têtes `set-cookie`. Incluez plutôt les cookies dans votre objet de charge utile de réponse. Lambda les interprète automatiquement et les ajoute en tant qu'`set-cookie`en-têtes dans votre HTTP réponse, comme dans l'exemple suivant.

| Sortie de la fonction Lambda | HTTPréponse (ce que voit le client) |
| --- | --- |
| <pre>{<br>   "statusCode": 201,<br>    "headers": {<br>        "Content-Type": "application/<br>json",<br>        "My-Custom-Header": "Custom<br> Value"<br>    },<br>    "body": JSON.stringify({<br>        "message": "Hello, world!"</pre> | <pre>HTTP/2 201<br>date: Wed, 08 Sep 2021 18:02:24 GMT<br>content-type: application/json<br>content-length: 27<br>my-custom-header: Custom Value<br>set-cookie: Cookie_1=Value2;<br> Expires=21 Oct 2021 07:48 GMT<br>set-cookie: Cookie_2=Value2; Max-<br>Age=78000</pre> |

| Sortie de la fonction Lambda | HTTPréponse (ce que voit le client) |
| --- | --- |
| <pre>    }),<br>    "cookies": [<br>        "Cookie_1=Value1; Expires=21<br> Oct 2021 07:48 GMT",<br>        "Cookie_2=Value2; Max-Age=7<br>8000"<br>    ],<br>    "isBase64Encoded": false<br>}</pre> | <pre>{<br>  "message": "Hello, world!"<br>}</pre> |

# Surveillance des URL de fonctions Lambda

Vous pouvez utiliser AWS CloudTrail Amazon CloudWatch pour surveiller les URL de vos fonctions.

Rubriques


- URL des fonctions de surveillance avec CloudTrail
- CloudWatch métriques pour les URL des fonctions


## URL des fonctions de surveillance avec CloudTrail

Pour les URL de fonctions, Lambda prend automatiquement en charge l'enregistrement des opérations d'API suivantes sous forme d'événements CloudTrail dans des fichiers journaux :

- CreateFunctionUrlConfig
- UpdateFunctionUrlConfig
- DeleteFunctionUrlConfig
- GetFunctionUrlConfig
- ListFunctionUrlConfigs

Chaque entrée de journal contient des informations sur l'identité de l'appelant, le moment où la demande a été faite, et d'autres détails. Vous pouvez voir tous les événements des 90 derniers jours en consultant l'historique de vos CloudTrail événements. Pour conserver les enregistrements de plus de 90 jours, vous pouvez créer un journal d'activité.

Par défaut, CloudTrail n'enregistre pas les `InvokeFunctionUrl` demandes, qui sont considérées comme des événements de données. Vous pouvez toutefois activer la connexion aux événements de données CloudTrail. Pour plus d’informations, veuillez consulter Consignation d’événements de données pour les journaux d’activité dans le Guide de l’utilisateur AWS CloudTrail .

## CloudWatch métriques pour les URL des fonctions

Lambda envoie des métriques agrégées concernant les demandes d'URL de fonction à. CloudWatch Grâce à ces indicateurs, vous pouvez surveiller les URL de vos fonctions, créer des tableaux de bord et configurer des alarmes dans la CloudWatch console.

Les URL de fonctions prennent en charge les métriques d'invocation suivantes. Nous vous recommandons de consulter ces métriques avec la statistique `Sum`.

- `UrlRequestCount` – Nombre de demandes faites vers cette URL de fonction.
- `Url4xxCount` – Nombre de demandes ayant retourné un code d'état HTTP 4XX. Les codes de la série 4XX indiquent des erreurs côté client, telles que des demandes erronées.
- `Url5xxCount` – Nombre de demandes ayant retourné un code d'état HTTP 5XX. Les codes de la série 5XX indiquent des erreurs côté serveur, telles que des erreurs de fonction et des délais d'attente.

Les URL de fonctions prennent également en charge la métrique de performance suivante. Nous vous recommandons de consulter ces métriques avec les statistiques `Average` ou `Max`.

- `UrlRequestLatency` – Le temps écoulé entre le moment où l'URL de fonction reçoit une demande et celui où l'URL de fonction renvoie une réponse.

Chacune de ces métriques d'invocation et de performance prend en charge les dimensions suivantes :

- `FunctionName` – Affichez les métriques agrégées pour les URL de fonctions attribuées à la version non publiée `$LATEST` d'une fonction, ou à tout alias de la fonction. Par exemple, `hello-world-function`.
- `Resource` – Affichez les métriques d'une URL de fonction spécifique. Cela est défini par un nom de fonction, accompagné de la version non publiée `$LATEST` de la fonction ou d'un alias de la fonction. Par exemple, `hello-world-function:$LATEST`.
- `ExecutedVersion` – Affichez les métriques d'une URL de fonction spécifique en fonction de la version exécutée. Vous pouvez utiliser cette dimension principalement pour suivre l'URL de la fonction affectée à la version non publiée `$LATEST`.

# Didacticiel : Creating a Lambda function with a function URL (Création d'une fonction Lambda avec une URL de fonction)

Dans ce didacticiel, vous créez une fonction Lambda définie comme archive de fichier ZIP avec un point de terminaison d'URL de fonction public qui renvoie le produit de deux nombres. Pour de plus amples informations sur la configuration des URL de fonctions, consultez Fonction URLs.

## Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines commandes CLI Bash que vous utilisez couramment avec Lambda (par exemple `zip`) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. Les exemples de commandes CLI de ce guide utilisent le formatage Linux. Les commandes qui incluent des documents JSON en ligne doivent être reformatées si vous utilisez la CLI Windows.

## Créer un rôle d’exécution

Créez le rôle d’exécution qui donne à votre fonction Lambda l’autorisation d’accéder aux ressources AWS .

Pour créer un rôle d’exécution

1. Ouvrez la page Rôles de la console AWS Identity and Access Management (IAM).
2. Sélectionnez Créer un rôle.
3. Pour Type d'entité fiable, sélectionnez AWS service, puis pour Cas d'utilisation, sélectionnez Lambda.
4. Choisissez Suivant.
5. Dans le volet Permissions policies, entrez **`AWSLambdaBasicExecutionRole`** dans le champ de recherche.
6. Cochez la case à côté de la politique `AWSLambdaBasicExecutionRole` AWS gérée, puis choisissez Next.
7. Entrez **`lambda-url-role`** le nom du rôle, puis choisissez Créer un rôle.

La `AWSLambdaBasicExecutionRole` politique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux sur Amazon Logs. Plus loin dans le didacticiel, vous aurez besoin du nom de ressource Amazon (ARN) du rôle pour créer votre fonction Lambda.

Pour trouver l'ARN de votre rôle d'exécution

1. Ouvrez la page Rôles de la console AWS Identity and Access Management (IAM).
2. Sélectionnez le rôle que vous venez de créer (`lambda-url-role`).
3. Dans le volet Résumé, copiez l'ARN.

## Création d’une fonction Lambda avec une URL de fonction (archive de fichier ZIP)

Créer une fonction Lambda avec un point de terminaison d’URL de fonction en utilisant une archive de fichier .zip.

Pour créer la fonction

1. Copiez l’exemple de code suivant dans un fichier nommé `index.js`.

Example index.js

```
exports.handler = async (event) => {
    let body = JSON.parse(event.body);
    const product = body.num1 * body.num2;
    const response = {
        statusCode: 200,
        body: "The product of " + body.num1 + " and " + body.num2 + " is " +
 product,
    };
    return response;
};
```

2. Créez un package de déploiement.

```
zip function.zip index.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`. Assurez-vous de remplacer l'ARN du rôle par l'ARN de votre propre rôle d'exécution que vous avez copié plus tôt dans le didacticiel.

```
aws lambda create-function \
    --function-name my-url-function \
    --runtime nodejs18.x \
    --zip-file fileb://function.zip \
    --handler index.handler \
    --role arn:aws:iam::123456789012:role/lambda-url-role
```

4. Ajoutez une stratégie basée sur les ressources à votre fonction en accordant des autorisations pour autoriser l'accès public à l'URL de votre fonction.

```
aws lambda add-permission \
    --function-name my-url-function \
    --action lambda:InvokeFunctionUrl \
    --principal "*" \
    --function-url-auth-type "NONE" \
    --statement-id url
```

5. Créez un point de terminaison d'URL pour la fonction avec la commande `create-function-url-config`.

```
aws lambda create-function-url-config \
    --function-name my-url-function \
    --auth-type NONE
```

## Test du point de terminaison d'URL de fonction

Appelez votre fonction Lambda en appelant le point de terminaison d'URL de la fonction avec un client HTTP tel que curl ou Postman.

```
curl 'https://abcdefg.lambda-url.us-east-1.on.aws/' \
-H 'Content-Type: application/json' \
-d '{"num1": "10", "num2": "10"}'
```

Vous devriez voir la sortie suivante :

```
The product of 10 and 10 is 100
```

## Création d'une fonction Lambda avec une URL de fonction () CloudFormation

Vous pouvez également créer une fonction Lambda avec un point de terminaison URL de fonction en utilisant le AWS CloudFormation type. `AWS::Lambda::Url`

```
Resources:
  MyUrlFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs18.x
      Role: arn:aws:iam::123456789012:role/lambda-url-role
      Code:
        ZipFile: |
          exports.handler = async (event) => {
              let body = JSON.parse(event.body);
              const product = body.num1 * body.num2;
              const response = {
                  statusCode: 200,
                  body: "The product of " + body.num1 + " and " + body.num2 + " is " +
 product,
              };
```

```
            return response;
        };
    Description: Create a function with a URL.
  MyUrlFunctionPermissions:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref MyUrlFunction
      Action: lambda:InvokeFunctionUrl
      Principal: "*"
      FunctionUrlAuthType: NONE
  MyFunctionUrl:
    Type: AWS::Lambda::Url
    Properties:
      TargetFunctionArn: !Ref MyUrlFunction
      AuthType: NONE
```

## Créer une fonction Lambda avec une URL de fonction (AWS SAM)

Vous pouvez également créer une fonction Lambda configurée avec une URL de fonction en utilisant AWS Serverless Application Model ()AWS SAM.

```
ProductFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: function/.
    Handler: index.handler
    Runtime: nodejs18.x
    AutoPublishAlias: live
    FunctionUrlConfig:
      AuthType: NONE
```

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer le rôle d’exécution

1. Ouvrez la page Roles (Rôles) de la console IAM.
2. Sélectionnez le rôle d’exécution que vous avez créé.

3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

# Comprendre la mise à l'échelle des fonctions Lambda

La simultanéité est le nombre de demandes en cours de vol traitées simultanément par votre AWS Lambda fonction. Pour chaque demande simultanée, Lambda fournit une instance distincte de votre environnement d'exécution. Au fur et à mesure que vos fonctions reçoivent des demandes, Lambda gère automatiquement la mise à l'échelle du nombre d'environnements d'exécution jusqu'à ce que vous atteigniez la limite de simultanéité de votre compte. Par défaut, Lambda fournit à votre compte une limite de simultanéité totale de 1 000 exécutions simultanées pour toutes les fonctions d'une Région AWS. Pour répondre aux besoins spécifiques de votre compte, vous pouvez demander une augmentation du quota et configurer les contrôles de simultanéité au niveau des fonctions afin que vos fonctions critiques ne soient pas limitées.

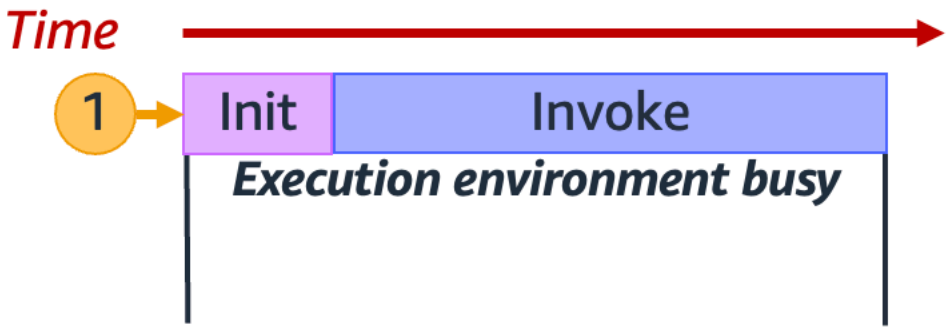
Cette rubrique explique les concepts de simultanéité et de mise à l'échelle des fonctions dans Lambda. À la fin de cette rubrique, vous serez en mesure de comprendre comment calculer la simultanéité, de visualiser les deux principales options de contrôle de la simultanéité (réservée et provisionnée), d'estimer les paramètres de contrôle de la simultanéité appropriés et de visualiser les métriques pour une optimisation supplémentaire.

Sections


- Comprendre et visualiser la simultanéité
- Calcul de la simultanéité d'une fonction
- Comprendre la simultanéité réservée et la simultanéité provisionnée
- Comprendre la simultanéité et le nombre de demandes par seconde
- Quotas de simultanéité
- Configuration de la simultanéité réservée pour une fonction
- Configuration de la simultanéité provisionnée pour une fonction
- Comportement de mise à l'échelle Lambda
- Surveillance de la simultanéité


## Comprendre et visualiser la simultanéité

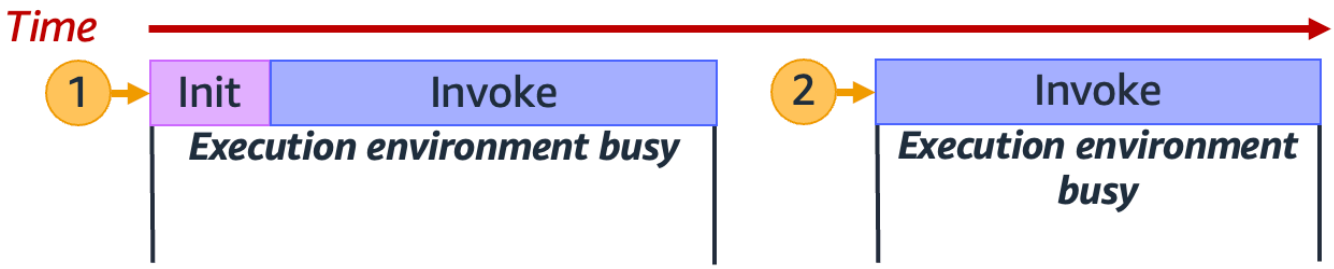
Lambda invoque votre fonction dans un environnement d'exécution sécurisé et isolé. Pour traiter une demande, Lambda doit d'abord initialiser un environnement d'exécution (la phase Init), avant de l'utiliser pour invoquer votre fonction (la phase Invoke) :




> **Note**
>
> Les durées réelles d'Init et Invoke peuvent varier en fonction de nombreux facteurs, tels que l'environnement d'exécution choisi et le code de la fonction Lambda. Le diagramme précédent n'est pas censé représenter les proportions exactes des durées des phases Init et Invoke.

Le diagramme précédent utilise un rectangle pour représenter un seul environnement d'exécution. Lorsque votre fonction reçoit sa toute première demande (représentée par le cercle jaune avec l'étiquette 1), Lambda crée un nouvel environnement d'exécution et exécute le code en dehors de votre gestionnaire principal pendant la phase Init. Ensuite, Lambda exécute le code du gestionnaire principal de votre fonction pendant la phase Invoke. Pendant tout ce processus, cet environnement d'exécution est occupé et ne peut pas traiter d'autres demandes.

Lorsque Lambda a fini de traiter la première demande, cet environnement d'exécution peut alors traiter des demandes supplémentaires pour la même fonction. Pour les demandes suivantes, Lambda n'a pas besoin de réinitialiser l'environnement.

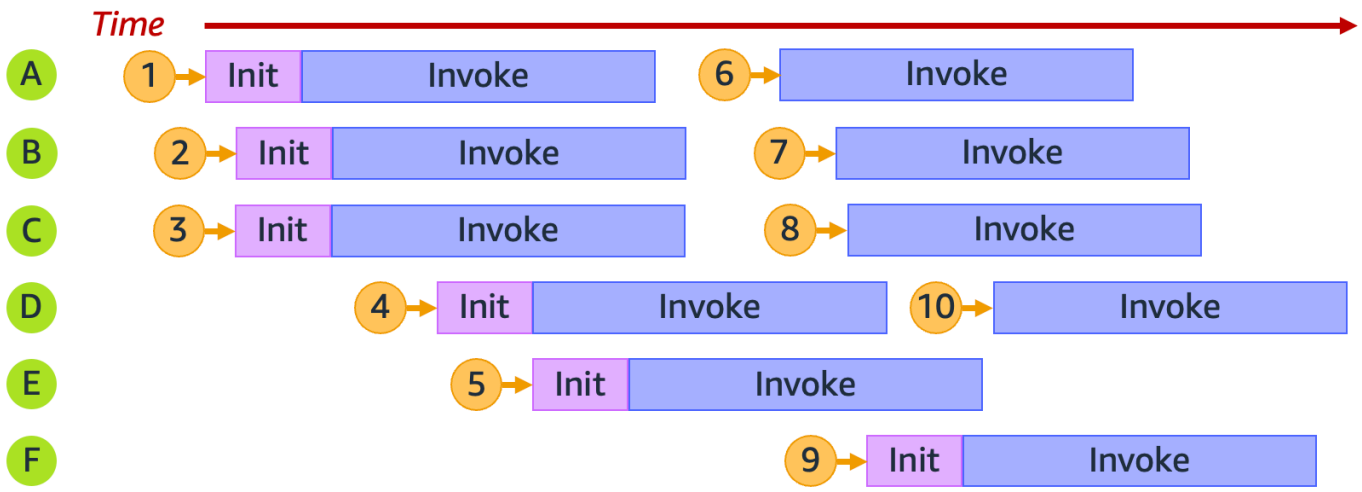


Dans le schéma précédent, Lambda réutilise l'environnement d'exécution pour traiter la deuxième demande (représentée par le cercle jaune avec l'étiquette 2).

Jusqu'à présent, nous nous sommes concentrés sur une seule instance de votre environnement d'exécution (c.-à-d. une simultanéité de 1). En pratique, Lambda peut avoir besoin de provisionner plusieurs instances d'environnement d'exécution en parallèle pour traiter toutes les demandes entrantes. Lorsque votre fonction reçoit une nouvelle demande, l'une des deux choses suivantes peut se produire :

- Si une instance d'environnement d'exécution pré-initialisée est disponible, Lambda l'utilise pour traiter la demande.
- Sinon, Lambda crée une nouvelle instance d'environnement d'exécution pour traiter la demande.

Par exemple, explorons ce qui se passe lorsque votre fonction reçoit 10 demandes :



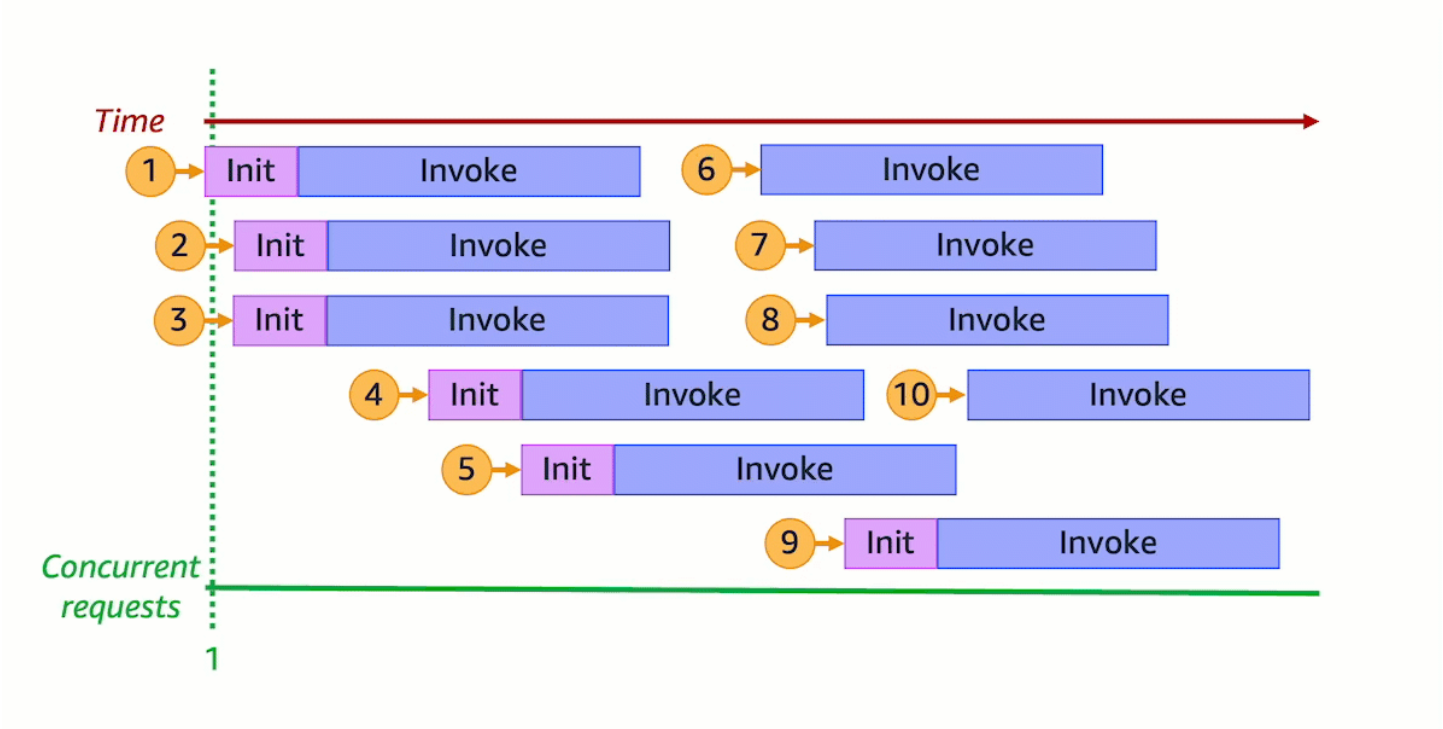

Dans le diagramme précédent, chaque plan horizontal représente une seule instance d'environnement d'exécution (étiquetée de A à F). Voici comment Lambda traite chaque demande :

| Demande | Comportement de Lambda | Raisonnement |
|---|---|---|
| 1 | Alloue un nouvel environnement A | C'est la première demande ; aucune instance d'environnement d'exécution n'est disponible. |

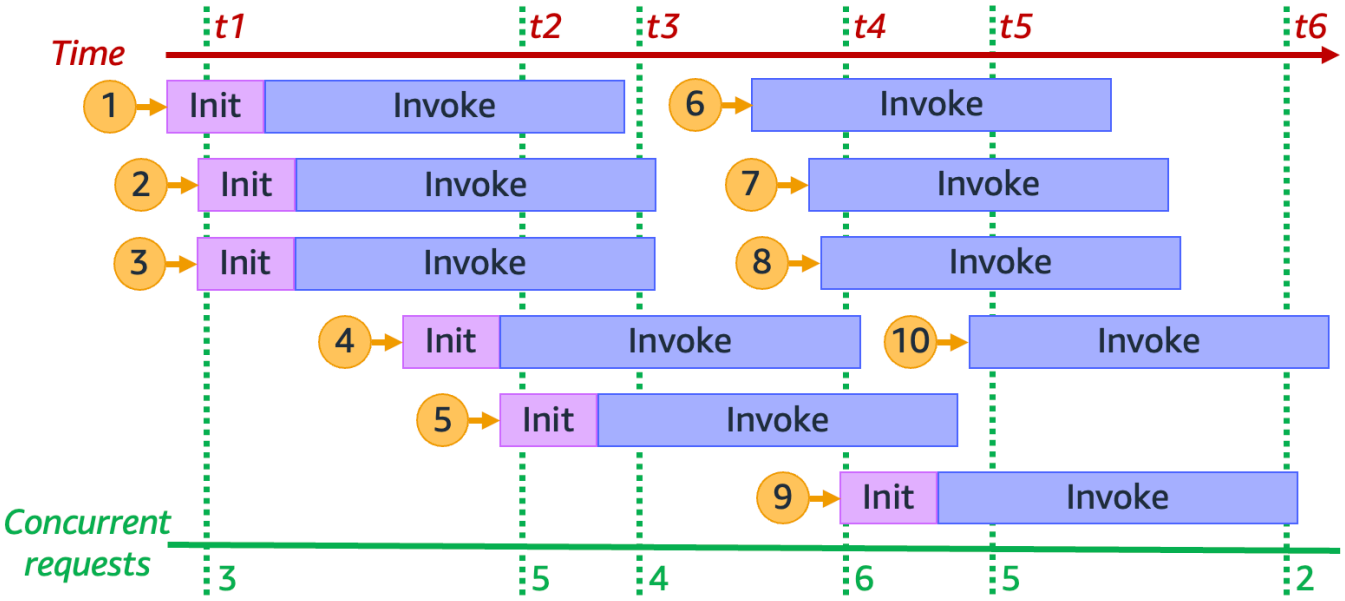
| Demande | Comportement de Lambda | Raisonnement |
| --- | --- | --- |
| 2 | Alloue un nouvel environnement B | L'instance A de l'environnement d'exécution existant est occupée. |
| 3 | Alloue un nouvel environnement C | Les instances d'environnement d'exécution existantes A et B sont toutes deux occupées. |
| 4 | Alloue un nouvel environnement D | Les instances d'environnement d'exécution existantes A, B et C sont toutes occupées. |
| 5 | Alloue un nouvel environnement E | Les instances A, B, C et D de l'environnement d'exécution existant sont toutes occupées. |
| 6 | Réutilise l'environnement A | L'instance d'environnement d'exécution A a fini de traiter la demande 1 et est maintenant disponible. |
| 7 | Réutilise l'environnement B | L'instance d'environnement d'exécution B a fini de traiter la demande 2 et est maintenant disponible. |
| 8 | Réutilise l'environnement C | L'instance d'environnement d'exécution C a terminé le traitement de la demande 3 et est maintenant disponible. |

| Demande | Comportement de Lambda | Raisonnement |
|---|---|---|
| 9 | Alloue un nouvel environnement F | Les instances d'environnement d'exécution existantes A, B, C, D et E sont toutes occupées. |
| 10 | Réutilise l'environnement D | L'instance d'environnement d'exécution D a fini de traiter la demande 4 et est maintenant disponible. |

Au fur et à mesure que votre fonction reçoit des demandes simultanées, Lambda augmente le nombre d'instances d'environnement d'exécution en réponse. L'animation suivante suit le nombre de demandes simultanées dans le temps :




En figeant l'animation précédente à six points distincts dans le temps, nous obtenons le diagramme suivant :




Dans le diagramme précédent, nous pouvons tracer une ligne verticale à tout moment et compter le nombre d'environnements qui croisent cette ligne. Cela nous donne le nombre de demandes simultanées à ce moment précis. Par exemple, au moment `t1`, il y a trois environnements actifs servant trois demandes simultanées. Le nombre maximum de demandes simultanées dans cette simulation se produit au moment `t4`, lorsqu'il y a six environnements actifs servant six demandes simultanées.

En résumé, la simultanéité de votre fonction est le nombre de demandes simultanées qu'elle traite en même temps. En réponse à une augmentation de la simultanéité de votre fonction, Lambda fournit plus d'instances d'environnement d'exécution pour répondre à la demande.

## Calcul de la simultanéité d'une fonction

En général, la simultanéité d'un système est la capacité de traiter plus d'une tâche simultanément. Dans Lambda, la simultanéité est le nombre de demandes en vol que votre fonction traite en même temps. Une façon rapide et pratique de mesurer la simultanéité d'une fonction Lambda est d'utiliser la formule suivante :

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

La simultanéité diffère des demandes par seconde. Par exemple, supposons que votre fonction reçoive 100 demandes par seconde en moyenne. Si la durée moyenne des demandes est de une seconde, il est vrai que la simultanéité est également de 100 :

```
Concurrency = (100 requests/second) * (1 second/request) = 100
```

Toutefois, si la durée moyenne des demandes est de 500 ms, la simultanéité est de 50 :

```
Concurrency = (100 requests/second) * (0.5 second/request) = 50
```

Que signifie une simultanéité de 50 dans la pratique ? Si la durée moyenne des demandes est de 500 ms, vous pouvez considérer qu'une instance de votre fonction est capable de traiter deux demandes par seconde. Ensuite, il faut 50 instances de votre fonction pour gérer une charge de 100 demandes par seconde. Une simultanéité de 50 signifie que Lambda doit fournir 50 instances d'environnement d'exécution pour gérer efficacement cette charge de travail sans être limité. Voici comment exprimer cela sous forme d'équation :

```
Concurrency = (100 requests/second) / (2 requests/second) = 50
```

Si votre fonction reçoit le double de demandes (200 demandes par seconde), mais ne nécessite que la moitié du temps pour traiter chaque demande (250 ms), la simultanéité est toujours de 50 :

```
Concurrency = (200 requests/second) * (0.25 second/request) = 50
```

## Testez de votre compréhension de la simultanéité

Supposons que vous ayez une fonction qui prend, en moyenne, 200 ms pour s'exécuter. Pendant les pics de charge, vous observez 5 000 demandes par seconde. Quelle est la simultanéité de votre fonction pendant la charge de pointe ?

Réponse

La durée moyenne de la fonction est de 200 ms, soit 0,20 seconde. En utilisant la formule de simultanéité, vous pouvez entrer les chiffres pour obtenir une simultanéité de 1000 :

```
Concurrency = (5,000 requests/second) * (0.2 seconds/request) = 1,000
```

Autrement dit, une durée moyenne de fonction de 200 ms signifie que votre fonction peut traiter 5 demandes par seconde. Pour traiter la charge de travail de 5 000 demandes par seconde, vous avez besoin de 1 000 instances d'environnement d'exécution. La simultanéité est donc de 1 000 :

```
Concurrency = (5,000 requests/second) / (5 requests/second) = 1,000
```

# Comprendre la simultanéité réservée et la simultanéité provisionnée

Par défaut, votre compte dispose d'une limite de simultanéité de 1 000 exécutions simultanées pour toutes les fonctions d'une Région. Vos fonctions partagent ce groupe de 1 000 simultanéités sur une base à la demande. Vos fonctions sont limitées (c.-à-d. qu'elles commencent à abandonner des demandes) si vous n'avez plus de simultanéité disponible.

Certaines de vos fonctions peuvent être plus critiques que d'autres. Par conséquent, vous pouvez vouloir configurer les paramètres de simultanéité pour vous assurer que les fonctions critiques obtiennent la simultanéité dont elles ont besoin. Il existe deux types de contrôles de simultanéité : la simultanéité réservée et la simultanéité provisionnée.

- Utilisez la simultanéité réservée pour réserver une partie de la simultanéité de votre compte pour une fonction. Ceci est utile si vous ne voulez pas que d'autres fonctions prennent toute la simultanéité non réservée disponible.
- Utilisez la simultanéité provisionnée pour pré-initialiser un certain nombre d'instances d'environnement pour une fonction. Ceci est utile pour réduire les latences de démarrage à froid.

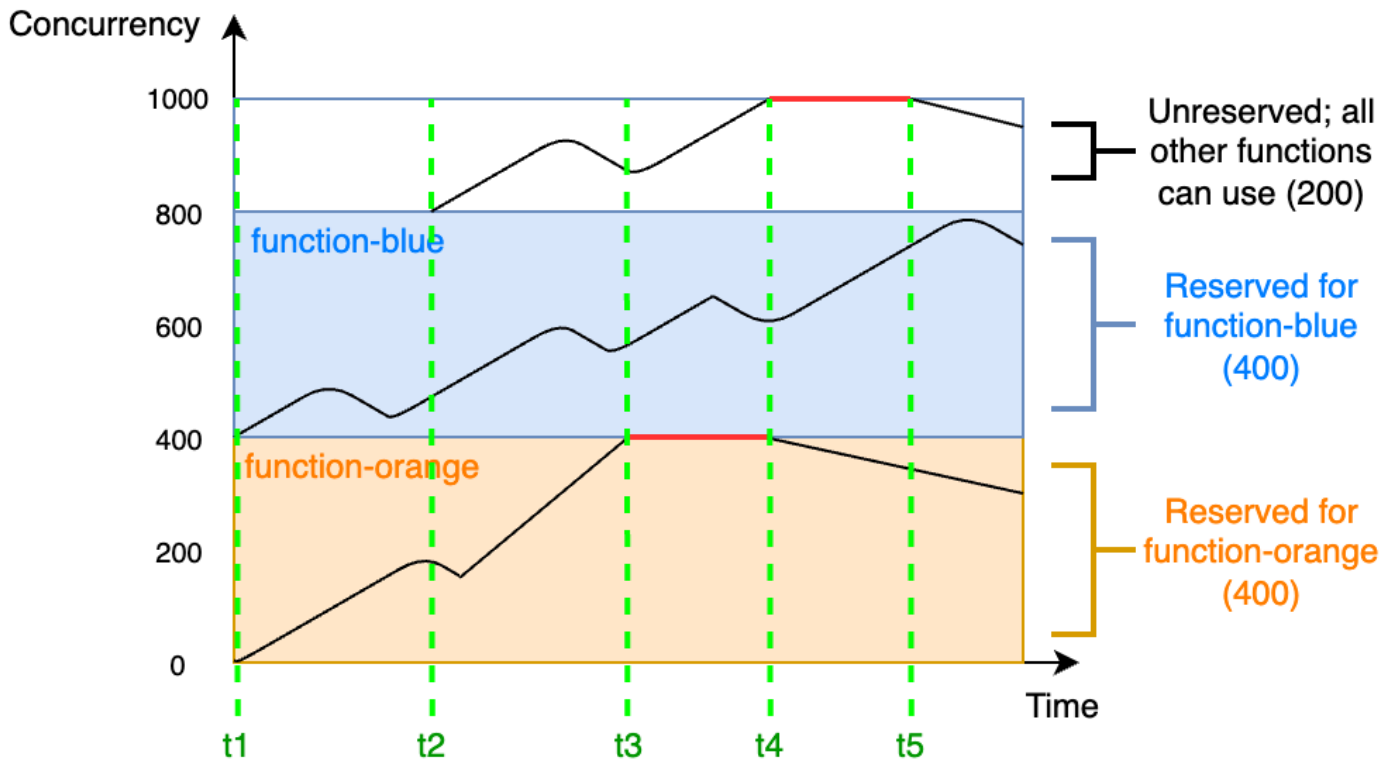
## Simultanéité réservée

Si vous voulez garantir qu'une certaine quantité de simultanéité est disponible pour votre fonction à tout moment, utilisez la simultanéité réservée.

La simultanéité réservée est le nombre maximum d'instances simultanées que vous voulez allouer à votre fonction. Lorsque vous consacrez une simultanéité réservée à une fonction, aucune autre fonction ne peut utiliser cette simultanéité. En d'autres termes, la définition de la simultanéité réservée peut avoir un impact sur le groupe de simultanéité disponible pour les autres fonctions. Les fonctions qui n'ont pas de simultanéité réservée partagent le groupe restant de simultanéité non réservée.

La configuration de la simultanéité réservée compte dans la limite globale de simultanéité de votre compte. Il n'y a pas de frais pour la configuration de la simultanéité réservée pour une fonction.

Pour mieux comprendre la simultanéité réservée, considérez le diagramme suivant :




Dans ce diagramme, la limite de simultanéité de votre compte pour toutes les fonctions de cette Région est à la limite par défaut de 1 000. Supposons que vous ayez deux fonctions critiques, `function-blue` et `function-orange`, qui s'attendent régulièrement à recevoir des volumes d'invocations élevés. Vous décidez d'accorder 400 unités de simultanéité réservée à `function-blue`, et 400 unités de simultanéité réservée à `function-orange`. Dans cet exemple, toutes les autres fonctions de votre compte doivent se partager les 200 unités restantes de simultanéité non réservée.

Le diagramme présente cinq points d'intérêt :

- À `t1`, `function-orange` et `function-blue` commencent à recevoir des demandes. Chaque fonction commence à utiliser sa portion d'unités de simultanéité réservées.
- À `t2`, `function-orange` et `function-blue` reçoivent de plus en plus de demandes. Au même moment, vous déployez d'autres fonctions Lambda, qui commencent à recevoir des demandes.

Vous n'allouez pas de simultanéité réservée à ces autres fonctions. Elles commencent à utiliser les 200 unités restantes de simultanéité non réservée.

- À `t3`, `function-orange` atteint la simultanéité maximale de 400. Bien qu'il existe une simultanéité non utilisée ailleurs dans le compte, `function-orange` ne peut pas y accéder. La ligne rouge indique que `function-orange` est limité et que Lambda peut abandonner des demandes.
- À `t4`, `function-orange` commence à recevoir moins de demandes et n'est plus limitée. Cependant, vos autres fonctions subissent un pic de trafic et commencent à être limitées. Bien qu'il y ait de la simultanéité inutilisée ailleurs dans votre compte, ces autres fonctions ne peuvent pas y accéder. La ligne rouge indique que vos autres fonctions sont limitées.
- À `t5`, les autres fonctions commencent à recevoir moins de demandes et ne sont plus limitées.

À partir de cet exemple, remarquez que la réservation de la simultanéité a les effets suivants :

- Votre fonction peut évoluer indépendamment des autres fonctions de votre compte. Toutes les fonctions de votre compte dans la même Région qui n'ont pas de simultanéité réservée partagent le groupe de simultanéité non réservée. Sans simultanéité réservée, d'autres fonctions peuvent utiliser toute votre simultanéité disponible. Cela empêche les fonctions critiques d'augmenter d'échelle si nécessaire.
- Votre fonction ne peut pas monter en puissance de façon incontrôlée. La simultanéité réservée limite la simultanéité maximale de votre fonction. Cela signifie que votre fonction ne peut pas utiliser la simultanéité réservée à d'autres fonctions, ou la simultanéité du groupe non réservé. Vous pouvez réserver de la simultanéité pour empêcher votre fonction d'utiliser toute la simultanéité disponible dans votre compte, ou de surcharger les ressources en aval.
- Il se peut que vous ne puissiez pas utiliser toute la simultanéité disponible de votre compte. La réservation de la simultanéité compte dans la limite de simultanéité de votre compte, mais cela signifie également que les autres fonctions ne peuvent pas utiliser cette partie de la simultanéité réservée. Si votre fonction n'utilise pas toute la simultanéité que vous lui réservez, vous gaspillez effectivement cette simultanéité. Ce n'est pas un problème, sauf si d'autres fonctions de votre compte peuvent bénéficier de la simultanéité gaspillée.

Pour gérer les paramètres de simultanéité réservée pour vos fonctions, consultez Configuration de la simultanéité réservée pour une fonction.

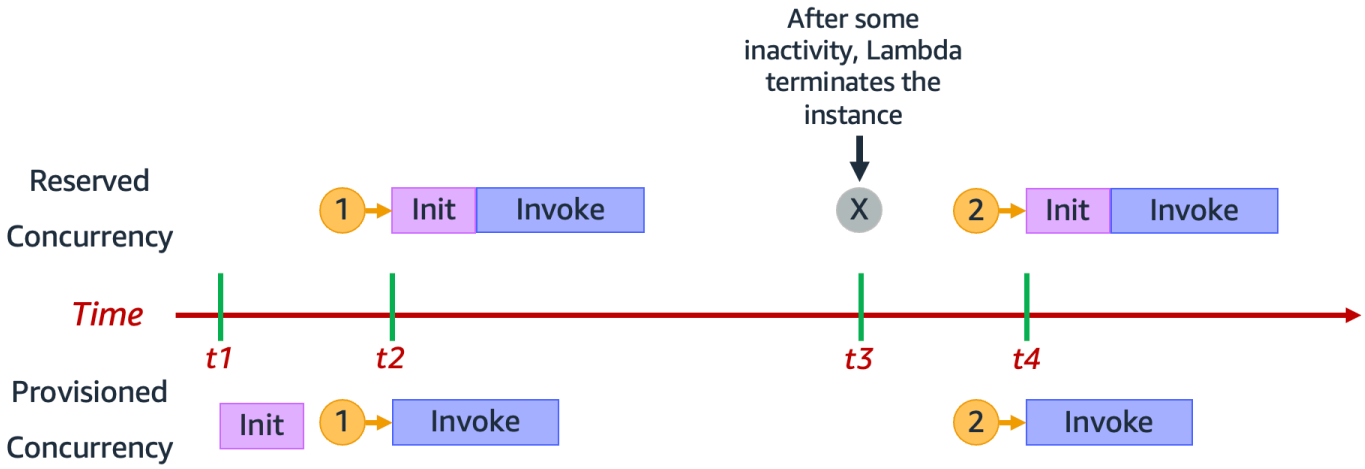
# Simultanéité allouée

Vous utilisez la simultanéité réservée pour définir le nombre maximal d'environnements d'exécution réservés à une fonction Lambda. Toutefois, aucun de ces environnements n'est préinitialisé. Par conséquent, les invocations de votre fonction peuvent prendre plus de temps, car Lambda doit d'abord initialiser le nouvel environnement avant de pouvoir l'utiliser pour invoquer votre fonction. Lorsque Lambda doit initialiser un nouvel environnement pour effectuer une invocation , on parle de démarrage à froid. Pour atténuer les démarrages à froid, vous pouvez utiliser la simultanéité provisionnée.

La simultanéité provisionnée est le nombre d'environnements d'exécution pré-initialisés que vous voulez allouer à votre fonction. Si vous définissez la simultanéité provisionnée sur une fonction, Lambda initialise ce nombre d'environnements d'exécution afin qu'ils soient prêts à répondre immédiatement aux demandes de la fonction.

> **Note**
>
> L'utilisation de la simultanéité provisionnée entraîne des frais supplémentaires sur votre compte. Si vous travaillez avec les environnements d'exécution Java 11 ou Java 17, vous pouvez également utiliser SnapStart Lambda pour atténuer les problèmes de démarrage à froid sans frais supplémentaires. SnapStart utilise des instantanés mis en cache de votre environnement d'exécution pour améliorer de manière significative les performances de démarrage. Vous ne pouvez pas utiliser les deux SnapStart et la simultanéité provisionnée sur la même version de fonction. Pour plus d'informations sur les SnapStart fonctionnalités, les limitations et les régions prises en charge, consultezAméliorer les performances de démarrage avec Lambda SnapStart.

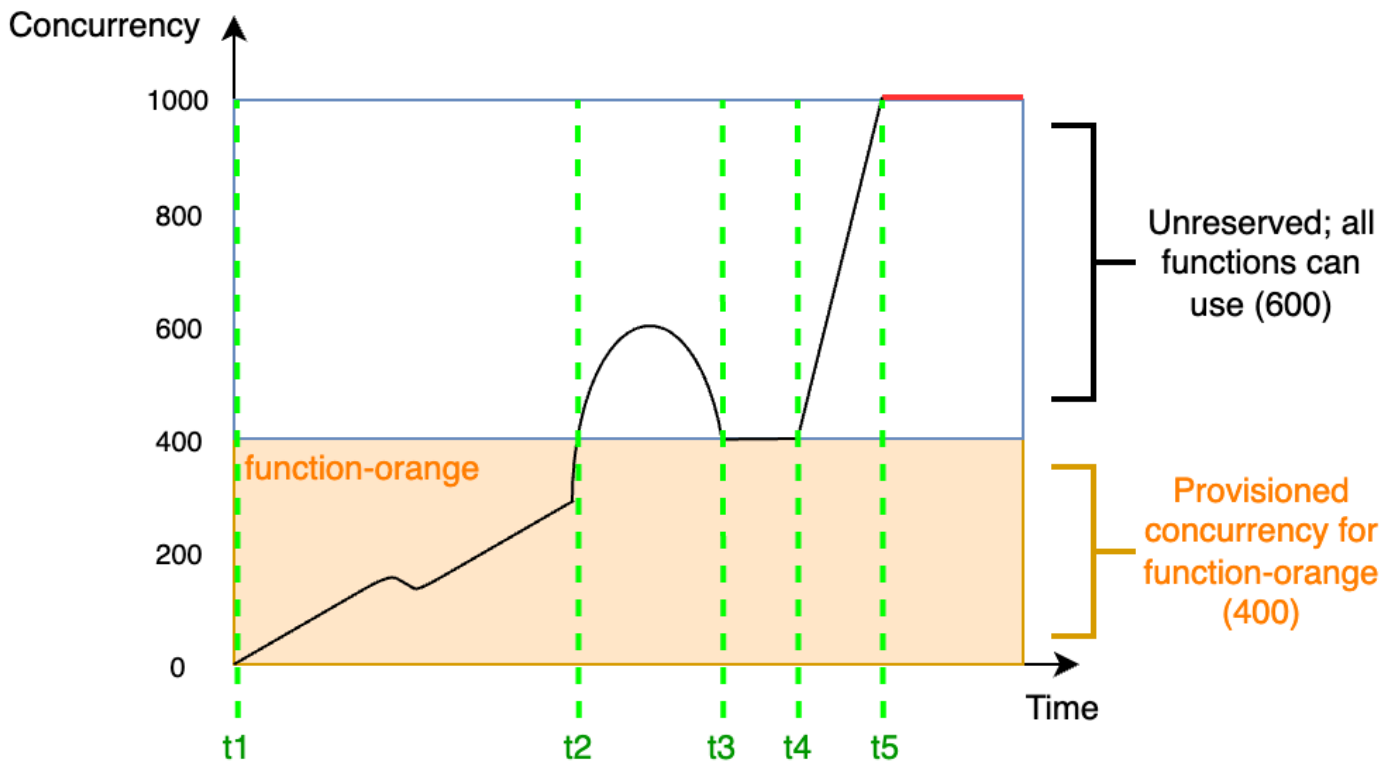
Lorsque vous utilisez la simultanéité provisionnée, Lambda recycle toujours les environnements d'exécution en arrière-plan. Cependant, à tout moment, Lambda s'assure toujours que le nombre d'environnements pré-initialisés est égal à la valeur du paramètre de simultanéité provisionnée de votre fonction. Ce comportement diffère de la simultanéité réservée, où Lambda peut résilier complètement un environnement après une période d'inactivité. Le diagramme suivant illustre cela en comparant le cycle de vie d'un environnement d'exécution unique lorsque vous configurez votre fonction en utilisant la simultanéité réservée par rapport à la simultanéité provisionnée.




Le diagramme présente quatre points d'intérêt :

| Heure | Simultanéité réservée | Simultanéité allouée |
|---|---|---|
| t1 | Rien ne se passe. | Lambda pré-initialise une instance d'environnement d'exécution. |
| t2 | La demande 1 arrive. Lambda doit initialiser une nouvelle instance d'environnement d'exécution. | La demande 1 arrive. Lambda utilise l'instance d'environ nement pré-initialisée. |
| t3 | Après un certain temps d'inactivité, Lambda met fin à l'instance d'environnement active. | Rien ne se passe. |
| t4 | La demande 2 arrive. Lambda doit initialiser une nouvelle instance d'environnement d'exécution. | La demande 2 arrive. Lambda utilise l'instance d'environ nement pré-initialisée. |

Pour mieux comprendre la simultanéité provisionnée, considérez le diagramme suivant :



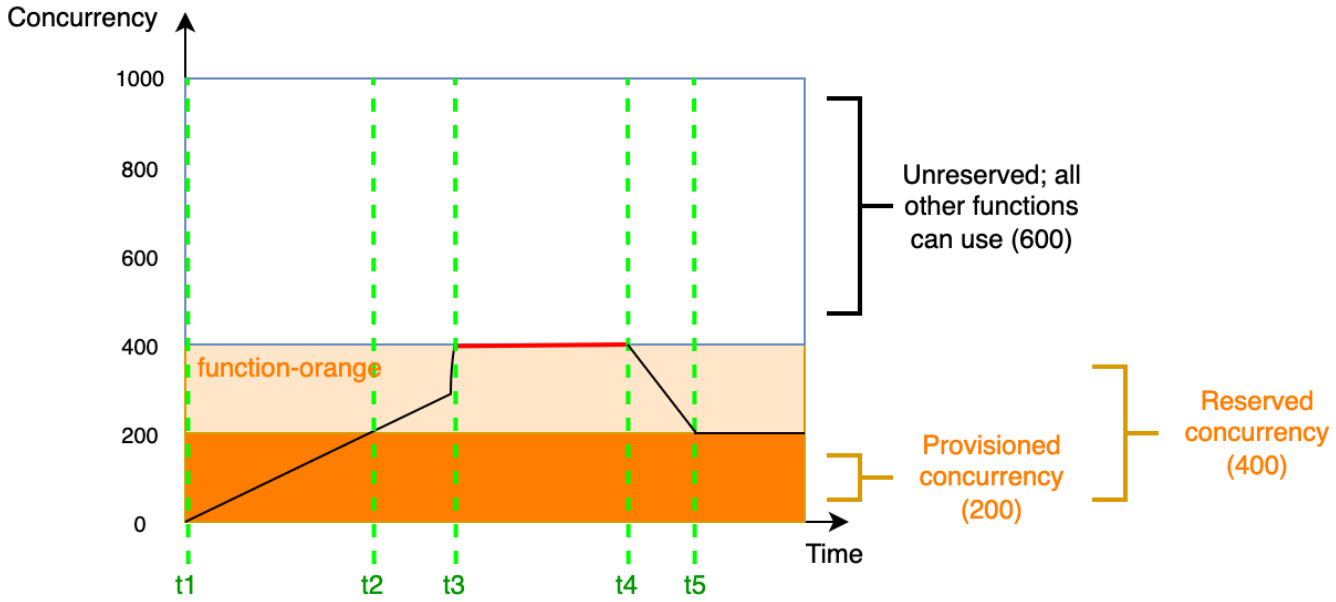

Dans ce diagramme, vous avez une limite de simultanéité de compte de 1 000. Vous décidez d'accorder 400 unités de simultanéité provisionnée à `function-orange`. Toutes les fonctions de votre compte, y compris `function-orange`, peuvent utiliser les 600 unités restantes de simultanéité non réservée.

Le diagramme présente cinq points d'intérêt :

- À `t1`, `function-orange` commence à recevoir des demandes. Puisque Lambda a pré-initialisé 400 instances d'environnement d'exécution, `function-orange` est prête pour une invocation immédiat.
- À `t2`, `function-orange` atteint 400 demandes simultanées. Par conséquent, `function-orange` n'a plus de simultanéité provisionnée. Cependant, comme il reste de la simultanéité non réservée, Lambda peut l'utiliser pour traiter des demandes supplémentaires vers `function-orange` (il n'y a pas de limitation). Lambda doit créer de nouvelles instances pour répondre à ces demandes, et votre fonction peut subir des latences de démarrage à froid.
- À `t3`, `function-orange` revient à 400 demandes simultanées après un bref pic de trafic. Lambda est à nouveau capable de traiter toutes les demandes sans latence de démarrage à froid.

- À `t4`, les fonctions de votre compte subissent un pic de trafic. Cette rafale peut provenir de `function-orange` ou de toute autre fonction de votre compte. Lambda utilise la simultanéité sans réserve pour traiter ces demandes.
- À `t5`, les fonctions de votre compte atteignent la limite maximale de simultanéité de 1 000 et sont limitées.

L'exemple précédent ne prenait en compte que la simultanéité provisionnée. En pratique, vous pouvez définir à la fois la simultanéité provisionnée et la simultanéité réservée sur une fonction. Vous pourriez le faire si vous aviez une fonction qui gère une charge constante d'invocations le week-end, mais qui connaît régulièrement des pics de trafic le week-end. Dans ce cas, vous pourriez utiliser la simultanéité provisionnée pour définir une quantité de base d'environnements pour traiter les demandes pendant les jours de la semaine, et utiliser la simultanéité réservée pour gérer les pics de trafic du week-end. Considérez le diagramme suivant :




Dans ce diagramme, supposons que vous configurez 200 unités de simultanéité provisionnée et 400 unités de simultanéité réservée pour `function-orange`. Parce que vous avez configuré la simultanéité réservée, `function-orange` ne peut utiliser aucune des 600 unités de simultanéité non réservée.

Ce diagramme comporte cinq points d'intérêt :

- À `t1`, `function-orange` commence à recevoir des demandes. Puisque Lambda a pré-initialisé 200 instances d'environnement d'exécution, `function-orange` est prête pour une invocation immédiat.
- À `t2`, `function-orange` utilise toute sa simultanéité provisionnée. `function-orange` peut continuer à servir les demandes en utilisant la simultanéité réservée, mais ces demandes peuvent subir des latences de démarrage à froid.
- À `t3`, `function-orange` atteint 400 demandes simultanées. Par conséquent, `function-orange` utilise toute sa simultanéité réservée. Comme `function-orange` ne peut pas utiliser la simultanéité non réservée, les demandes commencent à être limitées.
- À `t4`, `function-orange` commence à recevoir moins de demandes et ne se limite plus.
- À `t5`, `function-orange` tombe à 200 demandes simultanées, de sorte que toutes les demandes peuvent à nouveau utiliser la simultanéité provisionnée (c.-à-d. sans latence de démarrage à froid).

La simultanéité réservée et la simultanéité provisionnée comptent toutes deux dans la limite de simultanéité de votre compte et dans les quotas régionaux. En d'autres termes, l'allocation de la simultanéité réservée et provisionnée peut avoir un impact sur le groupe de simultanéité disponible pour les autres fonctions. La configuration de la simultanéité provisionnée entraîne des frais pour votre. Compte AWS

> **Note**
>
> Si la simultanéité provisionnée sur les versions et alias d'une fonction s'ajoute à la simultanéité réservée de la fonction, toutes les invocations s'exécutent sur la simultanéité provisionnée. Cette configuration a également pour effet de limiter la version non publiée de la fonction (`$LATEST`), ce qui empêche son exécution. Vous ne pouvez pas allouer plus de simultanéité provisionnée que de simultanéité réservée pour une fonction.

Pour gérer les paramètres de simultanéité provisionnée pour vos fonctions, consultez Configuration de la simultanéité provisionnée pour une fonction. Pour automatiser la mise à l'échelle de la concurrence provisionnée en fonction d'une planification ou de l'utilisation de l'application, consultez Utilisation d'Application Auto Scaling pour automatiser la gestion de la simultanéité provisionnée.

## Comment Lambda alloue la simultanéité provisionnée

La simultanéité provisionnée n’est pas mise en ligne immédiatement après la configuration. Lambda commence à allouer la simultanéité approvisionnée après une phase de préparation d’une ou deux minutes. Pour chaque fonction, Lambda peut provisionner jusqu'à 6 000 environnements d'exécution par minute, quel que soit le cas. Région AWS C'est exactement le même que le taux de mise à l'échelle de la simultanéité pour les fonctions.

Lorsque vous soumettez une demande d’allocation de simultanéité provisionnée, vous ne pouvez accéder à aucun de ces environnements tant que Lambda n’a pas terminé de les allouer. Par exemple, si vous demandez une simultanéité provisionnée de 5 000, aucune de vos demandes ne peut utiliser la simultanéité provisionnée tant que Lambda n'a pas complètement alloué les 5 000 environnements d'exécution.

## Comparaison de la simultanéité réservée et de la simultanéité provisionnée

Le tableau suivant résume et compare la simultanéité réservée et provisionnée.

| Rubrique | Simultanéité réservée | Simultanéité allouée |
|---|---|---|
| Définition | Nombre maximal d’instances d’environnement d’exécution pour votre fonction. | Nombre défini d’instances d’environnement d’exécution pré-provisionnées pour votre fonction. |
| Comportement de provisionnement | Lambda provisionne de nouvelles instances sur une base à la demande. | Lambda pré-provisionne les instances (c.-à-d. avant que votre fonction ne commence à recevoir des demandes). |
| Comportement de démarrage à froid | Latence de démarrage à froid possible, puisque Lambda doit créer de nouvelles instances à la demande. | Latence de démarrage à froid impossible, puisque Lambda ne doit pas créer d’instances à la demande. |
| Comportement de limitation | La fonction est limitée lorsque la limite de simultanéité réservée est atteinte. | Si la simultanéité réservée n’est pas définie : la fonction utilise la simultanéité non |

| Rubrique | Simultanéité réservée | Simultanéité allouée |
| --- | --- | --- |
| | | réservée lorsque la limite de simultanéité provisionnée est atteinte.<br><br>Si la simultanéité réservée est définie : la fonction est limitée lorsque la limite de simultané ité réservée est atteinte. |
| Comportement par défaut si non défini | La fonction utilise la simultané ité non réservée disponible dans votre compte. | Lambda ne pré-provisionne pas d'instances. Au lieu de cela, si la simultanéité réservée n'est pas définie : la fonction utilise la simultanéité non réservée disponible dans votre compte.<br><br>Si la simultanéité réservée est définie : la fonction utilise la simultanéité réservée. |
| Tarification | Pas de frais supplémentaires. | Entraîne des frais supplémen taires. |

# Comprendre la simultanéité et le nombre de demandes par seconde

Comme indiqué dans la section précédente, la simultanéité diffère des demandes par seconde. Cette distinction est particulièrement importante lorsque l'on travaille avec des fonctions dont la durée moyenne des requêtes est inférieure à 100 ms.

Pour toutes les fonctions de votre compte, Lambda applique une limite de demandes par seconde égale à 10 fois le nombre de demandes simultanées de votre compte. Par exemple, étant donné que la limite de simultanéité par défaut est de 1 000, les fonctions de votre compte peuvent traiter un maximum de 10 000 demandes par seconde.

Prenons l'exemple d'une fonction dont la durée moyenne des demandes est de 50 ms. À 20 000 requêtes par seconde, voici la simultanéité de cette fonction :

```
Concurrency = (20,000 requests/second) * (0.05 second/request) = 1,000
```

Sur la base de ce résultat, vous pouvez vous attendre à ce que la limite de simultanéité des comptes de 1 000 soit suffisante pour gérer cette charge. Toutefois, en raison de la limite de 10 000 demandes par seconde, votre fonction ne peut traiter que 10 000 demandes par seconde sur un total de 20 000 demandes. Cette fonction est soumise à une limitation.

La leçon à tirer est que vous devez tenir compte à la fois de la simultanéité et des demandes par seconde lorsque vous configurez les paramètres de simultanéité pour vos fonctions. Dans ce cas, vous devez demander une augmentation de la limite de simultanéité des comptes à 2 000, car cela porterait le nombre total de demandes par seconde à 20 000.

> **Note**
>
> Sur la base de cette limite de requêtes par seconde, il est faux de dire que chaque environnement d'exécution Lambda ne peut traiter qu'un maximum de 10 requêtes par seconde. Au lieu d'observer la charge sur un environnement d'exécution individuel, Lambda prend uniquement en compte la simultanéité globale et le nombre total de demandes par seconde lors du calcul de vos quotas.

## Testez votre compréhension de la concurrence (fonctions inférieures à 100 ms)

Supposons que vous ayez une fonction qui prend, en moyenne, 20 ms pour s'exécuter. Pendant les pics de charge, vous observez 30 000 demandes par seconde. Quelle est la simultanéité de votre fonction pendant la charge de pointe ?

Réponse

La durée moyenne de la fonction est de 20 ms, soit 0,02 seconde. À l'aide de la formule de simultanéité, vous pouvez saisir les chiffres pour obtenir une simultanéité de 600 :

```
Concurrency = (30,000 requests/second) * (0.02 seconds/request) = 600
```

Par défaut, la limite de simultanéité des comptes de 1 000 semble suffisante pour gérer cette charge. Cependant, la limite de 10 000 demandes par seconde n'est pas suffisante pour traiter les 30 000

demandes entrantes par seconde. Pour répondre pleinement aux 30 000 demandes, vous devez demander une augmentation de la limite de simultanéité des comptes à 3 000 ou plus.

La limite de demandes par seconde s'applique à tous les quotas de Lambda impliquant une simultanéité. En d'autres termes, cela s'applique aux fonctions synchrones à la demande, aux fonctions qui utilisent la simultanéité provisionnée et au comportement de dimensionnement de la simultanéité. Par exemple, voici quelques scénarios dans lesquels vous devez examiner attentivement vos limites de simultanéité et de demandes par seconde :

- Une fonction utilisant la simultanéité à la demande peut connaître une augmentation de 500 demandes simultanées toutes les 10 secondes, ou de 5 000 demandes par seconde toutes les 10 secondes, selon la première éventualité.
- Supposons que vous disposiez d'une fonction dotée d'une allocation de simultanéité provisionnée de 10. Cette fonction se transforme en simultanéité à la demande après 10 demandes simultanées ou 100 demandes par seconde, selon la première éventualité.

## Quotas de simultanéité

Lambda définit des quotas pour la quantité totale de simultanéité que vous pouvez utiliser sur toutes les fonctions d'une Région. Ces quotas existent à deux niveaux :

- Au niveau du compte, vos fonctions peuvent avoir jusqu'à 1 000 unités de simultanéité par défaut. Pour augmenter cette limite, consultez Demande d'augmentation de quota dans le Guide de l'utilisateur Service Quotas.
- Au niveau de la fonction, vous pouvez réserver par défaut jusqu'à 900 unités de simultanéité pour l'ensemble de vos fonctions. Quelle que soit la limite totale de simultanéité de votre compte, Lambda réserve toujours 100 unités de simultanéité à vos fonctions qui ne réservent pas explicitement la simultanéité. Par exemple, si vous avez augmenté la limite de simultanéité de votre compte à 2 000, vous pouvez réserver jusqu'à 1 900 unités de simultanéité au niveau de la fonction.
- Au niveau du compte comme au niveau de la fonction, Lambda impose également une limite de demandes par seconde égale à 10 fois le quota de simultanéité correspondant. Cela s'applique par exemple à la simultanéité au niveau du compte, aux fonctions utilisant la simultanéité à la demande, aux fonctions utilisant la simultanéité provisionnée et au comportement de mise à l'échelle de la simultanéité. Pour de plus amples informations, veuillez consulter the section called "Comprendre la simultanéité et le nombre de demandes par seconde".

Pour vérifier le quota de simultanéité actuel de votre compte, utilisez le AWS Command Line Interface (AWS CLI) pour exécuter la commande suivante :

```
aws lambda get-account-settings
```

Vous devriez voir une sortie semblable à la suivante :

```
{
    "AccountLimit": {
        "TotalCodeSize": 80530636800,
        "CodeSizeUnzipped": 262144000,
        "CodeSizeZipped": 52428800,
        "ConcurrentExecutions": 1000,
        "UnreservedConcurrentExecutions": 900
    },
    "AccountUsage": {
        "TotalCodeSize": 410759889,
        "FunctionCount": 8
    }
}
```

`ConcurrentExecutions` est le quota total de simultanéité au niveau de votre compte. `UnreservedConcurrentExecutions` est la quantité de concurrence réservée que vous pouvez encore allouer à vos fonctions.

Au fur et à mesure que votre fonction reçoit des demandes, Lambda augmente automatiquement le nombre d'environnements d'exécution pour traiter ces demandes jusqu'à ce que votre compte atteigne sa limite de simultanéité. Toutefois, pour éviter une mise à l'échelle excessive en réponse à des pics de trafic soudains, Lambda limite la rapidité avec laquelle vos fonctions peuvent être mises à l'échelle. Ce taux de mise à l'échelle simultanée est le taux maximal auquel les fonctions de votre compte peuvent évoluer en réponse à l'augmentation du nombre de demandes. (C'est-à-dire la rapidité avec laquelle Lambda peut créer de nouveaux environnements d'exécution.) Le taux d'échelonnement de la simultanéité est différent de la limite de simultanéité au niveau du compte, qui est le montant total de simultanéité disponible pour vos fonctions.

Dans chacune et pour chaque fonction Région AWS, votre taux de mise à l'échelle de la simultanéité est de 1 000 instances d'environnement d'exécution toutes les 10 secondes (ou 10 000 requêtes par seconde toutes les 10 secondes). En d'autres termes, toutes les 10 secondes, Lambda peut allouer au maximum 1 000 instances d'environnement d'exécution supplémentaires, ou répondre à 10 000 requêtes supplémentaires par seconde, à chacune de vos fonctions.

En général, il n’est pas nécessaire de se soucier de cette limitation. La vitesse de mise à l’échelle de Lambda est suffisante dans la plupart des cas d’utilisation.

Il est important de noter que le taux de mise à l'échelle de la simultanéité est une limite au niveau de la fonction. Cela signifie que chaque fonction de votre compte peut être mise à l’échelle indépendamment des autres fonctions.

Pour plus d’informations sur les comportements de mise à l’échelle, consultez Comportement de mise à l’échelle Lambda.

# Configuration de la simultanéité réservée pour une fonction

Dans Lambda, la simultanéité est le nombre de demandes en vol que votre fonction traite en même temps. Il existe deux types de contrôles de la simultanéité disponibles :

- La simultanéité réservée représente le nombre maximum d'instances simultanées allouées à votre fonction. Lorsqu'une fonction dispose de la simultanéité réservée, aucune autre fonction ne peut utiliser cette simultanéité. La simultanéité réservée est utile pour garantir que vos fonctions les plus critiques disposent toujours d'une simultanéité suffisante pour traiter les demandes entrantes. Il n'y a pas de frais supplémentaires pour la configuration de la simultanéité réservée pour une fonction.
- La simultanéité provisionnée est le nombre d'environnements d'exécution pré-initialisés alloués à votre fonction. Ces environnements d'exécution sont prêts à répondre immédiatement aux demandes de fonctions entrantes. La simultanéité provisionnée est utile pour réduire les latences de démarrage à froid des fonctions. La configuration de la simultanéité provisionnée entraîne des frais supplémentaires pour votre. Compte AWS

Cette rubrique explique comment gérer et configurer la simultanéité réservée. Pour une présentation conceptuelle de ces deux types de contrôles de simultanéité, consultez Simultanéité réservée et simultanéité provisionnée. Pour plus d'informations sur la configuration de la simultanéité provisionnée, consultez the section called "Configuration de la simultanéité provisionnée".

> **Note**
>
> Les fonctions Lambda liées à un mappage des sources d'événements Amazon MQ ont une simultanéité maximale par défaut. Pour Apache Active MQ, le nombre maximum d'instances simultanées est de 5. Pour Rabbit MQ, le nombre maximum d'instances simultanées est de 1. La définition d'une simultanéité réservée ou provisionnée pour votre fonction ne modifie pas ces limites. Pour demander une augmentation de la simultanéité maximale par défaut lors de l'utilisation d'Amazon MQ, contactez AWS Support.

Sections


- Configuration de la simultanéité réservée
- Estimation précise de la simultanéité réservée requise pour une fonction

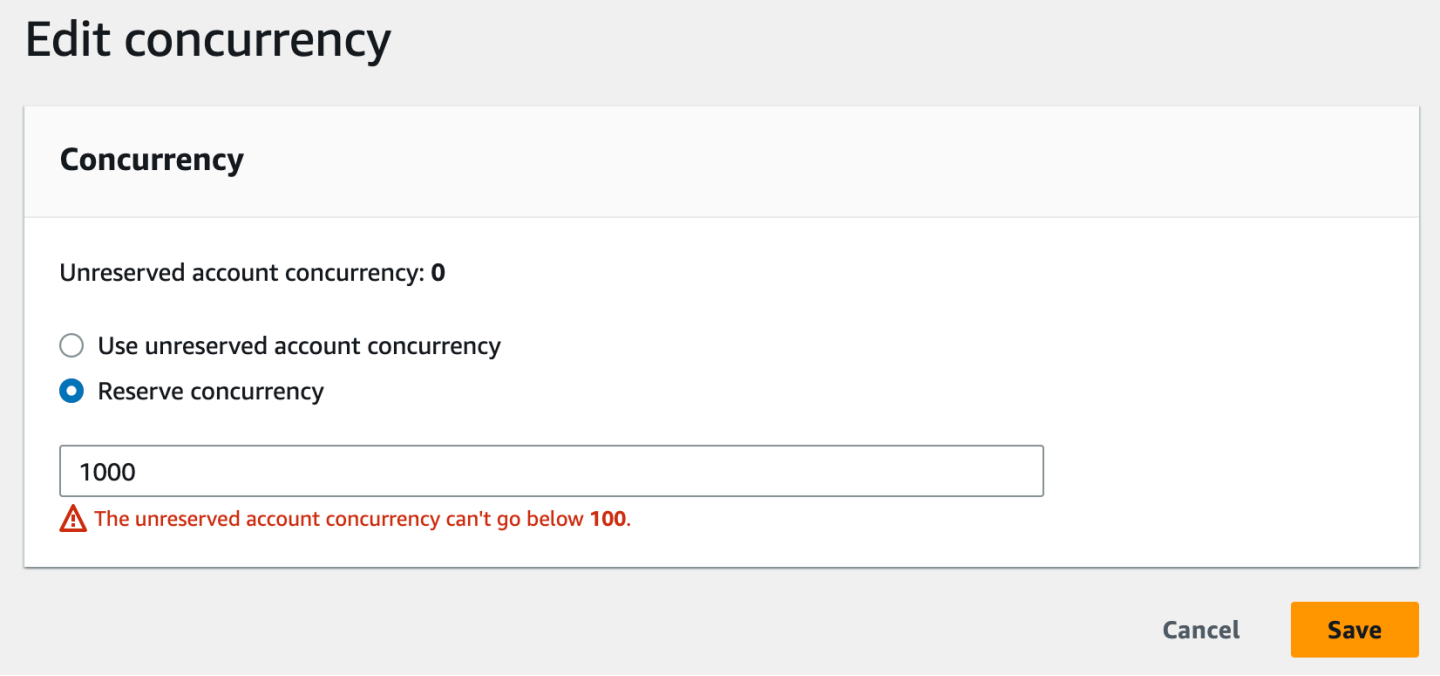
# Configuration de la simultanéité réservée

Vous pouvez configurer les paramètres de simultanéité réservés pour une fonction à l'aide de la console Lambda ou du Lambda. API

Réserver la simultanéité pour une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction pour laquelle vous souhaitez réserver la simultanéité.
3. Sélectionnez Configuration, puis Concurrency (Simultanéité).
4. Sous Concurrency (Simultanéité), choisissez Edit (Modifier).
5. Choisissez Reserve concurrency (Réserver la simultanéité). Saisissez la quantité de simultanéité à réserver pour la fonction.
6. Choisissez Save (Enregistrer).

Vous pouvez réserver jusqu'à la valeur de la simultanéité du compte non réservé moins 100. Les 100 unités de simultanéité restantes concernent les fonctions qui n'utilisent pas la simultanéité réservée. Par exemple, si votre compte a une limite de concurrence de 1 000, vous ne pouvez pas réserver les 1 000 unités de concurrence pour une seule fonction.




Le fait de réserver la simultanéité à une fonction peut avoir des conséquences sur le groupe de simultanéité disponible pour d'autres fonctions. Par exemple, si vous réservez 100 unités de

simultanéité pour `function-a`, les autres fonctions de votre compte doivent partager les 900 unités de simultanéité restantes, même si `function-a` n'utilisent pas les 100 unités de simultanéité réservées.

Pour limiter intentionnellement une fonction, définissez la simultanéité réservée à zéro. Cela empêche votre fonction de traiter des événements jusqu'à ce que vous supprimiez la limite.

Pour configurer la simultanéité réservée avec le API Lambda, utilisez les API opérations suivantes.

- PutFunctionConcurrency
- GetFunctionConcurrency
- DeleteFunctionConcurrency

Par exemple, pour configurer la simultanéité réservée avec le AWS Command Line Interface (CLI), utilisez la `put-function-concurrency` commande. La commande suivante réserve 100 unités de simultanéité pour une fonction nommée `my-function` :

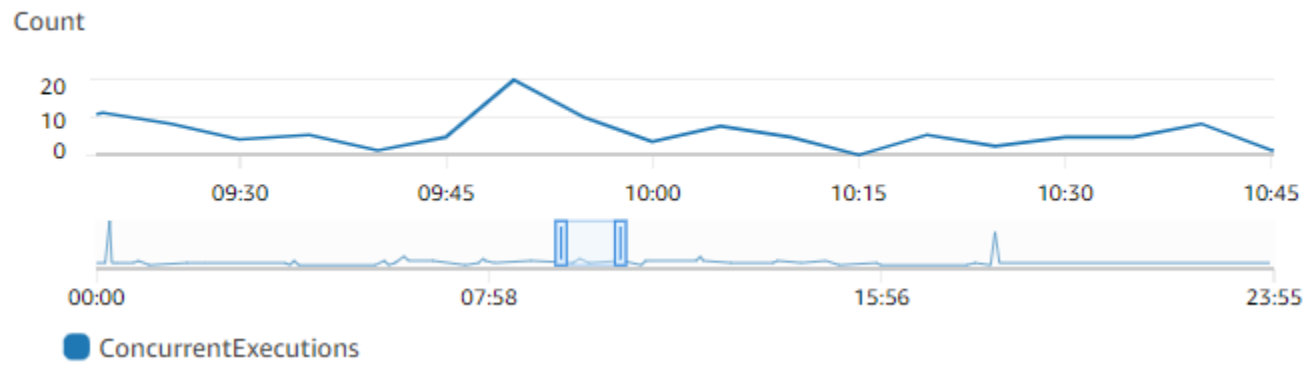
```
aws lambda put-function-concurrency --function-name my-function \
    --reserved-concurrent-executions 100
```

Vous devriez voir une sortie semblable à la suivante :

```
{
    "ReservedConcurrentExecutions": 100
}
```

## Estimation précise de la simultanéité réservée requise pour une fonction

Si votre fonction gère actuellement du trafic, vous pouvez facilement consulter ses indicateurs de simultanéité à l'aide de CloudWatch métriques. Plus précisément, la métrique `ConcurrentExecutions` vous montre le nombre d'invocations simultanées pour chaque fonction de votre compte.




Le graphique précédent indique que cette fonction répond à une moyenne de 5 à 10 demandes simultanées à tout moment, et qu'elle atteint un pic de 20 demandes lors d'une journée typique. Supposons qu'il y ait beaucoup d'autres fonctions dans votre compte.  Si cette fonction est essentielle à votre application et que vous ne voulez pas perdre de demandes, utilisez un nombre supérieur ou égal à 20 comme paramètre de simultanéité réservé.

Par ailleurs, rappelez-vous que vous pouvez également calculer la simultanéité à l'aide de la formule suivante :

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

En multipliant le nombre moyen de demandes par seconde par la durée moyenne des demandes en secondes, vous obtenez une estimation approximative du niveau de simultanéité que vous devez réserver. Vous pouvez estimer les demandes moyennes par seconde à l'aide de la métrique `Invocation`, et la durée moyenne des demandes en secondes à l'aide de la métrique `Duration`. Pour plus d'informations, consultez Afficher les métriques des fonctions Lambda.

Vous devez également connaître vos contraintes de débit en amont et en aval. Bien que les fonctions Lambda se mettent à l'échelle parfaitement, les dépendances en amont et en aval peuvent avoir des mêmes capacités de débit différentes. Si vous devez limiter le niveau d'évolutivité de votre fonction, configurez la simultanéité réservée sur votre fonction.

# Configuration de la simultanéité provisionnée pour une fonction

Dans Lambda, la simultanéité est le nombre de demandes en vol que votre fonction traite en même temps. Il existe deux types de contrôles de la simultanéité disponibles :

- La simultanéité réservée représente le nombre maximum d'instances simultanées allouées à votre fonction. Lorsqu'une fonction dispose de la simultanéité réservée, aucune autre fonction ne peut utiliser cette simultanéité. La simultanéité réservée est utile pour garantir que vos fonctions les plus critiques disposent toujours d'une simultanéité suffisante pour traiter les demandes entrantes. Il n'y a pas de frais supplémentaires pour la configuration de la simultanéité réservée pour une fonction.
- La simultanéité provisionnée est le nombre d'environnements d'exécution pré-initialisés alloués à votre fonction. Ces environnements d'exécution sont prêts à répondre immédiatement aux demandes de fonctions entrantes. La simultanéité provisionnée est utile pour réduire les latences de démarrage à froid des fonctions. La configuration de la simultanéité provisionnée entraîne des frais supplémentaires pour votre. Compte AWS

Cette rubrique explique comment gérer et configurer la simultanéité provisionnée. Pour une présentation conceptuelle de ces deux types de contrôles de simultanéité, consultez Simultanéité réservée et simultanéité provisionnée. Pour plus d'informations sur la configuration de la simultanéité réservée, consultez the section called "Configuration de la simultanéité réservée".

> **Note**
>
> Les fonctions Lambda liées à un mappage des sources d'événements Amazon MQ ont une simultanéité maximale par défaut. Pour Apache Active MQ, le nombre maximum d'instances simultanées est de 5. Pour Rabbit MQ, le nombre maximum d'instances simultanées est de 1. La définition d'une simultanéité réservée ou provisionnée pour votre fonction ne modifie pas ces limites. Pour demander une augmentation de la simultanéité maximale par défaut lors de l'utilisation d'Amazon MQ, contactez AWS Support.

Sections


- Configuration de la simultanéité provisionnée
- Estimation précise de la simultanéité provisionnée requise pour une fonction
- Optimisation du code de fonction lors de l'utilisation de la simultanéité provisionnée

- Utilisation de variables d'environnement pour visualiser et contrôler le comportement de simultanéité provisionné
- Comprendre le comportement de journalisation et de facturation grâce à la simultanéité provisionnée
- Utilisation d'Application Auto Scaling pour automatiser la gestion de la simultanéité provisionnée

## Configuration de la simultanéité provisionnée

Vous pouvez configurer les paramètres de simultanéité provisionnés pour une fonction à l'aide de la console Lambda ou du Lambda. API

Pour allouer de la simultanéité provisionnée pour une fonction (console)

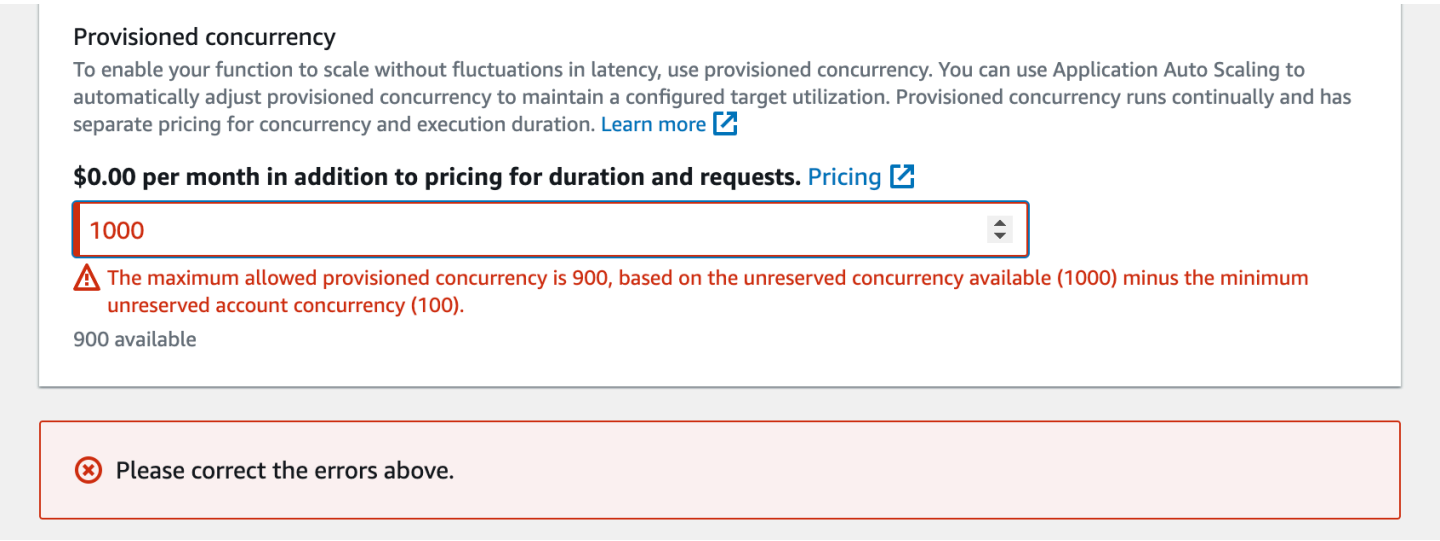
1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction pour laquelle vous souhaitez allouer de la simultanéité provisionnée.
3. Sélectionnez Configuration, puis Concurrency (Simultanéité).
4. Sous Provisioned concurrency configurations (Configurations de simultanéité provisionnée), sélectionnez Add configuration (Ajouter une configuration).
5. Choisissez le type de qualificateur, ainsi que l'alias ou la version.

> **Note**
>
> Vous ne pouvez pas utiliser la simultanéité provisionnée avec la LATEST version $ d'aucune fonction.
> Si votre fonction possède une source d'événement, assurez-vous que la source d'événement pointe vers le bon alias ou la bonne version de la fonction. Sinon, votre fonction n'utilisera pas les environnements de simultanéité provisionnés.

6. Saisissez un nombre sous Simultanéité provisionnée. Lambda fournit une estimation des coûts mensuels.
7. Choisissez Save (Enregistrer).

Vous pouvez configurer jusqu'à la simultanéité du compte non réservé dans votre compte, moins 100. Les 100 unités de simultanéité restantes concernent les fonctions qui n'utilisent pas la simultanéité réservée. Par exemple, si votre compte a une limite de simultanéité de 1 000 et que vous

n'avez pas attribué de simultanéité réservée ou provisionnée à l'une de vos autres fonctions, vous pouvez configurer un maximum de 900 unités de simultanéité provisionnées pour une seule fonction.




La configuration de la simultanéité provisionnée pour une fonction a des conséquences sur le groupe de simultanéité disponible pour d'autres fonctions. Par exemple, si vous configurez 100 unités de simultanéité provisionnée pour `function-a`, les autres fonctions de votre compte doivent partager les 900 unités de simultanéité restantes. Et ce même si `function-a` n'utilise pas les 100 unités.

Il est possible d'allouer à la fois de la simultanéité réservée et de la simultanéité provisionnée pour la même fonction. Dans de tels cas, la simultanéité provisionnée ne peut pas dépasser la simultanéité réservée.

Cette limite s'applique aux versions de fonctions. La simultanéité provisionnée maximale que vous pouvez allouer à une version de fonction spécifique est égale à la simultanéité réservée de la fonction moins la simultanéité provisionnée sur les autres versions de fonction.

Pour configurer la simultanéité provisionnée avec le LambdaAPI, utilisez les opérations suivantes. API

- PutProvisionedConcurrencyConfig
- GetProvisionedConcurrencyConfig
- ListProvisionedConcurrencyConfigs
- DeleteProvisionedConcurrencyConfig

Par exemple, pour configurer la simultanéité provisionnée avec le AWS Command Line Interface (CLI), utilisez la `put-provisioned-concurrency-config` commande. La commande suivante

alloue 100 unités de simultanéité provisionnée pour l'alias BLUE d'une fonction nommée `my-function` :

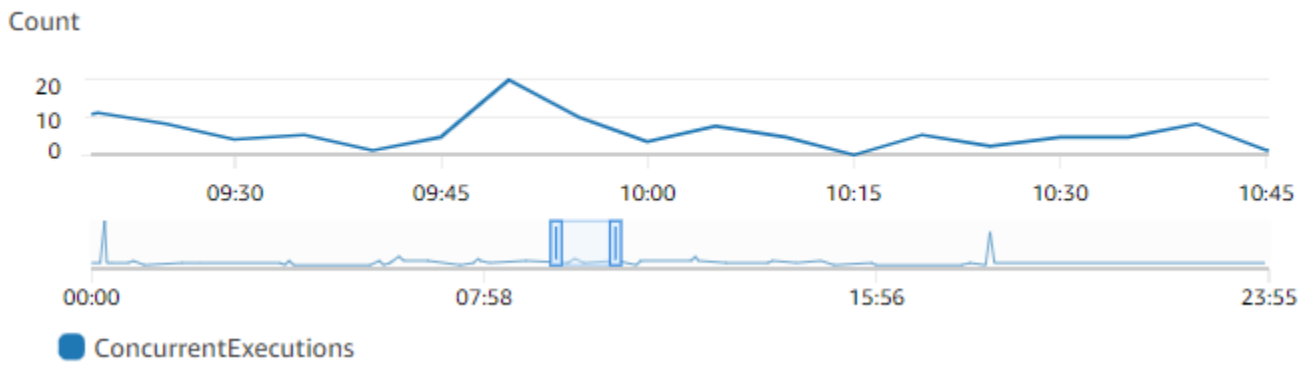
```
aws lambda put-provisioned-concurrency-config --function-name my-function \
  --qualifier BLUE \
  --provisioned-concurrent-executions 100
```

Vous devriez voir une sortie semblable à la suivante :

```
{
  "Requested ProvisionedConcurrentExecutions": 100,
  "Allocated ProvisionedConcurrentExecutions": 0,
  "Status": "IN_PROGRESS",
  "LastModified": "2023-01-21T11:30:00+0000"
}
```

## Estimation précise de la simultanéité provisionnée requise pour une fonction

Vous pouvez consulter les mesures de simultanéité de n'importe quelle fonction active à l'aide de CloudWatch métriques. Plus précisément, la métrique `ConcurrentExecutions` vous montre le nombre d'invocations simultanées pour les fonctions de votre compte.




Le graphique précédent indique que cette fonction répond à une moyenne de 5 à 10 demandes simultanées à tout moment, et qu'elle atteint un pic de 20 demandes. Supposons qu'il y ait beaucoup d'autres fonctions dans votre compte.  Si cette fonction est essentielle à votre application et que vous avez besoin d'une réponse à faible latence à chaque invocation, configurez au moins 20 unités de simultanéité provisionnée.

Rappelez-vous que vous pouvez également calculer la simultanéité à l'aide de la formule suivante :

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Pour estimer le niveau de simultanéité dont vous avez besoin, multipliez le nombre moyen de demandes par seconde par la durée moyenne des demandes en secondes. Vous pouvez estimer les demandes moyennes par seconde à l'aide de la métrique `Invocation`, et la durée moyenne des demandes en secondes à l'aide de la métrique `Duration`.

Lors de la configuration de la simultanéité provisionnée, Lambda suggère d'ajouter un tampon de 10 % en plus de la quantité de simultanéité dont votre fonction a généralement besoin. Par exemple, si votre fonction atteint habituellement un pic de 200 demandes simultanées, définissez votre simultanéité provisionnée à 220 (200 demandes simultanées + 10 % = 220 simultanéités provisionnées).

## Optimisation du code de fonction lors de l'utilisation de la simultanéité provisionnée

Si vous utilisez la simultanéité provisionnée, pensez à restructurer votre code de fonction pour optimiser la faible latence. Pour les fonctions utilisant la simultanéité provisionnée, Lambda exécute n'importe quel code d'initialisation, tel que le chargement de bibliothèques et l'instanciation de clients, pendant le temps d'allocation. Il est donc conseillé de déplacer un maximum d'initialisation en dehors du gestionnaire de la fonction principale pour éviter d'avoir un impact sur la latence lors des invocations à la fonction. En revanche, l'initialisation de bibliothèques ou l'instanciation de clients dans le code de votre gestionnaire principal signifie que votre fonction doit l'exécuter chaque fois qu'elle est invoquée (que vous utilisiez ou non la simultanéité provisionnée).

Pour les appels à la demande, Lambda peut avoir besoin de réexécuter votre code d'initialisation chaque fois que votre fonction démarre à froid. Pour de telles fonctions, vous pouvez choisir de différer l'initialisation d'une fonctionnalité spécifique jusqu'à ce que la fonction ait besoin d'elle. Par exemple, prenons le flux de contrôle suivant pour un gestionnaire Lambda :

```
def handler(event, context):
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
    else:
        // Do nothing
```

Dans l'exemple précédent, au lieu d'initialiser `CLIENT_A` en dehors du gestionnaire principal, le développeur a initialisé dans l'instruction `if`. Ainsi, Lambda n'exécute ce code que si `some_condition` est satisfaite. Si vous initialisez `CLIENT_A` en dehors du gestionnaire principal, Lambda exécute ce code à chaque démarrage à froid. Cela peut augmenter le temps de latence global.

## Utilisation de variables d'environnement pour visualiser et contrôler le comportement de simultanéité provisionné

Il est possible que votre fonction utilise la totalité de sa simultanéité provisionnée. Lambda utilise des instances à la demande pour gérer tout trafic excédentaire. Pour déterminer quel type d'initialisation Lambda a utilisé pour un environnement particulier, vérifiez la valeur de la variable d'environnement `AWS_LAMBDA_INITIALIZATION_TYPE`. Cette variable a deux valeurs possibles : `provisioned-concurrency` ou`on-demand`. La valeur de `AWS_LAMBDA_INITIALIZATION_TYPE` est immuable et reste constante pendant toute la durée de vie de l'environnement. Pour vérifier la valeur d'une variable d'environnement dans le code de votre fonction, consultez???.

Si vous utilisez le. NET6 ou. NET7 environnements d'exécution, vous pouvez configurer la variable d'`AWS_LAMBDA_DOTNET_PREJIT`environnement pour améliorer la latence des fonctions, même si elles n'utilisent pas la simultanéité provisionnée. Le. NETruntime utilise une compilation et une initialisation paresseuses pour chaque bibliothèque que votre code appelle pour la première fois. Par conséquent, la première invocation d'une fonction Lambda peut prendre plus de temps que les invocations suivantes. Pour atténuer ce problème, vous pouvez choisir l'une des trois valeurs `AWS_LAMBDA_DOTNET_PREJIT` :

- `ProvisionedConcurrency`: Lambda effectue la ahead-of-time JIT compilation pour tous les environnements en utilisant la simultanéité provisionnée. C'est la valeur par défaut.
- `Always`: Lambda effectue la ahead-of-time JIT compilation pour chaque environnement, même si la fonction n'utilise pas la simultanéité provisionnée.
- `Never`: Lambda désactive la ahead-of-time JIT compilation pour tous les environnements.

## Comprendre le comportement de journalisation et de facturation grâce à la simultanéité provisionnée

Pour les environnements de simultanéité provisionnée, le code d'initialisation de votre fonction s'exécute pendant l'allocation et périodiquement lorsque Lambda recycle les instances actives de votre environnement. Vous pouvez consulter le temps d'initialisation dans les journaux et les traces

après le traitement d'une demande par une instance d'environnement. Il est important de noter que Lambda vous facture l'initialisation même si l'instance d'environnement ne traite jamais de demande. La simultanéité provisionnée s'exécute en continu et est facturée séparément des coûts d'initialisation et d'invocation. Pour plus d'informations, consultez Tarification AWS Lambda.

En outre, lorsque vous configurez une fonction Lambda avec une simultanéité provisionnée, Lambda pré-initialise cet environnement d'exécution afin qu'il soit disponible avant les demandes d'invocation de fonction. Cependant, votre fonction publie des journaux d'invocation CloudWatch uniquement lorsque la fonction est réellement invoquée. Par conséquent, le champ Init Duration apparaît dans la ligne de `REPORT` journal de la première invocation de fonction, même si l'initialisation a eu lieu à l'avance. Cela ne signifie pas que la fonction a démarré à froid.

## Utilisation d'Application Auto Scaling pour automatiser la gestion de la simultanéité provisionnée

Vous pouvez utiliser Application Auto Scaling pour gérer la simultanéité provisionnée selon une planification ou en fonction de l'utilisation. Si vous observez des schémas prévisibles de trafic vers votre fonction, utilisez la mise à l'échelle programmée. Si vous souhaitez que votre fonction maintienne un pourcentage d'utilisation spécifique, utilisez une politique de mise à l'échelle de suivi cible.

### Mise à l'échelle planifiée

Avec Application Auto Scaling, vous pouvez définir votre propre planification de mise à l'échelle en fonction des changements de charge prévisibles. Pour plus d'informations et des exemples, consultez les sections Scheduled Scaling for Application Auto Scaling dans le Guide de l'utilisateur d'Application Auto Scaling et Scheduling AWS Lambda Provisioned Concurrency pour les pics d'utilisation récurrents sur le AWS Compute Blog.

### Suivi de la cible

Grâce au suivi des cibles, Application Auto Scaling crée et gère un ensemble d' CloudWatch alarmes en fonction de la façon dont vous définissez votre politique de dimensionnement. Lorsque ces alarmes sont activées, Application Auto Scaling ajuste automatiquement la quantité d'environnements alloués à l'aide de la simultanéité provisionnée. Le suivi des cibles est idéal pour les applications dont les modèles de trafic ne sont pas prévisibles.

Pour dimensionner la simultanéité provisionnée à l'aide du suivi des cibles, utilisez les opérations Application Auto Scaling `RegisterScalableTarget` et `PutScalingPolicy` Application Auto

ScalingAPI. Par exemple, si vous utilisez le AWS Command Line Interface (CLI), procédez comme suit :

1. Enregistrez l'alias d'une fonction en tant que cible de mise à l'échelle. L'exemple suivant enregistre l'BLUEalias d'une fonction nommée `my-function` :

```
aws application-autoscaling register-scalable-target --service-namespace lambda \
    --resource-id function:my-function:BLUE --min-capacity 1 --max-capacity 100 \
    --scalable-dimension lambda:function:ProvisionedConcurrency
```

2. Ensuite, appliquez une stratégie de mise à l'échelle à la cible. L'exemple suivant configure Application Auto Scaling pour ajuster la configuration de simultanéité provisionnée pour un alias afin de maintenir le taux d'utilisation proche de 70 %, mais vous pouvez appliquer n'importe quelle valeur comprise entre 10 % et 90 %.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace lambda \
    --scalable-dimension lambda:function:ProvisionedConcurrency \
    --resource-id function:my-function:BLUE \
    --policy-name my-policy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration '{ "TargetValue":
 0.7, "PredefinedMetricSpecification": { "PredefinedMetricType":
 "LambdaProvisionedConcurrencyUtilization" }}'
```

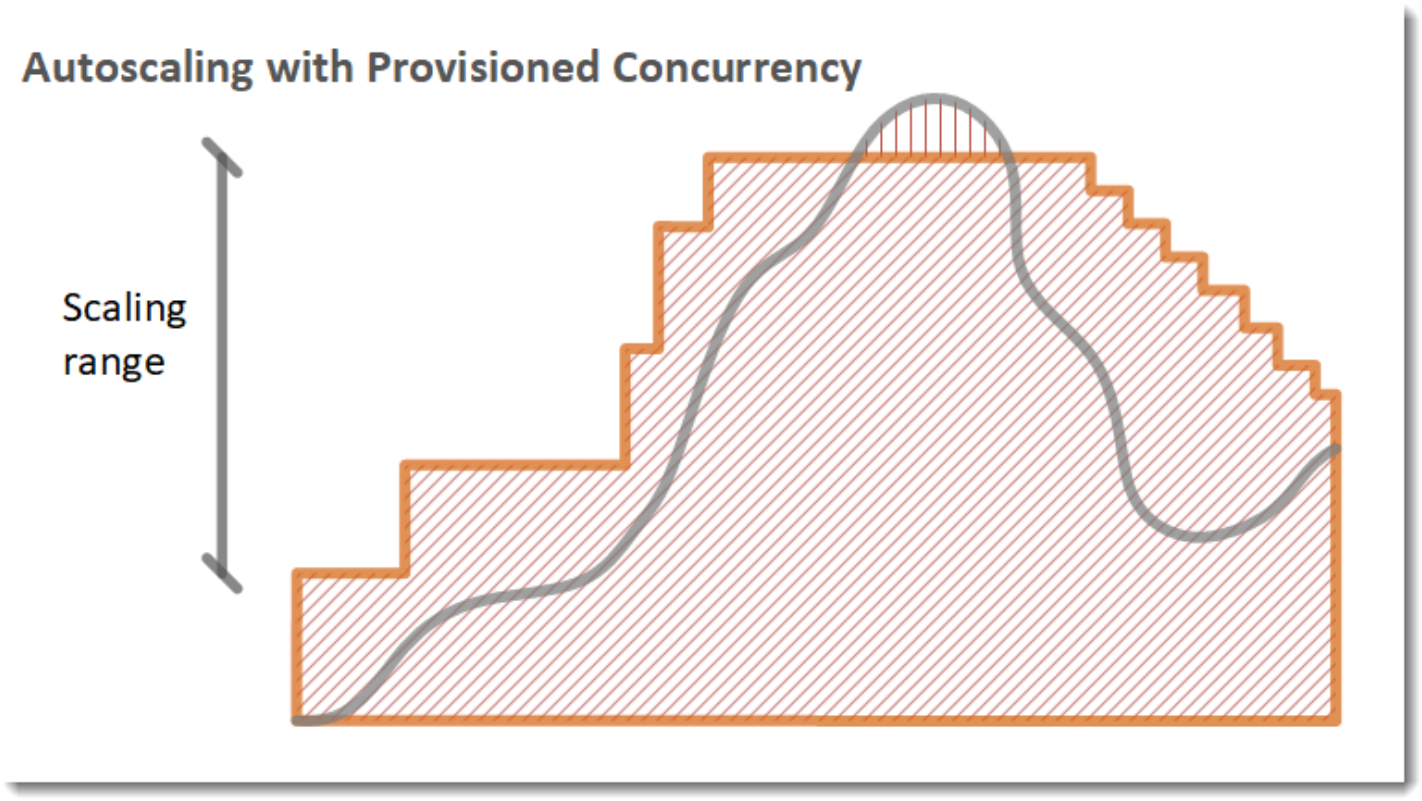
Vous devriez obtenir un résultat du type suivant :

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:12266dbb-1524-xmpl-a64e-9a0a34b996fa:resource/lambda/
function:my-function:BLUE:policyName/my-policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
            "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7"
        },
        {
```

```
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66",
            "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66"
        }
    ]
}
```

Application Auto Scaling crée deux alarmes dans CloudWatch. La première alarme se déclenche lorsque l'utilisation de la simultanéité provisionnée dépasse systématiquement 70 %. Lorsque cela se produit, Application Auto Scaling alloue davantage de simultanéité approvisionnée afin de réduire l'utilisation. La deuxième alarme se déclenche lorsque l'utilisation est constamment inférieure à 63 % (90 % de la cible de 70 %). Lorsque cela se produit, Application Auto Scaling réduit la simultanéité approvisionnée de l'alias.

Dans l'exemple suivant, une fonction adapte son échelle entre une quantité minimum et maximum de simultanéité approvisionnée en fonction de l'utilisation.



Légende

- Instances de la fonction
- Demandes ouvertes
- Simultanéité provisionnée
- Simultanéité standard

Quand le nombre de demandes ouvertes augmente, Application Auto Scaling augmente la simultanéité provisionnée par échelons jusqu'à ce qu'elle atteigne le maximum configuré. Une fois le maximum atteint, la fonction peut continuer à se mettre à l'échelle en fonction de la simultanéité standard, non réservée, si votre compte n'a pas atteint sa limite de simultanéité. Lorsque l'utilisation chute et reste constamment faible, Application Auto Scaling réduit la simultanéité provisionnée par échelons périodiques plus petits.

Les deux alarmes gérées par Application Auto Scaling utilisent la statistique moyenne par défaut. Les fonctions qui subissent des rafales de trafic peuvent ne pas déclencher ces alarmes. Par exemple, supposons que votre fonction Lambda s'exécute rapidement (c'est-à-dire entre 20 et 100 ms) et que votre modèle de trafic se produise sous forme de rafales rapides. Dans ce cas, le nombre de demandes dépasse la simultanéité allouée pendant la rafale. Cependant, Application Auto Scaling nécessite que la charge en rafale soit maintenue pendant au moins 3 minutes afin de provisionner des environnements supplémentaires. De plus, les deux CloudWatch alarmes nécessitent 3 points de données atteignant la moyenne cible pour activer la politique de dimensionnement automatique. Si votre fonction connaît des pics de trafic rapides, l'utilisation de la statistique Maximum au lieu de la statistique Average peut être plus efficace pour dimensionner la simultanéité provisionnée afin de minimiser les démarrages à froid.

Pour plus d'informations sur l'utilisation des politiques de mise à l'échelle du suivi des cibles, consultez Politiques de mise à l'échelle du suivi des cibles pour Application Auto Scaling.

# Comportement de mise à l’échelle Lambda

Au fur et à mesure que votre fonction reçoit des demandes, Lambda augmente automatiquement le nombre d’environnements d’exécution pour traiter ces demandes jusqu’à ce que votre compte atteigne sa limite de simultanéité. Toutefois, pour éviter une mise à l’échelle excessive en réponse à des pics de trafic soudains, Lambda limite la rapidité avec laquelle vos fonctions peuvent être mises à l’échelle. Ce taux de mise à l'échelle simultanée est le taux maximal auquel les fonctions de votre compte peuvent évoluer en réponse à l'augmentation du nombre de demandes. (C’est-à-dire la rapidité avec laquelle Lambda peut créer de nouveaux environnements d’exécution.) Le taux d'échelonnement de la simultanéité est différent de la limite de simultanéité au niveau du compte, qui est le montant total de simultanéité disponible pour vos fonctions.

## Taux de mise à l’échelle de la simultanéité

Dans chacune et pour chaque fonction Région AWS, votre taux de mise à l'échelle de la simultanéité est de 1 000 instances d'environnement d'exécution toutes les 10 secondes (ou 10 000 requêtes par seconde toutes les 10 secondes). En d'autres termes, toutes les 10 secondes, Lambda peut allouer au maximum 1 000 instances d'environnement d'exécution supplémentaires, ou répondre à 10 000 requêtes supplémentaires par seconde, à chacune de vos fonctions.

En général, il n’est pas nécessaire de se soucier de cette limitation. La vitesse de mise à l’échelle de Lambda est suffisante dans la plupart des cas d’utilisation.

Il est important de noter que le taux de mise à l'échelle de la simultanéité est une limite au niveau de la fonction. Cela signifie que chaque fonction de votre compte peut être mise à l’échelle indépendamment des autres fonctions.

> **Note**
>
> Dans la pratique, Lambda s’efforce d’augmenter votre taux de simultanéité de manière continue au fil du temps, plutôt que de procéder à une seule recharge de 1 000 unités toutes les 10 secondes.

Lambda n’accumule pas les portions inutilisées de votre taux de mise à l’échelle de la simultanéité. Cela signifie qu’à tout moment, votre taux de mise à l’échelle est toujours de 1 000 unités simultanées au maximum. Par exemple, si vous n’utilisez aucune de vos 1 000 unités de simultanéité disponibles dans un intervalle de 10 secondes, vous n’accumulerez pas 1 000 unités

supplémentaires dans le prochain intervalle de 10 secondes. Votre taux de mise à l'échelle de la simultanéité est toujours de 1 000 dans le prochain intervalle de 10 secondes.

Tant que votre fonction continue de recevoir un nombre croissant de demandes, Lambda est mis à l'échelle au rythme le plus rapide à votre disposition, jusqu'à la limite de simultanéité de votre compte. Vous pouvez limiter le niveau de simultanéité que les fonctions individuelles peuvent utiliser en configurant la simultanéité réservée. Si l'entrée des demandes est plus rapide que la capacité de mise à l'échelle de votre fonction ou si votre fonction atteint la simultanéité maximale, des demandes supplémentaires échouent alors avec un code de limitation (code d'état 429).

# Surveillance de la simultanéité

Lambda émet des métriques CloudWatch Amazon pour vous aider à surveiller la simultanéité de vos fonctions. Cette rubrique explique ces métriques et comment les interpréter.

Sections


- Métriques de simultanéité générales
- Métriques de simultanéité provisionnée
- Travailler avec la métrique ClaimedAccountConcurrency


## Métriques de simultanéité générales

Utilisez les métriques suivantes pour surveiller la simultanéité de vos fonctions Lambda. La granularité de chaque métrique est d'une minute.

- `ConcurrentExecutions` : le nombre d'invocations simultanées actives à un moment donné. Lambda émet cette métrique pour toutes les fonctions, versions et alias. Pour toute fonction de la console Lambda, Lambda affiche le graphique pour `ConcurrentExecutions` nativement native dans l'onglet Surveillance, sous Métriques. Affichez cette métrique à l'aide de MAX.
- `UnreservedConcurrentExecutions` : le nombre d'invocations simultanées actives qui utilisent la simultanéité non réservée. Lambda émet cette métrique pour toutes les fonctions d'une région. Affichez cette métrique à l'aide de MAX.
- `ClaimedAccountConcurrency` – Le niveau de simultanéité qui n'est pas disponible pour les appels à la demande. `ClaimedAccountConcurrency` est égal à `UnreservedConcurrentExecutions` plus le montant de la simultanéité allouée (c'est-à-dire le total de la simultanéité réservée plus le total de la simultanéité provisionnée). Si `ClaimedAccountConcurrency` dépasse la limite de simultanéité de votre compte, vous pouvez demander une limite de simultanéité de compte plus élevée. Affichez cette métrique à l'aide de MAX. Pour de plus amples informations, veuillez consulter Travailler avec la métrique `ClaimedAccountConcurrency`.

## Métriques de simultanéité provisionnée

Utilisez les métriques suivantes pour surveiller les fonctions Lambda utilisant la simultanéité provisionnée. La granularité de chaque métrique est d'une minute.

- `ProvisionedConcurrentExecutions` : le nombre d'instances d'environnement d'exécution qui traitent activement une invocation sur une simultanéité allouée. Lambda émet cette métrique pour chaque version de fonction et alias avec simultanéité provisionnée configurée. Affichez cette métrique à l'aide de MAX.

`ProvisionedConcurrentExecutions` n'est pas le même que le nombre total de simultanéités provisionnées que vous allouez. Par exemple, supposons que vous allouiez 100 unités de simultanéité provisionnée à une version de fonction. Au cours d'une minute donnée, si au plus 50 de ces 100 environnements d'exécution géraient les invocations simultanément, la valeur de MAX(`ProvisionedConcurrentExecutions`) est 50.

- `ProvisionedConcurrencyInvocations` – Le nombre de fois où Lambda invoque votre code de fonction en utilisant la simultanéité provisionnée. Lambda émet cette métrique pour chaque version de fonction et alias avec simultanéité provisionnée configurée. Affichez cette métrique à l'aide de SUM.

`ProvisionedConcurrencyInvocations` diffère de `ProvisionedConcurrentExecutions`, car `ProvisionedConcurrencyInvocations` compte le nombre total d'invocations, tandis que `ProvisionedConcurrentExecutions` compte le nombre d'environnements actifs. Pour comprendre cette distinction, examinons le scénario suivant :

Dans cet exemple, supposons que vous receviez une invocation par minute et que chaque invocation dure deux minutes. Chaque barre horizontale orange représente une demande unique. Supposons que vous allouiez dix unités de simultanéité provisionnée à cette fonction, de sorte que chaque demande s'exécute sur la simultanéité provisionnée.

Entre les minutes 0 et 1, `Request 1` entre en jeu. À la minute 1, la valeur de MAX(`ProvisionedConcurrentExecutions`) est 1, car au plus un environnement d'exécution était actif au cours de la dernière minute. La valeur de SUM(`ProvisionedConcurrencyInvocations`) est également 1, puisqu'une nouvelle demande est arrivée au cours de la dernière minute.

Entre les minutes 1 et 2, `Request 2` entre en jeu et `Request 1` continue de fonctionner. À la minute 2, la valeur de MAX(`ProvisionedConcurrentExecutions`) est 2, car au plus 2 environnements d'exécution étaient actifs au cours de la dernière minute. Cependant, la valeur de SUM(`ProvisionedConcurrencyInvocations`) est 1, car une seule nouvelle demande est arrivée au cours de la dernière minute. Ce comportement métrique se poursuit jusqu'à la fin de l'exemple.

- `ProvisionedConcurrencySpilloverInvocations` : le nombre de fois où Lambda invoque votre fonction sur la simultanéité standard (réservée ou non) lorsque toute la simultanéité provisionnée est utilisée. Lambda émet cette métrique pour chaque version de fonction et alias avec simultanéité provisionnée configurée. Affichez cette métrique à l'aide de SUM. La valeur de `ProvisionedConcurrencyInvocations` + `ProvisionedConcurrencySpilloverInvocations` doit être égale au nombre total d'invocations de fonctions (c'est-à-dire à la métrique `Invocations`).

  `ProvisionedConcurrencyUtilization` : le pourcentage de la simultanéité provisionnée utilisée (c'est à dire la valeur de `ProvisionedConcurrentExecutions` divisée par la quantité totale de la simultanéité provisionnée allouée). Lambda émet cette métrique pour chaque version de fonction et alias avec simultanéité provisionnée configurée. Affichez cette métrique à l'aide de MAX.

Par exemple, supposons que vous provisionniez 100 unités de simultanéité provisionnée à une version de fonction. Au cours d'une minute donnée, si au plus 60 de ces 100 environnements d'exécution géraient des invocations simultanément, la valeur de MAX(`ProvisionedConcurrentExecutions`) est 60 et la valeur de MAX(`ProvisionedConcurrencyUtilization`) est 0,6.

Une valeur élevée pour `ProvisionedConcurrencySpilloverInvocations` peut indiquer que vous devez allouer un accès simultané supplémentaire à votre fonction. Vous pouvez également configurer Application Auto Scaling pour gérer la mise à l'échelle automatique de la simultanéité provisionnée en fonction de seuils prédéfinis.

À l'inverse, des valeurs constamment faibles pour `ProvisionedConcurrencyUtilization` peuvent indiquer que vous avez trop alloué de simultanéité provisionnée pour votre fonction.

## Travailler avec la métrique **`ClaimedAccountConcurrency`**

Lambda utilise la métrique `ClaimedAccountConcurrency` pour déterminer le niveau de simultanéité disponible sur votre compte pour les appels à la demande. Lambda calcule `ClaimedAccountConcurrency` à l'aide de la formule suivante :

```
ClaimedAccountConcurrency = UnreservedConcurrentExecutions + (allocated concurrency)
```

`UnreservedConcurrentExecutions` correspond au nombre d'invocations simultanées actives qui utilisent la simultanéité non réservée. La simultanéité allouée est la somme des deux parties suivantes (en les remplaçant RC en tant que « simultanéité réservée » et PC en tant que « simultanéité provisionnée ») :

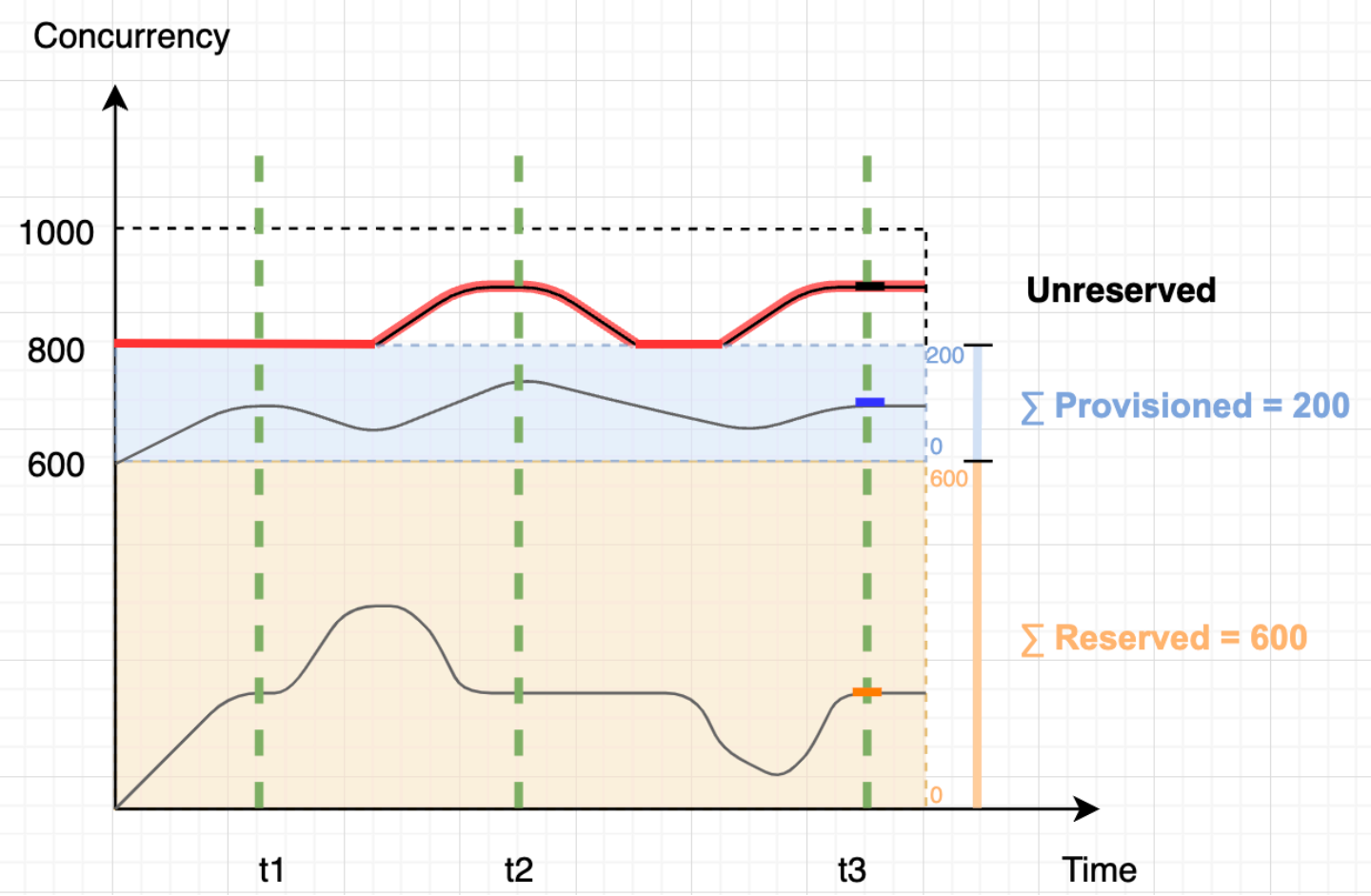
- Total pour RC entre toutes les fonctions d'une région.
- Total de PC entre toutes les fonctions d'une région qui utilisent PC, à l'exception des fonctions qui utilisent RC.

> **Note**
>
> Vous ne pouvez pas allouer plus de PC que ce RC pour une fonction. Ainsi, le RC d'une fonction est toujours supérieur ou égal à son PC. Pour calculer la contribution à la simultanéité allouée pour de telles fonctions avec PC et RC, Lambda ne prend en compte que RC, qui est le maximum des deux.

Lambda utilise la métrique `ClaimedAccountConcurrency` plutôt que `ConcurrentExecutions` pour déterminer le niveau de simultanéité disponible pour les appels à la demande. Bien que la métrique `ConcurrentExecutions` soit utile pour suivre le nombre d'appels simultanés actifs, elle ne reflète pas toujours votre véritable disponibilité simultanée. En effet, Lambda prend également en compte la simultanéité réservée et la simultanéité provisionnée pour déterminer la disponibilité.

Pour illustrer `ClaimedAccountConcurrency`, imaginez un scénario dans lequel vous configurez une grande partie de la simultanéité réservée et de la simultanéité provisionnée entre vos fonctions qui restent largement inutilisées. Dans l'exemple suivant, supposons que la limite de simultanéité de

votre compte est de 1 000 et que votre compte comporte deux fonctions principales : `function-orange` et `function-blue`. Vous allouez 600 unités de simultanéité réservée pour `function-orange`. Vous allouez 200 unités de simultanéité provisionnée pour `function-blue`. Supposons qu'au fil du temps, vous déployiez des fonctions supplémentaires et observiez le schéma de trafic suivant :




Dans le schéma précédent, les lignes noires indiquent l'utilisation simultanée réelle au fil du temps, et la ligne rouge indique la valeur de `ClaimedAccountConcurrency` au fil du temps. Dans ce scénario, `ClaimedAccountConcurrency` est égal à 800 au minimum malgré une faible utilisation réelle de la simultanéité entre vos fonctions. Cela est dû au fait que vous avez alloué 800 unités de simultanéité au total pour `function-orange` et `function-blue`. Du point de vue de Lambda, vous avez « revendiqué » cette simultanéité pour l'utiliser, il ne vous reste donc que 200 unités de simultanéité pour les autres fonctions.

Pour ce scénario, la simultanéité allouée est de 800 dans la formule `ClaimedAccountConcurrency`. Nous pouvons ensuite déduire la valeur de `ClaimedAccountConcurrency` en différents points du diagramme :

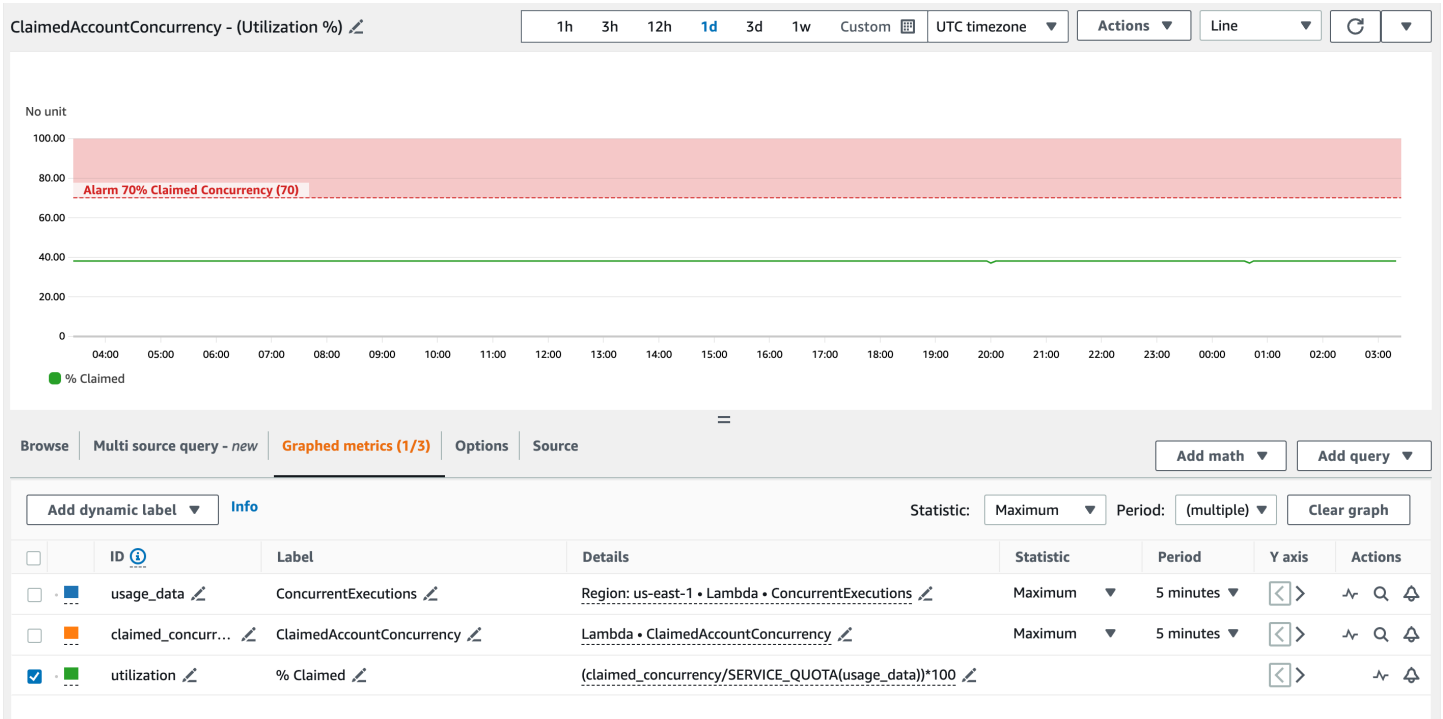
- Au niveau de `t1`, `ClaimedAccountConcurrency` est égal à 800 (800 + 0 `UnreservedConcurrentExecutions`).
- Au niveau de `t2`, `ClaimedAccountConcurrency` est égal à 900 (800 + 100 `UnreservedConcurrentExecutions`).
- Au niveau de `t3`, `ClaimedAccountConcurrency` est à nouveau égal à 900 (800 + 100 `UnreservedConcurrentExecutions`).

## Configuration de la **`ClaimedAccountConcurrency`** métrique dans CloudWatch

Lambda émet la métrique en entrée. `ClaimedAccountConcurrency` CloudWatch Utilisez cette métrique avec la valeur de `SERVICE_QUOTA(ConcurrentExecutions)` pour obtenir le pourcentage d'utilisation de la simultanéité sur votre compte, comme indiqué dans la formule suivante :

```
Utilization = (ClaimedAccountConcurrency/SERVICE_QUOTA(ConcurrentExecutions)) * 100%
```

La capture d'écran suivante montre comment vous pouvez représenter graphiquement cette formule CloudWatch. La ligne verte `claim_utilization` représente l'utilisation simultanée de ce compte, qui est d'environ 40 % :



La capture d'écran précédente inclut également une CloudWatch alarme qui s'ALARMactive lorsque l'utilisation simultanée dépasse 70 %. Vous pouvez utiliser la métrique `ClaimedAccountConcurrency` ainsi que des alarmes similaires pour déterminer de manière proactive le moment où vous pourriez avoir besoin de demander une limite de simultanéité de comptes plus élevée.

# Création de fonctions Lambda avec Node.js

Vous pouvez exécuter JavaScript du code avec Node.js dans AWS Lambda. Lambda fournit des environnements d'exécution pour Node.js, qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement qui inclut le AWS SDK for JavaScript, avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez. Pour en savoir plus sur les SDK versions incluses dans les environnements d'exécution de Node.js, consultezthe section called "Versions incluses dans Runtime SDK".

Lambda prend en charge les environnements d'exécution Node.js suivants.

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Node.js 20 | `nodejs20.x` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Node.js 18 | `nodejs18.x` | Amazon Linux 2 | 31 juillet 2025 | 1 sept. 2025 | 1 octobre 2025 |

> **Note**
>
> Les environnements d'exécution de Node.js 18 et versions ultérieures sont utilisés AWS SDK pour la JavaScript version 3. Pour migrer une fonction depuis un environnement d'exécution antérieur, suivez l'atelier de migration sur GitHub. Pour plus d'informations sur AWS SDK JavaScript la version 3, consultez le billet de blog Modular AWS SDK for JavaScript is now generally available.

Pour créer une fonction Node.js.

1. Ouvrez la console Lambda.
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :

- Nom de la fonction : saisissez le nom de la fonction.
- Exécution : choisissez Node.js 20.x.

4. Sélectionnez Créer une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Event name (Nom de l'événement), saisissez **`test`**.
7. Sélectionnez Save Changes.
8. Pour invoquer la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `index.js` ou `index.mjs`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'éditeur de code intégré. Choisissez Save pour enregistrer les changements. Ensuite, pour exécuter votre code, choisissez Test.

Le fichier `index.js` ou `index.mjs` exporte une fonction nommée `handler` qui accepte un objet événement et un objet contexte. Il s'agit de la fonction de gestionnaire que Lambda appel lors de l'invocation de la fonction. Le runtime de la fonction Node.js obtient des événements d'invocations à partir de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `index.handler`.

Lorsque vous enregistrez votre code de fonction, la console Lambda crée un package de déploiement d'archive de fichiers .zip. Lorsque vous développez votre code de fonction en dehors de la console (à l'aide d'unIDE), vous devez créer un package de déploiement pour télécharger votre code dans la fonction Lambda.

Le runtime de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'invocation. L'objet de contexte contient des informations supplémentaires sur l'invocation, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs. La fonction runtime envoie les détails de chaque appel à CloudWatch Logs. Il relaie tous les journaux que votre fonction génère pendant l'invocation. Si votre fonction renvoie une erreur, Lambda met en forme l'erreur et la renvoie à l'appelant.

Rubriques

- Initialisation de Node.js
- Versions incluses dans Runtime SDK

- Utilisation de keep-alive pour les connexions TCP
- Chargement des certificats CA
- Définir le gestionnaire de fonctions Lambda dans Node.js
- Déployer des fonctions Lambda en Node.js avec des archives de fichiers .zip
- Déployer des fonctions Lambda en Node.js avec des images conteneurs
- Utilisation de couches pour les fonctions Lambda de Node.js
- Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Node.js
- Enregistrez et surveillez les fonctions Lambda de Node.js
- Instrumentation du code Node.js dans AWS Lambda

## Initialisation de Node.js

Node.js possède un modèle de boucle d'événements exclusif qui rend son comportement d'initialisation différent des autres exécutions. Plus précisément, Node.js utilise un modèle d'I/O non bloquant qui prend en charge les opérations asynchrones. Ce modèle permet à Node.js de fonctionner efficacement pour la plupart des charges de travail. Par exemple, si une fonction Node.js effectue un appel réseau, cette demande peut être désignée comme une opération asynchrone et placée dans une file d'attente de rappel. La fonction peut continuer à traiter d'autres opérations dans la pile d'appels principale sans être bloquée en attendant le retour de l'appel réseau. Une fois l'appel réseau terminé, son rappel est exécuté, puis supprimé de la file d'attente de rappel.

Certaines tâches d'initialisation peuvent s'exécuter de manière asynchrone. L'exécution de ces tâches asynchrones n'est pas garantie avant l'invocation. Par exemple, le code qui effectue un appel réseau pour récupérer un paramètre depuis AWS le Parameter Store peut ne pas être terminé au moment où Lambda exécute la fonction de gestion. Par conséquent, la variable peut être nulle lors d'une invocation. Pour éviter cela, assurez-vous que les variables et autres codes asynchrones sont entièrement initialisés avant de poursuivre le reste de la logique métier principale de la fonction.

Vous pouvez également désigner votre code de fonction comme module ES, ce qui vous permet d'utiliser `await` au niveau supérieur du fichier, hors du champ d'application de votre gestionnaire de fonctions. Lorsque vous utilisez `await` chaque `Promise`, le code d'initialisation asynchrone se termine avant les invocations du gestionnaire, maximisant ainsi l'efficacité de la simultanéité provisionnée dans la réduction de la latence de démarrage à froid. Pour obtenir des informations et un exemple, consultez Utilisation des modules ES Node.js et de premier niveau d'attente dans AWS Lambda.

## Désignation d'un gestionnaire de fonctions en tant que module ES

Par défaut, Lambda traite les fichiers portant le suffixe `.js` comme des modules CommonJS. En option, vous pouvez désigner votre code comme un module ES. Vous pouvez le faire de deux manières : en spécifiant le suffixe `type` comme `module` dans le fichier `package.json` de la fonction, ou en utilisant l'extension de nom de fichier `.mjs`. Dans la première approche, le code de votre fonction traite tous les fichiers `.js` comme des modules ES, tandis que dans le second scénario, seul le fichier que vous spécifiez avec `.mjs` est un module ES. Vous pouvez mélanger les modules ES et les modules CommonJS en les nommant respectivement `.mjs` et `.cjs`, car les fichiers `.mjs` sont toujours des modules ES et les fichiers `.cjs` sont toujours des modules CommonJS.

Lambda recherche des dossiers dans la variable d'`NODE_PATH`environnement lors du chargement des modules ES. Vous pouvez charger AWS SDK ce qui est inclus dans le runtime à l'aide des `import` instructions du module ES. Vous pouvez également charger des modules ES à partir de couches.

# Versions incluses dans Runtime SDK

La version du moteur d'exécution AWS SDK inclus dans le fichier d'exécution Node.js dépend de la version d'exécution et de votre Région AWS. Pour trouver la version SDK incluse dans le moteur d'exécution que vous utilisez, créez une fonction Lambda avec le code suivant.

> **Note**
>
> L'exemple de code ci-dessous pour les versions 18 et supérieures de Node.js utilise le format CommonJS. Si vous créez la fonction dans la console Lambda, veillez à renommer le fichier contenant le code en. `index.js`

Example Node.js 18 et versions ultérieures

```
const { version } = require("@aws-sdk/client-s3/package.json");

exports.handler = async () => ({ version });
```

Cela renvoie une réponse au format suivant :

```
{
  "version": "3.462.0"
}
```

## Utilisation de keep-alive pour les connexions TCP

L'HTTPSagent Node.jsHTTP/par défaut crée une nouvelle TCP connexion pour chaque nouvelle demande. Pour éviter les coûts liés à l'établissement de nouvelles connexions, vous pouvez `keepAlive: true` réutiliser les connexions que votre fonction effectue à l'aide du AWS SDK for JavaScript. Keep-Alive peut réduire le temps de requête pour les fonctions Lambda qui effectuent plusieurs API appels à l'aide du. SDK

Dans la AWS SDK version JavaScript 3.x, qui est incluse dans les environnements d'exécution Lambda `nodejs18.x` et versions ultérieures, keep-alive est activé par défaut. Pour désactiver keep-alive, consultez la section Réutilisation des connexions avec keep-alive dans le fichier Node.js du guide du AWS SDK développeur pour 3.x. JavaScript  Pour plus d'informations sur l'utilisation de keep-alive, voir HTTPkeep-alive is on by default in modular AWS SDK for JavaScript sur le AWS blog des outils de développement.

## Chargement des certificats CA

Pour les versions d'exécution de Node.js antérieures à Node.js 18, Lambda charge automatiquement les certificats CA (autorité de certification) spécifiques à Amazon afin de vous permettre de créer plus facilement des fonctions qui interagissent avec d'autres services. AWS Par exemple, Lambda inclut les RDS certificats Amazon nécessaires pour valider le certificat d'identité du serveur installé sur votre base de données Amazon. RDS Ce comportement peut avoir un impact sur les performances lors des démarrages à froid.

À partir de Node.js 20, Lambda ne charge plus de certificats CA supplémentaires par défaut. L'exécution Node.js 20 contient un fichier de certificat contenant tous les certificats Amazon CA situés à l'adresse `/var/runtime/ca-cert.pem`. Pour restaurer le même comportement à partir de Node.js 18 et exécutions antérieures, définissez la variable d'environnement `NODE_EXTRA_CA_CERTS`sur `/var/runtime/ca-cert.pem`.

Pour des performances optimales, nous vous recommandons d'effectuer la création d'une offre groupée uniquement pour les certificats dont vous avez besoin avec votre package de déploiement et de les charger via la variable d'environnement `NODE_EXTRA_CA_CERTS`. Le fichier

de certificats doit être composé d'un ou de plusieurs certificats d'autorité de certification racine ou intermédiaire approuvés au PEM format. Par exemple, pourRDS, incluez les certificats requis à côté de votre code en tant que`certificates/rds.pem`. Chargez ensuite les certificats en réglant `NODE_EXTRA_CA_CERTS` sur `/var/task/certificates/rds.pem`.

# Définir le gestionnaire de fonctions Lambda dans Node.js

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Rubriques


- Notions de base sur le gestionnaire Node.js
- Dénomination
- Utilisation d'async/await
- Utilisation de callbacks
- Bonnes pratiques en matière de code pour les fonctions Lambda de Node.js


## Notions de base sur le gestionnaire Node.js

L'exemple de fonction suivant enregistre le contenu de l'objet d'événement et renvoie l'emplacement des journaux.

> **Note**
>
> Cette page présente des exemples de gestionnaires de modules CommonJS et ES. Pour en savoir plus sur la différence entre ces deux types de gestionnaires, consultez Désignation d'un gestionnaire de fonctions en tant que module ES.

ES module handler

Example

```
export const handler = async (event, context) => {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

CommonJS module handler

Example

```
exports.handler = async function (event, context) {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

Lorsque vous configurez une fonction, la valeur du paramètre du gestionnaire est le nom du fichier et le nom de la méthode du gestionnaire exporté, séparés par un point. La valeur par défaut dans la console et pour les exemples de ce guide est `index.handler`. Cela indique la méthode `handler` qui est exportée à partir du fichier `index.js`.

L'exécution transmet des arguments à la méthode du gestionnaire. Le premier argument est l'objet `event`, qui contient les informations de l'appelant. L'invocateur transmet ces informations sous forme de chaîne JSON formatée lorsqu'il appelle Invoke, et le moteur d'exécution les convertit en objet. Lorsqu'un AWS service invoque votre fonction, la structure de l'événement varie d'un service à l'autre.

Le deuxième argument est l'objet de contexte, qui contient des informations sur l'invocation, la fonction et l'environnement d'exécution. Dans l'exemple précédent, la fonction obtient le nom du flux de journaux de l'objet de contexte et le renvoie au mécanisme d'invocation.

Vous pouvez également utiliser un argument callback, qui est une fonction que vous pouvez appeler dans les gestionnaires non asynchrones pour envoyer une réponse. Nous vous recommandons d'utiliser async/await plutôt que des callbacks. Async/await améliore la lisibilité, la gestion des erreurs et l'efficacité. Pour plus d'informations sur les différences entre async/await et les callbacks, consultez Utilisation de callbacks.

## Dénomination

Lorsque vous configurez une fonction, la valeur du paramètre du gestionnaire est le nom du fichier et le nom de la méthode du gestionnaire exporté, séparés par un point. La valeur par défaut pour les fonctions créées dans la console et pour les exemples présentés dans ce guide est`index.handler`. Cela indique la `handler` méthode qui est exportée depuis le `index.mjs` fichier `index.js` or.

Si vous créez une fonction dans la console en utilisant un nom de fichier ou un nom de gestionnaire de fonction différent, vous devez modifier le nom du gestionnaire par défaut.

Pour modifier le nom du gestionnaire de fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Cliquez sur l'onglet Code.
3. Faites défiler l'écran jusqu'au volet Paramètres d'exécution et choisissez Modifier.
4. Dans Gestionnaire, saisissez le nouveau nom de votre gestionnaire de fonction.
5. Choisissez Save (Enregistrer).

## Utilisation d’async/await

Si votre code exécute une tâche asynchrone, utilisez le modèle async/await pour vous assurer que le gestionnaire termine son exécution. Async/await est un moyen concis et lisible d’écrire du code asynchrone dans Node.js, sans avoir besoin de callbacks imbriqués ou de promesses en chaîne. Avec async/await, vous pouvez écrire du code qui se lit comme du code synchrone, tout en étant asynchrone et non bloquant.

Le mot-clé `async` marque une fonction comme étant asynchrone, et le mot-clé `await` met en pause l’exécution de la fonction jusqu’à ce qu’une `Promise` soit résolue.

> **Note**
>
> Assurez-vous d’attendre la fin des événements asynchrones. Si la fonction retourne avant que les événements asynchrones ne soient terminés, elle risque d’échouer ou de provoquer un comportement inattendu dans votre application. Cela peut se produire lorsqu’une boucle `forEach` contient un événement asynchrone. Les boucles `forEach` attendent un appel synchrone. Pour plus d'informations, consultez Array.prototype. forEach() dans la documentation de Mozilla.

ES module handler

Example — HTTP demande avec async/await

```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
```

```
        // fetch is available in Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

CommonJS module handler

Example — HTTP demande avec async/await

```
const https = require("https");
let url = "https://aws.amazon.com/";

exports.handler = async function (event) {
  let statusCode;
  await new Promise(function (resolve, reject) {
    https.get(url, (res) => {
        statusCode = res.statusCode;
        resolve(statusCode);
      }).on("error", (e) => {
        reject(Error(e));
      });
  });
  console.log(statusCode);
  return statusCode;
};
```

L'exemple suivant utilise async/await pour répertorier vos compartiments Amazon Simple Storage Service.

> **Note**
>
> Avant d'utiliser cet exemple, assurez-vous que le rôle d'exécution de votre fonction dispose d'autorisations de lecture Amazon S3.

ES module handler

Example — AWS SDK v3 avec async/await

Cet exemple utilise la AWS SDK for JavaScript version 3, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

```
import {S3Client, ListBucketsCommand} from '@aws-sdk/client-s3';
const s3 = new S3Client({region: 'us-east-1'});

export const handler = async(event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;

};
```

CommonJS module handler

Example — AWS SDK v3 avec async/await

Cet exemple utilise la AWS SDK for JavaScript version 3, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

```
const { S3Client, ListBucketsCommand } = require('@aws-sdk/client-s3');
const s3 = new S3Client({ region: 'us-east-1' });

exports.handler = async (event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;
};
```

## Utilisation de callbacks

Nous vous recommandons d'utiliser async/await pour déclarer le gestionnaire de fonctions au lieu d'utiliser des callbacks. Async/await est un meilleur choix pour plusieurs raisons :

- Lisibilité : le code Async/await est plus facile à lire et à comprendre que le code callback, qui peut rapidement devenir difficile à suivre et entraîner l'enfer du callback.
- Débogage et gestion des erreurs : le débogage du code basé sur des callbacks peut être difficile. La pile d'appels peut devenir difficile à suivre et les erreurs peuvent facilement se perdre. Avec async/await, vous pouvez utiliser des blocs try/catch pour gérer les erreurs.

- Efficacité : les callbacks nécessitent souvent de basculer entre différentes parties du code. Async/await permet de réduire le nombre de changements de contexte, ce qui se traduit par un code plus efficace.

Lorsque vous utilisez des callbacks dans votre gestionnaire, la fonction continue à s'exécuter jusqu'à ce que la boucle d'événement soit vide ou que la fonction s'arrête. La réponse n'est pas envoyée à l'appelant tant que toutes les tâches d'événement de boucle ne sont pas terminées. Si la fonction expire, une erreur est renvoyée à la place. Vous pouvez configurer le moteur d'exécution pour envoyer la réponse immédiatement en définissant le contexte. callbackWaitsForEmptyEventLoopà faux.

La fonction de rappel accepte deux arguments : un(e) `Error` et une réponse. L'objet de réponse doit être compatible avec `JSON.stringify`.

L'exemple de fonction suivant vérifie a URL et renvoie le code d'état à l'invocateur.

ES module handler

Example — HTTP demande avec rappel

```
import https from "https";
let url = "https://aws.amazon.com/";

export function handler(event, context, callback) {
  https.get(url, (res) => {
      callback(null, res.statusCode);
    }).on("error", (e) => {
      callback(Error(e));
    });
}
```

CommonJS module handler

Example — HTTP demande avec rappel

```
const https = require("https");
let url = "https://aws.amazon.com/";

exports.handler = function (event, context, callback) {
  https.get(url, (res) => {
      callback(null, res.statusCode);
```

```
    }).on("error", (e) => {
      callback(Error(e));
    });
};
```

Dans l'exemple suivant, la réponse du Simple Storage Service (Amazon S3) est renvoyée au mécanisme d'invocation dès qu'elle est disponible. Le délai d'expiration de l'événement en boucle est gelé et reprend lors de l'invocation suivante de la fonction.

> **Note**
>
> Avant d'utiliser cet exemple, assurez-vous que le rôle d'exécution de votre fonction dispose d'autorisations de lecture Amazon S3.

ES module handler

Example — AWS SDK v3 avec callbackWaitsFor EmptyEventLoop

Cet exemple utilise la AWS SDK for JavaScript version 3, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

```
import AWS from "@aws-sdk/client-s3";
const s3 = new AWS.S3({});

export const handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

CommonJS module handler

Example — AWS SDK v3 avec callbackWaitsFor EmptyEventLoop

Cet exemple utilise la AWS SDK for JavaScript version 3, qui est disponible dans les environnements d'exécution `nodejs18.x` et ultérieurs.

```
const AWS = require("@aws-sdk/client-s3");
```

```
const s3 = new AWS.S3({});

exports.handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

## Bonnes pratiques en matière de code pour les fonctions Lambda de Node.js

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire. Dans Node.js, vous obtenez ce qui suit :

```
exports.myHandler = function(event, context, callback) {
 var foo = event.foo;
 var bar = event.bar;
 var result = MyLambdaFunction (foo, bar);

 callback(null, result);
}

function MyLambdaFunction (foo, bar) {
 // MyLambdaFunction logic here
}
```

- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour les environnements d'exécution Node.js et Python, ceux-ci incluent le AWS SDKs. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l’environnement d’exécution.

- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation.

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles

que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.

- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Déployer des fonctions Lambda en Node.js avec des archives de fichiers .zip

Le code de votre AWS Lambda fonction comprend un fichier .js ou .mjs contenant le code du gestionnaire de votre fonction, ainsi que les packages et modules supplémentaires dont dépend votre code. Pour déployer ce code de fonction vers Lambda, vous utilisez un package de déploiement. Ce package peut être une archive de fichier .zip ou une image de conteneur. Pour plus d'informations sur l'utilisation d'images de conteneur avec Node.js, consultez la page Déployer des fonctions Lambda Node.js avec des images de conteneur.

Pour créer votre package de déploiement sous forme d'archive de fichier .zip, vous pouvez utiliser l'utilitaire d'archivage .zip intégré à votre outil de ligne de commande, ou tout autre utilitaire .zip tel que 7zip. Les exemples présentés dans les sections suivantes supposent que vous utilisez un outil `zip` de ligne de commande dans un environnement Linux ou macOS. Pour utiliser les mêmes commandes sous Windows, vous pouvez installer le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et de Bash.

Notez que Lambda utilise des autorisations de POSIX fichier. Vous devrez donc peut-être définir des autorisations pour le dossier du package de déploiement avant de créer l'archive de fichiers .zip.

Rubriques


- Dépendances d'exécution dans Node.js
- Création d'un package de déploiement .zip sans dépendances
- Création d'un package de déploiement .zip avec dépendances
- Création d'une couche Node.js pour vos dépendances
- Chemin de recherche des dépendances et bibliothèques incluses dans l'exécution
- Création et mise à jour de fonctions Lambda Node.js à l'aide de fichiers .zip


## Dépendances d'exécution dans Node.js

Pour les fonctions Lambda qui utilisent l'exécution Node.js, une dépendance peut être n'importe quel package ou module Node.js. L'exécution Node.js comprend un certain nombre de bibliothèques courantes, ainsi qu'une version de AWS SDK for JavaScript. Le moteur d'exécution `nodejs16.x` Lambda inclut la version 2.x du. SDK Les versions d'exécution `nodejs18.x` et ultérieures incluent la version 3 duSDK. Pour utiliser la version 2 SDK avec les versions d'exécution `nodejs18.x` et les versions ultérieures, ajoutez-la SDK à votre package de déploiement de fichiers .zip. Si le runtime

que vous avez choisi inclut la version SDK que vous utilisez, vous n'avez pas besoin d'inclure la SDK bibliothèque dans votre fichier .zip. Pour savoir quelle version du SDK est incluse dans le moteur d'exécution que vous utilisez, consultezthe section called “Versions incluses dans Runtime SDK”.

Lambda met régulièrement à jour les SDK bibliothèques du runtime Node.js pour inclure les dernières fonctionnalités et mises à niveau de sécurité. Lambda applique également des correctifs de sécurité et des mises à jour aux autres bibliothèques incluses dans l’exécution. Pour avoir un contrôle total sur les dépendances de votre package, vous pouvez ajouter votre version préférée de n’importe quelle dépendance incluse dans l’exécution à votre package de déploiement. Par exemple, si vous souhaitez utiliser une version particulière du SDK for JavaScript, vous pouvez l'inclure dans votre fichier .zip en tant que dépendance. Pour plus d’informations sur l’ajout de dépendances incluses dans l’exécution à votre fichier .zip, consultez Chemin de recherche des dépendances et bibliothèques incluses dans l’exécution.

Dans le cadre du modèle de responsabilité partagée AWS, vous êtes responsable de la gestion de toutes les dépendances dans les packages de déploiement de vos fonctions. Cela inclut l’application de mises à jour et de correctifs de sécurité. Pour mettre à jour les dépendances dans le package de déploiement de votre fonction, créez d’abord un nouveau fichier .zip, puis chargez-le sur Lambda. Pour plus d’informations, consultez Création d’un package de déploiement .zip avec dépendances et Création et mise à jour de fonctions Lambda Node.js à l’aide de fichiers .zip.

## Création d’un package de déploiement .zip sans dépendances

Si le code de votre fonction ne comporte aucune dépendance à l’exception des bibliothèques incluses dans l’exécution Lambda, votre fichier .zip contient uniquement le fichier `index.js` ou `index.mjs` avec le code du gestionnaire de votre fonction. Utilisez votre utilitaire zip préféré pour créer un fichier .zip avec votre fichier `index.js` ou `index.mjs` à la racine. Si le fichier contenant le code de votre gestionnaire ne se trouve pas à la racine de votre fichier .zip, Lambda ne sera pas en mesure d’exécuter votre code.

Pour savoir comment déployer votre fichier .zip pour créer une nouvelle fonction Lambda ou mettre à jour une fonction Lambda existante, veuillez consulter la rubrique Création et mise à jour de fonctions Lambda Node.js à l’aide de fichiers .zip.

## Création d’un package de déploiement .zip avec dépendances

Si votre code de fonction dépend de packages ou de modules qui ne sont pas inclus dans l’environnement d’exécution Lambda Node.js, vous pouvez soit ajouter ces dépendances à votre fichier .zip avec votre code de fonction, soit utiliser une couche Lambda. Les instructions de cette

section vous indiquent comment inclure vos dépendances dans votre package de déploiement .zip. Pour obtenir des instructions sur la façon d'inclure vos dépendances dans une couche, voir the section called "Création d'une couche Node.js pour vos dépendances".

Les exemples de CLI commandes suivants créent un fichier .zip nommé `my_deployment_package.zip` contenant le `index.mjs` fichier `index.js` ou contenant le code du gestionnaire de votre fonction et ses dépendances. Dans l'exemple, vous installez les dépendances à l'aide du gestionnaire de packages npm.

Pour créer le package de déploiement

1. Accédez au répertoire du projet qui contient votre fichier de code source `index.js` ou `index.mjs`. Dans cet exemple, le répertoire est nommé `my_function`.

   ```
   cd my_function
   ```

2. Installez les bibliothèques requises pour votre fonction dans le répertoire `node_modules` à l'aide de la commande `npm install`. Dans cet exemple, vous installez Kit SDK AWS X-Ray pour Node.js.

   ```
   npm install aws-xray-sdk
   ```

   Cela crée une structure de dossiers similaire à ce qui suit :

   ```
   ~/my_function
   ### index.mjs
   ### node_modules
       ### async
       ### async-listener
       ### atomic-batcher
       ### aws-sdk
       ### aws-xray-sdk
       ### aws-xray-sdk-core
   ```

   Vous pouvez également ajouter des modules personnalisés que vous créez vous-même à votre package de déploiement. Créez un répertoire sous `node_modules` portant le nom de votre module et enregistrez-y vos packages écrits personnalisés.

3. Créez un fichier .zip reprenant le contenu de votre dossier de projet à la racine. Utilisez l'option `r` (récursive) pour veiller à ce que zip compresse les sous-dossiers.

```
zip -r my_deployment_package.zip .
```

## Création d'une couche Node.js pour vos dépendances

Les instructions de cette section vous indiquent comment inclure les dépendances dans une couche. Pour obtenir des instructions sur la façon d'inclure vos dépendances dans votre package de déploiement, voir the section called "Création d'un package de déploiement .zip avec dépendances".

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d'exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

- `nodejs/node_modules`
- `nodejs/node16/node_modules (NODE_PATH)`
- `nodejs/node18/node_modules (NODE_PATH)`
- `nodejs/node20/node_modules (NODE_PATH)`

Par exemple, la structure du fichier .zip de votre couche peut ressembler à ce qui suit :

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

En outre, Lambda détecte automatiquement toutes les bibliothèques dans le répertoire `/opt/lib` et tous les fichiers binaires dans le répertoire `/opt/bin`. Pour que Lambda trouve correctement le contenu de votre couche, vous pouvez aussi créer une couche avec la structure suivante :

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Après avoir empaqueté votre couche, reportez-vous à the section called “Création et suppression de couches” et à the section called “Ajout de couches” pour terminer la configuration de votre couche.

## Chemin de recherche des dépendances et bibliothèques incluses dans l’exécution

L’exécution Node.js comprend un certain nombre de bibliothèques courantes, ainsi qu’une version de AWS SDK for JavaScript. Si vous souhaitez utiliser une version différente d’une bibliothèque incluse dans l’exécution, vous pouvez le faire en l’associant à votre fonction ou en l’ajoutant en tant que dépendance dans votre package de déploiement. Par exemple, vous pouvez utiliser une version différente du SDK en l'ajoutant à votre package de déploiement .zip. Vous pouvez également l’inclure dans une couche Lambda pour votre fonction.

Lorsque vous utilisez une instruction `import` ou `require` dans votre code, l’exécution Node.js recherche les répertoires dans le chemin `NODE_PATH` jusqu’à ce qu’elle trouve le module. Par défaut, le premier emplacement dans lequel recherche l’exécution est le répertoire dans lequel votre package de déploiement .zip est décompressé et monté (`/var/task`). Si vous ajoutez une version d’une bibliothèque incluse dans l’exécution dans votre package de déploiement, cette version prévaudra sur la version incluse dans l’exécution. Les dépendances de votre package de déploiement ont également la priorité sur les dépendances des couches.

Lorsque vous ajoutez une dépendance à une couche, Lambda l’extrait vers `/opt/nodejs/nodexx/node_modules`, où `nodexx` représente la version de l’exécution que vous utilisez. Dans le chemin de recherche, ce répertoire a la priorité sur le répertoire contenant les bibliothèques incluses dans l’exécution (`/var/lang/lib/node_modules`). Les bibliothèques des couches de fonctions ont donc la priorité sur les versions incluses dans l’exécution.

Vous pouvez voir le chemin de recherche complet de votre fonction Lambda en ajoutant la ligne de code suivante.

```
console.log(process.env.NODE_PATH)
```

Vous pouvez également ajouter des dépendances dans un dossier distinct au sein de votre package .zip. Par exemple, vous pouvez ajouter un module personnalisé à un dossier votre package .zip appelé `common`. Lorsque votre package .zip est décompressé et monté, ce dossier est placé dans le répertoire `/var/task`. Pour utiliser une dépendance provenant d'un dossier de votre package de déploiement .zip dans votre code, utilisez une `const { } = require()` instruction `import { } from` or, selon que vous utilisez CJS ou la résolution du ESM module. Par exemple :

```
import { myModule } from './common'
```

Si vous regroupez votre code à `esbuild`, `rollup`, ou similaire, les dépendances utilisées par votre fonction sont regroupées dans un ou plusieurs fichiers. Nous vous recommandons d'utiliser cette méthode pour vendre des dépendances chaque fois que cela est possible. Par rapport à l'ajout de dépendances à votre package de déploiement, le regroupement de votre code permet d'améliorer les performances en raison de la réduction des opérations d'E/S.

## Création et mise à jour de fonctions Lambda Node.js à l'aide de fichiers .zip

Après avoir créé votre package de déploiement .zip, vous pouvez l'utiliser pour créer une nouvelle fonction Lambda ou mettre à jour une fonction existante. Vous pouvez déployer votre package .zip à l'aide de la console Lambda, AWS Command Line Interface du, et du Lambda. API Vous pouvez également créer et mettre à jour des fonctions Lambda à l'aide de l' AWS Serverless Application Model (AWS SAM) et de AWS CloudFormation.

La taille maximale d'un package de déploiement .zip pour Lambda est de 250 Mo (décompressé). Notez que cette limite s'applique à la taille combinée de tous les fichiers que vous chargez, y compris les couches Lambda.

Le runtime Lambda a besoin d'une autorisation pour lire les fichiers de votre package de déploiement. Dans la notation octale des autorisations Linux, Lambda a besoin de 644 autorisations pour les fichiers non exécutables (rw-r--r--) et de 755 autorisations () pour les répertoires et les fichiers exécutables. rwxr-xr-x

Sous Linux et macOS, utilisez la commande `chmod` pour modifier les autorisations de fichiers sur les fichiers et les répertoires de votre package de déploiement. Par exemple, pour accorder les autorisations appropriées à un fichier exécutable, exécutez la commande suivante.

```
chmod 755 <filepath>
```

Pour modifier les autorisations relatives aux fichiers dans Windows, voir Définir, afficher, modifier ou supprimer des autorisations sur un objet dans la documentation Microsoft Windows.

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide de la console

Pour créer une nouvelle fonction, vous devez d'abord créer la fonction dans la console, puis charger votre archive .zip. Pour mettre à jour une fonction existante, ouvrez la page de votre fonction, puis suivez la même procédure pour ajouter votre fichier .zip mis à jour.

Si votre fichier .zip fait moins de 50 Mo, vous pouvez créer ou mettre à jour une fonction en chargeant le fichier directement à partir de votre ordinateur local. Pour les fichiers .zip de plus de 50 Mo, vous devez d'abord charger votre package dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS Management Console, consultez Getting started with Amazon S3. Pour télécharger des fichiers à l'aide de AWS CLI, voir Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Pour créer une nouvelle fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Choisissez Créer à partir de zéro.
3. Sous Informations de base, procédez comme suit :
   a. Pour Nom de la fonction, saisissez le nom de la fonction.
   b. Pour Exécution, sélectionnez l'exécution que vous souhaitez utiliser.
   c. (Facultatif) Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour votre fonction. L'architecture par défaut est x86_64. Assurez-vous que le package de déploiement .zip pour votre fonction est compatible avec l'architecture de l'ensemble d'instructions que vous sélectionnez.
4. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou en utiliser un existant.
5. Sélectionnez Create function (Créer une fonction). Lambda crée une fonction de base « Hello world » à l'aide de l'exécution de votre choix.

Pour charger une archive .zip à partir de votre ordinateur local (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger le fichier .zip.

2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez Fichier .zip.
5. Pour charger un fichier .zip, procédez comme suit :

    a. Sélectionnez Charger, puis choisissez votre fichier .zip dans le sélecteur de fichiers.

    b. Choisissez Ouvrir.

    c. Choisissez Save (Enregistrer).

Pour charger une archive .zip depuis un compartiment Amazon S3 (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger un nouveau fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez l'emplacement Amazon S3.
5. Collez le lien Amazon S3 URL de votre fichier .zip et choisissez Enregistrer.

## Mise à jour des fonctions du fichier .zip à l'aide de l'éditeur de code de la console

Pour certaines fonctions avec des packages de déploiement .zip, vous pouvez utiliser l'éditeur de code intégré de la console Lambda pour mettre à jour le code de votre fonction directement. Pour utiliser cette fonctionnalité, votre fonction doit répondre aux critères suivants :

- Votre fonction doit utiliser l'une des exécutions des langages interprétés (Python, Node.js ou Ruby).
- Le package de déploiement de votre fonction doit être inférieur à 3 Mo.

Le code des fonctions avec les packages de déploiement d'images de conteneurs ne peut pas être édité directement dans la console.

Pour mettre à jour le code de fonction à l'aide de l'éditeur de code de la console

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Sélectionnez l'onglet Code.

3. Dans le volet Source du code, sélectionnez votre fichier de code source et modifiez-le dans l'éditeur de code intégré.
4. Lorsque vous avez fini de modifier votre code, choisissez Déployer pour enregistrer vos modifications et mettre à jour votre fonction.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide du AWS CLI

Vous pouvez utiliser l'AWS CLI pour créer une nouvelle fonction ou pour mettre à jour une fonction existante à l'aide d'un fichier .zip. Utilisez la fonction de création et update-function-codeles commandes pour déployer votre package .zip. Si votre fichier .zip est inférieur à 50 Mo, vous pouvez charger le package .zip à partir d'un emplacement de fichier sur votre machine de génération locale. Pour les fichiers plus volumineux, vous devez charger votre package .zip à partir d'un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Si vous chargez votre fichier .zip depuis un compartiment Amazon S3 à l'aide de AWS CLI, le compartiment doit se trouver au même endroit Région AWS que votre fonction.

Pour créer une nouvelle fonction à l'aide d'un fichier .zip avec le AWS CLI, vous devez spécifier les éléments suivants :

- Le nom de votre fonction (`--function-name`)
- L'exécution de votre fonction (`--runtime`)
- Le nom de ressource Amazon (ARN) du rôle d'exécution de votre fonction (`--role`)
- Le nom de la méthode du gestionnaire dans votre code de fonction (`--handler`)

Vous devez également indiquer l'emplacement de votre fichier .zip. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda create-function --function-name myFunction \
--runtime nodejs20.x --handler index.handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez l'option `--code` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `S3ObjectVersion` pour les objets soumis à la gestion des versions.

```
aws lambda create-function --function-name myFunction \
--runtime nodejs20.x --handler index.handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Pour mettre à jour une fonction existante à l'aide duCLI, vous devez spécifier le nom de votre fonction à l'aide du `--function-name` paramètre. Vous devez également spécifier l'emplacement du fichier .zip que vous souhaitez utiliser pour mettre à jour votre code de fonction. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez les options `--s3-bucket` et `--s3-key` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `--s3-object-version` pour les objets soumis à la gestion des versions.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide du Lambda API

Pour créer et mettre à jour des fonctions à l'aide d'une archive de fichiers .zip, effectuez les API opérations suivantes :

- CreateFunction
- UpdateFunctionCode

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS SAM

The AWS Serverless Application Model (AWS SAM) est une boîte à outils qui permet de rationaliser le processus de création et d'exécution d'applications sans serveur sur AWS. Vous définissez les ressources de votre application dans un JSON modèle YAML or et vous utilisez l'interface de ligne de AWS SAM commande (AWS SAM CLI) pour créer, empaqueter et déployer vos applications. Lorsque vous créez une fonction Lambda à partir d'un AWS SAM modèle, elle crée AWS SAM automatiquement un package de déploiement ou une image de conteneur .zip avec le code de votre fonction et les dépendances que vous spécifiez. Pour en savoir plus sur l'utilisation des fonctions Lambda AWS SAM pour créer et déployer des fonctions Lambda, consultez Getting started with AWS SAM dans le Guide du AWS Serverless Application Model développeur.

Vous pouvez également créer une fonction Lambda AWS SAM à l'aide d'une archive de fichiers .zip existante. Pour créer une fonction Lambda à l'aide de AWS SAM, vous pouvez enregistrer votre fichier .zip dans un compartiment Amazon S3 ou dans un dossier local sur votre machine de génération. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS SAM modèle, la `AWS::Serverless::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l’aide d’une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `CodeUri`- défini sur l'Amazon S3URI, le chemin d'accès au dossier local ou à l'FunctionCodeobjet du code de fonction
- `Runtime` : défini sur votre exécution choisie

Ainsi AWS SAM, si votre fichier .zip est supérieur à 50 Mo, vous n'avez pas besoin de le télécharger au préalable dans un compartiment Amazon S3. AWS SAM peut télécharger des packages .zip jusqu'à la taille maximale autorisée de 250 Mo (décompressés) à partir d'un emplacement sur votre machine de compilation locale.

Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS SAM, voir :Serverless AWS : :Function dans le manuel du développeur.AWS SAM

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip. Pour créer une fonction Lambda à partir d'un fichier .zip, vous devez d'abord charger votre fichier dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS CloudFormation modèle, la `AWS::Lambda::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l'aide d'une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `Code` : saisissez le nom du compartiment Amazon S3 et le nom du fichier .zip dans les champs `S3Bucket` et `S3Key`
- `Runtime` : défini sur votre exécution choisie

Le fichier .zip AWS CloudFormation généré ne peut pas dépasser 4 Mo. Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS CloudFormation, voir :Lambda AWS : :Function dans le guide de l'utilisateur.AWS CloudFormation

# Déployer des fonctions Lambda en Node.js avec des images conteneurs

Il existe trois méthodes pour créer une image de conteneur pour une fonction Lambda Node.js :

- Utilisation d'une image AWS de base pour Node.js

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Node.js dans l'image.

- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Node.js dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

Rubriques


- AWS images de base pour Node.js
- Utilisation d'une image AWS de base pour Node.js
- Utilisation d'une autre image de base avec le client d'interface d'exécution


## AWS images de base pour Node.js

AWS fournit les images de base suivantes pour Node.js :

| Balises | Environnement d'exécution | Système d'exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| 20 | Node.js 20 | Amazon Linux | Dockerfile pour Node.js 2.0 sur GitHub | Non prévu |
| 18 | Node.js 18 | Amazon Linux 2 | Dockerfile pour Node.js 18 sur GitHub | 31 juillet 2025 |

ECRRéférentiel Amazon : gallery.ecr.aws/lambda/nodejs

Les images de base de Node.js 20 et versions ultérieures sont basées sur l'image de conteneur minimale Amazon Linux 2023. Les images de base antérieures utilisaient Amazon Linux 2. AL2Le 023 offre plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que`glibc`.

AL2Les images basées sur le format 023 sont utilisées `microdnf` (en lien symbolique comme`dnf`) comme gestionnaire de packages au lieu de`yum`, qui est le gestionnaire de packages par défaut dans Amazon Linux 2. `microdnf`est une implémentation autonome de. `dnf` Pour obtenir la liste des packages inclus dans les images AL2 basées sur la version 023, reportez-vous aux colonnes Conteneur minimal de la section Comparaison des packages installés sur les images de conteneurs Amazon Linux 2023. Pour plus d'informations sur les différences entre AL2 023 et Amazon Linux 2, consultez Présentation du runtime Amazon Linux 2023 AWS Lambda sur le blog AWS Compute.

> **Note**
>
> Pour exécuter des images AL2 basées sur 023 localement, y compris avec AWS Serverless Application Model (AWS SAM), vous devez utiliser Docker version 20.10.10 ou ultérieure.

## Utilisation d'une image AWS de base pour Node.js

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker (version minimale 20.10.10 pour les images de base de Node.js 2.0 et versions ultérieures)
- Node.js

Création d'une image à partir d'une image de base

Pour créer une image de conteneur à partir d'une image AWS de base pour Node.js

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir example
cd example
```

2. Créez un nouveau projet Node.js avec `npm`. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

```
npm init
```

3. Créez un nouveau fichier appelé `index.js`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

   Example Gestionnaire CommonJS

```
exports.handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
```

```
    };
    return response;
};
```

4. Si votre fonction dépend de bibliothèques autres que celles-ci AWS SDK for JavaScript, utilisez npm pour les ajouter à votre package.
5. Créez un nouveau Dockerfile avec la configuration suivante :

   - Définissez la `FROM` propriété sur URIl'image de base.
   - Utilisez la COPY commande pour copier le code de fonction et les dépendances d'exécution dans une variable `{LAMBDA_TASK_ROOT}` d'environnement définie par Lambda.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:20

# Copy function code
COPY index.js ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "index.handler" ]
```

6. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une

fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

1. Démarrez votre image Docker à l'aide de la commande docker run. Dans cet exemple, `docker-image` est le nom de l'image et `test` est la balise.

   ```
   docker run --platform linux/amd64 -p 9000:8080 --read-only docker-image:test
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

2. À partir d'une nouvelle fenêtre de terminal, publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
```

```
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilisation d’une autre image de base avec le client d’interface d’exécution

Si vous utilisez une image de base uniquement pour le système d’exploitation ou une autre image de base, vous devez inclure le client d'interface d'exécution dans votre image. Le client d’interface d’exécution étend leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l’interaction entre Lambda et votre code de fonction.

Installez le client d’interface d’exécution Node.js à l’aide du gestionnaire de packages npm :

```
npm install aws-lambda-ric
```

Vous pouvez également télécharger le client d'interface d'exécution Node.js depuis GitHub. Le client d’interface d’exécution prend en charge les versions Node.js suivantes :

- 14.x
- 16.x

- 18.x
- 20.x

L'exemple suivant montre comment créer une image de conteneur pour Node.js à l'aide d'une image non AWS basique. L'exemple Dockerfile utilise une image de base `buster`. Le Dockerfile inclut le client d'interface d'exécution.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker
- Node.js

Création d'une image à partir d'une image de base alternative

Pour créer une image de conteneur à partir d'une image non AWS basique

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir example
cd example
```

2. Créez un nouveau projet Node.js avec `npm`. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

```
npm init
```

3. Créez un nouveau fichier appelé `index.js`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

   Example Gestionnaire CommonJS

```
exports.handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
```

```
    };
    return response;
};
```

4. Créez un nouveau fichier Docker. Le Dockerfile suivant utilise une image de base `buster` au lieu d'une image de base AWS. Le Dockerfile inclut le client d'interface d'exécution, ce qui rend l'image compatible avec Lambda. Le Dockerfile utilise une création à plusieurs étapes. La première étape crée une image de génération, qui est un environnement Node.js standard dans lequel les dépendances de la fonction sont installées. La deuxième étape crée une image plus fine qui inclut le code de la fonction et ses dépendances. Cela permet de réduire la taille de l'image finale.

   - Définissez la propriété `FROM` pour l'identifiant de l'image de base.
   - Utilisez la commande `COPY` pour copier le code de la fonction et les dépendances de l'exécution.
   - Définissez le `ENTRYPOINT` sur le module que vous souhaitez que le conteneur Docker exécute lorsqu'il démarre. Dans ce cas, le module est le client d'interface d'exécution.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM node:20-buster as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Install build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
```

```
    cmake \
    unzip \
    libcurl4-openssl-dev

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

WORKDIR ${FUNCTION_DIR}

# Install Node.js dependencies
RUN npm install

# Install the runtime interface client
RUN npm install aws-lambda-ric

# Grab a fresh slim copy of the image to reduce the final size
FROM node:20-buster-slim

# Required for Node runtimes which use npm@8.6.0+ because
# by default npm writes logs under /home/.npm and Lambda fs is read-only
ENV NPM_CONFIG_CACHE=/tmp/.npm

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT ["/usr/local/bin/npx", "aws-lambda-ric"]
# Pass the name of the function handler as an argument to the runtime
CMD ["index.handler"]
```

5. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester l'image localement. Vous pouvez intégrer l'émulateur à votre image ou utiliser la procédure suivante pour l'installer sur votre machine locale.

Pour installer et exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Depuis le répertoire de votre projet, exécutez la commande suivante pour télécharger l'émulateur d'interface d'exécution (architecture x86-64) GitHub et l'installer sur votre machine locale.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Pour installer l'émulateur arm64, remplacez le GitHub référentiel de URL la commande précédente par le suivant :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }
   ```

```
$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Pour installer l'émulateur arm64, remplacez `$downloadLink` par ce qui suit :

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Démarrez votre image Docker à l'aide de la commande docker run. Notez ce qui suit :

   - `docker-image` est le nom de l'image et `test` est la balise.
   - `/usr/local/bin/npx aws-lambda-ric index.handler` est le ENTRYPOINT suivi du CMD depuis votre Dockerfile.

   Linux/macOS

   ```
   docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
    9000:8080 \
       --entrypoint /aws-lambda/aws-lambda-rie \
       --read-only \
       docker-image:test \
           /usr/local/bin/npx aws-lambda-ric index.handler
   ```

   PowerShell

   ```
   docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
    9000:8080 `
   --entrypoint /aws-lambda/aws-lambda-rie `
   --read-only `
   docker-image:test `
       /usr/local/bin/npx aws-lambda-ric index.handler
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

3. Publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
    "application/json"
   ```

4. Obtenez l'ID du conteneur.

   ```
   docker ps
   ```

5. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.

   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
```

```
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemple :

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

# Utilisation de couches pour les fonctions Lambda de Node.js

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration. La création d'une couche implique trois étapes générales :

1. Package le contenu de votre couche. Cela implique de créer une archive de fichiers .zip contenant les dépendances que vous souhaitez utiliser dans vos fonctions.
2. Créez la couche dans Lambda.
3. Ajoutez la couche à vos fonctions.

Cette rubrique contient des étapes et des conseils sur la manière d'empaqueter et de créer correctement une couche Lambda Node.js avec des dépendances de bibliothèque externes.

Rubriques


- Prérequis
- Compatibilité de la couche Node.js avec l'environnement d'exécution Lambda
- Chemins de couche pour les environnements d'exécution de Node.js
- Empaqueter le contenu de la couche
- Création de la couche
- Ajouter la couche à votre fonction


## Prérequis

Pour suivre les étapes décrites dans cette section, vous devez disposer des éléments suivants :

- Node.js 20 et le gestionnaire de packages npm. Pour plus d'informations sur l'installation de Node.js, consultez la section Installation de Node.js via le gestionnaire de packages dans la documentation Node.js.
- AWS Command Line Interface (AWS CLI) version 2

Tout au long de cette rubrique, nous faisons référence à l'exemple d'application layer-nodejs dans le référentiel. aws-lambda-developer-guide GitHub Cette application contient des scripts qui téléchargent une dépendance et l'empaquetent dans une couche. L'application contient également

les fonctions correspondantes qui utilisent la dépendance de la couche. Après avoir créé une couche, vous pouvez déployer et invoquer la fonction correspondante pour vérifier que tout fonctionne. Cet exemple d'application utilise le moteur d'exécution Node.js 20. Si vous ajoutez des dépendances supplémentaires à votre couche, elles doivent être compatibles avec Node.js 20.

L'`layer-nodejs`exemple d'application va empaqueter la bibliothèque lodash dans une couche Lambda. Le `layer` répertoire contient les scripts permettant de générer la couche. Le `function` répertoire contient un exemple de fonction permettant de vérifier le fonctionnement de la couche. Ce document explique comment créer, empaqueter, déployer et tester cette couche.

## Compatibilité de la couche Node.js avec l'environnement d'exécution Lambda

Lorsque vous empaquetez du code dans une couche Node.js, vous spécifiez les environnements d'exécution Lambda avec lesquels le code est compatible. Pour évaluer la compatibilité du code avec un environnement d'exécution, déterminez les versions de Node.js, les systèmes d'exploitation et les architectures de jeu d'instructions pour lesquelles le code est conçu.

Les environnements d'exécution Lambda Node.js spécifient leur version Node.js et leur système d'exploitation. Dans ce document, vous allez utiliser le moteur d'exécution Node.js 20, basé sur AL2 023. Pour plus d'informations sur les versions d'exécution, consultezthe section called “Environnements d'exécution pris en charge”. Lorsque vous créez une fonction Lambda, vous spécifiez l'architecture du jeu d'instructions. Dans ce document, vous allez utiliser l'`arm64`architecture. Pour plus d'informations sur les architectures Lambda, consultez. the section called “Jeux d'instructions (ARM/x86)”

Lorsque vous utilisez le code fourni dans un package, chaque responsable du package définit indépendamment sa compatibilité. La plupart des développements de Node.js sont conçus pour fonctionner indépendamment du système d'exploitation et de l'architecture du jeu d'instructions. De plus, il n'est pas courant de rompre les incompatibilités avec les nouvelles versions de Node.js. Attendez-vous à passer plus de temps à évaluer la compatibilité entre les packages qu'à évaluer la compatibilité des packages avec la version de Node.js, le système d'exploitation ou l'architecture du jeu d'instructions.

Parfois, les packages Node.js incluent du code compilé, ce qui vous oblige à prendre en compte la compatibilité du système d'exploitation et de l'architecture du jeu d'instructions. Si vous devez évaluer la compatibilité du code de vos packages, vous devez inspecter les packages et leur documentation. Les packages NPM inclus peuvent spécifier leur compatibilité via les `cpu` champs `enginesos`, et

de leur fichier `package.json` manifeste. Pour plus d'informations sur `package.json` les fichiers, consultez package.json dans la documentation. NPM

## Chemins de couche pour les environnements d'exécution de Node.js

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans l'environnement d'exécution. Si votre couche regroupe les dépendances dans des chemins de dossier spécifiques, l'environnement d'exécution Node.js reconnaîtra les modules et vous pourrez les référencer à partir de votre code de fonction.

Pour vous assurer que vos modules sont sélectionnés, empaquez-les dans le fichier .zip de votre couche dans l'un des chemins de dossier suivants. Ces fichiers sont stockés dans `/opt` la variable d'PATHenvironnement et les chemins de dossier sont chargés dans cette variable.

- `nodejs/node_modules`
- `nodejs/node`*X*`/node_modules`

Par exemple, le fichier de couche .zip obtenu que vous créez dans ce didacticiel possède la structure de répertoire suivante :

```
layer_content.zip
# nodejs
    # node20
        # node_modules
            # lodash
            # <other potential dependencies>
            # ...
```

Vous allez placer la `lodash`bibliothèque dans le `nodejs/node20/node_modules` répertoire. Cela garantit que Lambda peut localiser la bibliothèque lors des appels de fonctions.

## Empaqueter le contenu de la couche

Dans cet exemple, vous allez empaqueter la `lodash` bibliothèque dans un fichier .zip de couche. Procédez comme suit pour installer et empaqueter le contenu de la couche.

Pour installer et empaqueter le contenu de votre couche

1. Clonez le référentiel aws-lambda-developer-guide à partir GitHub duquel vous trouverez l'exemple de code dont vous avez besoin dans le `sample-apps/layer-nodejs` répertoire.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Accédez au `layer` répertoire de l'`layer-nodejs`exemple d'application. Ce répertoire contient les scripts que vous utilisez pour créer et empaqueter correctement la couche.

```
cd aws-lambda-developer-guide/sample-apps/layer-nodejs/layer
```

3. Vérifiez les listes de `package.json` fichiers`lodash`. Ce fichier définit les dépendances que vous souhaitez inclure dans la couche. Vous pouvez mettre à jour ce fichier pour inclure les dépendances que vous souhaitez dans votre couche.

> **Note**
>
> Les données `package.json` utilisées dans cette étape ne sont ni stockées ni utilisées avec vos dépendances une fois qu'elles ont été téléchargées sur une couche Lambda. Il est uniquement utilisé dans le processus d'empaquetage des couches et ne spécifie pas de commande d'exécution ni de compatibilité comme le ferait le fichier dans une application Node.js ou un package publié.

4. Assurez-vous que vous disposez de l'autorisation du shell pour exécuter les scripts du `layer` répertoire.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Exécutez le script 1-install.sh à l'aide de la commande suivante :

```
./1-install.sh
```

Ce script s'exécute`npm install`, il lit vos dépendances `package.json` et télécharge les dépendances qui y sont définies.

Example 1-install.sh

```
npm install .
```

6. Exécutez le script 2-package.sh à l'aide de la commande suivante :

```
./2-package.sh
```

Ce script copie le contenu du `node_modules` répertoire dans un nouveau répertoire nommé`nodejs/node20`. Il compresse ensuite le contenu du `nodejs` répertoire dans un fichier nommé`layer_content.zip`. Il s'agit du fichier .zip de votre couche. Vous pouvez décompresser le fichier et vérifier qu'il contient la bonne structure de fichier, comme indiqué dans la the section called “Chemins de couche pour les environnements d'exécution de Node.js” section.

Example 2-package.sh

```
mkdir -p nodejs/node20
cp -r node_modules nodejs/node20/
zip -r layer_content.zip nodejs
```

## Création de la couche

Prenez le `layer_content.zip` fichier que vous avez généré dans la section précédente et téléchargez-le en tant que couche Lambda. Vous pouvez télécharger une couche à l'aide du AWS Management Console ou du Lambda API via le AWS Command Line Interface ()AWS CLI. Lorsque vous téléchargez le fichier .zip de votre couche, dans la PublishLayerVersion AWS CLI commande suivante, spécifiez `nodejs20.x` comme environnement d'exécution compatible et `arm64` comme architecture compatible.

```
aws lambda publish-layer-version --layer-name node-lodash-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes nodejs20.x \
    --compatible-architectures "arm64"
```

Dans la réponse, notez le`LayerVersionArn`, qui ressemble à`arn:aws:lambda:us-east-1:123456789012:layer:node-lodash-layer:1`. Vous aurez besoin de cet Amazon Resource Name (ARN) à l'étape suivante de ce didacticiel, lorsque vous ajouterez la couche à votre fonction.

## Ajouter la couche à votre fonction

Déployez un exemple de fonction Lambda qui utilise la `lodash` bibliothèque dans son code de fonction, puis attachez la couche que vous avez créée. Pour déployer la fonction, vous avez besoin d'un rôle d'exécution. Pour de plus amples informations, veuillez consulter the section called “Rôle

d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources)". Si vous n'avez pas de rôle d'exécution existant, suivez les étapes décrites dans la section pliable. Sinon, passez à la section suivante pour déployer la fonction.

(Facultatif) Créez un rôle d'exécution

Pour créer un rôle d'exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :

   - Entité de confiance – Lambda.
   - Autorisations — AWSLambdaBasicExecutionRole.
   - Nom de rôle – **`lambda-role`**.

   La AWSLambdaBasicExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux dans Logs.

L'exemple de code de fonction utilise la `_.findLastIndex` méthode lodash pour lire un tableau d'objets. Il compare les objets par rapport à un critère pour trouver l'indice d'une correspondance. Il renvoie ensuite l'index et la valeur de l'objet dans la réponse Lambda.

```
import _ from "lodash"

export const handler = async (event) => {

  var users = [
  { 'user': 'Carlos',  'active': true },
  { 'user': 'Gil-dong',    'active': false },
  { 'user': 'Pat', 'active': false }
  ];

  let out = _.findLastIndex(users, function(o) { return o.user == 'Pat'; });
  const response = {
    statusCode: 200,
    body: JSON.stringify(out + ", " + users[out].user),
  };
  return response;
```

```
};
```

Pour déployer la fonction Lambda

1. Accédez au répertoire `function/`. Si vous vous trouvez actuellement dans le `layer/` répertoire, exécutez la commande suivante :

   ```
   cd ../function
   ```

2. Créez un package de déploiement de fichiers .zip à l'aide de la commande suivante :

   ```
   zip my_deployment_package.zip index.mjs
   ```

3. Déployez la fonction. Dans la AWS CLI commande suivante, remplacez le `--role` paramètre par votre rôle d'exécution ARN :

   ```
   aws lambda create-function --function-name nodejs_function_with_layer \
       --runtime nodejs20.x \
       --architectures "arm64" \
       --handler index.handler \
       --role arn:aws:iam::123456789012:role/lambda-role \
       --zip-file fileb://my_deployment_package.zip
   ```

4. Attachez la couche à votre fonction. Dans la AWS CLI commande suivante, remplacez le `--layers` paramètre par la version de couche ARN que vous avez indiquée précédemment :

   ```
   aws lambda update-function-configuration --function-name nodejs_function_with_layer
    \
       --cli-binary-format raw-in-base64-out \
       --layers "arn:aws:lambda:us-east-1:123456789012:layer:nodejs-lodash-layer:1"
   ```

5. Appelez votre fonction pour vérifier qu'elle fonctionne à l'aide de la AWS CLI commande suivante :

   ```
   aws lambda invoke --function-name nodejs_function_with_layer \
       --cli-binary-format raw-in-base64-out \
       --payload '{}' response.json
   ```

   Vous devriez obtenir un résultat du type suivant :

   ```
   {
   ```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Le `response.json` fichier de sortie contient des informations détaillées sur la réponse.

(Facultatif) Nettoyez vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la couche Lambda

1. Ouvrez la page Couches de la console Lambda.
2. Sélectionnez la couche que vous avez créée.
3. Choisissez Supprimer, puis sélectionnez à nouveau Supprimer.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

# Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Node.js

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'invocation, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `getRemainingTimeInMillis()` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `functionName` – Nom de la fonction Lambda.
- `functionVersion` – Version de la fonction.
- `invokedFunctionArn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `memoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `awsRequestId` – Identifiant de la demande d'invocation.
- `logGroupName` – Groupe de journaux pour la fonction.
- `logStreamName` – Flux de journal de l'instance de fonction.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
  - `cognitoIdentityId` – Identité Amazon Cognito authentifiée.
  - `cognitoIdentityPoolId` – Groupe d'identités Amazon Cognito ayant autorisé l'invocation.
- `clientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`

  - `env.platform`
  - `env.make`
  - `env.model`
  - `env.locale`
  - `Custom` – Personnalisez les valeurs qui sont définies par l'application client.
- `callbackWaitsForEmptyEventLoop` – Définissez ce paramètre sur faux pour envoyer la réponse dès que le rappel s'exécute, au lieu d'attendre que la boucle d'événement Node.js soit vide. Si ce paramètre est faux, les événements restants continueront de s'exécuter lors du prochain appel.

Dans l'exemple suivant, la fonction enregistre des informations de contexte et renvoie l'emplacement des journaux.

Example Fichier index.js

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

# Enregistrez et surveillez les fonctions Lambda de Node.js

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et envoie les journaux à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des informations sur chaque invocation au flux de journaux et transmet les journaux et autres sorties provenant du code de votre fonction. Pour de plus amples informations, veuillez consulter Utilisation des CloudWatch journaux et journaux avec Lambda.

Cette page explique comment générer une sortie de journal à partir du code de votre fonction Lambda et comment accéder aux journaux à l'aide de la AWS Command Line Interface console Lambda ou de la console. CloudWatch

Sections


- Création d'une fonction qui renvoie des journaux
- Utilisation des contrôles de journalisation avancés de Lambda avec Node.js
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux


## Création d'une fonction qui renvoie des journaux

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur l'objet console ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

> **Note**
>
> Nous vous recommandons d'utiliser des techniques telles que la validation des entrées et le codage des sorties lors de la journalisation des entrées. Si vous enregistrez directement les données d'entrée, un attaquant peut utiliser votre code pour rendre les altérations difficiles à détecter, falsifier des entrées de journal ou contourner les moniteurs de journal. Pour plus d'informations, consultez la section Neutralisation incorrecte des sorties pour les journaux dans l'énumération des faiblesses courantes.

Example Fichier index.js – Journalisation

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.info("EVENT\n" + JSON.stringify(event, null, 2))
  console.warn("Event not processed.")
  return context.logStreamName
}
```

Example format des journaux

```
START RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Version: $LATEST
2019-06-07T19:11:20.562Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO ENVIRONMENT
 VARIABLES
{
  "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
  "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/my-function",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2019/06/07/[$LATEST]e6f4a0c4241adcd70c262d34c0bbc85c",
  "AWS_EXECUTION_ENV": "AWS_Lambda_nodejs12.x",
  "AWS_LAMBDA_FUNCTION_NAME": "my-function",
  "PATH": "/var/lang/bin:/usr/local/bin:/usr/bin/:/bin:/opt/bin",
  "NODE_PATH": "/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/
node_modules",
  ...
}
2019-06-07T19:11:20.563Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO EVENT
{
  "key": "value"
}
2019-06-07T19:11:20.564Z c793869b-ee49-115b-a5b6-4fd21e8dedac WARN Event not processed.
END RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac
REPORT RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Duration: 128.83 ms Billed
 Duration: 200 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms
 XRAY TraceId: 1-5d9d007f-0a8c7fd02xmpl480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled:
 true
```

Le runtime Node.js enregistre les lignes `START`, `END` et `REPORT` pour chaque invocation. Il ajoute un horodatage, un ID de demande et un niveau de journalisation sur chaque entrée consignée par la fonction. La ligne de rapport fournit les détails suivants.

REPORTchamps de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les appels partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour tous les appels. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled – Pour les demandes suivies, résultat de l'échantillonnage.

Vous pouvez consulter les journaux dans la console Lambda, dans la console CloudWatch Logs ou depuis la ligne de commande.

## Utilisation des contrôles de journalisation avancés de Lambda avec Node.js

Pour mieux contrôler la manière dont les journaux de vos fonctions sont capturés, traités et consommés, vous pouvez configurer les options de journalisation suivantes pour les environnements d'exécution Node.js pris en charge :

- Format de journal : choisissez entre le format texte brut et le JSON format structuré pour les journaux de votre fonction
- Niveau de journalisation : pour les journaux au JSON format, choisissez le niveau de détail des journaux que Lambda envoie à Amazon CloudWatch, par exempleERROR, DEBUG ou INFO
- Groupe de journaux : choisissez le groupe de CloudWatch journaux auquel votre fonction envoie les journaux

Pour plus d'informations sur ces options de journalisation et pour savoir comment configurer votre fonction pour les utiliser, consultez the section called "Configuration des journaux de fonctions".

Pour utiliser le format de journal et les options de niveau de journal avec vos fonctions Node.js Lambda, consultez les instructions des sections suivantes.

## Utilisation de JSON journaux structurés avec Node.js

Si vous sélectionnez JSON le format de journal de votre fonction, Lambda enverra les journaux en utilisant les méthodes de console telles que`console.trace`,,`console.debug`, `console.log` `console.info``console.error`, et `console.warn` to CloudWatch telles que structurées. JSON Chaque objet JSON journal contient au moins quatre paires clé-valeur avec les clés suivantes :

- `"timestamp"` - heure à laquelle le message de journal a été généré
- `"level"` - niveau de journalisation attribué au message
- `"message"` - contenu du message de journal
- `"requestId"` - identifiant unique de la demande pour l'invocation de la fonction

Selon la méthode de journalisation utilisée par votre fonction, cet JSON objet peut également contenir des paires de clés supplémentaires. Par exemple, si votre fonction utilise des `console` méthodes pour enregistrer les objets d'erreur à l'aide de plusieurs arguments, l'JSONobjet contiendra des paires clé-valeur supplémentaires avec les clés `errorMessage``errorType`, et`stackTrace`.

Si votre code utilise déjà une autre bibliothèque de journalisation, telle que Powertools for AWS Lambda, pour produire des journaux JSON structurés, vous n'avez pas besoin d'apporter de modifications. Lambda n'encode pas deux fois les journaux déjà JSON codés, de sorte que les journaux d'application de votre fonction continueront d'être capturés comme avant.

Pour plus d'informations sur l'utilisation du package Powertools for AWS Lambda logging afin de créer des journaux JSON structurés dans le runtime Node.js, consultezthe section called "Journalisation".

Exemples de sorties de journal JSON formatées

Les exemples suivants montrent comment les différentes sorties de journal générées à l'aide `console` des méthodes avec des arguments uniques et multiples sont capturées dans CloudWatch Logs lorsque vous définissez le format de journal de votre fonction surJSON.

Le premier exemple utilise la méthode `console.error` pour générer une chaîne simple.

Example Code de journalisation Node.js

```
export const handler = async (event) => {
```

```
  console.error("This is a warning message");
  ...
}
```

Example JSONenregistrement du journal

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"ERROR",
    "message":"This is a warning message",
    "requestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Vous pouvez également générer des messages de journal structurés plus complexes en utilisant un ou plusieurs arguments avec les méthodes `console`. Dans l'exemple suivant, vous pouvez utiliser `console.log` pour générer deux paires clé-valeur avec un seul argument. Notez que le `"message"` champ de l'JSONobjet que Lambda envoie à CloudWatch Logs n'est pas stringifié.

Example Code de journalisation Node.js

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

Example JSONenregistrement du journal

```
{
    "timestamp": "2023-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "data": 12.3,
        "flag": false
    }
}
```

Dans l'exemple suivant, vous utilisez à nouveau la méthode `console.log` pour créer une sortie de journal. Cette fois, la méthode prend deux arguments, une carte contenant deux paires clé-valeur et une chaîne d'identification. Notez que dans ce cas, parce que vous avez fourni deux arguments, Lambda stringifie le champ `"message"`.

Example Code de journalisation Node.js

```
export const handler = async (event) => {
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

Example JSONenregistrement du journal

```
{
    "timestamp": "2023-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": "Some object -  { data: 12.3, flag: false }"
}
```

Lambda attribue les sorties générées à l'aide `console.log` du niveau de journalisation. INFO

Le dernier exemple montre comment les objets d'erreur peuvent être générés dans CloudWatch Logs à l'aide `console` des méthodes. Notez que lorsque vous enregistrez des objets d'erreur à l'aide de plusieurs arguments, Lambda ajoute les champs `errorMessage`, `errorType` et `stackTrace` à la sortie du journal.

Example Code de journalisation Node.js

```
export const handler = async (event) => {
  let e1 = new ReferenceError("some reference error");
  let e2 = new SyntaxError("some syntax error");
  console.log(e1);
  console.log("errors logged - ", e1, e2);
};
```

Example JSONenregistrement du journal

```
{
    "timestamp": "2023-12-08T23:21:04.632Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "errorType": "ReferenceError",
        "errorMessage": "some reference error",
        "stackTrace": [
```

```
            "ReferenceError: some reference error",
            "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
            "    at Runtime.handleOnceNonStreaming (file:///var/runtime/
index.mjs:1173:29)"
        ]
    }
}

{
    "timestamp": "2023-12-08T23:21:04.646Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": "errors logged -  ReferenceError: some reference error
\n    at Runtime.handler (file:///var/task/index.mjs:3:12)\n    at
 Runtime.handleOnceNonStreaming
    (file:///var/runtime/index.mjs:1173:29) SyntaxError: some syntax
 error\n    at Runtime.handler (file:///var/task/index.mjs:4:12)\n    at
 Runtime.handleOnceNonStreaming
    (file:///var/runtime/index.mjs:1173:29)",
    "errorType": "ReferenceError",
    "errorMessage": "some reference error",
    "stackTrace": [
        "ReferenceError: some reference error",
        "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
        "    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"
    ]
}
```

Lors de la journalisation de plusieurs types d'erreur, les champs supplémentaires `errorMessage`, `errorType`, et `stackTrace` sont extraits du premier type d'erreur fourni à la méthode `console`.

## Utilisation de bibliothèques clientes au format métrique intégré (EMF) avec des JSON journaux structurés

AWS fournit des bibliothèques clientes open source pour Node.js que vous pouvez utiliser pour créer des journaux au format métrique intégré (EMF). Si vous avez des fonctions existantes qui utilisent ces bibliothèques et que vous modifiez le format de journal de votre fonction enJSON, il est CloudWatch possible que les métriques émises par votre code ne soient plus reconnues.

Si votre code émet actuellement EMF des journaux directement en utilisant `console.log` ou en utilisant Powertools for AWS Lambda (TypeScript), CloudWatch vous ne pourrez pas non plus les analyser si vous modifiez le format de journal de votre fonction en. JSON

> **Important**
>
> Pour vous assurer que les EMF journaux de vos fonctions continuent d'être correctement analysés par CloudWatch, mettez à jour vos bibliothèques EMFet celles de Powertools pour les AWS Lambda bibliothèques avec les dernières versions. Si vous passez au format JSON journal, nous vous recommandons également d'effectuer des tests pour garantir la compatibilité avec les métriques intégrées de votre fonction. Si votre code émet EMF des journaux directement à l'aide `deconsole.log`, modifiez votre code pour qu'il affiche directement ces métriques, `stdout` comme indiqué dans l'exemple de code suivant.

Example code émettant des métriques intégrées pour **`stdout`**

```
process.stdout.write(JSON.stringify(
    {
        "_aws": {
            "Timestamp": Date.now(),
            "CloudWatchMetrics": [{
                "Namespace": "lambda-function-metrics",
                "Dimensions": [["functionVersion"]],
                "Metrics": [{
                    "Name": "time",
                    "Unit": "Milliseconds",
                    "StorageResolution": 60
                }]
            }]
        },
        "functionVersion": "$LATEST",
        "time": 100,
        "requestId": context.awsRequestId
    }
) + "\n")
```

## Utilisation du filtrage au niveau du journal avec Node.js

AWS Lambda Pour filtrer les journaux de vos applications en fonction de leur niveau de journalisation, votre fonction doit utiliser des journaux JSON formatés. Vous pouvez effectuer cette opération de deux façons :

- Créez des sorties de journal à l'aide des méthodes de console standard et configurez votre fonction pour utiliser le formatage des JSON journaux. AWS Lambda filtre ensuite les sorties de votre journal en utilisant la paire clé-valeur « niveau » dans l'JSONobjet décrit dansthe section called “Utilisation de JSON journaux structurés avec Node.js”. Pour savoir comment configurer le format de journal de votre fonction, consultez the section called “Configuration des journaux de fonctions”.
- Utilisez une autre bibliothèque ou méthode de journalisation pour créer des journaux JSON structurés dans votre code qui incluent une paire clé-valeur « niveau » définissant le niveau de sortie du journal. Par exemple, vous pouvez utiliser Powertools pour générer des sorties AWS Lambda de journal JSON structurées à partir de votre code. Consultez the section called “Journalisation” pour en savoir plus sur l’utilisation de Powertools avec l’exécution Node.js.

  Pour que Lambda puisse filtrer les journaux de votre fonction, vous devez également inclure une paire `"timestamp"` clé-valeur dans la sortie de votre JSON journal. L'heure doit être spécifiée dans un format d'horodatage RFC3339 valide. Si vous ne fournissez pas d'horodatage valide, Lambda attribuera le niveau au journal INFO et ajoutera un horodatage pour vous.

Lorsque vous configurez votre fonction pour utiliser le filtrage au niveau des journaux, vous sélectionnez le niveau de journaux que vous souhaitez envoyer AWS Lambda à CloudWatch Logs parmi les options suivantes :

| Niveau de journalisation | Utilisation standard |
|---|---|
| TRACE(le plus détaillé) | Les informations les plus précises utilisées pour tracer le chemin d’exécution de votre code |
| DEBUG | Informations détaillées pour le débogage du système |
| INFO | Messages qui enregistrent le fonctionnement normal de votre fonction |
| WARN | Messages relatifs à des erreurs potentielles susceptibles d’entraîner un comportement inattendu si elles ne sont pas traitées |

| Niveau de journalisation | Utilisation standard |
| --- | --- |
| ERROR | Messages concernant les problèmes qui empêchent le code de fonctionner comme prévu |
| FATAL(moindre détail) | Messages relatifs à des erreurs graves entraînant l'arrêt du fonctionnement de l'application |

Lambda envoie les journaux du niveau sélectionné et inférieur à. CloudWatch Par exemple, si vous configurez un niveau de journalisation deWARN, Lambda enverra des journaux correspondant aux niveaux WARNERROR, etFATAL.

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique,

nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

## Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
```

```
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
```

```
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
```

```
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

# Instrumentation du code Node.js dans AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des deux SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- Kit SDK AWS X-Ray pour Node.js— Et SDK pour générer et envoyer des données de trace à X-Ray.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections


- Utilisation ADOT pour instrumenter vos fonctions Node.js
- Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Node.js
- Activation du suivi avec la console Lambda
- Activation du traçage avec le Lambda API
- Activation du traçage avec AWS CloudFormation

- Interprétation d'un suivi X-Ray
- Stockage des dépendances d'exécution dans une couche (X-RaySDK)

## Utilisation ADOT pour instrumenter vos fonctions Node.js

ADOTfournit des couches Lambda entièrement gérées qui regroupent tout ce dont vous avez besoin pour collecter des données de télémétrie à l'aide du. OTel SDK En consommant cette couche, vous pouvez instrumenter vos fonctions Lambda sans avoir à modifier le code de fonction. Vous pouvez également configurer votre couche pour effectuer une initialisation personnalisée deOTel. Pour plus d'informations, consultez la section Configuration personnalisée du ADOT collecteur sur Lambda dans la ADOT documentation.

Pour les environnements d'exécution de Node.js, vous pouvez ajouter la couche Lambda AWS gérée ADOT pour Javascript afin d'instrumenter automatiquement vos fonctions. Pour obtenir des instructions détaillées sur la façon d'ajouter cette couche, consultez AWS Distro for OpenTelemetry Lambda JavaScript Support ADOT dans la documentation.

## Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Node.js

Pour enregistrer des détails sur les appels effectués par votre fonction Lambda à d'autres ressources de votre application, vous pouvez également utiliser le Kit SDK AWS X-Ray pour Node.js. Pour obtenir leSDK, ajoutez le `aws-xray-sdk-core` package aux dépendances de votre application.

Example blank-nodejs/package.json

```
{
  "name": "blank-nodejs",
  "version": "1.0.0",
  "private": true,
  "devDependencies": {
    "jest": "29.7.0"
  },
  "dependencies": {
    "@aws-sdk/client-lambda": "3.345.0",
    "aws-xray-sdk-core": "3.5.3"
  },
  "scripts": {
    "test": "jest"
  }
}
```

Pour instrumenter AWS SDK les clients dans la AWS SDK for JavaScript v3, encapsulez l'instance du client avec la `captureAWSv3Client` méthode.

Example blank-nodejs/function/index.js — Suivi d'un AWS SDK client

```
const AWSXRay = require('aws-xray-sdk-core');
const { LambdaClient, GetAccountSettingsCommand } = require('@aws-sdk/client-lambda');

// Create client outside of handler to reuse
const lambda = AWSXRay.captureAWSv3Client(new LambdaClient());

// Handler
exports.handler = async function(event, context) {
    event.Records.forEach(record => {
  ...
```

Le moteur d'exécution Lambda définit certaines variables d'environnement pour configurer le X-Ray. SDK Par exemple, Lambda définit sur pour éviter `AWS_XRAY_CONTEXT_MISSING` de `LOG_ERROR` générer des erreurs d'exécution depuis le X-Ray. SDK Pour définir une stratégie manquante de contexte personnalisée, remplacez la variable d'environnement dans votre configuration de fonction pour n'avoir aucune valeur, puis vous pouvez définir la stratégie manquante de contexte par programme.

Example Exemple de code d'initialisation

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Pour de plus amples informations, veuillez consulter the section called “Configuration des variables d’environnement”.

Après avoir ajouté les dépendances correctes et apporté les modifications de code nécessaires, activez le suivi dans la configuration de votre fonction via la console Lambda ou le. API

## Activation du suivi avec la console Lambda

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

## Activation du traçage avec le Lambda API

Configurez le suivi sur votre fonction Lambda avec AWS CLI ou utilisez AWS SDK les opérations suivantes : API

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant active le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du traçage avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
```

```
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

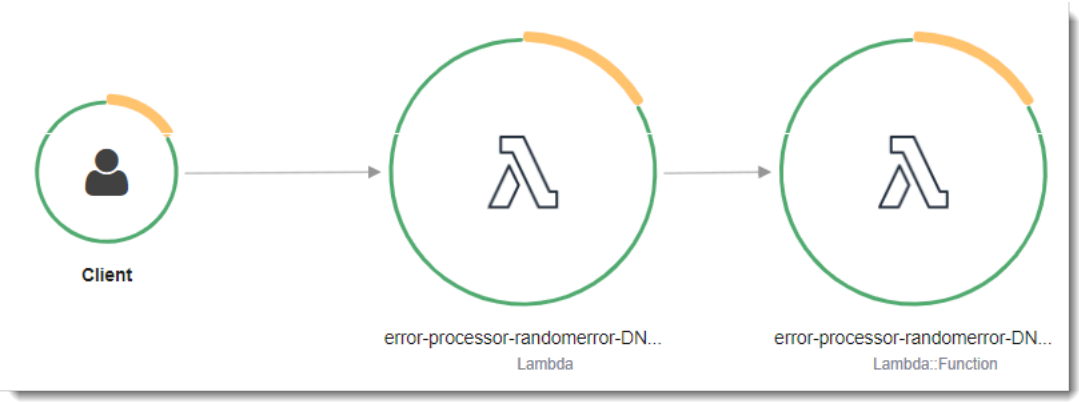
## Interprétation d'un suivi X-Ray

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Dans le cas contraire, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

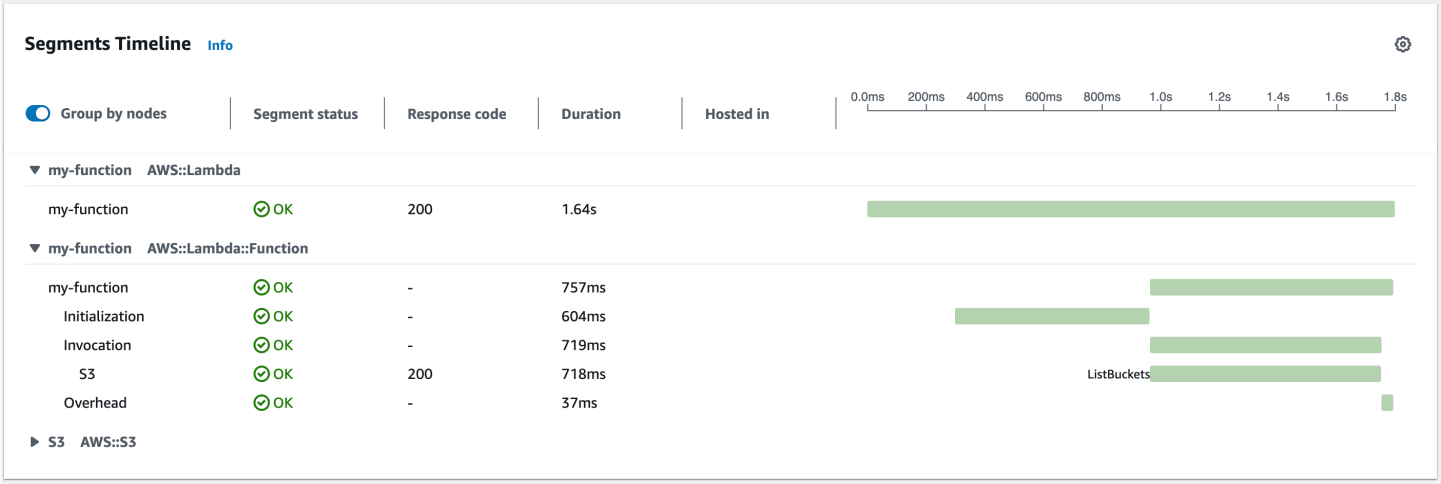
X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le

`AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez Kit SDK AWS X-Ray pour Node.js dans le AWS X-Ray Guide du développeur.

> **Tarification**
>
> Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

## Stockage des dépendances d'exécution dans une couche (X-RaySDK)

Si vous utilisez le X-Ray SDK pour instrumenter votre code de fonction chez les AWS SDK clients, votre package de déploiement peut devenir assez volumineux. Pour éviter de télécharger les dépendances d'exécution chaque fois que vous mettez à jour le code de votre fonction, empaquetez le X-Ray SDK dans une couche Lambda.

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le Kit SDK AWS X-Ray pour Node.js.

Example template.yml : couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
```

```
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - nodejs16.x
```

Avec cette configuration, vous ne mettez à jour les fichiers de couche de bibliothèque que si vous modifiez vos dépendances d'exécution. Étant donné que le package de déploiement de la fonction contient uniquement votre code, cela peut contribuer à réduire les temps de chargement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application blank-nodejs.

# Création de fonctions Lambda avec TypeScript

Vous pouvez utiliser le moteur d'exécution Node.js pour exécuter TypeScript du code dans AWS Lambda. Étant donné que Node.js n'exécute pas TypeScript le code de manière native, vous devez d'abord le transpiler dans TypeScript . JavaScript Utilisez ensuite les JavaScript fichiers pour déployer votre code de fonction sur Lambda. Votre code s'exécute dans un environnement qui inclut le AWS SDK for JavaScript, avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez. Pour en savoir plus sur les SDK versions incluses dans les environnements d'exécution de Node.js, consultezthe section called “Versions incluses dans Runtime SDK”.

Lambda prend en charge les environnements d’exécution Node.js suivants.

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Node.js 20 | `nodejs20.x` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Node.js 18 | `nodejs18.x` | Amazon Linux 2 | 31 juillet 2025 | 1 sept. 2025 | 1 octobre 2025 |

Rubriques

- Configuration d'un environnement TypeScript de développement
- Définissez le gestionnaire de fonctions Lambda dans TypeScript
- Déployez TypeScript du code transpilé dans Lambda avec des archives de fichiers .zip
- Déployez TypeScript du code transpilé dans Lambda avec des images de conteneur
- Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions TypeScript
- Enregistrez et surveillez les fonctions TypeScript Lambda
- TypeScript Code de suivi dans AWS Lambda

# Configuration d'un environnement TypeScript de développement

Utilisez un environnement de développement intégré local (IDE), un éditeur de texte ou AWS Cloud9pour écrire votre code de TypeScript fonction. Vous ne pouvez pas créer de TypeScript code sur la console Lambda.

Pour transpiler votre TypeScript code, configurez un compilateur tel que esbuild ou le TypeScript compilateur (`tsc`) de Microsoft, fourni avec la distribution. TypeScript Vous pouvez utiliser le AWS Serverless Application Model (AWS SAM) ou le AWS Cloud Development Kit (AWS CDK)pour simplifier la création et le déploiement TypeScript du code. Les deux outils utilisent esbuild pour transpiler le TypeScript code dans. JavaScript

Tenez compte des éléments suivants lorsque vous utilisez esbuild :

- Il y a plusieurs TypeScript mises en garde.
- Vous devez configurer vos paramètres de TypeScript transpilation pour qu'ils correspondent à l'environnement d'exécution Node.js que vous prévoyez d'utiliser. Pour plus d'informations, consultez Cible dans la documentation d'esbuild. Pour un exemple de fichier tsconfig.json qui montre comment cibler une version spécifique de Node.js prise en charge par Lambda, reportez-vous au référentiel. TypeScript GitHub
- esbuild n'effectue pas de vérifications du type. Pour vérifier les types, utilisez le compilateur `tsc`. Exécutez `tsc -noEmit` ou ajoutez un paramètre `"noEmit"` dans votre fichier tsconfig.json comme illustré dans l'exemple suivant. Cela permet de ne pas `tsc` émettre de JavaScript fichiers. Après avoir vérifié les types, utilisez esbuild pour convertir les TypeScript fichiers en JavaScript.

Example tsconfig.json

```
{
 "compilerOptions": {
   "target": "es2020",
   "strict": true,
   "preserveConstEnums": true,
   "noEmit": true,
   "sourceMap": false,
   "module":"commonjs",
   "moduleResolution":"node",
   "esModuleInterop": true,
   "skipLibCheck": true,
   "forceConsistentCasingInFileNames": true,
```

```
    "isolatedModules": true,
  },
  "exclude": ["node_modules", "**/*.test.ts"]
}
```

# Définissez le gestionnaire de fonctions Lambda dans TypeScript

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Rubriques


- Notions de base sur le gestionnaire Typescript
- Utilisation d'async/await
- Utilisation de callbacks
- Utilisation de types pour l'objet d'événement
- Meilleures pratiques en matière de code pour les fonctions Typescript Lambda


## Notions de base sur le gestionnaire Typescript

Example TypeScript gestionnaire

Cet exemple de fonction enregistre le contenu de l'objet événement et renvoie l'emplacement des journaux. Notez ce qui suit :

- Avant d'utiliser ce code dans une fonction Lambda, vous devez ajouter le package @types/aws-lambda en tant que dépendance de développement. Ce package contient les définitions de type pour Lambda. Quand `@types/aws-lambda` est installé, l'instruction `import` (`import ... from 'aws-lambda'`) importe les définitions de type. Il n'importe pas le `aws-lambda` NPM package, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub
- Dans cet exemple, le gestionnaire est un module ES et doit être désigné comme tel dans votre fichier `package.json` ou en utilisant l'extension de fichier `.mjs`. Pour plus d'informations, consultez Désignation d'un gestionnaire de fonctions en tant que module ES.

```
import { Handler } from 'aws-lambda';

export const handler: Handler = async (event, context) => {
    console.log('EVENT: \n' + JSON.stringify(event, null, 2));
    return context.logStreamName;
};
```

L'exécution transmet des arguments à la méthode du gestionnaire. Le premier argument est l'objet `event`, qui contient les informations de l'appelant. L'invocateur transmet ces informations sous forme de chaîne JSON formatée lorsqu'il appelle Invoke, et le moteur d'exécution les convertit en objet. Lorsqu'un AWS service invoque votre fonction, la structure de l'événement varie d'un service à l'autre. Avec TypeScript, nous vous recommandons d'utiliser des annotations de type pour l'objet d'événement. Pour de plus amples informations, veuillez consulter Utilisation de types pour l'objet d'événement.

Le deuxième argument est l'objet de contexte, qui contient des informations sur l'invocation, la fonction et l'environnement d'exécution. Dans l'exemple précédent, la fonction obtient le nom du flux de journaux de l'objet de contexte et le renvoie au mécanisme d'invocation.

Vous pouvez également utiliser un argument callback, qui est une fonction que vous pouvez appeler dans les gestionnaires non asynchrones pour envoyer une réponse. Nous vous recommandons d'utiliser async/await plutôt que des callbacks. Async/await améliore la lisibilité, la gestion des erreurs et l'efficacité. Pour plus d'informations sur les différences entre async/await et les callbacks, consultez Utilisation de callbacks.

## Utilisation d'async/await

Si votre code exécute une tâche asynchrone, utilisez le modèle async/await pour vous assurer que le gestionnaire termine son exécution. Async/await est un moyen concis et lisible d'écrire du code asynchrone dans Node.js, sans avoir besoin de callbacks imbriqués ou de promesses en chaîne. Avec async/await, vous pouvez écrire du code qui se lit comme du code synchrone, tout en étant asynchrone et non bloquant.

Le mot-clé `async` marque une fonction comme étant asynchrone, et le mot-clé `await` met en pause l'exécution de la fonction jusqu'à ce qu'une `Promise` soit résolue.

Example TypeScript fonction — asynchrone

Cet exemple utilise `fetch`, qui est disponible dans l'exécution `nodejs18.x`. Notez ce qui suit :

- Avant d'utiliser ce code dans une fonction Lambda, vous devez ajouter le package @types/aws-lambda en tant que dépendance de développement. Ce package contient les définitions de type pour Lambda. Quand `@types/aws-lambda` est installé, l'instruction `import` (`import ... from 'aws-lambda'`) importe les définitions de type. Il n'importe pas le `aws-lambda` NPM package, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub

- Dans cet exemple, le gestionnaire est un module ES et doit être désigné comme tel dans votre fichier `package.json` ou en utilisant l'extension de fichier `.mjs`. Pour plus d'informations, consultez Désignation d'un gestionnaire de fonctions en tant que module ES.

```
import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
const url = 'https://aws.amazon.com/';
export const lambdaHandler = async (event: APIGatewayProxyEvent):
 Promise<APIGatewayProxyResult> => {
    try {
        // fetch is available with Node.js 18
        const res = await fetch(url);
        return {
            statusCode: res.status,
            body: JSON.stringify({
                message: await res.text(),
            }),
        };
    } catch (err) {
        console.log(err);
        return {
            statusCode: 500,
            body: JSON.stringify({
                message: 'some error happened',
            }),
        };
    }
};
```

## Utilisation de callbacks

Nous vous recommandons d'utiliser async/await pour déclarer le gestionnaire de fonctions au lieu d'utiliser des callbacks. Async/await est un meilleur choix pour plusieurs raisons :

- Lisibilité : le code Async/await est plus facile à lire et à comprendre que le code callback, qui peut rapidement devenir difficile à suivre et entraîner l'enfer du callback.
- Débogage et gestion des erreurs : le débogage du code basé sur des callbacks peut être difficile. La pile d'appels peut devenir difficile à suivre et les erreurs peuvent facilement se perdre. Avec async/await, vous pouvez utiliser des blocs try/catch pour gérer les erreurs.

- Efficacité : les callbacks nécessitent souvent de basculer entre différentes parties du code. Async/await permet de réduire le nombre de changements de contexte, ce qui se traduit par un code plus efficace.

Lorsque vous utilisez des callbacks dans votre gestionnaire, la fonction continue à s'exécuter jusqu'à ce que la boucle d'événement soit vide ou que la fonction s'arrête. La réponse n'est pas envoyée à l'appelant tant que toutes les tâches d'événement de boucle ne sont pas terminées. Si la fonction expire, une erreur est renvoyée à la place. Vous pouvez configurer le moteur d'exécution pour envoyer la réponse immédiatement en définissant le contexte. callbackWaitsForEmptyEventLoopà faux.

La fonction de rappel accepte deux arguments : un(e) `Error` et une réponse. L'objet de réponse doit être compatible avec `JSON.stringify`.

Example TypeScript fonction avec rappel

Cet exemple de fonction reçoit un événement d'Amazon API Gateway, enregistre l'événement et les objets de contexte, puis renvoie une réponse à API Gateway. Notez ce qui suit :

- Avant d'utiliser ce code dans une fonction Lambda, vous devez ajouter le package @types/aws-lambda en tant que dépendance de développement. Ce package contient les définitions de type pour Lambda. Quand `@types/aws-lambda` est installé, l'instruction `import` (`import ... from 'aws-lambda'`) importe les définitions de type. Il n'importe pas le `aws-lambda` NPM package, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub
- Dans cet exemple, le gestionnaire est un module ES et doit être désigné comme tel dans votre fichier `package.json` ou en utilisant l'extension de fichier `.mjs`. Pour plus d'informations, consultez Désignation d'un gestionnaire de fonctions en tant que module ES.

```
import { Context, APIGatewayProxyCallback, APIGatewayEvent } from 'aws-lambda';

export const lambdaHandler = (event: APIGatewayEvent, context: Context, callback:
 APIGatewayProxyCallback): void => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    callback(null, {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
```

```
        }),
    });
};
```

## Utilisation de types pour l'objet d'événement

Nous vous recommandons de ne pas utiliser le type Any (Tout) pour les arguments du gestionnaire et du type de retour, car vous perdez la possibilité de vérifier les types. Générez plutôt un événement à l'aide de la AWS Serverless Application Model CLI commande sam local generate-event ou utilisez une définition open source du package @types /aws-lambda.

Génération d'un événement à l'aide de la commande sam local generate-event (génération d'événement local sam)

1. Générez un événement proxy Amazon Simple Storage Service (Amazon S3).

   ```
   sam local generate-event s3 put >> S3PutEvent.json
   ```

2. Utilisez l'utilitaire quicktype pour générer des définitions de type à partir du PutEventfichier .json S3.

   ```
   npm install -g quicktype
   quicktype S3PutEvent.json -o S3PutEvent.ts
   ```

3. Utilisez les types générés dans votre code.

   ```
   import { S3PutEvent } from './S3PutEvent';

   export const lambdaHandler = async (event: S3PutEvent): Promise<void> => {
     event.Records.map((record) => console.log(record.s3.object.key));
   };
   ```

Génération d'un événement à l'aide d'une définition Open source à partir du pack @types/aws-lambda

1. Ajoutez le pack @types/aws-lambda en tant que dépendance de développement.

   ```
   npm install -D @types/aws-lambda
   ```

2. Utilisez les types dans votre code.

```
import { S3Event } from "aws-lambda";

export const lambdaHandler = async (event: S3Event): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

## Meilleures pratiques en matière de code pour les fonctions Typescript Lambda

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire. Dans Node.js, vous obtenez ce qui suit :

```
exports.myHandler = function(event, context, callback) {
 var foo = event.foo;
 var bar = event.bar;
 var result = MyLambdaFunction (foo, bar);

 callback(null, result);
}

function MyLambdaFunction (foo, bar) {
 // MyLambdaFunction logic here
}
```

- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour les environnements d'exécution Node.js et Python, ceux-ci incluent le AWS SDKs. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.

- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation.

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles

que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.

- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Déployez TypeScript du code transpilé dans Lambda avec des archives de fichiers .zip

Avant de pouvoir déployer TypeScript du code surAWS Lambda, vous devez le transpiler dans JavaScript. Cette page explique trois méthodes pour créer et déployer TypeScript du code sur Lambda avec des archives de fichiers .zip :

- Utilisation d'AWS Serverless Application Model (AWS SAM)
- Utilisation du AWS Cloud Development Kit (AWS CDK)
- Utilisation de la AWS Command Line Interface (AWS CLI) et esbuild

AWS SAMet AWS CDK simplifiez la création et le déploiement TypeScript des fonctions. La spécification du modèle AWS SAM fournit une syntaxe simple et propre pour décrire les fonctions Lambda, les API, les autorisations, les configurations et les événements qui composent votre application sans serveur. Le AWS CDK vous permet de créer des applications fiables, évolutives et rentables dans le cloud, avec la puissance expressive considérable d'un langage de programmation. Le AWS CDK est destiné aux utilisateurs AWS modérément à très expérimentés. Le AWS CDK et le AWS SAM utilisent esbuild pour transpiler le TypeScript code dans. JavaScript

## Utilisation AWS SAM pour déployer TypeScript du code sur Lambda

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d' TypeScript application Hello World à l'aide duAWS SAM. Cette application implémente un backend API de base. Elle se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une demande GET au point de terminaison API Gateway, la fonction Lambda est invoquée. La fonction renvoie un message `hello world`.

> **Note**
>
> AWS SAMutilise esbuild pour créer des fonctions Lambda Node.js à TypeScript partir du code. le support d'esbuild est actuellement en version préliminaire publique. Au cours de la version préliminaire publique, la prise en charge d'esbuild peut faire l'objet de modifications incompatibles.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS CLI version 2
- CLI AWS SAM version 1.75 ou ultérieure.
- Node.js 18.x

Déployer un exemple d'application AWS SAM

1. Initialisez l'application à l'aide du TypeScript modèle Hello World.

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs18.x
```

2. (Facultatif) L'exemple d'application inclut des configurations pour les outils couramment utilisés, tels que ESLint pour la validation de code et de Jest pour les tests unitaires. Pour exécuter des commandes de test et de validation :

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Créez l'application.

```
cd sam-app
sam build
```

4. Déployez l'application.

```
sam deploy --guided
```

5. Suivez les invites à l'écran. Répondez avec `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.
6. La sortie affiche le point de terminaison de l'API REST. Ouvrez le point de terminaison dans un navigateur pour tester la fonction. Vous devriez voir la réponse suivante :

```
{"message":"hello world"}
```

7. Il s'agit d'un point de terminaison d'API public accessible par Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

## Utilisation du AWS CDK pour déployer du TypeScript code sur Lambda

Suivez les étapes ci-dessous pour créer et déployer un exemple d' TypeScript application à l'aide duAWS CDK. Cette application implémente un backend API de base. Elle se compose d'un point de terminaison API Gateway et d'une fonction Lambda. Lorsque vous envoyez une demande GET au point de terminaison API Gateway, la fonction Lambda est invoquée. La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS CLI version 2
- AWS CDK version 2
- Node.js 18.x
- Docker ou esbuild

Déployer un exemple d'application AWS CDK

1. Créez un répertoire de projets pour votre nouvelle application.

```
mkdir hello-world
cd hello-world
```

2. Initialisez l'application.

```
cdk init app --language typescript
```

3. Ajoutez le pack @types/aws-lambda en tant que dépendance de développement. Ce package contient les définitions de type pour Lambda.

```
npm install -D @types/aws-lambda
```

4. Ouvrez le répertoire lib. Vous devriez voir un fichier appelé hello-world-stack.ts. Créez deux nouveaux fichiers dans ce répertoire : hello-world.function.ts et hello-world.ts.

5. Ouvrez hello-world.function.ts et ajoutez le code suivant au fichier. Il s'agit du code de la fonction Lambda.

> **Note**
>
> L'instruction `import` importe les définitions de type depuis @types /aws-lambda. Elle n'importe pas le package NPM `aws-lambda`, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    return {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
        }),
    };
};
```

6. Ouvrez hello-world.ts et ajoutez le code suivant au fichier. Il contient la NodejsFunction construction, qui crée la fonction Lambda, et la LambdaRestApi construction, qui crée l'API REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';

export class HelloWorld extends Construct {
```

```
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function');
    new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
  }
}
```

Le composant `NodejsFunction` suppose les éléments suivants par défaut :

- Votre gestionnaire de fonctions se nomme `handler`.
- Le fichier .ts qui contient le code de fonction (hello-world.function.ts) se trouve dans le même répertoire que le fichier .ts qui contient le composant (hello-world.ts). Le composant utilise l'ID du composant (« hello-world ») et le nom du fichier du gestionnaire Lambda (« fonction ») pour trouver le code de la fonction. Par exemple, si le code de votre fonction se trouve dans un fichier nommé hello-world.my-fonction.ts, le fichier hello-world.ts doit référencer le code de la fonction comme suit :

  ```
  const helloFunction = new NodejsFunction(this, 'my-function');
  ```

Vous pouvez modifier ce comportement et configurer d'autres paramètres esbuild. Pour de plus amples informations, veuillez consulter Configuration d'esbuild dans les références de l'API du AWS CDK.

7. Ouvrez hello-world-stack.ts. C'est le code qui définit votre pile de AWS CDK. Remplacez le code par ce qui suit :

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

8. à partir du répertoire `hello-world` contenant votre fichier `cdk.json`, déployez votre application.

```
cdk deploy
```

9. Le AWS CDK crée et conditionne la fonction Lambda à l'aide d'esbuild, puis la déploie dans l'environnement d'exécution Lambda. La sortie affiche le point de terminaison de l'API REST. Ouvrez le point de terminaison dans un navigateur pour tester la fonction. Vous devriez voir la réponse suivante :

```
{"message":"hello world"}
```

   Il s'agit d'un point de terminaison d'API public accessible par Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

## Utilisation de AWS CLI et esbuild pour déployer TypeScript du code sur Lambda

L'exemple suivant montre comment transpiler et déployer TypeScript du code sur Lambda à l'aide d'esbuild et de. esbuild produit JavaScript un fichier avec toutes AWS CLI les dépendances. Il s'agit du seul fichier que vous devez ajouter à l'archive .zip.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS CLI version 2
- Node.js 18.x
- Un rôle d'exécution pour votre fonction Lambda
- Pour les utilisateurs de Windows, un utilitaire de fichier zip tel que 7zip.

Déployez un exemple de fonction

1. Sur votre machine locale, créez un répertoire de projets pour votre nouvelle fonction.
2. Créez un nouveau projet Node.js avec npm ou un gestionnaire de packs de votre choix.

```
npm init
```

3. Ajoutez les packs @types/aws-lambda et esbuild en tant que dépendances de développement. Le package `@types/aws-lambda` contient les définitions de type pour Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Créez un nouveau fichier nommé index.ts. Ajoutez le code suivant au nouveau fichier. Il s'agit du code de la fonction Lambda. La fonction renvoie un message `hello world`. La fonction ne crée aucune ressource API Gateway.

> **Note**
>
> L'instruction `import` importe les définitions de type depuis @types /aws-lambda. Elle n'importe pas le package NPM `aws-lambda`, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
      statusCode: 200,
      body: JSON.stringify({
          message: 'hello world',
      }),
   };
};
```

5. Ajoutez un script de génération au fichier package.json. Ceci permet de configurer esbuild pour créer automatiquement le pack de déploiement .zip. Pour plus d'informations, consultez Scripts de génération dans la documentation d'esbuild.

   Linux and MacOS

```
"scripts": {
```

```
  "prebuild": "rm -rf dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && zip -r index.zip index.js*"
},
```

Windows

Dans cet exemple, la commande `"postbuild"` utilise l'utilitaire 7zip pour créer votre fichier .zip. Utilisez votre propre utilitaire Windows zip préféré et modifiez la commande si nécessaire.

```
"scripts": {
  "prebuild": "del /q dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && 7z a -tzip index.zip index.js*"
},
```

6. Créez le pack.

```
npm run build
```

7. Créez une fonction Lambda à l'aide du pack de déploiement .zip. Remplacez le texte surligné par l'Amazon Resource Name (ARN) de votre rôle d'exécution.

```
aws lambda create-function --function-name hello-world --runtime "nodejs18.x" --
role arn:aws:iam::123456789012:role/lambda-ex --zip-file "fileb://dist/index.zip"
 --handler index.handler
```

8. Exécutez un événement de test pour confirmer que la fonction renvoie la réponse suivante. Si vous souhaitez invoquez cette fonction à l'aide d'API Gateway, créez et configurez l'API REST.

```
{
  "statusCode": 200,
  "body": "{\"message\":\"hello world\"}"
}
```

# Déployez TypeScript du code transpilé dans Lambda avec des images de conteneur

Vous pouvez déployer votre TypeScript code sur une AWS Lambda fonction sous forme d'image de conteneur Node.js. AWS fournit des images de base pour Node.js afin de vous aider à créer l'image du conteneur. Ces images de base sont préchargées avec un environnement d'exécution du langage et d'autres composants nécessaires pour exécuter l'image sur Lambda. AWS fournit un Dockerfile pour chacune des images de base afin de vous aider à créer votre image de conteneur.

Si vous utilisez une image de base d'une communauté ou d'une entreprise privée, vous devez ajouter le client d'interface d'exécution Node.js (RIC) à l'image de base pour la rendre compatible avec Lambda.

Lambda fournit un émulateur d'interface d'exécution pour les tests locaux. Les images AWS de base de Node.js incluent l'émulateur d'interface d'exécution. Si vous utilisez une autre image de base, telle qu'une image Alpine Linux ou Debian, vous pouvez intégrer l'émulateur dans votre image ou l'installer sur votre ordinateur local.

## Utilisation d'une image de base Node.js pour créer et empaqueter le code de TypeScript fonction

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker
- Node.js 20x

Création d'une image à partir d'une image de base

Pour créer une image à partir d'une image de AWS base pour Lambda

1. Sur votre machine locale, créez un répertoire de projets pour votre nouvelle fonction.
2. Créez un nouveau projet Node.js avec `npm` ou un gestionnaire de packs de votre choix.

```
npm init
```

3. Ajoutez les packs @types/aws-lambda et esbuild en tant que dépendances de développement. Le package `@types/aws-lambda` contient les définitions de type pour Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Ajoutez un script de création au fichier `package.json`.

```
  "scripts": {
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js"
}
```

5. Créez un nouveau fichier appelé `index.ts`. Ajoutez l'exemple de code suivant au nouveau fichier. Il s'agit du code de la fonction Lambda. La fonction renvoie un message `hello world`.

> **Note**
>
> L'instruction `import` importe les définitions de type depuis @types /aws-lambda. Il n'importe pas le `aws-lambda` NPM package, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    return {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
        }),
    };
};
```

6. Créez un nouveau Dockerfile avec la configuration suivante :

- Définissez la `FROM` propriété sur URI l'image de base.
- Définissez l'argument `CMD` pour spécifier le gestionnaire de la fonction Lambda.

L'exemple de fichier Docker suivant utilise une génération en plusieurs étapes. La première étape consiste à transspirer le TypeScript code en. JavaScript La deuxième étape produit une image de conteneur contenant uniquement des JavaScript fichiers et des dépendances de production.

Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:20 as builder
WORKDIR /usr/app
COPY package.json index.ts  ./
RUN npm install
RUN npm run build

FROM public.ecr.aws/lambda/nodejs:20
WORKDIR ${LAMBDA_TASK_ROOT}
COPY --from=builder /usr/app/dist/* ./
CMD ["index.handler"]
```

7. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une

fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

1. Démarrez votre image Docker à l'aide de la commande docker run. Dans cet exemple, `docker-image` est le nom de l'image et `test` est la balise.

   ```
   docker run --platform linux/amd64 -p 9000:8080 --read-only docker-image:test
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

2. À partir d'une nouvelle fenêtre de terminal, publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:testest` le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
```

```
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

# Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions TypeScript

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'invocation, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `getRemainingTimeInMillis()` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `functionName` – Nom de la fonction Lambda.
- `functionVersion` – Version de la fonction.
- `invokedFunctionArn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `memoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `awsRequestId` – Identifiant de la demande d'invocation.
- `logGroupName` – Groupe de journaux pour la fonction.
- `logStreamName` – Flux de journal de l'instance de fonction.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
  - `cognitoIdentityId` – Identité Amazon Cognito authentifiée.
  - `cognitoIdentityPoolId` – Groupe d'identités Amazon Cognito ayant autorisé l'invocation.
- `clientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`

  - `env.platform`
  - `env.make`
  - `env.model`
  - `env.locale`
  - `Custom` – Personnalisez les valeurs qui sont définies par l'application client.
- `callbackWaitsForEmptyEventLoop` – Définissez ce paramètre sur faux pour envoyer la réponse dès que le rappel s'exécute, au lieu d'attendre que la boucle d'événement Node.js soit vide. Si ce paramètre est faux, les événements restants continueront de s'exécuter lors de la prochaine invocation.

Vous pouvez utiliser le package npm @types/aws-lambda pour travailler avec l'objet de contexte.

Example Fichier index.ts

Dans l'exemple suivant, la fonction enregistre des informations de contexte et renvoie l'emplacement des journaux.

> **Note**
>
> Avant d'utiliser ce code dans une fonction Lambda, vous devez ajouter le package @types/aws-lambda en tant que dépendance de développement. Ce package contient les définitions de type pour Lambda. Quand `@types/aws-lambda` est installé, l'instruction `import` (`import ... from 'aws-lambda'`) importe les définitions de type. Il n'importe pas le `aws-lambda` NPM package, qui est un outil tiers indépendant. Pour plus d'informations, consultez aws-lambda dans le référentiel. DefinitelyTyped GitHub

```
import { Context } from 'aws-lambda';
export const lambdaHandler = async (event: string, context: Context): Promise<string>
 => {
  console.log('Remaining time: ', context.getRemainingTimeInMillis());
  console.log('Function name: ', context.functionName);
  return context.logStreamName;
};
```

# Enregistrez et surveillez les fonctions TypeScript Lambda

AWS Lambda surveille automatiquement les fonctions Lambda et envoie des entrées de journal à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des détails sur chaque invocation et d'autres sorties provenant du code de votre fonction au flux de journaux. Pour plus d'informations sur CloudWatch les journaux, consultezUtilisation des CloudWatch journaux et journaux avec Lambda.

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur l'objet console. Pour une journalisation plus détaillée, vous pouvez utiliser n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`.

Sections

- Utilisation d'outils de journalisation et de bibliothèques
- Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour la journalisation structurée
- Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour la journalisation structurée
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console

## Utilisation d'outils de journalisation et de bibliothèques

Powertools for AWS Lambda (TypeScript) est une boîte à outils destinée aux développeurs qui permet de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs. L'utilitaire Logger fournit un enregistreur optimisé pour Lambda qui inclut des informations supplémentaires sur le contexte de fonction pour toutes vos fonctions avec une sortie structurée en JSON. Utilisez cet utilitaire pour effectuer les opérations suivantes :

- Capturer les champs clés du contexte Lambda, démarrer à froid et structurer la sortie de la journalisation sous forme de JSON
- Journaliser les événements d'invocation Lambda lorsque cela est demandé (désactivé par défaut)
- Imprimer tous les journaux uniquement pour un pourcentage d'invocations via l'échantillonnage des journaux (désactivé par défaut)
- Ajouter des clés supplémentaires au journal structuré à tout moment

- Utiliser un formateur de journaux personnalisé (Apportez votre propre formateur) pour produire des journaux dans une structure compatible avec le RFC de journalisation de votre organisation

## Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d' TypeScript application Hello World avec des modules Powertools for AWS Lambda (TypeScript) intégrés à l'aide du AWS SAM. Cette application met en œuvre un backend API de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Elle se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une requête GET au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Node.js 18.x ou version ultérieure
- AWS CLI version 2
- AWS SAM CLI version 1.75 ou ultérieure. Si vous disposez d'une ancienne version de la AWS SAM CLI, reportez-vous à la section Mise à niveau de la AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du TypeScript modèle Hello World.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x
```

2. Créez l'application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez l'URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoquez le point de terminaison de l'API :

```
curl <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

La sortie du journal se présente comme suit :

```
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.552000
 START RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Version: $LATEST
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.594000
 2022-08-31T09:33:10.557Z 70693159-7e94-4102-a2af-98a6343fb8fb
 INFO {"_aws":{"Timestamp":1661938390556,"CloudWatchMetrics":
[{"Namespace":"sam-app","Dimensions":[["service"]],"Metrics":
[{"Name":"ColdStart","Unit":"Count"}]}]},"service":"helloWorld","ColdStart":1}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.595000
 2022-08-31T09:33:10.595Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
```

```
 {"level":"INFO","message":"This is an INFO log - sending HTTP 200 - hello world
 response","service":"helloWorld","timestamp":"2022-08-31T09:33:10.594Z"}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.655000
 2022-08-31T09:33:10.655Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
 {"_aws":{"Timestamp":1661938390655,"CloudWatchMetrics":[{"Namespace":"sam-
app","Dimensions":[["service"]],"Metrics":[]}]},"service":"helloWorld"}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000 END
 RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000
 REPORT RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Duration: 201.55 ms Billed
 Duration: 202 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 252.42
 ms
XRAY TraceId: 1-630f2ad5-1de22b6d29a658a466e7ecf5 SegmentId: 567c116658fbf11a
 Sampled: true
```

8. Il s'agit d'un point de terminaison d'API public accessible par Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

## Gestion de la conservation des journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation après laquelle les journaux CloudWatch sont automatiquement supprimés. Pour configurer la conservation des journaux, ajoutez ce qui suit à votre AWS SAM modèle :

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
    # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

# Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d' TypeScript application Hello World avec des modules Powertools for AWS Lambda (TypeScript) intégrés à l'aide du AWS CDK. Cette application met en œuvre un backend API de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Elle se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une requête GET au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Node.js 18.x ou version ultérieure
- AWS CLI version 2
- AWS CDK version 2
- AWS SAM CLI version 1.75 ou ultérieure. Si vous disposez d'une ancienne version de la AWS SAM CLI, reportez-vous à la section Mise à niveau de la AWS SAM CLI.

Déployer un exemple d' AWS CDK application

1. Créez un répertoire de projets pour votre nouvelle application.

```
mkdir hello-world
cd hello-world
```

2. Initialisez l'application.

```
cdk init app --language typescript
```

3. Ajoutez le pack @types/aws-lambda en tant que dépendance de développement.

```
npm install -D @types/aws-lambda
```

4. Installez l'utilitaire Powertools Logger (français non garanti).

```
npm install @aws-lambda-powertools/logger
```

5. Ouvrez le répertoire lib. Vous devriez voir un fichier appelé hello-world-stack.ts. Créez deux nouveaux fichiers dans ce répertoire : hello-world.function.ts et hello-world.ts.
6. Ouvrez hello-world.function.ts et ajoutez le code suivant au fichier. Il s'agit du code de la fonction Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Logger } from '@aws-lambda-powertools/logger';
const logger = new Logger();

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  logger.info('This is an INFO log - sending HTTP 200 - hello world response');
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

7. Ouvrez hello-world.ts et ajoutez le code suivant au fichier. Il contient la NodejsFunction construction qui crée la fonction Lambda, configure les variables d'environnement pour Powertools et définit la durée de conservation des journaux à une semaine. Il inclut également la LambdaRestApi construction qui crée l'API REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { RetentionDays } from 'aws-cdk-lib/aws-logs';
import { CfnOutput } from 'aws-cdk-lib';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        Powertools_SERVICE_NAME: 'helloWorld',
        LOG_LEVEL: 'INFO',
      },
```

```
      logRetention: RetentionDays.ONE_WEEK,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}
```

8. Ouvrez hello-world-stack.ts. C'est le code qui définit votre pile de AWS CDK. Remplacez le code par ce qui suit :

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Revenez au répertoire du projet.

```
cd hello-world
```

10. Déployez votre application.

```
cdk deploy
```

11. Obtenez l'URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Invoquez le point de terminaison de l'API :

```
curl <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

13. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

La sortie du journal se présente comme suit :

```
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.047000
 START RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Version: $LATEST
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.050000 {
"level": "INFO",
"message": "This is an INFO log - sending HTTP 200 - hello world response",
"service": "helloWorld",
"timestamp": "2022-08-31T14:48:37.048Z",
"xray_trace_id": "1-630f74c4-2b080cf77680a04f2362bcf2"
}
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000 END
 RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000
 REPORT RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Duration: 34.60 ms Billed
 Duration: 35 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 173.48
 ms
```

14. Il s’agit d’un point de terminaison d’API public accessible par Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
cdk destroy
```

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (***your-function-name***/aws/lambda/).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

# TypeScript Code de suivi dans AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des trois SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- AWS X-Ray SDKpour Node.js — Et SDK pour générer et envoyer des données de trace à X-Ray.
- Powertools for AWS Lambda (TypeScript) — Une boîte à outils pour les développeurs permettant de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections


- Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour le traçage
- Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour le traçage
- Interprétation d'un suivi X-Ray

# Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS SAM pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d' TypeScript application Hello World avec des modules Powertools for AWS Lambda (TypeScript) intégrés à l'aide du AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Node.js 18.x ou version ultérieure
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du TypeScript modèle Hello World.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x --no-tracing
```

2. Créez l’application.

```
cd sam-app && sam build
```

3. Déployez l’application.

```
sam deploy --guided
```

4. Suivez les invites à l’écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l’expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
XRay Event [revision 1] at (2023-01-31T11:29:40.527000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.483s)
  - 0.425s - sam-app/Prod [HTTP: 200]
    - 0.422s - Lambda [HTTP: 200]
  - 0.406s - sam-app-HelloWorldFunction-Xyzv11a1bcde [HTTP: 200]
  - 0.172s - sam-app-HelloWorldFunction-Xyzv11a1bcde
    - 0.179s - Initialization
    - 0.112s - Invocation
      - 0.052s - ## app.lambdaHandler
        - 0.001s - ### MySubSegment
    - 0.059s - Overhead
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

## Utilisation de Powertools pour AWS Lambda (TypeScript) et AWS CDK pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d' TypeScript application Hello World avec des modules Powertools for AWS Lambda (TypeScript) intégrés à l'aide du AWS CDK. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Node.js 18.x ou version ultérieure
- AWS CLI version 2
- AWS CDK version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS Cloud Development Kit (AWS CDK) application

1. Créez un répertoire de projets pour votre nouvelle application.

```
mkdir hello-world
```

```
cd hello-world
```

2. Initialisez l'application.

```
cdk init app --language typescript
```

3. Ajoutez le pack @types/aws-lambda en tant que dépendance de développement.

```
npm install -D @types/aws-lambda
```

4. Installez l'utilitaire Powertools Tracer.

```
npm install @aws-lambda-powertools/tracer
```

5. Ouvrez le répertoire lib. Vous devriez voir un fichier appelé hello-world-stack.ts. Créez deux nouveaux fichiers dans ce répertoire : hello-world.function.ts et hello-world.ts.

6. Ouvrez hello-world.function.ts et ajoutez le code suivant au fichier. Il s'agit du code de la fonction Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Tracer } from '@aws-lambda-powertools/tracer';
const tracer = new Tracer();

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  // Get facade segment created by Lambda
  const segment = tracer.getSegment();

  // Create subsegment for the function and set it as active
  const handlerSegment = segment.addNewSubsegment(`## ${process.env._HANDLER}`);
  tracer.setSegment(handlerSegment);

  // Annotate the subsegment with the cold start and serviceName
  tracer.annotateColdStart();
  tracer.addServiceNameAnnotation();

  // Add annotation for the awsRequestId
  tracer.putAnnotation('awsRequestId', context.awsRequestId);
  // Create another subsegment and set it as active
  const subsegment = handlerSegment.addNewSubsegment('### MySubSegment');
  tracer.setSegment(subsegment);
  let response: APIGatewayProxyResult = {
```

```
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
  // Close subsegments (the Lambda one is closed automatically)
  subsegment.close(); // (### MySubSegment)
  handlerSegment.close(); // (## index.handler)

  // Set the facade segment as active again (the one created by Lambda)
  tracer.setSegment(segment);
  return response;
};
```

7. Ouvrez hello-world.ts et ajoutez le code suivant au fichier. Il contient la NodejsFunction construction qui crée la fonction Lambda, configure les variables d'environnement pour Powertools et définit la durée de conservation des journaux à une semaine. Il inclut également la LambdaRestApi construction, qui crée le RESTAPI.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { CfnOutput } from 'aws-cdk-lib';
import { Tracing } from 'aws-cdk-lib/aws-lambda';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        POWERTOOLS_SERVICE_NAME: 'helloWorld',
      },
      tracing: Tracing.ACTIVE,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}
```

8. Ouvrez hello-world-stack.ts. C'est le code qui définit votre pile de AWS CDK. Remplacez le code par ce qui suit :

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Déployez votre application.

```
cd ..
cdk deploy
```

10. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Appelez le API point de terminaison :

```
curl <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

12. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.449s)
  - 0.350s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde [HTTP: 200]
```

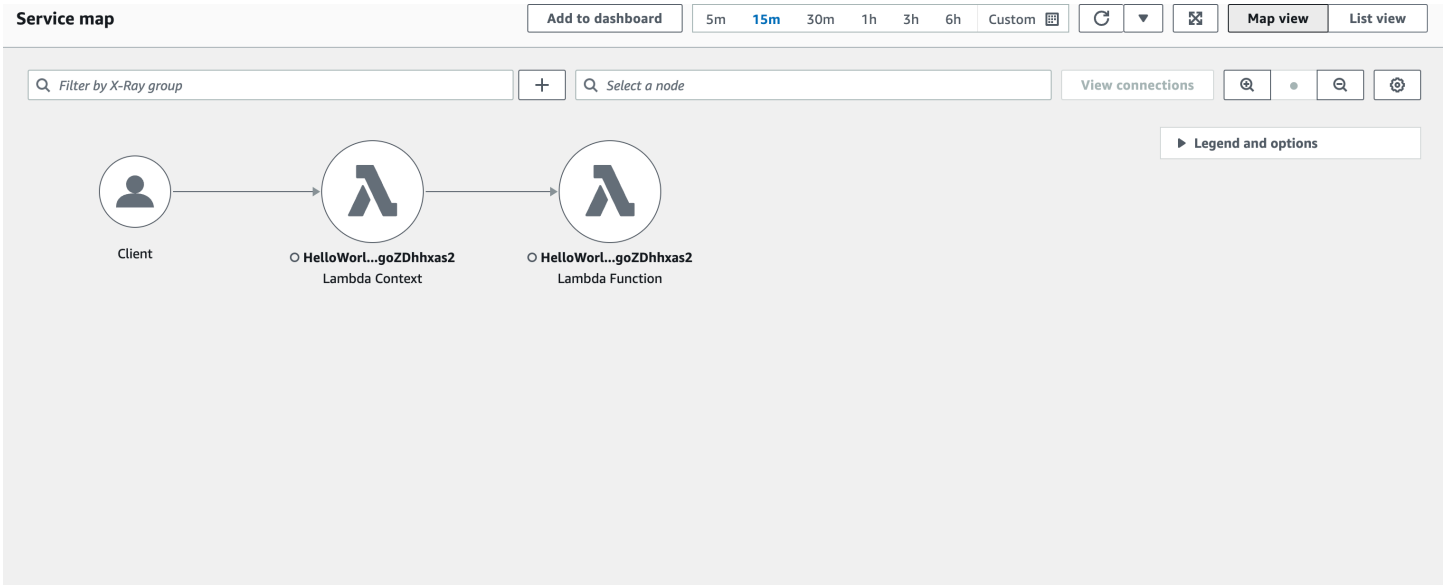
```
  - 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde
    - 0.169s - Initialization
    - 0.058s - Invocation
      - 0.055s - ## index.handler
        - 0.000s - ### MySubSegment
    - 0.099s - Overhead
```

13. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
cdk destroy
```

## Interprétation d'un suivi X-Ray

Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. La mappe de trace X-Ray fournit des informations sur votre application et tous ses composants. L'exemple suivant montre une trace à partir de l'exemple d'application :



# Création de fonctions Lambda avec Python

Vous pouvez exécuter du code Python dans AWS Lambda. Lambda fournit des runtimes pour Python qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement qui inclut le SDK pour Python (Boto3), avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez. Pour en savoir plus sur les SDK versions incluses dans les environnements d'exécution Python, consultezthe section called “Versions incluses dans Runtime SDK”.

Lambda prend en charge les runtimes Python suivants.

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Python 3.12 | `python3.12` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.11 | `python3.11` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.10 | `python3.10` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.9 | `python3.9` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Python 3.8 | `python3.8` | Amazon Linux 2 | 14 octobre 2024 | 28 février 2025 | 31 mars 2025 |

Pour créer une fonction Python

1. Ouvrez la console Lambda.
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :
   - Nom de la fonction : saisissez le nom de la fonction.

- Exécution : sélectionnez Python 3.12.

4. Sélectionnez Créer une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Event name (Nom de l'événement), saisissez **`test`**.
7. Sélectionnez Save Changes.
8. Pour invoquer la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `lambda_function`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'éditeur de code intégré. Choisissez Save pour enregistrer les changements. Ensuite, pour exécuter votre code, choisissez Test.

Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs. La fonction runtime envoie les détails de chaque appel à CloudWatch Logs. Il relaie tous les journaux que votre fonction génère pendant l'invocation. Si votre fonction renvoie une erreur, Lambda met en forme l'erreur et la renvoie à l'appelant.

Rubriques


- Versions incluses dans Runtime SDK
- Format de la réponse
- Arrêt progressif pour les extensions
- Définir le gestionnaire de fonctions Lambda en Python
- Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python
- Déployer des fonctions Lambda en Python avec des images conteneurs
- Utilisation de couches pour les fonctions Lambda en Python
- Utilisation de l'objet de contexte Lambda pour récupérer les informations d'une fonction Python
- Enregistrez et surveillez les fonctions Lambda en Python
- Test de la fonction AWS Lambda dans Python
- Instrumentation du code Python dans AWS Lambda

## Versions incluses dans Runtime SDK

La version AWS SDK incluse dans le runtime Python dépend de la version d'exécution et de votre Région AWS. Pour trouver la version SDK incluse dans le moteur d'exécution que vous utilisez, créez une fonction Lambda avec le code suivant.

```
import boto3
import botocore

def lambda_handler(event, context):
   print(f'boto3 version: {boto3.__version__}')
   print(f'botocore version: {botocore.__version__}')
```

## Format de la réponse

Dans les environnements d'exécution Python 3.12 et versions ultérieures, les fonctions renvoient des caractères Unicode dans le cadre de leur JSON réponse. Les environnements d'exécution Python antérieurs renvoyaient des séquences échappées pour les caractères Unicode dans les réponses. Par exemple, dans Python 3.11, si vous renvoyez une chaîne Unicode telle que »こんにちは«, il échappe aux caractères Unicode et renvoie « \ u3053 \ u3093 \ u306b \ u3061 \ u306f ». Le runtime Python 3.12 renvoie l'original »こんにちは".

L'utilisation de réponses Unicode réduit la taille des réponses Lambda, ce qui facilite l'adaptation de réponses plus importantes à la taille de charge utile maximale de 6 Mo pour les fonctions synchrones. Dans l'exemple précédent, la version échappée est de 32 octets, contre 17 octets pour la chaîne Unicode.

Lors de la mise à niveau vers Python 3.12, vous devrez peut-être ajuster votre code pour tenir compte du nouveau format de réponse. Si l'appelant s'attend à une chaîne Unicode échappée, vous devez soit ajouter du code à la fonction de renvoi pour échapper l'Unicode manuellement, soit ajuster l'appelant pour qu'il gère le retour Unicode.

## Arrêt progressif pour les extensions

Les environnements d’exécution Python 3.12 et versions ultérieures offrent des fonctionnalités d'arrêt progressif améliorées pour les fonctions dotées d'extensions externes. Quand Lambda arrête l'exécution, il envoie un signal `SIGTERM` à l'exécution, puis un événement `SHUTDOWN` à chaque extension externe enregistrée. Vous pouvez intercepter le signal `SIGTERM` dans votre fonction

Lambda et nettoyer les ressources telles que les connexions de base de données créées par la fonction.

Pour en savoir plus sur le cycle de vie de l'environnement d'exécution, veuillez consulter Comprendre le cycle de vie de l'environnement d'exécution Lambda. Pour des exemples d'utilisation de l'arrêt progressif avec des extensions, consultez le GitHub référentiel AWS Samples.

# Définir le gestionnaire de fonctions Lambda en Python

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Vous pouvez utiliser la syntaxe générale suivante lors de la création d'un gestionnaire de fonctions dans Python :

```
def handler_name(event, context):
    ...
    return some_value
```

Rubriques


- Dénomination
- Comment ça marche
- Renvoi d'une valeur
- Exemples
- Bonnes pratiques en matière de code pour les fonctions Lambda en Python


## Dénomination

Le nom du gestionnaire de fonction Lambda spécifié au moment de la création d'une fonction Lambda est dérivé des éléments suivants :

- Nom du fichier dans lequel se trouve la fonction du gestionnaire Lambda.
- Nom de la fonction du gestionnaire Python.

Un gestionnaire de fonction peut porter n'importe quel nom mais le nom par défaut dans la console Lambda est `lambda_function.lambda_handler`. Ce nom de gestionnaire de fonction reflète le nom de la fonction (`lambda_handler`) et le fichier dans lequel le code du gestionnaire est stocké dans (`lambda_function.py`).

Si vous créez une fonction dans la console en utilisant un nom de fichier ou un nom de gestionnaire de fonction différent, vous devez modifier le nom du gestionnaire par défaut.

Pour modifier le nom du gestionnaire de fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Cliquez sur l'onglet Code.
3. Faites défiler l'écran jusqu'au volet Paramètres d'exécution et choisissez Modifier.
4. Dans Gestionnaire, saisissez le nouveau nom de votre gestionnaire de fonction.
5. Choisissez Save (Enregistrer).

## Comment ça marche

Lorsque Lambda invoque votre gestionnaire de fonction, le runtime Lambda transmet deux arguments au gestionnaire de fonction :

- Le premier argument est l'objet d'événement. Un événement est un document JSON formaté qui contient des données à traiter par une fonction Lambda. Le runtime Lambda convertit l'événement en objet et la transmet au code de votre fonction. Il est généralement du type `dict` Python. Il peut également être de type `list`, `str`, `int`, `float` ou `NoneType`.

  L'objet d'événement contient des informations provenant du service invoquant. Lorsque vous invoquez une fonction, vous déterminez la structure et le contenu de l'événement. Lorsqu'un AWS service appelle votre fonction, il définit la structure de l'événement. Pour plus d'informations sur les événements liés AWS aux services, consultezInvoquer Lambda avec des événements provenant d'autres services AWS.
- Le deuxième argument est l'objet de contexte. Un objet contexte est transmis à votre fonction par Lambda au niveau du runtime. Cet objet contient des méthodes et des propriétés qui fournissent des informations sur l'invocation, la fonction et l'environnement de runtime.

## Renvoi d'une valeur

Un gestionnaire peut renvoyer une valeur. Ce qu'il advient de la valeur renvoyée dépend du type d'invocation et du service qui a invoqué la fonction. Par exemple :

- Si vous utilisez le type d'`RequestResponse`invocation, tel queInvoquer une fonction Lambda de manière synchrone, AWS Lambda renvoie le résultat de l'appel de la fonction Python au client invoquant la fonction Lambda (dans la HTTP réponse à la demande d'appel, sérialisée dans). JSON Par exemple, la console AWS Lambda utilise le type d'invocation `RequestResponse`. Dès

lors, lorsque vous invoquez la fonction par le biais de la console, cette dernière affiche la valeur renvoyée.

- Si le gestionnaire renvoie des objets qui ne peuvent pas être sérialisées par `json.dumps`, le runtime renvoie une erreur.
- Si le gestionnaire renvoie `None`, comme le font implicitement les fonctions Python sans une instruction `return`, le runtime renvoie `null`.
- Si vous exécutez le type d'invocation `Event` (une invocation asynchrone), la valeur sera ignorée.

> **Note**
>
> Dans Python 3.9 et versions ultérieures, Lambda inclut l'invocation dans requestId la réponse d'erreur.

## Exemples

La section suivante comprend des exemples de fonctions Python que vous pouvez utiliser avec Lambda. Si vous utilisez la console Lambda pour créer votre fonction, vous n'avez pas besoin d'attacher un fichier d'archive .zip pour exécuter les fonctions dans cette section. Ces fonctions utilisent des bibliothèques Python standard, qui sont incluses avec le runtime Lambda que vous avez sélectionné. Pour de plus amples informations, veuillez consulter ???.

### Renvoi d'un message

L'exemple suivant montre une fonction appelée `lambda_handler`. La fonction accepte l'entrée de l'utilisateur (prénom et nom), et renvoie un message contenant les données de l'événement qu'elle a reçu en tant qu'entrée.

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

Vous pouvez utiliser les données d'événement suivantes afin d'invoquer la fonction :

```
{
    "first_name": "John",
```

```
    "last_name": "Smith"
}
```

La réponse montre les données d'événement transmises en tant qu'entrée :

```
{
    "message": "Hello John Smith!"
}
```

## Analyse d'une réponse

L'exemple suivant montre une fonction appelée `lambda_handler`. La fonction utilise les données d'événement transmises par Lambda au niveau du runtime. Il analyse la variable d'environnement `AWS_REGION` renvoyée dans la JSON réponse.

```
import os
import json

def lambda_handler(event, context):
    json_region = os.environ['AWS_REGION']
    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps({
            "Region ": json_region
        })
    }
```

Vous pouvez utiliser n'importe quelle donnée d'événement afin d'invoquer la fonction :

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Les runtimes Lambda définissent plusieurs variables d'environnement lors de l'initialisation. Pour plus d'informations sur les variables d'environnement renvoyées dans la réponse lors du runtime, consultez Utiliser les variables d'environnement Lambda pour configurer les valeurs dans le code.

La fonction de cet exemple dépend de la réussite de la réponse (in`200`) de l'InvokeAPI. Pour plus d'informations sur le API statut Invoke, consultez la syntaxe de la réponse Invoke.

### Renvoi d'un calcul

L'exemple suivant montre une fonction appelée `lambda_handler`. La fonction accepte l'entrée de l'utilisateur et renvoie un calcul à l'utilisateur. Pour plus d'informations sur cet exemple, consultez le aws-doc-sdk-examples GitHub référentiel.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    ...
    result = None
    action = event.get('action')
    if action == 'increment':
        result = event.get('number', 0) + 1
        logger.info('Calculated result of %s', result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {'result': result}
    return response
```

Vous pouvez utiliser les données d'événement suivantes afin d'invoquer la fonction :

```
{
    "action": "increment",
    "number": 3
}
```

## Bonnes pratiques en matière de code pour les fonctions Lambda en Python

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire. Par exemple, en Python, cela peut ressembler à ceci :

```
def lambda_handler(event, context):
    foo = event['foo']
    bar = event['bar']

    result = my_lambda_function(foo, bar)

def my_lambda_function(foo, bar):
    // MyLambdaFunction logic here
```

- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour les environnements d'exécution Node.js et Python, ceux-ci incluent le AWS SDKs. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation.

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion

persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.

- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.
- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python

Le code de votre AWS Lambda fonction comprend un fichier .py contenant le code du gestionnaire de votre fonction, ainsi que les packages et modules supplémentaires dont dépend votre code. Pour déployer ce code de fonction vers Lambda, vous utilisez un package de déploiement. Ce package peut être une archive de fichier .zip ou une image de conteneur. Pour plus d'informations sur l'utilisation d'images de conteneur avec Python, consultez la page Déployer des fonctions Lambda Python avec des images de conteneur.

Pour créer votre package de déploiement sous forme d'archive de fichier .zip, vous pouvez utiliser l'utilitaire d'archivage .zip intégré à votre outil de ligne de commande, ou tout autre utilitaire .zip tel que 7zip. Les exemples présentés dans les sections suivantes supposent que vous utilisez un outil `zip` de ligne de commande dans un environnement Linux ou macOS. Pour utiliser les mêmes commandes sous Windows, vous pouvez installer le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et de Bash.

Notez que Lambda utilise des autorisations de POSIX fichier. Vous devrez donc peut-être définir des autorisations pour le dossier du package de déploiement avant de créer l'archive de fichiers .zip.

Rubriques


- Dépendances d'exécution dans Python
- Création d'un package de déploiement .zip sans dépendances
- Création d'un package de déploiement .zip avec dépendances
- Chemin de recherche des dépendances et bibliothèques incluses dans l'exécution
- Utilisation des dossiers __pycache__
- Création de packages de déploiement .zip avec des bibliothèques natives
- Création et mise à jour de fonctions Lambda Python à l'aide de fichiers .zip


## Dépendances d'exécution dans Python

Pour les fonctions Lambda qui utilisent l'exécution Python, une dépendance peut être n'importe quel package ou module Python. Lorsque vous déployez votre fonction à l'aide d'une archive .zip, vous pouvez soit ajouter ces dépendances à votre fichier .zip avec votre code de fonction, soit utiliser une couche Lambda. Une couche est un fichier .zip séparé qui peut contenir du code supplémentaire et

d'autres contenus. Pour en savoir plus sur l'utilisation des couches Lambda en Python, consultez. the section called “Couches”

Les environnements d'exécution Lambda Python incluent le AWS SDK for Python (Boto3) et ses dépendances. Lambda fournit SDK dans l'environnement d'exécution des scénarios de déploiement dans lesquels vous ne pouvez pas ajouter vos propres dépendances. Ces scénarios incluent la création de fonctions dans la console à l'aide de l'éditeur de code intégré ou à l'utilisation de fonctions intégrées dans AWS Serverless Application Model (AWS SAM) ou de AWS CloudFormation modèles.

Lambda met régulièrement à jour les bibliothèques d'exécution Python afin d'inclure les dernières mises à jour et correctifs de sécurité. Si votre fonction utilise la version du Boto3 SDK incluse dans le runtime mais que votre package de déploiement inclut des SDK dépendances, cela peut entraîner des problèmes de désalignement des versions. Par exemple, votre package de déploiement peut inclure la SDK dépendance urllib3. Lorsque Lambda met à jour le SDK dans l'environnement d'exécution, des problèmes de compatibilité entre la nouvelle version du moteur d'exécution et la version d'urllib3 de votre package de déploiement peuvent entraîner l'échec de votre fonction.

> **Important**
>
> Pour conserver un contrôle total sur vos dépendances et éviter d'éventuels problèmes d'alignement de versions, nous vous recommandons d'ajouter toutes les dépendances de votre fonction à votre package de déploiement, même si des versions de celles-ci sont incluses dans l'exécution Lambda. Cela inclut le Boto3SDK.

Pour savoir quelle version de SDK for Python (Boto3) est incluse dans le runtime que vous utilisez, consultez. the section called “Versions incluses dans Runtime SDK”

Dans le cadre du modèle de responsabilité partagée AWS, vous êtes responsable de la gestion de toutes les dépendances dans les packages de déploiement de vos fonctions. Cela inclut l’application de mises à jour et de correctifs de sécurité. Pour mettre à jour les dépendances dans le package de déploiement de votre fonction, créez d’abord un nouveau fichier .zip, puis chargez-le sur Lambda. Pour plus d’informations, consultez Création d'un package de déploiement .zip avec dépendances et Création et mise à jour de fonctions Lambda Python à l'aide de fichiers .zip.

## Création d'un package de déploiement .zip sans dépendances

Si le code de votre fonction ne comporte aucune dépendance, votre fichier .zip contient uniquement le fichier .py contenant le code du gestionnaire de votre fonction. Utilisez votre utilitaire zip préféré

pour créer un fichier .zip avec votre fichier .py à la racine. Si le fichier .py ne se trouve pas à la racine de votre fichier .zip, Lambda ne sera pas en mesure d'exécuter votre code.

Pour savoir comment déployer votre fichier .zip pour créer une nouvelle fonction Lambda ou mettre à jour une fonction Lambda existante, veuillez consulter la rubrique Création et mise à jour de fonctions Lambda Python à l'aide de fichiers .zip.

## Création d'un package de déploiement .zip avec dépendances

Si le code de votre fonction dépend de packages ou de modules supplémentaires, vous pouvez soit ajouter ces dépendances à votre fichier .zip avec votre code de fonction, soit utiliser une couche Lambda. Les instructions de cette section vous indiquent comment inclure vos dépendances dans votre package de déploiement .zip. Pour que Lambda puisse exécuter votre code, le fichier .py contenant le code de votre gestionnaire et toutes les dépendances de votre fonction doit être installé à la racine du fichier .zip.

Supposons que votre code de fonction soit enregistré dans un fichier nommé `lambda_function.py`. Les exemples de CLI commandes suivants créent un fichier .zip nommé `my_deployment_package.zip` contenant votre code de fonction et ses dépendances. Vous pouvez soit installer vos dépendances directement dans un dossier du répertoire de votre projet, soit utiliser un environnement virtuel Python.

Pour créer le package de déploiement (répertoire du projet)

1. Accédez au répertoire du projet qui contient votre fichier de code source `lambda_function.py`. Dans cet exemple, le répertoire est nommé `my_function`.

   ```
   cd my_function
   ```

2. Créez un nouveau répertoire nommé « package » dans lequel vous installerez vos dépendances.

   ```
   mkdir package
   ```

   Notez que pour un package de déploiement .zip, Lambda s'attend à ce que votre code source et ses dépendances se trouvent tous à la racine du fichier .zip. Cependant, l'installation de dépendances directement dans le répertoire de votre projet peut introduire un grand nombre de nouveaux fichiers et dossiers et rendre la navigation IDE difficile. Vous créez ici un répertoire `package` distinct pour séparer vos dépendances de votre code source.

3. Installez les dépendances dans le répertoire `package`. L'exemple ci-dessous installe le Boto3 à partir du Python Package Index à SDK l'aide de pip. Si le code de votre fonction utilise des packages Python que vous avez créés vous-même, enregistrez-les dans le répertoire `package`.

   ```
   pip install --target ./package boto3
   ```

4. Créez un fichier .zip avec les bibliothèques installées à la racine.

   ```
   cd package
   zip -r ../my_deployment_package.zip .
   ```

   Cela génère un fichier `my_deployment_package.zip` dans votre répertoire de projet.

5. Ajoutez le fichier lambda_function.py à la racine du fichier .zip.

   ```
   cd ..
   zip my_deployment_package.zip lambda_function.py
   ```

   Votre fichier .zip doit avoir une structure de répertoires horizontale, avec le code du gestionnaire de votre fonction et tous vos dossiers de dépendances installés à la racine comme suit.

   ```
   my_deployment_package.zip
   |- bin
   |  |-jp.py
   |- boto3
   |  |-compat.py
   |  |-data
   |  |-docs
   ...
   |- lambda_function.py
   ```

   Si le fichier .py contenant le code du gestionnaire de votre fonction ne se trouve pas à la racine de votre fichier .zip, Lambda ne sera pas en mesure d'exécuter votre code.

Pour créer le package de déploiement (environnement virtuel)

1. Créez et activez un environnement virtuel dans le répertoire de votre projet. Dans cet exemple, le répertoire du projet est nommé `my_function`.

   ```
   ~$ cd my_function
   ```

```
~/my_function$ python3.12 -m venv my_virtual_env
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Installez les bibliothèques requises à l'aide de pip. L'exemple suivant installe le Boto3 SDK

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Utilisez `pip show` pour trouver l'emplacement dans votre environnement virtuel où pip a installé vos dépendances.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

   Le dossier dans lequel pip installe vos bibliothèques peut être nommé `site-packages` ou `dist-packages`. Ce dossier peut se trouver dans le répertoire `lib/python3.x` ou `lib64/python3.x` (où python3.x représente la version de Python que vous utilisez).

4. Désactivation de l'environnement virtuel

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Accédez au répertoire contenant les dépendances que vous avez installées avec pip et créez un fichier .zip dans le répertoire de votre projet avec les dépendances installées à la racine. Dans cet exemple, pip a installé vos dépendances dans le répertoire `my_virtual_env/lib/python3.12/site-packages`.

```
~/my_function$ cd my_virtual_env/lib/python3.12/site-packages
~/my_function/my_virtual_env/lib/python3.12/site-packages$ zip -r ../../../../
my_deployment_package.zip .
```

6. Accédez à la racine du répertoire de votre projet où se trouve le fichier .py contenant le code de votre gestionnaire et ajoutez ce fichier à la racine de votre package .zip. Dans cet exemple, votre fichier de code de fonction est nommé `lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.12/site-packages$ cd ../../../../
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

# Chemin de recherche des dépendances et bibliothèques incluses dans l'exécution

Lorsque vous utilisez une instruction `import` dans votre code, l'exécution Python recherche les répertoires dans son chemin de recherche jusqu'à ce qu'elle trouve le module ou le package. Par défaut, le premier emplacement dans lequel recherche l'exécution est le répertoire dans lequel votre package de déploiement .zip est décompressé et monté (`/var/task`). Si vous ajoutez une version d'une bibliothèque incluse dans l'exécution dans votre package de déploiement, votre version aura la priorité sur la version incluse dans l'exécution. Les dépendances de votre package de déploiement ont également la priorité sur les dépendances des couches.

Lorsque vous ajoutez une dépendance à une couche, Lambda l'extrait vers `/opt/python/lib/python3.x/site-packages` (où `python3.x` représente la version de l'exécution que vous utilisez) ou `/opt/python`. Dans le chemin de recherche, ces répertoires ont la priorité sur les répertoires contenant les bibliothèques incluses dans l'exécution et les bibliothèques installées par pip (`/var/runtime` et `/var/lang/lib/python3.x/site-packages`). Les bibliothèques des couches de fonctions ont donc la priorité sur les versions incluses dans l'exécution.

> **Note**
>
> Dans le runtime géré et l'image de base de Python 3.11, le AWS SDK et ses dépendances sont installés dans le `/var/lang/lib/python3.11/site-packages` répertoire.

Vous pouvez voir le chemin de recherche complet de votre fonction Lambda en ajoutant l'extrait de code suivant.

```
import sys

search_path = sys.path
print(search_path)
```

> **Note**
>
> Comme les dépendances de votre package ou de vos couches de déploiement ont priorité sur les bibliothèques incluses dans l'exécution, cela peut entraîner des problèmes de désalignement des versions si vous incluez une SDK dépendance telle que urllib3 dans votre package sans l'inclure également. SDK Si vous déployez votre propre version d'une

dépendance de Boto3, vous devez également déployer Boto3 en tant que dépendance dans votre package de déploiement. Nous vous recommandons de regrouper toutes les dépendances de votre fonction, même si des versions de celles-ci sont incluses dans l'exécution.

Vous pouvez également ajouter des dépendances dans un dossier distinct au sein de votre package .zip. Par exemple, vous pouvez ajouter une version du Boto3 dans un dossier SDK de votre package .zip appelé. `common` Lorsque votre package .zip est décompressé et monté, ce dossier est placé dans le répertoire `/var/task`. Pour utiliser dans votre code une dépendance provenant d'un dossier de votre package de déploiement .zip, utilisez une instruction `import from`. Par exemple, pour utiliser une version de Boto3 provenant d'un dossier nommé `common` dans votre package .zip, utilisez l'instruction suivante.

```
from common import boto3
```

## Utilisation des dossiers __pycache__

Nous vous recommandons de ne pas inclure de dossiers `__pycache__` dans le package de déploiement de votre fonction. Le bytecode Python compilé sur une machine de génération avec une architecture ou un système d'exploitation différent peut ne pas être compatible avec l'environnement d'exécution Lambda.

## Création de packages de déploiement .zip avec des bibliothèques natives

Si votre fonction utilise uniquement des packages et des modules Python purs, vous pouvez utiliser la commande `pip install` pour installer vos dépendances sur n'importe quelle machine de génération locale et créer votre fichier .zip. De nombreuses bibliothèques Python populaires, y compris NumPy Pandas, ne sont pas du Python pur et contiennent du code écrit en C ou C++. Lorsque vous ajoutez des bibliothèques contenant du code C/C++ à votre package de déploiement, vous devez créer votre package correctement pour vous assurer qu'il est compatible avec l'environnement d'exécution Lambda.

La plupart des packages disponibles sur le Python Package Index (PyPI) sont disponibles sous forme de « wheels » (fichiers .whl). Un fichier .whl est un type de ZIP fichier qui contient une distribution intégrée avec des binaires précompilés pour un système d'exploitation et une architecture de jeu d'instructions particuliers. Pour rendre votre package de déploiement compatible avec Lambda, vous

devez installer le fichier wheel pour les systèmes d'exploitation Linux et l'architecture de l'ensemble d'instructions de votre fonction.

Certains packages peuvent uniquement être disponibles en tant que distributions sources. Pour ces packages, vous devez compiler et créer vous-même les composants C/C++.

Pour connaître les distributions disponibles pour le package dont vous avez besoin, procédez comme suit :

1. Recherchez le nom du package sur la page principale du Python Package Index.
2. Choisissez la version du package que vous souhaitez utiliser.
3. Choisissez Télécharger les fichiers.

## Utilisation des distributions intégrées (fichiers wheels)

Pour télécharger un fichier wheel compatible avec Lambda, utilisez l'option `--platform` pip.

Si votre fonction Lambda utilise l'architecture de l'ensemble d'instructions x86_64, exécutez la commande `pip install` suivante pour installer un fichier wheel compatible dans votre répertoire `package`. Remplacez `--python 3.x` par la version d'exécution Python que vous utilisez.

```
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Si votre fonction utilise l'architecture de l'ensemble d'instructions arm64, exécutez la commande suivante. Remplacez `--python 3.x` par la version d'exécution Python que vous utilisez.

```
pip install \
--platform manylinux2014_aarch64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

### Utilisation des distributions sources

Si votre package n'est disponible que sous la forme d'une distribution source, vous devez créer vous-même les bibliothèques C/C++. Pour rendre votre package compatible avec l'environnement d'exécution Lambda, vous devez le créer dans un environnement qui utilise le même système d'exploitation Amazon Linux 2. Vous pouvez le faire en créant votre package dans une instance Amazon EC2 Linux.

Pour savoir comment lancer une instance Amazon EC2 Linux et s'y connecter, consultez Tutoriel : Commencer à utiliser les instances Amazon EC2 Linux dans le Guide de EC2 l'utilisateur Amazon pour les instances Linux.

## Création et mise à jour de fonctions Lambda Python à l'aide de fichiers .zip

Après avoir créé votre package de déploiement .zip, vous pouvez l'utiliser pour créer une nouvelle fonction Lambda ou mettre à jour une fonction existante. Vous pouvez déployer votre package .zip à l'aide de la console Lambda, AWS Command Line Interface du, et du Lambda. API Vous pouvez également créer et mettre à jour des fonctions Lambda à l'aide de l' AWS Serverless Application Model (AWS SAM) et de AWS CloudFormation.

La taille maximale d'un package de déploiement .zip pour Lambda est de 250 Mo (décompressé). Notez que cette limite s'applique à la taille combinée de tous les fichiers que vous chargez, y compris les couches Lambda.

Le runtime Lambda a besoin d'une autorisation pour lire les fichiers de votre package de déploiement. Dans la notation octale des autorisations Linux, Lambda a besoin de 644 autorisations pour les fichiers non exécutables (rw-r--r--) et de 755 autorisations () pour les répertoires et les fichiers exécutables. rwxr-xr-x

Sous Linux et macOS, utilisez la commande `chmod` pour modifier les autorisations de fichiers sur les fichiers et les répertoires de votre package de déploiement. Par exemple, pour accorder les autorisations appropriées à un fichier exécutable, exécutez la commande suivante.

```
chmod 755 <filepath>
```

Pour modifier les autorisations relatives aux fichiers dans Windows, voir Définir, afficher, modifier ou supprimer des autorisations sur un objet dans la documentation Microsoft Windows.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de la console

Pour créer une nouvelle fonction, vous devez d'abord créer la fonction dans la console, puis charger votre archive .zip. Pour mettre à jour une fonction existante, ouvrez la page de votre fonction, puis suivez la même procédure pour ajouter votre fichier .zip mis à jour.

Si votre fichier .zip fait moins de 50 Mo, vous pouvez créer ou mettre à jour une fonction en chargeant le fichier directement à partir de votre ordinateur local. Pour les fichiers .zip de plus de 50 Mo, vous devez d'abord charger votre package dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS Management Console, consultez Getting started with Amazon S3. Pour télécharger des fichiers à l'aide de AWS CLI, voir Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Pour créer une nouvelle fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Choisissez Créer à partir de zéro.
3. Sous Informations de base, procédez comme suit :
   a. Pour Nom de la fonction, saisissez le nom de la fonction.
   b. Pour Exécution, sélectionnez l'exécution que vous souhaitez utiliser.
   c. (Facultatif) Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour votre fonction. L'architecture par défaut est x86_64. Assurez-vous que le package de déploiement .zip pour votre fonction est compatible avec l'architecture de l'ensemble d'instructions que vous sélectionnez.
4. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou en utiliser un existant.

5. Sélectionnez Create function (Créer une fonction). Lambda crée une fonction de base « Hello world » à l'aide de l'exécution de votre choix.

Pour charger une archive .zip à partir de votre ordinateur local (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger le fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez Fichier .zip.
5. Pour charger un fichier .zip, procédez comme suit :
   a. Sélectionnez Charger, puis choisissez votre fichier .zip dans le sélecteur de fichiers.
   b. Choisissez Ouvrir.
   c. Choisissez Save (Enregistrer).

Pour charger une archive .zip depuis un compartiment Amazon S3 (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger un nouveau fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez l'emplacement Amazon S3.
5. Collez le lien Amazon S3 URL de votre fichier .zip et choisissez Enregistrer.

## Mise à jour des fonctions du fichier .zip à l'aide de l'éditeur de code de la console

Pour certaines fonctions avec des packages de déploiement .zip, vous pouvez utiliser l'éditeur de code intégré de la console Lambda pour mettre à jour le code de votre fonction directement. Pour utiliser cette fonctionnalité, votre fonction doit répondre aux critères suivants :

- Votre fonction doit utiliser l'une des exécutions des langages interprétés (Python, Node.js ou Ruby).
- Le package de déploiement de votre fonction doit être inférieur à 3 Mo.

Le code des fonctions avec les packages de déploiement d'images de conteneurs ne peut pas être édité directement dans la console.

Pour mettre à jour le code de fonction à l'aide de l'éditeur de code de la console

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, sélectionnez votre fichier de code source et modifiez-le dans l'éditeur de code intégré.
4. Lorsque vous avez fini de modifier votre code, choisissez Déployer pour enregistrer vos modifications et mettre à jour votre fonction.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide du AWS CLI

Vous pouvez utiliser l'AWS CLI pour créer une nouvelle fonction ou pour mettre à jour une fonction existante à l'aide d'un fichier .zip. Utilisez la fonction de création et update-function-codeles commandes pour déployer votre package .zip. Si votre fichier .zip est inférieur à 50 Mo, vous pouvez charger le package .zip à partir d'un emplacement de fichier sur votre machine de génération locale. Pour les fichiers plus volumineux, vous devez charger votre package .zip à partir d'un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Si vous chargez votre fichier .zip depuis un compartiment Amazon S3 à l'aide de AWS CLI, le compartiment doit se trouver au même endroit Région AWS que votre fonction.

Pour créer une nouvelle fonction à l'aide d'un fichier .zip avec le AWS CLI, vous devez spécifier les éléments suivants :

- Le nom de votre fonction (`--function-name`)
- L'exécution de votre fonction (`--runtime`)
- Le nom de ressource Amazon (ARN) du rôle d'exécution de votre fonction (`--role`)
- Le nom de la méthode du gestionnaire dans votre code de fonction (`--handler`)

Vous devez également indiquer l'emplacement de votre fichier .zip. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda create-function --function-name myFunction \
--runtime python3.12 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez l'option `--code` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `S3ObjectVersion` pour les objets soumis à la gestion des versions.

```
aws lambda create-function --function-name myFunction \
--runtime python3.12 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Pour mettre à jour une fonction existante à l'aide duCLI, vous devez spécifier le nom de votre fonction à l'aide du `--function-name` paramètre. Vous devez également spécifier l'emplacement du fichier .zip que vous souhaitez utiliser pour mettre à jour votre code de fonction. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez les options `--s3-bucket` et `--s3-key` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `--s3-object-version` pour les objets soumis à la gestion des versions.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide du Lambda API

Pour créer et mettre à jour des fonctions à l'aide d'une archive de fichiers .zip, effectuez les API opérations suivantes :

- CreateFunction
- UpdateFunctionCode

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS SAM

The AWS Serverless Application Model (AWS SAM) est une boîte à outils qui permet de rationaliser le processus de création et d'exécution d'applications sans serveur sur AWS. Vous définissez les ressources de votre application dans un JSON modèle YAML or et vous utilisez l'interface de ligne de AWS SAM commande (AWS SAM CLI) pour créer, empaqueter et déployer vos applications. Lorsque vous créez une fonction Lambda à partir d'un AWS SAM modèle, elle crée AWS SAM automatiquement un package de déploiement ou une image de conteneur .zip avec le code de votre fonction et les dépendances que vous spécifiez. Pour en savoir plus sur l'utilisation des fonctions Lambda AWS SAM pour créer et déployer des fonctions Lambda, consultez Getting started with AWS SAM dans le Guide du AWS Serverless Application Model développeur.

Vous pouvez également l'utiliser AWS SAM pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip existante. Pour créer une fonction Lambda à l'aide de AWS SAM, vous pouvez enregistrer votre fichier .zip dans un compartiment Amazon S3 ou dans un dossier local sur votre machine de génération. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS SAM modèle, la `AWS::Serverless::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l'aide d'une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `CodeUri`- défini sur l'Amazon S3URI, le chemin d'accès au dossier local ou à l'FunctionCodeobjet du code de fonction
- `Runtime` : défini sur votre exécution choisie

Ainsi AWS SAM, si votre fichier .zip est supérieur à 50 Mo, vous n'avez pas besoin de le télécharger au préalable dans un compartiment Amazon S3. AWS SAM peut télécharger des packages .zip

jusqu'à la taille maximale autorisée de 250 Mo (décompressés) à partir d'un emplacement sur votre machine de compilation locale.

Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS SAM, voir :Serverless AWS : :Function dans le manuel du développeur.AWS SAM

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip. Pour créer une fonction Lambda à partir d'un fichier .zip, vous devez d'abord charger votre fichier dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Pour les environnements d'exécution Node.js et Python, vous pouvez également fournir du code source intégré dans votre AWS CloudFormation modèle. AWS CloudFormation crée ensuite un fichier .zip contenant votre code lorsque vous créez votre fonction.

### Utilisation d'un fichier .zip existant

Dans votre AWS CloudFormation modèle, la `AWS::Lambda::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l'aide d'une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `Code` : saisissez le nom du compartiment Amazon S3 et le nom du fichier .zip dans les champs `S3Bucket` et `S3Key`
- `Runtime` : défini sur votre exécution choisie

### Création d'un fichier .zip à partir du code en ligne

Vous pouvez déclarer des fonctions simples écrites en Python ou en Node.js en ligne dans un AWS CloudFormation modèle. Le code étant intégré dans YAML ouJSON, vous ne pouvez pas ajouter de dépendances externes à votre package de déploiement. Cela signifie que votre fonction doit utiliser la AWS SDK version incluse dans le runtime. Les exigences du modèle, telles que le fait de devoir échapper à certains caractères, compliquent également l'utilisation de vos fonctionnalités de vérification IDE de syntaxe et de complétion de code. Cela signifie que votre modèle peut nécessiter

des tests supplémentaires. En raison de ces limitations, la déclaration de fonctions en ligne convient mieux à un code très simple qui ne change pas fréquemment.

Pour créer un fichier .zip à partir du code en ligne pour les exécutions Node.js et Python, définissez les propriétés suivantes dans la ressource `AWS::Lambda::Function` de votre modèle :

- `PackageType` : défini sur `Zip`
- `Code` : saisissez votre code de fonction dans le champ `ZipFile`
- `Runtime` : défini sur votre exécution choisie

Le fichier .zip AWS CloudFormation généré ne peut pas dépasser 4 Mo. Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS CloudFormation, voir :Lambda AWS : :Function dans le guide de l'utilisateur.AWS CloudFormation

# Déployer des fonctions Lambda en Python avec des images conteneurs

Il existe trois méthodes pour créer une image de conteneur pour une fonction Lambda Python :

- Utilisation d'une image AWS de base pour Python

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Python dans l'image.

- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Python dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

Rubriques


- AWS images de base pour Python
- Utilisation d'une image AWS de base pour Python
- Utilisation d’une autre image de base avec le client d’interface d’exécution


## AWS images de base pour Python

AWS fournit les images de base suivantes pour Python :

| Balises | Environnement d’exécution | Système d’exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| 3,12 | Python 3.12 | Amazon Linux | Dockerfile pour Python 3.12 sur GitHub | Non prévu |
| 3,11 | Python 3.11 | Amazon Linux 2 | Dockerfile pour Python 3.11 sur GitHub | Non prévu |
| 3,10 | Python 3.10 | Amazon Linux 2 | Dockerfile pour Python 3.10 sur GitHub | Non prévu |
| 3.9 | Python 3.9 | Amazon Linux 2 | Dockerfile pour Python 3.9 sur GitHub | Non prévu |
| 3.8 | Python 3.8 | Amazon Linux 2 | Dockerfile pour Python 3.8 sur GitHub | 14 octobre 2024 |

ECRRéférentiel Amazon : gallery.ecr.aws/lambda/python

Les images de base de Python 3.12 et versions ultérieures sont basées sur l'image de conteneur minimale Amazon Linux 2023. Les images de base Python 3.8-3.11 sont basées sur l'image Amazon Linux 2. AL2Les images basées sur la version 023 présentent plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que. `glibc`

AL2Les images basées sur le format 023 sont utilisées `microdnf` (en lien symbolique comme`dnf`) comme gestionnaire de packages au lieu de`yum`, qui est le gestionnaire de packages par défaut dans Amazon Linux 2. `microdnf`est une implémentation autonome de. `dnf` Pour obtenir la liste des packages inclus dans les images AL2 basées sur la version 023, reportez-vous aux colonnes Conteneur minimal de la section Comparaison des packages installés sur les images de conteneurs Amazon Linux 2023. Pour plus d'informations sur les différences entre AL2 023 et Amazon Linux 2, consultez Présentation du runtime Amazon Linux 2023 AWS Lambda sur le blog AWS Compute.

> **Note**
>
> Pour exécuter des images AL2 basées sur 023 localement, y compris avec AWS Serverless Application Model (AWS SAM), vous devez utiliser Docker version 20.10.10 ou ultérieure.

## Chemin de recherche des dépendances dans les images de base

Lorsque vous utilisez une instruction `import` dans votre code, l'exécution Python recherche les répertoires dans son chemin de recherche jusqu'à ce qu'elle trouve le module ou le package. Par défaut, l'exécution recherche d'abord dans le répertoire `{LAMBDA_TASK_ROOT}`. Si vous ajoutez une version d'une bibliothèque incluse dans l'exécution dans votre image, votre version aura la priorité sur la version incluse dans l'exécution.

Les autres étapes du chemin de recherche dépendent de la version de l'image de base Lambda pour Python que vous utilisez :

- Python 3.11 et versions ultérieures : Les bibliothèques incluses dans l'exécution et les bibliothèques installées par pip sont installées dans le répertoire `/var/lang/lib/python3.11/site-packages`. Ce répertoire a la priorité sur `/var/runtime` dans le chemin de recherche. Vous pouvez le remplacer en SDK utilisant pip pour installer une version plus récente. Vous pouvez utiliser pip pour vérifier que le runtime inclus SDK et ses dépendances sont compatibles avec tous les packages que vous installez.
- Python 3.8-3.10 : Les bibliothèques incluses dans l'exécution sont installées dans le répertoire `/var/runtime`. Les bibliothèques installées par PIP sont installées dans le répertoire `/var/lang/lib/python3.x/site-packages`. Le répertoire `/var/runtime` a la priorité sur `/var/lang/lib/python3.x/site-packages` dans le chemin de recherche.

Vous pouvez voir le chemin de recherche complet de votre fonction Lambda en ajoutant l'extrait de code suivant.

```
import sys

search_path = sys.path
print(search_path)
```

## Utilisation d'une image AWS de base pour Python

### Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker (version minimale 20.10.10 pour les images de base de Python 3.12 et versions ultérieures)
- Python

### Création d'une image à partir d'une image de base

Pour créer une image de conteneur à partir d'une image AWS de base pour Python

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

   ```
   mkdir example
   cd example
   ```

2. Créez un nouveau fichier appelé `lambda_function.py`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

   Example Fonction Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Créez un nouveau fichier appelé `requirements.txt`. Si vous utilisez l'exemple de code de fonction de l'étape précédente, vous pouvez laisser le fichier vide, car il n'y a pas de dépendances. Sinon, répertoriez chaque bibliothèque requise. Par exemple, voici à quoi votre fichier `requirements.txt` doit ressembler si votre fonction utilise AWS SDK for Python (Boto3) :

Example requirements.txt

```
boto3
```

4. Créez un nouveau Dockerfile avec la configuration suivante :

   - Définissez la `FROM` propriété sur URIl'image de base.
   - Utilisez la `COPY` commande pour copier le code de fonction et les dépendances d'exécution dans une variable `{LAMBDA_TASK_ROOT}` d'environnement définie par Lambda.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. `USER` Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/python:3.12

# Copy requirements.txt
COPY requirements.txt ${LAMBDA_TASK_ROOT}

# Install the specified packages
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "lambda_function.handler" ]
```

5. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

1. Démarrez votre image Docker à l'aide de la commande docker run. Dans cet exemple, `docker-image` est le nom de l'image et `test` est la balise.

   ```
   docker run --platform linux/amd64 -p 9000:8080 --read-only docker-image:test
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

2. À partir d'une nouvelle fenêtre de terminal, publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

- `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
- Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilisation d’une autre image de base avec le client d’interface d’exécution

Si vous utilisez une image de base uniquement pour le système d’exploitation ou une autre image de base, vous devez inclure le client d'interface d'exécution dans votre image. Le client d’interface d’exécution étend leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l’interaction entre Lambda et votre code de fonction.

Installez le client d’interface d’exécution pour Python à l’aide du gestionnaire de packages pip :

```
pip install awslambdaric
```

Vous pouvez également télécharger le client d'interface d'exécution Python depuis GitHub.

L'exemple suivant montre comment créer une image de conteneur pour Python à l'aide d'une image non AWS basique. L'exemple Dockerfile utilise une image de base Python officielle. Le Dockerfile inclut le client d'interface d'exécution pour Python.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker
- Python

Création d'une image à partir d'une image de base alternative

Pour créer une image de conteneur à partir d'une image non AWS basique

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

   ```
   mkdir example
   cd example
   ```

2. Créez un nouveau fichier appelé `lambda_function.py`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

   Example Fonction Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Créez un nouveau fichier appelé `requirements.txt`. Si vous utilisez l'exemple de code de fonction de l'étape précédente, vous pouvez laisser le fichier vide, car il n'y a pas de dépendances. Sinon, répertoriez chaque bibliothèque requise. Par exemple, voici à quoi votre fichier `requirements.txt` doit ressembler si votre fonction utilise AWS SDK for Python (Boto3) :

Example requirements.txt

```
boto3
```

4. Créez un nouveau fichier Docker. Le Dockerfile suivant utilise une image de base Python officielle au lieu d'une image de base AWS. Le Dockerfile inclut le client d'interface d'exécution, ce qui rend l'image compatible avec Lambda. L'exemple de fichier Docker suivant utilise une génération en plusieurs étapes.

   - Définissez la propriété `FROM` sur l'image de base.
   - Définissez le `ENTRYPOINT` sur le module que vous souhaitez que le conteneur Docker exécute lorsqu'il démarre. Dans ce cas, le module est le client d'interface d'exécution.
   - Définissez le `CMD` sur le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM python:3.12 as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

# Install the function's dependencies
RUN pip install \
    --target ${FUNCTION_DIR} \
        awslambdaric

# Use a slim version of the base Python image to reduce the final image size
```

```
FROM python:3.12-slim

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdaric" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "lambda_function.handler" ]
```

5. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester l'image localement. Vous pouvez intégrer l'émulateur à votre image ou utiliser la procédure suivante pour l'installer sur votre machine locale.

Pour installer et exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Depuis le répertoire de votre projet, exécutez la commande suivante pour télécharger l'émulateur d'interface d'exécution (architecture x86-64) GitHub et l'installer sur votre machine locale.

Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
    curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
    chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Pour installer l'émulateur arm64, remplacez le GitHub référentiel de URL la commande précédente par le suivant :

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Pour installer l’émulateur arm64, remplacez `$downloadLink` par ce qui suit :

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
```

2. Démarrez votre image Docker à l’aide de la commande docker run. Notez ce qui suit :

   - `docker-image` est le nom de l’image et `test` est la balise.
   - `/usr/local/bin/python -m awslambdaric lambda_function.handler` est le `ENTRYPOINT` suivi du `CMD` depuis votre Dockerfile.

Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
 9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    --read-only \
    docker-image:test \
        /usr/local/bin/python -m awslambdaric lambda_function.handler
```

PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
 9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
--read-only `
docker-image:test `
    /usr/local/bin/python -m awslambdaric lambda_function.handler
```

Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

3. Publiez un événement au point de terminaison local.

Linux/macOS

Sous Linux et macOS, exécutez la commande `curl` suivante :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenez l'ID du conteneur.

```
docker ps
```

5. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

- `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
- Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Pour un exemple de création d'une image Python à partir d'une image de base Alpine, consultez Prise en charge des images conteneurs pour Lambda sur le blog AWS .

# Utilisation de couches pour les fonctions Lambda en Python

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration. La création d'une couche implique trois étapes générales :

1. Package le contenu de votre couche. Cela implique de créer une archive de fichiers .zip contenant les dépendances que vous souhaitez utiliser dans vos fonctions.
2. Créez la couche dans Lambda.
3. Ajoutez la couche à vos fonctions.

Cette rubrique contient des étapes et des conseils sur la manière d'empaqueter et de créer correctement une couche Lambda Python avec des dépendances de bibliothèque externes.

Rubriques


- Prérequis
- Compatibilité de la couche Python avec Amazon Linux
- Chemins de couche pour les environnements d'exécution Python
- Empaqueter le contenu de la couche
- Création de la couche
- Ajouter la couche à votre fonction
- Utilisation des distributions manylinux de roues


## Prérequis

Pour suivre les étapes décrites dans cette section, vous devez disposer des éléments suivants :

- Python 3.11 et le programme d'installation du package pip
- AWS Command Line Interface (AWS CLI) version 2

Tout au long de cette rubrique, nous faisons référence à l'`layer-python`exemple d'application sur le référentiel awsdocs GitHub . Cette application contient des scripts qui téléchargent les dépendances et génèrent les couches. L'application contient également les fonctions

correspondantes qui utilisent les dépendances des couches. Après avoir créé une couche, vous pouvez déployer et invoquer la fonction correspondante pour vérifier que tout fonctionne correctement. Comme vous utilisez le moteur d'exécution Python 3.11 pour les fonctions, les couches doivent également être compatibles avec Python 3.11.

L'`layer-python`exemple d'application contient deux exemples :

- Le premier exemple consiste à empaqueter la `requests`bibliothèque dans une couche Lambda. Le `layer/` répertoire contient les scripts permettant de générer la couche. Le `function/` répertoire contient un exemple de fonction permettant de vérifier le fonctionnement de la couche. La majeure partie de ce didacticiel explique comment créer et empaqueter cette couche.
- Le deuxième exemple consiste à empaqueter la numpybibliothèque dans une couche Lambda. Le `layer-numpy/` répertoire contient les scripts permettant de générer la couche. Le `function-numpy/` répertoire contient un exemple de fonction permettant de vérifier le fonctionnement de la couche. Pour un exemple de création et d'empaquetage de cette couche, consultezthe section called “Utilisation des distributions `manylinux` de roues”.

## Compatibilité de la couche Python avec Amazon Linux

La première étape de la création d'une couche consiste à regrouper l'ensemble du contenu de la couche dans une archive de fichiers .zip. Parce que les fonctions Lambda s'exécutent sur Amazon Linux, le contenu de votre couche doit pouvoir être compilé et construit dans un environnement Linux.

En Python, la plupart des paquets sont disponibles sous forme de roues (`.whl`fichiers) en plus de la distribution source. Chaque roue est un type de distribution intégrée qui prend en charge une combinaison spécifique de versions de Python, de systèmes d'exploitation et de jeux d'instructions machine.

Les roues sont utiles pour garantir la compatibilité de votre couche avec Amazon Linux. Lorsque vous téléchargez vos dépendances, téléchargez la roue universelle si possible. (Par défaut, `pip` installe la roue universelle si elle est disponible.) La roue universelle contient `any` une balise de plate-forme, indiquant qu'elle est compatible avec toutes les plateformes, y compris Amazon Linux.

Dans l'exemple suivant, vous allez empaqueter la `requests` bibliothèque dans une couche Lambda. La `requests` bibliothèque est un exemple de package disponible sous forme de roue universelle.

Tous les packages Python ne sont pas distribués sous forme de roues universelles. Par exemple, numpypossède plusieurs distributions de roues, chacune supportant un ensemble différent de

plateformes. Pour de tels packages, téléchargez la `manylinux` distribution afin de garantir la compatibilité avec Amazon Linux. Pour obtenir des instructions détaillées sur l'empaquetage de telles couches, reportez-vous à la sectionthe section called “Utilisation des distributions `manylinux` de roues”.

Dans de rares cas, un package Python peut ne pas être disponible sous forme de roue. Si seule la distribution source (sdist) existe, nous vous recommandons d'installer et d'empaqueter vos dépendances dans un environnement Docker basé sur l'image du conteneur de base Amazon Linux 2023. Nous recommandons également cette approche si vous souhaitez inclure vos propres bibliothèques personnalisées écrites dans d'autres langages tels que C/C++. Cette approche imite l'environnement d'exécution Lambda dans Docker et garantit que les dépendances de vos packages autres que Python sont compatibles avec Amazon Linux.

## Chemins de couche pour les environnements d'exécution Python

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d’exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

- `python`
- `python/lib/python3.x/site-packages`

Par exemple, le fichier de couche .zip obtenu que vous créez dans ce didacticiel possède la structure de répertoire suivante :

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
                # ...
```

La `requests`bibliothèque est correctement située dans le `python/lib/python3.11/site-packages` répertoire. Cela garantit que Lambda peut localiser la bibliothèque lors des appels de fonctions.

## Empaqueter le contenu de la couche

Dans cet exemple, vous allez empaqueter la `requests` bibliothèque Python dans un fichier .zip de couche. Procédez comme suit pour installer et empaqueter le contenu de la couche.

Pour installer et empaqueter le contenu de votre couche

1. Clonez le aws-lambda-developer-guide GitHub dépôt, qui contient l'exemple de code dont vous avez besoin dans le `sample-apps/layer-python` répertoire.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Accédez au `layer` répertoire de l'`layer-python`exemple d'application. Ce répertoire contient les scripts que vous utilisez pour créer et empaqueter correctement la couche.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-python/layer
   ```

3. Examinez le `requirements.txt`fichier. Ce fichier définit les dépendances que vous souhaitez inclure dans la couche, à savoir la `requests` bibliothèque. Vous pouvez mettre à jour ce fichier pour inclure les dépendances que vous souhaitez inclure dans votre propre couche.

   Example requirements.txt

   ```
   requests==2.31.0
   ```

4. Vérifiez que vous êtes autorisé à exécuter les deux scripts.

   ```
   chmod 744 1-install.sh && chmod 744 2-package.sh
   ```

5. Exécutez le `1-install.sh`script à l'aide de la commande suivante :

   ```
   ./1-install.sh
   ```

   Ce script permet `venv` de créer un environnement virtuel Python nommé`create_layer`. Il installe ensuite toutes les dépendances requises dans le `create_layer/lib/python3.11/site-packages` répertoire.

   Example 1-install.sh

   ```
   python3.11 -m venv create_layer
   ```

```
source create_layer/bin/activate
pip install -r requirements.txt
```

6. Exécutez le 2-package.shscript à l'aide de la commande suivante :

```
./2-package.sh
```

Ce script copie le contenu du `create_layer/lib` répertoire dans un nouveau répertoire nommé`python`. Il compresse ensuite le contenu du `python` répertoire dans un fichier nommé`layer_content.zip`. Il s'agit du fichier .zip de votre couche. Vous pouvez décompresser le fichier et vérifier qu'il contient la bonne structure de fichier, comme indiqué dans la the section called “Chemins de couche pour les environnements d'exécution Python” section.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

## Création de la couche

Dans cette section, vous allez prendre le `layer_content.zip` fichier que vous avez généré dans la section précédente et le télécharger en tant que couche Lambda. Vous pouvez télécharger une couche à l'aide du AWS Management Console ou du Lambda API via le AWS Command Line Interface ()AWS CLI. Lorsque vous téléchargez le fichier .zip de votre couche, dans la PublishLayerVersion AWS CLI commande suivante, spécifiez `python3.11` comme environnement d'exécution compatible et `arm64` comme architecture compatible.

```
aws lambda publish-layer-version --layer-name python-requests-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes python3.11 \
    --compatible-architectures "arm64"
```

Dans la réponse, notez le`LayerVersionArn`, qui ressemble à`arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1`. Vous aurez besoin de cet Amazon Resource Name (ARN) à l'étape suivante de ce didacticiel, lorsque vous ajouterez la couche à votre fonction.

## Ajouter la couche à votre fonction

Dans cette section, vous déployez un exemple de fonction Lambda qui utilise la `requests` bibliothèque dans son code de fonction, puis vous attachez la couche. Pour déployer la fonction, vous avez besoin d'unthe section called “Rôle d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources)”. Si vous n'avez pas de rôle d'exécution existant, suivez les étapes décrites dans la section pliable.

(Facultatif) Créez un rôle d'exécution

Pour créer un rôle d’exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :

   - Entité de confiance – Lambda.
   - Autorisations — AWSLambdaBasicExecutionRole.
   - Nom de rôle – **`lambda-role`**.

   La AWSLambdaBasicExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux dans Logs.

Le code de fonction Lambda importe la `requests` bibliothèque, fait une HTTP demande simple, puis renvoie le code d'état et le corps.

```
import requests

def lambda_handler(event, context):
    print(f"Version of requests library: {requests.__version__}")
    request = requests.get('https://api.github.com/')
    return {
        'statusCode': request.status_code,
        'body': request.text
    }
```

Pour déployer la fonction Lambda

1. Accédez au répertoire `function/`. Si vous êtes actuellement dans le `layer/` répertoire, exécutez la commande suivante :

```
cd ../function
```

2. Créez un package de déploiement de fichiers .zip à l'aide de la commande suivante :

```
zip my_deployment_package.zip lambda_function.py
```

3. Déployez la fonction. Dans la AWS CLI commande suivante, remplacez le `--role` paramètre par votre rôle d'exécution ARN :

```
aws lambda create-function --function-name python_function_with_layer \
    --runtime python3.11 \
    --architectures "arm64" \
    --handler lambda_function.lambda_handler \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://my_deployment_package.zip
```

4. Ensuite, attachez la couche à votre fonction. Dans la AWS CLI commande suivante, remplacez le `--layers` paramètre par la version de couche ARN que vous avez indiquée précédemment :

```
aws lambda update-function-configuration --function-name python_function_with_layer
 \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

5. Enfin, essayez d'appeler votre fonction à l'aide de la AWS CLI commande suivante :

```
aws lambda invoke --function-name python_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

Vous devriez obtenir un résultat du type suivant :

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Le `response.json` fichier de sortie contient des informations détaillées sur la réponse.

(Facultatif) Nettoyez vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la couche Lambda

1. Ouvrez la page Couches de la console Lambda.
2. Sélectionnez la couche que vous avez créée.
3. Choisissez Supprimer, puis sélectionnez à nouveau Supprimer.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

## Utilisation des distributions **`manylinux`** de roues

Parfois, un package que vous souhaitez inclure en tant que dépendance n'aura pas de roue universelle (en particulier, il n'en a pas `any` comme balise de plate-forme). Dans ce cas, téléchargez `manylinux` plutôt la roue qui le supporte. Cela garantit que vos bibliothèques de couches sont compatibles avec Amazon Linux.

`numpy`est un ensemble qui n'a pas de roue universelle. Si vous souhaitez inclure le `numpy` package dans votre couche, vous pouvez suivre les exemples d'étapes suivants pour installer et empaqueter correctement votre couche.

Pour installer et empaqueter le contenu de votre couche

1. Clonez le `aws-lambda-developer-guide` GitHub dépôt, qui contient l'exemple de code dont vous avez besoin dans le `sample-apps/layer-python` répertoire.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Accédez au `layer-numpy` répertoire de l'`layer-python`exemple d'application. Ce répertoire contient les scripts que vous utilisez pour créer et empaqueter correctement la couche.

```
cd aws-lambda-developer-guide/sample-apps/layer-python/layer-numpy
```

3. Examinez le `requirements.txt`fichier. Ce fichier définit les dépendances que vous souhaitez inclure dans votre couche, à savoir la `numpy` bibliothèque. Ici, vous spécifiez la distribution URL de la `manylinux` roue compatible avec Python 3.11, Amazon Linux et le jeu `x86_64` d'instructions :

   Example requirements.txt

```
https://files.pythonhosted.org/packages/3a/d0/
edc009c27b406c4f9cbc79274d6e46d634d139075492ad055e3d68445925/numpy-1.26.4-cp311-
cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

4. Vérifiez que vous êtes autorisé à exécuter les deux scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Exécutez le `1-install.sh`script à l'aide de la commande suivante :

```
./1-install.sh
```

   Ce script permet `venv` de créer un environnement virtuel Python nommé`create_layer`. Il installe ensuite toutes les dépendances requises dans le `create_layer/lib/python3.11/site-packages` répertoire. La `pip` commande est différente dans ce cas, car vous devez spécifier le `--platform` tag comme`manylinux2014_x86_64`. Cela indique `pip` d'installer la bonne `manylinux` roue, même si votre machine locale utilise macOS ou Windows.

   Example 1-install.sh

```
python3.11 -m venv create_layer
source create_layer/bin/activate
pip install -r requirements.txt --platform=manylinux2014_x86_64 --only-binary=:all:
 --target ./create_layer/lib/python3.11/site-packages
```

6. Exécutez le 2-package.shscript à l'aide de la commande suivante :

```
./2-package.sh
```

Ce script copie le contenu du `create_layer/lib` répertoire dans un nouveau répertoire nommé`python`. Il compresse ensuite le contenu du `python` répertoire dans un fichier nommé`layer_content.zip`. Il s'agit du fichier .zip de votre couche. Vous pouvez décompresser le fichier et vérifier qu'il contient la bonne structure de fichier, comme indiqué dans la the section called “Chemins de couche pour les environnements d'exécution Python” section.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Pour télécharger cette couche sur Lambda, utilisez la commande suivante : PublishLayerVersion AWS CLI

```
aws lambda publish-layer-version --layer-name python-numpy-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes python3.11 \
    --compatible-architectures "x86_64"
```

Dans la réponse, notez le`LayerVersionArn`, qui ressemble à`arn:aws:lambda:us-east-1:123456789012:layer:python-numpy-layer:1`. Pour vérifier que votre couche fonctionne comme prévu, déployez la fonction Lambda dans le `function-numpy` répertoire.

Pour déployer la fonction Lambda

1. Accédez au répertoire `function-numpy/`. Si vous êtes actuellement dans le `layer-numpy/` répertoire, exécutez la commande suivante :

```
cd ../function-numpy
```

2. Vérifiez le code de fonction. La fonction importe la `numpy` bibliothèque, crée un `numpy` tableau simple, puis renvoie un code d'état et un corps fictifs.

```
import json
```

```
import numpy as np

def lambda_handler(event, context):

    x = np.arange(15, dtype=np.int64).reshape(3, 5)
    print(x)

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

3. Créez un package de déploiement de fichiers .zip à l'aide de la commande suivante :

```
zip my_deployment_package.zip lambda_function.py
```

4. Déployez la fonction. Dans la AWS CLI commande suivante, remplacez le `--role` paramètre par votre rôle d'exécution ARN :

```
aws lambda create-function --function-name python_function_with_numpy \
    --runtime python3.11 \
    --handler lambda_function.lambda_handler \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://my_deployment_package.zip
```

5. Ensuite, attachez la couche à votre fonction. Dans la AWS CLI commande suivante, remplacez le `--layers` paramètre par la version de votre couche ARN :

```
aws lambda update-function-configuration --function-name python_function_with_numpy
 \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

6. Enfin, essayez d'appeler votre fonction à l'aide de la AWS CLI commande suivante :

```
aws lambda invoke --function-name python_function_with_numpy \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

Vous devriez obtenir un résultat du type suivant :

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Vous pouvez examiner les journaux des fonctions pour vérifier que le code imprime le `numpy` tableau conformément à la norme.

# Utilisation de l'objet de contexte Lambda pour récupérer les informations d'une fonction Python

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'appel, la fonction et l'environnement d'exécution. Pour en savoir plus sur la façon dont l'objet de contexte est transmis au gestionnaire de fonctions, consultez Définir le gestionnaire de fonctions Lambda en Python.

Méthodes de contexte

- `get_remaining_time_in_millis` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `function_name` – Nom de la fonction Lambda.
- `function_version` – Version de la fonction.
- `invoked_function_arn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `memory_limit_in_mb` – Quantité de mémoire allouée à la fonction.
- `aws_request_id` – Identifiant de la demande d'invocation.
- `log_group_name` – Groupe de journaux pour la fonction.
- `log_stream_name` – Flux de journal de l'instance de fonction.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
  - `cognito_identity_id` – Identité Amazon Cognito authentifiée.
  - `cognito_identity_pool_id` – Groupe d'identités Amazon Cognito ayant autorisé l'invocation.
- `client_context` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`

- `custom` – `dict` de valeurs personnalisées définies par l'application client mobile.
- `env`— A `dict` des informations environnementales fournies par le AWS SDK.

Powertools for Lambda (Python) fournit une définition d'interface pour l'objet de contexte Lambda. Vous pouvez utiliser la définition de l'interface pour les indications de type ou pour examiner plus en détail la structure de l'objet de contexte Lambda. Pour la définition de l'interface, consultez le fichier lambda_context.py dans le powertools-lambda-pythonréférentiel sur GitHub.

L'exemple suivant montre une fonction de gestionnaire qui consigne les informations de contexte.

Example handler.py

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining in
 get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", context.get_remaining_time_in_millis())
```

Outre les options répertoriées ci-dessus, vous pouvez également utiliser le AWS X-Ray SDK Instrumentation du code Python dans AWS Lambda pour identifier les chemins de code critiques, suivre leurs performances et capturer les données à des fins d'analyse.

# Enregistrez et surveillez les fonctions Lambda en Python

AWS Lambda surveille automatiquement les fonctions Lambda et envoie des entrées de journal à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des détails sur chaque invocation et d'autres sorties provenant du code de votre fonction au flux de journaux. Pour plus d'informations sur CloudWatch les journaux, consultezUtilisation des CloudWatch journaux et journaux avec Lambda.

Pour produire des journaux à partir de votre code de fonction, vous pouvez utiliser le module intégré `logging`. Pour des entrées plus détaillées, vous pouvez utiliser n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`.

## Impression dans le journal

Pour envoyer une sortie de base aux journaux, vous pouvez utiliser une méthode `print` dans votre fonction. L'exemple suivant enregistre les valeurs du groupe et du flux de CloudWatch journaux Logs, ainsi que l'objet de l'événement.

Notez que si votre fonction génère des journaux à l'aide d'`print`instructions Python, Lambda ne peut envoyer des résultats de journal à CloudWatch Logs qu'au format texte brut. Pour capturer des journaux de manière structuréeJSON, vous devez utiliser une bibliothèque de journalisation prise en charge. Pour plus d'informations, consultez the section called "Utilisation des contrôles de journalisation avancés de Lambda avec Python".

Example lambda_function.py

```
import os
def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    print(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    print('## EVENT')
    print(event)
```

Example sortie de journal

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
/aws/lambda/my-function
```

```
2023/08/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
 Duration: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1   SegmentId: 07f5xmpl2d1f6f85
 Sampled: true
```

L'environnement d'exécution Python enregistre les lignes `START`, `END` et `REPORT` pour chaque invocation. La ligne `REPORT` comprend les données suivantes :

REPORTchamps de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled : pour les demandes suivies, résultat de l'échantillonnage.

## Utilisation d'une bibliothèque de journalisation

Pour des journaux plus détaillés, utilisez le module de journalisation de la bibliothèque standard, ou de toute bibliothèque de journalisation tierce qui écrit à `stdout` ou `stderr`.

Pour les environnements d'exécution Python pris en charge, vous pouvez choisir si les journaux créés à l'aide du `logging` module standard sont capturés en texte brut ouJSON. Pour en savoir plus, consultez the section called "Utilisation des contrôles de journalisation avancés de Lambda avec Python".

Actuellement, le format de journal par défaut pour tous les environnements d'exécution Python est le texte brut. L'exemple suivant montre comment les sorties de journal créées à l'aide du `logging` module standard sont capturées en texte brut dans CloudWatch Logs.

```
import os
import logging
logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    logger.info(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    logger.info('## EVENT')
    logger.info(event)
```

La sortie de `logger` inclut le niveau de journal, l'horodatage et l'ID de demande.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
 ENVIRONMENT VARIABLES
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    /aws/
lambda/my-function
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    2023/01/31/
[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d
[INFO]  2023-08-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT
[INFO]  2023-08-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    {'key':
 'value'}
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125  Duration: 2.75 ms   Billed
 Duration: 3 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370   SegmentId: 073cxmpl3e442861
 Sampled: true
```

> **Note**
>
> Lorsque le format de journal de votre fonction est défini sur du texte brut, le paramètre de niveau de journal par défaut pour les environnements d'exécution Python est. WARN Cela signifie que Lambda envoie uniquement des sorties de journal de niveau ou inférieur à WARN CloudWatch Logs. Pour modifier le niveau de journalisation par défaut, utilisez la méthode

Python `logging` `setLevel()` comme indiqué dans cet exemple de code. Si vous définissez le format de journal de votre fonction surJSON, nous vous recommandons de configurer le niveau de journalisation de votre fonction à l'aide de Lambda Advanced Logging Controls et non en définissant le niveau de journalisation dans le code. Pour en savoir plus, consultez the section called “Utilisation du filtrage au niveau du journal avec Python”

## Utilisation des contrôles de journalisation avancés de Lambda avec Python

Pour mieux contrôler la manière dont les journaux de vos fonctions sont capturés, traités et consommés, vous pouvez configurer les options de journalisation suivantes pour les environnements d’exécution Lambda Python pris en charge :

- Format de journal : choisissez entre le format texte brut et le JSON format structuré pour les journaux de votre fonction
- Niveau de journalisation : pour les journaux au JSON format, choisissez le niveau de détail des journaux que Lambda envoie à Amazon CloudWatch, par exempleERROR, DEBUG ou INFO
- Groupe de journaux : choisissez le groupe de CloudWatch journaux auquel votre fonction envoie les journaux

Pour plus d’informations sur ces options de journalisation et pour savoir comment configurer votre fonction pour les utiliser, consultez the section called “Configuration des journaux de fonctions”.

Pour en savoir plus sur l’utilisation du format de journal et les options de niveau de journal avec vos fonctions Python Lambda, consultez les instructions des sections suivantes.

### Utilisation de JSON journaux structurés avec Python

Si vous sélectionnez le JSON format de journal de votre fonction, Lambda enverra les journaux produits par la bibliothèque de journalisation standard Python CloudWatch sous forme structurée. JSON Chaque objet JSON journal contient au moins quatre paires clé-valeur avec les clés suivantes :

- `"timestamp"` - heure à laquelle le message de journal a été généré
- `"level"` - niveau de journalisation attribué au message
- `"message"` - contenu du message de journal
- `"requestId"` - identifiant unique de la demande pour l’invocation de la fonction

La `logging` bibliothèque Python peut également ajouter des paires clé-valeur supplémentaires, comme dans `"logger"` cet JSON objet.

Les exemples présentés dans les sections suivantes montrent comment les sorties de journal générées à l'aide de la `logging` bibliothèque Python sont capturées dans CloudWatch Logs lorsque vous configurez le format de journal de votre fonction en tant queJSON.

Notez que si vous utilisez la `print` méthode pour produire des sorties de journal de base comme décrit dansthe section called “Impression dans le journal”, Lambda capturera ces sorties sous forme de texte brut, même si vous configurez le format de journalisation de votre fonction en tant que. JSON

Sorties de JSON journal standard à l'aide de la bibliothèque de journalisation Python

Les exemples d'extrait de code et de sortie de journal suivants montrent comment les sorties de journal standard générées à l'aide de la `logging` bibliothèque Python sont capturées dans CloudWatch Logs lorsque le format de journal de votre fonction est défini sur. JSON

Example Code de journalisation Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example JSONenregistrement du journal

```
{
    "timestamp":"2023-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"Inside the handler function",
    "logger": "root",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Enregistrement de paramètres supplémentaires dans JSON

Lorsque le format de journal de votre fonction est défini surJSON, vous pouvez également enregistrer des paramètres supplémentaires avec la `logging` bibliothèque Python standard en utilisant le `extra` mot-clé pour transmettre un dictionnaire Python à la sortie du journal.

Example Code de journalisation Python

```
import logging

def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
        extra={"a":"b", "b": [3]},
    )
```

Example JSONenregistrement du journal

```
{
  "timestamp": "2023-11-02T15:26:28Z",
  "level": "INFO",
  "message": "extra parameters example",
  "logger": "root",
  "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",
  "a": "b",
  "b": [
    3
  ]
}
```

Enregistrement des exceptions dans JSON

L'extrait de code suivant montre comment les exceptions Python sont capturées dans la sortie du journal de votre fonction lorsque vous configurez le format du journal en tant que. JSON Notez que le niveau de journalisation est attribué aux sorties `logging.exception` générées à l'aide de ce paramètreERROR.

Example Code de journalisation Python

```
import logging

def lambda_handler(event, context):
    try:
        raise Exception("exception")
    except:
        logging.exception("msg")
```

Example JSONenregistrement du journal

```
{
  "timestamp": "2023-11-02T16:18:57Z",
  "level": "ERROR",
  "message": "msg",
  "logger": "root",
  "stackTrace": [
    "  File \"/var/task/lambda_function.py\", line 15, in lambda_handler\n    raise
 Exception(\"exception\")\n"
  ],
  "errorType": "Exception",
  "errorMessage": "exception",
  "requestId": "3f9d155c-0f09-46b7-bdf1-e91dab220855",
  "location": "/var/task/lambda_function.py:lambda_handler:17"
}
```

JSONjournaux structurés avec d'autres outils de journalisation

Si votre code utilise déjà une autre bibliothèque de journalisation, telle que Powertools for AWS Lambda, pour produire des journaux JSON structurés, vous n'avez pas besoin d'apporter de modifications. AWS Lambda ne double code aucun journal déjà JSON codé. Même si vous configurez votre fonction pour utiliser le format JSON journal, vos sorties de journalisation apparaissent CloudWatch dans la JSON structure que vous définissez.

L'exemple suivant montre comment les sorties de journal générées à l'aide du AWS Lambda package Powertools for sont capturées dans CloudWatch Logs. Le format de cette sortie de journal est le même, que la configuration de journalisation de votre fonction soit définie sur JSON ouTEXT. Pour plus d'informations sur l'utilisation de Powertools pour AWS Lambda, consultez the section called “Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour la journalisation structurée” et the section called “Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour la journalisation structurée”

Example Extrait de code de journalisation en Python (à l'aide de Powertools pour) AWS Lambda

```
from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example JSONenregistrement du journal (à l'aide de Powertools pour AWS Lambda)

```
{
    "level": "INFO",
    "location": "lambda_handler:7",
    "message": "Inside the handler function",
    "timestamp": "2023-10-31 22:38:21,010+0000",
    "service": "service_undefined",
    "xray_trace_id": "1-654181dc-65c15d6b0fecbdd1531ecb30"
}
```

## Utilisation du filtrage au niveau du journal avec Python

En configurant le filtrage au niveau des journaux, vous pouvez choisir de n'envoyer que les journaux ayant un certain niveau de journalisation ou un niveau inférieur à CloudWatch Logs. Pour savoir comment configurer le filtrage de niveau journal de votre fonction, consultez the section called “Filtrage au niveau du journal”.

AWS Lambda Pour filtrer les journaux de vos applications en fonction de leur niveau de journalisation, votre fonction doit utiliser des journaux JSON formatés. Vous pouvez effectuer cette opération de deux façons :

- Créez des sorties de journal à l'aide de la `logging` bibliothèque Python standard et configurez votre fonction pour utiliser le formatage des JSON journaux. AWS Lambda filtre ensuite les sorties de votre journal à l'aide de la paire clé-valeur « niveau » de l'JSONobjet décrit dansthe section called “Utilisation de JSON journaux structurés avec Python”. Pour savoir comment configurer le format de journal de votre fonction, consultez the section called “Configuration des journaux de fonctions”.
- Utilisez une autre bibliothèque ou méthode de journalisation pour créer des journaux JSON structurés dans votre code qui incluent une paire clé-valeur « niveau » définissant le niveau de sortie du journal. Par exemple, vous pouvez utiliser Powertools pour générer des sorties AWS Lambda de journal JSON structurées à partir de votre code.

  Vous pouvez également utiliser une instruction print pour générer un JSON objet contenant un identifiant de niveau de journalisation. L'instruction d'impression suivante produit une sortie JSON formatée dans laquelle le niveau de journalisation est défini sur. INFO AWS Lambda enverra l'JSONobjet à CloudWatch Logs si le niveau de journalisation de votre fonction est défini sur INFODEBUG, ouTRACE.

```
print('{"msg":"My log message", "level":"info"}')
```

Pour que Lambda puisse filtrer les journaux de votre fonction, vous devez également inclure une paire `"timestamp"` clé-valeur dans la sortie de votre JSON journal. L'heure doit être spécifiée dans un format d'horodatage RFC3339 valide. Si vous ne fournissez pas d'horodatage valide, Lambda attribuera le niveau au journal INFO et ajoutera un horodatage pour vous.

## Affichage des journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans CloudWatch la console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

# Afficher les journaux avec AWS CLI

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-`

`in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
```

```
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

## Utilisation d'autres outils et bibliothèques de journalisation

Powertools for AWS Lambda (Python) est une boîte à outils destinée aux développeurs qui permet de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs. L'utilitaire Logger fournit un enregistreur optimisé pour Lambda qui inclut des informations supplémentaires sur le contexte des fonctions dans toutes vos fonctions avec une sortie structurée sous la forme. JSON Utilisez cet utilitaire pour effectuer les opérations suivantes :

- Capturez les champs clés du contexte Lambda, du démarrage à froid et des structures enregistrant la sortie sous JSON
- Journaliser les événements d'invocation Lambda lorsque cela est demandé (désactivé par défaut)
- Imprimer tous les journaux uniquement pour un pourcentage d'invocations via l'échantillonnage des journaux (désactivé par défaut)
- Ajouter des clés supplémentaires au journal structuré à tout moment
- Utilisez un formateur de journal personnalisé (Bring Your Own Formatter) pour générer les journaux dans une structure compatible avec la journalisation de votre organisation RFC

# Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World Python avec des modules Powertools for Python (français non garanti) intégrés à l'aide d' AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison de la API passerelle, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Python 3.9
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du modèle Hello World Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Créez l'application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

La sortie du journal se présente comme suit :

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2023-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
```

```
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
```

```
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 17 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
 Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
 Sampled: true
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

## Gestion de la conservation des journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation après laquelle les journaux CloudWatch sont automatiquement supprimés. Pour configurer la conservation des journaux, ajoutez ce qui suit à votre AWS SAM modèle :

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties
```

```
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World Python avec des modules Powertools pour AWS Lambda (Python) intégrés à l'aide du AWS CDK. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison de la API passerelle, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message hello world.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Python 3.9
- AWS CLI version 2
- AWS CDK version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS CDK application

1. Créez un répertoire de projets pour votre nouvelle application.

   ```
   mkdir hello-world
   cd hello-world
   ```

2. Initialisez l’application.

```
cdk init app --language python
```

3. Installez les dépendances de Python.

```
pip install -r requirements.txt
```

4. Créez un répertoire lambda_function dans le dossier racine.

```
mkdir lambda_function
cd lambda_function
```

5. Créez un fichier app.py et ajoutez-y le code suivant. Il s'agit du code de la fonction Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
 value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
```

```
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
 ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

6. Ouvrez le répertoire hello_world. Vous devriez voir un fichier nommé hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Ouvrez hello_world_stack.py et ajoutez le code suivant au fichier. Il contient le constructeur Lambda, qui crée la fonction Lambda, configure les variables d'environnement pour Powertools et définit la durée de conservation des journaux à une semaine, et le constructeur 1, qui crée le ApiGatewayv. REST API

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
 constructor is in Alpha, o we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
```

```
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_9,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
                "LOG_LEVEL": "INFO"
            }
        )

        apigw = apigwv1.RestApi(self, "PowertoolsAPI",
 deploy_options=apigwv1.StageOptions(stage_name="dev"))

        hello_api = apigw.root.add_resource("hello")
        hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
 proxy=True))

        CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")
```

8. Déployez votre application.

```
cd ..
cdk deploy
```

9. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Appelez le API point de terminaison :

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

11. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

La sortie du journal se présente comme suit :

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2023-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
    "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
```

```
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
```

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 17 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
 Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
 Sampled: true
```

12. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
cdk destroy
```

# Test de la fonction AWS Lambda dans Python

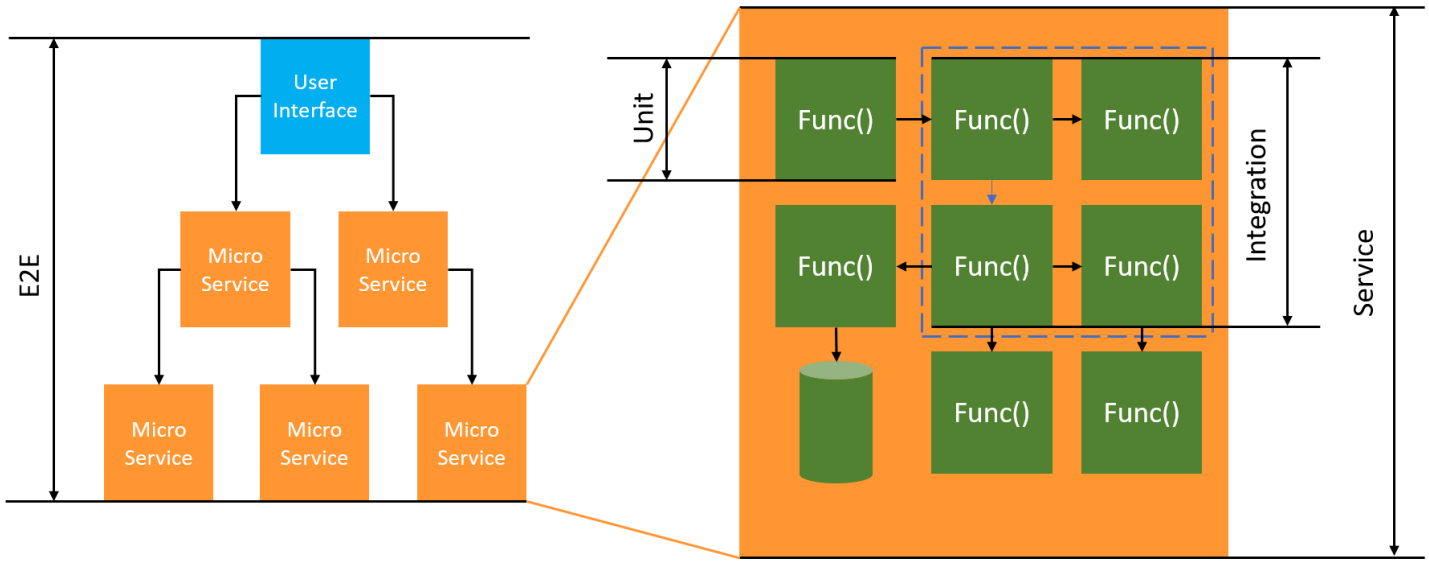
> **Note**
>
> Consultez le chapitre Test des fonctions pour une présentation complète des techniques et des bonnes pratiques pour tester les solutions sans serveur.

Le test des fonctions sans serveur utilise les types et techniques de tests traditionnels, mais vous devez également envisager de tester les applications sans serveur dans leur ensemble. Les tests basés sur le cloud fourniront la mesure la plus précise de la qualité de vos fonctions et de vos applications sans serveur.

Une architecture d'application sans serveur comprend des services gérés qui fournissent des fonctionnalités d'applications critiques par le biais d'appels d'API. C'est pourquoi votre cycle de développement doit inclure des tests automatisés qui vérifient la fonctionnalité lorsque votre fonction et vos services interagissent.

Si vous ne créez pas de tests basés sur le cloud, vous pouvez rencontrer des problèmes en raison des différences entre votre environnement local et l'environnement déployé. Votre processus d'intégration continue doit exécuter des tests sur une série de ressources allouées dans le cloud avant de promouvoir votre code vers l'environnement de déploiement suivant, comme l'assurance qualité, la mise en place ou la production.

Poursuivez la lecture de ce petit guide pour en savoir plus sur les stratégies de test pour les applications sans serveur, ou rendez-vous sur le référentiel Serverless Test Samples pour vous plonger dans des exemples pratiques, spécifiques au langage et à l'exécution que vous avez choisis.




Pour les tests sans serveur, vous continuerez à écrire des unités, des tests d'intégration et end-to-enddes tests.

- Tests unitaires : tests exécutés sur un bloc de code isolé. Par exemple, la vérification de la logique métier permettant de calculer les frais de livraison en fonction d'un article et d'une destination donnés.
- Tests d'intégration : tests impliquant au moins deux composants ou services qui interagissent, généralement dans un environnement cloud. Par exemple, la vérification d'une fonction traite les événements d'une file d'attente.
- nd-to-end Tests E : tests qui vérifient le comportement de l'ensemble d'une application. Il s'agit par exemple de s'assurer que l'infrastructure est correctement mise en place et que les événements circulent entre les services comme prévu pour enregistrer la commande d'un client.

## Test de vos applications sans serveur

Vous utiliserez généralement une combinaison d'approches pour tester le code de votre application sans serveur, notamment des tests dans le cloud, des tests avec des simulations et, occasionnellement, des tests avec des émulateurs.

### Tests dans le cloud

Les tests dans le cloud sont utiles pour toutes les phases de test, y compris les tests unitaires, les tests d'intégration et les end-to-end tests. Vous exécutez des tests sur du code déployé dans le cloud

et interagissez avec des services basés sur le cloud. Cette approche fournit la mesure la plus précise de la qualité de votre code.

Un moyen pratique de déboguer votre fonction Lambda dans le cloud est de passer par la console avec un événement de test. Un événement de test est une entrée JSON pour votre fonction. Si votre fonction ne nécessite pas d'entrée, l'événement peut être un document JSON vide (`{}`). La console fournit des exemples d'événements pour diverses intégrations de services. Après avoir créé un événement dans la console, vous pouvez le partager avec votre équipe pour faciliter les tests et les rendre plus cohérents.

> **Note**
>
> Tester une fonction dans la console est un moyen rapide de démarrer, mais l'automatisation de vos cycles de test garantit la qualité des applications et la rapidité du développement.

## Outils de test

Il existe des outils et des techniques permettant d'accélérer les boucles de rétroaction du développement. Par exemple, AWS SAM Accelerate et le mode de surveillance AWS CDK réduisent tous deux le temps nécessaire à la mise à jour des environnements cloud.

Moto est une bibliothèque Python qui permet de simuler des services et des ressources AWS. Vous pouvez ainsi tester vos fonctions avec peu ou pas de modifications à l'aide de décorateurs pour intercepter et simuler des réponses.

La fonctionnalité de validation de Powertools pour AWS Lambda (Python) fournit des décorateurs qui vous permettent de valider les événements d'entrée et les réponses de sortie de vos fonctions Python.

Pour plus d'informations, consultez l'article de blog Unit Testing Lambda with Python and Mock AWS Services.

Pour réduire le temps de latence liés aux itérations de déploiement dans le cloud, consultez AWS Serverless Application Model (AWS SAM) et le mode de surveillance AWS Cloud Development Kit (AWS CDK). Ces outils surveillent les modifications apportées à votre infrastructure et à votre code. Ils réagissent à ces modifications en créant et en déployant automatiquement des mises à jour incrémentielles dans votre environnement cloud.

Des exemples utilisant ces outils sont disponibles dans le référentiel de code Python Test Samples.

# Instrumentation du code Python dans AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des trois SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- Kit SDK AWS X-Ray pour Python— Et SDK pour générer et envoyer des données de trace à X-Ray.
- Powertools for AWS Lambda (Python) — Une boîte à outils pour les développeurs permettant de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections

- Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour le traçage
- Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour le traçage
- Utilisation ADOT pour instrumenter vos fonctions Python

- Utiliser le X-Ray SDK pour instrumenter vos fonctions Python
- Activation du suivi avec la console Lambda
- Activation du traçage avec le Lambda API
- Activation du traçage avec AWS CloudFormation
- Interprétation d'un suivi X-Ray
- Stockage des dépendances d'exécution dans une couche (X-RaySDK)

## Utilisation de Powertools pour AWS Lambda (Python) et AWS SAM pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World Python avec des modules Powertools pour AWS Lambda (Python) intégrés à l'aide du AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message hello world.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Python 3.9
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l’application à l’aide du modèle Hello World Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Créez l'application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
```

```
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 %

de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

## Utilisation de Powertools pour AWS Lambda (Python) et AWS CDK pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World Python avec des modules Powertools pour AWS Lambda (Python) intégrés à l'aide du AWS CDK. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message hello world.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Python 3.9
- AWS CLI version 2
- AWS CDK version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS CDK application

1. Créez un répertoire de projets pour votre nouvelle application.

```
mkdir hello-world
cd hello-world
```

2. Initialisez l’application.

```
cdk init app --language python
```

3. Installez les dépendances de Python.

```
pip install -r requirements.txt
```

4. Créez un répertoire lambda_function dans le dossier racine.

```
mkdir lambda_function
cd lambda_function
```

5. Créez un fichier app.py et ajoutez-y le code suivant. Il s'agit du code de la fonction Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
 value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
 ColdStart metric
```

```
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

6. Ouvrez le répertoire hello_world. Vous devriez voir un fichier nommé hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Ouvrez hello_world_stack.py et ajoutez le code suivant au fichier. Il contient le constructeur Lambda, qui crée la fonction Lambda, configure les variables d'environnement pour Powertools et définit la durée de conservation des journaux à une semaine, et le constructeur 1, qui crée le ApiGatewayv. REST API

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
 constructor is in Alpha, o we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
```

```
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_9,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
                "LOG_LEVEL": "INFO"
            }
        )

        apigw = apigwv1.RestApi(self, "PowertoolsAPI",
 deploy_options=apigwv1.StageOptions(stage_name="dev"))

        hello_api = apigw.root.add_resource("hello")
        hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
 proxy=True))

        CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")
```

8. Déployez votre application.

```
cd ..
cdk deploy
```

9. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

11. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie des traces ressemble à ceci :

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

12. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
cdk destroy
```

## Utilisation ADOT pour instrumenter vos fonctions Python

ADOTfournit des couches Lambda entièrement gérées qui regroupent tout ce dont vous avez besoin pour collecter des données de télémétrie à l'aide du. OTel SDK En consommant cette couche, vous pouvez instrumenter vos fonctions Lambda sans avoir à modifier le code de fonction. Vous pouvez également configurer votre couche pour effectuer une initialisation personnalisée deOTel. Pour plus d'informations, consultez la section Configuration personnalisée du ADOT collecteur sur Lambda dans la ADOT documentation.

Pour les environnements d'exécution Python, vous pouvez ajouter la couche Lambda AWS gérée pour ADOT Python afin d'instrumenter automatiquement vos fonctions. Cette couche fonctionne à la fois pour les architectures arm64 et x86_64. Pour obtenir des instructions détaillées sur la façon d'ajouter cette couche, consultez AWS Distro for OpenTelemetry Lambda Support for Python ADOT dans la documentation.

## Utiliser le X-Ray SDK pour instrumenter vos fonctions Python

Pour enregistrer des détails sur les appels effectués par votre fonction Lambda à d'autres ressources de votre application, vous pouvez également utiliser le Kit SDK AWS X-Ray pour Python. Pour obtenir leSDK, ajoutez le `aws-xray-sdk` package aux dépendances de votre application.

Example requirements.txt

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Dans votre code de fonction, vous pouvez instrumenter AWS SDK les clients en appliquant le `aws_xray_sdk.core` module à la `boto3` bibliothèque.

Example function — Suivi d'un AWS SDK client

```
import boto3
from aws_xray_sdk.core import xray_recorder
```

```
from aws_xray_sdk.core import patch_all

logger = logging.getLogger()
logger.setLevel(logging.INFO)
patch_all()

client = boto3.client('lambda')
client.get_account_settings()

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
  ...
```

Après avoir ajouté les dépendances correctes et apporté les modifications de code nécessaires, activez le suivi dans la configuration de votre fonction via la console Lambda ou le. API

## Activation du suivi avec la console Lambda

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

## Activation du traçage avec le Lambda API

Configurez le suivi sur votre fonction Lambda avec AWS CLI ou utilisez AWS SDK les opérations suivantes : API

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant permet d'activer le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du traçage avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

## Interprétation d'un suivi X-Ray

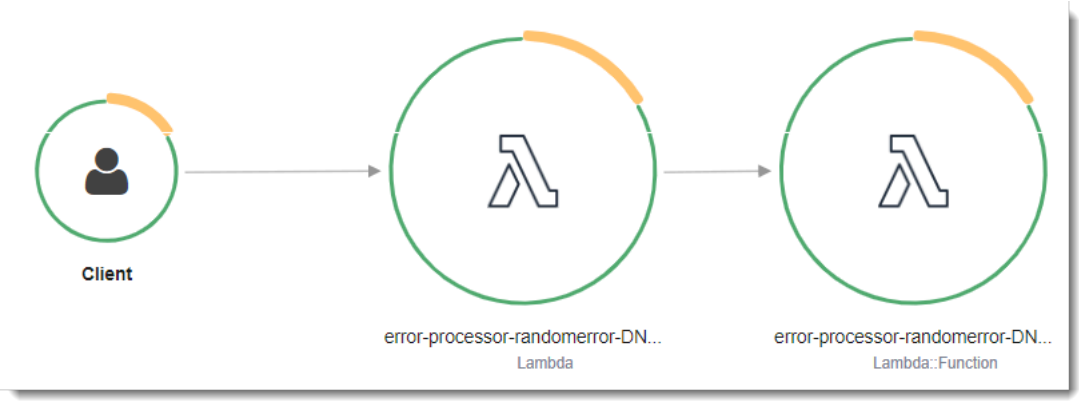
Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les

autorisations requises au rôle d'exécution de votre fonction. Dans le cas contraire, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

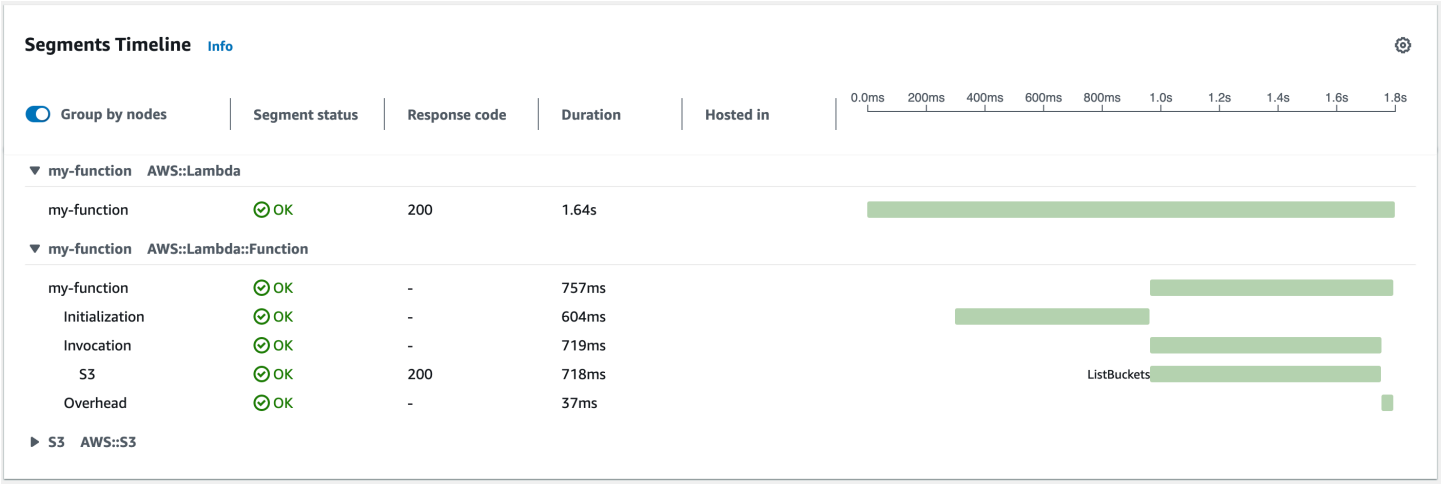
Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS

L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez Kit SDK AWS X-Ray pour Python dans le AWS X-Ray Guide du développeur.

Tarification

Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

## Stockage des dépendances d'exécution dans une couche (X-RaySDK)

Si vous utilisez le X-Ray SDK pour instrumenter votre code de fonction chez les AWS SDK clients, votre package de déploiement peut devenir assez volumineux. Pour éviter de télécharger les

dépendances d'exécution chaque fois que vous mettez à jour le code de votre fonction, empaquetez le X-Ray SDK dans une couche Lambda.

L'exemple suivant montre une ressource `AWS::Serverless::LayerVersion` qui stocke le Kit SDK AWS X-Ray pour Python.

Example template.yml : couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-python-lib
      Description: Dependencies for the blank-python sample app.
      ContentUri: package/.
      CompatibleRuntimes:
        - python3.8
```

Avec cette configuration, vous ne mettez à jour les fichiers de couche de bibliothèque que si vous modifiez vos dépendances d'exécution. Étant donné que le package de déploiement de la fonction contient uniquement votre code, cela peut contribuer à réduire les temps de chargement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application blank-python.

# Création de fonctions Lambda avec Ruby

Vous pouvez exécuter du code Ruby dans AWS Lambda. Lambda fournit des runtimes pour Ruby qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement qui inclut le AWS SDK for Ruby, avec les informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez. Pour en savoir plus sur les SDK versions incluses dans les environnements d'exécution Ruby, consultezthe section called “Versions incluses dans Runtime SDK”.

Lambda prend en charge les runtimes Ruby suivants.

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Ruby 3.3 | `ruby3.3` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Ruby 3.2 | `ruby3.2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

Pour créer une fonction Ruby

1. Ouvrez la console Lambda.
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :
   - Nom de la fonction : saisissez le nom de la fonction.
   - Exécution : choisissez Ruby 3.2.
4. Sélectionnez Créer une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Event name (Nom de l’événement), saisissez **`test`**.
7. Sélectionnez Save Changes.
8. Pour invoquer la fonction, choisissez Test.

La console crée une fonction Lambda avec un seul fichier source nommé `lambda_function.rb`. Vous pouvez modifier ce fichier et ajouter d'autres fichiers dans l'éditeur de code intégré. Choisissez Save pour enregistrer les changements. Ensuite, pour exécuter votre code, choisissez Test.

Le fichier `lambda_function.rb` exporte une fonction nommée `lambda_handler` qui accepte un objet événement et un objet contexte. Il s'agit de la fonction de gestionnaire que Lambda appelle lors de l'invocation de la fonction. L'environnement d'exécution (runtime) de la fonction Ruby obtient les événements d'invocations de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `lambda_function.lambda_handler`.

Lorsque vous enregistrez votre code de fonction, la console Lambda crée un package de déploiement d'archive de fichiers .zip. Lorsque vous développez votre code de fonction en dehors de la console (à l'aide d'unIDE), vous devez créer un package de déploiement pour télécharger votre code dans la fonction Lambda.

Le runtime de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'invocation. L'objet de contexte contient des informations supplémentaires sur l'invocation, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs. La fonction runtime envoie les détails de chaque appel à CloudWatch Logs. Il relaie tous les journaux que votre fonction génère pendant l'invocation. Si votre fonction renvoie une erreur, Lambda met en forme l'erreur et la renvoie à l'appelant.

Rubriques


- Versions incluses dans Runtime SDK
- Activer encore un autre Ruby JIT (YJIT)
- Définir le gestionnaire de fonctions Lambda dans Ruby
- Déployer des fonctions Lambda en Ruby avec des archives de fichiers .zip
- Déploiement de fonctions Lambda en Ruby avec des images conteneurs
- Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Ruby
- Enregistrez et surveillez les fonctions Ruby Lambda
- Instrumentation du code Ruby dans AWS Lambda

# Versions incluses dans Runtime SDK

La version AWS SDK incluse dans le runtime Ruby dépend de la version d'exécution et de votre Région AWS. Le AWS SDK for Ruby est conçu pour être modulaire et est séparé par Service AWS. Pour trouver le numéro de version d'une gemme de service particulière incluse dans le moteur d'exécution que vous utilisez, créez une fonction Lambda avec un code au format suivant. Remplacez `aws-sdk-s3` et `Aws::S3` par le nom des gemmes de service utilisées par votre code.

```
require 'aws-sdk-s3'

def lambda_handler(event:, context:)
  puts "Service gem version: #{Aws::S3::GEM_VERSION}"
  puts "Core version: #{Aws::CORE_GEM_VERSION}"
end
```

# Activer encore un autre Ruby JIT (YJIT)

Le runtime Ruby 3.2 supporte YJITun JIT compilateur Ruby léger et minimaliste. YJITfournit des performances nettement supérieures, mais utilise également plus de mémoire que l'interpréteur Ruby. YJITest recommandé pour les charges de travail Ruby on Rails.

YJITn'est pas activé par défaut. YJITPour activer une fonction Ruby 3.2, définissez la variable d'`RUBY_YJIT_ENABLE`environnement sur`1`. Pour confirmer que cette option YJIT est activée, imprimez le résultat de la `RubyVM::YJIT.enabled?` méthode.

Example — Confirmez que YJIT c'est activé

```
puts(RubyVM::YJIT.enabled?())
# => true
```

# Définir le gestionnaire de fonctions Lambda dans Ruby

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Rubriques


- Notions de base sur Ruby Handler
- Bonnes pratiques en matière de code pour les fonctions Ruby Lambda


## Notions de base sur Ruby Handler

Dans l'exemple suivant, le fichier `function.rb` définit une méthode de gestionnaire nommée `handler`. La fonction de gestion prend deux objets en entrée et renvoie un JSON document.

Example function.rb

```
require 'json'

def handler(event:, context:)
    { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Dans la configuration de votre fonction, le paramètre `handler` indique à Lambda où trouver le gestionnaire. Pour l'exemple précédent, la valeur correcte pour ce paramètre est **`function.handler`**. Elle inclut deux noms séparées par un point : le nom du fichier et le nom de la méthode de gestionnaire.

Vous pouvez également définir votre méthode de gestionnaire dans une classe. L'exemple suivant définit une méthode de gestionnaire nommée `process` sur une classe nommée `Handler` dans un module appelé `LambdaFunctions`.

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
```

```
    end
  end
end
```

Dans ce cas, le paramètre de gestionnaire est **`source.LambdaFunctions::Handler.process`**.

Les deux objets qui sont acceptés par le gestionnaire sont l'événement d'invocation et le contexte. L'événement est un objet Ruby qui contient la charge utile qui est fournie par le mécanisme d'invocation. Si la charge utile est un JSON document, l'objet de l'événement est un hachage Ruby. Sinon, il s'agit d'une chaîne. L'objet de contexte contient des méthodes et des propriétés qui fournissent des informations sur l'invocation, la fonction et l'environnement d'exécution.

Le gestionnaire de fonction est exécuté chaque fois que votre fonction Lambda est invoquée. Le code statique à l'extérieur du gestionnaire est exécuté une fois par instance de la fonction. Si votre gestionnaire utilise des ressources telles que des SDK clients et des connexions à des bases de données, vous pouvez les créer en dehors de la méthode du gestionnaire afin de les réutiliser pour plusieurs appels.

Chaque instance de votre fonction peut traiter plusieurs événements d'invocations, mais elle ne traite qu'un événement à la fois. Le nombre d'instances traitant un événement à un moment donné est la simultanéité de votre fonction. Pour plus d'informations sur l'environnement d'exécution Lambda, consultez Comprendre le cycle de vie de l'environnement d'exécution Lambda.

## Bonnes pratiques en matière de code pour les fonctions Ruby Lambda

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire. Par exemple, dans Ruby, cela peut ressembler à ceci :

```
def lambda_handler(event:, context:)
    foo = event['foo']
    bar = event['bar']

    result = my_lambda_function(foo:, bar:)

end

def my_lambda_function(foo:, bar:)
```

```
    // MyLambdaFunction logic here

end
```

- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour le runtime Ruby, il s'agit notamment du AWS SDK. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation. Pour les fonctions créées dans Ruby, évitez de télécharger l'intégralité de la AWS SDK bibliothèque dans le cadre de votre package de déploiement. Dépendez plutôt de manière sélective des gemmes qui récupèrent les composants dont SDK vous avez besoin (par exemple, les gemmes DynamoDB ou Amazon SDK S3).

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.

- Utilisez des variables d’environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d’environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d’invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.
- Écriture du code idempotent. L’écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Déployer des fonctions Lambda en Ruby avec des archives de fichiers .zip

Le code de votre AWS Lambda fonction comprend un fichier .rb contenant le code du gestionnaire de votre fonction, ainsi que toutes les dépendances supplémentaires (gemmes) dont dépend votre code. Pour déployer ce code de fonction vers Lambda, vous utilisez un package de déploiement. Ce package peut être une archive de fichier .zip ou une image de conteneur. Pour plus d'informations sur l'utilisation d'images de conteneur avec Ruby, consultez la page Déployer des fonctions Lambda Ruby avec des images de conteneur.

Pour créer votre package de déploiement sous forme d'archive de fichier .zip, vous pouvez utiliser l'utilitaire d'archivage .zip intégré à votre outil de ligne de commande, ou tout autre utilitaire .zip tel que 7zip. Les exemples présentés dans les sections suivantes supposent que vous utilisez un outil `zip` de ligne de commande dans un environnement Linux ou macOS. Pour utiliser les mêmes commandes sous Windows, vous pouvez installer le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et de Bash.

Notez que Lambda utilise des autorisations de POSIX fichier. Vous devrez donc peut-être définir des autorisations pour le dossier du package de déploiement avant de créer l'archive de fichiers .zip.

Les exemples de commandes présentés dans les sections suivantes utilisent l'utilitaire Bundler pour ajouter des dépendances à votre package de déploiement. Pour installer Bundler, exécutez la commande suivante.

```
gem install bundler
```

Sections


- Dépendances dans Ruby
- Création d'un package de déploiement .zip sans dépendances
- Création d'un package de déploiement .zip avec dépendances
- Création d'une couche Ruby pour vos dépendances
- Création de packages de déploiement .zip avec des bibliothèques natives
- Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip

# Dépendances dans Ruby

Pour les fonctions Lambda qui utilisent l'exécution Ruby, une dépendance peut être n'importe quelle GEM Ruby. Lorsque vous déployez votre fonction à l'aide d'une archive .zip, vous pouvez soit ajouter ces dépendances à votre fichier .zip avec votre code de fonction, soit utiliser une couche Lambda. Une couche est un fichier .zip séparé qui peut contenir du code supplémentaire et d'autres contenus. Pour en savoir plus sur l'utilisation des couches Lambda, consultez Couches Lambda.

Le moteur d'exécution Ruby inclut le AWS SDK for Ruby. Si votre fonction utilise leSDK, vous n'avez pas besoin de l'associer à votre code. Toutefois, pour conserver le contrôle total de vos dépendances ou pour utiliser une version spécifique duSDK, vous pouvez l'ajouter au package de déploiement de votre fonction. Vous pouvez soit l'inclure SDK dans votre fichier .zip, soit l'ajouter à l'aide d'une couche Lambda. Les dépendances de votre fichier .zip ou des couches Lambda sont prioritaires sur les versions incluses dans l'exécution. Pour savoir quelle version de SDK for Ruby est incluse dans votre version d'exécution, consultezthe section called “Versions incluses dans Runtime SDK”.

Dans le cadre du modèle de responsabilité partagée AWS, vous êtes responsable de la gestion de toutes les dépendances dans les packages de déploiement de vos fonctions. Cela inclut l’application de mises à jour et de correctifs de sécurité. Pour mettre à jour les dépendances dans le package de déploiement de votre fonction, créez d’abord un nouveau fichier .zip, puis chargez-le sur Lambda. Pour plus d’informations, consultez Création d’un package de déploiement .zip avec dépendances et Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip.

## Création d’un package de déploiement .zip sans dépendances

Si le code de votre fonction ne comporte aucune dépendance, votre fichier .zip contient uniquement le fichier .rb contenant le code du gestionnaire de votre fonction. Utilisez votre utilitaire zip préféré pour créer un fichier .zip avec votre fichier .rb à la racine. Si le fichier .rb ne se trouve pas à la racine de votre fichier .zip, Lambda ne sera pas en mesure d'exécuter votre code.

Pour savoir comment déployer votre fichier .zip pour créer une nouvelle fonction Lambda ou mettre à jour une fonction Lambda existante, veuillez consulter la rubrique Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip.

## Création d’un package de déploiement .zip avec dépendances

Si votre code de fonction dépend de GEM Ruby supplémentaires, vous pouvez soit ajouter ces dépendances à votre fichier .zip avec votre code de fonction, soit utiliser une couche Lambda. Les

instructions de cette section vous indiquent comment inclure les dépendances dans votre package de déploiement .zip. Pour obtenir des instructions sur la façon d'inclure vos dépendances dans une couche, voir the section called “Création d'une couche Ruby pour vos dépendances”.

Supposons que votre code de fonction soit enregistré dans un fichier nommé `lambda_function.rb` dans le répertoire de votre projet. Les exemples de CLI commandes suivants créent un fichier .zip nommé `my_deployment_package.zip` contenant le code de votre fonction et ses dépendances.

Pour créer le package de déploiement

1. Dans le répertoire de votre projet, créez un fichier `Gemfile` pour y spécifier vos dépendances.

   ```
   bundle init
   ```

2. En utilisant l'éditeur de texte de votre choix, modifiez le fichier `Gemfile` pour spécifier les dépendances de votre fonction. Par exemple, pour utiliser la TZInfo gemme, modifiez-la `Gemfile` pour qu'elle ressemble à ce qui suit.

   ```
   source "https://rubygems.org"
   gem "tzinfo"
   ```

3. Exécutez la commande suivante pour installer les GEM spécifiées dans votre fichier `Gemfile` dans votre répertoire de projet. Cette commande définit `vendor/bundle` comme chemin par défaut pour les installations de GEM.

   ```
   bundle config set --local path 'vendor/bundle' && bundle install
   ```

   Vous devez visualiser des résultats similaires à ce qui suit.

   ```
   Fetching gem metadata from https://rubygems.org/..........
   Resolving dependencies...
   Using bundler 2.4.13
   Fetching tzinfo 2.0.6
   Installing tzinfo 2.0.6
   ...
   ```

   > **Note**
   >
   > Pour réinstaller globalement les GEM ultérieurement, exécutez la commande suivante.

```
bundle config set --local system 'true'
```

4. Créez une archive de fichiers .zip contenant le fichier `lambda_function.rb` avec le code du gestionnaire de votre fonction et les dépendances que vous avez installées à l'étape précédente.

```
zip -r my_deployment_package.zip lambda_function.rb vendor
```

Vous devez visualiser des résultats similaires à ce qui suit.

```
adding: lambda_function.rb (deflated 37%)
  adding: vendor/ (stored 0%)
  adding: vendor/bundle/ (stored 0%)
  adding: vendor/bundle/ruby/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/build_info/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

## Création d'une couche Ruby pour vos dépendances

Les instructions de cette section vous indiquent comment inclure les dépendances dans une couche. Pour obtenir des instructions sur la façon d'inclure vos dépendances dans votre package de déploiement, voir the section called "Création d'un package de déploiement .zip avec dépendances".

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d'exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

- `ruby/gems/2.7.0` (`GEM_PATH`)
- `ruby/lib` (`RUBYLIB`)

Par exemple, la structure du fichier .zip de votre couche peut ressembler à ce qui suit :

```
json.zip
```

```
# ruby/gems/2.7.0/
                | build_info
                | cache
                | doc
                | extensions
                | gems
                | # json-2.1.0
                # specifications
                  # json-2.1.0.gemspec
```

En outre, Lambda détecte automatiquement toutes les bibliothèques dans le répertoire `/opt/lib` et tous les fichiers binaires dans le répertoire `/opt/bin`. Pour que Lambda trouve correctement le contenu de votre couche, vous pouvez aussi créer une couche avec la structure suivante :

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Après avoir empaqueté votre couche, reportez-vous à the section called “Création et suppression de couches” et à the section called “Ajout de couches” pour terminer la configuration de votre couche.

## Création de packages de déploiement .zip avec des bibliothèques natives

De nombreuses GEM Ruby courantes telles que `nokogiri`, `nio4r` et `mysql` contiennent des extensions natives écrites en C. Lorsque vous ajoutez des bibliothèques contenant du code C à votre package de déploiement, vous devez créer votre package correctement pour vous assurer qu'il est compatible avec l'environnement d'exécution Lambda.

Pour les applications de production, nous vous recommandons de créer et de déployer votre code à l'aide de AWS Serverless Application Model (AWS SAM). AWS SAM Utilisez l'`sam build --use-container`option pour créer votre fonction dans un conteneur Docker de type Lambda. Pour en savoir plus sur l'utilisation AWS SAM de votre code de fonction pour déployer votre code de fonction, consultez la section Création d'applications dans le Guide du AWS SAM développeur.

Pour créer un package de déploiement .zip contenant des gemmes avec des extensions natives sans les utiliser AWS SAM, vous pouvez également utiliser un conteneur pour regrouper vos dépendances

dans un environnement identique à l'environnement d'exécution Lambda Ruby. Pour réaliser ces étapes, Docker doit être installé sur votre machine de génération. Pour en savoir plus sur l'installation de Docker, consultez Installation du moteur Docker.

Pour créer un package de déploiement .zip dans un conteneur Docker

1. Créez un dossier sur votre machine de génération locale pour y enregistrer votre conteneur. Dans ce dossier, créez un fichier nommé `dockerfile` et collez-y le code suivant.

   ```
   FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
   RUN gem update bundler
   CMD "/bin/bash"
   ```

2. Dans le dossier dans lequel vous avez créé votre fichier `dockerfile`, exécutez la commande suivante pour créer le conteneur Docker.

   ```
   docker build -t awsruby32 .
   ```

3. Accédez au répertoire du projet contenant le fichier `.rb` avec le code du gestionnaire de votre fonction et le fichier `Gemfile` spécifiant les dépendances de votre fonction. À partir de ce répertoire, exécutez la commande suivante pour démarrer le conteneur Lambda Ruby.

   Linux/MacOS

   ```
   docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
   ```

   > **Note**
   >
   > Sous macOS, il se peut qu'un avertissement vous informe que la plateforme de l'image demandée ne correspond pas à la plateforme de l'hôte détectée. Ignorez cet avertissement.

   Windows PowerShell

   ```
   docker run --rm -it -v ${pwd}:var/task -w /var/task awsruby32
   ```

   Lorsque votre conteneur démarre, vous devriez voir une invite bash.

```
bash-4.2#
```

4. Configurez l'utilitaire bundle pour installer les GEM spécifiées dans votre fichier `Gemfile` dans votre répertoire `vendor/bundle` local et installer vos dépendances.

```
bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
```

5. Créez le package de déploiement .zip avec le code de votre fonction et ses dépendances. Dans cet exemple, le fichier contenant le code du gestionnaire de votre fonction est nommé `lambda_function.rb`.

```
bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
```

6. Quittez le conteneur et retournez dans votre répertoire de projet local.

```
bash-4.2# exit
```

Vous pouvez désormais utiliser le package de déploiement du fichier .zip pour créer ou mettre à jour votre fonction Lambda. Consultez Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip.

## Création et mise à jour de fonctions Lambda Ruby à l'aide de fichiers .zip

Après avoir créé votre package de déploiement .zip, vous pouvez l'utiliser pour créer une nouvelle fonction Lambda ou mettre à jour une fonction existante. Vous pouvez déployer votre package .zip à l'aide de la console Lambda, AWS Command Line Interface du, et du Lambda. API Vous pouvez également créer et mettre à jour des fonctions Lambda à l’aide de l’ AWS Serverless Application Model (AWS SAM) et de AWS CloudFormation.

La taille maximale d’un package de déploiement .zip pour Lambda est de 250 Mo (décompressé). Notez que cette limite s’applique à la taille combinée de tous les fichiers que vous chargez, y compris les couches Lambda.

Le runtime Lambda a besoin d’une autorisation pour lire les fichiers de votre package de déploiement. Dans la notation octale des autorisations Linux, Lambda a besoin de 644 autorisations pour les fichiers non exécutables (rw-r--r--) et de 755 autorisations () pour les répertoires et les fichiers exécutables. rwxr-xr-x

Sous Linux et macOS, utilisez la commande chmod pour modifier les autorisations de fichiers sur les fichiers et les répertoires de votre package de déploiement. Par exemple, pour accorder les autorisations appropriées à un fichier exécutable, exécutez la commande suivante.

```
chmod 755 <filepath>
```

Pour modifier les autorisations relatives aux fichiers dans Windows, voir Définir, afficher, modifier ou supprimer des autorisations sur un objet dans la documentation Microsoft Windows.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de la console

Pour créer une nouvelle fonction, vous devez d'abord créer la fonction dans la console, puis charger votre archive .zip. Pour mettre à jour une fonction existante, ouvrez la page de votre fonction, puis suivez la même procédure pour ajouter votre fichier .zip mis à jour.

Si votre fichier .zip fait moins de 50 Mo, vous pouvez créer ou mettre à jour une fonction en chargeant le fichier directement à partir de votre ordinateur local. Pour les fichiers .zip de plus de 50 Mo, vous devez d'abord charger votre package dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS Management Console, consultez Getting started with Amazon S3. Pour télécharger des fichiers à l'aide de AWS CLI, voir Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Pour créer une nouvelle fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Choisissez Créer à partir de zéro.
3. Sous Informations de base, procédez comme suit :
   a. Pour Nom de la fonction, saisissez le nom de la fonction.
   b. Pour Exécution, sélectionnez l'exécution que vous souhaitez utiliser.

c. (Facultatif) Pour Architecture, choisissez l'architecture de l'ensemble des instructions pour votre fonction. L'architecture par défaut est x86_64. Assurez-vous que le package de déploiement .zip pour votre fonction est compatible avec l'architecture de l'ensemble d'instructions que vous sélectionnez.

4. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou en utiliser un existant.
5. Sélectionnez Create function (Créer une fonction). Lambda crée une fonction de base « Hello world » à l'aide de l'exécution de votre choix.

Pour charger une archive .zip à partir de votre ordinateur local (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger le fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez Fichier .zip.
5. Pour charger un fichier .zip, procédez comme suit :

   a. Sélectionnez Charger, puis choisissez votre fichier .zip dans le sélecteur de fichiers.

   b. Choisissez Ouvrir.

   c. Choisissez Save (Enregistrer).

Pour charger une archive .zip depuis un compartiment Amazon S3 (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger un nouveau fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez l'emplacement Amazon S3.
5. Collez le lien Amazon S3 URL de votre fichier .zip et choisissez Enregistrer.

## Mise à jour des fonctions du fichier .zip à l'aide de l'éditeur de code de la console

Pour certaines fonctions avec des packages de déploiement .zip, vous pouvez utiliser l'éditeur de code intégré de la console Lambda pour mettre à jour le code de votre fonction directement. Pour utiliser cette fonctionnalité, votre fonction doit répondre aux critères suivants :

- Votre fonction doit utiliser l'une des exécutions des langages interprétés (Python, Node.js ou Ruby).
- Le package de déploiement de votre fonction doit être inférieur à 3 Mo.

Le code des fonctions avec les packages de déploiement d'images de conteneurs ne peut pas être édité directement dans la console.

Pour mettre à jour le code de fonction à l'aide de l'éditeur de code de la console

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, sélectionnez votre fichier de code source et modifiez-le dans l'éditeur de code intégré.
4. Lorsque vous avez fini de modifier votre code, choisissez Déployer pour enregistrer vos modifications et mettre à jour votre fonction.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide du AWS CLI

Vous pouvez utiliser l'AWS CLI pour créer une nouvelle fonction ou pour mettre à jour une fonction existante à l'aide d'un fichier .zip. Utilisez la fonction de création et update-function-codeles commandes pour déployer votre package .zip. Si votre fichier .zip est inférieur à 50 Mo, vous pouvez charger le package .zip à partir d'un emplacement de fichier sur votre machine de génération locale. Pour les fichiers plus volumineux, vous devez charger votre package .zip à partir d'un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Si vous chargez votre fichier .zip depuis un compartiment Amazon S3 à l'aide de AWS CLI, le compartiment doit se trouver au même endroit Région AWS que votre fonction.

Pour créer une nouvelle fonction à l'aide d'un fichier .zip avec le AWS CLI, vous devez spécifier les éléments suivants :

- Le nom de votre fonction (`--function-name`)
- L'exécution de votre fonction (`--runtime`)
- Le nom de ressource Amazon (ARN) du rôle d'exécution de votre fonction (`--role`)
- Le nom de la méthode du gestionnaire dans votre code de fonction (`--handler`)

Vous devez également indiquer l'emplacement de votre fichier .zip. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda create-function --function-name myFunction \
--runtime ruby3.2 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez l'option `--code` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `S3ObjectVersion` pour les objets soumis à la gestion des versions.

```
aws lambda create-function --function-name myFunction \
--runtime ruby3.2 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Pour mettre à jour une fonction existante à l'aide duCLI, vous devez spécifier le nom de votre fonction à l'aide du `--function-name` paramètre. Vous devez également spécifier l'emplacement du fichier .zip que vous souhaitez utiliser pour mettre à jour votre code de fonction. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez les options `--s3-bucket` et `--s3-key` comme le montre l'exemple de commande suivant. Vous devez

uniquement utiliser le paramètre `--s3-object-version` pour les objets soumis à la gestion des versions.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide du Lambda API

Pour créer et mettre à jour des fonctions à l'aide d'une archive de fichiers .zip, effectuez les API opérations suivantes :

- CreateFunction
- UpdateFunctionCode

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS SAM

The AWS Serverless Application Model (AWS SAM) est une boîte à outils qui permet de rationaliser le processus de création et d'exécution d'applications sans serveur sur AWS. Vous définissez les ressources de votre application dans un JSON modèle YAML or et vous utilisez l'interface de ligne de AWS SAM commande (AWS SAM CLI) pour créer, empaqueter et déployer vos applications. Lorsque vous créez une fonction Lambda à partir d'un AWS SAM modèle, elle crée AWS SAM automatiquement un package de déploiement ou une image de conteneur .zip avec le code de votre fonction et les dépendances que vous spécifiez. Pour en savoir plus sur l'utilisation des fonctions Lambda AWS SAM pour créer et déployer des fonctions Lambda, consultez la section Getting started with AWS SAM dans le Guide du AWS Serverless Application Model développeur.

Vous pouvez également l'utiliser AWS SAM pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip existante. Pour créer une fonction Lambda à l'aide de AWS SAM, vous pouvez enregistrer votre fichier .zip dans un compartiment Amazon S3 ou dans un dossier local sur votre machine de génération. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS SAM modèle, la `AWS::Serverless::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l’aide d’une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`

- `CodeUri`- défini sur l'Amazon S3URI, le chemin d'accès au dossier local ou à l'FunctionCodeobjet du code de fonction
- `Runtime` : défini sur votre exécution choisie

Ainsi AWS SAM, si votre fichier .zip est supérieur à 50 Mo, vous n'avez pas besoin de le télécharger au préalable dans un compartiment Amazon S3. AWS SAM peut télécharger des packages .zip jusqu'à la taille maximale autorisée de 250 Mo (décompressés) à partir d'un emplacement sur votre machine de compilation locale.

Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS SAM, voir :Serverless AWS : :Function dans le manuel du développeur.AWS SAM

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip. Pour créer une fonction Lambda à partir d'un fichier .zip, vous devez d'abord charger votre fichier dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS CloudFormation modèle, la `AWS::Lambda::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l'aide d'une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `Code` : saisissez le nom du compartiment Amazon S3 et le nom du fichier .zip dans les champs `S3Bucket` et `S3Key`
- `Runtime` : défini sur votre exécution choisie

Le fichier .zip AWS CloudFormation généré ne peut pas dépasser 4 Mo. Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS CloudFormation, voir :Lambda AWS : :Function dans le guide de l'utilisateur.AWS CloudFormation

# Déploiement de fonctions Lambda en Ruby avec des images conteneurs

Il existe trois méthodes pour créer une image de conteneur pour une fonction Lambda Ruby :

- Utiliser une image AWS de base pour Ruby

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Ruby dans l'image.

- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Ruby dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

Rubriques


- AWS images de base pour Ruby
- Utiliser une image AWS de base pour Ruby
- Utilisation d'une autre image de base avec le client d'interface d'exécution


## AWS images de base pour Ruby

AWS fournit les images de base suivantes pour Ruby :

| Balises | Environnement d'exécution | Système d'exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| 3.3 | Ruby 3.3 | Amazon Linux | Dockerfile pour Ruby 3.3 sur GitHub | Non prévu |
| 3.2 | Ruby 3.2 | Amazon Linux 2 | Dockerfile pour Ruby 3.2 sur GitHub | Non prévu |

ECRRéférentiel Amazon : gallery.ecr.aws/lambda/ruby

## Utiliser une image AWS de base pour Ruby

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker
- Ruby

Création d'une image à partir d'une image de base

Pour créer une image de conteneur pour Ruby

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir example
cd example
```

2. Créez un nouveau fichier appelé `Gemfile`. C'est ici que vous listez les RubyGems packages requis pour votre application. Le AWS SDK for Ruby est disponible auprès de RubyGems. Vous devez choisir des gemmes AWS de service spécifiques à installer. Par exemple, pour utiliser la gem Ruby pour Lambda, votre Gemfile doit ressembler à ceci :

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

   Sinon, la gemme aws-sdk contient toutes les gemmes de service disponibles. AWS Cette gem est très large. Nous vous recommandons de ne l'utiliser que si vous dépendez de nombreux AWS services.
3. Installez les dépendances spécifiées dans le fichier Gemfile à l'aide de l'installation groupée.

```
bundle install
```

4. Créez un nouveau fichier appelé `lambda_function.rb`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

   Example Fonction Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Créez un nouveau fichier Docker. Voici un exemple de Dockerfile qui utilise une image de base AWS. Le Dockerfile utilise la configuration suivante :

   - Définissez la `FROM` propriété sur URI l'image de base.
   - Utilisez la COPY commande pour copier le code de fonction et les dépendances d'exécution dans une variable `{LAMBDA_TASK_ROOT}` d'environnement définie par Lambda.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

Example Dockerfile

```
FROM public.ecr.aws/lambda/ruby:3.2

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

1. Démarrez votre image Docker à l'aide de la commande docker run. Dans cet exemple, `docker-image` est le nom de l'image et `test` est la balise.

   ```
   docker run --platform linux/amd64 -p 9000:8080 --read-only docker-image:test
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

2. À partir d'une nouvelle fenêtre de terminal, publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image dans Amazon ECR reste la même. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilisation d’une autre image de base avec le client d’interface d’exécution

Si vous utilisez une image de base uniquement pour le système d’exploitation ou une autre image de base, vous devez inclure le client d'interface d'exécution dans votre image. Le client d’interface d’exécution étend leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l’interaction entre Lambda et votre code de fonction.

Installez le client d'interface d'exécution Lambda pour Ruby à l'aide du gestionnaire de packages RubyGems .org :

```
gem install aws_lambda_ric
```

Vous pouvez également télécharger le client d'interface d'exécution Ruby depuis GitHub. Le client d’interface d’exécution prend en charge les versions 2.5.x à 2.7.x de Ruby.

L'exemple suivant montre comment créer une image de conteneur pour Ruby à l'aide d'une image non AWS basique. L’exemple Dockerfile utilise une image de base Ruby officielle. Le Dockerfile inclut le client d’interface d’exécution.

### Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- Docker
- Ruby

### Création d'une image à partir d'une image de base alternative

Pour créer une image de conteneur pour Ruby au moyen d'une autre image de base

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

   ```
   mkdir example
   cd example
   ```

2. Créez un nouveau fichier appelé `Gemfile`. C'est ici que vous listez les RubyGems packages requis pour votre application. Le AWS SDK for Ruby est disponible auprès de RubyGems. Vous devez choisir des gemmes AWS de service spécifiques à installer. Par exemple, pour utiliser la gem Ruby pour Lambda, votre Gemfile doit ressembler à ceci :

   ```
   source 'https://rubygems.org'

   gem 'aws-sdk-lambda'
   ```

   Sinon, la gemme aws-sdk contient toutes les gemmes de service disponibles. AWS Cette gem est très large. Nous vous recommandons de ne l'utiliser que si vous dépendez de nombreux AWS services.

3. Installez les dépendances spécifiées dans le fichier Gemfile à l'aide de l'installation groupée.

   ```
   bundle install
   ```

4. Créez un nouveau fichier appelé `lambda_function.rb`. Vous pouvez ajouter l'exemple de code de fonction suivant au fichier pour le tester, ou utiliser le vôtre.

Example Fonction Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Créez un nouveau fichier Docker. Le Dockerfile suivant utilise une image de base Ruby au lieu d'une image de base AWS. Le Dockerfile inclut le client d'interface d'exécution pour Ruby, ce qui rend l'image compatible avec Lambda. Vous pouvez également ajouter le client de l'interface d'exécution au Gemfile de votre application.

   - Définissez la propriété `FROM` sur l'image de base Ruby.
   - Créez un répertoire pour le code de fonction et une variable d'environnement pointant vers ce répertoire. Dans cet exemple, le répertoire est`/var/task`, qui reflète l'environnement d'exécution Lambda. Cependant, vous pouvez choisir n'importe quel répertoire pour le code de fonction car le Dockerfile n'utilise pas d'image de AWS base.
   - Définissez le `ENTRYPOINT` sur le module que vous souhaitez que le conteneur Docker exécute lorsqu'il démarre. Dans ce cas, le module est le client d'interface d'exécution.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM ruby:2.7

# Install the runtime interface client for Ruby
RUN gem install aws_lambda_ric

# Add the runtime interface client to the PATH
ENV PATH="/usr/local/bundle/bin:${PATH}"
```

```
# Create a directory for the Lambda function
ENV LAMBDA_TASK_ROOT=/var/task
RUN mkdir -p ${LAMBDA_TASK_ROOT}
WORKDIR ${LAMBDA_TASK_ROOT}

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "aws_lambda_ric" ]

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester l'image localement. Vous pouvez intégrer l'émulateur à votre image ou utiliser la procédure suivante pour l'installer sur votre machine locale.

Pour installer et exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Depuis le répertoire de votre projet, exécutez la commande suivante pour télécharger l'émulateur d'interface d'exécution (architecture x86-64) GitHub et l'installer sur votre machine locale.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Pour installer l'émulateur arm64, remplacez le GitHub référentiel de URL la commande précédente par le suivant :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }

   $downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie"
   $destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
   Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
   ```

   Pour installer l'émulateur arm64, remplacez `$downloadLink` par ce qui suit :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

2. Démarrez votre image Docker à l'aide de la commande docker run. Notez ce qui suit :

- `docker-image` est le nom de l'image et `test` est la balise.
- `aws_lambda_ric lambda_function.LambdaFunction::Handler.process` est le `ENTRYPOINT` suivi du `CMD` depuis votre Dockerfile.

Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
 9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    --read-only \
    docker-image:test \
        aws_lambda_ric lambda_function.LambdaFunction::Handler.process
```

PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
 9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
--read-only `
docker-image:test `
    aws_lambda_ric lambda_function.LambdaFunction::Handler.process
```

Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

3. Publiez un événement au point de terminaison local.

Linux/macOS

Sous Linux et macOS, exécutez la commande `curl` suivante :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenez l'ID du conteneur.

```
docker ps
```

5. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.

- Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
- `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.

4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemple :

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   aws lambda create-function \
     --function-name hello-world \
     --package-type Image \
     --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
     --role arn:aws:iam::111122223333:role/lambda-ex
   ```

   > **Note**
   >
   > Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image dans Amazon ECR reste la même. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

# Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Ruby

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'invocation, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `get_remaining_time_in_millis` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.

Propriétés du contexte

- `function_name` – Nom de la fonction Lambda.
- `function_version` – Version de la fonction.
- `invoked_function_arn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `memory_limit_in_mb` – Quantité de mémoire allouée à la fonction.
- `aws_request_id` – Identifiant de la demande d'invocation.
- `log_group_name` – Groupe de journaux pour la fonction.
- `log_stream_name` – Flux de journal de l'instance de fonction.
- `deadline_ms` – Date d'expiration de l'exécution, exprimée en millisecondes au format horaire Unix.
- `identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `client_context` – (applications mobiles) Contexte client fourni à Lambda par l'application client.

# Enregistrez et surveillez les fonctions Ruby Lambda

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et envoie les journaux à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des informations sur chaque invocation au flux de journaux et transmet les journaux et autres sorties provenant du code de votre fonction. Pour de plus amples informations, veuillez consulter Utilisation des CloudWatch journaux et journaux avec Lambda.

Cette page explique comment générer une sortie de journal à partir du code de votre fonction Lambda et comment accéder aux journaux à l'aide de la AWS Command Line Interface console Lambda ou de la console. CloudWatch

Sections


- Création d'une fonction qui renvoie des journaux
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux
- Utilisation de la bibliothèque Ruby Logger


## Création d'une fonction qui renvoie des journaux

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des déclarations `puts` ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant consigne les valeurs des variables d'environnement et l'objet d'événement.

Example lambda_function.rb

```
# lambda_function.rb

def handler(event:, context:)
    puts "## ENVIRONMENT VARIABLES"
    puts ENV.to_a
    puts "## EVENT"
    puts event.to_a
```

```
end
```

Example format des journaux

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
 'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c',
 'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
 Duration: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1   SegmentId: 07f5xmpl2d1f6f85
 Sampled: true
```

L'exécution Ruby enregistre les lignes `START`, `END` et `REPORT` pour chaque invocation. La ligne de rapport fournit les détails suivants.

REPORTchamps de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled : pour les demandes suivies, résultat de l'échantillonnage.

Pour obtenir des journaux plus détaillés, utilisez la bibliothèque the section called "Utilisation de la bibliothèque Ruby Logger".

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

## Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
```

```
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

## Utilisation de la bibliothèque Ruby Logger

La bibliothèque Ruby logger renvoie des journaux simplifiés, faciles à lire. Utilisez l'utilitaire logger pour obtenir des informations détaillées, des messages et des codes d'erreur relatifs à votre fonction.

```
# lambda_function.rb

require 'logger'

def handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info('## ENVIRONMENT VARIABLES')
  logger.info(ENV.to_a)
  logger.info('## EVENT')
  logger.info(event)
  event.to_a
end
```

La sortie de `logger` inclut le niveau de journal, l'horodatage et l'ID de demande.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO]  2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
 ENVIRONMENT VARIABLES

[INFO]  2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125
    environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
 'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d',
 'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})

[INFO]  2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT

[INFO]  2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    {'key':
 'value'}

END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
```

```
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125  Duration: 2.75 ms   Billed
 Duration: 3 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370   SegmentId: 073cxmpl3e442861
 Sampled: true
```

# Instrumentation du code Ruby dans AWS Lambda

Lambda s'intègre AWS X-Ray pour vous permettre de suivre, de déboguer et d'optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsqu'elle traverse les ressources de votre application, depuis le frontend API jusqu'au stockage et à la base de données sur le backend. En ajoutant simplement la SDK bibliothèque X-Ray à votre configuration de build, vous pouvez enregistrer les erreurs et le temps de latence pour chaque appel que votre fonction fait à un AWS service.

Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.

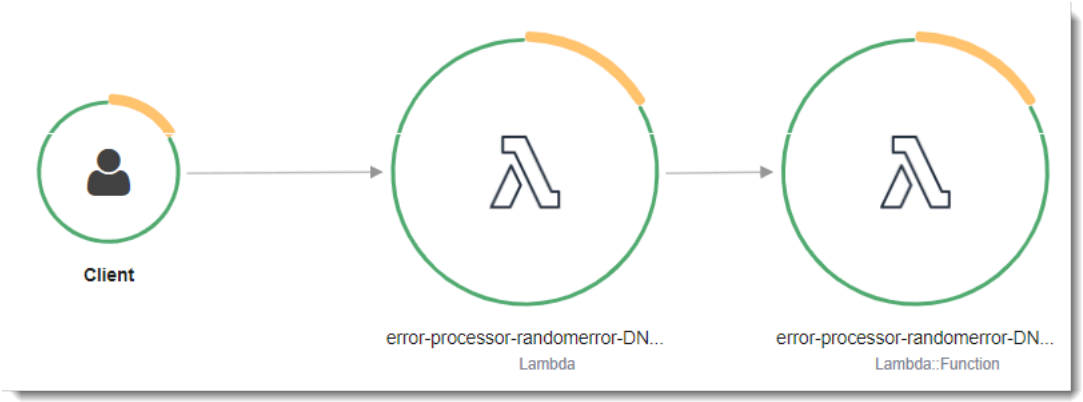
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

> **Tarification**
>
> Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Dans le cas contraire, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

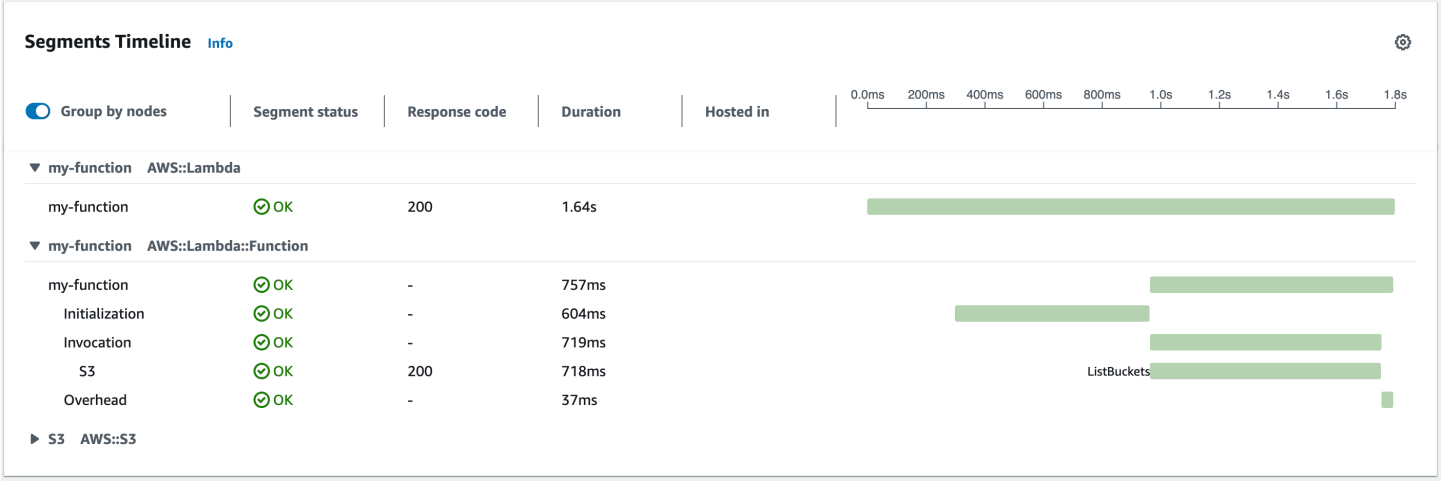
X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine

de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.

- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer les détails des appels que votre gestionnaire fait à d'autres ressources et services, utilisez le X-Ray SDK for Ruby. Pour obtenir leSDK, ajoutez le `aws-xray-sdk` package aux dépendances de votre application.

Example blank-ruby/function/Gemfile

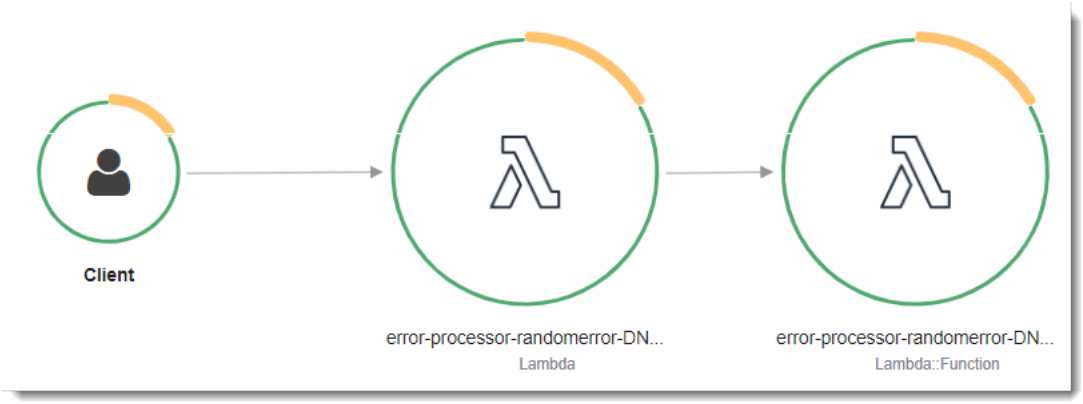
```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Pour instrumenter AWS SDK les clients, vous devez avoir besoin du `aws-xray-sdk/lambda` module après avoir créé un client dans le code d'initialisation.

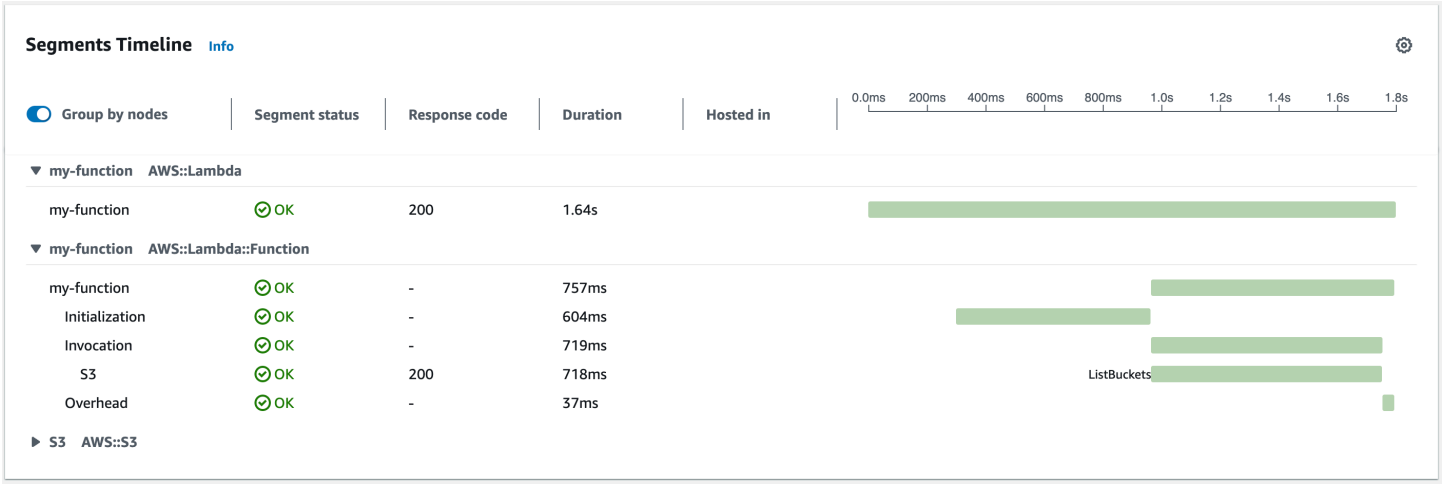
Example blank-ruby/function/lambda_function.rb — Suivi d'un client AWS SDK

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez The X-Ray SDK for Ruby dans le manuel du AWS X-Ray développeur.

Sections

- Activer le traçage actif avec le Lambda API
- Activation du suivi actif avec AWS CloudFormation
- Stockage des dépendances d’exécution dans une couche

## Activer le traçage actif avec le Lambda API

Pour gérer la configuration du suivi avec le AWS CLI ou AWS SDK, utilisez les API opérations suivantes :

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant permet d'activer le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
```

```
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

## Stockage des dépendances d'exécution dans une couche

Si vous utilisez le X-Ray SDK pour instrumenter votre code de fonction chez les AWS SDK clients, votre package de déploiement peut devenir assez volumineux. Pour éviter de télécharger les dépendances d'exécution chaque fois que vous mettez à jour le code de votre fonction, empaquetez le X-Ray SDK dans une couche Lambda.

L'exemple suivant montre une `AWS::Serverless::LayerVersion` ressource qui stocke X-Ray SDK for Ruby.

Example template.yml : couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-ruby-lib
      Description: Dependencies for the blank-ruby sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - ruby2.5
```

Avec cette configuration, vous ne mettez à jour les fichiers de couche de bibliothèque que si vous modifiez vos dépendances d'exécution. Étant donné que le package de déploiement de la fonction contient uniquement votre code, cela peut contribuer à réduire les temps de chargement.

La création d'une couche de dépendances nécessite des modifications de génération pour créer l'archive des couches avant le déploiement. Pour un exemple fonctionnel, consultez l'exemple d'application blank-ruby.

# Création de fonctions Lambda avec Java

Vous pouvez exécuter du code Java dans AWS Lambda. Lambda fournit des runtimes pour Java qui exécutent votre code afin de traiter des événements. Votre code s'exécute dans un environnement Amazon Linux qui inclut les AWS informations d'identification d'un rôle AWS Identity and Access Management (IAM) que vous gérez.

Lambda prend en charge les environnements d’exécution Java suivants.

| Nom | Identifiant | Système d’exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Java 21 | `java21` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Java 17 | `java17` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Java 11 | `java11` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Java 8 | `java8.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

Lambda fournit les bibliothèques suivantes pour les fonctions Java :

- com.amazonaws : aws-lambda-java-core (obligatoire) — Définit les interfaces des méthodes du gestionnaire et l'objet de contexte que le moteur d'exécution transmet au gestionnaire. Si vous définissez vos propres types d’entrée, c’est la seule bibliothèque dont vous avez besoin.
- com.amazonaws : aws-lambda-java-events — Types d'entrée pour les événements provenant de services qui invoquent des fonctions Lambda.
- com.amazonaws : aws-lambda-java-log 4j2 — Une bibliothèque d'ajout pour Apache Log4j 2 que vous pouvez utiliser pour ajouter l'ID de demande pour l'appel en cours dans vos journaux de fonctions.
- AWS SDKpour Java 2.0 — Le langage AWS SDK de programmation officiel Java.

> **Important**
>
> N'utilisez pas de composants privés de JDKAPI, tels que des champs privés, des méthodes ou des classes. Les API composants non publics peuvent être modifiés ou supprimés dans n'importe quelle mise à jour, ce qui peut entraîner l'interruption de votre application.

Pour créer une fonction Java

1. Ouvrez la console Lambda.
2. Sélectionnez Créer une fonction.
3. Configurez les paramètres suivants :
   - Nom de la fonction : saisissez le nom de la fonction.
   - Runtime : Choisissez Java 21.
4. Sélectionnez Créer une fonction.
5. Pour configurer un événement de test, choisissez Test.
6. Dans Event name (Nom de l'événement), saisissez **`test`**.
7. Sélectionnez Save Changes.
8. Pour invoquer la fonction, choisissez Test.

La console crée une fonction Lambda avec une classe de gestionnaire nommée `Hello`. Étant donné que Java est un langage compilé, vous ne pouvez pas afficher ou modifier le code source dans la console Lambda, mais vous pouvez modifier sa configuration, l'invoquer et configurer des déclencheurs.

> **Note**
>
> Pour démarrer le développement d'applications dans votre environnement local, déployez l'un des exemples d'applications disponibles dans le GitHub référentiel de ce guide.

La classe `Hello` comporte une fonction nommée `handleRequest` qui accepte un objet d'événement et un objet de contexte. Il s'agit de la fonction de gestionnaire que Lambda appelle lors de l'invocation de la fonction. Le runtime de la fonction Java obtient les événements d'invocations

à partir de Lambda et les transmet au gestionnaire. Dans la configuration de fonction, la valeur de gestionnaire est `example.Hello::handleRequest`.

Pour mettre à jour le code de la fonction, vous créez un package de déploiement, qui est une archive .zip contenant le code de votre fonction. Au fur et à mesure du développement de votre fonction, nous vous conseillons de stocker le code de votre fonction dans le contrôle de code source, d'ajouter des bibliothèques et d'automatiser les déploiements. Commencez par créer un package de déploiement et mettre à jour votre code dans la ligne de commande.

Le runtime de la fonction transmet un objet de contexte au gestionnaire, en plus de l'événement d'invocation. L'objet de contexte contient des informations supplémentaires sur l'invocation, la fonction et l'environnement d'exécution. Des informations supplémentaires sont disponibles dans les variables d'environnement.

Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs. La fonction runtime envoie les détails de chaque appel à CloudWatch Logs. Il relaie tous les journaux que votre fonction génère pendant l'invocation. Si votre fonction renvoie une erreur, Lambda met en forme l'erreur et la renvoie à l'appelant.

Rubriques

- Définir le gestionnaire de fonctions Lambda en Java
- Déployez des fonctions Java Lambda avec des archives .zip ou de fichiers JAR
- Déployer des fonctions Lambda en Java avec des images conteneurs
- Utilisation de couches pour les fonctions Java Lambda
- Améliorer les performances de démarrage avec Lambda SnapStart
- Personnalisation du comportement de démarrage de l'exécution Java pour les fonctions Lambda
- Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions Java
- Enregistrez et surveillez les fonctions Java Lambda
- Instrumentation du code Java dans AWS Lambda
- Exemples d'applications Java pour AWS Lambda

# Définir le gestionnaire de fonctions Lambda en Java

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Le GitHub référentiel de ce guide fournit des easy-to-deploy exemples d'applications illustrant différents types de gestionnaires. Pour de plus amples informations, veuillez consulter la fin de cette rubrique.

Sections


- Exemple de gestionnaire : exécutions Java 17
- Exemple de gestionnaire : exécutions Java 11 et antérieures
- Code d'initialisation
- Choix des types d'entrée et de sortie
- Interfaces du gestionnaire
- Bonnes pratiques en matière de code pour les fonctions Java Lambda
- Exemple de code de gestionnaire


## Exemple de gestionnaire : exécutions Java 17

Dans l'exemple Java 17 suivant, une classe nommée `HandlerIntegerJava17` définit une méthode de gestionnaire nommée `handleRequest`. La méthode du gestionnaire prend en compte les entrées suivantes :

- Un `IntegerRecord`, qui est un enregistrement Java personnalisé qui représente les données d'un événement. Dans cet exemple, nous définissons `IntegerRecord` comme suit :

```
record IntegerRecord(int x, int y, String message) {
}
```

- Un objet de contexte, qui fournit des méthodes et des propriétés fournissant des informations sur l'invocation, la fonction et l'environnement d'exécution.

Supposons que nous voulions écrire une fonction qui enregistre le `message` de l'entrée `IntegerRecord` et renvoie la somme de `x` et `y`. Le code de la fonction est le suivant :

Example HandlerIntegerJava17.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

// Handler value: example.HandlerInteger
public class HandlerIntegerJava17 implements RequestHandler<IntegerRecord, Integer>{

  @Override
  /*
   * Takes in an InputRecord, which contains two integers and a String.
   * Logs the String, then returns the sum of the two Integers.
   */
  public Integer handleRequest(IntegerRecord event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    logger.log("String found: " + event.message());
    return event.x() + event.y();
  }
}

record IntegerRecord(int x, int y, String message) {
}
```

Vous spécifiez quelle méthode vous voulez que Lambda invoque en définissant le paramètre de gestionnaire sur la configuration de votre fonction. Vous pouvez exprimer le gestionnaire dans les formats suivants :

- *package*.*Class*::*method* – Format complet. olpPar exemple : `example.Handler::handleRequest`.
- *package*.*Class* – Format abrégé pour les catégories qui implémentent une interface de gestionnaire. olpPar exemple : `example.Handler`.

Lorsque Lambda appelle votre gestionnaire, le moteur d'exécution Lambda reçoit un événement sous forme de chaîne JSON formatée et le convertit en objet. Dans l'exemple précédent, un exemple d'événement peut s'apparenter à ce qui suit :

Example event.json

```
{
  "x": 1,
  "y": 20,
  "message": "Hello World!"
}
```

Vous pouvez enregistrer ce fichier et tester votre fonction localement à l'aide de la commande suivante AWS Command Line Interface (CLI) :

```
aws lambda invoke --function-name function_name --payload file://event.json out.json
```

## Exemple de gestionnaire : exécutions Java 11 et antérieures

Lambda prend en charge les enregistrements dans les exécutions Java 17 et ultérieures. Dans toutes les exécutions Java, vous pouvez utiliser une classe pour représenter les données des événements. L'exemple suivant prend une liste d'entiers et un objet contextuel en entrée, et renvoie la somme de tous les entiers de la liste.

Example handler.java

Dans l'exemple suivant, une classe nommée `Handler` définit une méthode de gestionnaire nommée `handleRequest`. La méthode de gestionnaire prend un événement et un objet de contexte en entrée et renvoie une chaîne.

Example HandlerList.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.List;

// Handler value: example.HandlerList
public class HandlerList implements RequestHandler<List<Integer>, Integer>{

  @Override
  /*
```

```
    * Takes a list of Integers and returns its sum.
    */
  public Integer handleRequest(List<Integer> event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    logger.log("EVENT TYPE: " + event.getClass().toString());
    return event.stream().mapToInt(Integer::intValue).sum();
  }
}
```

Pour plus d'exemples, voir Exemple de code de gestionnaire.

## Code d'initialisation

Lambda exécute votre code statique et le constructeur de classe pendant la phase d'initialisation avant d'invoquer votre fonction pour la première fois. Les ressources créées pendant l'initialisation restent en mémoire entre les invocations et peuvent être réutilisées par le gestionnaire des milliers de fois. Vous pouvez ajouter du code d'initialisation en dehors de votre méthode de gestionnaire principale pour économiser du temps de calcul et réutiliser les ressources sur plusieurs invocations.

Dans l'exemple suivant, le code d'initialisation du client se trouve en dehors de la méthode du gestionnaire principal. L'exécution initialise le client avant que la fonction n'exécute son premier événement. Les événements suivants sont beaucoup plus rapides, car Lambda n'a pas besoin d'initialiser à nouveau le client.

Example handler.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.Map;

import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.GetAccountSettingsResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String> {
```

```
    private static final LambdaClient lambdaClient = LambdaClient.builder().build();

    @Override
    public String handleRequest(Map<String,String> event, Context context) {

        LambdaLogger logger = context.getLogger();
        logger.log("Handler invoked");

        GetAccountSettingsResponse response = null;
        try {
            response = lambdaClient.getAccountSettings();
        } catch(LambdaException e) {
            logger.log(e.getMessage());
        }
        return response != null ? "Total code size for your account is " +
 response.accountLimit().totalCodeSize() + " bytes" : "Error";
    }
}
```

## Choix des types d’entrée et de sortie

Vous spécifiez le type d’objet auquel l’événement est mappé dans la signature de la méthode de gestion. Dans l'exemple précédent, le moteur d'exécution Java désérialise l'événement dans un type qui implémente l'`Map<String,String>`interface. String-to-stringles cartes fonctionnent pour les événements plats tels que les suivants :

Example Event.json – Données météorologiques.

```
{
  "temperatureK": 281,
  "windKmh": -3,
  "humidityPct": 0.55,
  "pressureHPa": 1020
}
```

Cependant, la valeur de chaque champ doit être une chaîne ou un nombre. Si l’événement inclut un champ qui a un objet comme valeur, le moteur d’exécution ne peut pas le désérialiser et renvoie une erreur.

Choisissez un type d’entrée qui fonctionne avec les données d’événement traitées par votre fonction. Vous pouvez utiliser un type de base, un type générique ou un type bien défini.

Types d'entrée

- `Integer`, `Long`, `Double`, etc. – L'événement est un nombre sans mise en forme supplémentaire, par exemple, `3.5`. Le moteur d'exécution convertit la valeur en un objet du type spécifié.
- `String`— L'événement est une JSON chaîne, y compris des guillemets, par exemple,. `"My string."` Le moteur d'exécution convertit la valeur (sans guillemets) en objet `String`.
- *`Type`*, `Map<String,`*`Type`*`>` etc. — L'événement est un JSON objet. Le moteur d'exécution le désérialise en un objet du type ou de l'interface spécifié.
- `List<Integer>`, `List<String>`, `List<Object>`, etc. — L'événement est un JSON tableau. Le moteur d'exécution le désérialise en un objet du type ou de l'interface spécifié.
- `InputStream`— L'événement est de n'importe quel JSON type. Le moteur d'exécution transmet un flux d'octets du document au gestionnaire sans modification. Vous désérialisez l'entrée et écrivez la sortie dans un flux de sortie.
- Type de bibliothèque : pour les événements envoyés par AWS les services, utilisez les types de la aws-lambda-java-eventsbibliothèque.

Si vous définissez votre propre type d'entrée, il doit s'agir d'un ancien objet Java désérialisable et mutable (POJO), avec un constructeur et des propriétés par défaut pour chaque champ de l'événement. Les clés de l'événement qui ne correspondent pas à une propriété, ainsi que les propriétés qui ne sont pas incluses dans l'événement sont supprimées sans erreur.

Le type de sortie peut être un objet ou `void`. Le moteur d'exécution sérialise les valeurs de retour dans le texte. Si le résultat est un objet avec des champs, le moteur d'exécution le sérialise dans un JSON document. S'il s'agit d'un type qui enveloppe une valeur primitive, le moteur d'exécution renvoie une représentation textuelle de cette valeur.

## Interfaces du gestionnaire

La aws-lambda-java-corebibliothèque définit deux interfaces pour les méthodes de gestion. Utilisez les interfaces fournies pour simplifier la configuration du gestionnaire et valider la signature de la méthode du gestionnaire au moment de la compilation.

- com.amazonaws.services.lambda.runtime. RequestHandler
- com.amazonaws.services.lambda.runtime. RequestStreamHandler

L'interface `RequestHandler` est un type générique qui prend deux paramètres : le type d'entrée et le type de sortie. Les deux types doivent être des objets. Lorsque vous utilisez cette interface, le moteur d'exécution Java désérialise l'événement dans un objet avec le type d'entrée et sérialise la sortie en texte. Utilisez cette interface lorsque la sérialisation intégrée fonctionne avec vos types d'entrée et de sortie.

Example Handler.java – Interface du gestionnaire.

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String>{
  @Override
  public String handleRequest(Map<String,String> event, Context context)
```

Pour utiliser votre propre sérialisation, implémentez l'interface `RequestStreamHandler`. Avec cette interface, Lambda transmet à votre gestionnaire un flux d'entrée et un flux de sortie. Le gestionnaire lit les octets du flux d'entrée, écrit dans le flux de sortie et renvoie une valeur vide.

L'exemple Java 21 suivant montre comment utiliser une fonction Lambda pour traiter les commandes. L'exemple utilise des types de lecteur et d'écriture mis en mémoire tampon pour travailler avec les flux d'entrée et de sortie, et montre comment définir des enregistrements Java personnalisés à utiliser dans votre fonction.

Example HandlerStream.java

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.List;

public class HandlerStream implements RequestStreamHandler {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public void handleRequest(InputStream input, OutputStream output, Context context)
 throws IOException {
```

```
        Order order = objectMapper.readValue(input, Order.class);

        processOrder(order);
        OrderAccepted orderAccepted = new OrderAccepted(order.orderId);

        objectMapper.writeValue(output, orderAccepted);
    }

    private void processOrder(Order order) {
        // business logic
    }

    public record Order(@JsonProperty("orderId") String orderId, @JsonProperty("items")
 List<Item> items) { }

    public record Item(@JsonProperty("name") String name, @JsonProperty("quantity")
 Integer quantity) { }

    public record OrderAccepted(@JsonProperty("orderId") String orderId) { }
}
```

## Bonnes pratiques en matière de code pour les fonctions Java Lambda

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire.
- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution. Par exemple, préférez les infrastructures d'injection (IoC) de dépendances Java plus simples comme Dagger ou Guice, à des plus complexes comme Spring Framework.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de

déploiement avant l'invocation. Pour les fonctions créées en Java, évitez de télécharger l'intégralité de la AWS SDK bibliothèque dans le cadre de votre package de déploiement. Dépendez plutôt de manière sélective des modules qui récupèrent les composants dont SDK vous avez besoin (par exemple DynamoDB, les SDK modules Amazon S3 et les bibliothèques principales Lambda).

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles

que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.

- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

- Évitez d'utiliser le DNS cache Java. Les fonctions Lambda mettent déjà en cache DNS les réponses. Si vous utilisez un autre DNS cache, vous risquez de rencontrer des délais de connexion.

  La `java.util.logging.Logger` classe peut activer indirectement le JVM DNS cache. Pour remplacer les paramètres par défaut, définissez networkaddress.cache.ttl sur 0 avant l'initialisation. `logger` Exemple :

```
public class MyHandler {
  // first set TTL property
  static{
   java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
  }
 // then instantiate logger
  var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

- Réduisez le temps que Lambda met à décompresser les packages de déploiement créés dans Java en plaçant vos fichiers `.jar` de dépendance dans un répertoire /lib distinct. Cette solution est plus rapide que le placement de tout le code de votre fonction dans un seul fichier .jar comprenant une multitude de fichiers `.class`. Pour obtenir des instructions, consultez Déployez des fonctions Java Lambda avec des archives .zip ou de fichiers JAR.

## Exemple de code de gestionnaire

Le GitHub référentiel de ce guide inclut des exemples d'applications illustrant l'utilisation de différents types de gestionnaires et d'interfaces. Chaque exemple d'application inclut des scripts pour faciliter le déploiement et le nettoyage, un AWS SAM modèle et des ressources de support.

Exemples d'applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d'événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

Les `s3-java` applications `java-events` et prennent un événement AWS de service en entrée et renvoient une chaîne. L'application `java-basic` comprend plusieurs types de gestionnaires :

- Handler.java – Prend une `Map<String,String>` comme entrée.
- HandlerInteger.java — Prend un `Integer` comme entrée.
- HandlerList.java — Prend un `List<Integer>` comme entrée.
- HandlerStream.java — Prend un `InputStream` et `OutputStream` comme entrée.
- HandlerString.java — Prend un `String` comme entrée.
- HandlerWeatherData.java — Prend un type personnalisé en entrée.

Pour tester différents types de gestionnaires, il suffit de modifier la valeur du gestionnaire dans le AWS SAM modèle. Pour obtenir des instructions détaillées, consultez le fichier readme de l'exemple d'application.

# Déployez des fonctions Java Lambda avec des archives .zip ou de fichiers JAR

Le code de votre AWS Lambda fonction se compose de scripts ou de programmes compilés et de leurs dépendances. Pour déployer votre code de fonction vers Lambda, vous utilisez un package de déploiement. Lambda prend en charge deux types de packages de déploiement : les images conteneurs et les archives de fichiers .zip.

Cette page explique comment créer votre package de déploiement sous forme de fichier .zip ou Jar, puis utiliser le package de déploiement pour déployer votre code de fonction à AWS Lambda l'aide de AWS Command Line Interface (AWS CLI).

Sections


- Prérequis
- Outils et bibliothèques
- Création d’un package de déploiement avec Gradle
- Création d’une couche Java pour vos dépendances
- Création d’un package de déploiement avec Maven
- Chargement d’un package de déploiement avec la console Lambda
- Téléchargement d'un package de déploiement à l'aide du AWS CLI
- Téléchargement d'un package de déploiement avec AWS SAM


## Prérequis

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

## Outils et bibliothèques

Lambda fournit les bibliothèques suivantes pour les fonctions Java :

- com.amazonaws : aws-lambda-java-core (obligatoire) — Définit les interfaces des méthodes du gestionnaire et l'objet de contexte que le moteur d'exécution transmet au gestionnaire. Si vous définissez vos propres types d'entrée, c'est la seule bibliothèque dont vous avez besoin.
- com.amazonaws : aws-lambda-java-events — Types d'entrée pour les événements provenant de services qui invoquent des fonctions Lambda.
- com.amazonaws : aws-lambda-java-log 4j2 — Une bibliothèque d'ajout pour Apache Log4j 2 que vous pouvez utiliser pour ajouter l'ID de demande pour l'appel en cours dans vos journaux de fonctions.
- AWS SDKpour Java 2.0 — Le langage AWS SDK de programmation officiel Java.

Ces bibliothèques sont disponibles via le Référentiel central Maven. Ajoutez-les à votre définition de build comme suit :

Gradle

```
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'
}
```

Maven

```
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.2.2</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-events</artifactId>
      <version>3.11.1</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-log4j2</artifactId>
      <version>1.5.1</version>
    </dependency>
```

```
  </dependencies>
```

Pour créer un package de déploiement, compilez le code de votre fonction et ses dépendances dans un seul fichier .zip ou un fichier Java Archive (JAR). Pour Gradle, utilisez le type de build `Zip`. Pour Apache Maven, utilisez le plug-in Maven Shade. Pour télécharger votre package de déploiement, utilisez la console Lambda, le Lambda ou API (). AWS Serverless Application Model AWS SAM

> **Note**
>
> Pour que la taille de votre package de déploiement reste petite, empaquetez les dépendances de votre fonction en couches. Les couches vous permettent de gérer vos dépendances de manière indépendante. Elles peuvent être utilisées par plusieurs fonctions et partagées avec d'autres comptes. Pour de plus amples informations, veuillez consulter Couches Lambda.

## Création d'un package de déploiement avec Gradle

Afin de créer un package de déploiement avec le code et les dépendances de votre fonction dans Gradle, utilisez le type de build `Zip`. Voici un exemple tiré d'un fichier exemple build.gradle complet :

Example build.gradle – Tâche de génération

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
}
```

Cette configuration de build donne lieu à un package de déploiement dans le répertoire `build/distributions`. Dans l'instruction `into('lib')`, la tâche `jar` assemble une archive jar contenant vos classes principales dans un dossier nommé `lib`. De plus, la tâche `configurations.runtimeClassPath` copie les bibliothèques de dépendances du chemin de classe de génération dans le même dossier `lib`.

Example build.gradle – Dépendances

```
dependencies {
```

```
    ...
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'
    implementation 'org.apache.logging.log4j:log4j-api:2.17.1'
    implementation 'org.apache.logging.log4j:log4j-core:2.17.1'
    runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.17.1'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'
    ...
}
```

Lambda charge JAR les fichiers dans l'ordre alphabétique Unicode. Si plusieurs JAR fichiers du `lib` répertoire contiennent la même classe, la première est utilisée. Vous pouvez utiliser le script shell suivant afin d'identifier les classes en double :

Example test-zip.sh

```
mkdir -p expanded
unzip path/to/my/function.zip -d expanded
find ./expanded/lib -name '*.jar' | xargs -n1 zipinfo -1 | grep '.*.class' | sort |
 uniq -c | sort
```

## Création d'une couche Java pour vos dépendances

> **Note**
>
> Il se peut que l'utilisation de couches avec des fonctions dans un langage compilé tel que Java ne procure pas les mêmes avantages qu'avec un langage interprété tel que Python. Java étant un langage compilé, vos fonctions doivent toujours charger manuellement les assemblages partagés dans la mémoire pendant la phase d'initialisation, ce qui peut augmenter les temps de démarrage à froid. Nous recommandons plutôt d'inclure tout le code partagé au moment de la compilation afin de tirer parti des optimisations intégrées du compilateur.

Les instructions de cette section vous indiquent comment inclure les dépendances dans une couche. Pour obtenir des instructions sur la façon d'inclure vos dépendances dans votre package de déploiement, voir the section called "Création d'un package de déploiement avec Gradle" ou the section called "Création d'un package de déploiement avec Maven".

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d'exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

- `java/lib` (`CLASSPATH`)

Par exemple, la structure du fichier .zip de votre couche peut ressembler à ce qui suit :

```
jackson.zip
# java/lib/jackson-core-2.2.3.jar
```

En outre, Lambda détecte automatiquement toutes les bibliothèques dans le répertoire `/opt/lib` et tous les fichiers binaires dans le répertoire `/opt/bin`. Pour que Lambda trouve correctement le contenu de votre couche, vous pouvez aussi créer une couche avec la structure suivante :

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Après avoir empaqueté votre couche, reportez-vous à the section called “Création et suppression de couches” et à the section called “Ajout de couches” pour terminer la configuration de votre couche.

## Création d'un package de déploiement avec Maven

Pour créer un package de déploiement avec Maven, utilisez le plug-in Maven Shade. Le plugin crée un JAR fichier qui contient le code de fonction compilé et toutes ses dépendances.

Example pom.xml – Configuration du plugin

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
```

```
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
```

Pour créer le package de déploiement, utilisez la commande `mvn package`.

```
[INFO] Scanning for projects...
[INFO] ----------------------< com.example:java-maven >----------------------
[INFO] Building java-maven-function 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ java-maven ---
[INFO] Building jar: target/java-maven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.2.2:shade (default) @ java-maven ---
[INFO] Including com.amazonaws:aws-lambda-java-core:jar:1.2.2 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:3.11.1 in the shaded jar.
[INFO] Including joda-time:joda-time:jar:2.6 in the shaded jar.
[INFO] Including com.google.code.gson:gson:jar:2.8.6 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing target/java-maven-1.0-SNAPSHOT.jar with target/java-maven-1.0-
SNAPSHOT-shaded.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.321 s
[INFO] Finished at: 2020-03-03T09:07:19Z
[INFO] ------------------------------------------------------------------------
```

Cette commande génère un JAR fichier dans le `target` répertoire.

> **Note**
>
> Si vous travaillez avec une version multiple JAR (MRJAR), vous devez inclure le MRJAR (c'est-à-dire le shaded JAR produit par le plugin Maven Shade) dans le `lib` répertoire et

le compresser avant de télécharger votre package de déploiement sur Lambda. Sinon, Lambda risque de ne pas décompresser correctement votre JAR fichier, ce qui entraînera son `MANIFEST.MF` ignorance.

Si vous utilisez la bibliothèque appender (`aws-lambda-java-log4j2`), vous devez également configurer un transformateur pour le plug-in Maven Shade. La bibliothèque de transformateurs combine les versions d'un fichier cache qui apparaissent à la fois dans la bibliothèque appender et dans Log4j.

Example pom.xml – Configuration du plugin avec l'appender Log4j 2

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCacheFile
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
        <dependencies>
          <dependency>
            <groupId>com.github.edwgiz</groupId>
            <artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
            <version>2.13.0</version>
          </dependency>
        </dependencies>
      </plugin>
```

# Chargement d’un package de déploiement avec la console Lambda

Pour créer une nouvelle fonction, vous devez d'abord créer la fonction dans la console, puis télécharger votre fichier .zip ou JAR fichier. Pour mettre à jour une fonction existante, ouvrez la page correspondante, puis suivez la même procédure pour ajouter votre fichier .zip ou JAR fichier mis à jour.

Si votre fichier package de déploiement fait moins de 50 Mo, vous pouvez créer ou mettre à jour une fonction en chargeant le fichier directement à partir de votre ordinateur local. Pour les JAR fichiers .zip ou supérieurs à 50 Mo, vous devez d'abord télécharger votre package dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS Management Console, consultez Getting started with Amazon S3. Pour télécharger des fichiers à l'aide de AWS CLI, voir Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Vous ne pouvez pas modifier le type de package de déploiement (.zip ou image de conteneur) pour une fonction existante. Par exemple, vous ne pouvez pas convertir une fonction d'image de conteneur pour utiliser une archive de fichier .zip. Vous devez créer une nouvelle fonction.

Pour créer une nouvelle fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Choisissez Créer à partir de zéro.
3. Sous Informations de base, procédez comme suit :
   a. Pour Nom de la fonction, saisissez le nom de la fonction.
   b. Pour Exécution, sélectionnez l’exécution que vous souhaitez utiliser.
   c. (Facultatif) Pour Architecture, choisissez l’architecture de l’ensemble des instructions pour votre fonction. L’architecture par défaut est x86_64. Assurez-vous que le package de déploiement .zip pour votre fonction est compatible avec l’architecture de l’ensemble d’instructions que vous sélectionnez.

4. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou en utiliser un existant.
5. Sélectionnez Create function (Créer une fonction). Lambda crée une fonction de base « Hello world » à l'aide de l'exécution de votre choix.

Pour télécharger un fichier .zip ou une JAR archive depuis votre machine locale (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez télécharger le fichier .zip ou JAR le fichier.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez un fichier .zip ou .jar.
5. Pour télécharger le fichier .zip ou JAR le fichier, procédez comme suit :
   a. Sélectionnez Télécharger, puis sélectionnez votre fichier .zip ou votre JAR fichier dans le sélecteur de fichiers.
   b. Choisissez Ouvrir.
   c. Choisissez Save (Enregistrer).

Pour télécharger un fichier .zip ou une JAR archive depuis un compartiment Amazon S3 (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez télécharger un nouveau fichier .zip ou JAR un nouveau fichier.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez l'emplacement Amazon S3.
5. Collez le lien Amazon S3 URL de votre fichier .zip et choisissez Enregistrer.

## Téléchargement d'un package de déploiement à l'aide du AWS CLI

Vous pouvez utiliser le AWS CLIpour créer une nouvelle fonction ou pour mettre à jour une fonction existante à l'aide d'un fichier .zip ou d'un JAR fichier. Utilisez la fonction de création et update-function-codeles commandes pour déployer votre fichier .zip ou votre package. JAR Si votre fichier

est inférieur à 50 Mo, vous pouvez charger le package à partir d'un emplacement de fichier sur votre machine de génération locale. Pour les fichiers plus volumineux, vous devez télécharger votre fichier .zip ou votre JAR package depuis un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Si vous chargez votre JAR fichier .zip ou depuis un compartiment Amazon S3 à l'aide de AWS CLI, le compartiment doit se trouver au même endroit Région AWS que votre fonction.

Pour créer une nouvelle fonction à l'aide d'un JAR fichier .zip ou contenant le AWS CLI, vous devez spécifier les éléments suivants :

- Le nom de votre fonction (`--function-name`)
- L'exécution de votre fonction (`--runtime`)
- Le nom de ressource Amazon (ARN) du rôle d'exécution de votre fonction (`--role`)
- Le nom de la méthode du gestionnaire dans votre code de fonction (`--handler`)

Vous devez également spécifier l'emplacement de votre fichier .zip ou JAR fichier. Si votre fichier .zip ou JAR fichier se trouve dans un dossier de votre machine de génération locale, utilisez l'`--zip-file`option pour spécifier le chemin du fichier, comme illustré dans l'exemple de commande suivant.

```
aws lambda create-function --function-name myFunction \
--runtime java21 --handler example.handler \
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez l'option `--code` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `S3ObjectVersion` pour les objets soumis à la gestion des versions.

```
aws lambda create-function --function-name myFunction \
--runtime java21 --handler example.handler \
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Pour mettre à jour une fonction existante à l'aide duCLI, vous devez spécifier le nom de votre fonction à l'aide du `--function-name` paramètre. Vous devez également spécifier l'emplacement du fichier .zip que vous souhaitez utiliser pour mettre à jour votre code de fonction. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez les options `--s3-bucket` et `--s3-key` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `--s3-object-version` pour les objets soumis à la gestion des versions.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

## Téléchargement d'un package de déploiement avec AWS SAM

Vous pouvez l'utiliser AWS SAM pour automatiser les déploiements de votre code de fonction, de votre configuration et de vos dépendances. AWS SAM est une extension de AWS CloudFormation qui fournit une syntaxe simplifiée pour définir des applications sans serveur. L'exemple de modèle suivant définit une fonction avec un package de déploiement dans le répertoire `build/distributions` utilisé par Gradle :

Example template.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: build/distributions/java-basic.zip
      Handler: example.Handler
      Runtime: java21
      Description: Java function
      MemorySize: 512
```

```
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
        - AWSLambdaVPCAccessExecutionRole
      Tracing: Active
```

Pour créer la fonction, utilisez les commandes `package` et `deploy`. Ces commandes sont des personnalisations de l AWS CLI. Ils encapsulent d'autres commandes pour télécharger le package de déploiement sur Amazon S3, réécrivent le modèle avec l'objet URI et mettent à jour le code de la fonction.

L'exemple de script suivant exécute une build Gradle et télécharge le package de déploiement qu'il crée. Il crée une AWS CloudFormation pile la première fois que vous l'exécutez. Si la pile existe déjà, le script la met à jour.

Example deploy.sh

```
#!/bin/bash
set -eo pipefail
aws cloudformation package --template-file template.yml --s3-bucket MY_BUCKET --output-
template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name java-basic --
capabilities CAPABILITY_NAMED_IAM
```

Afin d'obtenir un exemple de travail complet, consultez les exemples d'applications suivants :

Exemples d'applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d'événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.

- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

# Déployer des fonctions Lambda en Java avec des images conteneurs

Il existe trois méthodes pour créer une image de conteneur pour une fonction Lambda Java :

- Utilisation d'une image AWS de base pour Java

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Java dans l'image.

- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour Java dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

Rubriques


- AWS images de base pour Java
- Utilisation d'une image AWS de base pour Java
- Utilisation d’une autre image de base avec le client d’interface d’exécution


## AWS images de base pour Java

AWS fournit les images de base suivantes pour Java :

| Balises | Environnement d’exécution | Système d’exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| 21 | Java 21 | Amazon Linux | Dockerfile pour Java 21 sur GitHub | Non prévu |
| 17 | Java 17 | Amazon Linux 2 | Dockerfile pour Java 17 sur GitHub | Non prévu |
| 11 | Java 11 | Amazon Linux 2 | Dockerfile pour Java 11 sur GitHub | Non prévu |
| 8.al2 | Java 8 | Amazon Linux 2 | Dockerfile pour Java 8 sur GitHub | Non prévu |

ECRRéférentiel Amazon : gallery.ecr.aws/lambda/java

Les images de base de Java 21 et versions ultérieures sont basées sur l'image de conteneur minimale Amazon Linux 2023. Les images de base antérieures utilisaient Amazon Linux 2. AL2Le 023 offre plusieurs avantages par rapport à Amazon Linux 2, notamment un encombrement de déploiement réduit et des versions mises à jour de bibliothèques telles que`glibc`.

AL2Les images basées sur le format 023 sont utilisées `microdnf` (en lien symbolique comme`dnf`) comme gestionnaire de packages au lieu de`yum`, qui est le gestionnaire de packages par défaut dans Amazon Linux 2. `microdnf`est une implémentation autonome de. `dnf` Pour obtenir la liste des packages inclus dans les images AL2 basées sur la version 023, reportez-vous aux colonnes

Conteneur minimal de la section Comparaison des packages installés sur les images de conteneurs Amazon Linux 2023. Pour plus d'informations sur les différences entre AL2 023 et Amazon Linux 2, consultez Présentation du runtime Amazon Linux 2023 AWS Lambda sur le blog AWS Compute.

> **Note**
>
> Pour exécuter des images AL2 basées sur 023 localement, y compris avec AWS Serverless Application Model (AWS SAM), vous devez utiliser Docker version 20.10.10 ou ultérieure.

## Utilisation d'une image AWS de base pour Java

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Java (par exemple, Amazon Corretto)
- Docker (version minimale 20.10.10 pour les images de base de Java 21 et versions ultérieures)
- Apache Maven ou Gradle
- AWS Command Line Interface (AWS CLI) version 2

Création d'une image à partir d'une image de base

Maven

1. Exécutez la commande suivante pour créer un projet Maven en utilisant l'archétype de Lambda. Les paramètres suivants sont obligatoires :

   - service — Le Service AWS client à utiliser dans la fonction Lambda. Pour une liste des sources disponibles, voir aws-sdk-java-v2/services on GitHub.
   - region — L' Région AWS endroit où vous souhaitez créer la fonction Lambda.
   - groupId— L'espace de noms complet du package de votre application.
   - artifactId— Le nom de votre projet. Cette valeur devient le nom du répertoire de votre projet.

   Sous Linux et macOS, exécutez cette commande :

```
mvn -B archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
   -DgroupId=com.example.myapp \
   -DartifactId=myapp
```

Dans PowerShell, exécutez cette commande :

```
mvn -B archetype:generate `
   "-DarchetypeGroupId=software.amazon.awssdk" `
   "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2"
 `
   "-DgroupId=com.example.myapp" `
   "-DartifactId=myapp"
```

L'archétype Maven pour Lambda est préconfiguré pour compiler avec Java SE 8 et inclut une dépendance au kit AWS SDK for Java. Si vous créez votre projet avec un archétype différent ou en utilisant une autre méthode, vous devez configurer le compilateur Java pour Maven et le déclarer SDK comme dépendance.

2. Ouvrez le répertoire *myapp*`/src/main/java/com/example/`*myapp* et trouvez le fichier `App.java`. Il s'agit du code de la fonction Lambda. Vous pouvez utiliser l'exemple de code fourni pour les tests ou le remplacer par le vôtre.
3. Retournez au répertoire racine du projet, puis créez un nouveau Dockerfile avec la configuration suivante :

   - Définissez la `FROM` propriété sur URIl'image de base.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21
```

```
# Copy function code and runtime dependencies from Maven layout
COPY target/classes ${LAMBDA_TASK_ROOT}
COPY target/dependency/* ${LAMBDA_TASK_ROOT}/lib/

# Set the CMD to your handler (could also be done as a parameter override
 outside of the Dockerfile)
CMD [ "com.example.myapp.App::handleRequest" ]
```

4. Compilez le projet et rassemblez les dépendances de l'environnement d'exécution.

```
mvn compile dependency:copy-dependencies -DincludeScope=runtime
```

5. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

Gradle

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir example
cd example
```

2. Exécutez la commande suivante pour que Gradle génère un nouveau projet d'application Java dans le répertoire `example` de votre environnement. Pour Select build script DSL, choisissez 2 : Groovy.

```
gradle init --type java-application
```

3. Ouvrez le répertoire */example*/app/src/main/java/*example* et trouvez le fichier App.java. Il s'agit du code de la fonction Lambda. Vous pouvez utiliser l'exemple de code suivant pour le tester, ou le remplacer par le vôtre.

   Example App.java

   ```
   package com.example;
   import com.amazonaws.services.lambda.runtime.Context;
   import com.amazonaws.services.lambda.runtime.RequestHandler;
   public class App implements RequestHandler<Object, String> {
       public String handleRequest(Object input, Context context) {
           return "Hello world!";
       }
   }
   ```

4. Ouvrez le fichier build.gradle. Si vous utilisez l'exemple de code de fonction de l'étape précédente, remplacez le contenu de build.gradle par ce qui suit. Si vous utilisez votre propre code de fonction, modifiez votre fichier build.gradle selon vos besoins.

   Example build.gradle (Groovy) DSL

   ```
   plugins {
     id 'java'
   }
   group 'com.example'
   version '1.0-SNAPSHOT'
   sourceCompatibility = 1.8
   repositories {
     mavenCentral()
   }
   dependencies {
     implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
   }
   jar {
     manifest {
         attributes 'Main-Class': 'com.example.App'
     }
   }
   ```

5. La commande `gradle init` de l'étape 2 a également généré un cas de test fictif dans le répertoire `app/test`. Pour les besoins de ce tutoriel, passez outre l'exécution des tests en supprimant le répertoire `/test`.
6. Générez le projet.

```
gradle build
```

7. Dans le répertoire racine du projet (`/example`), créez un fichier Docker avec la configuration suivante :

   - Définissez la `FROM` propriété sur URIl'image de base.
   - Utilisez la COPY commande pour copier le code de fonction et les dépendances d'exécution dans une variable `{LAMBDA_TASK_ROOT}` d'environnement définie par Lambda.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Gradle layout
COPY app/build/classes/java/main ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override
 outside of the Dockerfile)
CMD [ "com.example.App::handleRequest" ]
```

8. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

1. Démarrez votre image Docker à l'aide de la commande docker run. Dans cet exemple, `docker-image` est le nom de l'image et `test` est la balise.

   ```
   docker run --platform linux/amd64 -p 9000:8080 --read-only docker-image:test
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

2. À partir d'une nouvelle fenêtre de terminal, publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

- `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
- Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilisation d’une autre image de base avec le client d’interface d’exécution

Si vous utilisez une image de base uniquement pour le système d’exploitation ou une autre image de base, vous devez inclure le client d'interface d'exécution dans votre image. Le client d’interface d’exécution étend leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l’interaction entre Lambda et votre code de fonction.

Installez le client d'interface d'exécution pour Java dans votre Dockerfile ou en tant que dépendance dans votre projet. Par exemple, pour installer le client d'interface d'exécution à l'aide du gestionnaire de packages Maven, ajoutez ce qui suit à votre fichier `pom.xml` :

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>2.3.2</version>
</dependency>
```

Pour plus de détails sur le package, consultez Client d'interface d'exécution AWS Lambda dans le référentiel central Maven. Vous pouvez également consulter le code source du client de l'interface d'exécution dans le GitHub référentiel AWS Lambda Java Support Libraries.

L'exemple suivant montre comment créer une image de conteneur pour Java à l'aide d'une image Amazon Corretto. Amazon Corretto est une distribution gratuite, multiplateforme et prête à la production du kit de développement Open Java (Open). JDK Le projet Maven inclut le client d'interface d'exécution en tant que dépendance.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Java (par exemple, Amazon Corretto)
- Docker
- Apache Maven
- AWS Command Line Interface (AWS CLI) version 2

Création d'une image à partir d'une image de base alternative

1. Créez un projet Maven. Les paramètres suivants sont obligatoires :
    - groupId— L'espace de noms complet du package de votre application.
    - artifactId— Le nom de votre projet. Cette valeur devient le nom du répertoire de votre projet.

Linux/macOS

```
mvn -B archetype:generate \
   -DarchetypeArtifactId=maven-archetype-quickstart \
   -DgroupId=example \
   -DartifactId=myapp \
   -DinteractiveMode=false
```

PowerShell

```
mvn -B archetype:generate `
   -DarchetypeArtifactId=maven-archetype-quickstart `
   -DgroupId=example `
   -DartifactId=myapp `
   -DinteractiveMode=false
```

2. Ouvrez le répertoire du projet.

```
cd myapp
```

3. Ouvrez le fichier `pom.xml` et remplacez le contenu par ce qui suit. Ce fichier inclut le aws-lambda-java-runtime-interface-client en tant que dépendance. Vous pouvez également installer le client de l'interface d'exécution dans le Dockerfile. Toutefois, l'approche la plus simple consiste à inclure la bibliothèque en tant que dépendance.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>example</groupId>
  <artifactId>hello-lambda</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello-lambda</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
```

```
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
      <version>2.3.2</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>3.1.2</version>
        <executions>
          <execution>
            <id>copy-dependencies</id>
            <phase>package</phase>
            <goals>
              <goal>copy-dependencies</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

4. Ouvrez le répertoire *myapp*`/src/main/java/com/example/`*myapp* et trouvez le fichier `App.java`. Il s'agit du code de la fonction Lambda. Remplacez le code par ce qui suit.

   Example gestionnaire de fonctions

   ```
   package example;

   public class App {
       public static String sayHello() {
           return "Hello world!";
       }
   }
   ```

5. La commande `mvn -B archetype:generate` de l'étape 1 a également généré un cas de test fictif dans le répertoire `src/test`. Pour les besoins de ce tutoriel, passez outre l'exécution des tests en supprimant l'intégralité de ce répertoire `/test` généré.

6. Retournez au répertoire racine du projet, puis créez un nouveau Dockerfile. L'exemple de Dockerfile suivant utilise une image Amazon Corretto. Amazon Corretto est une distribution gratuite, multiplateforme et prête à la production de l'Open. JDK

   - Définissez la `FROM` propriété sur URI l'image de base.
   - Définissez le `ENTRYPOINT` sur le module que vous souhaitez que le conteneur Docker exécute lorsqu'il démarre. Dans ce cas, le module est le client d'interface d'exécution.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

   Example Dockerfile

```
FROM public.ecr.aws/amazoncorretto/amazoncorretto:21 as base

# Configure the build environment
FROM base as build
RUN yum install -y maven
WORKDIR /src

# Cache and copy dependencies
ADD pom.xml .
RUN mvn dependency:go-offline dependency:copy-dependencies

# Compile the function
ADD . .
RUN mvn package

# Copy the function artifact and dependencies onto a clean base
FROM base
WORKDIR /function

COPY --from=build /src/target/dependency/*.jar ./
COPY --from=build /src/target/*.jar ./

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/bin/java", "-cp", "./*",
 "com.amazonaws.services.lambda.runtime.api.client.AWSLambda" ]
```

```
# Pass the name of the function handler as an argument to the runtime
CMD [ "example.App::sayHello" ]
```

7. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester l'image localement. Vous pouvez intégrer l'émulateur à votre image ou utiliser la procédure suivante pour l'installer sur votre machine locale.

Pour installer et exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Depuis le répertoire de votre projet, exécutez la commande suivante pour télécharger l'émulateur d'interface d'exécution (architecture x86-64) GitHub et l'installer sur votre machine locale.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Pour installer l'émulateur arm64, remplacez le GitHub référentiel de URL la commande précédente par le suivant :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Pour installer l'émulateur arm64, remplacez `$downloadLink` par ce qui suit :

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Démarrez votre image Docker à l'aide de la commande docker run. Notez ce qui suit :

   - `docker-image` est le nom de l'image et `test` est la balise.
   - `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello` est le `ENTRYPOINT` suivi du `CMD` depuis votre Dockerfile.

   Linux/macOS

   ```
   docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
    9000:8080 \
       --entrypoint /aws-lambda/aws-lambda-rie \
       --read-only \
       docker-image:test \
           /usr/bin/java -cp './*'
    com.amazonaws.services.lambda.runtime.api.client.AWSLambda
    example.App::sayHello
   ```

   PowerShell

   ```
   docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
    9000:8080 `
   ```

```
--entrypoint /aws-lambda/aws-lambda-rie `
--read-only `
docker-image:test `
    /usr/bin/java -cp './*'
 com.amazonaws.services.lambda.runtime.api.client.AWSLambda
 example.App::sayHello
```

Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

3. Publiez un événement au point de terminaison local.

   Linux/macOS

   Sous Linux et macOS, exécutez la commande `curl` suivante :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenez l'ID du conteneur.

```
docker ps
```

5. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemple :

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image qui a été utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image ECR reste la même dans Amazon. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

# Utilisation de couches pour les fonctions Java Lambda

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration. La création d'une couche implique trois étapes générales :

1. Package le contenu de votre couche. Cela implique de créer une archive de fichiers .zip contenant les dépendances que vous souhaitez utiliser dans vos fonctions.
2. Créez la couche dans Lambda.
3. Ajoutez la couche à vos fonctions.

Cette rubrique contient des étapes et des conseils sur la manière d'empaqueter et de créer correctement une couche Java Lambda avec des dépendances de bibliothèque externes.

Rubriques


- Prérequis
- Compatibilité de la couche Java avec Amazon Linux
- Chemins de couche pour les environnements d'exécution Java
- Empaqueter le contenu de la couche
- Création de la couche
- Ajouter la couche à votre fonction


## Prérequis

Pour suivre les étapes décrites dans cette section, vous devez disposer des éléments suivants :

- Java 21
- Apache Maven 3.8.6 ou version ultérieure
- AWS Command Line Interface (AWS CLI) version 2

> **Note**
>
> Assurez-vous que la version Java à laquelle Maven fait référence est identique à la version Java de la fonction que vous souhaitez déployer. Par exemple, pour une fonction Java 21, la `mvn -v` commande doit indiquer la version 21 de Java dans la sortie :
>
> ```
> Apache Maven 3.8.6
> ...
> Java version: 21.0.2, vendor: Oracle Corporation, runtime: /Library/Java/
> JavaVirtualMachines/jdk-21.jdk/Contents/Home
> ...
> ```

Tout au long de cette rubrique, nous faisons référence à l'`layer-java`exemple d'application sur le référentiel awsdocs GitHub . Cette application contient des scripts qui téléchargent les dépendances et génèrent la couche. L'application contient également une fonction correspondante qui utilise les dépendances de la couche. Après avoir créé une couche, vous pouvez déployer et invoquer la fonction correspondante pour vérifier que tout fonctionne correctement. Comme vous utilisez le runtime Java 21 pour les fonctions, les couches doivent également être compatibles avec Java 21.

L'`layer-java`exemple d'application contient un seul exemple dans deux sous-répertoires. Le `layer` répertoire contient un `pom.xml` fichier qui définit les dépendances des couches, ainsi que des scripts permettant de générer la couche. Le `function` répertoire contient un exemple de fonction permettant de vérifier le fonctionnement de la couche. Ce didacticiel explique comment créer et empaqueter cette couche.

## Compatibilité de la couche Java avec Amazon Linux

La première étape de la création d'une couche consiste à regrouper l'ensemble du contenu de la couche dans une archive de fichiers .zip. Parce que les fonctions Lambda s'exécutent sur Amazon Linux, le contenu de votre couche doit pouvoir être compilé et construit dans un environnement Linux.

Le code Java est conçu pour être indépendant de la plate-forme. Vous pouvez donc empaqueter vos couches sur votre machine locale même si celle-ci n'utilise pas d'environnement Linux. Une fois que vous aurez chargé la couche Java sur Lambda, elle sera toujours compatible avec Amazon Linux.

## Chemins de couche pour les environnements d'exécution Java

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d'exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

- `java/lib`

Par exemple, le fichier de couche .zip obtenu que vous créez dans ce didacticiel possède la structure de répertoire suivante :

```
layer_content.zip
# java
    # lib
        # layer-java-layer-1.0-SNAPSHOT.jar
```

Le `layer-java-layer-1.0-SNAPSHOT.jar` JAR fichier (un fichier uber-jar contenant toutes les dépendances requises) est correctement situé dans le répertoire. `java/lib` Cela garantit que Lambda peut localiser la bibliothèque lors des appels de fonctions.

## Empaqueter le contenu de la couche

Dans cet exemple, vous regroupez les deux bibliothèques Java suivantes dans un seul JAR fichier :

- aws-lambda-java-core— Un ensemble minimal de définitions d'interface pour travailler avec Java dans AWS Lambda
- Jackson — Une suite populaire d'outils de traitement de données, en particulier pour travailler avecJSON.

Procédez comme suit pour installer et empaqueter le contenu de la couche.

Pour installer et empaqueter le contenu de votre couche

1. Clonez le `aws-lambda-developer-guide` GitHub dépôt, qui contient l'exemple de code dont vous avez besoin dans le `sample-apps/layer-java` répertoire.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Accédez au `layer` répertoire de l'`layer-java`exemple d'application. Ce répertoire contient les scripts que vous utilisez pour créer et empaqueter correctement la couche.

```
cd aws-lambda-developer-guide/sample-apps/layer-java/layer
```

3. Examinez le `pom.xml`fichier. Dans `<dependencies>` cette section, vous définissez les dépendances que vous souhaitez inclure dans la couche, à savoir les `jackson-databind` bibliothèques `aws-lambda-java-core` et. Vous pouvez mettre à jour ce fichier pour inclure les dépendances que vous souhaitez inclure dans votre propre couche.

   Example pom.xml

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-lambda-java-core</artifactId>
        <version>1.2.3</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.0</version>
    </dependency>
</dependencies>
```

> **Note**
>
> La `<build>` section de ce `pom.xml` fichier contient deux plugins. maven-compiler-pluginCompile ensuite le code source. Ils maven-shade-pluginempaquetent vos artefacts dans un seul uber-jar.

4. Vérifiez que vous êtes autorisé à exécuter les deux scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Exécutez le `1-install.sh`script à l'aide de la commande suivante :

```
./1-install.sh
```

Ce script s'exécute `mvn clean install` dans le répertoire en cours. Cela crée l'uber-jar avec toutes les dépendances requises dans le répertoire. `target/`

Example 1-install.sh

```
mvn clean install
```

6. Exécutez le `2-package.sh`script à l'aide de la commande suivante :

```
./2-package.sh
```

Ce script crée la structure de `java/lib` répertoire dont vous avez besoin pour empaqueter correctement le contenu de votre couche. Il copie ensuite le fichier uber-jar du `/target` répertoire dans le répertoire nouvellement créé. `java/lib` Enfin, le script compresse le contenu du `java` répertoire dans un fichier nommé`layer_content.zip`. Il s'agit du fichier .zip de votre couche. Vous pouvez décompresser le fichier et vérifier qu'il contient la bonne structure de fichier, comme indiqué dans la the section called “Chemins de couche pour les environnements d'exécution Java” section.

Example 2-package.sh

```
mkdir java
mkdir java/lib
cp -r target/layer-java-layer-1.0-SNAPSHOT.jar java/lib/
zip -r layer_content.zip java
```

## Création de la couche

Dans cette section, vous allez prendre le `layer_content.zip` fichier que vous avez généré dans la section précédente et le télécharger en tant que couche Lambda. Vous pouvez télécharger une couche à l'aide du AWS Management Console ou du Lambda API via le AWS Command Line Interface ()AWS CLI. Lorsque vous téléchargez le fichier .zip de votre couche, dans la PublishLayerVersion AWS CLI commande suivante, spécifiez `java21` comme environnement d'exécution compatible et `arm64` comme architecture compatible.

```
aws lambda publish-layer-version --layer-name java-jackson-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes java21 \
    --compatible-architectures "arm64"
```

Dans la réponse, notez le`LayerVersionArn`, qui ressemble à`arn:aws:lambda:us-east-1:123456789012:layer:java-jackson-layer:1`. Vous aurez besoin de cet Amazon Resource Name (ARN) à l'étape suivante de ce didacticiel, lorsque vous ajouterez la couche à votre fonction.

## Ajouter la couche à votre fonction

Dans cette section, vous déployez un exemple de fonction Lambda qui utilise la bibliothèque Jackson dans son code de fonction, puis vous attachez la couche. Pour déployer la fonction, vous avez besoin d'unthe section called “Rôle d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources)”. Si vous n'avez pas de rôle d'exécution existant, suivez les étapes décrites dans la section pliable.

(Facultatif) Créez un rôle d'exécution

Pour créer un rôle d’exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :

   - Entité de confiance – Lambda.
   - Autorisations — AWSLambdaBasicExecutionRole.
   - Nom de rôle – **`lambda-role`**.

   La AWSLambdaBasicExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour écrire des CloudWatch journaux dans Logs.

Le code de la fonction Lambda prend une entrée `Map<String, String>` comme entrée et utilise Jackson pour écrire l'entrée sous forme de JSON chaîne avant de la convertir en un objet Java F1Car prédéfini. Enfin, la fonction utilise les champs de l'objet F1Car pour créer une chaîne qu'elle renvoie.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.util.Map;

public class Handler {

    public String handleRequest(Map<String, String> input, Context context) throws
 IOException {
        // Parse the input JSON
        ObjectMapper objectMapper = new ObjectMapper();
        F1Car f1Car = objectMapper.readValue(objectMapper.writeValueAsString(input),
 F1Car.class);

        StringBuilder finalString = new StringBuilder();
        finalString.append(f1Car.getDriver());
        finalString.append(" is a driver for team ");
        finalString.append(f1Car.getTeam());
        return finalString.toString();
    }
}
```

Pour déployer la fonction Lambda

1. Accédez au répertoire `function/`. Si vous vous trouvez actuellement dans le `layer/` répertoire, exécutez la commande suivante :

   ```
   cd ../function
   ```

2. Créez le projet à l'aide de la commande Maven suivante :

   ```
   mvn package
   ```

   Cette commande produit un JAR fichier dans le `target/` répertoire nommé`layer-java-function-1.0-SNAPSHOT.jar`.

3. Déployez la fonction. Dans la AWS CLI commande suivante, remplacez le `--role` paramètre par votre rôle d'exécution ARN :

```
aws lambda create-function --function-name java_function_with_layer \
    --runtime java21 \
    --architectures "arm64" \
    --handler example.Handler::handleRequest \
    --timeout 30 \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://target/layer-java-function-1.0-SNAPSHOT.jar
```

4. Ensuite, attachez la couche à votre fonction. Dans la AWS CLI commande suivante, remplacez le `--layers` paramètre par la version de couche ARN que vous avez indiquée précédemment :

```
aws lambda update-function-configuration --function-name java_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:java-jackson-layer:1"
```

5. Enfin, essayez d'appeler votre fonction à l'aide de la AWS CLI commande suivante :

```
aws lambda invoke --function-name java_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "driver": "Max Verstappen", "team": "Red Bull" }' response.json
```

Vous devriez obtenir un résultat du type suivant :

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Cela indique que la fonction a pu utiliser la dépendance de Jackson pour exécuter correctement la fonction. Vous pouvez vérifier que le `response.json` fichier de sortie contient la bonne chaîne renvoyée :

```
"Max Verstappen is a driver for team Red Bull"
```

### (Facultatif) Nettoyez vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la couche Lambda

1. Ouvrez la page Couches de la console Lambda.
2. Sélectionnez la couche que vous avez créée.
3. Choisissez Supprimer, puis sélectionnez à nouveau Supprimer.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

# Améliorer les performances de démarrage avec Lambda SnapStart

Lambda SnapStart pour Java peut améliorer les performances de démarrage des applications sensibles à la latence jusqu'à 10 fois sans frais supplémentaires, généralement sans modifier votre code de fonction. Le principal facteur de latence au démarrage (souvent appelé temps de démarrage à froid) est le temps que Lambda passe à initialiser la fonction, ce qui inclut le chargement du code de la fonction, le démarrage de l'environnement d'exécution et l'initialisation du code de la fonction.

Avec SnapStart, Lambda initialise votre fonction lorsque vous publiez une version de fonction. Lambda prend un instantané Firecracker microVM de l'état de la mémoire et du disque de l'environnement d'exécution initialisé, chiffre l'instantané et le met en cache pour un accès à faible latence. Lorsque vous invoquez la version de la fonction pour la première fois, et au fur et à mesure que les invocations augmentent, Lambda reprend les nouveaux environnements d'exécution à partir de l'instantané mis en cache au lieu de les initialiser à partir de zéro, ce qui améliore la latence au démarrage.

> **Important**
>
> Si vos applications dépendent de l'unicité de l'état, vous devez évaluer votre code de fonction et vérifier qu'il est résilient aux opérations instantanées. Pour de plus amples informations, veuillez consulter Gérer l'unicité avec Lambda SnapStart.

Rubriques


- Fonctions prises en charge et limitations
- Régions prises en charge
- Considérations de compatibilité
- SnapStart tarification
- Comparaison entre Lambda SnapStart et la simultanéité provisionnée
- Ressources supplémentaires
- Activation et gestion de Lambda SnapStart
- Gérer l'unicité avec Lambda SnapStart
- Implémenter le code avant ou après les captures d'écran des fonctions Lambda
- Surveillance pour Lambda SnapStart
- Modèle de sécurité pour Lambda SnapStart

- Maximisez les performances Lambda SnapStart
- Résolution SnapStart des erreurs liées aux fonctions Lambda Java

## Fonctions prises en charge et limitations

SnapStart prend en charge les environnements d'exécution gérés par Java 11 et versions ultérieures. Les autres environnements d'exécution gérés (tels que `nodejs20.x` et `python3.12`), Exécutions uniquement basées sur le système d'exploitationet les images de conteneurs ne sont pas pris en charge.

SnapStart ne prend pas en charge la simultanéité provisionnée, Amazon Elastic File System EFS (Amazon) ou le stockage éphémère supérieur à 512 Mo.

Pour travailler avec SnapStart, vous pouvez utiliser la console Lambda, le AWS Command Line Interface (AWS CLI), le LambdaAPI, le, AWS SDK for Java AWS CloudFormation, AWS Serverless Application Model (AWS SAM) et. AWS Cloud Development Kit (AWS CDK) Pour de plus amples informations, veuillez consulter Activation et gestion de Lambda SnapStart.

> **Note**
>
> Vous ne pouvez l'utiliser SnapStart que sur des versions de fonctions publiées et des alias pointant vers des versions. Vous ne pouvez pas l'utiliser SnapStart sur la version non publiée d'une fonction ($LATEST).

## Régions prises en charge

SnapStart est disponible dans les versions suivantes Régions AWS :

- USA Est (Virginie du Nord)
- USA Est (Ohio)
- USA Ouest (Californie du Nord)
- USA Ouest (Oregon)
- Afrique (Le Cap)
- Asie-Pacifique (Hong Kong)
- Asie-Pacifique (Mumbai)

- Asie-Pacifique (Hyderabad)
- Asie-Pacifique (Tokyo)
- Asie-Pacifique (Séoul)
- Asie-Pacifique (Osaka)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Sydney)
- Asie-Pacifique (Jakarta)
- Asie-Pacifique (Melbourne)
- Canada (Centre)
- Europe (Stockholm)
- Europe (Francfort)
- Europe (Zurich)
- Europe (Irlande)
- Europe (Londres)
- Europe (Paris)
- Europe (Milan)
- Europe (Espagne)
- Moyen-Orient (UAE)
- Moyen-Orient (Bahreïn)
- Amérique du Sud (Sao Paulo)

## Considérations de compatibilité

Avec SnapStart, Lambda utilise un seul instantané comme état initial pour plusieurs environnements d'exécution. Si votre fonction utilise l'un des éléments suivants pendant la phase d'initialisation, vous devrez peut-être apporter quelques modifications avant de l'utiliser SnapStart :

Unicité

Si votre code d’initialisation génère un contenu unique qui est inclus dans l’instantané, ce contenu peut ne pas être unique lorsqu’il est réutilisé dans plusieurs environnements d’exécution. Pour conserver l'unicité lors de l'utilisation SnapStart, vous devez générer un contenu unique après l'initialisation. Cela inclut des secrets uniques et uniquesIDs, ainsi que l'entropie utilisée pour

générer du pseudo-hasard. Pour savoir comment restaurer l'unicité, consultez Gérer l'unicité avec Lambda SnapStart.

Connexions réseau

L'état des connexions que votre fonction établit pendant la phase d'initialisation n'est pas garanti lorsque Lambda reprend votre fonction à partir d'un instantané. Validez l'état de vos connexions réseau et rétablissez-les si nécessaire. Dans la plupart des cas, les connexions réseau établies par AWS SDK an reprennent automatiquement. Pour les autres connexions, consultez les bonnes pratiques.

Données temporaires

Certaines fonctions téléchargent ou initialisent des données éphémères, telles que des informations d'identification temporaires ou des horodatages mis en cache, pendant la phase d'initialisation. Actualisez les données éphémères dans le gestionnaire de fonctions avant de les utiliser, même lorsque vous ne les utilisez pas. SnapStart

## SnapStart tarification

Il n'y a aucun coût supplémentaire pour SnapStart. Vous êtes facturé en fonction du nombre de demandes pour vos fonctions, de la durée d'exécution de votre code et de la mémoire configurée pour votre fonction. La durée est calculée à partir du moment où votre code commence à s'exécuter jusqu'à son retour ou sa fin, arrondie à la milliseconde la plus proche.

Les frais de durée s'appliquent au code exécuté dans le gestionnaire de fonctions, au code d'initialisation déclaré en dehors du gestionnaire, au temps nécessaire au chargement du runtime (JVM) et à tout code exécuté dans un hook d'exécution. Pour plus d'informations sur la façon dont Lambda calcule la durée, consultez Surveillance pour Lambda SnapStart.

Pour les fonctions configurées avec SnapStart, Lambda recycle régulièrement les environnements d'exécution et réexécute votre code d'initialisation. Pour des raisons de résilience, Lambda crée des instantanés dans plusieurs régions. Des frais s'appliquent chaque fois que Lambda réexécute votre code d'initialisation dans une autre région. Pour plus d'informations sur la façon dont Lambda calcule les frais, consultez Tarification AWS Lambda.

## Comparaison entre Lambda SnapStart et la simultanéité provisionnée

La Lambda SnapStart et la simultanéité provisionnée peuvent réduire les démarrages à froid et les latences exceptionnelles lorsqu'une fonction prend de l'ampleur. SnapStart vous aide à améliorer les

performances de démarrage jusqu'à 10 fois sans frais supplémentaires. La simultanéité provisionnée permet aux fonctions d'être initialisées et prêtes à répondre en un nombre de millisecondes à deux chiffres. La configuration de la simultanéité provisionnée entraîne des frais pour votre. Compte AWS Utilisez la simultanéité provisionnée si votre application a des exigences strictes en matière de latence de démarrage à froid. Vous ne pouvez pas utiliser les deux SnapStart et la simultanéité provisionnée sur la même version de fonction.

> **Note**
>
> SnapStart fonctionne mieux lorsqu'il est utilisé avec des invocations de fonctions à grande échelle. Les fonctions qui sont rarement invoquées peuvent ne pas bénéficier des mêmes améliorations de performance.

## Ressources supplémentaires

En plus de lire les autres rubriques de ce chapitre, nous vous recommandons également d'essayer l' AWS Lambda SnapStartatelier Starting up faster with with et de regarder la session Fast cold starts for your Java functions dans AWS re:Invent 2022.

# Activation et gestion de Lambda SnapStart

Pour l'utiliser SnapStart, activez-la SnapStart sur une fonction Lambda nouvelle ou existante. Ensuite, publiez et invoquez une version de la fonction.

Rubriques


- Activation SnapStart (console)
- Activation SnapStart (AWS CLI)
- Activation SnapStart (API)
- Lambda SnapStart et états des fonctions
- Mise à jour d’un instantané
- En utilisant SnapStart avec AWS SDK for Java
- Utilisation SnapStart avec AWS CloudFormationAWS SAM, et AWS CDK
- Suppression d’instantanés


## Activation SnapStart (console)

SnapStart Pour activer une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d’une fonction.
3. Choisissez Configuration, puis General configuration (Configuration générale).
4. Dans le volet General configuration (Configuration générale), choisissez Edit (Modifier).
5. Sur la page Modifier les paramètres de base, pour SnapStart, choisissez Versions publiées.
6. Choisissez Save (Enregistrer).
7. Publiez une version de la fonction. Lambda initialise votre code, crée un instantané de l’environnement d’exécution initialisé, puis met en cache l’instantané pour un accès à faible latence.
8. Invoquez la version de la fonction..

## Activation SnapStart (AWS CLI)

SnapStart Pour activer une fonction existante

1. Mettez à jour la configuration de la fonction en exécutant la update-function-configurationcommande avec l'--snap-startoption.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --snap-start ApplyOn=PublishedVersions
```

2. Publiez une version de la fonction avec la commande publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Vérifiez qu'elle SnapStart est activée pour la version de la fonction en exécutant la get-function-configurationcommande et en spécifiant le numéro de version. L'exemple suivant spécifie la version 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

   Si la réponse indique que OptimizationStatusc'est le cas On et que l'état l' SnapStart est`Active`, alors il est activé et un instantané est disponible pour la version de fonction spécifiée.

```
"SnapStart": {
    "ApplyOn": "PublishedVersions",
    "OptimizationStatus": "On"
 },
 "State": "Active",
```

4. Invoquez la version de la fonction en exécutant la commande invoke et en spécifiant la version. L'exemple suivant invoque la version 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

À activer SnapStart lorsque vous créez une nouvelle fonction

1. Créez une fonction en exécutant la commande create-function avec l'option --snap-start. Pour--role, spécifiez le nom de ressource Amazon (ARN) de votre rôle d'exécution.

```
aws lambda create-function \
  --function-name my-function \
  --runtime "java21" \
  --zip-file fileb://my-function.zip \
  --handler my-function.handler \
  --role arn:aws:iam::111122223333:role/lambda-ex \
  --snap-start ApplyOn=PublishedVersions
```

2. Créez une version avec la commande publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Vérifiez qu'elle SnapStart est activée pour la version de la fonction en exécutant la get-function-configurationcommande et en spécifiant le numéro de version. L'exemple suivant spécifie la version 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

Si la réponse indique que OptimizationStatusc'est le cas On et que l'état l' SnapStart est`Active`, alors il est activé et un instantané est disponible pour la version de fonction spécifiée.

```
"SnapStart": {
     "ApplyOn": "PublishedVersions",
     "OptimizationStatus": "On"
  },
```

```
  "State": "Active",
```

4. Invoquez la version de la fonction en exécutant la commande invoke et en spécifiant la version. L'exemple suivant invoque la version 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

## Activation SnapStart (API)

Pour activer SnapStart

1. Effectuez l'une des actions suivantes :
   - Créez une nouvelle fonction SnapStart activée en utilisant l'CreateFunctionAPIaction associée au SnapStartparamètre.
   - SnapStart Activez une fonction existante en utilisant l'UpdateFunctionConfigurationaction associée au SnapStartparamètre.
2. Publiez une version de fonction avec l'PublishVersionaction. Lambda initialise votre code, crée un instantané de l'environnement d'exécution initialisé, puis met en cache l'instantané pour un accès à faible latence.
3. Vérifiez qu'elle SnapStart est activée pour la version de la fonction en utilisant l'GetFunctionConfigurationaction. Spécifiez un numéro de version pour confirmer qu' SnapStart il est activé pour cette version. Si la réponse indique que OptimizationStatusc'est le cas 0n et que l'état l' SnapStart est`Active`, alors il est activé et un instantané est disponible pour la version de fonction spécifiée.

```
"SnapStart": {
        "ApplyOn": "PublishedVersions",
```

```
        "OptimizationStatus": "On"
    },
    "State": "Active",
```

4. Invoquez la version de la fonction avec l'action Invoke.

## Lambda SnapStart et états des fonctions

Les états de fonction suivants peuvent se produire lorsque vous utilisez SnapStart. Ils peuvent également se produire lorsque Lambda recycle périodiquement l'environnement d'exécution et réexécute le code d'initialisation d'une fonction configurée avec. SnapStart

- `Pending` – Lambda initialise votre code et prend un instantané de l'environnement d'exécution initialisé. Toute invocation ou autre API action agissant sur la version de la fonction échouera.
- `Active` – La création de l'instantané est terminée et vous pouvez invoquer la fonction. Pour l'utiliser SnapStart, vous devez invoquer la version publiée de la fonction, et non la version non publiée ($LATEST).
- `Inactive` – La version de la fonction n'a pas été invoquée depuis 14 jours. Lorsque la version de la fonction devient `Inactive`, Lambda supprime l'instantané. Si vous invoquez la version de la fonction après 14 jours, Lambda renvoie une réponse `SnapStartNotReadyException` et commence à initialiser un nouvel instantané. Attendez que la fonction version atteigne l'état `Active`, puis invoquez-la à nouveau. L'`Inactive`état peut également se produire lorsque Lambda effectue un recyclage périodique de l'environnement d'exécution. Dans ce cas, si votre fonction ne parvient pas à s'initialiser, elle peut entrer dans un `Inactive` état.
- `Failed` – Lambda a rencontré une erreur lors de l'exécution du code d'initialisation ou de la création de l'instantané.

## Mise à jour d'un instantané

Lambda crée un instantané pour chaque version de fonction publiée. Pour mettre à jour un instantané, publiez une nouvelle version de la fonction. Lambda met automatiquement à jour vos instantanés avec les derniers correctifs d'environnement d'exécution et de sécurité.

## En utilisant SnapStart avec AWS SDK for Java

Pour effectuer des AWS SDK appels depuis votre fonction, Lambda génère un ensemble éphémère d'informations d'identification en assumant le rôle d'exécution de votre fonction. Ces

informations d'identification sont disponibles en tant que variables d'environnement lors de l'invocation de votre fonction. Il n'est pas nécessaire de fournir des informations d'identification pour le code SDK directement saisi. Par défaut, la chaîne de fournisseurs d'informations d'identification vérifie séquentiellement chaque endroit où vous pouvez définir des informations d'identification et sélectionne le premier disponible, généralement les variables d'environnement (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` et `AWS_SESSION_TOKEN`).

> **Note**
>
> Lorsqu'il SnapStart est activé, le moteur d'exécution Java utilise automatiquement les informations d'identification du conteneur (`AWS_CONTAINER_CREDENTIALS_FULL_URI`et`AWS_CONTAINER_AUTHORIZATION_TOKEN`) au lieu des variables d'environnement clés d'accès. Cela évite que les informations d'identification n'expirent avant la restauration de la fonction.

## Utilisation SnapStart avec AWS CloudFormationAWS SAM, et AWS CDK

- AWS CloudFormation: Déclarez l'SnapStartentité dans votre modèle.
- AWS Serverless Application Model (AWS SAM) : Déclarez la SnapStartpropriété dans votre modèle.
- AWS Cloud Development Kit (AWS CDK): Utilisez le SnapStartPropertytype.

## Suppression d'instantanés

Lambda supprime les instantanés lorsque :

- Vous supprimez la fonction ou la version de la fonction.
- Vous n'invoquez pas la version de la fonction pendant 14 jours. Après 14 jours sans invocation, la version de fonction passe à l'état Inactif. Si vous invoquez la version de la fonction après 14 jours, Lambda renvoie une réponse `SnapStartNotReadyException` et commence à initialiser un nouvel instantané. Attendez que la version de la fonction atteigne l'état Actif, puis invoquez-la à nouveau.

Lambda supprime toutes les ressources associées aux instantanés supprimés conformément au règlement général sur la protection des données (). GDPR

# Gérer l'unicité avec Lambda SnapStart

Lorsque les invocations augmentent sur une SnapStart fonction, Lambda utilise un seul instantané initialisé pour reprendre plusieurs environnements d'exécution. Si votre code d'initialisation génère un contenu unique qui est inclus dans l'instantané, ce contenu peut ne pas être unique lorsqu'il est réutilisé dans plusieurs environnements d'exécution. Pour conserver l'unicité lors de l'utilisation SnapStart, vous devez générer un contenu unique après l'initialisation. Cela inclut des secrets uniques et uniquesIDs, ainsi que l'entropie utilisée pour générer du pseudo-hasard.

Nous vous recommandons les bonnes pratiques suivantes pour vous aider à maintenir l'unicité de votre code. Lambda fournit également un outil d'SnapStart analyse open source pour aider à vérifier le code qui suppose un caractère unique. Si vous générez des données uniques pendant la phase d'initialisation, vous pouvez utiliser un hook d'exécution pour rétablir l'unicité. Avec les hooks d'exécution, vous pouvez exécuter un code spécifique immédiatement avant que Lambda ne prenne un instantané ou immédiatement après que Lambda ait repris une fonction à partir d'un instantané.

## Évitez de sauvegarder un état qui dépend de l'unicité pendant l'initialisation

Pendant la phase d'initialisation de votre fonction, évitez de mettre en cache des données qui sont censées être uniques, comme la génération d'un ID unique pour la journalisation. Nous vous recommandons plutôt de générer des données uniques dans votre gestionnaire de fonction ou d'utiliser un hook d'exécution.

Example – Génération d'un ID unique dans le gestionnaire de fonctions

L'exemple suivant montre comment générer un UUID dans le gestionnaire de fonctions.

```
import java.util.UUID;
  public class Handler implements RequestHandler<String, String> {
    private static UUID uniqueSandboxId = null;
    @Override
    public String handleRequest(String event, Context context) {
      if (uniqueSandboxId == null)
        uniqueSandboxId = UUID.randomUUID();
      System.out.println("Unique Sandbox Id: " + uniqueSandboxId);
      return "Hello, World!";
    }
  }
```

## Utiliser des générateurs de nombres pseudo-aléatoires sécurisés par cryptographie () CSPRNGs

Si votre application repose sur le caractère aléatoire, nous vous recommandons d'utiliser des générateurs de nombres aléatoires sécurisés par cryptographie ()CSPRNGs. Le runtime géré Lambda pour Java inclut deux fonctions intégrées CSPRNGs (Open SSL 1.0.2 et`java.security.SecureRandom`) qui maintiennent automatiquement le caractère aléatoire avec. SnapStart Logiciel qui obtient toujours des nombres aléatoires `/dev/random` ou qui `/dev/urandom` maintient le caractère aléatoire avec SnapStart.

AWS les bibliothèques de cryptographie conservent automatiquement le caractère aléatoire en SnapStart commençant par les versions minimales spécifiées dans le tableau suivant. Si vous utilisez ces bibliothèques avec vos fonctions Lambda, assurez-vous d'utiliser les versions minimales ou ultérieures suivantes :

| d'outils | Version minimale prise en charge (x86) | Version minimale prise en charge (ARM) |
|---|---|---|
| AWS libcrypto (AWS-LC) | 1.16.0 | 1.30.0 |
| AWS libcrypto FIPS | 2.0.13 | 2.0.13 |

Si vous empaquetez les bibliothèques cryptographiques précédentes avec vos fonctions Lambda sous forme de dépendances transitives via les bibliothèques suivantes, assurez-vous d'utiliser les versions minimales suivantes ou des versions ultérieures :

| d'outils | Version minimale prise en charge (x86) | Version minimale prise en charge (ARM) |
|---|---|---|
| AWS SDK for Java 2.x | 2,23,20 | 2,26,12 |
| AWS Common Runtime pour Java | 0,29,8 | 0,29,25 |
| Fournisseur de cryptomon naies Amazon Corretto | 2.4.1 | 2.4.1 |

| d'outils | Version minimale prise en charge (x86) | Version minimale prise en charge (ARM) |
| --- | --- | --- |
| Fournisseur de cryptomonnaies Amazon Corretto FIPS | 2.4.1 | 2.4.1 |

Example — java.security. SecureRandom

L'exemple suivant utilise `java.security.SecureRandom`, qui génère des séquences de nombres uniques même lorsque la fonction est restaurée à partir d'un instantané.

```
import java.security.SecureRandom;
  public class Handler implements RequestHandler<String, String> {
    private static SecureRandom rng = new SecureRandom();
    @Override
    public String handleRequest(String event, Context context) {
      for (int i = 0; i < 10; i++) {
        System.out.println(rng.next());
      }
      return "Hello, World!";
    }
  }
```

## SnapStart outil de numérisation

Lambda fournit un outil d'analyse pour vous aider à vérifier le code qui suppose l'unicité. L'outil de SnapStart numérisation est un SpotBugsplugin open source qui exécute une analyse statique par rapport à un ensemble de règles. L'outil d'analyse aide à identifier les implémentations de code potentielles qui pourraient enfreindre les suppositions concernant l'unicité. Pour les instructions d'installation et la liste des vérifications effectuées par l'outil d'analyse, consultez le référentiel aws-lambda-snapstart-java-rules sur GitHub.

Pour en savoir plus sur la gestion de l'unicité avec SnapStart, consultez Starting up faster with AWS Lambda SnapStart sur le blog AWS Compute.

# Implémenter le code avant ou après les captures d'écran des fonctions Lambda

Vous pouvez utiliser des hooks d'exécution pour implémenter du code avant que Lambda ne crée un instantané ou après que Lambda ait repris une fonction à partir d'un instantané. Les hooks d'exécution sont disponibles dans le cadre du projet open source Coordinated Restore at Checkpoint (CRaC). CRaC est en cours de développement pour l'Open Java Development Kit (OpenJDK). Pour un exemple d'utilisation de CRac avec une application de référence, consultez le référentiel CRac sur. GitHub CRaC utilise trois éléments principaux :

- `Resource` – Une interface avec deux méthodes, `beforeCheckpoint()` et `afterRestore()`. Utilisez ces méthodes pour implémenter le code que vous voulez exécuter avant un instantané et après une restauration.
- `Context <R extends Resource>` – Pour recevoir des notifications pour les points de contrôle et les restaurations, une `Resource` doit être enregistrée avec un `Context`.
- `Core` – Le service de coordination, qui fournit le `Context` global par défaut via la méthode statique `Core.getGlobalContext()`.

Pour plus d'informations sur `Context` et `Resource`, consultez Package org.crac dans la documentation de CRaC.

Suivez les étapes suivantes pour implémenter des hooks d'exécution avec le package org.crac. L'environnement d'exécution Lambda contient une implémentation de contexte CRaC personnalisée qui appelle vos hooks d'exécution avant le point de contrôle et après la restauration.

## Étape 1 : mettre à jour de configuration de création

Ajoutez la dépendance `org.crac` à la configuration de compilation. L'exemple suivant utilise Gradle. Pour des exemples concernant d'autres systèmes de compilation, consultez la documentation Apache Maven.

```
dependencies {
    compile group: 'com.amazonaws', name: 'aws-lambda-java-core', version: '1.2.1'
    # All other project dependecies go here:
    # ...
    # Then, add the org.crac dependency:
 implementation group: 'org.crac', name: 'crac', version: '1.4.0'
}
```

## Étape 2 : mettre à jour le gestionnaire Lambda

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Pour plus d'informations, consultez Définir le gestionnaire de fonctions Lambda en Java.

L'exemple de gestionnaire suivant montre comment exécuter le code avant le point de contrôle (`beforeCheckpoint()`) et après la restauration (`afterRestore()`). Ce gestionnaire enregistre également la `Resource` dans le `Context` global géré par l'environnement d'exécution.

> **Note**
>
> Lorsque Lambda crée un instantané, votre code d'initialisation peut s'exécuter jusqu'à 15 minutes. Le délai d'attente est de 130 secondes ou le délai d'expiration de la fonction configurée (900 secondes au maximum), la valeur la plus élevée étant retenue. Vos hooks d'exécution `beforeCheckpoint()` sont comptabilisés dans le délai d'attente du code d'initialisation. Lorsque Lambda restaure un instantané, l'exécution (JVM) doit se charger et les hooks d'exécution `afterRestore()` doivent se terminer dans le délai imparti (10 secondes). Sinon, vous obtiendrez un SnapStartTimeoutException.

```
...
  import org.crac.Resource;
  import org.crac.Core;
  ...
public class CRaCDemo implements RequestStreamHandler, Resource {
    public CRaCDemo() {
      Core.getGlobalContext().register(this);
    }
    public String handleRequest(String name, Context context) throws IOException {
      System.out.println("Handler execution");
      return "Hello " + name;
    }
    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("Before checkpoint");
    }
    @Override
```

```
    public void afterRestore(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("After restore");
```

`Context` maintient uniquement une `WeakReference` à l'objet enregistré. Si une `Resource` est récupérée par le récupérateur de mémoire, les hooks d'exécution ne s'exécutent pas. Votre code doit maintenir une référence forte à la `Resource` pour garantir l'exécution du hook d'exécution.

Voici deux exemples de modèles à éviter :

Example – Objet sans référence forte

```
Core.getGlobalContext().register( new MyResource() );
```

Example – Objets de classes anonymes

```
Core.getGlobalContext().register( new Resource() {

   @Override
   public void afterRestore(Context<? extends Resource> context) throws Exception {
    // ...
   }

   @Override
   public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
    // ...
   }

} );
```

Au lieu de cela, maintenez une référence forte. Dans l'exemple suivant, la ressource enregistrée n'est pas récupérée par le récupérateur de mémoire et les hooks d'exécution s'exécutent de manière cohérente.

Example – Objet avec une référence forte

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent
 the registered resource from being garbage collected
Core.getGlobalContext().register( myResource );
```

# Surveillance pour Lambda SnapStart

Vous pouvez surveiller vos SnapStart fonctions Lambda à l'aide d'Amazon CloudWatch, AWS X-Ray, et du. Accès aux données de télémétrie en temps réel pour les extensions à l'aide de la télémétrie API

> **Note**
>
> Les variables `AWS_LAMBDA_LOG_GROUP_NAME` d'`AWS_LAMBDA_LOG_STREAM_NAME`environnement et ne sont pas disponibles dans les fonctions Lambda SnapStart .

## CloudWatch pour SnapStart

Il existe quelques différences avec le format du flux de CloudWatch log pour les SnapStart fonctions :

- Journaux d'initialisation – Lorsqu'un nouvel environnement d'exécution est créé, le `REPORT` n'inclut pas le champ `Init Duration`. Cela est dû au fait que Lambda initialise les SnapStart fonctions lorsque vous créez une version plutôt que lors de l'appel de fonctions. Pour SnapStart les fonctions, le `Init Duration` champ se trouve dans l'`INIT_REPORT`enregistrement. Cet enregistrement indique les détails de la durée du Phase d'initialisation, y compris la durée de tout hook d'exécution `beforeCheckpoint`.
- Journaux d'invocation – Lorsqu'un nouvel environnement d'exécution est créé, le `REPORT` inclut les champs `Restore Duration` et `Billed Restore Duration` :
  - `Restore Duration` : le temps nécessaire à Lambda pour restaurer un instantané, charger l'exécution (JVM) et exécuter les éventuels hooks d'exécution `afterRestore`. Le processus de restauration des instantanés peut inclure du temps consacré à des activités en dehors de la MicroVM. Ce temps est indiqué dans `Restore Duration`.
  - `Billed Restore Duration` : le temps nécessaire à Lambda pour charger l'exécution (JVM) et exécuter les éventuels hooks d'exécution `afterRestore`. Le temps nécessaire à la restauration d'un instantané ne vous est pas facturé.

> **Note**
>
> Les frais de durée s'appliquent au code qui s'exécute dans le gestionnaire de fonction, au code d'initialisation déclaré en dehors du gestionnaire, au temps nécessaire au chargement

de l'environnement d'exécution (JVM) et à tout code exécuté dans un hook d'exécution. Pour plus d'informations, consultez SnapStart tarification.

La durée du démarrage à froid est la somme de `Restore Duration` + `Duration`.

L'exemple suivant est une requête Lambda Insights qui renvoie les percentiles de latence pour les fonctions. SnapStart Pour plus d'informations sur les demandes Lambda Insights, consultez Exemple de flux de travail utilisant des demandes pour dépanner une fonction.

```
filter @type = "REPORT"
  | parse @log /\d+:\/aws\/lambda\/(?<function>.*)/
  | parse @message /Restore Duration: (?<restoreDuration>.*?) ms/
  | stats
count(*) as invocations,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 50) as p50,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 90) as p90,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99) as p99,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99.9) as p99.9
group by function, (ispresent(@initDuration) or ispresent(restoreDuration)) as
 coldstart
  | sort by coldstart desc
```

## X-Ray Active Tracing pour SnapStart

Vous pouvez utiliser X-Ray pour suivre les requêtes adressées aux fonctions Lambda SnapStart . Il existe quelques différences entre les sous-segments X-Ray relatifs aux SnapStart fonctions :

- Il n'existe aucun `Initialization` sous-segment pour les SnapStart fonctions.
- Le sous-segment `Restore` indique le temps nécessaire à Lambda pour restaurer un instantané, charger l'exécution (JVM) et exécuter les éventuels hooks d'exécution `afterRestore`. Le processus de restauration des instantanés peut inclure du temps consacré à des activités en dehors de la MicroVM. Cette heure est indiquée dans le sous-segment `Restore`. Le temps passé en dehors de la microVM pour restaurer un instantané ne vous est pas facturé.

## Événements de l'API de télémétrie pour SnapStart

Lambda envoie les SnapStart événements suivants au : Télémétrie API

- `platform.restoreStart` – Indique l'heure à laquelle la phase `Restore` a commencé.

- `platform.restoreRuntimeDone` – Indique si la phase `Restore` a réussi. Lambda envoie ce message lorsque l'environnement d'exécution envoie une demande d'API d'exécution `restore/next`. Il existe trois statuts possibles : succès, échec et dépassement de délai.
- `platform.restoreReport` – Indique la durée de la phase `Restore` et le nombre de millisecondes pour lesquelles vous avez été facturé pendant cette phase.

## Métriques de l'URL de la fonction et de l'Amazon API Gateway

Si vous créez une API Web à l'aide d'API Gateway, vous pouvez utiliser la IntegrationLatencymétrique pour mesurer la end-to-end latence (le temps entre le moment où API Gateway transmet une demande au backend et le moment où il reçoit une réponse du backend).

Si vous utilisez une URL de fonction Lambda, vous pouvez utiliser la UrlRequestLatencymétrique pour mesurer la end-to-end latence (le temps entre le moment où l'URL de la fonction reçoit une demande et le moment où l'URL de la fonction renvoie une réponse).

# Modèle de sécurité pour Lambda SnapStart

Lambda SnapStart prend en charge le chiffrement au repos. Lambda chiffre les instantanés avec un AWS KMS key. Par défaut, Lambda utilise une Clé gérée par AWS. Si ce comportement par défaut convient à votre flux, vous n'avez pas besoin de configurer autre chose. Sinon, vous pouvez utiliser l'`--kms-key-arn`option de la fonction de création ou de la update-function-configurationcommande pour fournir une clé gérée par AWS KMS le client. Vous pouvez procéder ainsi pour contrôler la rotation de la clé KMS ou pour répondre aux exigences de votre organisation en matière de gestion des clés KMS. Les clés gérés par le client entraînent des frais AWS KMS standard. Pour en savoir plus, consultez AWS Key Management Service Tarification.

Lorsque vous supprimez une SnapStart fonction ou une version de fonction, toutes les `Invoke` demandes adressées à cette fonction ou version de fonction échouent. Lambda supprime automatiquement les instantanés qui ne sont pas appelés pendant 14 jours. Lambda supprime toutes les ressources associées aux instantanés supprimés, conformément au Règlement général sur la protection des données (RGPD).

# Maximisez les performances Lambda SnapStart

Rubriques


- Personnalisation de performances
- Bonnes pratiques de mise en réseau


## Personnalisation de performances

> **Note**
>
> SnapStart fonctionne mieux lorsqu'il est utilisé avec des invocations de fonctions à grande échelle. Les fonctions qui sont rarement invoquées peuvent ne pas bénéficier des mêmes améliorations de performance.

Pour optimiser les avantages de SnapStart, nous vous recommandons de précharger les classes qui contribuent à la latence de démarrage dans votre code d'initialisation plutôt que dans le gestionnaire de fonctions. Cela permet de déplacer la latence associée au chargement intensif de classes hors du chemin d'invocation, optimisant ainsi les performances de démarrage avec SnapStart.

Si vous ne pouvez pas précharger les classes pendant l'initialisation, nous vous recommandons de le faire avec des appels fictifs. Pour ce faire, mettez à jour le code du gestionnaire de fonctions, comme indiqué dans l'exemple suivant, à partir de la fonction pet store du GitHub référentiel AWS Labs.

```
private static SpringLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;
  static {
      try {
          handler =
 SpringLambdaContainerHandler.getAwsProxyHandler(PetStoreSpringAppConfig.class);

          // Use the onStartup method of the handler to register the custom filter
          handler.onStartup(servletContext -> {
              FilterRegistration.Dynamic registration =
 servletContext.addFilter("CognitoIdentityFilter", CognitoIdentityFilter.class);
              registration.addMappingForUrlPatterns(EnumSet.of(DispatcherType.REQUEST),
 false, "/*");
          });
```

```
            // Send a fake Amazon API Gateway request to the handler to load classes
    ahead of time
            ApiGatewayRequestIdentity identity = new ApiGatewayRequestIdentity();
            identity.setApiKey("foo");
            identity.setAccountId("foo");
            identity.setAccessKey("foo");

            AwsProxyRequestContext reqCtx = new AwsProxyRequestContext();
            reqCtx.setPath("/pets");
            reqCtx.setStage("default");
            reqCtx.setAuthorizer(null);
            reqCtx.setIdentity(identity);

            AwsProxyRequest req = new AwsProxyRequest();
            req.setHttpMethod("GET");
            req.setPath("/pets");
            req.setBody("");
            req.setRequestContext(reqCtx);

            Context ctx = new TestContext();
            handler.proxy(req, ctx);


        } catch (ContainerInitializationException e) {
            // if we fail here. We re-throw the exception to force another cold start
            e.printStackTrace();
            throw new RuntimeException("Could not initialize Spring framework", e);
        }
    }
```

## Bonnes pratiques de mise en réseau

L'état des connexions que votre fonction établit pendant la phase d'initialisation n'est pas garanti lorsque Lambda reprend votre fonction à partir d'un instantané. Dans la plupart des cas, les connexions réseau établies par AWS SDK an reprennent automatiquement. Pour les autres connexions, nous vous recommandons les bonnes pratiques suivantes.

### Rétablissez les connexions réseau

Rétablissez toujours vos connexions réseau lorsque votre fonction reprend à partir d'un instantané. Nous vous recommandons de rétablir les connexions réseau dans le gestionnaire de fonction. Vous pouvez également utiliser un hook d'exécution `afterRestore`.

N'utilisez pas le nom d'hôte comme identifiant unique d'environnement d'exécution

Nous vous déconseillons d'utiliser `hostname` pour identifier votre environnement d'exécution comme un nœud ou un conteneur unique dans vos applications. Avec SnapStart, un seul instantané est utilisé comme état initial pour plusieurs environnements d'exécution, et tous les environnements d'exécution renvoient la même `hostname` valeur pour`InetAddress.getLocalHost()`. Pour les applications qui nécessitent une identité ou une valeur `hostname` d'environnement d'exécution unique, nous vous recommandons de générer un identifiant unique dans le gestionnaire de fonction. Ou bien, utilisez un hook d'exécution `afterRestore` pour générer un ID unique, puis utilisez cet ID unique comme identifiant de l'environnement d'exécution.

Évitez de lier les connexions à des ports source fixes

Nous vous recommandons d'éviter de lier les connexions réseau à des ports source fixes. Les connexions sont rétablies lorsqu'une fonction reprend à partir d'un instantané, et les connexions réseau qui sont liées à un port source fixe peuvent échouer.

Évitez d'utiliser le DNS cache Java

Les fonctions Lambda mettent déjà en cache DNS les réponses. Si vous utilisez un autre DNS cache avec SnapStart, vous risquez de rencontrer des délais de connexion lorsque la fonction reprend à partir d'un instantané.

La `java.util.logging.Logger` classe peut activer indirectement le JVM DNS cache. Pour remplacer les paramètres par défaut, définissez networkaddress.cache.ttl sur 0 avant l'initialisation. `logger` Exemple :

```
public class MyHandler {
  // first set TTL property
  static{
   java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
  }
 // then instantiate logger
  var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

Pour éviter les `UnknownHostException` défaillances de l'environnement d'exécution de Java 11, nous vous recommandons de `networkaddress.cache.negative.ttl` définir la valeur 0. Dans les environnements d'exécution de Java 17 et versions ultérieures, cette étape n'est pas nécessaire. Vous pouvez définir cette propriété pour une fonction Lambda avec la variable d'`AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`environnement.

La désactivation du JVM DNS cache ne désactive pas la mise en cache gérée DNS de Lambda.

# Résolution SnapStart des erreurs liées aux fonctions Lambda Java

Cette page aborde les problèmes courants qui se produisent lors de l'utilisation de Lambda SnapStart, notamment les erreurs de création de snapshots, les erreurs de délai d'expiration et les erreurs de service internes.

## SnapStartNotReadyException

Erreur : une erreur s'est produite (SnapStartNotReadyException) lors de l'appel de l'opération Invoke20150331 : Lambda initialise votre fonction. Il sera prêt à être invoqué une fois que l'état de votre fonction sera atteintACTIVE.

### Causes courantes

Cette erreur se produit lorsque vous essayez d'invoquer une version de fonction qui est dans l'`Inactive`état. La version de votre fonction devient `Inactive` lorsqu'elle n'a pas été invoquée depuis 14 jours ou lorsque Lambda recycle périodiquement l'environnement d'exécution

### Résolution

Attendez que la fonction version atteigne l'état `Active`, puis invoquez-la à nouveau.

## SnapStartTimeoutException

Problème : vous recevez un `SnapStartTimeoutException` lorsque vous essayez d'invoquer une version de SnapStart fonction.

### Cause commune

Pendant la phase de restauration, Lambda restaure le runtime Java et exécute tous les hooks `afterRestore()` d'exécution. Si un `afterRestore()` hook d'exécution s'exécute pendant plus de 10 secondes, la `Restore` phase expire et vous obtenez une erreur lorsque vous essayez d'invoquer la fonction. Les problèmes de connexion réseau et d'identification peuvent également entraîner des délais d'attente de `Restore` phase.

### Résolution

Consultez les CloudWatch journaux de la fonction pour détecter les erreurs de temporisation survenues pendant la phase de restauration. Assurez-vous que tous les `afterRestore()` hameçons sont terminés en moins de 10 secondes.

Example CloudWatch journal

```
{ "cause": "Lambda couldn't restore the snapshot within the timeout limit. (Service:
 Lambda, Status Code: 408, Request ID: 11a222c3-410f-427c-ab22-931d6bcbf4f2)", "error":
 "Lambda.SnapStartTimeoutException"}
```

## 500 Erreur de service interne

Erreur : Lambda n'a pas pu créer de nouvel instantané car vous avez atteint votre limite de création d'instantanés simultanés.

Cause commune

Une erreur 500 est une erreur interne au sein du service Lambda lui-même, plutôt qu'un problème lié à votre fonction ou à votre code. Ces erreurs sont souvent intermittentes.

Résolution

Réessayez de publier la version de la fonction.

## 401 Accès non autorisé

Erreur : jeton de session ou clé d'en-tête incorrect

Cause commune

Cette erreur se produit lors de l'utilisation du AWS Systems Manager Parameter Store et de AWS Secrets Manager l'extension avec Lambda SnapStart.

Résolution

Le AWS Systems Manager Parameter Store et AWS Secrets Manager son extension ne sont pas compatibles avec SnapStart. L'extension génère des informations d'identification avec lesquelles communiquer AWS Secrets Manager lors de l'initialisation de la fonction, ce qui provoque des erreurs d'identification expirées lorsqu'elle est utilisée avec. SnapStart

## UnknownHostException

Erreur : Impossible d'exécuter la HTTP demande : le certificat pour `abc.us-east-1.amazonaws.com` ne correspond à aucun des noms alternatifs du sujet.

Cause commune

Les fonctions Lambda mettent déjà en cache DNS les réponses. Si vous utilisez un autre DNS cache avec SnapStart, vous risquez de rencontrer des délais de connexion lorsque la fonction reprend à partir d'un instantané.

Résolution

Pour éviter les `UnknownHostException` défaillances de l'environnement d'exécution de Java 11, nous vous recommandons de `networkaddress.cache.negative.ttl` définir la valeur 0. Dans les environnements d'exécution de Java 17 et versions ultérieures, cette étape n'est pas nécessaire. Vous pouvez définir cette propriété pour une fonction Lambda avec la variable d'`AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`environnement.

## Échec de création de snapshots

Erreur : AWS Lambda impossible d'appeler votre SnapStart fonction. Si cette erreur persiste, consultez les CloudWatch journaux de votre fonction pour détecter les erreurs d'initialisation.

Résolution

Consultez les CloudWatch journaux Amazon de votre fonction pour connaître les délais d'expiration des délais `beforeCheckpoint()` d'exécution. Vous pouvez également essayer de publier une nouvelle version de fonction, ce qui peut parfois résoudre le problème.

## Latence de création de snapshots

Problème : Lorsque vous publiez une nouvelle version de fonction, celle-ci reste dans son `Pending` état pendant une longue période.

Cause commune

Lorsque Lambda crée un instantané, votre code d'initialisation peut s'exécuter jusqu'à 15 minutes. Le délai d'attente est de 130 secondes ou le délai d'expiration de la fonction configurée (900 secondes au maximum), la valeur la plus élevée étant retenue.

Si votre fonction est attachée à un VPC, Lambda devra peut-être également créer des interfaces réseau avant que la fonction ne le devienne. `Active` Si vous essayez d'invoquer la version de la fonction alors que la fonction est en `Pending` cours, vous pourriez obtenir un `409ResourceConflictException`. Si la fonction est invoquée à l'aide d'un point de terminaison Amazon API Gateway, une erreur 500 peut s'afficher dans API Gateway.

## Résolution

Attendez au moins 15 minutes que la version de la fonction soit initialisée avant de l'invoquer.

# Personnalisation du comportement de démarrage de l'exécution Java pour les fonctions Lambda

Cette page décrit les paramètres spécifiques aux fonctions Java dans AWS Lambda. Vous pouvez utiliser ces paramètres pour personnaliser le comportement de démarrage de l'exécution Java. Cela permet de réduire le temps de latence et d'améliorer les performances globales de la fonction, sans avoir à modifier le code.

Sections


- Comprendre la variable d'JAVA_TOOL_OPTIONSenvironnement


## Comprendre la variable d'**`JAVA_TOOL_OPTIONS`**environnement

Sur Java, Lambda prend en charge la variable d'environnement `JAVA_TOOL_OPTIONS` pour définir des variables de ligne de commande supplémentaires dans Lambda. Vous pouvez utiliser cette variable d'environnement de différentes manières, par exemple pour personnaliser les paramètres de compilation par niveaux. L'exemple suivant montre comment utiliser la variable d'environnement `JAVA_TOOL_OPTIONS` pour ce cas d'utilisation.

### Exemple : personnalisation des paramètres de compilation à plusieurs niveaux

La compilation à plusieurs niveaux est une fonctionnalité de la machine virtuelle Java (JVM). Vous pouvez utiliser des paramètres de compilation hiérarchisés spécifiques pour tirer le meilleur parti des compilateurs JVM's just-in-time (JIT). Généralement, le compilateur C1 est optimisé pour un démarrage rapide. Le compilateur C2 est optimisé pour obtenir les meilleures performances globales, mais il utilise également plus de mémoire et prend plus de temps pour y parvenir.

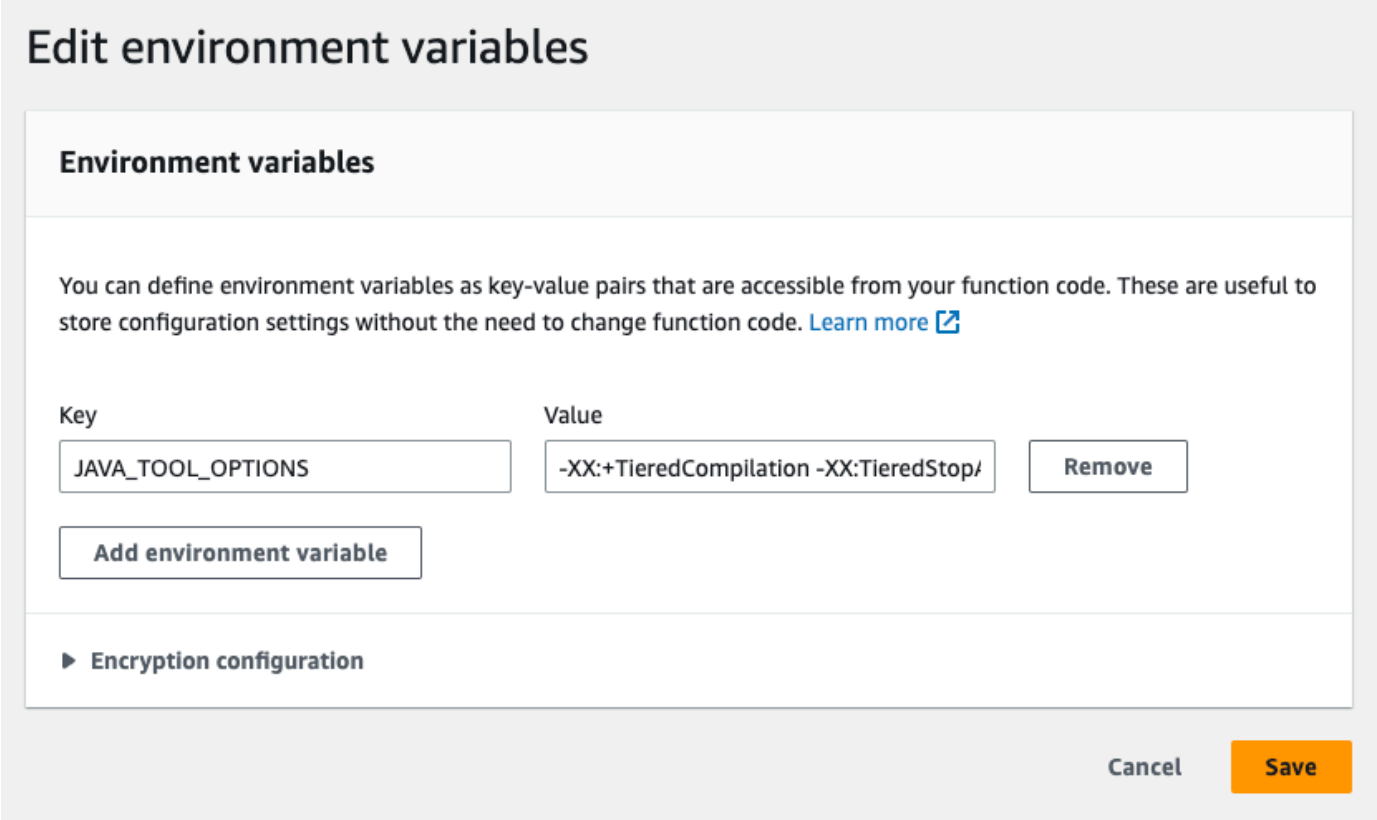
Il existe cinq niveaux différents de compilation par niveaux. Au niveau 0, le code JVM octet Java est interprété. Au niveau 4, il JVM utilise le compilateur C2 pour analyser les données de profilage collectées lors du démarrage de l'application. Au fil du temps, elle surveille l'utilisation du code pour identifier les meilleures optimisations.

La personnalisation du niveau de compilation par niveau peut vous aider à réduire la latence de démarrage à froid de la fonction Java. Par exemple, définissez le niveau de compilation hiérarchisé sur 1 pour JVM utiliser le compilateur C1. Ce compilateur produit rapidement un code natif optimisé, mais il ne génère aucune donnée de profilage et n'utilise jamais le compilateur C2.

Dans l'environnement d'exécution de Java 17, l'JVMindicateur de compilation à plusieurs niveaux est défini pour s'arrêter au niveau 1 par défaut. Pour l'exécution Java 11 et les versions antérieures, vous pouvez définir le niveau de compilation par niveaux sur 1 en procédant comme suit :

Pour personnaliser les paramètres de compilation par niveaux (console)

1. Ouvrez la page Fonctions de la console Lambda.
2. Choisissez une fonction Java pour laquelle vous souhaitez personnaliser la compilation par niveau.
3. Choisissez l'onglet Configuration, puis Variables d'environnement dans le menu de gauche.
4. Choisissez Modifier.
5. Choisissez Ajouter une variable d'environnement.
6. Pour Nom de la clé, saisissez `JAVA_TOOL_OPTIONS`. Pour Valeur, saisissez `-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.




7. Choisissez Save (Enregistrer).

> **Note**
>
> Vous pouvez également utiliser Lambda SnapStart pour atténuer les problèmes de démarrage à froid. SnapStartutilise des instantanés mis en cache de votre environnement d'exécution pour améliorer de manière significative les performances de démarrage. Pour plus d'informations sur les SnapStart fonctionnalités, les limitations et les régions prises en charge, consultezAméliorer les performances de démarrage avec Lambda SnapStart.

## Exemple : personnalisation du comportement du GC à l'aide de JAVA _ _ TOOL OPTIONS

Les exécutions Java 11 utilisent le récupérateur de mémoire (GC) Serial pour la récupération de mémoire. Par défaut, les exécutions Java 17 utilisent également le GC Serial. Toutefois, avec Java 17, vous pouvez également utiliser la variable d'environnement `JAVA_TOOL_OPTIONS` pour modifier le GC par défaut. Vous pouvez choisir entre le GC Parallel et le GC Shenandoah.

Par exemple, si votre charge de travail utilise plus de mémoire et de multiplesCPUs, pensez à utiliser le Parallel GC pour de meilleures performances. Vous pouvez le faire en ajoutant ce qui suit à la valeur de votre variable d'environnement `JAVA_TOOL_OPTIONS` :

```
-XX:+UseParallelGC
```

# Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions Java

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés fournissant des informations sur l'invocation, la fonction et l'environnement d'exécution.

Méthodes de contexte

- `getRemainingTimeInMillis()` – Renvoie le nombre de millisecondes restant avant l'expiration de l'exécution.
- `getFunctionName()` – Renvoie le nom de la fonction Lambda.
- `getFunctionVersion()` – Renvoie la version de la fonction.
- `getInvokedFunctionArn()`— Renvoie le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `getMemoryLimitInMB()` – Renvoie la quantité de mémoire allouée à la fonction.
- `getAwsRequestId()` – Renvoie l'identifiant de la demande d'invocation.
- `getLogGroupName()` – Renvoie le groupe de journaux pour la fonction.
- `getLogStreamName()` – Renvoie le flux de journal de l'instance de fonction.
- `getIdentity()` – (applications mobiles) Renvoie des informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `getClientContext()` – (applications mobiles) Renvoie le contexte client fourni à Lambda par l'application client.
- `getLogger()` – Renvoie l'objet enregistreur pour la fonction.

L'exemple suivant montre une fonction qui utilise l'objet contexte pour accéder à l'enregistreur Lambda.

Example handler.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import java.util.Map;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, Void>{

  @Override
  public Void handleRequest(Map<String,String> event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    logger.log("EVENT TYPE: " + event.getClass());
    return null;
  }
}
```

La fonction enregistre le type de classe de l'événement entrant avant de le renvoyer`null`.

Example sortie de journal

```
EVENT TYPE: class java.util.LinkedHashMap
```

L'interface de l'objet de contexte est disponible dans la aws-lambda-java-corebibliothèque. Vous pouvez implémenter cette interface pour créer une classe de contexte à des fins de test. L'exemple suivant montre une classe de contexte qui renvoie des valeurs factices pour la plupart des propriétés et un enregistreur de test opérationnel.

Example src/test/java/exemple/ .java TestContext

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.ClientContext;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class TestContext implements Context{

  public TestContext() {}
  public String getAwsRequestId(){
    return new String("495b12a8-xmpl-4eca-8168-160484189f99");
  }
  public String getLogGroupName(){
    return new String("/aws/lambda/my-function");
  }
```

```
  public String getLogStreamName(){
    return new String("2020/02/26/[$LATEST]704f8dxmpla04097b9134246b8438f1a");
  }
  public String getFunctionName(){
    return new String("my-function");
  }
  public String getFunctionVersion(){
    return new String("$LATEST");
  }
  public String getInvokedFunctionArn(){
    return new String("arn:aws:lambda:us-east-2:123456789012:function:my-function");
  }
  public CognitoIdentity getIdentity(){
    return null;
  }
  public ClientContext getClientContext(){
    return null;
  }
  public int getRemainingTimeInMillis(){
    return 300000;
  }
  public int getMemoryLimitInMB(){
    return 512;
  }
  public LambdaLogger getLogger(){
    return new TestLogger();
  }

}
```

Pour de plus amples informations sur la journalisation, veuillez consulter Enregistrez et surveillez les fonctions Java Lambda.

## Contexte dans des exemples d'applications

Le GitHub référentiel de ce guide inclut des exemples d'applications illustrant l'utilisation de l'objet de contexte. Chaque exemple d'application inclut des scripts pour faciliter le déploiement et le nettoyage, un modèle AWS Serverless Application Model (AWS SAM) et des ressources de support.

Exemples d’applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d’événement en entrée.

- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

# Enregistrez et surveillez les fonctions Java Lambda

AWS Lambda surveille automatiquement les fonctions Lambda et envoie des entrées de journal à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des détails sur chaque invocation et d'autres sorties provenant du code de votre fonction au flux de journaux. Pour plus d'informations sur CloudWatch les journaux, consultezUtilisation des CloudWatch journaux et journaux avec Lambda.

Pour produire des journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur java.lang.System, ou tout module de journalisation qui écrit sur stdout ou stderr.

Sections


- Création d'une fonction qui renvoie des journaux
- Utilisation des contrôles de journalisation avancés de Lambda avec Java
- Implémentation de la journalisation avancée avec Log4j2 et SLF4J
- Utilisation d'autres outils et bibliothèques de journalisation
- Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour la journalisation structurée
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux
- Exemple de code de journalisation


## Création d'une fonction qui renvoie des journaux

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur java.lang.System ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. La aws-lambda-java-corebibliothèque fournit une classe d'enregistreur nommée à `LambdaLogger` laquelle vous pouvez accéder depuis l'objet de contexte. La classe d'enregistreur prend en charge les journaux multilignes.

L'exemple suivant utilise l'enregistreur `LambdaLogger` fourni par l'objet de contexte.

### Example handler.java

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Object, String>{
  Gson gson = new GsonBuilder().setPrettyPrinting().create();
  @Override
  public String handleRequest(Object event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    String response = new String("SUCCESS");
    // log execution details
    logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
    logger.log("CONTEXT: " + gson.toJson(context));
    // process event
    logger.log("EVENT: " + gson.toJson(event));
    return response;
  }
}
```

### Example format des journaux

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
EVENT:
{
  "records": [
    {
      "messageId": "19dd0b57-xmpl-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
```

```
        ...
    }
  ]
}
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed
 Duration: 200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

L'environnement d'exécution Java enregistre les lignes `START`, `END` et `REPORT` pour chaque invocation. La ligne de rapport fournit les détails suivants.

REPORTchamps de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled : pour les demandes suivies, résultat de l'échantillonnage.

## Utilisation des contrôles de journalisation avancés de Lambda avec Java

Pour mieux contrôler la manière dont les journaux de vos fonctions sont capturés, traités et consommés, vous pouvez configurer les options de journalisation suivantes pour les environnements d'exécution Java pris en charge :

- Format de journal : choisissez entre le format texte brut et le JSON format structuré pour les journaux de votre fonction
- Niveau de journalisation : pour les journaux au JSON format, choisissez le niveau de détail des journaux auxquels Lambda envoie CloudWatch, par exempleERROR, DEBUG ou INFO

- Groupe de journaux : choisissez le groupe de CloudWatch journaux auquel votre fonction envoie les journaux

Pour plus d'informations sur ces options de journalisation et pour savoir comment configurer votre fonction pour les utiliser, consultez the section called "Configuration des journaux de fonctions".

Pour utiliser le format de journal et les options de niveau de journal avec vos fonctions Java Lambda, consultez les instructions des sections suivantes.

## Utilisation d'un format de JSON journal structuré avec Java

Si vous sélectionnez le JSON format de journal de votre fonction, Lambda enverra les journaux en utilisant la `LambdaLogger` classe telle que structurée. JSON Chaque objet JSON journal contient au moins quatre paires clé-valeur avec les clés suivantes :

- `"timestamp"` - heure à laquelle le message de journal a été généré
- `"level"` - niveau de journalisation attribué au message
- `"message"` - contenu du message de journal
- `"AWSrequestId"` - identifiant unique de la demande pour l'invocation de la fonction

Selon la méthode de journalisation que vous utilisez, les sorties de journal de votre fonction capturées au JSON format peuvent également contenir des paires clé-valeur supplémentaires.

Pour attribuer un niveau aux journaux que vous créez à l'aide de l'enregistreur `LambdaLogger`, vous devez fournir un argument `LogLevel` dans votre commande de journalisation, comme illustré dans l'exemple suivant.

Example Code de journalisation Java

```
LambdaLogger logger = context.getLogger();
logger.log("This is a debug log", LogLevel.DEBUG);
```

Cette sortie de journal produite par cet exemple de code serait capturée dans CloudWatch Logs comme suit :

Example JSONenregistrement du journal

```
{
```

```
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"DEBUG",
    "message":"This is a debug log",
    "AWSrequestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Si vous n'attribuez aucun niveau à la sortie de votre journal, Lambda lui attribuera automatiquement le niveau. INFO

Si votre code utilise déjà une autre bibliothèque de journalisation pour produire des journaux JSON structurés, vous n'avez aucune modification à apporter. Lambda ne double code aucun journal déjà codé. JSON Même si vous configurez votre fonction pour utiliser le format JSON journal, vos sorties de journalisation apparaissent CloudWatch dans la JSON structure que vous définissez.

### Utilisation du filtrage au niveau du journal avec Java

AWS Lambda Pour filtrer les journaux de vos applications en fonction de leur niveau de journalisation, votre fonction doit utiliser des journaux JSON formatés. Vous pouvez effectuer cette opération de deux façons :

- Créez des sorties de journal à l'aide de la norme `LambdaLogger` et configurez votre fonction pour utiliser le formatage des JSON journaux. Lambda filtre ensuite les sorties de votre journal à l'aide de la paire clé-valeur « niveau » de l'JSONobjet décrit dans. the section called “Utilisation d'un format de JSON journal structuré avec Java” Pour savoir comment configurer le format de journal de votre fonction, consultez the section called “Configuration des journaux de fonctions”.
- Utilisez une autre bibliothèque ou méthode de journalisation pour créer des journaux JSON structurés dans votre code qui incluent une paire clé-valeur « niveau » définissant le niveau de sortie du journal. Vous pouvez utiliser n'importe quelle bibliothèque de journalisation capable d'écrire JSON des journaux dans `stdout` ou`stderr`. Par exemple, vous pouvez utiliser Powertools for AWS Lambda ou le package Log4j2 pour générer des sorties de journal JSON structurées à partir de votre code. Pour en savoir plus, consultez the section called “Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour la journalisation structurée” et the section called “Implémentation de la journalisation avancée avec Log4j2 et SLF4J”.

Lorsque vous configurez votre fonction pour utiliser le filtrage au niveau des journaux, vous devez sélectionner l'une des options suivantes pour le niveau de journaux que Lambda doit envoyer à Logs : CloudWatch

| Niveau de journalisation | Utilisation standard |
| --- | --- |
| TRACE(le plus détaillé) | Les informations les plus précises utilisées pour tracer le chemin d'exécution de votre code |
| DEBUG | Informations détaillées pour le débogage du système |
| INFO | Messages qui enregistrent le fonctionnement normal de votre fonction |
| WARN | Messages relatifs à des erreurs potentielles susceptibles d'entraîner un comportement inattendu si elles ne sont pas traitées |
| ERROR | Messages concernant les problèmes qui empêchent le code de fonctionner comme prévu |
| FATAL(moindre détail) | Messages relatifs à des erreurs graves entraînant l'arrêt du fonctionnement de l'applica tion |

Pour que Lambda puisse filtrer les journaux de votre fonction, vous devez également inclure une paire `"timestamp"` clé-valeur dans la sortie de votre JSON journal. L'heure doit être spécifiée dans un format d'horodatage RFC3339 valide. Si vous ne fournissez pas d'horodatage valide, Lambda attribuera le niveau au journal INFO et ajoutera un horodatage pour vous.

Lambda envoie les journaux du niveau sélectionné et d'un niveau inférieur à. CloudWatch Par exemple, si vous configurez un niveau de journalisation deWARN, Lambda enverra des journaux correspondant aux niveaux WARNERROR, etFATAL.

# Implémentation de la journalisation avancée avec Log4j2 et SLF4J

> **Note**
>
> AWS Lambda n'inclut pas Log4j2 dans ses environnements d'exécution gérés ni dans ses images de conteneur de base. Ils ne sont donc pas concernés par les problèmes décrits dans CVE -2021-44228, -2021-45046 et CVE -2021-45105. CVE
> Dans les cas où une fonction client inclut une version de Log4j2 affectée, nous avons apporté une modification aux environnements d'exécution gérés par Lambda Java et aux images des conteneurs de base afin d'atténuer les problèmes dans les versions -2021-44228, -2021-45046 et CVE -2021-45105. CVE CVE À la suite de cette modification, les clients utilisant Log4J2 peuvent voir une entrée de journal supplémentaire, similaire à « `Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)` ». Toute chaîne de journal qui fait référence au mappeur jndi dans la sortie Log4J2 sera remplacée par « `Patched JndiLookup::lookup()` ».
> Indépendamment de ce changement, nous encourageons vivement tous les clients dont les fonctions incluent Log4j2 à mettre à jour vers la dernière version. Plus précisément, les clients utilisant la bibliothèque aws-lambda-java -log4j2 dans leurs fonctions doivent passer à la version 1.5.0 (ou ultérieure) et redéployer leurs fonctions. Cette version met à jour les dépendances sous-jacentes de l'utilitaire Log4j2 vers la version 2.17.0 (ou ultérieure). Le binaire aws-lambda-java -log4j2 mis à jour est disponible dans le dépôt Maven et son code source est disponible sur Github.
> Enfin, notez que les bibliothèques liées à aws-lambda-java-log4j (v1.0.0 ou 1.0.1) ne doivent en aucun cas être utilisées. Ces bibliothèques sont liées à la version 1.x de log4j qui a pris fin en 2015. Les bibliothèques ne sont pas prises en charge, ne sont pas maintenues, ne sont pas corrigées et présentent des failles de sécurité connues.

Pour personnaliser la sortie du journal, prendre en charge la journalisation pendant les tests unitaires et enregistrer les AWS SDK appels, utilisez Apache Log4j2 avec. SLF4J Log4j est une bibliothèque de journalisation pour les programmes Java qui vous permet de configurer les niveaux de journalisation et d'utiliser les bibliothèques appender. SLF4Jest une bibliothèque de façade qui vous permet de changer la bibliothèque que vous utilisez sans changer le code de votre fonction.

Pour ajouter l'ID de demande aux journaux de votre fonction, utilisez l'annexe de la bibliothèque aws-lambda-java-log4j2.

Example src/main/resources/log4j2.xml – Configuration de l'Appender

```
<Configuration>
  <Appenders>
    <Lambda name="Lambda" format="${env:AWS_LAMBDA_LOG_FORMAT:-TEXT}">
       <LambdaTextFormat>
         <PatternLayout>
             <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n </
pattern>
         </PatternLayout>
       </LambdaTextFormat>
       <LambdaJSONFormat>
         <JsonTemplateLayout eventTemplateUri="classpath:LambdaLayout.json" />
       </LambdaJSONFormat>
     </Lambda>
   </Appenders>
   <Loggers>
     <Root level="${env:AWS_LAMBDA_LOG_LEVEL:-INFO}">
       <AppenderRef ref="Lambda"/>
     </Root>
     <Logger name="software.amazon.awssdk" level="WARN" />
     <Logger name="software.amazon.awssdk.request" level="DEBUG" />
   </Loggers>
 </Configuration>
```

Vous pouvez décider comment vos journaux Log4j2 sont configurés pour le texte brut ou pour les JSON sorties en spécifiant une mise en page sous les `<LambdaTextFormat>` balises et. `<LambdaJSONFormat>`

Dans cet exemple, en mode texte, chaque ligne est précédée de la date, de l'heure, de l'ID de demande, du niveau de journal et du nom de classe. En JSON mode, le `<JsonTemplateLayout>` est utilisé avec une configuration fournie avec la `aws-lambda-java-log4j2` bibliothèque.

SLF4Jest une bibliothèque de façade permettant de se connecter en code Java. Dans votre code de fonction, vous utilisez l'usine d'enregistrement pour récupérer un SLF4J enregistreur avec des méthodes pour les niveaux de journalisation telles que `info()` et. `warn()` Dans votre configuration de compilation, vous incluez la bibliothèque de journalisation et l'SLF4Jadaptateur dans le chemin de classe. En changeant les bibliothèques dans la configuration de build, vous pouvez changer le type d'enregistreur sans changer votre code de fonction. SLF4Jest nécessaire pour capturer les journaux SDK depuis Java.

Dans l'exemple de code suivant, la classe de gestionnaire utilise SLF4J pour récupérer un enregistreur.

Example src/main/java/example/HandlerS3.java — Connexion avec SLF4J

```
package example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;

import static org.apache.logging.log4j.CloseableThreadContext.put;


public class HandlerS3 implements RequestHandler<S3Event, String>{
    private static final Logger logger = LoggerFactory.getLogger(HandlerS3.class);

    @Override
    public String handleRequest(S3Event event, Context context) {
        for(var record : event.getRecords()) {
            try (var loggingCtx = put("awsRegion", record.getAwsRegion())) {
                loggingCtx.put("eventName", record.getEventName());
                loggingCtx.put("bucket", record.getS3().getBucket().getName());
                loggingCtx.put("key", record.getS3().getObject().getKey());

                logger.info("Handling s3 event");
            }
        }

        return "Ok";
    }
}
```

Ce code produit des sorties de journal similaires à ce qui suit :

Example format des journaux

```
{
    "timestamp": "2023-11-15T16:56:00.815Z",
    "level": "INFO",
```

```
    "message": "Handling s3 event",
    "logger": "example.HandlerS3",
    "AWSRequestId": "0bced576-3936-4e5a-9dcd-db9477b77f97",
    "awsRegion": "eu-south-1",
    "bucket": "java-logging-test-input-bucket",
    "eventName": "ObjectCreated:Put",
    "key": "test-folder/"
}
```

La configuration de construction prend en compte les dépendances d'exécution sur l'appendice et l'SLF4Jadaptateur Lambda, ainsi que les dépendances d'implémentation sur Log4j2.

Example build.gradle – Dépendances de journalisation

```
dependencies {
    ...
    'com.amazonaws:aws-lambda-java-log4j2:[1.6.0,)',
    'com.amazonaws:aws-lambda-java-events:[3.11.3,)',
    'org.apache.logging.log4j:log4j-layout-template-json:[2.17.1,)',
    'org.apache.logging.log4j:log4j-slf4j2-impl:[2.19.0,)',
    ...
}
```

Lorsque vous exécutez votre code localement pour des tests, l'objet contecte avec l'enregistreur Lambda n'est pas disponible, et il n'y a pas d'ID de requête que l'appender Lambda puisse utiliser. Pour des exemples de configurations de test, consultez les exemples d'applications dans la section suivante.

## Utilisation d'autres outils et bibliothèques de journalisation

Powertools for AWS Lambda (Java) est une boîte à outils destinée aux développeurs qui permet de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs. L'utilitaire Logging fournit un enregistreur optimisé pour Lambda qui inclut des informations supplémentaires sur le contexte des fonctions dans toutes vos fonctions avec une sortie structurée comme suit. JSON Utilisez cet utilitaire pour effectuer les opérations suivantes :

- Capturez les champs clés du contexte Lambda, du démarrage à froid et des structures enregistrant la sortie sous JSON
- Journaliser les événements d'invocation Lambda lorsque cela est demandé (désactivé par défaut)

- Imprimer tous les journaux uniquement pour un pourcentage d'invocations via l'échantillonnage des journaux (désactivé par défaut)
- Ajouter des clés supplémentaires au journal structuré à tout moment
- Utilisez un formateur de journal personnalisé (Bring Your Own Formatter) pour générer les journaux dans une structure compatible avec la journalisation de votre organisation RFC

## Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Java Hello World avec des modules Powertools for AWS Lambda (Java) ~ intégrés à l'aide du AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Java 11
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du modèle Hello World Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Créez l'application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

La sortie du journal se présente comme suit :

```
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.095000
 INIT_START Runtime Version: java:11.v15    Runtime Version ARN: arn:aws:lambda:eu-
central-1::runtime:0a25e3e7a1cc9ce404bc435eeb2ad358d8fa64338e618d0c224fe509403583ca
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.114000
 Picked up JAVA_TOOL_OPTIONS: -XX:+TieredCompilation -XX:TieredStopAtLevel=1
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.793000
 Transforming org/apache/logging/log4j/core/lookup/JndiLookup
 (lambdainternal.CustomerClassLoader@1a6c5a9e)
```

```
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:35.252000
 START RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765 Version: $LATEST
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.531000 {
  "_aws": {
    "Timestamp": 1675416276051,
    "CloudWatchMetrics": [
      {
        "Namespace": "sam-app-powerools-java",
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ],
        "Dimensions": [
          [
            "Service",
            "FunctionName"
          ]
        ]
      }
    ]
  },
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
 "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "FunctionName": "sam-app-HelloWorldFunction-y9Iu1FLJJBGD",
  "functionVersion": "$LATEST",
  "ColdStart": 1.0,
  "Service": "service_undefined",
  "logStreamId": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.974000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.AWSXRayRecorder <init>
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.config.DaemonConfiguration <init>
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000
 INFO: Environment variable AWS_XRAY_DAEMON_ADDRESS is set. Emitting to daemon on
 address XXXX.XXXX.XXXX.XXXX:2000.
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.331000
 09:24:37.294 [main] INFO  helloworld.App - {"version":null,"resource":"/
hello","path":"/hello/","httpMethod":"GET","headers":{"Accept":"*/
*","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-
```

```
Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-
SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-
Viewer-ASN":"16509","CloudFront-Viewer-Country":"IE","Host":"XXXX.execute-
api.eu-central-1.amazonaws.com","User-Agent":"curl/7.86.0","Via":"2.0
 f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":"t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q==","X-
Amzn-Trace-Id":"Root=1-63dcd2d1-25f90b9d1c753a783547f4dd","X-Forwarded-
For":"XX.XXX.XXX.XX, XX.XXX.XXX.XX","X-Forwarded-Port":"443","X-
Forwarded-Proto":"https"},"multiValueHeaders":{"Accept":["*/
*"],"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-Viewer":
["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-SmartTV-
Viewer":["false"],"CloudFront-Is-Tablet-Viewer":["false"],"CloudFront-Viewer-
ASN":["16509"],"CloudFront-Viewer-Country":["IE"],"Host":["XXXX.execute-
api.eu-central-1.amazonaws.com"],"User-Agent":["curl/7.86.0"],"Via":["2.0
 f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)"],"X-Amz-
Cf-Id":["t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q=="],"X-
Amzn-Trace-Id":["Root=1-63dcd2d1-25f90b9d1c753a783547f4dd"],"X-Forwarded-
For":["XXX, XXX"],"X-Forwarded-Port":["443"],"X-Forwarded-Proto":
["https"]},"queryStringParameters":null,"multiValueQueryStringParameters":null,"pathParamet
{"accountId":"XXX","stage":"Prod","resourceId":"at73a1","requestId":"ba09ecd2-
acf3-40f6-89af-fad32df67597","operationName":null,"identity":
{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"apiK
hello","httpMethod":"GET","apiId":"XXX","path":"/Prod/
hello/","authorizer":null},"body":null,"isBase64Encoded":false}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.351000
 09:24:37.351 [main] INFO  helloworld.App - Retrieving https://
checkip.amazonaws.com
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.313000 {
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
 "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "xray_trace_id": "1-63dcd2d1-25f90b9d1c753a783547f4dd",
  "functionVersion": "$LATEST",
  "Service": "service_undefined",
  "logStreamId": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000 END
 RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000
 REPORT RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765    Duration: 4118.98 ms
 Billed Duration: 4119 ms    Memory Size: 512 MB    Max Memory Used: 152 MB    Init
 Duration: 1155.47 ms
```

```
XRAY TraceId: 1-63dcd2d1-25f90b9d1c753a783547f4dd    SegmentId: 3a028fee19b895cb
 Sampled: true
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

### Gestion de la conservation des journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation après laquelle les journaux CloudWatch sont automatiquement supprimés. Pour configurer la conservation des journaux, ajoutez ce qui suit à votre AWS SAM modèle :

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l’éditeur Code intégré, vous trouverez les journaux dans les résultats d’exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

## Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
```

```
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

# Exemple de code de journalisation

Le GitHub référentiel de ce guide inclut des exemples d'applications illustrant l'utilisation de différentes configurations de journalisation. Chaque exemple d'application inclut des scripts pour faciliter le déploiement et le nettoyage, un AWS SAM modèle et des ressources de support.

Exemples d’applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d’événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

L’exemple `java-basic` d’application présente une configuration de journalisation minimale qui prend en charge les tests de journalisation. Le code du gestionnaire utilise l’enregistreur `LambdaLogger` fourni par l’objet de contexte. Pour les tests, l’application utilise une classe `TestLogger` personnalisée qui implémente l’interface `LambdaLogger` avec un logger Log4j2. Il est utilisé SLF4J comme façade pour assurer la compatibilité avec le AWS SDK. Les bibliothèques de journalisation sont exclues de la sortie de build pour limiter la taille du package de déploiement.

# Instrumentation du code Java dans AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des deux SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- Kit SDK AWS X-Ray pour Java— Et SDK pour générer et envoyer des données de trace à X-Ray.
- Powertools for AWS Lambda (Java) — Une boîte à outils pour les développeurs permettant de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections

- Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour le traçage
- Utilisation de Powertools pour AWS Lambda (Java) et AWS CDK pour le traçage
- Utilisation ADOT pour instrumenter vos fonctions Java
- Utiliser le X-Ray SDK pour instrumenter vos fonctions Java

- Activation du suivi avec la console Lambda
- Activation du traçage avec le Lambda API
- Activation du traçage avec AWS CloudFormation
- Interprétation d'un suivi X-Ray
- Stockage des dépendances d'exécution dans une couche (X-RaySDK)
- Traçage par rayons X dans les applications d'échantillonnage (X-RaySDK)

## Utilisation de Powertools pour AWS Lambda (Java) et AWS SAM pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Java Hello World avec des modules Powertools for AWS Lambda (Java) intégrés à l'aide du AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Java 11
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l’application à l’aide du modèle Hello World Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Créez l’application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
New XRay Service Graph
  Start time: 2023-02-03 14:31:48+01:00
  End time: 2023-02-03 14:31:48+01:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-y9Iu1FLJJBGD -
 Edges: []
   Summary_statistics:
```

```
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 5.587
  Reference Id: 1 - client - sam-app-HelloWorldFunction-y9Iu1FLJJBGD - Edges: [0]
   Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 3] at (2023-02-03T14:31:48.500000) with id
 (1-63dd0cc4-3c869dec72a586875da39777) and duration (5.603s)
 - 5.587s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD [HTTP: 200]
 - 4.053s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD
   - 1.181s - Initialization
   - 4.037s - Invocation
     - 1.981s - ## handleRequest
       - 1.840s - ## getPageContents
   - 0.000s - Overhead
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

## Utilisation de Powertools pour AWS Lambda (Java) et AWS CDK pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Java Hello World avec des modules Powertools for AWS Lambda (Java) intégrés à l'aide du AWS CDK. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message hello world.

### Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Java 11
- AWS CLI version 2
- AWS CDK version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

### Déployer un exemple d' AWS CDK application

1. Créez un répertoire de projets pour votre nouvelle application.

```
mkdir hello-world
cd hello-world
```

2. Initialisez l'application.

```
cdk init app --language java
```

3. Créez un projet Maven avec la commande suivante :

```
mkdir app
cd app
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Ouvrez `pom.xml` dans le répertoire `hello-world\app\Function` et remplacez le code existant par le code suivant qui inclut les dépendances et les plugins Maven pour Powertools.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>helloworld</groupId>
  <artifactId>Function</artifactId>
  <packaging>jar</packaging>
```

```
  <version>1.0-SNAPSHOT</version>
  <name>Function</name>
  <url>http://maven.apache.org</url>
<properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <log4j.version>2.17.2</log4j.version>
</properties>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-tracing</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-metrics</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-logging</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.2</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-events</artifactId>
            <version>3.11.1</version>
        </dependency>
  </dependencies>
<build>
    <plugins>
        <plugin>
```

```
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>aspectj-maven-plugin</artifactId>
            <version>1.14.0</version>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
                <complianceLevel>${maven.compiler.target}</complianceLevel>
                <aspectLibraries>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-tracing</artifactId>
                    </aspectLibrary>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-metrics</artifactId>
                    </aspectLibrary>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-logging</artifactId>
                    </aspectLibrary>
                </aspectLibraries>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.4.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
```

```
 implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCache
                                </transformer>
                            </transformers>
                            <createDependencyReducedPom>false</
createDependencyReducedPom>
                            <finalName>function</finalName>

                        </configuration>
                    </execution>
                </executions>
                <dependencies>
                    <dependency>
                        <groupId>com.github.edwgiz</groupId>
                        <artifactId>maven-shade-plugin.log4j2-cachefile-
transformer</artifactId>
                        <version>2.15</version>
                    </dependency>
                </dependencies>
        </plugin>
    </plugins>
</build>
</project>
```

5. Créez le répertoire `hello-world\app\src\main\resource` et créez `log4j.xml` pour la configuration du journal.

```
mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml
```

6. Ouvrez `log4j.xml` et ajoutez le code suivant.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="JsonAppender" target="SYSTEM_OUT">
            <JsonTemplateLayout
 eventTemplateUri="classpath:LambdaJsonLayout.json" />
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="JsonLogger" level="INFO" additivity="false">
```

```
            <AppenderRef ref="JsonAppender"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="JsonAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

7. Ouvrez `App.java` à partir du répertoire `hello-world\app\Function\src\main\java\helloworld` et remplacez le code existant par le code suivant. Il s'agit du code de la fonction Lambda.

```
package helloworld;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.metrics.Metrics;
import software.amazon.lambda.powertools.tracing.CaptureMode;
import software.amazon.lambda.powertools.tracing.Tracing;

import static software.amazon.lambda.powertools.tracing.CaptureMode.*;

/**
 * Handler for requests to Lambda function.
 */
public class App implements RequestHandler<APIGatewayProxyRequestEvent,
 APIGatewayProxyResponseEvent> {
    Logger log = LogManager.getLogger(App.class);
```

```
    @Logging(logEvent = true)
    @Tracing(captureMode = DISABLED)
    @Metrics(captureColdStart = true)
    public APIGatewayProxyResponseEvent handleRequest(final
 APIGatewayProxyRequestEvent input, final Context context) {
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        headers.put("X-Custom-Header", "application/json");

        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent()
                .withHeaders(headers);
        try {
            final String pageContents = this.getPageContents("https://
checkip.amazonaws.com");
            String output = String.format("{ \"message\": \"hello world\",
 \"location\": \"%s\" }", pageContents);

            return response
                    .withStatusCode(200)
                    .withBody(output);
        } catch (IOException e) {
            return response
                    .withBody("{}")
                    .withStatusCode(500);
        }
    }
    @Tracing(namespace = "getPageContents")
    private String getPageContents(String address) throws IOException {
        log.info("Retrieving {}", address);
        URL url = new URL(address);
        try (BufferedReader br = new BufferedReader(new
 InputStreamReader(url.openStream()))) {
            return br.lines().collect(Collectors.joining(System.lineSeparator()));
        }
    }
}
```

8. Ouvrez `HelloWorldStack.java` à partir du répertoire `hello-world\src\main\java\com\myorg` et remplacez le code existant par le code suivant. Ce code utilise le constructeur Lambda et le constructeur ApiGatewayv 2 pour créer une fonction Lambda et REST API une fonction Lambda.

```
package com.myorg;
```

```
import software.amazon.awscdk.*;
import software.amazon.awscdk.services.apigatewayv2.alpha.*;
import
 software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegration;
import
 software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegrationProps
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.FunctionProps;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.lambda.Tracing;
import software.amazon.awscdk.services.logs.RetentionDays;
import software.amazon.awscdk.services.s3.assets.AssetOptions;
import software.constructs.Construct;

import java.util.Arrays;
import java.util.List;

import static java.util.Collections.singletonList;
import static software.amazon.awscdk.BundlingOutput.ARCHIVED;

public class HelloWorldStack extends Stack {
    public HelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloWorldStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        List<String> functionPackagingInstructions = Arrays.asList(
                "/bin/sh",
                "-c",
                "cd Function " +
                        "&& mvn clean install " +
                        "&& cp /asset-input/Function/target/function.jar /asset-
output/"
        );
        BundlingOptions.Builder builderOptions = BundlingOptions.builder()
                .command(functionPackagingInstructions)
                .image(Runtime.JAVA_11.getBundlingImage())
                .volumes(singletonList(
```

```
                        // Mount local .m2 repo to avoid download all the
dependencies again inside the container
                        DockerVolume.builder()
                                .hostPath(System.getProperty("user.home") +
"/.m2/")
                                .containerPath("/root/.m2/")
                                .build()
                ))
                .user("root")
                .outputType(ARCHIVED);

        Function function = new Function(this, "Function", FunctionProps.builder()
                .runtime(Runtime.JAVA_11)
                .code(Code.fromAsset("app", AssetOptions.builder()
                        .bundling(builderOptions
                                .command(functionPackagingInstructions)
                                .build())
                        .build()))
                .handler("helloworld.App::handleRequest")
                .memorySize(1024)
                .tracing(Tracing.ACTIVE)
                .timeout(Duration.seconds(10))
                .logRetention(RetentionDays.ONE_WEEK)
                .build());

        HttpApi httpApi = new HttpApi(this, "sample-api", HttpApiProps.builder()
                .apiName("sample-api")
                .build());

        httpApi.addRoutes(AddRoutesOptions.builder()
                .path("/")
                .methods(singletonList(HttpMethod.GET))
                .integration(new HttpLambdaIntegration("function", function,
HttpLambdaIntegrationProps.builder()
                        .payloadFormatVersion(PayloadFormatVersion.VERSION_2_0)
                        .build()))
                .build());

        new CfnOutput(this, "HttpApi", CfnOutputProps.builder()
                .description("Url for Http Api")
                .value(httpApi.getApiEndpoint())
                .build());
    }
```

```
}
```

9. Ouvrez `pom.xml` à partir du répertoire `hello-world` et remplacez le code existant par le code suivant.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd"
         xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.myorg</groupId>
    <artifactId>hello-world</artifactId>
    <version>0.1</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <cdk.version>2.70.0</cdk.version>
        <constructs.version>[10.0.0,11.0.0)</constructs.version>
        <junit.version>5.7.1</junit.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <mainClass>com.myorg.HelloWorldApp</mainClass>
                </configuration>
            </plugin>
```

```
        </plugins>
    </build>

    <dependencies>
        <!-- AWS Cloud Development Kit -->
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>aws-cdk-lib</artifactId>
            <version>${cdk.version}</version>
        </dependency>
        <dependency>
            <groupId>software.constructs</groupId>
            <artifactId>constructs</artifactId>
            <version>${constructs.version}</version>
        </dependency>
        <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter</artifactId>
          <version>${junit.version}</version>
          <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigatewayv2-alpha</artifactId>
            <version>${cdk.version}-alpha.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigatewayv2-integrations-alpha</artifactId>
            <version>${cdk.version}-alpha.0</version>
        </dependency>
    </dependencies>
</project>
```

10. Assurez-vous d'être dans le répertoire `hello-world` et déployez votre application.

```
cdk deploy
```

11. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`HttpApi`].OutputValue' --output text
```

12. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

13. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0
```

```
XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

14. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
cdk destroy
```

## Utilisation ADOT pour instrumenter vos fonctions Java

ADOTfournit des couches Lambda entièrement gérées qui regroupent tout ce dont vous avez besoin pour collecter des données de télémétrie à l'aide du. OTel SDK En consommant cette couche, vous pouvez instrumenter vos fonctions Lambda sans avoir à modifier le code de fonction. Vous pouvez également configurer votre couche pour effectuer une initialisation personnalisée deOTel. Pour plus d'informations, consultez la section Configuration personnalisée du ADOT collecteur sur Lambda dans la ADOT documentation.

Pour les environnements d'exécution Java, vous pouvez choisir entre deux couches à consommer :

- AWS couche Lambda gérée pour ADOT Java (agent d'instrumentation automatique) : cette couche transforme automatiquement le code de votre fonction au démarrage pour collecter des données de suivi. Pour obtenir des instructions détaillées sur la façon d'utiliser cette couche avec l'agent ADOT Java, consultez AWS Distro for OpenTelemetry Lambda Support for Java (agent d'instrumentation automatique) ADOT dans la documentation.
- AWS couche Lambda gérée pour ADOT Java — Cette couche fournit également une instrumentation intégrée pour les fonctions Lambda, mais elle nécessite quelques modifications de code manuelles pour initialiser le. OTel SDK Pour obtenir des instructions détaillées sur la façon de consommer cette couche, consultez AWS Distro for OpenTelemetry Lambda Support for Java ADOT dans la documentation.

## Utiliser le X-Ray SDK pour instrumenter vos fonctions Java

Pour enregistrer les données relatives aux appels que votre fonction fait à d'autres ressources et services de votre application, vous pouvez ajouter X-Ray SDK pour Java à votre configuration de build. L'exemple suivant montre une configuration de build Gradle qui inclut les bibliothèques qui activent l'instrumentation automatique des AWS SDK for Java 2.x clients.

Example build.gradle – Dépendances du suivi

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.16.1')
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')
    ...
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk'
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'
    ...
}
```

Après avoir ajouté les dépendances correctes et apporté les modifications de code nécessaires, activez le suivi dans la configuration de votre fonction via la console Lambda ou le. API

## Activation du suivi avec la console Lambda

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

## Activation du traçage avec le Lambda API

Configurez le suivi sur votre fonction Lambda avec AWS CLI ou utilisez AWS SDK les opérations suivantes : API

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant active le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du traçage avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
```

```
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

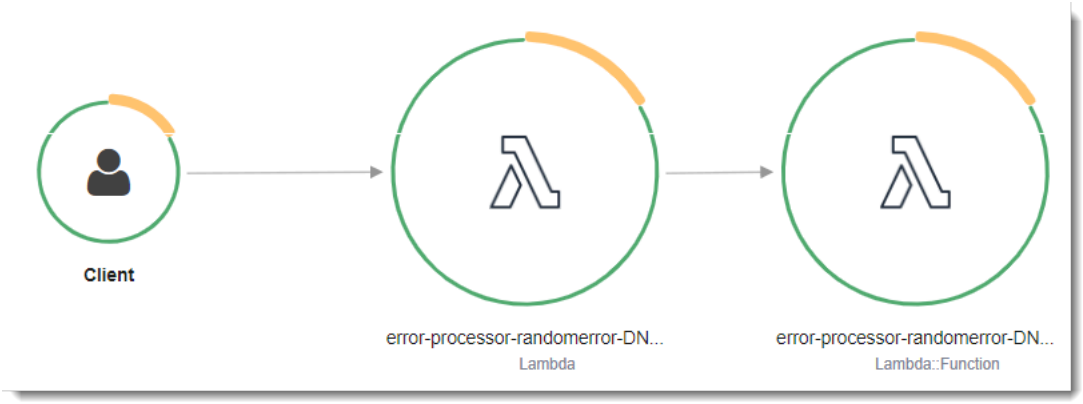
## Interprétation d'un suivi X-Ray

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Sinon, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

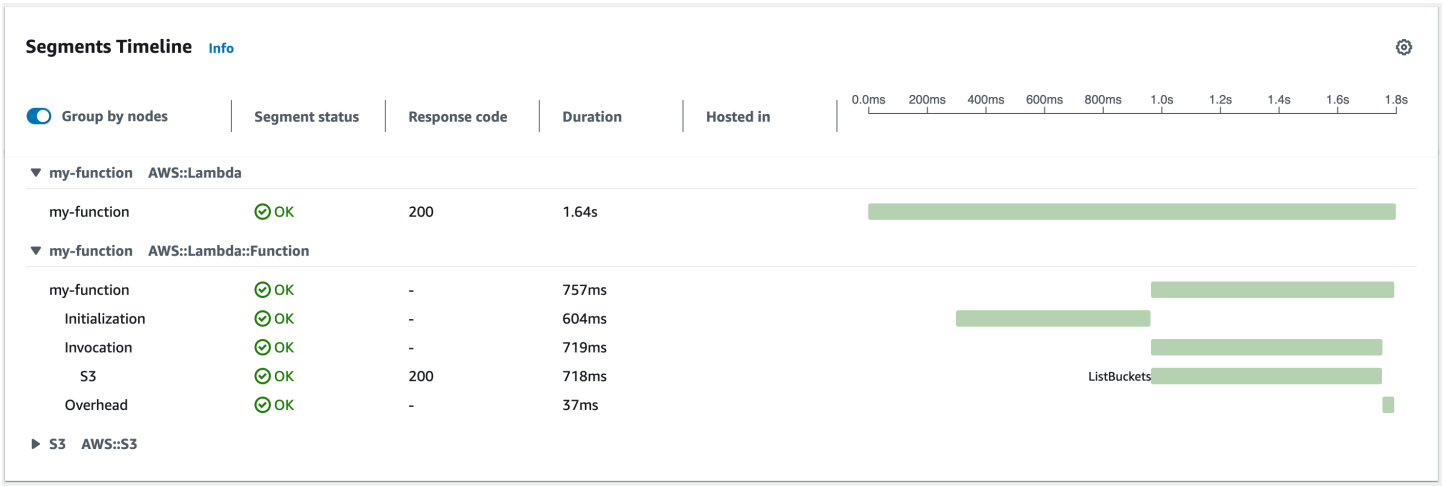
Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

> **Note**
>
> Les fonctions Lambda SnapStart incluent également un sous-segment `Restore`. Le `Restore` sous-segment indique le temps nécessaire à Lambda pour restaurer un instantané, charger le runtime JVM () et exécuter les `afterRestore` éventuels hooks d'exécution. Le processus de restauration des instantanés peut inclure du temps consacré à des activités en dehors de la MicroVM. Cette heure est indiquée dans le sous-segment `Restore`. Le temps passé en dehors de la microVM pour restaurer un instantané ne vous est pas facturé.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez Kit SDK AWS X-Ray pour Java dans le Guide du développeur AWS X-Ray .

> **Tarification**
>
> Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

## Stockage des dépendances d'exécution dans une couche (X-RaySDK)

Si vous utilisez le X-Ray SDK pour instrumenter votre code de fonction chez les AWS SDK clients, votre package de déploiement peut devenir assez volumineux. Pour éviter de télécharger les dépendances d'exécution chaque fois que vous mettez à jour le code de votre fonction, empaquetez le X-Ray SDK dans une couche Lambda.

L'exemple suivant montre une `AWS::Serverless::LayerVersion` ressource qui stocke le AWS SDK for Java et X-Ray SDK pour Java.

Example template.yml : couche de dépendances

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: build/distributions/blank-java.zip
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-java-lib
      Description: Dependencies for the blank-java sample app.
      ContentUri: build/blank-java-lib.zip
      CompatibleRuntimes:
        - java21
```

Avec cette configuration, vous ne mettez à jour les fichiers de couche de bibliothèque que si vous modifiez vos dépendances d'exécution. Étant donné que le package de déploiement de la fonction contient uniquement votre code, cela peut contribuer à réduire les temps de chargement.

La création d'une couche pour les dépendances nécessite des modifications de configuration de création pour créer l'archive des couches avant le déploiement. Pour un exemple pratique, consultez l'exemple d'application java-basic sur. GitHub

## Traçage par rayons X dans les applications d'échantillonnage (X-RaySDK)

Le GitHub référentiel de ce guide inclut des exemples d'applications illustrant l'utilisation du traçage par rayons X. Chaque exemple d'application inclut des scripts pour faciliter le déploiement et le nettoyage, un AWS SAM modèle et des ressources de support.

Exemples d’applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d’événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

Tous les exemples d'applications ont un suivi actif activé pour les fonctions Lambda. Par exemple, l'`s3-java`application montre l'instrumentation automatique des AWS SDK for Java 2.x clients, la gestion des segments pour les tests, les sous-segments personnalisés et l'utilisation de couches Lambda pour stocker les dépendances d'exécution.

# Exemples d'applications Java pour AWS Lambda

Le GitHub référentiel de ce guide fournit des exemples d'applications illustrant l'utilisation de Java dans AWS Lambda. Chaque exemple d'application inclut des scripts pour faciliter le déploiement et le nettoyage, un AWS CloudFormation modèle et des ressources de support.

Exemples d’applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d’événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java-events — Ensemble de fonctions Java contenant du code squelette expliquant comment gérer les événements provenant de divers services tels qu'Amazon API GatewaySQS, Amazon et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java — Fonction Java qui traite les événements de notification provenant d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers image téléchargés.
- Utilisez API Gateway pour appeler une fonction Lambda : fonction Java qui analyse une table Amazon DynamoDB contenant des informations sur les employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour appeler la fonction.

Exécution de cadres Java populaires sur Lambda

- spring-cloud-function-samples— Un exemple tiré de Spring qui montre comment utiliser le framework Spring Cloud Function pour créer des fonctions AWS Lambda.
- Démo de l'application Spring Boot sans serveur : exemple qui montre comment configurer une application Spring Boot typique dans un environnement d'exécution Java géré avec ou sans SnapStart, ou en tant qu'image native de GraalVM avec un environnement d'exécution personnalisé.
- Démonstration de l'application Micronaut sans serveur — Un exemple qui montre comment utiliser Micronaut dans un environnement d'exécution Java géré avec et sans SnapStart, ou en tant

qu'image native de GraalVM avec un environnement d'exécution personnalisé. Pour en savoir plus, consultez les guides Micronaut/Lambda.

- Démo de l'application Quarkus sans serveur — Un exemple qui montre comment utiliser Quarkus dans un environnement d'exécution Java géré avec et sans SnapStart, ou en tant qu'image native de GraalVM avec un environnement d'exécution personnalisé. Pour en savoir plus, consultez le guide Quarkus/Lambda et le guide Quarkus/Lambda. SnapStart

Si vous débutez avec les fonctions Lambda dans Java, commencez par les exemples `java-basic`. Pour démarrer avec les sources d'événement Lambda, consultez les exemples `java-events`. Ces deux ensembles d'exemples montrent l'utilisation des bibliothèques Java de Lambda, des variables d'environnement AWS SDK, du et du AWS X-Ray SDK. Ces exemples nécessitent une configuration minimale et peuvent être déployé à partir de la ligne de commande en moins d'une minute.

# Création de fonctions Lambda avec Go

Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez un environnement d'exécution réservé au système d'exploitation (la famille `provided` d'exécution) pour déployer les fonctions Go sur Lambda.

Rubriques


- Prise en charge de l’exécution Go
- Outils et bibliothèques
- Définir le gestionnaire de fonctions Lambda dans Go
- Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Go
- Déployer des fonctions Lambda Go avec des archives de fichiers .zip
- Déployer des fonctions Lambda Go avec des images conteneurs
- Utilisation de couches pour les fonctions Go Lambda
- Enregistrez et surveillez les fonctions Go Lambda
- Instrumentation du code Go AWS Lambda


## Prise en charge de l’exécution Go

Le runtime géré Go 1.x pour Lambda est obsolète. Si vous avez des fonctions qui utilisent le runtime Go 1.x, vous devez migrer vos fonctions vers `provided.al2023` ou`provided.al2`. Les environnements d'`provided.al2`exécution `provided.al2023` et offrent plusieurs avantages`go1.x`, notamment la prise en charge de l'architecture arm64 (processeurs AWS Graviton2), des fichiers binaires plus petits et des temps d'appel légèrement plus rapides.

Aucune modification du code n’est requise pour cette migration. Les seules modifications requises concernent la façon dont vous créez votre package de déploiement et l’exécution que vous utilisez pour créer votre fonction. Pour plus d'informations, consultez la section Migration des AWS Lambda fonctions de l'environnement d'exécution Go1.x vers le runtime personnalisé sur Amazon Linux 2 sur le AWS blog Compute.

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Exécution réservée au système d'exploitation | `provided.al2023` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Exécution réservée au système d'exploitation | `provided.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

## Outils et bibliothèques

Lambda fournit les outils et bibliothèques suivants pour l'environnement d'exécution Go :

- AWS SDKpour Go : le langage de programmation officiel AWS SDK pour Go.
- github.com/aws/ aws-lambda-go /lambda : implémentation du modèle de programmation Lambda pour Go. Ce package est utilisé AWS Lambda pour appeler votre gestionnaire.
- github.com/aws/ aws-lambda-go /lambdacontext : aides pour accéder aux informations contextuelles à partir de l'objet de contexte.
- github.com/aws/ aws-lambda-go /events : cette bibliothèque fournit des définitions de type pour les intégrations de sources d'événements courantes.
- github.com/aws/ aws-lambda-go /cmd/ build-lambda-zip : Cet outil peut être utilisé pour créer une archive de fichiers .zip sous Windows.

Pour plus d'informations, voir aws-lambda-goci-dessous GitHub.

Lambda fournit les exemples d'applications suivants pour l'environnement d'exécution Go :

Exemples d'applications Lambda en Go

- go-al2 – Une fonction Hello World qui renvoie l'adresse IP publique. Cette application utilise l'exécution personnalisée `provided.al2`.

- blank-go — Une fonction Go qui montre l'utilisation des bibliothèques Go de Lambda, de la journalisation, des variables d'environnement et du. AWS SDK Cette application utilise l’exécution `go1.x`.

# Définir le gestionnaire de fonctions Lambda dans Go

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Rubriques


- Notions de base sur Go Handler
- Dénomination
- Gestionnaire de fonctions Lambda utilisant des types structurés
- Utilisation de l'état global
- Meilleures pratiques en matière de code pour les fonctions Go Lambda


## Notions de base sur Go Handler

Une fonction Lambda écrite en Go est créée en tant qu'exécutable Go. Dans le code de votre fonction Lambda, vous devez inclure le package github.com/aws/ aws-lambda-go /lambda, qui implémente le modèle de programmation Lambda pour Go. De plus, vous devez mettre en œuvre le code de fonction du gestionnaire et une fonction `main()`.

Example Une fonction Lambda dans Go

```
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(ctx context.Context, event *MyEvent) (*string, error) {
 if event == nil {
  return nil, fmt.Errorf("received nil event")
 }
```

```
 message := fmt.Sprintf("Hello %s!", event.Name)
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Voici un exemple d'entrée pour cette fonction :

```
{
  "name": "Jane"
}
```

Notez ce qui suit :

- package main : dans Go, le package contenant `func main()` doit toujours être nommé `main`.
- import : utilisez cette instruction pour inclure les bibliothèques que votre fonction Lambda requiert. Dans cette instance, elle inclut :
  - context: Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Go.
  - fmt : l'objet Go Formatting utilisé pour formater la valeur de retour de votre fonction.
  - github.com/aws/ aws-lambda-go /lambda : comme mentionné précédemment, implémente le modèle de programmation Lambda pour Go.
- func HandleRequest (ctx context.Context, event *MyEvent) (*string, error) : il s'agit de la signature de votre gestionnaire Lambda. C'est le point d'entrée de votre fonction Lambda et il contient la logique qui est exécutée lorsque votre fonction est invoquée. En outre, les paramètres inclus désignent les éléments suivants :
  - ctx context.Context : fournit des informations d'exécution pour l'invocation de votre fonction Lambda. `ctx` est la variable que vous déclarez pour exploiter les informations disponibles via Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Go.
  - event * MyEvent : Il s'agit d'un paramètre nommé `event` qui pointe vers`MyEvent`. Il représente l'entrée de la fonction Lambda.
  - *string, error : le gestionnaire renvoie deux valeurs. La première est un pointeur sur une chaîne de caractères qui contient le résultat de la fonction Lambda. La deuxième est un type d'erreur, qui correspond à `nil` s'il n'y a pas d'erreur et qui contient les informations d'erreur standard en cas de problème.

- return &message, nil : renvoie deux valeurs. La première est un pointeur vers un message sous forme de chaîne de caractères, qui est un message d'accueil construit à partir du champ `Name` de l'événement d'entrée. La deuxième valeur, `nil`, indique que la fonction n'a pas rencontré d'erreurs.
- func main() : point d'entrée qui exécute votre code de fonction Lambda. C'est obligatoire.

  En ajoutant `lambda.Start(HandleRequest)` entre les accolades du code `func main(){}`, votre fonction Lambda sera exécutée. Selon les normes du langage Go, l'accolade ouvrante, `{`, doit être placée directement à la fin de la signature de la fonction `main`.

## Dénomination

exécutions de provid.al2 et provid.al2023

Pour les fonctions Go qui utilisent l'exécution `provided.al2` ou `provided.al2023` dans un package de déploiement .zip, le fichier exécutable qui contient le code de votre fonction doit être nommé `bootstrap`. Si vous déployez la fonction avec un fichier .zip, le fichier `bootstrap` doit se trouver à la racine du fichier .zip. Pour les fonctions Go qui l'exécution `provided.al2` ou `provided.al2023` dans une image du conteneur, vous pouvez utiliser n'importe quel nom pour le fichier exécutable.

Vous pouvez utiliser n'importe quel nom pour le gestionnaire. Pour référencer la valeur du gestionnaire dans votre code, vous pouvez utiliser la variable d'environnement `_HANDLER`.

Exécution go1.x

Pour les fonctions Go qui utilisent l'exécution `go1.x`, le fichier exécutable et le gestionnaire peuvent partager n'importe quel nom. Par exemple, si vous définissez la valeur du gestionnaire sur `Handler`, Lambda appellera la fonction `main()` dans le fichier exécutable `Handler`.

Pour modifier le nom du gestionnaire de fonction dans la console Lambda, dans le volet Runtime settings (Paramètres d'exécution), choisissez Edit (Modifier).

## Gestionnaire de fonctions Lambda utilisant des types structurés

Dans l'exemple ci-dessus, le type d'entrée était une chaîne simple. Mais vous pouvez également transmettre des événements structurés à votre gestionnaire de fonction :

```
package main
```

```
import (
 "fmt"
 "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
 Name string `json:"What is your name?"`
 Age  int    `json:"How old are you?"`
}

type MyResponse struct {
 Message string `json:"Answer"`
}

func HandleLambdaEvent(event *MyEvent) (*MyResponse, error) {
 if event == nil {
  return nil, fmt.Errorf("received nil event")
 }
 return &MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,
 event.Age)}, nil
}

func main() {
 lambda.Start(HandleLambdaEvent)
}
```

Voici un exemple d'entrée pour cette fonction :

```
{
    "What is your name?": "Jim",
    "How old are you?": 33
}
```

La réponse ressemble à ceci :

```
{
    "Answer": "Jim is 33 years old!"
}
```

Pour être exportées, les noms de champs de la structure d'événement doivent être en majuscules. Pour plus d'informations sur la gestion des événements provenant de sources d' AWS aws-lambda-goévénements, consultez /events.

## Signatures de gestionnaire valides

Vous disposez de plusieurs options lorsque vous créez un gestionnaire de fonction Lambda dans Go, mais vous devez respecter les règles suivantes :

- Le gestionnaire doit être une fonction.
- Le gestionnaire peut accepter entre 0 et 2 arguments. S'il y a deux arguments, le premier argument doit implémenter `context.Context`.
- Le gestionnaire peut retourner entre 0 et 2 arguments. S'il y a une seule valeur renvoyée, elle doit implémenter `error`. S'il y a deux valeurs renvoyées, la seconde valeur doit implémenter `error`.

Voici la liste des signatures de gestionnaire valides. `TIn` et `TOut` représentent des types compatibles avec la bibliothèque encoding/json standard. Pour en savoir plus, reportez-vous à la section func Unmarshal pour savoir comment ces types sont désérialisés.

- `func ()`
- `func () error`
- `func (TIn) error`
- `func () (TOut, error)`
- `func (context.Context) error`
- `func (context.Context, TIn) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) (TOut, error)`

## Utilisation de l'état global

Vous pouvez déclarer et modifier des variables globales indépendantes du code du gestionnaire de votre fonction Lambda. De plus, votre gestionnaire peut déclarer une fonction `init` qui est exécutée lorsque votre gestionnaire est chargé. Cela se comporte de la même manière AWS Lambda que dans les programmes Go standard. Une instance unique de votre fonction Lambda ne gère jamais plusieurs événements simultanément.

Example Une fonction Go avec des variables globales

> **Note**
>
> Ce code utilise le AWS SDK for Go V2. Pour plus d'informations, consultez Getting Started with the AWS SDK for Go V2.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "log"
)

var invokeCount int
var myObjects []types.Object

func init() {
 // Load the SDK configuration
 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Fatalf("Unable to load SDK config: %v", err)
 }

 // Initialize an S3 client
 svc := s3.NewFromConfig(cfg)

 // Define the bucket name as a variable so we can take its address
 bucketName := "amzn-s3-demo-bucket"
```

```
 input := &s3.ListObjectsV2Input{
  Bucket: &bucketName,
 }

 // List objects in the bucket
 result, err := svc.ListObjectsV2(context.TODO(), input)
 if err != nil {
  log.Fatalf("Failed to list objects: %v", err)
 }
 myObjects = result.Contents
}

func LambdaHandler(ctx context.Context) (int, error) {
 invokeCount++
 for i, obj := range myObjects {
  log.Printf("object[%d] size: %d key: %s", i, obj.Size, *obj.Key)
 }
 return invokeCount, nil
}

func main() {
 lambda.Start(LambdaHandler)
}
```

## Meilleures pratiques en matière de code pour les fonctions Go Lambda

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation.

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp`

répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.

- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.
- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Go

Lorsque Lambda exécute votre fonction, il transmet un objet contexte au gestionnaire. Cet objet fournit des méthodes et des propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

La bibliothèque de contexte Lambda fournit les variables globales, méthodes et propriétés suivantes.

Variables globales

- `FunctionName` – Nom de la fonction Lambda.
- `FunctionVersion` – Version de la fonction.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `LogGroupName` – Groupe de journaux pour la fonction.
- `LogStreamName` – Flux de journal de l'instance de fonction.

Méthodes de contexte

- `Deadline` – Retourne la date d'expiration de l'exécution, exprimée en millisecondes au format horaire Unix.

Propriétés du contexte

- `InvokedFunctionArn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `AwsRequestID` – Identifiant de la demande d'appel.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.

## Accès aux informations du contexte d'appel

Les fonctions Lambda ont accès aux métadonnées sur leur environnement et la demande d'appel. Elles sont accessibles à l'adresse du contexte du package. Si votre gestionnaire inclut

`context.Context` en tant que paramètre, Lambda insère les informations sur votre fonction dans la propriété `Value` du contexte. Notez que vous devez importer la bibliothèque `lambdacontext` pour accéder au contenu de l'objet `context.Context`.

```
package main

import (
        "context"
        "log"
        "github.com/aws/aws-lambda-go/lambda"
        "github.com/aws/aws-lambda-go/lambdacontext"
)

func CognitoHandler(ctx context.Context) {
        lc, _ := lambdacontext.FromContext(ctx)
        log.Print(lc.Identity.CognitoIdentityPoolID)
}

func main() {
        lambda.Start(CognitoHandler)
}
```

Dans l'exemple ci-dessus, `lc` c'est la variable utilisée pour consommer les informations capturées par l'objet de contexte et `log.Print(lc.Identity.CognitoIdentityPoolID)` imprimer ces informations, dans ce cas, l' CognitoIdentityPoolID.

L'exemple suivant présente la façon d'utiliser l'objet contexte pour surveiller le temps nécessaire à l'exécution de votre fonction Lambda. Cela vous permet d'analyser les attentes de performance et d'ajuster le code de votre fonction en conséquence, si nécessaire.

```
package main

import (
        "context"
        "log"
        "time"
        "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

        deadline, _ := ctx.Deadline()
```

```
        deadline = deadline.Add(-100 * time.Millisecond)
        timeoutChannel := time.After(time.Until(deadline))

        for {

                select {

                case <- timeoutChannel:
                        return "Finished before timing out.", nil

                default:
                        log.Print("hello!")
                        time.Sleep(50 * time.Millisecond)
                }
        }
}

func main() {
        lambda.Start(LongRunningHandler)
}
```

# Déployer des fonctions Lambda Go avec des archives de fichiers .zip

Le code de votre AWS Lambda fonction se compose de scripts ou de programmes compilés et de leurs dépendances. Pour déployer votre code de fonction vers Lambda, vous utilisez un package de déploiement. Lambda prend en charge deux types de packages de déploiement : les images conteneurs et les archives de fichiers .zip.

Cette page explique comment créer un fichier .zip en tant que package de déploiement pour le moteur d'exécution Go, puis utiliser le fichier .zip pour déployer votre code de fonction à AWS Lambda l'aide des AWS Management Console, AWS Command Line Interface (AWS CLI) et AWS Serverless Application Model (AWS SAM).

Notez que Lambda utilise des autorisations de POSIX fichier. Vous devrez donc peut-être définir des autorisations pour le dossier du package de déploiement avant de créer l'archive de fichiers .zip.

Sections


- Création d’un fichier .zip sur macOS et Linux
- Création d’un fichier .zip sous Windows
- Création et mise à jour de fonctions Lambda Go à l’aide de fichiers .zip


## Création d’un fichier .zip sur macOS et Linux

Les étapes suivantes montrent comment compiler votre exécutable à l’aide de la commande go `build` et créer un package de déploiement de fichiers .zip pour Lambda. Avant de compiler votre code, assurez-vous d'avoir installé le package lambda depuis. GitHub Ce module fournit une implémentation de l’interface d’exécution, qui gère l’interaction entre Lambda et le code de votre fonction. Pour télécharger cette bibliothèque, exécutez la commande suivante.

```
go get github.com/aws/aws-lambda-go/lambda
```

Si votre fonction utilise le AWS SDK for Go, téléchargez l'ensemble standard de SDK modules, ainsi que tous les API clients de AWS service requis par votre application. Pour savoir comment installer le SDK for Go, consultez Getting Started with the AWS SDK for Go V2.

## Utilisation de la famille d'exécution fournie

Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez un environnement d'exécution réservé au système d'exploitation (la famille `provided` d'exécution) pour déployer les fonctions Go sur Lambda.

Pour créer un package de déploiement .zip (macOS/Linux)

1. Dans le répertoire du projet qui contient le fichier `main.go` de votre application, compilez votre exécutable. Notez ce qui suit :

   - L'exécutable doit être nommé `bootstrap`. Pour de plus amples informations, veuillez consulter Dénomination.
   - Définissez l'architecture de jeu d'instructions cible. Les environnements d'exécution réservés au système d'exploitation prennent en charge arm64 et x86_64.
   - Vous pouvez utiliser la `lambda.norpc` balise optionnelle pour exclure le composant Remote Procedure Call (RPC) de la bibliothèque Lambda. Le RPC composant n'est requis que si vous utilisez le runtime Go 1.x obsolète. L'exclusion de RPC réduit la taille du package de déploiement.

   Pour l'architecture arm64 :

   ```
   GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
   ```

   Pour l'architecture x86_64 :

   ```
   GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
   ```

2. (Facultatif) Vous devrez peut-être compiler des packages avec `CGO_ENABLED=0` configuré sur Linux :

   ```
   GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc
    main.go
   ```

   Cette commande crée un package binaire stable pour les versions de bibliothèque C standard (`libc`), qui peuvent être différentes sur Lambda et d'autres appareils.
3. Créez un package de déploiement en empaquetant l'exécutable dans un fichier .zip.

```
zip myFunction.zip bootstrap
```

> **Note**
>
> Le fichier `bootstrap` doit se situer à la racine du fichier .zip.

4. Créez la fonction . Notez ce qui suit :

   - Le binaire doit être nommé `bootstrap`, mais le nom du gestionnaire peut être quelconque. Pour de plus amples informations, veuillez consulter Dénomination.
   - L'option `--architectures` n'est nécessaire que si vous utilisez arm64. La valeur par défaut est x86_64.
   - Pour`--role`, spécifiez le nom de ressource Amazon (ARN) du rôle d'exécution.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--architectures arm64 \
--role arn:aws:iam::111122223333:role/lambda-ex \
--zip-file fileb://myFunction.zip
```

## Création d'un fichier .zip sous Windows

Les étapes suivantes indiquent comment télécharger l'build-lambda-zipoutil pour Windows depuis GitHub, compiler votre exécutable et créer un package de déploiement .zip.

> **Note**
>
> Si vous ne l'avez pas déjà fait, vous devez installer git, puis ajouter l'exécutable `git` à votre variable d'environnement Windows `%PATH%`.

Avant de compiler votre code, assurez-vous d'avoir installé la bibliothèque Lambda depuis. GitHub Pour télécharger cette bibliothèque, exécutez la commande suivante.

```
go get github.com/aws/aws-lambda-go/lambda
```

Si votre fonction utilise le AWS SDK for Go, téléchargez l'ensemble standard de SDK modules, ainsi que tous les API clients de AWS service requis par votre application. Pour savoir comment installer le SDK for Go, consultez Getting Started with the AWS SDK for Go V2.

## Utilisation de la famille d'exécution fournie

Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez un environnement d'exécution réservé au système d'exploitation (la famille `provided` d'exécution) pour déployer les fonctions Go sur Lambda.

Pour créer un package de déploiement .zip (Windows)

1. Téléchargez l'build-lambda-zipoutil depuis GitHub.

   ```
   go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
   ```

2. Utilisez l'outil de votre `GOPATH` pour créer un fichier .zip. Si vous disposez d'une installation de Go par défaut, l'outil est généralement dans `%USERPROFILE%\Go\bin`. Dans le cas contraire, accédez à l'endroit où vous avez installé l'environnement d'exécution Go et effectuez l'une des instructions suivantes :

   cmd.exe

   Dans cmd.exe, exécutez l'une des commandes suivantes, en fonction de l'architecture de jeu d'instructions cible. Les environnements d'exécution réservés au système d'exploitation prennent en charge arm64 et x86_64.

   Vous pouvez utiliser la `lambda.norpc` balise optionnelle pour exclure le composant Remote Procedure Call (RPC) de la bibliothèque Lambda. Le RPC composant n'est requis que si vous utilisez le runtime Go 1.x obsolète. L'exclusion de RPC réduit la taille du package de déploiement.

   Example — Pour l'architecture x86_64

   ```
   set GOOS=linux
   set GOARCH=amd64
   set CGO_ENABLED=0
   go build -tags lambda.norpc -o bootstrap main.go
   %USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
   ```

Example — Pour l'architecture arm64

```
set GOOS=linux
set GOARCH=arm64
set CGO_ENABLED=0
go build -tags lambda.norpc -o bootstrap main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

PowerShell

Dans PowerShell, exécutez l'une des opérations suivantes, en fonction de l'architecture de votre jeu d'instructions cible. Les environnements d'exécution réservés au système d'exploitation prennent en charge arm64 et x86_64.

Vous pouvez utiliser la `lambda.norpc` balise optionnelle pour exclure le composant Remote Procedure Call (RPC) de la bibliothèque Lambda. Le RPC composant n'est requis que si vous utilisez le runtime Go 1.x obsolète. L'exclusion de RPC réduit la taille du package de déploiement.

Pour l'architecture x86_64 :

```
$env:GOOS = "linux"
$env:GOARCH = "amd64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

Pour l'architecture arm64 :

```
$env:GOOS = "linux"
$env:GOARCH = "arm64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

3. Créez la fonction . Notez ce qui suit :

- Le binaire doit être nommé `bootstrap`, mais le nom du gestionnaire peut être quelconque. Pour de plus amples informations, veuillez consulter Dénomination.

- L'option `--architectures` n'est nécessaire que si vous utilisez arm64. La valeur par défaut est x86_64.
- Pour`--role`, spécifiez le nom de ressource Amazon (ARN) du rôle d'exécution.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--architectures arm64 \
--role arn:aws:iam::111122223333:role/lambda-ex \
--zip-file fileb://myFunction.zip
```

## Création et mise à jour de fonctions Lambda Go à l'aide de fichiers .zip

Après avoir créé votre package de déploiement .zip, vous pouvez l'utiliser pour créer une nouvelle fonction Lambda ou mettre à jour une fonction existante. Vous pouvez déployer votre package .zip à l'aide de la console Lambda, AWS Command Line Interface du, et du Lambda. API Vous pouvez également créer et mettre à jour des fonctions Lambda à l'aide de l' AWS Serverless Application Model (AWS SAM) et de AWS CloudFormation.

La taille maximale d'un package de déploiement .zip pour Lambda est de 250 Mo (décompressé). Notez que cette limite s'applique à la taille combinée de tous les fichiers que vous chargez, y compris les couches Lambda.

Le runtime Lambda a besoin d'une autorisation pour lire les fichiers de votre package de déploiement. Dans la notation octale des autorisations Linux, Lambda a besoin de 644 autorisations pour les fichiers non exécutables (rw-r--r--) et de 755 autorisations () pour les répertoires et les fichiers exécutables. rwxr-xr-x

Sous Linux et macOS, utilisez la commande `chmod` pour modifier les autorisations de fichiers sur les fichiers et les répertoires de votre package de déploiement. Par exemple, pour accorder les autorisations appropriées à un fichier exécutable, exécutez la commande suivante.

```
chmod 755 <filepath>
```

Pour modifier les autorisations relatives aux fichiers dans Windows, voir Définir, afficher, modifier ou supprimer des autorisations sur un objet dans la documentation Microsoft Windows.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de la console

Pour créer une nouvelle fonction, vous devez d'abord créer la fonction dans la console, puis charger votre archive .zip. Pour mettre à jour une fonction existante, ouvrez la page de votre fonction, puis suivez la même procédure pour ajouter votre fichier .zip mis à jour.

Si votre fichier .zip fait moins de 50 Mo, vous pouvez créer ou mettre à jour une fonction en chargeant le fichier directement à partir de votre ordinateur local. Pour les fichiers .zip de plus de 50 Mo, vous devez d'abord charger votre package dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS Management Console, consultez Getting started with Amazon S3. Pour télécharger des fichiers à l'aide de AWS CLI, voir Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Vous ne pouvez pas convertir une fonction d'image de conteneur existante pour utiliser une archive .zip. Vous devez créer une nouvelle fonction.

Pour créer une nouvelle fonction (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez Créer une fonction.
2. Choisissez Créer à partir de zéro.
3. Sous Informations de base, procédez comme suit :
   a. Pour Nom de la fonction, saisissez le nom de la fonction.
   b. Dans Runtime (Exécution), choisissez `provided.al2023`.
4. (Facultatif) Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Vous pouvez créer un rôle d'exécution ou en utiliser un existant.
5. Sélectionnez Create function (Créer une fonction). Lambda crée une fonction de base « Hello world » à l'aide de l'exécution de votre choix.

Pour charger une archive .zip à partir de votre ordinateur local (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger le fichier .zip.

2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez Fichier .zip.
5. Pour charger un fichier .zip, procédez comme suit :

    a. Sélectionnez Charger, puis choisissez votre fichier .zip dans le sélecteur de fichiers.
    b. Choisissez Ouvrir.
    c. Choisissez Save (Enregistrer).

Pour charger une archive .zip depuis un compartiment Amazon S3 (console)

1. Sur la page Fonctions de la console Lambda, choisissez la fonction pour laquelle vous souhaitez charger un nouveau fichier .zip.
2. Sélectionnez l'onglet Code.
3. Dans le volet Source du code, choisissez Charger à partir de.
4. Choisissez l'emplacement Amazon S3.
5. Collez le lien Amazon S3 URL de votre fichier .zip et choisissez Enregistrer.

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide du AWS CLI

Vous pouvez utiliser l'AWS CLI pour créer une nouvelle fonction ou pour mettre à jour une fonction existante à l'aide d'un fichier .zip. Utilisez la fonction de création et update-function-codeles commandes pour déployer votre package .zip. Si votre fichier .zip est inférieur à 50 Mo, vous pouvez charger le package .zip à partir d'un emplacement de fichier sur votre machine de génération locale. Pour les fichiers plus volumineux, vous devez charger votre package .zip à partir d'un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

> **Note**
>
> Si vous chargez votre fichier .zip depuis un compartiment Amazon S3 à l'aide de AWS CLI, le compartiment doit se trouver au même endroit Région AWS que votre fonction.

Pour créer une nouvelle fonction à l'aide d'un fichier .zip avec le AWS CLI, vous devez spécifier les éléments suivants :

- Le nom de votre fonction (`--function-name`)
- L'exécution de votre fonction (`--runtime`)
- Le nom de ressource Amazon (ARN) du rôle d'exécution de votre fonction (`--role`)
- Le nom de la méthode du gestionnaire dans votre code de fonction (`--handler`)

Vous devez également indiquer l'emplacement de votre fichier .zip. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez l'option `--code` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `S3ObjectVersion` pour les objets soumis à la gestion des versions.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Pour mettre à jour une fonction existante à l'aide duCLI, vous devez spécifier le nom de votre fonction à l'aide du `--function-name` paramètre. Vous devez également spécifier l'emplacement du fichier .zip que vous souhaitez utiliser pour mettre à jour votre code de fonction. Si votre fichier .zip se trouve dans un dossier sur votre machine de génération locale, utilisez l'option `--zip-file` pour spécifier le chemin d'accès du fichier, comme le montre l'exemple de commande suivant.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Pour spécifier l'emplacement du fichier .zip dans un compartiment Amazon S3, utilisez les options `--s3-bucket` et `--s3-key` comme le montre l'exemple de commande suivant. Vous devez uniquement utiliser le paramètre `--s3-object-version` pour les objets soumis à la gestion des versions.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide du Lambda API

Pour créer et mettre à jour des fonctions à l'aide d'une archive de fichiers .zip, effectuez les API opérations suivantes :

- CreateFunction
- UpdateFunctionCode

### Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS SAM

The AWS Serverless Application Model (AWS SAM) est une boîte à outils qui permet de rationaliser le processus de création et d'exécution d'applications sans serveur sur AWS. Vous définissez les ressources de votre application dans un JSON modèle YAML or et vous utilisez l'interface de ligne de AWS SAM commande (AWS SAM CLI) pour créer, empaqueter et déployer vos applications. Lorsque vous créez une fonction Lambda à partir d'un AWS SAM modèle, elle crée AWS SAM automatiquement un package de déploiement ou une image de conteneur .zip avec le code de votre fonction et les dépendances que vous spécifiez. Pour en savoir plus sur l'utilisation des fonctions Lambda AWS SAM pour créer et déployer des fonctions Lambda, consultez la section Getting started with AWS SAM dans le Guide du AWS Serverless Application Model développeur.

Vous pouvez également l'utiliser AWS SAM pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip existante. Pour créer une fonction Lambda à l'aide de AWS SAM, vous pouvez enregistrer votre fichier .zip dans un compartiment Amazon S3 ou dans un dossier local sur votre machine de génération. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS SAM modèle, la `AWS::Serverless::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l’aide d’une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `CodeUri`- défini sur l'Amazon S3URI, le chemin d'accès au dossier local ou à l'FunctionCodeobjet du code de fonction
- `Runtime` : défini sur votre exécution choisie

Ainsi AWS SAM, si votre fichier .zip est supérieur à 50 Mo, vous n'avez pas besoin de le télécharger au préalable dans un compartiment Amazon S3. AWS SAM peut télécharger des packages .zip jusqu'à la taille maximale autorisée de 250 Mo (décompressés) à partir d'un emplacement sur votre machine de compilation locale.

Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS SAM, voir :Serverless AWS : :Function dans le manuel du développeur.AWS SAM

Exemple : utilisation AWS SAM pour créer une fonction Go avec provided.al2023

1. Créez un AWS SAM modèle avec les propriétés suivantes :

   - BuildMethod: Spécifie le compilateur de votre application. Utilisez `go1.x`.
   - Runtime : utilisez `provided.al2023`.
   - CodeUri: Entrez le chemin d'accès à votre code.
   - Architectures : Utilisez `[arm64]` pour l'architecture arm64. Pour l'architecture x86_64, utilisez `[amd64]` ou supprimez la propriété `Architectures`.

   Example template.yaml

   ```
   AWSTemplateFormatVersion: '2010-09-09'
   Transform: 'AWS::Serverless-2016-10-31'
   Resources:
     HelloWorldFunction:
       Type: AWS::Serverless::Function
       Metadata:
         BuildMethod: go1.x
       Properties:
         CodeUri: hello-world/ # folder where your main program resides
         Handler: bootstrap
         Runtime: provided.al2023
         Architectures: [arm64]
   ```

2. Utilisez la commande sam build pour compiler l'exécutable.

   ```
   sam build
   ```

3. Utilisez la commande sam deploy pour déployer la fonction vers Lambda.

```
sam deploy --guided
```

## Création et mise à jour de fonctions avec des fichiers .zip à l'aide de AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une fonction Lambda à l'aide d'une archive de fichiers .zip. Pour créer une fonction Lambda à partir d'un fichier .zip, vous devez d'abord charger votre fichier dans un compartiment Amazon S3. Pour savoir comment charger un fichier dans un compartiment Amazon S3 à l'aide du AWS CLI, consultez la section Déplacer des objets dans le guide de AWS CLI l'utilisateur.

Dans votre AWS CloudFormation modèle, la `AWS::Lambda::Function` ressource spécifie votre fonction Lambda. Dans cette ressource, définissez les propriétés suivantes pour créer une fonction à l'aide d'une archive de fichiers .zip :

- `PackageType` : défini sur `Zip`
- `Code` : saisissez le nom du compartiment Amazon S3 et le nom du fichier .zip dans les champs `S3Bucket` et `S3Key`
- `Runtime` : défini sur votre exécution choisie

Le fichier .zip AWS CloudFormation généré ne peut pas dépasser 4 Mo. Pour en savoir plus sur le déploiement de fonctions à l'aide d'un fichier .zip dans AWS CloudFormation, voir :Lambda AWS : :Function dans le guide de l'utilisateur.AWS CloudFormation

# Déployer des fonctions Lambda Go avec des images conteneurs

Il existe deux manières de créer une image de conteneur pour une fonction Go Lambda :

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez une image de base réservée au système d'exploitation pour créer des images Go pour Lambda. Pour rendre l'image compatible avec Lambda, vous devez inclure le package `aws-lambda-go/lambda` dans l'image.
- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le package `aws-lambda-go/lambda` dans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

## AWS images de base pour le déploiement des fonctions Go

Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez une image de base réservée au système d'exploitation pour déployer les fonctions Go sur Lambda.

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| Exécution réservée au système d'exploitation | `provided.al2023` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| Exécution réservée au système d'exploitation | `provided.al2` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |

Galerie publique Amazon Elastic Container Registry : gallery.ecr.aws/lambda/fourni

## Client d'interface d'environnement d'exécution Go

Le package `aws-lambda-go/lambda` inclut une implémentation de l'interface d'exécution. Pour obtenir des exemples d'utilisation de `aws-lambda-go/lambda` dans votre image, consultez Utilisation d'une image de base AWS uniquement pour le système d'exploitation ou Utilisation d'une image non AWS basique.

## Utilisation d'une image de base AWS uniquement pour le système d'exploitation

Go est implémenté différemment des autres exécutions gérées. Comme Go compile nativement en un binaire exécutable, il ne nécessite pas d'environnement d'exécution de langage dédié. Utilisez une image de base réservée au système d'exploitation pour créer des images de conteneur pour les fonctions Go.

| Balises | Environnement d'exécution | Système d'exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| al2023 | Exécution uniquement basée sur le système d'exploitation | Amazon Linux | Dockerfile pour OS uniquement Runtime sur GitHub | Non prévu |
| al2 | Exécution uniquement basée sur le système d'exploitation | Amazon Linux 2 | Dockerfile pour OS uniquement Runtime sur GitHub | Non prévu |

Pour plus d'informations sur ces images de base, consultez la galerie ECR publique d'Amazon.

Vous devez inclure le package aws-lambda-go/lambda dans votre gestionnaire Go. Ce package implémente le modèle de programmation pour Go, y compris l'interface d'exécution.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Go
- Docker
- AWS Command Line Interface (AWS CLI) version 2

Création d'une image à partir de l'image de base provided.al2023

Pour créer et déployer une fonction Go avec l'image de base **`provided.al2023`**

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir hello
cd hello
```

2. Initialisez un nouveau module Go.

```
go mod init example.com/hello-world
```

3. Ajoutez la bibliothèque lambda comme dépendance de votre nouveau module.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Créez un fichier nommé `main.go` et ouvrez-le dans un éditeur de texte. Il s'agit du code de la fonction Lambda. Vous pouvez utiliser l'exemple de code suivant pour le tester, ou le remplacer par le vôtre.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
 (events.APIGatewayProxyResponse, error) {
 response := events.APIGatewayProxyResponse{
  StatusCode: 200,
  Body:       "\"Hello from Lambda!\"",
 }
 return response, nil
}

func main() {
 lambda.Start(handler)
}
```

5. Utilisez un éditeur de texte afin de créer un Dockerfile dans votre répertoire de projets.

   - L'exemple de fichier Docker suivant utilise une génération en plusieurs étapes. Cela vous permet d'utiliser une image de base différente à chaque étape. Vous pouvez utiliser une image, telle qu'une image de base Go, pour compiler votre code et générer le binaire

exécutable. Vous pouvez ensuite utiliser une image différente, telle que `provided.al2023`, dans l'instruction finale `FROM` pour définir l'image que vous déployez sur Lambda. Le processus de génération est séparé de l'image de déploiement final, de sorte que l'image finale ne contient que les fichiers nécessaires à l'exécution de l'application.

- Vous pouvez utiliser la `lambda.norpc` balise optionnelle pour exclure le composant Remote Procedure Call (RPC) de la bibliothèque Lambda. Le RPC composant n'est requis que si vous utilisez le runtime Go 1.x obsolète. L'exclusion de RPC réduit la taille du package de déploiement.
- Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

Example – Dockerfile de génération en plusieurs étapes

> **Note**
>
> Assurez-vous que la version de Go que vous spécifiez dans votre Dockerfile (par exemple, `golang:1.20`) est la même que celle que vous avez utilisée pour créer votre application.

```
FROM golang:1.20 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build with optional lambda.norpc tag
COPY main.go .
RUN go build -tags lambda.norpc -o main main.go
# Copy artifacts to a clean image
FROM public.ecr.aws/lambda/provided:al2023
COPY --from=build /helloworld/main ./main
ENTRYPOINT [ "./main" ]
```

6. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester votre image localement. L'émulateur d'interface d'exécution est inclus dans l'image de base `provided.al2023`.

Pour exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Démarrez votre image Docker à l'aide de la commande docker run. Notez ce qui suit :

   - `docker-image` est le nom de l'image et `test` est la balise.
   - `./main` est le `ENTRYPOINT` de votre Dockerfile.

   ```
   docker run -d -p 9000:8080 \
   --entrypoint /usr/local/bin/aws-lambda-rie \
   docker-image:test ./main
   ```

   Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.
2. À partir d'une nouvelle fenêtre de terminal, publiez un événement sur le point de terminaison suivant au moyen d'une commande curl :

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Certaines fonctions peuvent nécessiter une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

3. Obtenez l'ID du conteneur.

```
docker ps
```

4. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.
   - Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
   - `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.
4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemple :

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour `ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin du URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image dans Amazon ECR reste la même. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilisation d'une image non AWS basique

Vous pouvez créer une image de conteneur pour Go à partir d'une image non AWS basique. L'exemple de Dockerfile dans les étapes suivantes utilise une image de base Alpine.

Vous devez inclure le package aws-lambda-go/lambda dans votre gestionnaire Go. Ce package implémente le modèle de programmation pour Go, y compris l'interface d'exécution.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- Go
- Docker
- AWS Command Line Interface (AWS CLI) version 2

Création d'une image à partir d'une image de base alternative

Pour créer et déployer une fonction Go avec une image de base Alpine

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

   ```
   mkdir hello
   cd hello
   ```

2. Initialisez un nouveau module Go.

```
go mod init example.com/hello-world
```

3. Ajoutez la bibliothèque lambda comme dépendance de votre nouveau module.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Créez un fichier nommé `main.go` et ouvrez-le dans un éditeur de texte. Il s'agit du code de la fonction Lambda. Vous pouvez utiliser l'exemple de code suivant pour le tester, ou le remplacer par le vôtre.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
 (events.APIGatewayProxyResponse, error) {
 response := events.APIGatewayProxyResponse{
  StatusCode: 200,
  Body:       "\"Hello from Lambda!\"",
 }
 return response, nil
}

func main() {
 lambda.Start(handler)
}
```

5. Utilisez un éditeur de texte afin de créer un Dockerfile dans votre répertoire de projets. L'exemple de Dockerfile suivant utilise une image de base Alpine. Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune USER instruction n'est fournie.

Example Dockerfile

> **Note**
>
> Assurez-vous que la version de Go que vous spécifiez dans votre Dockerfile (par exemple, `golang:1.20`) est la même que celle que vous avez utilisée pour créer votre application.

```
FROM golang:1.20.2-alpine3.16 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build
COPY main.go .
RUN go build -o main main.go
# Copy artifacts to a clean image
FROM alpine:3.16
COPY --from=build /helloworld/main /main
ENTRYPOINT [ "/main" ]
```

6. Générez l'image Docker à l'aide de la commande docker build. L'exemple suivant nomme l'image `docker-image` et lui donne la balise `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> La commande spécifie l'option `--platform linux/amd64` pour garantir la compatibilité de votre conteneur avec l'environnement d'exécution Lambda, quelle que soit l'architecture de votre machine de génération. Si vous avez l'intention de créer une fonction Lambda à l'aide de l'architecture du jeu ARM64 d'instructions, veillez à modifier la commande pour utiliser l'`--platform linux/arm64`option à la place.

(Facultatif) Testez l'image localement

Utilisez l'émulateur d'interface d'exécution pour tester l'image localement. Vous pouvez intégrer l'émulateur à votre image ou utiliser la procédure suivante pour l'installer sur votre machine locale.

Pour installer et exécuter l'émulateur d'interface d'exécution sur votre ordinateur local

1. Depuis le répertoire de votre projet, exécutez la commande suivante pour télécharger l'émulateur d'interface d'exécution (architecture x86-64) GitHub et l'installer sur votre machine locale.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Pour installer l'émulateur arm64, remplacez le GitHub référentiel de URL la commande précédente par le suivant :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }

   $downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie"
   $destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
   Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
   ```

   Pour installer l'émulateur arm64, remplacez `$downloadLink` par ce qui suit :

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

2. Démarrez votre image Docker à l'aide de la commande docker run. Notez ce qui suit :

- `docker-image` est le nom de l'image et `test` est la balise.
- `/main` est le `ENTRYPOINT` de votre Dockerfile.

Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
 9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    --read-only \
    docker-image:test \
        /main
```

PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
 9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
--read-only `
docker-image:test `
    /main
```

Cette commande exécute l'image en tant que conteneur et crée un point de terminaison local à `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Si vous avez créé l'image Docker pour l'architecture du jeu ARM64 d'instructions, veillez à utiliser l'`--platform linux/arm64`option au lieu de`--platform linux/amd64`.

3. Publiez un événement au point de terminaison local.

Linux/macOS

Sous Linux et macOS, exécutez la commande `curl` suivante :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dans PowerShell, exécutez la `Invoke-WebRequest` commande suivante :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Cette commande invoque la fonction avec un événement vide et renvoie une réponse. Si vous utilisez votre propre code de fonction plutôt que l'exemple de code de fonction, vous souhaiterez peut-être invoquer la fonction avec une JSON charge utile. Exemple :

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenez l'ID du conteneur.

```
docker ps
```

5. Utilisez la commande docker kill pour arrêter le conteneur. Dans cette commande, remplacez `3766c4ab331c` par l'ID du conteneur de l'étape précédente.

```
docker kill 3766c4ab331c
```

Déploiement de l'image

Pour télécharger l'image sur Amazon ECR et créer la fonction Lambda

1. Exécutez la get-login-passwordcommande pour authentifier le Docker auprès CLI de votre registre AmazonECR.

- Définissez la `--region` valeur à l' Région AWS endroit où vous souhaitez créer le ECR référentiel Amazon.
- `111122223333`Remplacez-le par votre Compte AWS identifiant.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Créez un référentiel dans Amazon à ECR l'aide de la commande create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Le ECR référentiel Amazon doit être Région AWS identique à la fonction Lambda.

En cas de succès, vous obtenez une réponse comme celle-ci :

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Copiez le `repositoryUri` à partir de la sortie de l'étape précédente.

4. Exécutez la commande docker tag pour étiqueter votre image locale dans votre ECR référentiel Amazon en tant que dernière version. Dans cette commande :

   - `docker-image:test`est le nom et le tag de votre image Docker. Il s'agit du nom et de la balise de l'image que vous avez spécifiés dans la `docker build` commande.
   - Remplacez <ECRrepositoryUri> par l'`repositoryUri` que vous avez copié. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemple :

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

5. Exécutez la commande docker push pour déployer votre image locale dans le référentiel AmazonECR. Assurez-vous de l'inclure `:latest` à la fin du référentielURI.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Créez un rôle d'exécution pour la fonction, si vous n'en avez pas déjà un. Vous aurez besoin du nom de ressource Amazon (ARN) du rôle à l'étape suivante.
7. Créez la fonction Lambda. Pour`ImageUri`, spécifiez le référentiel URI antérieur. Assurez-vous d'inclure `:latest` à la fin duURI.

   ```
   aws lambda create-function \
     --function-name hello-world \
     --package-type Image \
     --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
     --role arn:aws:iam::111122223333:role/lambda-ex
   ```

   > **Note**
   >
   > Vous pouvez créer une fonction en utilisant une image d'un autre AWS compte, à condition que l'image se trouve dans la même région que la fonction Lambda. Pour de plus amples informations, veuillez consulter Autorisations ECR multi-comptes Amazon.

8. Invoquer la fonction.

```
aws lambda invoke --function-name hello-world response.json
```

Vous devriez obtenir une réponse comme celle-ci :

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Pour voir la sortie de la fonction, consultez le fichier `response.json`.

Pour mettre à jour le code de fonction, vous devez créer à nouveau l'image, télécharger la nouvelle image dans le ECR référentiel Amazon, puis utiliser la update-function-codecommande pour déployer l'image sur la fonction Lambda.

Lambda résout la balise d'image en un condensé d'image spécifique. Cela signifie que si vous pointez la balise d'image utilisée pour déployer la fonction vers une nouvelle image dans AmazonECR, Lambda ne met pas automatiquement à jour la fonction pour utiliser la nouvelle image.

Pour déployer la nouvelle image sur la même fonction Lambda, vous devez utiliser la update-function-codecommande, même si la balise d'image dans Amazon ECR reste la même. Dans l'exemple suivant, l'`--publish`option crée une nouvelle version de la fonction à l'aide de l'image du conteneur mise à jour.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

# Utilisation de couches pour les fonctions Go Lambda

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration. La création d'une couche implique trois étapes générales :

1. Package le contenu de votre couche. Cela implique de créer une archive de fichiers .zip contenant les dépendances que vous souhaitez utiliser dans vos fonctions.
2. Créez la couche dans Lambda.
3. Ajoutez la couche à vos fonctions.

Nous déconseillons d'utiliser des couches pour gérer les dépendances des fonctions Lambda écrites en Go. Cela est dû au fait que les fonctions Lambda de Go sont compilées en un seul exécutable, que vous fournissez à Lambda lorsque vous déployez votre fonction. Cet exécutable contient votre code de fonction compilé, ainsi que toutes ses dépendances. L'utilisation de couches complique non seulement ce processus, mais entraîne également une augmentation des temps de démarrage à froid, car vos fonctions doivent charger manuellement des assemblages supplémentaires en mémoire pendant la phase d'initialisation.

Pour utiliser des dépendances externes avec vos gestionnaires Go, incluez-les directement dans votre package de déploiement. Ce faisant, vous simplifiez le processus de déploiement et profitez des optimisations du compilateur Go intégré. Pour un exemple de la façon d'importer et d'utiliser une dépendance telle que AWS SDK for Go dans votre fonction, consultezthe section called “Handler (Gestionnaire)”.

# Enregistrez et surveillez les fonctions Go Lambda

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et envoie les journaux à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des informations sur chaque invocation au flux de journaux et transmet les journaux et autres sorties provenant du code de votre fonction. Pour de plus amples informations, veuillez consulter Utilisation des CloudWatch journaux et journaux avec Lambda.

Cette page explique comment générer une sortie de journal à partir du code de votre fonction Lambda et comment accéder aux journaux à l'aide de la AWS Command Line Interface console Lambda ou de la console. CloudWatch

Sections


- Création d'une fonction qui renvoie des journaux
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux


## Création d'une fonction qui renvoie des journaux

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur le package fmt ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`. L'exemple suivant utilise le package de journal.

Example main.go – journalisation

```
func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
  // event
  eventJson, _ := json.MarshalIndent(event, "", "  ")
  log.Printf("EVENT: %s", eventJson)
  // environment variables
  log.Printf("REGION: %s", os.Getenv("AWS_REGION"))
  log.Println("ALL ENV VARS:")
  for _, element := range os.Environ() {
    log.Println(element)
```

```
  }
```

Example format des journaux

```
START RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Version: $LATEST
2020/03/27 03:40:05 EVENT: {
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "md5OfBody": "7b27xmplb47ff90a553787216d55d91d",
      "md5OfMessageAttributes": "",
      "attributes": {
        "ApproximateFirstReceiveTimestamp": "1523232000001",
        "ApproximateReceiveCount": "1",
        "SenderId": "123456789012",
        "SentTimestamp": "1523232000000"
      },
      ...
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMPL6XBC
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$LATEST
2020/03/27 03:40:05 AWS_EXECUTION_ENV=AWS_Lambda_go1.x
END RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71
REPORT RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Duration: 38.66 ms Billed
 Duration: 39 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled:
 true
```

L'environnement d'exécution Go enregistre les lignes `START`, `END` et `REPORT` pour chaque invocation. La ligne de rapport fournit les détails suivants.

`REPORT`champs de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.

- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled – Pour les demandes suivies, résultat de l'échantillonnage.

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

# Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-`

`in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
```

```
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

# Instrumentation du code Go AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des deux SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- AWS X-Ray SDK for Go — Un outil SDK pour générer et envoyer des données de suivi à X-Ray.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections


- Utilisation ADOT pour instrumenter les fonctions de votre Go
- Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Go
- Activation du suivi avec la console Lambda
- Activation du traçage avec le Lambda API
- Activation du traçage avec AWS CloudFormation
- Interprétation d'un suivi X-Ray

## Utilisation ADOT pour instrumenter les fonctions de votre Go

ADOTfournit des couches Lambda entièrement gérées qui regroupent tout ce dont vous avez besoin pour collecter des données de télémétrie à l'aide du. OTel SDK En consommant cette couche, vous pouvez instrumenter vos fonctions Lambda sans avoir à modifier le code de fonction. Vous pouvez également configurer votre couche pour effectuer une initialisation personnalisée deOTel. Pour plus d'informations, consultez la section Configuration personnalisée du ADOT collecteur sur Lambda dans la ADOT documentation.

Pour les environnements d'exécution Go, vous pouvez ajouter la couche Lambda AWS gérée ADOT pour que Go instrumente automatiquement vos fonctions. Pour obtenir des instructions détaillées sur la façon d'ajouter cette couche, consultez AWS Distro for OpenTelemetry Lambda Support for Go ADOT dans la documentation.

## Utiliser le X-Ray SDK pour instrumenter les fonctions de votre Go

Pour enregistrer les détails des appels que votre fonction Lambda effectue à d'autres ressources de votre application, vous pouvez également utiliser le AWS X-Ray SDK for Go. Pour l'obtenirSDK, téléchargez-le SDK depuis son GitHub dépôt avec `go get` :

```
go get github.com/aws/aws-xray-sdk-go
```

Pour AWS SDK instrumenter les clients, transmettez-les à la `xray.AWS()` méthode. Ensuite, vous pouvez suivre vos appels à l'aide de la version `WithContext` de la méthode.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Après avoir ajouté les dépendances correctes et apporté les modifications de code nécessaires, activez le suivi dans la configuration de votre fonction via la console Lambda ou le. API

## Activation du suivi avec la console Lambda

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

## Activation du traçage avec le Lambda API

Configurez le suivi sur votre fonction Lambda avec AWS CLI ou utilisez AWS SDK les opérations suivantes : API

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant active le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du traçage avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
```

```
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

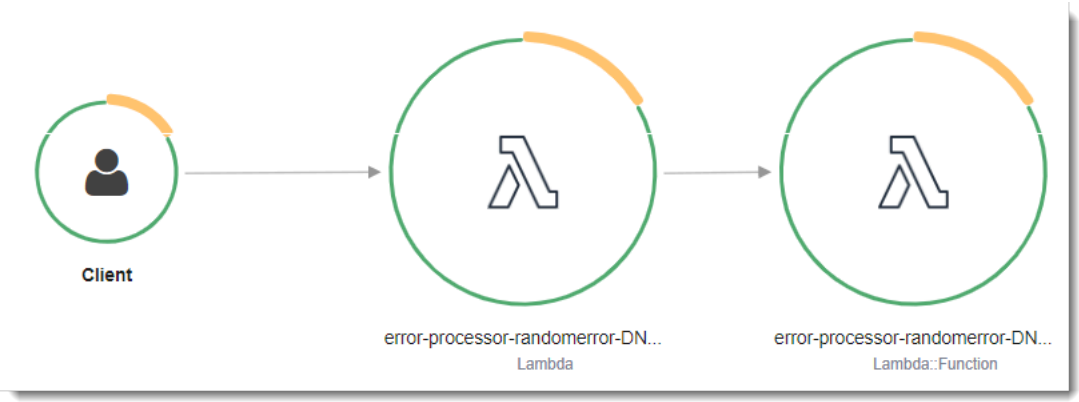
## Interprétation d'un suivi X-Ray

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Dans le cas contraire, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

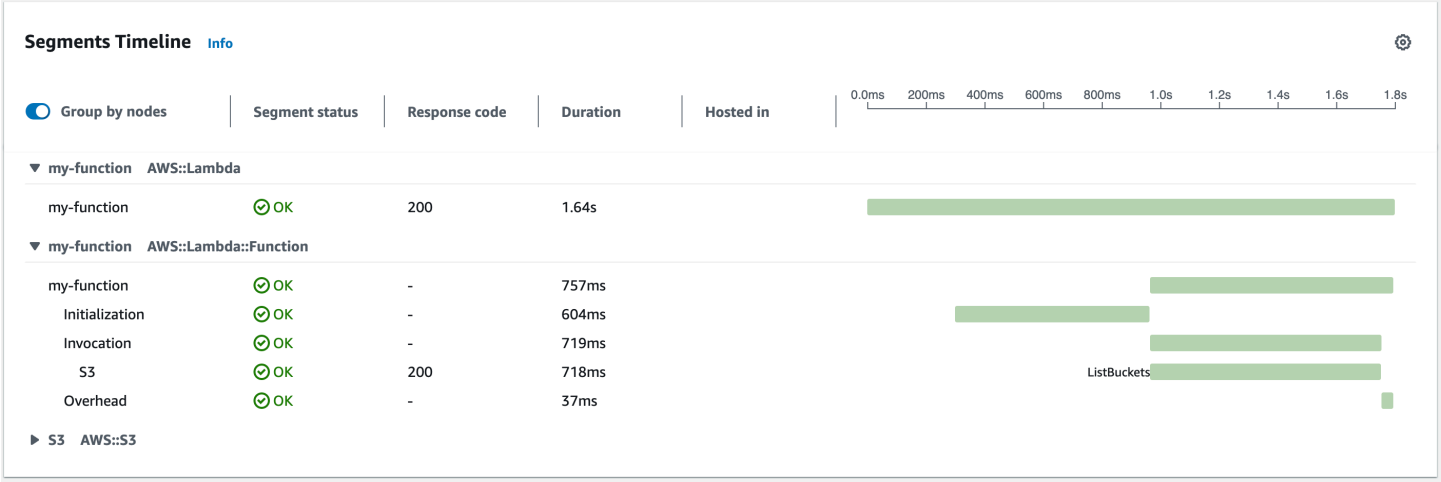
X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le

`AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.




Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez le guide du AWS X-Ray développeur AWS X-Ray SDK pour Go.

> **Tarification**
>
> Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

# Création de fonctions Lambda avec C#

Vous pouvez gérer votre. NETapplication dans Lambda utilisant le managé. NET6 ou. NET8 environnements d'exécution, un environnement d'exécution personnalisé ou une image de conteneur. Une fois le code de votre application compilé, vous pouvez le déployer dans Lambda sous la forme d'un fichier .zip ou d'une image de conteneur. Lambda fournit les environnements d'exécution suivants pour. NETlangues :

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| . NET8 | `dotnet8` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| . NET6 | `dotnet6` | Amazon Linux 2 | 20 déc. 2024 | 28 février 2025 | 31 mars 2025 |

## Configuration de votre. NETenvironnement de développement

Pour développer et créer vos fonctions Lambda, vous pouvez utiliser n'importe laquelle des fonctions couramment disponibles. NETenvironnements de développement intégrés (IDEs), notamment Microsoft Visual Studio, Visual Studio Code et JetBrains Rider. Pour simplifier votre expérience de développement, AWS fournit un ensemble de. NETles modèles de projet, ainsi que l'interface de ligne de `Amazon.Lambda.Tools` commande (CLI).

Exécutez ce qui suit. NETCLIcommandes pour installer ces modèles de projet et ces outils de ligne de commande.

### Installation du. NETmodèles de projets

Pour installer les modèles de projet (. NET8) :

```
dotnet new install Amazon.Lambda.Templates
```

Pour installer les modèles de projet (. NET6) :

```
dotnet new --install Amazon.Lambda.Templates
```

> **Note**
>
> Si vous utilisez le. NET6 environnements d'exécution Lambda gérés, nous vous recommandons de les mettre à niveau pour les utiliser. NET8. Pour en savoir plus, consultez les sections Gestion des mises à niveau AWS Lambda d'exécution et Présentation du. NET8 runtime pour AWS Lambda on the AWS Compute Blog.

## Installation et mise à jour des CLI outils

Exécutez les commandes suivantes pour installer, mettre à jour et désinstaller le `Amazon.Lambda.Tools`CLI.

Pour installer les outils de ligne de commande :

```
dotnet tool install -g Amazon.Lambda.Tools
```

Pour mettre à jour les outils de ligne de commande :

```
dotnet tool update -g Amazon.Lambda.Tools
```

Pour désinstaller les outils de ligne de commande :

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

# Définir le gestionnaire de fonctions Lambda en C#

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Lorsque votre fonction est invoquée et que Lambda exécute la méthode du gestionnaire de votre fonction, celle-ci transmet deux arguments à votre fonction. Le premier argument est l'objet `event`. Lorsqu'une autre Service AWS personne appelle votre fonction, l'`event`objet contient des données relatives à l'événement à l'origine de l'appel de votre fonction. Par exemple, un `event` objet de API Gateway contient des informations sur le chemin, la HTTP méthode et HTTP les en-têtes. La structure exacte de l'événement varie en fonction de l' Service AWS appel de votre fonction. Consultez Intégration d'autres services pour obtenir pour en savoir plus sur les formats d'événements des différents services.

Lambda transmet également un objet de `context` à votre fonction. Cet objet contient des informations sur l'invocation, la fonction et l'environnement d'exécution. Pour de plus amples informations, veuillez consulter the section called "Contexte".

Le format natif pour tous les événements Lambda est constitué de flux d'octets représentant l'événement JSON formaté. À moins que les paramètres d'entrée et de sortie de votre fonction ne soient de type`System.IO.Stream`, vous devez les sérialiser. Spécifiez le sérialiseur que vous souhaitez utiliser en définissant l'attribut d'assemblage `LambdaSerializer`. Pour de plus amples informations, veuillez consulter the section called "Sérialisation dans les fonctions Lambda".

Rubriques

- . NETmodèles d'exécution pour Lambda
- Gestionnaires de bibliothèques de classes
- Gestionnaires d'assemblages exécutables
- Sérialisation dans les fonctions Lambda
- Simplifiez le code de la fonction à l'aide du cadre d'annotations Lambda
- Restrictions liées au gestionnaire de fonctions Lambda
- Meilleures pratiques en matière de code pour les fonctions Lambda C#

# . NETmodèles d'exécution pour Lambda

Il existe deux modèles d'exécution différents pour exécuter les fonctions Lambda dans. NET: l'approche de la bibliothèque de classes et l'approche de l'assemblage exécutable.

Dans l'approche de la bibliothèque de classes, vous fournissez à Lambda une chaîne indiquant le `AssemblyName`, `ClassName`, et `Method` de la fonction à invoquer. Pour en savoir plus sur le format de cette chaîne, consultez the section called “Gestionnaires de bibliothèques de classes”. Pendant la phase d'initialisation de la fonction, la classe de votre fonction est initialisée et tout code contenu dans le constructeur est exécuté.

Dans l'approche de l'assemblage exécutable, vous utilisez la fonctionnalité des instructions de premier niveau de C# 9. Cette approche génère un assemblage exécutable que Lambda exécute chaque fois qu'il reçoit une commande d'invocation pour votre fonction. Vous ne fournissez à Lambda que le nom de l'assemblage exécutable à exécuter.

Les sections suivantes donnent des exemples de code de fonction pour ces deux approches.

## Gestionnaires de bibliothèques de classes

Le code de la fonction Lambda suivant montre un exemple d'une méthode du gestionnaire (`FunctionHandler`) pour une fonction Lambda qui utilise l'approche de la bibliothèque de classe. Dans cet exemple de fonction, Lambda reçoit un événement API de Gateway qui appelle la fonction. La fonction lit un enregistrement dans une base de données et le renvoie dans le cadre de la réponse de la API passerelle.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }
```

```
    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Lorsque vous créez une fonction Lambda, vous devez fournir à Lambda des informations sur le gestionnaire de votre fonction sous la forme d'une chaîne de gestionnaire. Cette chaîne indique à Lambda quelle méthode de votre code doit être exécutée lorsque votre fonction est invoquée. En C#, lorsque l'on utilise l'approche de la bibliothèque de classes, le format de la chaîne du gestionnaire est le suivant :

`ASSEMBLY::TYPE::METHOD`, où :

- `ASSEMBLY`est le nom du. NETfichier d'assemblage pour votre application. Si vous utilisez le `Amazon.Lambda.Tools` CLI pour créer votre application et que vous ne définissez pas le nom de l'assembly à l'aide de la `AssemblyName` propriété du fichier .csproj, il `ASSEMBLY` s'agit simplement du nom de votre fichier .csproj.
- `TYPE` est le nom complet du type de gestionnaire, composé de `Namespace` et de `ClassName`.
- `METHOD` est le nom de la méthode du gestionnaire de fonction dans votre code.

Dans l'exemple de code illustré, si l'assemblage est nommé `GetProductHandler`, alors la chaîne du gestionnaire sera `GetProductHandler::GetProductHandler.Function::FunctionHandler`.

## Gestionnaires d'assemblages exécutables

Dans l'exemple suivant, la fonction Lambda est définie comme un assemblage exécutable. La méthode du gestionnaire dans ce code est nommée `Handler`. Lors de l'utilisation d'assemblages exécutables, l'exécution Lambda doit être amorcée. Pour ce faire, vous devez utiliser la méthode

`LambdaBootstrapBuilder.Create`. Cette méthode prend comme entrée la méthode que votre fonction utilise en tant que gestionnaire et le sérialiseur Lambda à utiliser.

Pour plus d'informations sur l'utilisation des instructions de haut niveau, voir Présentation du. NET6 runtime pour AWS Lambda on the AWS compute blog.

```
namespace GetProductHandler;

IDatabaseRepository repo = new DatabaseRepository();

await LambdaBootstrapBuilder.Create<APIGatewayProxyRequest>(Handler, new
 DefaultLambdaJsonSerializer())
    .Build()
    .RunAsync();

async Task<APIGatewayProxyResponse> Handler(APIGatewayProxyRequest apigProxyEvent,
 ILambdaContext context)
{
    var id = apigProxyEvent.PathParameters["id"];

    var databaseRecord = await this.repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
};
```

Lors de l'utilisation d'assemblages exécutables, la chaîne du gestionnaire qui indique à Lambda comment exécuter votre code est le nom de l'assemblage. Dans cet exemple, ce serait `GetProductHandler`.

## Sérialisation dans les fonctions Lambda

Si votre fonction Lambda utilise des types d'entrée ou de sortie autres qu'un objet `Stream`, vous devez ajouter une bibliothèque de sérialisation à votre application. Vous pouvez mettre en œuvre la sérialisation en utilisant la sérialisation standard basée sur la réflexion fournie par `System.Text.Json` et `Newtonsoft.Json`, ou en utilisant la sérialisation générée par la source.

## Utilisation de la sérialisation générée par la source

La sérialisation générée par la source est une fonctionnalité de. NETversions 6 et ultérieures qui permettent de générer du code de sérialisation au moment de la compilation. Elle supprime le besoin de réflexion et peut améliorer les performances de votre fonction. Pour utiliser la sérialisation générée par la source dans votre fonction, procédez comme suit :

- Créez une nouvelle classe partielle qui hérite de `JsonSerializerContext`, en ajoutant des attributs `JsonSerializable` pour tous les types qui nécessitent une sérialisation ou une désérialisation.
- Configurez le `LambdaSerializer` afin d'utiliser un `SourceGeneratorLambdaJsonSerializer<T>`.
- Mettez à jour toute sérialisation ou désérialisation manuelle dans le code de votre application afin d'utiliser la classe nouvellement créée.

Un exemple de fonction utilisant la sérialisation générée par la source est présenté dans le code suivant.

```
[assembly:
 LambdaSerializer(typeof(SourceGeneratorLambdaJsonSerializer<CustomSerializer>))]

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
```

```
            Body = JsonSerializer.Serialize(databaseRecord,
 CustomSerializer.Default.Product)
        };
    }
}

[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{

}
```

> **Note**
>
> Si vous souhaitez utiliser la compilation native anticipée (AOT) avec Lambda, vous devez utiliser la sérialisation générée par la source.

## Utilisation de la sérialisation basée sur la réflexion

AWS fournit des bibliothèques prédéfinies pour vous permettre d'ajouter rapidement une sérialisation à votre application. Vous pouvez le configurer à l'aide des `Amazon.Lambda.Serialization.Json` NuGet packages `Amazon.Lambda.Serialization.SystemTextJson` ou. En arrière-plan, `Amazon.Lambda.Serialization.SystemTextJson` utilise `System.Text.Json` pour effectuer des tâches de sérialisation, et `Amazon.Lambda.Serialization.Json` utilise le package `Newtonsoft.Json`.

Vous pouvez également créer votre propre bibliothèque de sérialisation en implémentant l'interface `ILambdaSerializer`, disponible dans la bibliothèque `Amazon.Lambda.Core`. Cette interface définit deux méthodes :

- `T Deserialize<T>(Stream requestStream);`

  Vous implémentez cette méthode pour désérialiser la charge utile de la demande depuis l'`InvokeAPI`objet transmis à votre gestionnaire de fonctions Lambda.
- `T Serialize<T>(T response, Stream responseStream);`

Vous implémentez cette méthode pour sérialiser le résultat renvoyé par votre gestionnaire de fonctions Lambda dans la charge utile de réponse renvoyée par l'opération. `Invoke` API

## Simplifiez le code de la fonction à l'aide du cadre d'annotations Lambda

Lambda Annotations est un framework pour. NET6 et. NET8 qui simplifie l'écriture de fonctions Lambda en C#. Grâce au cadre d'annotations, vous pouvez remplacer une grande partie du code d'une fonction Lambda écrite à l'aide du modèle de programmation habituel. Le code écrit à l'aide du cadre utilise des expressions plus simples qui vous permettent de vous concentrer sur votre logique commerciale.

L'exemple de code suivant montre comment l'utilisation du cadre d'annotations peut simplifier l'écriture des fonctions Lambda. Le premier exemple montre le code écrit à l'aide du modèle de programme Lambda habituel, et le second montre l'équivalent à l'aide du cadre d'annotations.

```
public APIGatewayHttpApiV2ProxyResponse LambdaMathAdd(APIGatewayHttpApiV2ProxyRequest
 request, ILambdaContext context)
{
    if (!request.PathParameters.TryGetValue("x", out var xs))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    if (!request.PathParameters.TryGetValue("y", out var ys))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    var x = int.Parse(xs);
    var y = int.Parse(ys);
    return new APIGatewayHttpApiV2ProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = (x + y).ToString(),
        Headers = new Dictionary#string, string> { { "Content-Type", "text/plain" } }
    };
```

```
}
```

```
[LambdaFunction]
[HttpApi(LambdaHttpMethod.Get, "/add/{x}/{y}")]
public int Add(int x, int y)
{
    return x + y;
}
```

Pour un autre exemple de la façon dont l'utilisation des annotations Lambda peut simplifier votre code, consultez cet exemple d'application multiservice dans le référentiel. `awsdocs/aws-doc-sdk-examples` GitHub Le dossier `PamApiAnnotations` utilise des annotations Lambda dans le fichier principal `function.cs`. À titre de comparaison, le dossier `PamApi` contient des fichiers équivalents écrits à l'aide du modèle de programmation Lambda habituel.

Le cadre d'annotations utilise des générateurs de source pour générer un code qui traduit le modèle de programmation Lambda en code tel qu'il apparaît dans le deuxième exemple.

Pour plus d'informations sur l'utilisation des annotations Lambda pour. NET, consultez les ressources suivantes :

- Le `aws/aws-lambda-dotnet` GitHub référentiel.
- Présentant. NETAnnotations Lambda Framework (version préliminaire) sur le blog des outils AWS de développement.
- Le `Amazon.Lambda.Annotations` NuGet colis.

## Injection de dépendances grâce au cadre d'annotations Lambda

Vous pouvez également utiliser le cadre d'annotations Lambda pour ajouter l'injection de dépendance à vos fonctions Lambda en utilisant une syntaxe que vous connaissez bien. Lorsque vous ajoutez un attribut `[LambdaStartup]` à un fichier `Startup.cs`, le cadre d'annotations Lambda génère le code nécessaire au moment de la compilation.

```
[LambdaStartup]
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
```

```
    }
}
```

Votre fonction Lambda peut injecter des services en utilisant l'injection de constructeur ou en injectant dans des méthodes individuelles à l'aide de l'attribut `[FromServices]`.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(IDatabaseRepository repo)
    {
        this._repo = repo;
    }

    [LambdaFunction]
    [HttpApi(LambdaHttpMethod.Get, "/product/{id}")]
    public async Task<Product> FunctionHandler([FromServices] IDatabaseRepository
 repository, string id)
    {
        return await this._repo.GetById(id);
    }
}
```

## Restrictions liées au gestionnaire de fonctions Lambda

Notez que des restrictions s'appliquent à la signature du gestionnaire.

- Il peut ne pas s'agir du `unsafe` et utiliser des types de pointeur dans la signature du gestionnaire, bien que le contexte `unsafe` puisse être utilisé dans la méthode de gestionnaire et ses dépendances. Pour de plus amples informations, veuillez consulterdangereux (référence C#)sur le site Web Microsoft Docs.
- Il peut ne pas transférer un certain nombre de paramètres à l'aide du mot clé `params` ou utiliser `ArgIterator` comme paramètre d'entrée ou de retour, pour prendre en charge un nombre de paramètres variable.

- Le gestionnaire n'est peut-être pas une méthode générique, par exemple, IList <T>Sort <T>(IList<T>input).
- Les gestionnaires asynchrones avec une signature `async void` ne sont pas pris en charge.

## Meilleures pratiques en matière de code pour les fonctions Lambda C#

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire.
- Contrôlez les dépendances du package de déploiement de vos fonctions.  L'environnement AWS Lambda d'exécution contient un certain nombre de bibliothèques. Pour activer le dernier ensemble de mises à jour des fonctionnalités et de la sécurité, Lambda met régulièrement à jour ces bibliothèques. Ces mises à jour peuvent introduire de subtiles modifications dans le comportement de votre fonction Lambda. Pour disposer du contrôle total des dépendances que votre fonction utilise, empaquetez toutes vos dépendances avec votre package de déploiement.
- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution.  Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation. Pour les fonctions créées dans. NET, évitez de télécharger l'intégralité de la AWS SDK bibliothèque dans le cadre de votre package de déploiement. Dépendez plutôt de manière sélective des modules qui récupèrent les composants dont SDK vous avez besoin (par exemple DynamoDB, les SDK modules Amazon S3 et les bibliothèques principales Lambda).

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant

un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.

- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.
- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Créez et déployez des fonctions Lambda C# à l'aide des archives de fichiers .zip

Un package de déploiement .NET (archive de fichier .zip) contient l'assemblage compilé de votre fonction ainsi que toutes les dépendances de l'assemblage. Le package contient également un fichier *proj*`.deps.json`. Il indique à l'exécution .NET toutes les dépendances de votre fonction et un fichier *proj*`.runtimeconfig.json`, qui est utilisé pour configurer l'exécution.

Pour déployer des fonctions Lambda individuelles, vous pouvez utiliser la CLI .NET Lambda Global de `Amazon.Lambda.Tools`. L'utilisation de la commande `dotnet lambda deploy-function` crée automatiquement un package de déploiement .zip et le déploie sur Lambda. Toutefois, nous vous recommandons d'utiliser des frameworks tels que le AWS Serverless Application Model (AWS SAM) ou le AWS Cloud Development Kit (AWS CDK) pour déployer vos applications .NET AWS.

Les applications sans serveur comprennent généralement une combinaison de fonctions Lambda et d'autres fonctions Services AWS gérées qui fonctionnent ensemble pour effectuer une tâche métier particulière. AWS SAM et AWS CDK simplifiez la création et le déploiement de fonctions Lambda avec d'autres fonctions Services AWS à grande échelle. La spécification du AWS SAM modèle fournit une syntaxe simple et claire pour décrire les fonctions Lambda, les API, les autorisations, les configurations et les autres AWS ressources qui constituent votre application sans serveur. Grâce au AWS CDK, vous pouvez définir l'infrastructure cloud en tant que code pour vous aider à créer des applications fiables, évolutives et rentables dans le cloud à l'aide de langages de programmation et de cadres modernes tels que .NET. Vous pouvez AWS SAM utiliser AWS CDK la CLI globale .NET Lambda pour empaqueter vos fonctions.

Bien qu'il soit possible d'utiliser des couches Lambda avec des fonctions en C# à l'aide de la CLI .NET Core, nous vous le déconseillons. Les fonctions en C# qui utilisent des couches chargent manuellement les assemblages partagés dans la mémoire pendant le Phase d'initialisation, ce qui peut augmenter les temps de démarrage à froid. Incluez plutôt tout le code partagé au moment de la compilation pour tirer parti des optimisations intégrées du compilateur .NET.

Vous trouverez des instructions pour créer et déployer des fonctions .NET Lambda à l'aide de la AWS SAM CLI globale .NET Lambda et du .NET Lambda dans les sections suivantes. AWS CDK

Rubriques

- Utilisation de la CLI .NET Lambda Global
- Déployez les fonctions Lambda C# à l'aide de AWS SAM
- Déployez les fonctions Lambda C# à l'aide de AWS CDK

- Déployez des applications ASP.NET

## Utilisation de la CLI .NET Lambda Global

La CLI .NET et l'extension .NET Lambda Global Tools (`Amazon.Lambda.Tools`) offrent un moyen multiplateforme de créer des applications Lambda basées sur .NET, de les empaqueter et de les déployer dans Lambda. Dans cette section, vous apprendrez à créer de nouveaux projets Lambda .NET à l'aide de la CLI .NET et des modèles Amazon Lambda, et à les empaqueter et les déployer en utilisant `Amazon.Lambda.Tools`

Rubriques

- Prérequis
- Création de projets .NET à l'aide de la CLI .NET
- Déploiement de projets .NET à l'aide de la CLI .NET
- Utilisation de couches Lambda avec l'interface de ligne de commande .NET

### Prérequis

KIT DE DÉVELOPPEMENT LOGICIEL .NET 8

Si ce n'est pas déjà fait, installez le SDK .NET 8 et Runtime.

AWS Modèles de projets .NET Amazon.Lambda.Templates

Pour générer le code de votre fonction Lambda, utilisez le Amazon.Lambda.Templates NuGet package. Pour installer ce package de modèle, exécutez la commande suivante :

```
dotnet new install Amazon.Lambda.Templates
```

AWS Outils CLI globaux .NET d'Amazon.Lambda.Tools

Pour créer vos fonctions Lambda, utilisez l'Amazon.Lambda.Tools extension outils globaux .NET. Pour installer Amazon.Lambda.Tools, exécutez la commande suivante :

```
dotnet tool install -g Amazon.Lambda.Tools
```

Pour plus d'informations sur l'extension Amazon.Lambda.Tools .NET CLI, consultez le référentiel AWS Extensions for .NET CLI sur GitHub.

## Création de projets .NET à l'aide de la CLI .NET

Dans la CLI .NET, utilisez la commande `dotnet new` pour créer des projets .NET à partir de la ligne de commande. Lambda propose des modèles supplémentaires à l'aide du `Amazon.Lambda.Templates` NuGet package.

Après avoir installé ce package, exécutez la commande suivante pour obtenir la liste des modèles disponibles.

```
dotnet new list
```

Pour examiner les détails relatifs à un modèle, utilisez l'option `help`. Par exemple, pour obtenir des détails sur le modèle `lambda.EmptyFunction`, exécutez la commande suivante.

```
dotnet new lambda.EmptyFunction --help
```

Pour créer un modèle de base pour une fonction Lambda .NET, utilisez le modèle `lambda.EmptyFunction`. Celui-ci crée une fonction simple qui prend une chaîne comme entrée et la convertit en majuscules à l'aide de la méthode `ToUpper`. Ce modèle prend en charge les options suivantes :

- `--name` – Nom de la fonction.
- `--region`— La AWS région dans laquelle créer la fonction.
- `--profile`— Le nom d'un profil dans votre fichier AWS SDK for .NET d'informations d'identification. Pour en savoir plus sur les profils d'identification dans .NET, consultez la section Configurer les AWS informations d'identification dans le Guide du développeur du AWS SDK for .NET.

Dans cet exemple, nous créons une nouvelle fonction vide nommée `myDotnetFunction` en utilisant le profil et les Région AWS paramètres par défaut :

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Cette commande crée les fichiers et répertoires suivants dans le répertoire de votre projet.

```
### myDotnetFunction
    ### src
```

```
    #   ### myDotnetFunction
    #       ### Function.cs
    #       ### Readme.md
    #       ### aws-lambda-tools-defaults.json
    #       ### myDotnetFunction.csproj
    ### test
        ### myDotnetFunction.Tests
            ### FunctionTest.cs
            ### myDotnetFunction.Tests.csproj
```

Sous le répertoire `src/myDotnetFunction`, examinez les fichiers suivants :

- aws-lambda-tools-defaults.json : c'est ici que vous spécifiez les options de ligne de commande lors du déploiement de votre fonction Lambda. Exemples :

```
  "profile" : "default",
  "region" : "us-east-2",
  "configuration" : "Release",
  "function-architecture": "x86_64",
  "function-runtime":"dotnet8",
  "function-memory-size" : 256,
  "function-timeout" : 30,
  "function-handler" : "myDotnetFunction::myDotnetFunction.Function::FunctionHandler"
```

- Function.cs : code de fonction de votre gestionnaire Lambda. Il s'agit d'un modèle C # qui inclut la bibliothèque `Amazon.Lambda.Core` par défaut et un attribut `LambdaSerializer` par défaut. Pour plus d'informations sur les conditions de sérialisation et les options, consultez Sérialisation dans les fonctions Lambda. Il inclut également un exemple de fonction que vous pouvez modifier pour appliquer votre code de fonction Lambda.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted into
 a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializ

namespace myDotnetFunction;

public class Function
{
```

```
    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary#
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}
```

- my DotnetFunction .csproj : fichier MSBuild qui répertorie les fichiers et les assemblages qui composent votre application.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
    <!-- This property makes the build directory similar to a publish directory and
 helps the AWS .NET Lambda Mock Test Tool find project dependencies. -->
    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
    <!-- Generate ready to run images during publishing to improve cold start time.
 -->
    <PublishReadyToRun>true</PublishReadyToRun>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
 Version="2.4.0" />
  </ItemGroup>
</Project>
```

- Readme : utilisez ce fichier pour documenter votre fonction Lambda.

Sous le répertoire `myfunction/test`, examinez les fichiers suivants :

- my DotnetFunction .tests.csproj : Comme indiqué précédemment, il s'agit d'un fichier MSBuild qui répertorie les fichiers et les assemblages qui composent votre projet de test. Notez également qu'il

comprend aussi la bibliothèque `Amazon.Lambda.Core`, ce qui vous permet d'intégrer de manière transparente tout modèle Lambda requis pour tester votre fonction.

```
<Project Sdk="Microsoft.NET.Sdk">
   ...

    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
   ...
```

- FunctionTest.cs : le même fichier de modèle de code C# que celui inclus dans le `src` répertoire. Modifiez ce fichier pour mettre en miroir le code de production de votre fonction, et testez-le avant de télécharger votre fonction Lambda sur un environnement de production.

```
using Xunit;
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {

            // Invoke the lambda function and confirm the string was upper cased.
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

## Déploiement de projets .NET à l'aide de la CLI .NET

Pour créer votre package de déploiement et le déployer sur Lambda, utilisez les outils CLI `Amazon.Lambda.Tools`. Pour déployer votre fonction à partir des fichiers que vous avez créés

dans les étapes précédentes, accédez d'abord au dossier contenant le fichier `.csproj` de votre fonction.

```
cd myDotnetFunction/src/myDotnetFunction
```

Pour déployer votre code sur Lambda sous la forme d'un package de déploiement .zip, exécutez la commande suivante. Choisissez le nom de votre fonction.

```
dotnet lambda deploy-function myDotnetFunction
```

Pendant le déploiement, l'assistant vous demande de sélectionner unthe section called "Rôle d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources)". Pour cet exemple, sélectionnez le `lambda_basic_role`.

Une fois votre fonction déployée, vous pouvez la tester dans le cloud en utilisant la commande `dotnet lambda invoke-function`. Pour l'exemple de code dans le modèle `lambda.EmptyFunction`, vous pouvez tester votre fonction en transmettant une chaîne en utilisant l'option `--payload`.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
```

Si votre fonction a été déployée avec succès, vous devriez obtenir une sortie similaire à celle qui suit.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
Amazon Lambda Tools for .NET Core applications (5.8.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory
 Size: 256 MB     Max Memory Used: 12 MB
```

### Utilisation de couches Lambda avec l'interface de ligne de commande .NET

> **Note**
>
> Il se peut que l'utilisation de couches avec des fonctions dans un langage compilé tel que C# ne procure pas les mêmes avantages qu'avec un langage interprété tel que Python. C# étant un langage compilé, vos fonctions doivent toujours charger manuellement les assemblages partagés dans la mémoire pendant la phase d'initialisation, ce qui peut augmenter les temps de démarrage à froid. Nous recommandons plutôt d'inclure tout le code partagé au moment de la compilation afin de tirer parti des optimisations intégrées du compilateur.

L'interface de ligne de commande .NET prend en charge les commandes qui vous aident à publier des couches et à déployer des fonctions C# consommant des couches. Pour publier une couche dans un compartiment Amazon S3 spécifié, exécutez la commande suivante dans le même répertoire que votre fichier `.csproj` :

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-
bucket <s3_bucket_name>
```

Ensuite, lorsque vous déployez votre fonction à l'aide de l'interface de ligne de commande .NET, spécifiez l'ARN de couche à consommer dans la commande suivante :

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-
east-1:123456789012:layer:layer-name:1
```

Pour un exemple complet de fonction Hello World, consultez l' blank-csharp-with-layerexemple.

## Déployez les fonctions Lambda C# à l'aide de AWS SAM

The AWS Serverless Application Model (AWS SAM) est une boîte à outils qui permet de rationaliser le processus de création et d'exécution d'applications sans serveur sur AWS. Vous définissez les ressources de votre application dans un modèle YAML ou JSON et vous utilisez l'interface de ligne de AWS SAM commande (AWS SAM CLI) pour créer, empaqueter et déployer vos applications. Lorsque vous créez une fonction Lambda à partir d'un AWS SAM modèle, elle crée AWS SAM automatiquement un package de déploiement ou une image de conteneur .zip avec le code de votre fonction et les dépendances que vous spécifiez. AWS SAM déploie ensuite votre fonction à l'aide d'une AWS CloudFormation pile. Pour en savoir plus sur l'utilisation des fonctions Lambda AWS SAM

pour créer et déployer des fonctions Lambda, consultez Getting started with AWS SAM dans le Guide du AWS Serverless Application Model développeur.

Les étapes suivantes vous montrent comment télécharger, créer et déployer un exemple d'application Hello World en .NET à l'aide de AWS SAM. Cet exemple d'application utilise une fonction Lambda et un point de terminaison Amazon API Gateway pour implémenter une API backend de base. Lorsque vous envoyez une requête HTTP GET à votre point de terminaison API Gateway, API Gateway invoque votre fonction Lambda. La fonction renvoie un message « hello world », ainsi que l'adresse IP de l'instance de la fonction Lambda qui traite votre requête.

Lorsque vous créez et déployez votre application en utilisant AWS SAM, en arrière-plan, la AWS SAM CLI utilise la `dotnet lambda package` commande pour empaqueter les ensembles de codes de fonction Lambda individuels.

## Prérequis

KIT DE DÉVELOPPEMENT LOGICIEL .NET 8

Installez le SDK .NET 8 et le Runtime.

AWS SAM CLI version 1.39 ou ultérieure

Pour savoir comment installer la dernière version de la AWS SAM CLI, consultez la section Installation de la AWS SAM CLI.

## Déployer un exemple d' AWS SAM application

1. Initialisez l'application en utilisant le modèle Hello world .NET à l'aide de la commande suivante.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Cette commande crée les fichiers et répertoires suivants dans le répertoire de votre projet.

```
### sam-app
    ### README.md
    ### events
    #   ### event.json
    ### omnisharp.json
    ### samconfig.toml
    ### src
```

```
#   ### HelloWorld
#       ### Function.cs
#       ### HelloWorld.csproj
#       ### aws-lambda-tools-defaults.json
### template.yaml
### test
    ### HelloWorld.Test
        ### FunctionTest.cs
        ### HelloWorld.Tests.csproj
```

2. Accédez au répertoire contenant le `template.yaml file`. Ce fichier est un modèle qui définit les ressources AWS de votre application, notamment votre fonction Lambda et API Gateway.

```
cd sam-app
```

3. Pour créer la source de votre application, exécutez la commande suivante.

```
sam build
```

4. Pour déployer votre application sur AWS, exécutez la commande suivante.

```
sam deploy --guided
```

Cette commande permet d'empaqueter et de déployer votre application à l'aide de la série d'invites suivante. Pour accepter les options par défaut, appuyez sur Entrée.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous d'entrery.

- Nom de la pile : nom de la pile à déployer sur AWS CloudFormation. Ce nom doit être propre à votre Compte AWS et Région AWS.
- Région AWS: le sur lequel Région AWS vous souhaitez déployer votre application.
- Confirmer les modifications avant le déploiement : sélectionnez oui pour examiner manuellement tous les ensembles de modifications avant que AWS SAM ne déploie les modifications de l'application. Si vous sélectionnez Non, la AWS SAM CLI déploie automatiquement les modifications apportées aux applications.

- Autoriser la création de rôles IAM dans la CLI SAM : de nombreux AWS SAM modèles, y compris celui de Hello world dans cet exemple, créent des rôles AWS Identity and Access Management (IAM) pour autoriser vos fonctions Lambda à accéder à d'autres. Services AWS Sélectionnez Oui pour autoriser le déploiement d'une AWS CloudFormation pile qui crée ou modifie des rôles IAM.
- Désactiver le rollback : par défaut, AWS SAM en cas d'erreur lors de la création ou du déploiement de votre stack, il rétablit la version précédente. Sélectionnez Non pour accepter cette valeur par défaut.
- HelloWorldFunction l'autorisation n'est peut-être pas définie, est-ce que ça va : Entrezy.
- Enregistrer les arguments dans samconfig.toml : sélectionnez oui pour enregistrer vos choix de configuration. À l'avenir, vous pourrez à nouveau exécuter `sam deploy` sans paramètres pour déployer les modifications apportées à votre application.

5. Une fois le déploiement de votre application terminé, la CLI renvoie l'Amazon Resource Name (ARN) de la fonction Lambda Hello World et le rôle IAM créé pour cette fonction. Elle affiche également le point de terminaison de votre API Gateway. Pour tester votre application, ouvrez le point de terminaison dans un navigateur. Vous devriez voir une réponse similaire à la suivante.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Pour supprimer vos ressources, exécutez la commande suivante. Notez que le point de terminaison de l'API que vous avez créé est un point de terminaison public accessible via le réseau Internet. Nous vous recommandons de supprimer ce point de terminaison après les tests.

```
sam delete
```

## Étapes suivantes

Pour en savoir plus sur la création et le déploiement de fonctions Lambda AWS SAM à l'aide de .NET, consultez les ressources suivantes :

- Le Guide du développeur AWS Serverless Application Model (AWS SAM)
- Création d'applications .NET sans serveur avec AWS Lambda et avec la CLI SAM

# Déployez les fonctions Lambda C# à l'aide de AWS CDK

AWS Cloud Development Kit (AWS CDK) Il s'agit d'un framework de développement logiciel open source permettant de définir l'infrastructure cloud comme du code avec des langages de programmation modernes et des frameworks tels que .NET. AWS CDK les projets sont exécutés pour générer des AWS CloudFormation modèles qui sont ensuite utilisés pour déployer votre code.

Pour créer et déployer un exemple d'application Hello world .NET à l'aide de AWS CDK, suivez les instructions des sections suivantes. L'exemple d'application implémente une API backend de base composée d'un point de terminaison API Gateway et d'une fonction Lambda. Lorsque vous envoyez une requête HTTP GET au point de terminaison, API Gateway invoque la fonction Lambda. La fonction renvoie un message Hello world, ainsi que l'adresse IP de l'instance Lambda qui traite votre requête.

## Prérequis

KIT DE DÉVELOPPEMENT LOGICIEL .NET 8

Installez le SDK .NET 8 et le Runtime.

AWS CDK version 2

Pour savoir comment installer la dernière version de la version 2, AWS CDK voir Getting started with the AWS CDK in the AWS Cloud Development Kit (AWS CDK) v2 Developer Guide.

## Déployer un exemple d' AWS CDK application

1. Créez un répertoire de projet pour l'application modèle et accédez à ce répertoire.

```
mkdir hello-world
cd hello-world
```

2. Initialisez une nouvelle AWS CDK application en exécutant la commande suivante.

```
cdk init app --language csharp
```

La commande crée les fichiers et répertoires suivants dans votre répertoire de projet.

```
### README.md
### cdk.json
```

```
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
    ### HelloWorld.sln
```

3. Ouvrez le répertoire `src` et créez une nouvelle fonction Lambda à l'aide de la CLI .NET. Il s'agit de la fonction que vous allez déployer à l'aide de AWS CDK. Dans cet exemple, vous créez une fonction Hello world nommée `HelloWorldLambda` à l'aide du modèle `lambda.EmptyFunction`.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Après cette étape, la structure de votre répertoire à l'intérieur de votre répertoire de projet devrait ressembler à ce qui suit.

```
### README.md
### cdk.json
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
    ### HelloWorld.sln
    ### HelloWorldLambda
        ### src
        #   ### HelloWorldLambda
        #       ### Function.cs
        #       ### HelloWorldLambda.csproj
        #       ### Readme.md
        #       ### aws-lambda-tools-defaults.json
        ### test
            ### HelloWorldLambda.Tests
                ### FunctionTest.cs
                ### HelloWorldLambda.Tests.csproj
```

4. Ouvrez le fichier `HelloWorldStack.cs` à partir du répertoire `src/HelloWorld`. Remplacez le contenu du fichier par le code suivant.

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.Logs;
using Constructs;

namespace CdkTest
{
    public class HelloWorldStack : Stack
    {
        internal HelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var buildOption = new BundlingOptions()
            {
                Image = Runtime.DOTNET_8.BundlingImage,
                User = "root",
                OutputType = BundlingOutput.ARCHIVED,
                Command = new string[]{
            "/bin/sh",
                "-c",
                " dotnet tool install -g Amazon.Lambda.Tools"+
                " && dotnet build"+
                " && dotnet lambda package --output-package /asset-output/
function.zip"
                }
            };

             var helloWorldLambdaFunction = new Function(this,
 "HelloWorldFunction", new FunctionProps
            {
                Runtime = Runtime.DOTNET_8,
                MemorySize = 1024,
                LogRetention = RetentionDays.ONE_DAY,
                Handler =
 "HelloWorldLambda::HelloWorldLambda.Function::FunctionHandler",
                Code = Code.FromAsset("./src/HelloWorldLambda/src/
HelloWorldLambda", new Amazon.CDK.AWS.S3.Assets.AssetOptions
                {
                    Bundling = buildOption
                }),
            });
        }
    }
```

```
}
```

Il s'agit du code permettant de compiler et de regrouper le code de l'application, ainsi que de la définition de la fonction Lambda elle-même. L'objet `BundlingOptions` permet de créer un fichier zip, ainsi qu'un ensemble de commandes utilisées pour générer le contenu du fichier zip. Dans ce cas, la commande `dotnet lambda package` est utilisée pour compiler et générer le fichier zip.

5. Pour déployer votre application, exécutez la commande suivante.

```
cdk deploy
```

6. Invoquez votre fonction Lambda déployée à l'aide de la CLI .NET Lambda.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Une fois les tests terminés, vous pouvez supprimer les ressources que vous avez créées, à moins que vous souhaitez les conserver. Exécutez la commande suivante pour supprimer vos ressources.

```
cdk destroy
```

### Étapes suivantes

Pour en savoir plus sur la création et le déploiement de fonctions Lambda AWS CDK à l'aide de .NET, consultez les ressources suivantes :

- Utilisation du AWS CDK en C#
- Créez, empaquetez et publiez des fonctions Lambda .NET C# avec le CDK AWS

## Déployez des applications ASP.NET

Outre l'hébergement de fonctions événementielles, vous pouvez également utiliser .NET avec Lambda pour héberger des applications ASP.NET légères. Vous pouvez créer et déployer des applications ASP.NET à l'aide du `Amazon.Lambda.AspNetCoreServer` NuGet package. Dans cette section, vous apprendrez à déployer une API web ASP.NET sur Lambda à l'aide de l'outillage de la CLI .NET Lambda.

Rubriques


- Prérequis
- Déploiement d'une API Web ASP.NET sur Lambda
- Déploiement des API minimales ASP.NET sur Lambda


## Prérequis

KIT DE DÉVELOPPEMENT LOGICIEL .NET 8

Installez le SDK .NET 8 et ASP.NET Core Runtime.

Amazon.Lambda.Tools

Pour créer vos fonctions Lambda, utilisez l'Amazon.Lambda.Tools extension outils globaux .NET. Pour installer Amazon.Lambda.Tools, exécutez la commande suivante :

```
dotnet tool install -g Amazon.Lambda.Tools
```

Pour plus d'informations sur l'extension Amazon.Lambda.Tools .NET CLI, consultez le référentiel AWS Extensions for .NET CLI sur GitHub.

Amazon.Lambda.Templates

Pour générer le code de votre fonction Lambda, utilisez le Amazon.Lambda.Templates NuGet package. Pour installer ce package de modèle, exécutez la commande suivante :

```
dotnet new --install Amazon.Lambda.Templates
```

## Déploiement d'une API Web ASP.NET sur Lambda

Pour déployer une API web à l'aide d'ASP.NET, vous pouvez utiliser les modèles Lambda .NET pour créer un nouveau projet d'API web. Utilisez la commande suivante pour initialiser un nouveau projet d'API web ASP.NET. Dans l'exemple de commande, nous nommons le projet `AspNetOnLambda`.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Cette commande crée les fichiers et répertoires suivants dans le répertoire de votre projet.

```
.
```

```
### AspNetOnLambda
    ### src
    #   ### AspNetOnLambda
    #       ### AspNetOnLambda.csproj
    #       ### Controllers
    #       #   ### ValuesController.cs
    #       ### LambdaEntryPoint.cs
    #       ### LocalEntryPoint.cs
    #       ### Readme.md
    #       ### Startup.cs
    #       ### appsettings.Development.json
    #       ### appsettings.json
    #       ### aws-lambda-tools-defaults.json
    #       ### serverless.template
    ### test
        ### AspNetOnLambda.Tests
            ### AspNetOnLambda.Tests.csproj
            ### SampleRequests
            #   ### ValuesController-Get.json
            ### ValuesControllerTests.cs
            ### appsettings.json
```

Lorsque Lambda invoque votre fonction, le point d'entrée qu'elle utilise est le fichier `LambdaEntryPoint.cs`. Le fichier créé par le modèle Lambda .NET contient le code suivant.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

Le point d'entrée utilisé par Lambda doit hériter de l'une des trois classes de base du package `Amazon.Lambda.AspNetCoreServer`. Ces trois classes de base sont les suivantes :

- `APIGatewayProxyFunction`
- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

La classe par défaut utilisée lorsque vous créez votre fichier `LambdaEntryPoint.cs` à l'aide du modèle Lambda .NET fourni est `APIGatewayProxyFunction`. La classe de base que vous utilisez dans votre fonction dépend de la couche d'API qui se trouve devant votre fonction Lambda.

Chacune des trois classes de base contient une méthode publique appelée `FunctionHandlerAsync`. Le nom de cette méthode fera partie de la chaîne du gestionnaire que Lambda utilise pour invoquer votre fonction. La méthode `FunctionHandlerAsync` transforme la charge utile de l'événement entrant dans le format ASP.NET correct et la réponse ASP.NET en une charge utile de réponse Lambda. Pour l'exemple de projet `AspNetOnLambda` affiché, la chaîne du gestionnaire serait la suivante.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Pour déployer l'API sur Lambda, exécutez les commandes suivantes pour accéder au répertoire contenant votre fichier de code source et déployer votre fonction à l'aide de AWS CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

> **Tip**
>
> Lorsque vous déployez une API à l'aide de la **`dotnet lambda deploy-serverless`** commande, AWS CloudFormation attribuez à votre fonction Lambda un nom basé sur le nom de pile que vous avez spécifié lors du déploiement. Pour donner un nom personnalisé à votre fonction Lambda, modifiez le `serverless.template` fichier pour ajouter une `FunctionName` propriété à la `AWS::Serverless::Function` ressource. Voir Type de nom dans le guide de AWS CloudFormation l'utilisateur pour en savoir plus.

## Déploiement des API minimales ASP.NET sur Lambda

Pour déployer une API minimale ASP.NET sur Lambda, vous pouvez utiliser les modèles Lambda .NET pour créer un nouveau projet d'API minimale. Utilisez la commande suivante

pour initialiser un nouveau projet d'API minimale. Dans cet exemple, nous nommons le projet `MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

La commande crée les fichiers et répertoires suivants dans le répertoire de votre projet.

```
### MinimalApiOnLambda
    ### src
        ### MinimalApiOnLambda
            ### Controllers
            #   ### CalculatorController.cs
            ### MinimalApiOnLambda.csproj
            ### Program.cs
            ### Readme.md
            ### appsettings.Development.json
            ### appsettings.json
            ### aws-lambda-tools-defaults.json
            ### serverless.template
```

Le fichier `Program.cs` contient le code suivant.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();

// Add AWS Lambda support. When application is run in Lambda Kestrel is swapped out as
 the web server with Amazon.Lambda.AspNetCoreServer. This
// package will act as the webserver translating request and responses between the
 Lambda event source and ASP.NET Core.
builder.Services.AddAWSLambdaHosting(LambdaEventSource.RestApi);

var app = builder.Build();


app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.MapGet("/", () => "Welcome to running ASP.NET Core Minimal API on AWS Lambda");
```

```
app.Run();
```

Pour configurer votre API minimale afin qu'elle s'exécute sur Lambda, vous devrez peut-être modifier ce code pour que les demandes et les réponses entre Lambda et ASP.NET Core soient correctement traduites. La fonction est configurée par défaut pour une source d'événements d'API REST. Pour une API HTTP ou un équilibreur de charge d'application, remplacez `(LambdaEventSource.RestApi)` par l'une des options suivantes :

- `(LambdaEventSource.HttpAPi)`
- `(LambdaEventSource.ApplicationLoadBalancer)`

Pour déployer votre API minimale sur Lambda, exécutez les commandes suivantes pour accéder au répertoire contenant votre fichier de code source et déployer votre fonction à l'aide de AWS CloudFormation.

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda
dotnet lambda deploy-serverless
```

# Déployez. NETFonctions Lambda avec images de conteneurs

Il existe trois méthodes pour créer une image de conteneur pour un. NETFonction Lambda :

- Utilisation d'une image AWS de base pour. NET

  Les images de base AWS sont préchargées avec une exécution du langage, un client d'interface d'exécution pour gérer l'interaction entre Lambda et votre code de fonction, et un émulateur d'interface d'exécution pour les tests locaux.

- Utilisation d'une image de base AWS uniquement pour le système d'exploitation

  AWS Les images de base réservées au système d'exploitation contiennent une distribution Amazon Linux et l'émulateur d'interface d'exécution. Ces images sont couramment utilisées pour créer des images de conteneur pour les langages compilés, tels que Go et Rust, et pour une langue ou une version linguistique pour laquelle Lambda ne fournit pas d'image de base, comme Node.js 19. Vous pouvez également utiliser des images de base uniquement pour le système d'exploitation pour implémenter un environnement d'exécution personnalisé. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour. NETdans l'image.

- Utilisation d'une image non AWS basique

  Vous pouvez utiliser une autre image de base à partir d'un autre registre de conteneur, comme Alpine Linux ou Debian. Vous pouvez également utiliser une image personnalisée créée par votre organisation. Pour rendre l'image compatible avec Lambda, vous devez inclure le client d'interface d'exécution pour. NETdans l'image.

> **Tip**
>
> Pour réduire le temps nécessaire à l'activation des fonctions du conteneur Lambda, consultez Utiliser des générations en plusieurs étapes (français non garanti) dans la documentation Docker. Pour créer des images de conteneur efficaces, suivez la section Bonnes pratiques pour l'écriture de Dockerfiles (français non garanti).

Cette page explique comment créer, tester et déployer des images de conteneur pour Lambda.

Rubriques

- AWS images de base pour. NET
- Utilisation d'une image AWS de base pour. NET
- Utilisation d’une autre image de base avec le client d’interface d’exécution

## AWS images de base pour. NET

AWS fournit les images de base suivantes pour. NET:

| Balises | Environnement d’exécution | Système d’exploitation | Dockerfile | Obsolescence |
|---|---|---|---|---|
| 8 | . NET8 | Amazon Linux | Dockerfile pour. NET8 sur GitHub | Non prévu |
| 6 | . NET6 | Amazon Linux 2 | Dockerfile pour. NET6 sur GitHub | 20 déc. 2024 |

ECRRéférentiel Amazon : gallery.ecr.aws/lambda/dotnet

## Utilisation d'une image AWS de base pour. NET

### Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- . NETSDK— Les étapes suivantes utilisent le. NET8 image de base. Assurez-vous que votre. NETversion correspond à la version de l'image de base que vous spécifiez dans votre Dockerfile.
- Docker

### Création et déploiement d’une image à l’aide d’une image de base

Dans les étapes suivantes, vous utiliserez Amazon.Lambda.Templates et Amazon.Lambda.Tools pour créer un. NETprojet. Ensuite, vous créez une image Docker, vous la chargez sur Amazon ECR et vous la déployez sur une fonction Lambda.

1. Installez le package Amazon.Lambda.Templates. NuGet

```
dotnet new install Amazon.Lambda.Templates
```

2. Créez un. NETprojet utilisant le `lambda.image.EmptyFunction` modèle.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

3. Accédez au répertoire *`MyFunction`*`/src/`*`MyFunction`*. C'est ici que les fichiers du projet sont stockés. Examinez les fichiers suivants :

   - aws-lambda-tools-defaults.json — Ce fichier vous permet de spécifier les options de ligne de commande lors du déploiement de votre fonction Lambda.
   - Function.cs : le code de fonction de votre gestionnaire Lambda. Il s'agit d'un modèle C# qui inclut la bibliothèque `Amazon.Lambda.Core` par défaut et un attribut `LambdaSerializer` par défaut. Pour plus d'informations sur les conditions de sérialisation et les options, consultez Sérialisation dans les fonctions Lambda. Vous pouvez utiliser le code fourni pour les tests ou le remplacer par le vôtre.
   - MyFunction.csproj — A. NETfichier de projet, qui répertorie les fichiers et les assemblages qui constituent votre application.
   - Readme.md : ce fichier contient plus d'informations sur l'exemple de fonction Lambda.

4. Examinez le fichier Docker dans le répertoire `src/`*`MyFunction`*. Vous pouvez utiliser le Dockerfile fourni pour les tests ou le remplacer par le vôtre. Si vous utilisez le vôtre, assurez-vous de :

   - Définissez la `FROM` propriété sur URIl'image de base. Votre. NETla version doit correspondre à la version de l'image de base.
   - Définir l'argument `CMD` pour le gestionnaire de la fonction Lambda. Il doit correspondre à `image-command` dans `aws-lambda-tools-defaults.json`.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'`root`utilisateur lorsqu'aucune `USER` instruction n'est fournie.

Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM public.ecr.aws/lambda/dotnet:8

# Copy function code to Lambda-defined environment variable
COPY publish/* ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "MyFunction::MyFunction.Function::FunctionHandler" ]
```

5. Installez le Amazon.Lambda.Tools. NETOutil mondial.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Si Amazon.Lambda.Tools est déjà installé, assurez-vous que vous disposez de la dernière version.

```
dotnet tool update -g Amazon.Lambda.Tools
```

6. Changez le répertoire en *MyFunction*/src/*MyFunction*, si vous n'y êtes pas déjà.

```
cd src/MyFunction
```

7. Utilisez Amazon.Lambda.Tools pour créer l'image Docker, la transférer vers un nouveau référentiel Amazon ECR et déployer la fonction Lambda.

   Pour`--function-role`, spécifiez le nom du rôle, et non le nom de ressource Amazon (ARN), du rôle d'exécution de la fonction. Par exemple, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

   Pour plus d'informations sur le Amazon.Lambda.Tools. NETOutil global, voir les AWS extensions pour. NETCLIréférentiel sur GitHub.

8. Invoquer la fonction.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Si tout réussit, vous devez voir ce qui suit :

```
Payload:
"TESTING THE FUNCTION"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory
 Size: 256 MB     Max Memory Used: 12 MB
```

9. Supprimez la fonction Lambda.

```
dotnet lambda delete-function MyFunction
```

## Utilisation d’une autre image de base avec le client d’interface d’exécution

Si vous utilisez une image de base uniquement pour le système d’exploitation ou une autre image de base, vous devez inclure le client d'interface d'exécution dans votre image. Le client d’interface d’exécution étend leUtilisation du moteur d'exécution Lambda API pour des environnements d'exécution personnalisés, qui gère l’interaction entre Lambda et votre code de fonction.

L'exemple suivant montre comment créer une image de conteneur pour. NETen utilisant une image non AWS basique, et comment ajouter l'Amazon.Lambda. RuntimeSupport package, qui est le client d'interface d'exécution Lambda pour. NET. L'exemple de Dockerfile utilise le Microsoft. NET8 image de base.

### Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- . NETSDK— Les étapes suivantes utilisent un. NET8 image de base. Assurez-vous que votre. NETversion correspond à la version de l'image de base que vous spécifiez dans votre Dockerfile.
- Docker

## Création et déploiement d'une image à l'aide d'une image de base alternative

1. Installez le package Amazon.Lambda.Templates. NuGet

```
dotnet new install Amazon.Lambda.Templates
```

2. Créez un. NETprojet utilisant le `lambda.CustomRuntimeFunction` modèle. Ce modèle inclut l'Amazon.Lambda. RuntimeSupportcolis.

```
dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
```

3. Accédez au répertoire *MyFunction*`/src/`*MyFunction*. C'est ici que les fichiers du projet sont stockés. Examinez les fichiers suivants :

   - aws-lambda-tools-defaults.json — Ce fichier vous permet de spécifier les options de ligne de commande lors du déploiement de votre fonction Lambda.
   - Function.cs : le code contient une classe avec une méthode `Main` qui initialise la bibliothèque `Amazon.Lambda.RuntimeSupport` en tant qu'amorce. La méthode `Main` est le point d'entrée du processus de la fonction. La méthode `Main` enveloppe le gestionnaire de fonctions dans un encapsuleur avec lequel l'amorce peut travailler. Pour plus d'informations, consultez la section Utilisation d'Amazon.Lambda. RuntimeSupport en tant que bibliothèque de classes dans le GitHub référentiel.
   - MyFunction.csproj — A. NETfichier de projet, qui répertorie les fichiers et les assemblages qui constituent votre application.
   - Readme.md : ce fichier contient plus d'informations sur l'exemple de fonction Lambda.

4. Ouvrez le fichier `aws-lambda-tools-defaults.json` et ajoutez les lignes suivantes :

```
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
```

   - package-typee : définit le package de déploiement en tant qu'image de conteneur.
   - docker-host-build-output-dir : définit le répertoire de sortie pour le processus de construction.

   Example aws-lambda-tools-defaults.json

```
{
  "Information": [
```

```
    "This file provides default values for the deployment wizard inside Visual
 Studio and the AWS Lambda commands added to the .NET Core CLI.",
    "To learn more about the Lambda commands with the .NET Core CLI execute the
 following command at the command line in the project root directory.",
    "dotnet lambda help",
    "All the command line options for the Lambda command can be specified in this
 file."
  ],
  "profile": "",
  "region": "us-east-1",
  "configuration": "Release",
  "function-runtime": "provided.al2023",
  "function-memory-size": 256,
  "function-timeout": 30,
  "function-handler": "bootstrap",
  "msbuild-parameters": "--self-contained true",
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
}
```

5. Créez un Dockerfile dans le référentiel *MyFunction*/src/*MyFunction*. L'exemple suivant de Dockerfile utilise un Microsoft. NETimage de base au lieu d'une image AWS de base.

   - Définissez la propriété FROM pour l'identifiant de l'image de base. Votre. NETla version doit correspondre à la version de l'image de base.
   - Utilisez la commande COPY pour copier la fonction dans le répertoire /var/task.
   - Définissez le ENTRYPOINT sur le module que vous souhaitez que le conteneur Docker exécute lorsqu'il démarre. Dans ce cas, le module est l'amorce, qui initialise la bibliothèque Amazon.Lambda.RuntimeSupport.

   Notez que l'exemple de Dockerfile n'inclut aucune instruction. USER Lorsque vous déployez une image de conteneur sur Lambda, Lambda définit automatiquement un utilisateur Linux par défaut doté des autorisations les moins privilégiées. Ceci est différent du comportement standard de Docker qui est défini par défaut par l'rootutilisateur lorsqu'aucune USER instruction n'est fournie.

   Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM mcr.microsoft.com/dotnet/runtime:8.0

# Set the image's internal work directory
```

```
WORKDIR /var/task

# Copy function code to Lambda-defined environment variable
COPY "bin/Release/net8.0/linux-x64"  .

# Set the entrypoint to the bootstrap
ENTRYPOINT ["/usr/bin/dotnet", "exec", "/var/task/bootstrap.dll"]
```

6. Installez le Amazon.Lambda.Tools. NETExtension Global Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Si Amazon.Lambda.Tools est déjà installé, assurez-vous que vous disposez de la dernière version.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Utilisez Amazon.Lambda.Tools pour créer l'image Docker, la transférer vers un nouveau référentiel Amazon ECR et déployer la fonction Lambda.

Pour`--function-role`, spécifiez le nom du rôle, et non le nom de ressource Amazon (ARN), du rôle d'exécution de la fonction. Par exemple, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Pour plus d'informations sur le Amazon.Lambda.Tools. NETCLIextension, voir les AWS extensions pour. NETCLIréférentiel sur GitHub.

8. Invoquer la fonction.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Si tout réussit, vous devez voir ce qui suit :

```
Payload:
"TESTING THE FUNCTION"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
```

```
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory
 Size: 256 MB     Max Memory Used: 12 MB
```

9. Supprimez la fonction Lambda.

```
dotnet lambda delete-function MyFunction
```

# Compiler le code de fonction .NET Lambda dans un format d'exécution natif

.NET 8 prend en charge la compilation native ahead-of-time (AOT). Avec la compilation anticipée native, vous pouvez compiler le code de votre fonction Lambda dans un format d'environnement d'exécution natif, ce qui élimine la nécessité de compiler le code .NET au moment de l'exécution. La compilation anticipée native peut réduire le temps de démarrage à froid des fonctions Lambda que vous écrivez en .NET. Pour plus d'informations, consultez Présentation de l'environnement d'exécution .NET 8 AWS Lambda sur le AWS Compute Blog.

Sections


- Le fichier d'exécution Lambda
- Prérequis
- Premiers pas
- Sérialisation
- Réduction
- Résolution des problèmes


## Le fichier d'exécution Lambda

Pour déployer une fonction Lambda compilée avec une compilation AOT native, utilisez le runtime Lambda géré .NET 8. Ce runtime prend en charge l'utilisation des architectures x86_64 et arm64.

Lorsque vous déployez une fonction .NET Lambda sans utiliser AOT, votre application est d'abord compilée en code Intermediate Language (IL). Au moment de l'exécution, le compilateur just-in-time (JIT) du moteur d'exécution Lambda prend le code IL et le compile en code machine selon les besoins. Avec une fonction Lambda compilée à l'avance avec l'AOT natif, vous compilez votre code en code machine lorsque vous déployez votre fonction. Ainsi, vous ne dépendez pas du runtime .NET ou du SDK du runtime Lambda pour compiler votre code avant son exécution.

L'une des limites de l'AOT est que le code de votre application doit être compilé dans un environnement doté du même système d'exploitation Amazon Linux 2023 (AL2023) que celui utilisé par le moteur d'exécution .NET 8. La CLI .NET Lambda fournit des fonctionnalités permettant de compiler votre application dans un conteneur Docker à l'aide d'une image AL2023.

Pour éviter d'éventuels problèmes de compatibilité entre architectures, nous vous recommandons vivement de compiler votre code dans un environnement doté de la même architecture de processeur que celle que vous avez configurée pour votre fonction. Pour en savoir plus sur les limites de la compilation entre architectures, consultez la section Compilation croisée dans la documentation Microsoft .NET.

## Prérequis

Docker

Pour utiliser l'AOT natif, votre code de fonction doit être compilé dans un environnement doté du même système d'exploitation AL2023 que le moteur d'exécution .NET 8. Les commandes .NET CLI décrites dans les sections suivantes utilisent Docker pour développer et créer des fonctions Lambda dans un environnement AL2023.

KIT DE DÉVELOPPEMENT LOGICIEL .NET 8

La compilation native AOT est une fonctionnalité de .NET 8. Vous devez installer le SDK .NET 8 sur votre machine de compilation, et pas uniquement sur le moteur d'exécution.

Amazon.Lambda.Tools

Pour créer vos fonctions Lambda, utilisez l'Amazon.Lambda.Tools extension outils globaux .NET. Pour installer Amazon.Lambda.Tools, exécutez la commande suivante :

```
dotnet tool install -g Amazon.Lambda.Tools
```

Pour plus d'informations sur l'extension Amazon.Lambda.Tools .NET CLI, consultez le référentiel AWS Extensions for .NET CLI sur GitHub.

Amazon.Lambda.Templates

Pour générer le code de votre fonction Lambda, utilisez le Amazon.Lambda.Templates NuGet package. Pour installer ce package de modèle, exécutez la commande suivante :

```
dotnet new install Amazon.Lambda.Templates
```

## Premiers pas

La CLI globale .NET et le AWS Serverless Application Model (AWS SAM) fournissent tous deux des modèles de démarrage pour créer des applications utilisant l'AOT natif. Pour créer votre première

fonction Lambda à l'aide de la compilation anticipée native, suivez les étapes décrites dans les instructions suivantes.

Pour initialiser et déployer une fonction Lambda compilée à l'aide de la compilation anticipée native

1. Initialisez un nouveau projet en utilisant le modèle de la compilation anticipée native, puis naviguez dans le répertoire contenant les fichiers créés `.cs` et `.csproj`. Dans cet exemple, nous allons nommer notre fonction `NativeAotSample`.

   ```
   dotnet new lambda.NativeAOT -n NativeAotSample
   cd ./NativeAotSample/src/NativeAotSample
   ```

   Le fichier créé `Function.cs` par le modèle de la compilation anticipée native contient le code de fonction suivant.

   ```
   using Amazon.Lambda.Core;
   using Amazon.Lambda.RuntimeSupport;
   using Amazon.Lambda.Serialization.SystemTextJson;
   using System.Text.Json.Serialization;

   namespace NativeAotSample;

   public class Function
   {
       /// <summary>
       /// The main entry point for the Lambda function. The main function is called
    once during the Lambda init phase. It
       /// initializes the .NET Lambda runtime client passing in the function handler
    to invoke for each Lambda event and
       /// the JSON serializer to use for converting Lambda JSON format to the .NET
    types.
       /// </summary>
       private static async Task Main()
       {
           Func<string, ILambdaContext, string> handler = FunctionHandler;
           await LambdaBootstrapBuilder.Create(handler, new
    SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>())
               .Build()
               .RunAsync();
       }

       /// <summary>
   ```

```
    /// A simple function that takes a string and does a ToUpper.
    ///
    /// To use this handler to respond to an AWS event, reference the appropriate
 package from
    /// https://github.com/aws/aws-lambda-dotnet#events
    /// and change the string input parameter to the desired event type. When the
 event type
    /// is changed, the handler type registered in the main method needs to be
 updated and the LambdaFunctionJsonSerializerContext
    /// defined below will need the JsonSerializable updated. If the return type
 and event type are different then the
    /// LambdaFunctionJsonSerializerContext must have two JsonSerializable
 attributes, one for each type.
    ///
    // When using Native AOT extra testing with the deployed Lambda functions is
 required to ensure
    // the libraries used in the Lambda function work correctly with Native AOT. If
 a runtime
    // error occurs about missing types or methods the most likely solution will be
 to remove references to trim-unsafe
    // code or configure trimming options. This sample defaults to partial TrimMode
 because currently the AWS
    // SDK for .NET does not support trimming. This will result in a larger
 executable size, and still does not
    // guarantee runtime trimming errors won't be hit.
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public static string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

/// <summary>
/// This class is used to register the input event and return type for the
 FunctionHandler method with the System.Text.Json source generator.
/// There must be a JsonSerializable attribute for each type used as the input and
 return type or a runtime error will occur
/// from the JSON serializer unable to find the serialization information for
 unknown types.
/// </summary>
[JsonSerializable(typeof(string))]
```

```
public partial class LambdaFunctionJsonSerializerContext : JsonSerializerContext
{
    // By using this partial class derived from JsonSerializerContext, we can
 generate reflection free JSON Serializer code at compile time
    // which can deserialize our class and properties. However, we must attribute
 this class to tell it what types to generate serialization code for.
    // See https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-
text-json-source-generation
```

La compilation anticipée native permet de compiler votre application en un seul fichier binaire natif. Le point d'entrée de ce fichier binaire est la méthode `static Main`. Dans `static Main`, l'exécution Lambda est amorcée et la méthode `FunctionHandler` est configurée. Dans le cadre de l'amorçage de l'exécution, un sérialiseur généré par la source est configuré à l'aide de `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Pour déployer votre application sur Lambda, assurez-vous que Docker est en cours d'exécution dans votre environnement local et exécutez la commande suivante.

```
dotnet lambda deploy-function
```

Dans les coulisses, la CLI globale .NET télécharge une image Docker AL2023 et compile le code de votre application dans un conteneur en cours d'exécution. Le fichier binaire compilé est renvoyé vers votre système de fichiers local avant d'être déployé sur Lambda.

3. Testez votre fonction en exécutant la commande suivante. Remplacez `<FUNCTION_NAME>` par le nom que vous avez choisi pour votre fonction dans l'assistant de déploiement.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

La réponse de la CLI comprend des détails sur les performances pour le démarrage à froid (durée d'initialisation) et la durée totale d'exécution de l'invocation de la fonction.

4. Pour supprimer les AWS ressources que vous avez créées en suivant les étapes précédentes, exécutez la commande suivante. Remplacez `<FUNCTION_NAME>` par le nom que vous avez choisi pour votre fonction dans l'assistant de déploiement. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez que des frais inutiles ne soient facturés à votre Compte AWS compte.

```
dotnet lambda delete-function <FUNCTION_NAME>
```

## Sérialisation

Pour déployer des fonctions sur Lambda à l'aide de la compilation anticipée native, le code de votre fonction doit utiliser la sérialisation générée par la source. Au lieu d'utiliser la réflexion au moment de l'exécution pour rassembler les métadonnées nécessaires à l'accès aux propriétés des objets pour la sérialisation, les générateurs de sources génèrent des fichiers sources C# qui sont compilés lorsque vous créez votre application. Pour configurer correctement le sérialiseur généré par la source, assurez-vous d'inclure tous les objets d'entrée et de sortie utilisés par votre fonction, ainsi que tous les types personnalisés. Par exemple, une fonction Lambda qui reçoit des événements d'API Gateway pour une API REST et renvoie un type de `Product` personnalisé inclurait un sérialiseur défini comme suit.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{
}
```

## Réduction

La compilation anticipée native permet de réduire le code de votre application dans le cadre de la compilation afin de s'assurer que le fichier binaire est aussi petit que possible. .NET 8 pour Lambda offre une meilleure prise en charge du découpage par rapport aux versions précédentes de .NET. Support a été ajouté aux bibliothèques d'exécution Lambda, au SDK AWS .NET, aux annotations .NET Lambda et au .NET 8 lui-même.

Ces améliorations offrent la possibilité d'éliminer les avertissements de découpage au moment de la création, mais .NET ne sera jamais totalement sûr en matière de découpage. Cela signifie que certaines parties des bibliothèques sur lesquelles votre fonction repose peuvent être supprimées lors de l'étape de compilation. Vous pouvez gérer cela en le définissant dans `TrimmerRootAssemblies` le cadre de votre `.csproj` fichier, comme indiqué dans l'exemple suivant.

```
<ItemGroup>
    <TrimmerRootAssembly Include="AWSSDK.Core" />
    <TrimmerRootAssembly Include="AWSXRayRecorder.Core" />
    <TrimmerRootAssembly Include="AWSXRayRecorder.Handlers.AwsSdk" />
    <TrimmerRootAssembly Include="Amazon.Lambda.APIGatewayEvents" />
    <TrimmerRootAssembly Include="bootstrap" />
```

```
    <TrimmerRootAssembly Include="Shared" />
</ItemGroup>
```

Notez que lorsque vous recevez un avertissement de découpage, l'ajout de la classe qui génère l'avertissement `TrimmerRootAssembly` risque de ne pas résoudre le problème. Un avertissement de découpage indique que la classe essaie d'accéder à une autre classe qui ne peut être déterminée avant l'exécution. Pour éviter les erreurs d'exécution, ajoutez cette deuxième classe à`TrimmerRootAssembly`.

Pour en savoir plus sur la gestion des avertissements de découpage, voir Présentation des avertissements de découpage dans la documentation Microsoft .NET.

## Résolution des problèmes

Error: Cross-OS native compilation is not supported. (Erreur : la compilation native entre systèmes d’exploitation n’est pas prise en charge).

> Votre version de l’outil global .NET Core Amazon.Lambda.Tools n’est pas à jour. Mettez à jour vers la dernière version et réessayez.

Docker : l’image du système d’exploitation « linux » ne peut pas être utilisée sur cette plateforme.

> Docker sur votre système est configuré pour utiliser des conteneurs Windows. Passez aux conteneurs Linux pour exécuter l’environnement de création anticipée native.

Pour plus d'informations sur les erreurs courantes, consultez le référentiel AWS NativeAOT pour .NET sur. GitHub

# Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction C#

Lorsque Lambda exécute votre fonction, il transmet un objet de contexte au gestionnaire. Cet objet fournit les propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `FunctionName` – Nom de la fonction Lambda.
- `FunctionVersion` – Version de la fonction.
- `InvokedFunctionArn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `AwsRequestId` – Identifiant de la demande d'invocation.
- `LogGroupName` – Groupe de journaux pour la fonction.
- `LogStreamName` – Flux de journal de l'instance de fonction.
- `RemainingTime` (`TimeSpan`) – Nombre de millisecondes restant avant l'expiration de l'exécution.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
- `Logger` L'objet enregistreur d'événements pour la fonction.

Vous pouvez utiliser les informations contenues dans l'objet `ILambdaContext` pour générer des informations sur l'invocation de votre fonction à des fins de contrôle. Le code suivant montre comment ajouter des informations contextuelles à un cadre de journalisation structuré. Dans cet exemple, la fonction ajoute `AwsRequestId` aux sorties du journal. La fonction utilise également la propriété `RemainingTime` pour annuler une tâche en vol si le délai d'attente de la fonction Lambda est sur le point d'être atteint.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
```

```
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request, ILambdaContext context)
    {
        Logger.AppendKey("AwsRequestId", context.AwsRequestId);

        var id = request.PathParameters["id"];

        using var cts = new CancellationTokenSource();

        try
        {
            cts.CancelAfter(context.RemainingTime.Add(TimeSpan.FromSeconds(-1)));

            var databaseRecord = await this._repo.GetById(id, cts.Token);

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
        finally
        {
            cts.Cancel();

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
    }
}
```

# Enregistrez et surveillez les fonctions Lambda C#

AWS Lambda surveille automatiquement les fonctions Lambda et envoie des entrées de journal à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des détails sur chaque invocation et d'autres sorties provenant du code de votre fonction au flux de journaux. Pour plus d'informations sur CloudWatch les journaux, consultezUtilisation des CloudWatch journaux et journaux avec Lambda.

Sections


- Création d'une fonction qui renvoie des journaux
- Utilisation d'outils de journalisation et de bibliothèques
- Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour la journalisation structurée
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux


## Création d'une fonction qui renvoie des journaux

Pour générer les journaux à partir de votre code de fonction, vous pouvez utiliser des méthodes sur la classe de console ou n'importe quelle bibliothèque de journalisation qui écrit dans `stdout` ou `stderr`.

Le. NETruntime enregistre les `REPORT` lignes `START``END`, et pour chaque appel. La ligne de rapport fournit les détails suivants.

REPORTchamps de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.
- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.

- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled – Pour les demandes suivies, résultat de l'échantillonnage.

## Utilisation d'outils de journalisation et de bibliothèques

Outils électriques pour AWS Lambda (. NET) est une boîte à outils destinée aux développeurs qui permet de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs. L'utilitaire Logging fournit un enregistreur optimisé pour Lambda qui inclut des informations supplémentaires sur le contexte des fonctions dans toutes vos fonctions avec une sortie structurée comme suit. JSON Utilisez cet utilitaire pour effectuer les opérations suivantes :

- Capturez les champs clés du contexte Lambda, du démarrage à froid et des structures enregistrant la sortie sous JSON
- Journaliser les événements d'invocation Lambda lorsque cela est demandé (désactivé par défaut)
- Imprimer tous les journaux uniquement pour un pourcentage d'invocations via l'échantillonnage des journaux (désactivé par défaut)
- Ajouter des clés supplémentaires au journal structuré à tout moment
- Utilisez un formateur de journal personnalisé (Bring Your Own Formatter) pour générer les journaux dans une structure compatible avec la journalisation de votre organisation RFC

## Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour la journalisation structurée

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World C# avec Powertools intégrés pour AWS Lambda (. NET) modules utilisant le AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison API Gateway,

la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message `hello world`.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- . NET6 ou. NET8
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du TypeScript modèle Hello World.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Créez l’application.

```
cd sam-app && sam build
```

3. Déployez l’application.

```
sam deploy --guided
```

4. Suivez les invites à l’écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l’expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les journaux de la fonction, exécutez sam logs. Pour en savoir plus, consultez Utilisation des journaux dans le Guide du développeur AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

La sortie du journal se présente comme suit :

```
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8
 2023-02-20T14:15:27.988000 INIT_START Runtime Version:
 dotnet:6.v13        Runtime Version ARN: arn:aws:lambda:ap-
southeast-2::runtime:699f346a05dae24c58c45790bc4089f252bf17dae3997e79b17d939a288aa1ec
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:28.229000
 START RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Version: $LATEST
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:29.259000
 2023-02-20T14:15:29.201Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
   {"_aws":{"Timestamp":1676902528962,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ColdStart","Unit":"Count"}],"Dimensions":
[["FunctionName"],["Service"]]}]},"FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","Service":"PowertoolsHelloWorld","ColdStart":1}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.479000
 2023-02-20T14:15:30.479Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
 {"ColdStart":true,"XrayTraceId":"1-63f3807f-5dbcb9910c96f50742707542","CorrelationId":"d3d
a549-4d67b2fdc015","FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionVersion":"$LATEST","FunctionMemorySize":256,"FunctionArn":"arn:aws:l
southeast-2:123456789012:function:sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionRequestId":"bed25b38-d012-42e7-ba28-
f272535fb80e","Timestamp":"2023-02-20T14:15:30.4602970Z","Level":"Information","Service":"P
 world API - HTTP 200"}
```

```
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.599000
 2023-02-20T14:15:30.599Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
   {"_aws":{"Timestamp":1676902528922,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ApiRequestCount","Unit":"Count"}],"Dimensions":
[["Service"]]}]},"Service":"PowertoolsHelloWorld","ApiRequestCount":1}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000 END
 RequestId: bed25b38-d012-42e7-ba28-f272535fb80e
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000
 REPORT RequestId: bed25b38-d012-42e7-ba28-f272535fb80e  Duration: 2450.99 ms
  Billed Duration: 2451 ms Memory Size: 256 MB     Max Memory Used: 74 MB  Init
 Duration: 240.05 ms
XRAY TraceId: 1-63f3807f-5dbcb9910c96f50742707542       SegmentId: 16b362cd5f52cba0
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

## Gestion de la conservation des journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker les journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation après laquelle les journaux CloudWatch sont automatiquement supprimés. Pour configurer la conservation des journaux, ajoutez ce qui suit à votre AWS SAM modèle :

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

## Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
```

```
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

# Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

# Instrumentation du code C# dans AWS Lambda

Lambda s'intègre pour vous aider AWS X-Ray à suivre, à déboguer et à optimiser les applications Lambda. Vous pouvez utiliser X-Ray pour suivre une demande lorsque celle-ci parcourt les ressources de votre application, qui peuvent inclure des fonctions Lambda et d'autres services AWS .

Pour envoyer des données de suivi à X-Ray, vous pouvez utiliser l'une des trois SDK bibliothèques suivantes :

- AWS Distro for OpenTelemetry (ADOT) — Une distribution sécurisée, prête pour la production et AWS prise en charge du (). OpenTelemetry OTel SDK
- Kit SDK AWS X-Ray pour .NET— Et SDK pour générer et envoyer des données de trace à X-Ray.
- Outils électriques pour AWS Lambda (. NET) — Une boîte à outils pour les développeurs permettant de mettre en œuvre les meilleures pratiques sans serveur et d'accroître la rapidité des développeurs.

Chacune d'entre elles SDKs propose des moyens d'envoyer vos données de télémétrie au service X-Ray. Vous pouvez ensuite utiliser X-Ray pour afficher, filtrer et avoir un aperçu des métriques de performance de votre application, afin d'identifier les problèmes et les occasions d'optimiser votre application.

> **Important**
>
> Les outils X-Ray et Powertools pour AWS Lambda SDKs font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. Les couches ADOT Lambda font partie d'une norme industrielle pour les instruments de traçage qui collectent plus de données en général, mais peuvent ne pas convenir à tous les cas d'utilisation. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre solution. Pour en savoir plus sur le choix entre les deux, consultez Choosing between the AWS Distro for Open Telemetry and X-Ray. SDKs

Sections


- Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour le traçage
- Utiliser le X-Ray SDK pour instrumenter votre. NETfonctions
- Activation du suivi avec la console Lambda

- Activation du traçage avec le Lambda API
- Activation du traçage avec AWS CloudFormation
- Interprétation d'un suivi X-Ray

## Utilisation de Powertools pour AWS Lambda (. NET) et AWS SAM pour le traçage

Suivez les étapes ci-dessous pour télécharger, créer et déployer un exemple d'application Hello World C# avec Powertools intégrés pour AWS Lambda (. NET) modules utilisant le AWS SAM. Cette application implémente un API backend de base et utilise Powertools pour émettre des journaux, des métriques et des traces. Il se compose d'un point de terminaison Amazon API Gateway et d'une fonction Lambda. Lorsque vous envoyez une GET demande au point de terminaison de la API passerelle, la fonction Lambda appelle, envoie des journaux et des métriques au format métrique intégré à CloudWatch, et envoie des traces à. AWS X-Ray La fonction renvoie un message hello world.

Prérequis

Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- . NET6 ou. NET8
- AWS CLI version 2
- AWS SAM CLIversion 1.75 ou ultérieure. Si vous possédez une ancienne version du AWS SAM CLI, consultez la section Mise à niveau du AWS SAM CLI.

Déployer un exemple d' AWS SAM application

1. Initialisez l'application à l'aide du TypeScript modèle Hello World.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Créez l’application.

```
cd sam-app && sam build
```

3. Déployez l'application.

```
sam deploy --guided
```

4. Suivez les invites à l'écran. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

> **Note**
>
> Car l'autorisation n'a HelloWorldFunction peut-être pas été définie, est-ce que ça va ? , assurez-vous de participery.

5. Obtenez le code URL de l'application déployée :

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Appelez le API point de terminaison :

```
curl <URL_FROM_PREVIOUS_STEP>
```

En cas de succès, vous obtiendrez cette réponse :

```
{"message":"hello world"}
```

7. Pour obtenir les traces de la fonction, exécutez sam traces.

```
sam traces
```

La sortie de la trace ressemble à ceci :

```
New XRay Service Graph
  Start time: 2023-02-20 23:05:16+08:00
  End time: 2023-02-20 23:05:16+08:00
  Reference Id: 0 - AWS::Lambda - sam-app-HelloWorldFunction-pNjujb7mEoew - Edges:
 [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
```

```
    - fault count(5XX): 0
    - total response time: 2.814
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-pNjujb7mEoew
 - Edges: []
   Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 2.429
  Reference Id: 2 - (Root) AWS::ApiGateway::Stage - sam-app/Prod - Edges: [0]
   Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 2.839
  Reference Id: 3 - client - sam-app/Prod - Edges: [2]
   Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 3] at (2023-02-20T23:05:16.521000) with id
 (1-63f38c2c-270200bf1d292a442c8e8a00) and duration (2.877s)
 - 2.839s - sam-app/Prod [HTTP: 200]
   - 2.836s - Lambda [HTTP: 200]
 - 2.814s - sam-app-HelloWorldFunction-pNjujb7mEoew [HTTP: 200]
 - 2.429s - sam-app-HelloWorldFunction-pNjujb7mEoew
   - 0.230s - Initialization
   - 2.389s - Invocation
     - 0.600s - ## FunctionHandler
       - 0.517s - Get Calling IP
   - 0.039s - Overhead
```

8. Il s'agit d'un API point de terminaison public accessible via Internet. Nous vous recommandons de supprimer le point de terminaison après un test.

```
sam delete
```

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

## Utiliser le X-Ray SDK pour instrumenter votre. NETfonctions

Vous pouvez instrumenter le code de gestionnaire pour enregistrer les métadonnées et suivre les appels en aval. Pour enregistrer les détails des appels que votre fonction effectue vers d'autres ressources et services, utilisez le Kit SDK AWS X-Ray pour .NET. Pour les obtenirSDK, ajoutez les `AWSXRayRecorder` packages à votre fichier de projet.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="2.1.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.7.103.24" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.7.104.3" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.13.0" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.11.0" />
  </ItemGroup>
</Project>
```

Il existe une gamme de packages Nuget qui fournissent une instrumentation automatique pour AWS SDKs Entity Framework et HTTP les requêtes. Pour voir l'ensemble complet des options de configuration, reportez-vous à la section AWS X-Ray SDKpour. NETdans le Guide AWS X-Ray du développeur.

Une fois que vous avez ajouté les packages Nuget de votre choix, configurez l'auto-instrumentation. La meilleure pratique consiste à effectuer cette configuration en dehors de la fonction du gestionnaire de votre fonction. Cela vous permet de tirer parti de la réutilisation de l'environnement d'exécution afin d'améliorer les performances de votre fonction. Dans l'exemple de code suivant,

la `RegisterXRayForAllServices` méthode est appelée dans le constructeur de fonctions pour ajouter de l'instrumentation à tous les AWS SDK appels.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        // Add auto instrumentation for all AWS SDK calls
        // It is important to call this method before initializing any SDK clients
        AWSSDKHandler.RegisterXRayForAllServices();
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

## Activation du suivi avec la console Lambda

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

## Activation du traçage avec le Lambda API

Configurez le suivi sur votre fonction Lambda avec AWS CLI ou utilisez AWS SDK les opérations suivantes : API

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant active le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du traçage avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

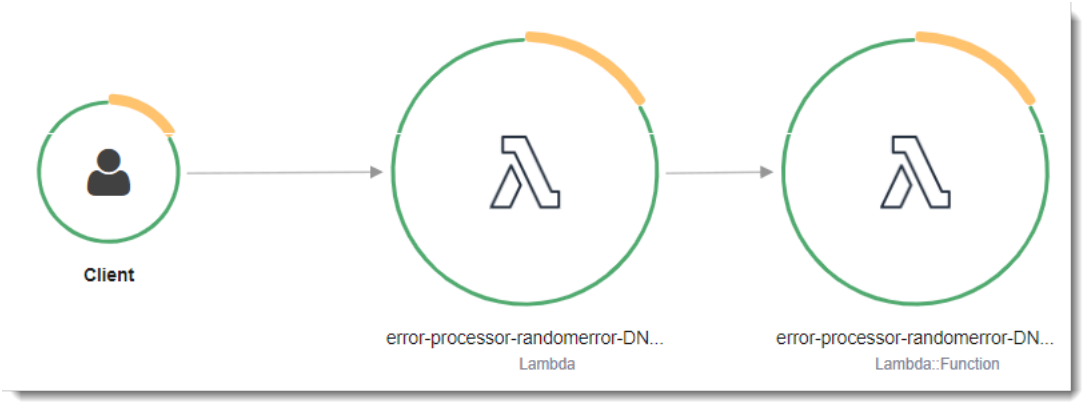
## Interprétation d'un suivi X-Ray

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Sinon, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

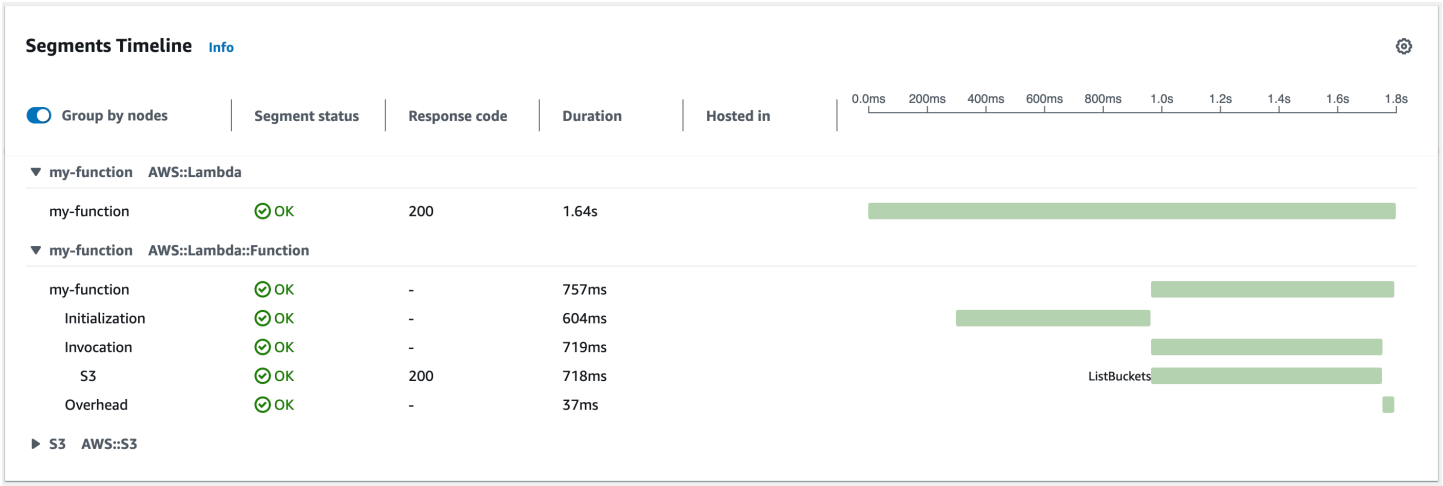
Une fois que vous avez configuré le suivi actif, vous pouvez observer des demandes spécifiques via votre application. Le graphique de services X-Ray affiche des informations sur votre application et tous ses composants. L'exemple suivant montre une application dotée de deux fonctions. La fonction principale traite les événements et renvoie parfois des erreurs. La deuxième fonction située en haut traite les erreurs qui apparaissent dans le groupe de journaux du premier et utilise le AWS SDK pour appeler X-Ray, Amazon Simple Storage Service (Amazon S3) et Amazon CloudWatch Logs.

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique. L'exemple suivant illustre une trace avec ces deux segments. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.



Cet exemple développe le `AWS::Lambda::Function` segment pour afficher ses trois sous-segments.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'exemple de trace présenté ici illustre le segment de fonction à l'ancienne. Les différences entre les segments de style ancien et nouveau sont décrites dans les paragraphes suivants. Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, seront transférées pour utiliser le nouveau format des messages de journal et des segments de trace.

Le segment de fonction à l'ancienne contient les sous-segments suivants :

- Initialization (Initialisation) : représente le temps passé à charger votre fonction et à exécuter le code d'initialisation. Ce sous-segment apparaît pour le premier événement traité par chaque instance de votre fonction.
- Invocation – Représente le temps passé à exécuter votre code de gestionnaire.
- Overhead (Travail supplémentaire) – Représente le temps que le fichier d'exécution Lambda passe à se préparer à gérer l'événement suivant.

Le segment de fonction du nouveau style ne contient pas de `Invocation` sous-segment. Au lieu de cela, les sous-segments clients sont rattachés directement au segment fonctionnel. Pour plus d'informations sur la structure des segments de fonction de l'ancien et du nouveau style, consultezthe section called “Comprendre les suivis X-Ray”.

Vous pouvez également instrumenter HTTP les clients, enregistrer SQL des requêtes et créer des sous-segments personnalisés avec des annotations et des métadonnées. Pour plus d'informations, consultez Kit SDK AWS X-Ray pour .NET dans le AWS X-Ray Guide du développeur.

**Tarification**

Vous pouvez utiliser le X-Ray Tracing gratuitement chaque mois jusqu'à une certaine limite dans le cadre du niveau AWS gratuit. Au-delà de ce seuil, X-Ray facture le stockage et la récupération du suivi. Pour en savoir plus, consultez Pricing AWS X-Ray (Tarification).

# Test de fonction AWS Lambda en C#

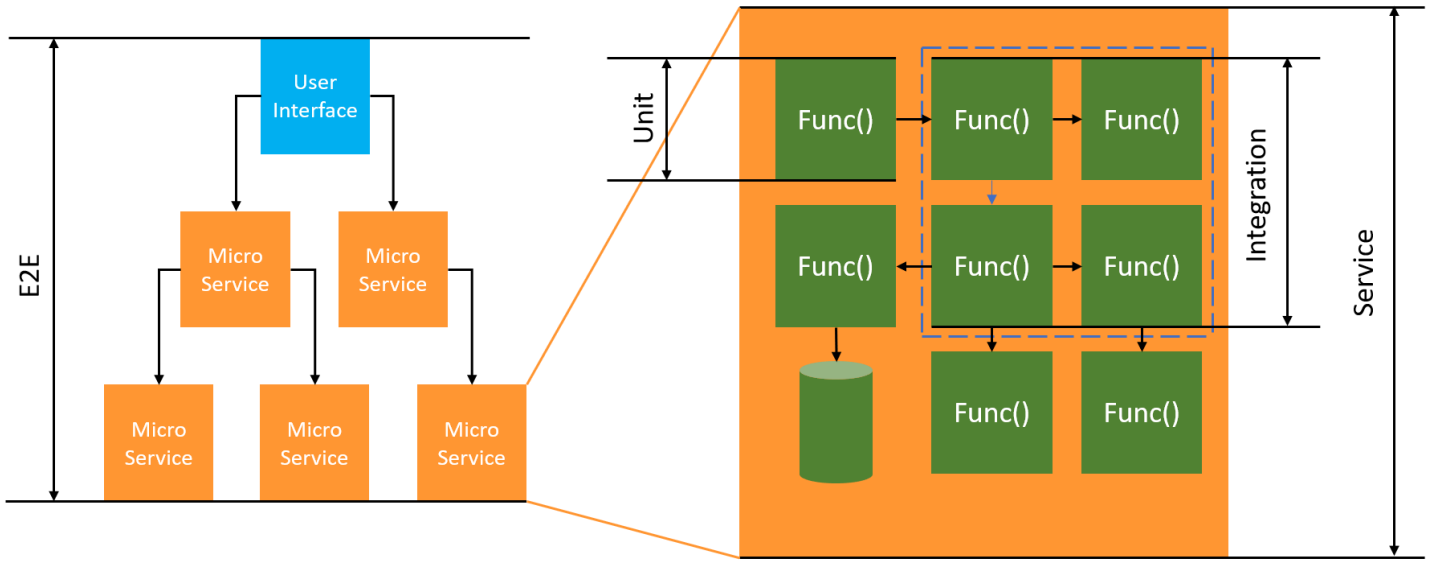
> **Note**
>
> Consultez le chapitre Test des fonctions pour une présentation complète des techniques et des bonnes pratiques pour tester les solutions sans serveur.

Le test des fonctions sans serveur utilise les types et techniques de tests traditionnels, mais vous devez également envisager de tester les applications sans serveur dans leur ensemble. Les tests basés sur le cloud fourniront la mesure la plus précise de la qualité de vos fonctions et de vos applications sans serveur.

Une architecture d'application sans serveur comprend des services gérés qui fournissent des fonctionnalités d'applications critiques par le biais d'appels d'API. C'est pourquoi votre cycle de développement doit inclure des tests automatisés qui vérifient la fonctionnalité lorsque votre fonction et vos services interagissent.

Si vous ne créez pas de tests basés sur le cloud, vous pouvez rencontrer des problèmes en raison des différences entre votre environnement local et l'environnement déployé. Votre processus d'intégration continue doit exécuter des tests sur une série de ressources allouées dans le cloud avant de promouvoir votre code vers l'environnement de déploiement suivant, comme l'assurance qualité, la mise en place ou la production.

Poursuivez la lecture de ce petit guide pour en savoir plus sur les stratégies de test pour les applications sans serveur, ou rendez-vous sur le référentiel Serverless Test Samples pour vous plonger dans des exemples pratiques, spécifiques au langage et à l'exécution que vous avez choisis.




Pour les tests sans serveur, vous continuerez à écrire des unités, des tests d'intégration et end-to-enddes tests.

- Tests unitaires : tests exécutés sur un bloc de code isolé. Par exemple, la vérification de la logique métier permettant de calculer les frais de livraison en fonction d'un article et d'une destination donnés.
- Tests d'intégration : tests impliquant au moins deux composants ou services qui interagissent, généralement dans un environnement cloud. Par exemple, la vérification d'une fonction traite les événements d'une file d'attente.
- nd-to-end Tests E : tests qui vérifient le comportement de l'ensemble d'une application. Il s'agit par exemple de s'assurer que l'infrastructure est correctement mise en place et que les événements circulent entre les services comme prévu pour enregistrer la commande d'un client.

## Test de vos applications sans serveur

Vous utiliserez généralement une combinaison d'approches pour tester le code de votre application sans serveur, notamment des tests dans le cloud, des tests avec des simulations et, occasionnellement, des tests avec des émulateurs.

### Tests dans le cloud

Les tests dans le cloud sont utiles pour toutes les phases de test, y compris les tests unitaires, les tests d'intégration et les end-to-end tests. Vous exécutez des tests sur du code déployé dans le cloud

et interagissez avec des services basés sur le cloud. Cette approche fournit la mesure la plus précise de la qualité de votre code.

Un moyen pratique de déboguer votre fonction Lambda dans le cloud est de passer par la console avec un événement de test. Un événement de test est une entrée JSON pour votre fonction. Si votre fonction ne nécessite pas d'entrée, l'événement peut être un document JSON vide (`{}`). La console fournit des exemples d'événements pour diverses intégrations de services. Après avoir créé un événement dans la console, vous pouvez le partager avec votre équipe pour faciliter les tests et les rendre plus cohérents.

> **Note**
>
> Tester une fonction dans la console est un moyen rapide de démarrer, mais l'automatisation de vos cycles de test garantit la qualité des applications et la rapidité du développement.

## Outils de test

Pour accélérer votre cycle de développement, il existe un certain nombre d'outils et de techniques que vous pouvez utiliser pour tester vos fonctions. Par exemple, Accélérer AWS SAM et le mode de surveillance AWS CDK réduisent tous deux le temps nécessaire pour mettre à jour les environnements cloud.

La façon dont vous définissez votre code de fonction Lambda facilite l'ajout de tests d'unité. Lambda nécessite un constructeur public sans paramètre pour initialiser votre catégorie. L'introduction d'un deuxième constructeur interne vous permet de contrôler les dépendances utilisées par votre application.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(): this(null)
    {
    }
```

```
    internal Function(IDatabaseRepository repo)
    {
        this._repo = repo ?? new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Pour écrire un test pour cette fonction, vous pouvez initialiser une nouvelle instance de votre catégorie `Function` et transmettre une implémentation simulée du `IDatabaseRepository`. Les exemples ci-dessous utilisent `XUnit`, `Moq`, et `FluentAssertions` pour écrire un test simple permettant de s'assurer que le `FunctionHandler` renvoie un code d'état 200.

```
using Xunit;
using Moq;
using FluentAssertions;

public class FunctionTests
{
    [Fact]
    public async Task TestLambdaHandler_WhenInputIsValid_ShouldReturn200StatusCode()
    {
        // Arrange
        var mockDatabaseRepository = new Mock<IDatabaseRepository>();

        var functionUnderTest = new Function(mockDatabaseRepository.Object);

        // Act
        var response = await functionUnderTest.FunctionHandler(new
 APIGatewayProxyRequest());
```

```
        // Assert
        response.StatusCode.Should().Be(200);
    }
}
```

Pour des exemples plus détaillés, notamment des exemples de tests asynchrones, consultez le référentiel d'échantillons de tests .NET sur. GitHub

# Création de fonctions Lambda avec PowerShell

Les sections suivantes expliquent comment les modèles de programmation courants et les concepts de base s'appliquent lorsque vous créez du code de fonction Lambda dans. PowerShell

Lambda fournit les exemples d'applications suivants pour : PowerShell

- blank-powershell — PowerShell Fonction qui montre l'utilisation de la journalisation, des variables d'environnement et du. AWS SDK

Avant de commencer, vous devez d'abord configurer un environnement PowerShell de développement. Pour obtenir des instructions sur la façon de procéder, veuillez consulter Configuration d'un environnement PowerShell de développement.

Pour savoir comment utiliser le AWSLambdaPSCore module pour télécharger des exemples de PowerShell projets à partir de modèles, créer des packages de PowerShell déploiement et déployer PowerShell des fonctions AWS dans le cloud, consultezDéployer des fonctions PowerShell Lambda avec des archives de fichiers .zip.

Lambda fournit les environnements d'exécution suivants pour. NETlangues :

| Nom | Identifiant | Système d'exploitation | Date d'obsolescence | Créer la fonction de blocage | Mettre à jour la fonction de blocage |
|---|---|---|---|---|---|
| . NET8 | `dotnet8` | Amazon Linux 2 | Non prévu | Non prévu | Non prévu |
| . NET6 | `dotnet6` | Amazon Linux 2 | 20 déc. 2024 | 28 février 2025 | 31 mars 2025 |

Rubriques

- Configuration d'un environnement PowerShell de développement
- Déployer des fonctions PowerShell Lambda avec des archives de fichiers .zip
- Définissez le gestionnaire de fonctions Lambda dans PowerShell
- Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions PowerShell

- Enregistrez et surveillez les fonctions Powershell Lambda

# Configuration d'un environnement PowerShell de développement

Lambda fournit un ensemble d'outils et de bibliothèques pour le PowerShell runtime. Pour les instructions d'installation, reportez-vous à la section Outils Lambda pour PowerShell activer. GitHub

Le AWSLambdaPSCore module inclut les applets de commande suivants pour aider à créer et à publier des fonctions Lambda PowerShell :

- Get- AWSPowerShellLambdaTemplate — Renvoie une liste de modèles de démarrage.
- Nouveau- AWSPowerShellLambda — Crée un PowerShell script initial basé sur un modèle.
- Publier- AWSPowerShellLambda — Publie un PowerShell script donné sur Lambda.
- Nouveau- AWSPowerShellLambdaPackage — Crée un package de déploiement Lambda que vous pouvez utiliser dans un système CI/CD pour le déploiement.

# Déployer des fonctions PowerShell Lambda avec des archives de fichiers .zip

Un package de déploiement pour l' PowerShell environnement d'exécution contient votre PowerShell script, les PowerShell modules requis pour votre PowerShell script et les assemblys nécessaires pour héberger PowerShell Core.

## Création de la fonction Lambda

Pour commencer à écrire et à invoquer un PowerShell script avec Lambda, vous pouvez utiliser `New-AWSPowerShellLambda` l'applet de commande pour créer un script de démarrage basé sur un modèle. Vous pouvez utiliser la cmdlet `Publish-AWSPowerShellLambda` pour déployer votre script vers Lambda. Ensuite, vous pouvez tester votre script via la ligne de commande ou la console Lambda.

Pour créer un nouveau PowerShell script, le télécharger et le tester, procédez comme suit :

1. Pour afficher la liste des modèles disponibles, exécutez la commande suivante :

   ```
   PS C:\> Get-AWSPowerShellLambdaTemplate

   Template               Description
   --------               -----------
   Basic                  Bare bones script
   CodeCommitTrigger      Script to process AWS CodeCommit Triggers
   ...
   ```

2. Pour créer un exemple de script basé sur le modèle `Basic`, exécutez la commande suivante :

   ```
   New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
   ```

   Un nouveau fichier nommé `MyFirstPSScript.ps1` est créé dans un nouveau sous-répertoire du répertoire actuel. Le nom de ce répertoire est basé sur le paramètre `-ScriptName`. Vous pouvez utiliser le paramètre `-Directory` pour choisir un autre répertoire.

   Vous pouvez voir que le nouveau fichier a le contenu suivant :

   ```
   # PowerShell script file to run as a Lambda function
   #
   # When executing in Lambda the following variables are predefined.
   ```

```
#   $LambdaInput - A PSObject that contains the Lambda function input data.
#   $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
 information about the currently running Lambda environment.
#
# The last item in the PowerShell pipeline is returned as the result of the Lambda
 function.
#
# To include PowerShell modules with your Lambda function, like the
 AWSPowerShell.NetCore module, add a "#Requires" statement
# indicating the module and version.

#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}

# Uncomment to send the input to CloudWatch Logs
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

3. Pour savoir comment les messages de journal de votre PowerShell script sont envoyés à Amazon CloudWatch Logs, décommentez la `Write-Host` ligne de l'exemple de script.

   Pour illustrer comment renvoyer des données à partir de vos fonctions Lambda, ajoutez une ligne à la fin du script avec `$PSVersionTable`. Cela ajoute le `$PSVersionTable` au PowerShell pipeline. Une fois le PowerShell script terminé, le dernier objet du PowerShell pipeline correspond aux données renvoyées par la fonction Lambda. `$PSVersionTable`est une variable PowerShell globale qui fournit également des informations sur l'environnement d'exécution.

   Après avoir effectué ces modifications, les deux dernières lignes de l'exemple de script ressemblent à ce qui suit :

   ```
   Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
   $PSVersionTable
   ```

4. Après avoir modifié le fichier `MyFirstPSScript.ps1`, modifiez le répertoire en spécifiant l'emplacement du script. Exécutez ensuite la commande suivante pour publier le script dans Lambda :

   ```
   Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name
    MyFirstPSScript -Region us-east-2
   ```

   Notez que le paramètre `-Name` spécifie le nom de la fonction Lambda qui s'affiche dans la console Lambda. Vous pouvez utiliser cette fonction pour appeler manuellement votre script.

5. Appelez votre fonction à l'aide de la commande AWS Command Line Interface (AWS CLI) `invoke`.

```
> aws lambda invoke --function-name MyFirstPSScript out
```

# Définissez le gestionnaire de fonctions Lambda dans PowerShell

Lorsqu'une fonction Lambda est invoquée, le gestionnaire Lambda appelle le script. PowerShell

Lorsque le PowerShell script est invoqué, les variables suivantes sont prédéfinies :

- *`$ LambdaInput`* — Un PSObject qui contient l'entrée du gestionnaire. Ces données d'entrée peuvent être des données d'événement (publiées par une source d'événement) ou des données d'entrée personnalisées que vous fournissez, telles qu'une chaîne ou n'importe quel objet de données personnalisé.
- *`$ LambdaContext`* — Un LambdaContext objet Amazon.Lambda.Core.I que vous pouvez utiliser pour accéder aux informations relatives à l'appel en cours, telles que le nom de la fonction en cours, la limite de mémoire, le temps d'exécution restant et la journalisation.

Par exemple, considérez l' PowerShell exemple de code suivant.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Ce script renvoie la FunctionName propriété obtenue à partir de la LambdaContext variable $.

> **Note**
>
> Vous devez utiliser l'`#Requires`instruction dans vos PowerShell scripts pour indiquer les modules dont dépendent vos scripts. Cette instruction effectue deux tâches importantes. 1) Il communique aux autres développeurs les modules utilisés par le script, et 2) il identifie les modules dépendants que les AWS PowerShell outils doivent intégrer au script, dans le cadre du déploiement. Pour plus d'informations sur l'`#Requires`instruction dans PowerShell, voir À propos des exigences. Pour plus d'informations sur les packages de PowerShell déploiement, consultezDéployer des fonctions PowerShell Lambda avec des archives de fichiers .zip. Lorsque votre fonction PowerShell Lambda utilise les AWS PowerShell applets de commande, veillez à définir une `#Requires` instruction qui fait référence au module, qui prend en charge PowerShell Core, et non au `AWSPowerShell.NetCore` module, qui ne prend en charge que `AWSPowerShell` Windows. PowerShell De plus, veillez à utiliser la version 3.3.270.0 ou plus récente de `AWSPowerShell.NetCore`, qui optimise le processus d'importation d'applet de commande. Si vous utilisez une version plus ancienne, vous

connaîtrez des démarrages à froid plus longs. Pour plus d'informations, consultez AWS Outils pour PowerShell.

## Renvoi de données

Certains appels Lambda sont destinés à renvoyer des données à leur appelant. Par exemple, si un appel répond à une demande web provenant d'API Gateway, notre fonction Lambda doit renvoyer la réponse. Pour PowerShell Lambda, le dernier objet ajouté au PowerShell pipeline est constitué des données renvoyées par l'invocation Lambda. Si l'objet est une chaîne, les données sont renvoyées en l'état. Dans le cas contraire, l'objet est converti en données JSON à l'aide de l'applet de commande `ConvertTo-Json`.

Par exemple, considérez l' PowerShell instruction suivante, qui `$PSVersionTable` s'ajoute au PowerShell pipeline :

```
$PSVersionTable
```

Une fois le PowerShell script terminé, le dernier objet du PowerShell pipeline correspond aux données renvoyées par la fonction Lambda. `$PSVersionTable`est une variable PowerShell globale qui fournit également des informations sur l'environnement d'exécution.

# Utilisation de l'objet de contexte Lambda pour récupérer les informations relatives aux fonctions PowerShell

Lorsque Lambda exécute votre fonction, il transmet les informations de contexte en mettant une variable `$LambdaContext` à la disposition du gestionnaire. Cette variable fournit des méthodes et des propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `FunctionName` – Nom de la fonction Lambda.
- `FunctionVersion` – Version de la fonction.
- `InvokedFunctionArn`— Le nom de ressource Amazon (ARN) utilisé pour appeler la fonction. Indique si l'appelant a spécifié un numéro de version ou un alias.
- `MemoryLimitInMB` – Quantité de mémoire allouée à la fonction.
- `AwsRequestId` – Identifiant de la demande d'invocation.
- `LogGroupName` – Groupe de journaux pour la fonction.
- `LogStreamName` – Flux de journal de l'instance de fonction.
- `RemainingTime` – Nombre de millisecondes restant avant l'expiration de l'exécution.
- `Identity` – (applications mobiles) Informations sur l'identité Amazon Cognito qui a autorisé la demande.
- `ClientContext` – (applications mobiles) Contexte client fourni à Lambda par l'application client.
- `Logger` – Objet enregistreur pour la fonction.

L'extrait de PowerShell code suivant montre une fonction de gestion simple qui imprime certaines informations contextuelles.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

# Enregistrez et surveillez les fonctions Powershell Lambda

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et envoie les journaux à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des informations sur chaque invocation au flux de journaux et transmet les journaux et autres sorties provenant du code de votre fonction. Pour de plus amples informations, veuillez consulter Utilisation des CloudWatch journaux et journaux avec Lambda.

Cette page explique comment générer une sortie de journal à partir du code de votre fonction Lambda et comment accéder aux journaux à l'aide de la AWS Command Line Interface console Lambda ou de la console. CloudWatch

Sections


- Création d'une fonction qui renvoie des journaux
- Afficher les journaux dans la console Lambda
- Afficher les journaux dans la CloudWatch console
- Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)
- Suppression de journaux


## Création d'une fonction qui renvoie des journaux

Pour générer des journaux à partir de votre code de fonction, vous pouvez utiliser des applets de commande sur Microsoft. PowerShell.Utility, ou tout module de journalisation qui écrit dans `stdout` ou`stderr`. L'exemple suivant utilise `Write-Host`.

Example function/Handler.ps1 – Journalisation

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host `## Environment variables
Write-Host AWS_LAMBDA_FUNCTION_VERSION=$Env:AWS_LAMBDA_FUNCTION_VERSION
Write-Host AWS_LAMBDA_LOG_GROUP_NAME=$Env:AWS_LAMBDA_LOG_GROUP_NAME
Write-Host AWS_LAMBDA_LOG_STREAM_NAME=$Env:AWS_LAMBDA_LOG_STREAM_NAME
Write-Host AWS_EXECUTION_ENV=$Env:AWS_EXECUTION_ENV
Write-Host AWS_LAMBDA_FUNCTION_NAME=$Env:AWS_LAMBDA_FUNCTION_NAME
Write-Host PATH=$Env:PATH
Write-Host `## Event
```

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 3)
```

Example format des journaux

```
START RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Version: $LATEST
Importing module ./Modules/AWSPowerShell.NetCore/3.3.618.0/AWSPowerShell.NetCore.psd1
[Information] - ## Environment variables
[Information] - AWS_LAMBDA_FUNCTION_VERSION=$LATEST
[Information] - AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/blank-powershell-
function-18CIXMPLHFAJJ
[Information] - AWS_LAMBDA_LOG_STREAM_NAME=2020/04/01/
[$LATEST]53c5xmpl52d64ed3a744724d9c201089
[Information] - AWS_EXECUTION_ENV=AWS_Lambda_dotnet6_powershell_1.0.0
[Information] - AWS_LAMBDA_FUNCTION_NAME=blank-powershell-function-18CIXMPLHFAJJ
[Information] - PATH=/var/lang/bin:/usr/local/bin:/usr/bin/:/bin:/opt/bin
[Information] - ## Event
[Information] -
{
    "Records": [
        {
            "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1523232000000",
                "SenderId": "123456789012",
                "ApproximateFirstReceiveTimestamp": "1523232000001"
            },
            ...
END RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed
REPORT RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Duration: 3906.38 ms Billed
 Duration: 4000 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19
 ms
XRAY TraceId: 1-5e843da6-733cxmple7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled:
 true
```

Le. NETruntime enregistre les `REPORT` lignes `START``END`, et pour chaque appel. La ligne de rapport fournit les détails suivants.

`REPORT`champs de données de ligne

- RequestId— L'identifiant de demande unique pour l'invocation.

- Duration – Temps que la méthode de gestion du gestionnaire de votre fonction a consacré au traitement de l'événement.
- Billed Duration : temps facturé pour l'invocation.
- Memory Size – Quantité de mémoire allouée à la fonction.
- Max Memory Used – Quantité de mémoire utilisée par la fonction. Lorsque les invocations partagent un environnement d'exécution, Lambda indique la mémoire maximale utilisée pour toutes les invocations. Ce comportement peut entraîner une valeur signalée plus élevée que prévu.
- Init Duration : pour la première requête servie, temps qu'il a pris à l'exécution charger la fonction et exécuter le code en dehors de la méthode du gestionnaire.
- XRAY TraceId— Pour les demandes suivies, l'ID de AWS X-Ray trace.
- SegmentId— Pour les demandes tracées, l'identifiant du segment X-Ray.
- Sampled – Pour les demandes suivies, résultat de l'échantillonnage.

## Afficher les journaux dans la console Lambda

Vous pouvez utiliser la console Lambda pour afficher la sortie du journal après avoir invoqué une fonction Lambda.

Si votre code peut être testé à partir de l'éditeur Code intégré, vous trouverez les journaux dans les résultats d'exécution. Lorsque vous utilisez la fonctionnalité de test de console pour invoquer une fonction, vous trouverez Sortie du journal dans la section Détails.

## Afficher les journaux dans la CloudWatch console

Vous pouvez utiliser la CloudWatch console Amazon pour consulter les journaux de toutes les invocations de fonctions Lambda.

Pour afficher les journaux sur la CloudWatch console

1. Ouvrez la page Groupes de journaux sur la CloudWatch console.
2. Choisissez le groupe de journaux pour votre fonction (/aws/lambda/***your-function-name***).
3. Choisissez un flux de journaux.

Chaque flux de journal correspond à une instance de votre fonction. Un flux de journaux apparaît lorsque vous mettez à jour votre fonction Lambda et lorsque des instances supplémentaires sont

créées pour traiter plusieurs invocations simultanées. Pour trouver les journaux d'un appel spécifique, nous vous recommandons d'instrumenter votre fonction avec. AWS X-Ray X-Ray enregistre des détails sur la demande et le flux de journaux dans le suivi.

## Afficher les journaux à l'aide de AWS Command Line Interface (AWS CLI)

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
```

```
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
```

```
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
```

```
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


## Suppression de journaux

Les groupes de journaux ne sont pas supprimés automatiquement quand vous supprimez une fonction. Pour éviter de stocker des journaux indéfiniment, supprimez le groupe de journaux ou configurez une période de conservation à l'issue de laquelle les journaux sont supprimés automatiquement.

# Création de fonctions Lambda avec Rust

Étant donné que Rust se compile en code natif, vous n'avez pas besoin d'une d'exécution dédiée pour exécuter du code Rust sur Lambda. Au lieu de cela, utilisez le client d'exécution Rust pour créer votre projet localement, puis déployez-le sur Lambda à l'aide de l'exécution `provided.al2023` ou `provided.al2`. Lorsque vous utilisez `provided.al2023` ou `provided.al2`, Lambda maintient automatiquement le système d'exploitation à jour avec les derniers correctifs.

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Outils et bibliothèques pour Rust

- Kit AWS SDK pour Rust: Le AWS SDK pour Rust fournit des API Rust pour interagir avec les services d'infrastructure Amazon Web Services.
- Client d'exécution Rust pour Lambda : le client d'exécution Rust est un package expérimental. Il est sujet à des modifications et n'est pas recommandé pour la production.
- Cargo Lambda : cette bibliothèque fournit une application en ligne de commande pour travailler avec des fonctions Lambda créées avec Rust.
- Lambda HTTP : cette bibliothèque fournit un wrapper pour travailler avec des événements HTTP.
- Extension Lambda : cette bibliothèque fournit un support pour écrire des extensions Lambda avec Rust.
- AWS Lambda Événements : cette bibliothèque fournit des définitions de type pour les intégrations de sources d'événements courantes.

Exemples d'applications Lambda pour Rust

- Fonction Lambda de base : une fonction Rust qui montre comment traiter des événements de base.
- Fonction Lambda avec gestion des erreurs : une fonction Rust qui montre comment gérer des erreurs Rust personnalisées dans Lambda.

- Fonction Lambda avec ressources partagées : un projet Rust qui initialise les ressources partagées avant de créer la fonction Lambda.
- Événements HTTP Lambda : une fonction Rust qui gère les événements HTTP.
- Événements HTTP Lambda avec en-têtes CORS : une fonction Rust qui utilise Tower pour injecter des en-têtes CORS.
- API REST Lambda : une API REST qui utilise Axum et Diesel pour la connexion à une base de données PostgreSQL.
- Démo Rust sans serveur : un projet Rust qui montre l'utilisation des bibliothèques Rust de Lambda, de la journalisation, des variables d'environnement et du AWS SDK.
- Extension Lambda de base : une extension Rust qui montre comment traiter les événements d’extension de base.
- Extension Lambda Logs Amazon Data Firehose : une extension Rust qui montre comment envoyer des journaux Lambda à Firehose.

Rubriques

- Définir le gestionnaire de fonctions Lambda dans Rust
- Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Rust
- Traitement des événements HTTP avec Rust
- Déploiement de fonctions Lambda Rust avec des archives de fichiers .zip
- Enregistrez et surveillez les fonctions de Rust Lambda

# Définir le gestionnaire de fonctions Lambda dans Rust

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Le gestionnaire de fonction Lambda est la méthode dans votre code de fonction qui traite les événements. Lorsque votre fonction est invoquée, Lambda exécute la méthode du gestionnaire. Votre fonction s'exécute jusqu'à ce que le gestionnaire renvoie une réponse, se ferme ou expire.

Rubriques

- Notions de base sur le gestionnaire Rust
- Utilisation de l'état partagé
- Bonnes pratiques en matière de code pour les fonctions Rust Lambda

## Notions de base sur le gestionnaire Rust

Écrivez le code de votre fonction Lambda sous forme d'un exécutable Rust. Mettez en œuvre le code de la fonction du gestionnaire et une fonction principale et incluez les éléments suivants :

- La caisse lambda_runtime de crates.io, qui met en œuvre le modèle de programmation Lambda pour Rust.
- Incluez Tokio dans vos dépendances. Le client d'exécution Rust pour Lambda utilise Tokio pour gérer les appels asynchrones.

Example — Gestionnaire Rust qui traite les événements JSON

L'exemple suivant utilise la caisse serde_json pour traiter les événements de base : JSON

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
 let payload = event.payload;
 let first_name = payload["firstName"].as_str().unwrap_or("world");
 Ok(json!({ "message": format!("Hello, {first_name}!") }))
```

```
}

#[tokio::main]
async fn main() -> Result<(), Error> {
 lambda_runtime::run(service_fn(handler)).await
}
```

Notez ce qui suit :

- `use` : importe les bibliothèques dont votre fonction Lambda a besoin.
- `async fn main` : le point d'entrée qui exécute le code de la fonction Lambda. Le client d'exécution Rust utilise Tokio en tant qu'exécution asynchrone, vous devez donc annoter la fonction principale avec `#[tokio::main]`.
- `async fn handler(event: LambdaEvent<Value>) -> Result<Value,Error>` : il s'agit de la signature du gestionnaire Lambda. Elle inclut le code qui s'exécute lorsque la fonction est invoquée.
  - `LambdaEvent<Value>` : c'est un type générique qui décrit l'événement reçu par l'exécution Lambda ainsi que le contexte de la fonction Lambda.
  - `Result<Value, Error>` : la fonction renvoie un type `Result`. Si la fonction fonctionne correctement, le résultat est une JSON valeur. Dans le cas contraire, le résultat est une erreur.

## Utilisation de l'état partagé

Vous pouvez déclarer des variables partagées indépendantes du code du gestionnaire de votre fonction Lambda. Ces variables peuvent vous aider à charger des informations d'état pendant le Phase d'initialisation, avant que votre fonction ne reçoive des événements.

Example – Partage du client Amazon S3 entre les instances de fonction

Notez ce qui suit :

- `use aws_sdk_s3::Client` : cet exemple vous demande d'ajouter `aws-sdk-s3 = "0.26.0"` à la liste des dépendances dans votre fichier `Cargo.toml`.
- `aws_config::from_env` : cet exemple vous demande d'ajouter `aws-config = "0.55.1"` à la liste des dépendances dans votre fichier `Cargo.toml`.

```
use aws_sdk_s3::Client;
```

```
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde::{Deserialize, Serialize};

#[derive(Deserialize)]
struct Request {
    bucket: String,
}

#[derive(Serialize)]
struct Response {
    keys: Vec<String>,
}

async fn handler(client: &Client, event: LambdaEvent<Request>) -> Result<Response,
 Error> {
    let bucket = event.payload.bucket;
    let objects = client.list_objects_v2().bucket(bucket).send().await?;
    let keys = objects
        .contents()
        .map(|s| s.iter().flat_map(|o| o.key().map(String::from)).collect())
        .unwrap_or_default();
    Ok(Response { keys })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    let shared_config = aws_config::from_env().load().await;
    let client = Client::new(&shared_config);
    let shared_client = &client;
    lambda_runtime::run(service_fn(move |event: LambdaEvent<Request>| async move {
        handler(&shared_client, event).await
    }))
    .await
}
```

## Bonnes pratiques en matière de code pour les fonctions Rust Lambda

Respectez les directives de la liste suivante pour utiliser les meilleures pratiques de codage lors de la création de vos fonctions Lambda :

- Séparez le gestionnaire Lambda de votre logique principale. Cela vous permet de créer une fonction testable plus unitaire.

- Réduisez la complexité de vos dépendances. Privilégiez les infrastructures plus simples qui se chargent rapidement au démarrage de l'environnement d'exécution.
- Réduisez la taille de votre package de déploiement selon ses besoins d'exécution. Cela contribue à réduire le temps nécessaire au téléchargement et à la décompression de votre package de déploiement avant l'invocation.

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.
- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à

jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.

- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

# Utilisation de l'objet de contexte Lambda pour récupérer les informations de la fonction Rust

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Lorsque Lambda exécute votre fonction, il ajoute un objet de contexte à LambdaEvent celui que le gestionnaire reçoit. Cet objet fournit les propriétés avec des informations sur l'appel, la fonction et l'environnement d'exécution.

Propriétés du contexte

- `request_id`: ID de AWS demande généré par le service Lambda.
- `deadline` : le délai d'exécution de l'invocation en cours en millisecondes.
- `invoked_function_arn`: le nom de ressource Amazon (ARN) de la fonction Lambda invoquée.
- `xray_trace_id`: ID de AWS X-Ray trace pour l'appel en cours.
- `client_content`: l'objet contextuel du client envoyé par le AWS mobileSDK. Ce champ est vide sauf si la fonction est invoquée via un AWS mobileSDK.
- `identity` : l'identité Amazon Cognito qui a invoqué la fonction. Ce champ est vide sauf si la demande d'invocation adressée au APIs Lambda a été faite à l' AWS aide d'informations d'identification émises par les groupes d'identités Amazon Cognito.
- `env_config` : la configuration de la fonction Lambda à partir des variables d'environnement locales. Cette propriété comprend des informations telles que le nom de la fonction, l'allocation de mémoire, la version et les flux de journaux.

## Accès aux informations du contexte d'appel

Les fonctions Lambda ont accès aux métadonnées sur leur environnement et la demande d'appel. L'objet `LambaEvent` que votre gestionnaire de fonction reçoit comprend les métadonnées `context` :

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};
```

```
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    let invoked_function_arn = event.context.invoked_function_arn;
    Ok(json!({ "message": format!("Hello, this is function
 {invoked_function_arn}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

# Traitement des événements HTTP avec Rust

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Les API Amazon API Gateway, les équilibreurs de charge d'application et les URL des fonctions Lambda peuvent envoyer des événements HTTP à Lambda. Vous pouvez utiliser la caisse aws_lambda_events de crates.io pour traiter les événements provenant de ces sources.

Example – Gestion de la demande de proxy API Gateway

Notez ce qui suit :

- `use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse}` : la caisse aws_lambda_events inclut de nombreux événements Lambda. Pour réduire le temps de compilation, utilisez les drapeaux de fonction pour activer les événements dont vous avez besoin. Exemple: `aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- `use http::HeaderMap` : cette importation nécessite d'ajouter la caisse http à vos dépendances.

```
use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse};
use http::HeaderMap;
use lambda_runtime::{service_fn, Error, LambdaEvent};

async fn handler(
    _event: LambdaEvent<ApiGatewayProxyRequest>,
) -> Result<ApiGatewayProxyResponse, Error> {
    let mut headers = HeaderMap::new();
    headers.insert("content-type", "text/html".parse().unwrap());
    let resp = ApiGatewayProxyResponse {
        status_code: 200,
        multi_value_headers: headers.clone(),
        is_base64_encoded: false,
        body: Some("Hello AWS Lambda HTTP request".into()),
        headers,
    };
```

```
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

Le client d'exécution Rust pour Lambda fournit également une abstraction sur ces types d'événements qui vous permet de travailler avec les types HTTP natifs, quel que soit le service qui envoie les événements. Le code suivant est équivalent à l'exemple précédent, et il fonctionne d'emblée avec les URL des fonctions Lambda, les Application Load Balancer et l'API Gateway.

> **Note**
>
> À la base, la caisse lambda_http utilise la caisse lambda_runtime. Vous n'avez pas besoin d'importer `lambda_runtime` séparément.

Example – Gestion des requêtes HTTP

```
use lambda_http::{service_fn, Error, IntoResponse, Request, RequestExt, Response};

async fn handler(event: Request) -> Result<impl IntoResponse, Error> {
    let resp = Response::builder()
        .status(200)
        .header("content-type", "text/html")
        .body("Hello AWS Lambda HTTP request")
        .map_err(Box::new)?;
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_http::run(service_fn(handler)).await
}
```

Pour un autre exemple d'utilisation`lambda_http`, consultez l'exemple de code http-axum sur le AWS référentiel Labs. GitHub

Exemples d'événements HTTP Lambda pour Rust

- Événements HTTP Lambda : une fonction Rust qui gère les événements HTTP.
- Événements HTTP Lambda avec en-têtes CORS : une fonction Rust qui utilise Tower pour injecter des en-têtes CORS.
- Événements HTTP Lambda avec ressources partagées : une fonction Rust qui utilise des ressources partagées initialisées avant la création du gestionnaire de fonction.

# Déploiement de fonctions Lambda Rust avec des archives de fichiers .zip

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

Cette page décrit comment compiler votre fonction Rust, puis déployer le binaire compilé à l' AWS Lambda aide de Cargo Lambda. Il montre également comment déployer le binaire compilé avec le AWS Command Line Interface et le AWS Serverless Application Model CLI.

Sections


- Prérequis
- Création de fonctions Rust sur macOS, Windows ou Linux
- Déploiement du binaire de la fonction Rust avec Cargo Lambda
- Invocation de votre fonction Rust avec Cargo Lambda


## Prérequis

- Rust
- AWS Command Line Interface (AWS CLI) version 2

## Création de fonctions Rust sur macOS, Windows ou Linux

Les étapes suivantes montrent comment créer le projet de votre première fonction Lambda avec Rust et le compiler avec Cargo Lambda.

1. Installez Cargo Lambda, une sous-commande Cargo, qui compile les fonctions Rust pour Lambda sur macOS, Windows et Linux.

   Pour installer Cargo Lambda sur tout système où Python 3 est installé, utilisez pip :

   ```
   pip3 install cargo-lambda
   ```

Pour installer Cargo Lambda sur macOS ou Linux, utilisez Homebrew :

```
brew tap cargo-lambda/cargo-lambda
brew install cargo-lambda
```

Pour installer Cargo Lambda sous Windows, utilisez Scoop :

```
scoop bucket add cargo-lambda
scoop install cargo-lambda/cargo-lambda
```

Pour d'autres options, consultez Installation dans la documentation de Cargo Lambda.

2. Créez la structure du package. Cette commande crée un code de fonction de base dans `src/main.rs`. Vous pouvez utiliser ce code pour le tester ou le remplacer par le vôtre.

```
cargo lambda new my-function
```

3. À l'intérieur du répertoire racine du package, exécutez la sous-commande build pour compiler le code de votre fonction.

```
cargo lambda build --release
```

(Facultatif) Si vous souhaitez utiliser AWS Graviton2 sur Lambda, ajoutez l'`--arm64`indicateur pour lequel compiler votre code. ARM CPUs

```
cargo lambda build --release --arm64
```

4. Avant de déployer votre fonction Rust, configurez les AWS informations d'identification sur votre machine.

```
aws configure
```

## Déploiement du binaire de la fonction Rust avec Cargo Lambda

Utilisez la sous-commande deploy pour déployer le binaire compilé vers Lambda. Cette commande crée un rôle d'exécution, puis la fonction Lambda. Pour spécifier un rôle d'exécution existant, utilisez l'indicateur --iam-role.

```
cargo lambda deploy my-function
```

## Déployer le binaire de votre fonction Rust avec le AWS CLI

Vous pouvez également déployer votre binaire avec le AWS CLI.

1. Utilisez la sous-commande build pour créer le package de déploiement .zip.

   ```
   cargo lambda build --release --output-format zip
   ```

2. Pour déployer le package .zip sur Lambda, exécutez la commande create-function.

   - Pour `--runtime`, spécifiez `provided.al2023`. Il s'agit d'un environnement d'exécution réservé au système d'exploitation. Les environnements d'exécution réservés au système d'exploitation sont utilisés pour déployer des fichiers binaires compilés et des environnements d'exécution personnalisés sur Lambda.
   - Pour`--role`, spécifiez ARN le rôle d'exécution.

   ```
   aws lambda create-function \
        --function-name my-function \
        --runtime provided.al2023 \
        --role arn:aws:iam::111122223333:role/lambda-role \
        --handler rust.handler \
        --zip-file fileb://target/lambda/my-function/bootstrap.zip
   ```

## Déployer le binaire de votre fonction Rust avec le AWS SAM CLI

Vous pouvez également déployer votre binaire avec le AWS SAM CLI.

1. Créez un AWS SAM modèle avec la définition de la ressource et de la propriété. Pour `Runtime`, spécifiez `provided.al2023`. Il s'agit d'un environnement d'exécution réservé au système d'exploitation. Les environnements d'exécution réservés au système d'exploitation sont utilisés pour déployer des fichiers binaires compilés et des environnements d'exécution personnalisés sur Lambda.

   Pour plus d'informations sur le déploiement de fonctions Lambda à l'aide AWS SAM de AWS: :Serverless : :Function dans le manuel du développeur.AWS Serverless Application Model

Example SAMdéfinition des ressources et des propriétés pour un binaire Rust

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: SAM template for Rust binaries
Resources:
  RustFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: target/lambda/my-function/
      Handler: rust.handler
      Runtime: provided.al2023
Outputs:
  RustFunction:
    Description: "Lambda Function ARN"
    Value: !GetAtt RustFunction.Arn
```

2. Utilisez la sous-commande build pour compiler la fonction.

```
cargo lambda build --release
```

3. Utilisez la commande sam deploy pour déployer la fonction vers Lambda.

```
sam deploy --guided
```

Pour plus d'informations sur la création de fonctions Rust avec le AWS SAM CLI, voir Création de fonctions Rust Lambda avec Cargo Lambda dans le manuel du développeur.AWS Serverless Application Model

## Invocation de votre fonction Rust avec Cargo Lambda

Utilisez la sous-commande invoke pour tester votre fonction avec une charge utile.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

### Invoquer votre fonction Rust avec le AWS CLI

Vous pouvez également utiliser le AWS CLI pour appeler la fonction.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"command": "Hello world"}' /tmp/out.txt
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

# Enregistrez et surveillez les fonctions de Rust Lambda

> **Note**
>
> Le client d'exécution Rust est un package expérimental. Il est susceptible d'être modifié et n'est destiné qu'à des fins d'évaluation.

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom et envoie les journaux à Amazon. CloudWatch Votre fonction Lambda est fournie avec un groupe de CloudWatch journaux Logs et un flux de journaux pour chaque instance de votre fonction. L'environnement d'exécution Lambda envoie des informations sur chaque invocation au flux de journaux et transmet les journaux et autres sorties provenant du code de votre fonction. Pour plus d'informations, consultez Utilisation des CloudWatch journaux et journaux avec Lambda. Cette page décrit comment produire des journaux à partir du code de votre fonction Lambda.

## Création d'une fonction qui écrit des journaux

Pour produire des journaux à partir du code de votre fonction, vous pouvez utiliser n'importe quelle fonction de journalisation qui écrit dans `stdout` ou `stderr`, comme la macro `println!`. L'exemple suivant utilise `println!` pour imprimer un message lorsque le gestionnaire de fonction démarre et avant qu'il ne se termine.

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    println!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    println!("Rust function responds to {}", &first_name);
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

# Implémentation de la journalisation avancée avec la caisse de traçage

Tracing est un cadre permettant d'instrumenter les programmes Rust afin de collecter des informations de diagnostic structurées et basées sur des événements. Ce cadre fournit des utilitaires pour personnaliser les niveaux et les formats de sortie des journaux, comme la création de messages de journal JSON structurés. Pour utiliser ce cadre, vous devez initialiser un `subscriber` avant de mettre en œuvre le gestionnaire de fonction. Ensuite, vous pouvez utiliser des macros de traçage comme `debug`, `info` et `error` pour spécifier le niveau de journal que vous voulez pour chaque scénario.

Example – Utilisation de la caisse Tracing

Notez ce qui suit :

- `tracing_subscriber::fmt().json()` : lorsque cette option est incluse, les journaux sont formatés en JSON. Pour utiliser cette option, vous devez inclure la fonction `json` dans la dépendance `tracing-subscriber` (par exemple, `tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]` : cette annotation génère une étendue à chaque fois que le gestionnaire est invoqué. L'étendue ajoute l'identifiant de la demande à chaque ligne de journal.
- `{ %first_name }` : cette construction ajoute le champ `first_name` à la ligne de journal où il est utilisé. La valeur de ce champ correspond à la variable du même nom.

```
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    tracing::info!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    tracing::info!({ %first_name }, "Rust function responds to event");
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt().json()
        .with_max_level(tracing::Level::INFO)
```

```
        // this needs to be set to remove duplicated information in the log.
        .with_current_span(false)
        // this needs to be set to false, otherwise ANSI color codes will
        // show up in a confusing manner in CloudWatch logs.
        .with_ansi(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        // remove the name of the function from every log entry
        .with_target(false)
        .init();
    lambda_runtime::run(service_fn(handler)).await
}
```

Lorsque cette fonction Rust est invoquée, elle imprime deux lignes de journal similaires à ce qui suit :

```
{"level":"INFO","fields":{"message":"Rust function invoked"},"spans":
[{"req_id":"45daaaa7-1a72-470c-9a62-e79860044bb5","name":"handler"}]}
{"level":"INFO","fields":{"message":"Rust function responds to
 event","first_name":"David"},"spans":[{"req_id":"45daaaa7-1a72-470c-9a62-
e79860044bb5","name":"handler"}]}
```

# Bonnes pratiques d'utilisation des AWS Lambda fonctions

Les bonnes pratiques suivantes sont recommandées pour l'utilisation d' AWS Lambda :

Rubriques


- Code de fonction
- Configuration de la fonction
- Extensibilité des fonctions
- Métriques et alarmes
- Utilisation des flux
- Bonnes pratiques de sécurité


## Code de fonction

- Tirez parti de la réutilisation de l'environnement d'exécution pour améliorer les performances de votre fonction. Initialisez les SDK clients et les connexions à la base de données en dehors du gestionnaire de fonctions et mettez en cache les actifs statiques localement dans le `/tmp` répertoire. Les invocations ultérieures traitées par la même instance de votre fonction peuvent réutiliser ces ressources. Cela permet d'économiser des coûts, tout en réduisant le temps d'exécution de la fonction.

  Pour éviter des éventuelles fuites de données entre les invocations, n'utilisez pas l'environnement d'exécution pour stocker des données utilisateur, des événements ou d'autres informations ayant un impact sur la sécurité. Si votre fonction repose sur un état réversible qui ne peut pas être stocké en mémoire dans le gestionnaire, envisagez de créer une fonction distincte ou des versions distinctes d'une fonction pour chaque utilisateur.
- Utilisez une directive keep-alive pour maintenir les connexions persistantes. Lambda purge les connexions inactives au fil du temps. Si vous tentez de réutiliser une connexion inactive lorsque vous invoquez une fonction, cela entraîne une erreur de connexion. Pour maintenir votre connexion persistante, utilisez la directive Keep-alive associée à votre environnement d'exécution. Pour obtenir un exemple, consultez Réutilisation des connexions avec Keep-Alive dans Node.js.
- Utilisez des variables d'environnement pour transmettre des paramètres opérationnels à votre fonction. Par exemple, si vous écrivez dans un compartiment Amazon S3 au lieu de coder en dur

le nom du compartiment dans lequel vous écrivez, configurez le nom du compartiment comme variable d'environnement.

- Évitez d'utiliser des invocations récursives dans votre fonction Lambda, lorsque la fonction s'invoque elle-même ou initie un processus susceptible de l'invoquer à nouveau. Cela peut entraîner un volume involontaire d'invocations de fonction et des coûts accrus. Si vous constatez un volume d'invocations involontaire, réglez la fonction réservée à la simultanéité sur `0` immédiatement afin de limiter toutes les invocations à la fonction pendant que vous mettez à jour le code.
- N'utilisez pas de code non documenté ni public APIs dans votre code de fonction Lambda. Pour les AWS Lambda environnements d'exécution gérés, Lambda applique régulièrement des mises à jour de sécurité et fonctionnelles aux applications internes de Lambda. APIs Ces API mises à jour internes peuvent être rétroincompatibles, ce qui peut entraîner des conséquences imprévues, telles que des échecs d'invocation si votre fonction dépend de ces mises à jour non publiques. APIs Voir la API référence pour une liste des documents accessibles au publicAPIs.
- Écriture du code idempotent. L'écriture de code idempotent pour vos fonctions garantit ne gestion identique des événements dupliqués. Votre code doit valider correctement les événements et gérer correctement les événements dupliqués. Pour de plus amples informations, veuillez consulterComment faire en sorte que ma fonction Lambda soit idempotente ?.

Pour connaître les meilleures pratiques de code spécifiques à chaque langue, reportez-vous aux sections suivantes :

- the section called "Bonnes pratiques en matière de code pour les fonctions Lambda de Node.js"
- the section called "Meilleures pratiques en matière de code pour les fonctions Typescript Lambda"
- the section called "Bonnes pratiques en matière de code pour les fonctions Lambda en Python"
- the section called "Bonnes pratiques en matière de code pour les fonctions Ruby Lambda"
- the section called "Bonnes pratiques en matière de code pour les fonctions Java Lambda"
- the section called "Meilleures pratiques en matière de code pour les fonctions Go Lambda"
- the section called "Meilleures pratiques en matière de code pour les fonctions Lambda C#"
- the section called "Bonnes pratiques en matière de code pour les fonctions Rust Lambda"

# Configuration de la fonction

- Le test de performance de votre fonction Lambda est une partie cruciale pour garantir que vous choisissez la configuration de taille mémoire optimale. Toute augmentation de la taille de la mémoire entraîne une augmentation équivalente de la CPU capacité disponible pour votre fonction. L'utilisation de la mémoire pour votre fonction est déterminée par appel et peut être consultée sur Amazon CloudWatch. À chaque invocation, une entrée `REPORT:` est créée, comme indiqué ci-dessous :

  ```
  REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed
   Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
  ```

  En analysant le champ `Max Memory Used:`, vous pouvez déterminer si votre fonction a besoin de plus de mémoire ou si vous avez surdimensionné la taille mémoire de votre fonction.

  Pour trouver la configuration de mémoire adaptée à vos fonctions, nous vous recommandons d'utiliser le projet open source AWS Lambda Power Tuning. Pour plus d'informations, voir Réglage AWS Lambda de l'alimentation activé GitHub.

  Pour optimiser les performances des fonctions, nous recommandons également de déployer des bibliothèques capables de tirer parti des extensions vectorielles avancées 2 (AVX2). Cela vous permet de traiter des charges de travail exigeantes, notamment l'inférence par apprentissage automatique, le traitement multimédia, le calcul haute performance (HPC), les simulations scientifiques et la modélisation financière. Pour plus d'informations, consultez la section Création de AWS Lambda fonctions plus rapides avec AVX2.
- Effectuez un test de charge de votre fonction Lambda pour déterminer une valeur de délai d’expiration optimale. Il importe d’analyser comment votre fonction s’exécute afin que vous puissiez mieux déterminer les problèmes de service de dépendance qui peuvent accroître la simultanéité de la fonction au-delà de ce que vous attendez. Cet aspect est particulièrement important quand votre fonction Lambda effectue des appels réseau aux ressources qui peuvent ne pas gérer la mise à l’échelle de Lambda.
- Utilisez les autorisations les plus restrictives lorsque vous définissez IAM des politiques. Maîtrisez les ressources et les opérations dont votre fonction Lambda a besoin et limitez le rôle d’exécution à ces autorisations. Pour plus d'informations, consultez Gestion des autorisations dans AWS Lambda.
- Familiarisez-vous avec Quotas Lambda. La taille de la charge utile, les descripteurs de fichiers et l'espace /tmp sont souvent ignorés lors de la détermination des limites des ressources.

- Supprimez les fonctions Lambda que vous n'utilisez plus. En procédant ainsi, les fonctions inutilisées n'interviendront plus inutilement dans la limite de taille de votre package de déploiement.
- Si vous utilisez Amazon Simple Queue Service en tant que source d'événement, assurez-vous que la valeur de la durée de l'invocation prévue de la fonction ne dépasse pas la valeur Délai de visibilité de la file d'attente. Cela s'applique à la fois à CreateFunctionet UpdateFunctionConfiguration.
  - Dans le cas de CreateFunction, le processus de création de la fonction AWS Lambda échouera.
  - Dans le cas de UpdateFunctionConfiguration, cela pourrait entraîner des appels dupliqués de la fonction.

## Extensibilité des fonctions

- Familiarisez-vous avec vos contraintes de débit en amont et en aval. Bien que les fonctions Lambda se mettent à l'échelle parfaitement, les dépendances en amont et en aval peuvent avoir des mêmes capacités de débit différentes. Si vous devez limiter le niveau d'évolutivité de votre fonction, vous pouvez configurer la simultanéité réservée pour votre fonction.
- Intégrez la tolérance à l'accélérateur. Si votre fonction synchrone est ralentie en raison d'un trafic dépassant le taux de scalabilité de Lambda, vous pouvez utiliser les stratégies suivantes pour améliorer la tolérance à l'accélérateur :
  - Utilisez les délais d'attente, les nouvelles tentatives et les interruptions avec Jitter. La mise en œuvre de ces stratégies facilite les nouvelles tentatives d'appel et permet de garantir que Lambda peut évoluer en quelques secondes afin de minimiser la limitation des utilisateurs finaux.
  - Utilisez la simultanéité provisionnée. La simultanéité provisionnée est le nombre d'environnements d'exécution préinitialisés que Lambda alloue à votre fonction. Lambda gère les demandes entrantes en utilisant la simultanéité provisionnée lorsqu'elle est disponible. Lambda peut également étendre votre fonction au-delà de votre paramètre de simultanéité provisionné si nécessaire. La configuration de la simultanéité provisionnée entraîne des frais supplémentaires pour votre compte. AWS

## Métriques et alarmes

- Utilisez Afficher les métriques des fonctions Lambda les  CloudWatch alarmes et au lieu de créer ou de mettre à jour une métrique à partir de votre code de fonction Lambda. C'est une manière beaucoup plus efficace de suivre l'état de vos fonctions Lambda, qui vous permet de détecter

de façon précoce d'éventuels problèmes dans le processus de développement. Par exemple, vous pouvez configurer une alarme basée sur la durée attendue de l'invocation de votre fonction Lambda afin de pouvoir prendre en compte les goulots d'étranglement ou les latences du code de votre fonction.

- Tirez parti de votre bibliothèque de journalisation et de vos AWS Lambda métriques et dimensions pour détecter les erreurs des applications (par exemple ERR ERRORWARNING,, etc.)
- Utiliser Détection des anomalies de coûts AWS pour détecter les activités inhabituelles sur votre compte. La détection des anomalies de coûts utilise le machine learning pour effectuer une surveillance permanent de votre coût et votre utilisation, tout en minimisant les alertes de faux positifs. La détection des anomalies de coût utilise les données provenant de AWS Cost Explorer, avec un délai pouvant aller jusqu'à 24 heures. Par conséquent, après utilisation, jusqu'à 24 heures peuvent être nécessaires pour détecter une anomalie. Afin de vous familiariser à la détection des anomalies de coûts, vous devez d'abord vous inscrire à Cost Explorer. Ensuite, accéder à la détection des anomalies des coûts.

## Utilisation des flux

- Testez différentes tailles de lot et d'enregistrement de telle sorte que la fréquence d'interrogation de chaque source d'événement soit réglée sur la vitesse à laquelle votre fonction peut exécuter sa tâche. Le CreateEventSourceMapping BatchSize paramètre contrôle le nombre maximum d'enregistrements qui peuvent être envoyés à votre fonction à chaque appel. Une taille de lot plus grande peut souvent absorber plus efficacement l'invocation sur un plus large ensemble d'enregistrements, ce qui accroît votre débit.

  Par défaut, Lambda invoque votre fonction dès que des enregistrements sont disponibles. Si le lot que Lambda lit à partir de la source d'événements ne comprend qu'un seul enregistrement, Lambda envoie un seul registre à la fonction. Pour éviter d'invoquer la fonction avec un petit nombre de registres, vous pouvez indiquer à la source d'événement de les mettre en mémoire tampon pendant 5 minutes en configurant une fenêtre de traitement par lots. Avant d'invoquer la fonction, Lambda continue de lire les registres de la source d'événements jusqu'à ce qu'il ait rassemblé un lot complet, que la fenêtre de traitement par lot expire ou que le lot atteigne la limite de charge utile de 6 Mo. Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

- Augmentez le débit de traitement du flux Kinesis en ajoutant des partitions. Un flux Kinesis est composé d'une ou plusieurs partitions. La vitesse à laquelle Lambda peut lire les données de Kinesis varie de façon linéaire en fonction du nombre de fragments. L'augmentation du nombre de partitions entraîne directement celle du nombre maximum d'invocations simultanés de fonctions Lambda, et peut accroître le débit de traitement de votre flux Kinesis. Pour plus d'informations sur la relation entre les partitions et les invocations de fonctions, consultez. the section called " Flux d'interrogation et de mise en lots" Si vous augmentez le nombre de partitions dans un flux Kinesis, assurez-vous d'avoir choisi une clé de partition appropriée (consultez Clés de partition) pour vos données, de façon à ce que les enregistrements associés se retrouvent sur les mêmes partitions et que vos données soient correctement distribuées.
- Utilisez Amazon CloudWatch on IteratorAge pour déterminer si votre flux Kinesis est en cours de traitement. Par exemple, configurez une CloudWatch alarme avec un réglage maximal de 30 000 (30 secondes).

## Bonnes pratiques de sécurité

- Surveillez votre utilisation AWS Lambda en ce qui concerne les meilleures pratiques de sécurité en utilisant AWS Security Hub. Security Hub utilise des contrôles de sécurité pour évaluer les configurations des ressources et les normes de sécurité afin de vous aider à respecter divers cadres de conformité. Pour plus d'informations sur l'utilisation de Security Hub pour évaluer les ressources Lambda, consultez la section AWS Lambda Contrôles du Guide de l' AWS Security Hub utilisateur.
- Surveillez les journaux d'activité du réseau Lambda à l'aide d'Amazon Lambda Protection GuardDuty . GuardDuty La protection Lambda vous aide à identifier les menaces de sécurité potentielles lorsque des fonctions Lambda sont invoquées dans votre. Compte AWS Par exemple, si l'une de vos fonctions interroge une adresse IP associée à une activité liée aux cryptomonnaies. GuardDuty surveille les journaux d'activité réseau générés lorsqu'une fonction

Lambda est invoquée. Pour en savoir plus, consultez la section Protection Lambda dans le guide de  GuardDuty l'utilisateur Amazon.

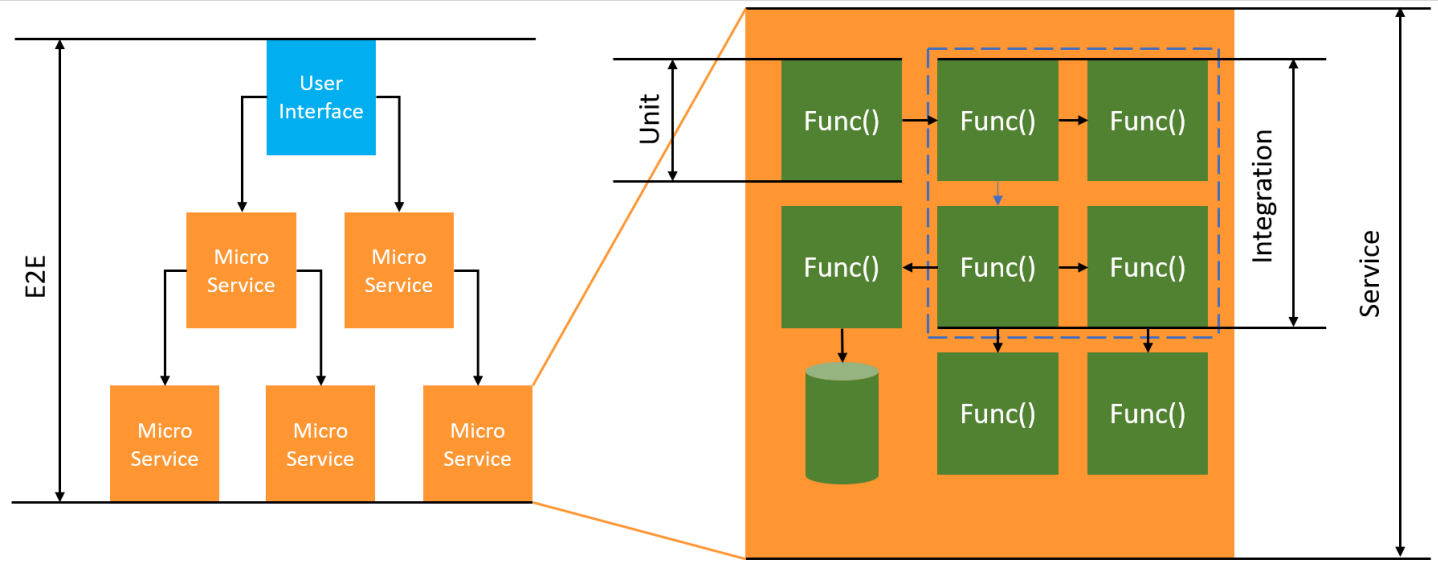
# Comment tester les fonctions et applications sans serveur

Le test des fonctions sans serveur utilise les types et techniques de tests traditionnels, mais vous devez également envisager de tester les applications sans serveur dans leur ensemble. Les tests basés sur le cloud fourniront la mesure la plus précise de la qualité de vos fonctions et de vos applications sans serveur.

Une architecture d'application sans serveur inclut des services gérés qui fournissent des fonctionnalités applicatives critiques par le biais d'APIappels. C'est pourquoi votre cycle de développement doit inclure des tests automatisés qui vérifient la fonctionnalité lorsque votre fonction et vos services interagissent.

Si vous ne créez pas de tests basés sur le cloud, vous pouvez rencontrer des problèmes en raison des différences entre votre environnement local et l'environnement déployé. Votre processus d'intégration continue doit exécuter des tests sur une série de ressources allouées dans le cloud avant de promouvoir votre code vers l'environnement de déploiement suivant, comme l'assurance qualité, la mise en place ou la production.

Poursuivez la lecture de ce petit guide pour en savoir plus sur les stratégies de test pour les applications sans serveur, ou rendez-vous sur le référentiel Serverless Test Samples pour vous plonger dans des exemples pratiques, spécifiques au langage et à l'exécution que vous avez choisis.




Pour les tests sans serveur, vous continuerez à écrire des unités, des tests d'intégration et end-to-enddes tests.

- Tests unitaires : tests exécutés sur un bloc de code isolé. Par exemple, la vérification de la logique métier permettant de calculer les frais de livraison en fonction d'un article et d'une destination donnés.
- Tests d'intégration : tests impliquant au moins deux composants ou services qui interagissent, généralement dans un environnement cloud. Par exemple, la vérification d'une fonction traite les événements d'une file d'attente.
- nd-to-end Tests E : tests qui vérifient le comportement de l'ensemble d'une application. Il s'agit par exemple de s'assurer que l'infrastructure est correctement mise en place et que les événements circulent entre les services comme prévu pour enregistrer la commande d'un client.

## Résultats commerciaux ciblés

Le test des solutions sans serveur peut nécessiter un peu plus de temps pour mettre en place des tests qui vérifient les interactions dirigées par les événements entre les services. En lisant ce guide, gardez à l'esprit les raisons commerciales pratiques suivantes :

- Améliorer la qualité de votre application
- Réduire le temps nécessaire à la création de fonctionnalités et à la correction des bogues

La qualité d'une application dépend de la réalisation de tests dans différents scénarios afin de vérifier sa fonctionnalité. Une réflexion approfondie sur les scénarios commerciaux et l'automatisation de ces tests pour les exécuter sur des services cloud amélioreront la qualité de votre application.

Les bogues logiciels et les problèmes de configuration ont le moins d'impact sur les coûts et le calendrier lorsqu'ils sont détectés au cours d'un cycle de développement itératif. Si des problèmes ne sont pas détectés au cours du développement, leur détection et leur résolution en production exigent davantage d'efforts de la part d'un plus grand nombre de personnes.

Une stratégie de test sans serveur bien planifiée augmentera la qualité des logiciels et améliorera le temps d'itération en vérifiant que vos fonctions Lambda et vos applications fonctionnent comme prévu dans un environnement cloud.

## Que faut-il tester ?

Nous vous recommandons d'adopter une stratégie de test qui évalue les comportements des services gérés, la configuration du cloud, les politiques de sécurité et l'intégration à votre code afin d'améliorer la qualité du logiciel. Les tests de comportement, également connus sous le nom de tests

en boîte noire, permettent de vérifier qu'un système fonctionne comme prévu sans en connaître tous les rouages.

- Exécutez des tests unitaires pour vérifier la logique métier au sein des fonctions Lambda.
- Vérifiez que les services intégrés sont réellement invoqués et que les paramètres d'entrée sont corrects.
- Vérifiez qu'un événement passe par tous les services attendus end-to-end dans un flux de travail.

Dans les architectures traditionnelles basées sur des serveurs, les équipes définissent souvent une portée de test afin d'inclure uniquement le code qui s'exécute sur le serveur d'application. Les autres composants, services ou dépendances sont souvent considérés comme externes et hors de portée des tests.

Les applications sans serveur sont souvent composées de petites unités de travail, telles que les fonctions Lambda qui récupèrent des produits d'une base de données, traitent des éléments d'une file d'attente ou redimensionnent une image stockée. Chaque composant s'exécute dans son propre environnement. Les équipes seront probablement responsables de plusieurs de ces petites unités au sein d'une même application.

Certaines fonctionnalités de l'application peuvent être entièrement déléguées à des services gérés tels qu'Amazon S3, ou créées sans utiliser de code développé en interne. Il n'est pas nécessaire de tester ces services gérés, mais vous devez tester l'intégration avec ces services.

## Comment tester le sans serveur

Vous savez probablement comment tester des applications déployées localement : vous rédigez des tests qui s'exécutent sur du code fonctionnant entièrement sur votre système d'exploitation de bureau ou à l'intérieur de conteneurs. Par exemple, vous pouvez invoquer un composant de service Web local avec une requête, puis faire des assertions sur la réponse.

Les solutions sans serveur sont conçues à partir du code de votre fonction et de services gérés basés sur le cloud, tels que les files d'attente, les bases de données, les bus d'événements et les systèmes de messagerie. Ces composants sont tous connectés par le biais d'une architecture orientée événements, dans laquelle les messages, appelés événements, circulent d'une ressource à l'autre. Ces interactions peuvent être synchrones, par exemple lorsqu'un service Web renvoie des résultats immédiatement, ou être asynchrones qui se terminent ultérieurement, comme le placement d'éléments dans une file d'attente ou le démarrage d'une étape du flux de travail. Votre stratégie de

test doit inclure les deux scénarios et tester les interactions entre les services. Pour les interactions asynchrones, vous devrez peut-être détecter les effets secondaires dans les composants en aval qui peuvent ne pas être immédiatement observables.

Il n'est pas pratique de répliquer l'ensemble d'un environnement cloud, y compris les files d'attente, les tables de base de données, les bus d'événements, les politiques de sécurité, etc. Vous rencontrerez inévitablement des problèmes en raison des différences entre votre environnement local et vos environnements déployés dans le cloud. Les variations entre vos environnements augmenteront le temps nécessaire à la reproduction et à la correction des bogues.

Dans les applications sans serveur, les composants de l'architecture existent généralement à part entière dans le cloud. Il est donc nécessaire de les tester par rapport au code et aux services du cloud pour développer des fonctionnalités et corriger les bogues.

## Techniques de test

En réalité, votre stratégie de test comprendra probablement une combinaison de techniques visant à améliorer la qualité de vos solutions. Vous utiliserez des tests interactifs rapides pour déboguer les fonctions de la console, des tests unitaires automatisés pour vérifier la logique métier isolée, la vérification des appels vers des services externes à l'aide de simulations, et des tests occasionnels sur des émulateurs imitant un service.

- Tests dans le cloud : vous déployez l'infrastructure et le code à tester avec des services réels, des politiques de sécurité, des configurations et des paramètres spécifiques à l'infrastructure. Les tests basés sur le cloud fournissent la mesure la plus précise de la qualité de votre code.

  Le débogage d'une fonction dans la console est un moyen rapide de la tester dans le cloud. Vous pouvez choisir parmi une bibliothèque d'exemples d'événements de test ou créer un événement personnalisé pour tester une fonction de manière isolée. Vous pouvez également partager des événements de test via la console avec votre équipe.

  Pour automatiser les tests au cours du cycle de vie du développement et de génération, vous devez effectuer des tests en dehors de la console. Consultez les sections de ce guide relatives aux tests spécifiques à chaque langue pour connaître les stratégies et les ressources d'automatisation.
- Tests à l'aide de simulations (également appelés faux) : les simulations sont des objets de votre code qui simulent et remplacent un service externe. Les simulations fournissent un comportement prédéfini pour vérifier les appels de service et les paramètres. Un faux est une implémentation fictive qui utilise des raccourcis pour simplifier ou améliorer les performances. Par exemple, un

faux objet d'accès aux données peut renvoyer des données depuis un entrepôt de données en mémoire. Les simulations peuvent imiter et simplifier des dépendances complexes, mais peuvent également conduire à d'autres simulations afin de remplacer les dépendances imbriquées.
- Tests à l'aide d'émulateurs : vous pouvez configurer des applications (parfois provenant d'un tiers) pour imiter un service cloud dans votre environnement local. La vitesse est leur force, mais la configuration et la parité avec les services de production sont leur point faible. Utilisez les émulateurs avec parcimonie.

## Tests dans le cloud

Les tests dans le cloud sont utiles pour toutes les phases de test, y compris les tests unitaires, les tests d'intégration et les end-to-end tests. Lorsque vous exécutez des tests sur du code basé sur le cloud qui interagit également avec des services basés sur le cloud, vous obtenez la mesure la plus précise de la qualité de votre code.

Un moyen pratique d'exécuter une fonction Lambda dans le cloud consiste à utiliser un événement de test dans la AWS Management Console. Un événement de test est une JSON entrée dans votre fonction. Si votre fonction ne nécessite aucune saisie, l'événement peut être un JSON document vide`({})`. La console fournit des exemples d'événements pour diverses intégrations de services. Après avoir créé un événement dans la console, vous pouvez également le partager avec votre équipe pour faciliter les tests et les rendre plus cohérents.

Découvrez comment déboguer un exemple de fonction dans la console.

> **Note**
>
> Bien que l'exécution de fonctions dans la console soit un moyen rapide de déboguer, l'automatisation de vos cycles de test est essentielle pour améliorer la qualité des applications et la vitesse de développement.

Des exemples d'automatisation des tests sont disponibles dans le référentiel Serverless Test Samples. La ligne de commande suivante exécute un exemple de test d'intégration Python automatique :

```
python -m pytest -s tests/integration -v
```

Bien que le test soit exécuté localement, il interagit avec des ressources basées dans le cloud. Ces ressources ont été déployées à l'aide de l'outil de ligne de AWS SAM commande AWS Serverless

Application Model and. Le code de test récupère d'abord les sorties de la pile déployée, qui incluent le API point de terminaison, la fonction ARN et le rôle de sécurité. Ensuite, le test envoie une demande au API point de terminaison, qui répond par une liste de compartiments Amazon S3. Ce test s'exécute entièrement sur des ressources basées sur le cloud afin de vérifier que ces ressources sont déployées, sécurisées et qu'elles fonctionnent comme prévu.

```
========================= test session starts ==========================
      platform darwin -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0
      -- /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-lambda/
venv/bin/python
      cachedir: .pytest_cache
      rootdir: /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-
lambda
      plugins: mock-3.10.0
      collected 1 item


      tests/integration/test_api_gateway.py::TestApiGateway::test_api_gateway

      --> Stack outputs:

        HelloWorldApi
        = https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
        > API Gateway endpoint URL for Prod stage for Hello World function

        PythonTestDemo
        = arn:aws:lambda:us-east-2:123456789012:function:testing-apigw-lambda-
PythonTestDemo-iSij8evaTdxl
        > Hello World Lambda Function ARN

        PythonTestDemoIamRole
        = arn:aws:iam::123456789012:role/testing-apigw-lambda-PythonTestDemoRole-
IZELQQ9MG4HQ
        > Implicit IAM Role created for Hello World function

      --> Found API endpoint for "testing-apigw-lambda" stack...
      --> https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
      API Gateway response:
      amplify-dev-123456789-deployment|myapp-prod-p-loggingbucket-123456|s3-java-
bucket-123456789
      PASSED

      ========================= 1 passed in 1.53s ==========================
```

Pour le développement d'applications natives cloud, les tests dans le cloud offrent les avantages suivants :

- Vous pouvez tester tous les services disponibles.
- Vous utilisez toujours les valeurs de service APIs et de retour les plus récentes.
- Un environnement de test dans le cloud ressemble beaucoup à votre environnement de production.
- Les tests peuvent porter sur les politiques de sécurité, les quotas de service, les configurations et les paramètres spécifiques à l'infrastructure.
- Chaque développeur peut rapidement créer un ou plusieurs environnements de test dans le cloud.
- Les tests dans le cloud augmentent la confiance dans le fait que votre code fonctionnera correctement en production.

Les tests dans le cloud présentent certains inconvénients. L'inconvénient le plus évident des tests dans le cloud est que les déploiements vers des environnements cloud prennent généralement plus de temps que les déploiements vers des environnements de bureau locaux.

Heureusement, des outils tels que AWS Serverless Application Model (AWS SAM) Accelerate, le mode veille du AWS Cloud Development Kit (AWS CDK) et SST(tierce partie) réduisent la latence associée aux itérations de déploiement dans le cloud. Ces outils peuvent surveiller votre infrastructure et votre code et déploient automatiquement des mises à jour incrémentielles dans votre environnement cloud.

> **Note**
>
> Découvrez comment créer une infrastructure sous forme de code dans le Guide du développeur sans serveur pour en savoir plus sur AWS Serverless Application Model AWS CloudFormation, et AWS Cloud Development Kit (AWS CDK).

Contrairement aux tests locaux, les tests dans le cloud nécessitent des ressources supplémentaires qui peuvent entraîner des coûts de service. La création d'environnements de test isolés peut alourdir la charge de travail de vos DevOps équipes, en particulier dans les organisations où les comptes et l'infrastructure sont soumis à des contrôles stricts. Néanmoins, lorsque l'on travaille avec des scénarios d'infrastructure complexes, le coût en temps de développement pour mettre en place et maintenir un environnement local complexe peut être similaire (voire plus coûteux) que l'utilisation

d'environnements de test jetables créés avec des outils d'automatisation d'infrastructure en tant que code.

Même en tenant compte de ces considérations, les tests dans le cloud restent le meilleur moyen de garantir la qualité de vos solutions sans serveur.

## Tester avec des simulations

Les tests avec des simulations sont une technique qui consiste à créer des objets de remplacement dans votre code afin de simuler le comportement d'un service cloud.

Par exemple, vous pouvez écrire un test qui utilise une maquette du service Amazon S3 qui renvoie une réponse spécifique chaque fois que la CreateObjectméthode est appelée. Lorsqu'un test s'exécute, la simulation renvoie cette réponse programmée sans appeler Amazon S3 ni aucun autre point de terminaison de service.

Les objets simulés sont souvent générés par un cadre simulé afin de réduire les efforts de développement. Certains frameworks fictifs sont génériques tandis que d'autres sont conçus spécifiquement pour AWS SDKs, comme Moto, une bibliothèque Python permettant de simuler AWS des services et des ressources.

Il convient de noter que les objets simulés diffèrent des émulateurs dans la mesure où les simulations sont généralement créées ou configurées par un développeur dans le cadre du code de test, alors que les émulateurs sont des applications autonomes qui exposent les fonctionnalités de la même manière que les systèmes qu'ils émulent.

Les avantages de l'utilisation de simulations incluent les avantages suivants :

- Mocks peut simuler des services tiers échappant au contrôle de votre application, tels que APIs des fournisseurs de logiciels en tant que service (SaaS), sans avoir besoin d'un accès direct à ces services.
- Les simulations sont utiles pour tester les conditions d'échec, en particulier lorsque ces conditions sont difficiles à simuler, comme dans le cas d'une panne de service.
- Une fois configurée, la simulation permet d'effectuer des tests locaux rapides.
- Les simulations peuvent fournir un comportement de substitution pour pratiquement n'importe quel type d'objet. Les stratégies de simulations peuvent donc couvrir une plus grande variété de services que les émulateurs.
- Lorsque de nouvelles fonctionnalités ou de nouveaux comportements sont disponibles, les tests simulés permettent de réagir plus rapidement. En utilisant un framework de simulation générique,

vous pouvez simuler de nouvelles fonctionnalités dès que les mises à jour AWS SDK seront disponibles.

Les tests simulés présentent les inconvénients suivants :

- Les simulations nécessitent généralement un effort d'installation et de configuration non négligeable, en particulier lorsqu'il s'agit de déterminer les valeurs de retour de différents services afin de simuler correctement les réponses.
- Les simulations sont écrites, configurées et doivent être mises à jour par les développeurs, ce qui accroît leurs responsabilités.
- Il se peut que vous deviez avoir accès au cloud afin de comprendre les valeurs des services APIs et de les rentabiliser.
- Les simulations peuvent être difficiles à entretenir. Lorsque les API signatures cloud simulées changent ou que les schémas de valeur de retour évoluent, vous devez mettre à jour vos maquettes. Les maquettes nécessitent également des mises à jour si vous étendez la logique de votre application pour passer des appels à de nouvelles APIs applications.
- Les tests utilisant des simulations peuvent réussir dans les environnements de bureau, mais échouer dans le cloud. Les résultats peuvent ne pas correspondre aux résultats actuelsAPI. La configuration du service et les quotas ne peuvent pas être testés.
- Les frameworks fictifs sont limités lorsqu'il s'agit de tester ou de détecter les limites de quotas ou de politiques d' AWS Identity and Access Management (IAM). Bien que les simulations permettent de mieux simuler l'échec d'une autorisation ou le dépassement d'un quota, les tests ne peuvent pas déterminer le résultat qui se produira réellement dans un environnement de production.

## Tester avec émulation

Les émulateurs sont généralement une application exécutée localement qui imite un service de production AWS .

Les émulateurs sont APIs similaires à leurs homologues du cloud et fournissent des valeurs de retour similaires. Ils peuvent également simuler des changements d'état initiés par API des appels. Par exemple, vous pouvez AWS SAM exécuter une fonction avec AWS SAM local pour émuler le service Lambda afin de pouvoir appeler rapidement une fonction. Pour plus d'informations, consultez AWS SAM local dans le Guide du développeur AWS Serverless Application Model .

Les avantages des tests avec émulateurs sont les suivants :

- Les émulateurs peuvent faciliter les itérations de développement et les tests locaux rapides.

- Les émulateurs fournissent un environnement familier aux développeurs habitués à développer du code dans un environnement local. Par exemple, si vous êtes habitué au développement d'une application à plusieurs niveaux, vous pouvez disposer d'un moteur de base de données et d'un serveur Web, similaires à ceux qui fonctionnent en production, exécutés sur votre ordinateur local afin de fournir une capacité de test rapide, locale et isolée.
- Les émulateurs ne nécessitent aucune modification de l'infrastructure cloud (telle que les comptes cloud des développeurs). Ils sont donc faciles à implémenter avec les modèles de test existants.

Les tests avec des émulateurs présentent les inconvénients suivants :

- Les émulateurs peuvent être difficiles à mettre en place et à répliquer, en particulier lorsqu'ils sont utilisés dans des pipelines CI/CD. Cela peut accroître la charge de travail du personnel informatique ou des développeurs qui gèrent leurs propres logiciels.
- Fonctionnalités émulées et APIs généralement en retard par rapport aux mises à jour de service. Cela peut entraîner des erreurs car le code testé ne correspond pas au code réel API et entrave l'adoption de nouvelles fonctionnalités.
- Les émulateurs nécessitent une assistance, des mises à jour, des corrections de bogues et des améliorations de la parité des fonctions. La responsabilité en incombe à l'auteur de l'émulateur, qui peut être une société tierce.
- Les tests qui s'appuient sur des émulateurs peuvent donner des résultats positifs au niveau local, mais échouer dans le cloud en raison des politiques de sécurité de la production, des configurations interservices ou du dépassement des quotas Lambda.
- De nombreux AWS services ne disposent pas d'émulateurs. Si vous comptez sur l'émulation, vous risquez de ne pas disposer d'une option de test satisfaisante pour certaines parties de votre application.

## Bonnes pratiques

Les sections suivantes fournissent des recommandations pour réussir les tests d'applications sans serveur.

Vous trouverez des exemples pratiques de tests et d'automatisation des tests dans le référentiel Serverless Test Samples.

## Prioriser les tests dans le cloud

Les tests dans le cloud fournissent la couverture de test la plus fiable, la plus précise et la plus complète. La réalisation de tests dans le contexte du cloud permettra de tester de manière exhaustive non seulement la logique métier, mais également les politiques de sécurité, les configurations de service, les quotas, ainsi que les API signatures et les valeurs de retour les plus récentes.

## Structurer votre code pour le rendre testable

Simplifiez vos tests et vos fonctions Lambda en séparant le code spécifique à Lambda de votre logique métier principale.

Votre gestionnaire de fonctions Lambda doit être un adaptateur léger qui prend en charge les données des événements et ne transmet que les détails importants à votre ou vos méthodes de logique métier. Avec cette stratégie, vous pouvez envelopper des tests complets autour de votre logique métier sans vous soucier des détails spécifiques à Lambda. Vos fonctions AWS Lambda ne devraient pas nécessiter la configuration d'un environnement complexe ou d'un grand nombre de dépendances pour créer et initialiser le composant testé.

D'une manière générale, vous devez écrire un gestionnaire qui extrait et valide les données des objets d'événement et de contexte entrants, puis envoie ces données aux méthodes qui exécutent votre logique métier.

## Accélérer les boucles de rétroaction du développement

Il existe des outils et des techniques permettant d'accélérer les boucles de rétroaction du développement. Par exemple, les modes AWS SAMAccelerate et AWS CDK Watch réduisent tous deux le temps nécessaire à la mise à jour des environnements cloud.

Les exemples du référentiel GitHub Serverless Test Samples explorent certaines de ces techniques.

Nous vous recommandons également de créer et de tester les ressources cloud le plus tôt possible au cours du développement, et pas seulement après une vérification du contrôle du code source. Cette pratique permet d'accélérer l'exploration et l'expérimentation lors du développement de solutions. En outre, l'automatisation du déploiement à partir d'une machine de développement vous permet de découvrir plus rapidement les problèmes de configuration du cloud et de réduire les efforts inutiles liés aux mises à jour et aux processus de révision du code.

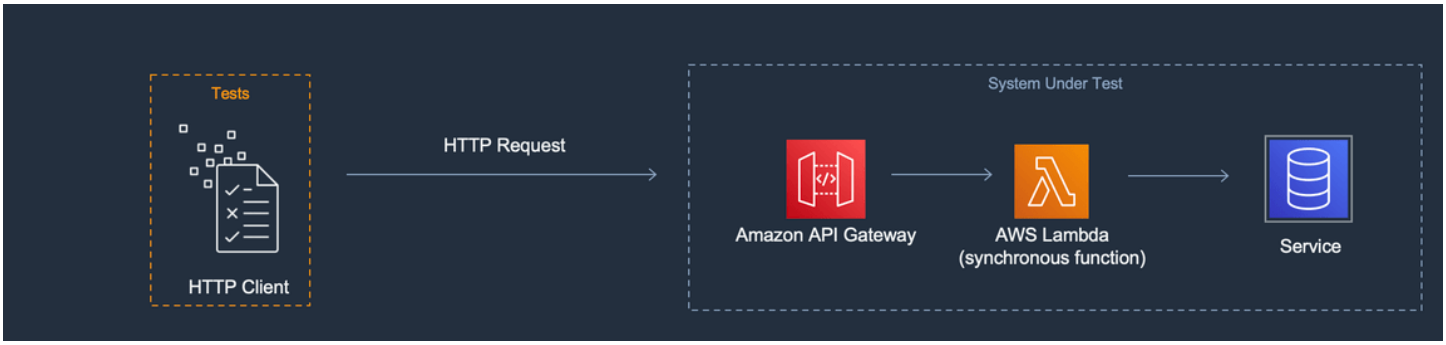
## Se concentrer sur les tests d'intégration

Lors de la création d'applications avec Lambda, il est recommandé de tester les composants ensemble.

Les tests exécutés sur deux composants architecturaux ou plus sont appelés tests d'intégration. L'objectif des tests d'intégration est de comprendre non seulement comment votre code s'exécutera entre les composants, mais aussi comment l'environnement qui héberge votre code se comportera. Les nd-to-end tests E sont des types spéciaux de tests d'intégration qui vérifient les comportements dans l'ensemble d'une application.

Pour créer des tests d'intégration, déployez votre application dans un environnement cloud. Cela peut se faire à partir d'un environnement local ou par le biais d'un pipeline CI/CD. Rédigez ensuite des tests pour exercer le système testé (SUT) et validez le comportement attendu.

Par exemple, le système testé peut être une application qui utilise API Gateway, Lambda et DynamoDB. Un test pourrait effectuer un HTTP appel synthétique vers un point de terminaison de API passerelle et valider que la réponse incluait la charge utile attendue. Ce test confirme que le code AWS Lambda est correct et que chaque service est correctement configuré pour traiter la demande, y compris IAM les autorisations entre eux. En outre, vous pouvez concevoir le test pour écrire des enregistrements de différentes tailles afin de vérifier que vos quotas de service, tels que la taille d'enregistrement maximale dans DynamoDB, sont correctement configurés.




## Créer des environnements de test isolés

Les tests dans le cloud nécessitent généralement des environnements de développement isolés, de sorte que les tests, les données et les événements ne se chevauchent pas.

L'une des approches consiste à fournir à chaque développeur un AWS compte dédié. Cela permettra d'éviter les conflits de dénomination des ressources qui peuvent survenir lorsque plusieurs développeurs travaillent dans une base de code partagée, tentent de déployer des ressources ou invoquent unAPI.

Les processus de test automatisés doivent créer des ressources portant un nom unique pour chaque pile. Par exemple, vous pouvez configurer des scripts ou des fichiers de TOML configuration afin que les commandes AWS SAM CLI sam deploy ou sam sync spécifient automatiquement une pile avec un préfixe unique.

Dans certains cas, les développeurs partagent un AWS compte. Cela peut être dû au fait que les ressources de votre pile sont coûteuses à exploiter, ou à provisionner et à configurer. Par exemple, une base de données peut être partagée pour faciliter la configuration et remplir correctement les données

Si les développeurs partagent un compte, vous devez définir des limites afin d'identifier les propriétaires et d'éliminer les chevauchements. Pour ce faire, vous pouvez préfixer les noms des piles avec l'utilisateur IDs développeur. Une autre approche populaire consiste à configurer des piles basées sur des branches de code. Avec les limites de branches, les environnements sont isolés, mais les développeurs peuvent toujours partager des ressources, telles qu'une base de données relationnelle. Cette approche est une bonne pratique lorsque les développeurs travaillent sur plusieurs branches à la fois.

Les tests dans le cloud sont utiles pour toutes les phases de test, y compris les tests unitaires, les tests d'intégration et les end-to-end tests. Il est essentiel de maintenir une isolation adéquate, mais vous souhaitez tout de même que votre environnement d'assurance qualité ressemble le plus possible à votre environnement de production. C'est pourquoi les équipes ajoutent des processus de contrôle des modifications pour les environnements d'assurance qualité.

Pour les environnements de pré-production et de production, les limites sont généralement définies au niveau du compte afin d'isoler les charges de travail de leurs bruyants voisins et de mettre en œuvre des contrôles de sécurité du moindre privilège pour protéger les données sensibles. Les charges de travail sont soumises à des quotas. Vous ne voulez pas que vos tests consomment les quotas alloués à la production (voisin bruyant) ou qu'ils aient accès aux données des clients. Les tests de charge constituent une autre activité que vous devez isoler de votre pile de production.

Dans tous les cas, les environnements doivent être configurés avec des alertes et des contrôles afin d'éviter des dépenses inutiles. Par exemple, vous pouvez limiter le type, le niveau ou la taille des ressources qui peuvent être créées, et configurer des alertes par e-mail lorsque les coûts estimés dépassent un seuil donné.

## Utiliser des simulations pour isoler la logique métier

Les cadres simulés sont un outil précieux pour écrire des tests unitaires rapides. Ils sont particulièrement utiles lorsque les tests couvrent une logique interne complexe, tels que des calculs mathématiques ou financiers ou des simulations. Recherchez les tests unitaires qui comportent un grand nombre de cas de test ou de variations d'entrée, dans lesquels ces entrées ne modifient ni le modèle ni le contenu des appels vers d'autres services cloud.

Le code couvert par des tests unitaires avec des simulations doit également être couvert par des tests dans le cloud. Cela est recommandé, car un ordinateur portable de développeur ou un environnement de machine de génération peuvent être configurés différemment d'un environnement de production dans le cloud. Par exemple, vos fonctions Lambda peuvent utiliser plus de mémoire ou de temps que ce qui est alloué lorsqu'elles sont exécutées avec certains paramètres d'entrée. Votre code peut également inclure des variables d'environnement qui ne sont pas configurées de la même manière (ou pas du tout), et les différences peuvent entraîner un comportement différent ou un échec du code.

L'avantage des simulations est moindre pour les tests d'intégration, car le niveau d'effort pour mettre en œuvre les simulations nécessaires augmente avec le nombre de points de connexion. nd-to-end Les tests E ne doivent pas utiliser de simulations, car ces tests portent généralement sur des états et une logique complexe qui ne peuvent pas être facilement simulés avec des frameworks fictifs.

Enfin, évitez d'utiliser des services cloud simulés pour valider la bonne mise en œuvre des appels de service. Au lieu de cela, faites des appels de services dans le cloud pour valider le comportement, la configuration et la mise en œuvre fonctionnelle.

## Utiliser les émulateurs avec parcimonie

Les émulateurs peuvent être pratiques dans certains cas d'utilisation, par exemple pour une équipe de développement disposant d'un accès Internet limité, peu fiable ou lent. Mais, dans la plupart des cas, choisissez d'utiliser les émulateurs avec parcimonie.

En évitant les émulateurs, vous serez en mesure de créer et d'innover avec les dernières fonctionnalités du service et à jourAPIs. Vous ne serez pas obligé d'attendre les versions des fournisseurs pour atteindre la parité des fonctionnalités. Vous réduirez vos dépenses initiales et continues liées à l'achat et à la configuration sur plusieurs systèmes de développement et machines de génération. En outre, vous éviterez le problème lié au fait que de nombreux services cloud n'ont tout simplement pas d'émulateurs disponibles. Une stratégie de test qui repose sur

l'émulation rendra impossible l'utilisation de ces services (ce qui entraînera des solutions de contournement potentiellement plus coûteuses) ou produira un code et des configurations qui ne sont pas correctement testés.

Lorsque vous utilisez l'émulation à des fins de test, vous devez tout de même effectuer des tests dans le cloud pour vérifier la configuration et tester les interactions avec les services cloud qui ne peuvent être simulés que dans un environnement émulé.

# Défis liés aux tests locaux

Lorsque vous utilisez des émulateurs et des appels simulés pour tester sur votre bureau local, vous pouvez rencontrer des incohérences de test lorsque votre code progresse d'un environnement à l'autre dans votre pipeline CI/CD. Les tests unitaires visant à valider la logique métier de votre application sur votre bureau peuvent ne pas tester avec précision certains aspects critiques des services cloud.

Les exemples suivants présentent des cas à surveiller lors de tests locaux à l'aide de simulations et d'émulateurs :

## Exemple : la fonction Lambda crée un compartiment S3

Si la logique d'une fonction Lambda dépend de la création d'un compartiment S3, un test complet doit confirmer qu'Amazon S3 a été appelé et que le compartiment a été créé avec succès.

- Dans le cas d'un test simulé, vous pouvez simuler une réponse de réussite et éventuellement ajouter un scénario de test pour gérer une réponse d'échec.
- Dans un scénario de test d'émulation, il CreateBucketAPIse peut qu'il soit appelé, mais vous devez savoir que l'identité effectuant l'appel local ne proviendra pas du service Lambda. L'identité appelante n'assumera pas un rôle de sécurité comme elle le ferait dans le cloud. C'est pourquoi une authentification par espace réservé sera utilisée à la place, éventuellement avec un rôle ou une identité d'utilisateur plus permissif qui sera différente lorsqu'elle sera exécutée dans le cloud.

Les configurations de simulation et d'émulation testeront ce que fera la fonction Lambda si elle appelle Amazon S3 ; toutefois, ces tests ne vérifieront pas que la fonction Lambda, telle que configurée, est capable de créer correctement le compartiment Amazon S3. Vous devez vous assurer que le rôle attribué à la fonction dispose d'une politique de sécurité attachée qui autorise la fonction à effectuer l'action `s3:CreateBucket`. Si ce n'est pas le cas, la fonction échouera probablement lorsqu'elle sera déployée dans un environnement en nuage.

## Exemple : la fonction Lambda traite les messages d'une file d'attente Amazon SQS

Si une SQS file d'attente Amazon est la source d'une fonction Lambda, un test complet doit vérifier que la fonction Lambda est correctement invoquée lorsqu'un message est placé dans une file d'attente.

Les tests d'émulation et les tests fictifs sont généralement configurés pour exécuter directement le code de la fonction Lambda et pour simuler l'intégration SQS Amazon en transmettant JSON une charge utile d'événement (ou un objet désérialisé) comme entrée du gestionnaire de fonctions.

Les tests locaux qui simulent l'SQSintégration Amazon testeront ce que fera la fonction Lambda lorsqu'elle est appelée par SQS Amazon avec une charge utile donnée, mais le test ne vérifiera pas qu'SQSAmazon invoquera avec succès la fonction Lambda lorsqu'elle sera déployée dans un environnement cloud.

Voici quelques exemples de problèmes de configuration que vous pourriez rencontrer avec Amazon SQS et Lambda :

- Le délai de SQS visibilité d'Amazon est trop faible, ce qui entraîne plusieurs invocations alors qu'une seule était prévue.
- Le rôle d'exécution de la fonction Lambda ne permet pas de lire les messages de la file d'attente (via `sqs:ReceiveMessage`, `sqs:DeleteMessage`, ou `sqs:GetQueueAttributes`).
- L'exemple d'événement transmis à la fonction Lambda dépasse le quota de taille des SQS messages Amazon. Par conséquent, le test n'est pas valide car Amazon ne SQS serait jamais en mesure d'envoyer un message de cette taille.

Comme le montrent ces exemples, les tests qui couvrent la logique métier mais pas les configurations entre les services cloud sont susceptibles de fournir des résultats peu fiables.

# FAQ

J'ai une fonction Lambda qui effectue des calculs et renvoie un résultat sans appeler aucun autre service. Dois-je vraiment le tester dans le cloud ?

Oui. Les fonctions Lambda comportent des paramètres de configuration susceptibles de modifier le résultat du test. Tout le code de fonction Lambda dépend du délai d'expiration et des paramètres de mémoire, ce qui peut entraîner l'échec de la fonction si ces paramètres ne sont pas définis

correctement. Les politiques Lambda permettent également la journalisation des sorties standard sur Amazon. CloudWatch Même si votre code n'appelle pas CloudWatch directement, une autorisation est nécessaire pour activer la journalisation. Cette autorisation requise ne peut pas être simulée ou émulée avec précision.

Comment les tests dans le cloud peuvent-ils faciliter les tests unitaires ? S'il se trouve dans le cloud et se connecte à d'autres ressources, ne s'agit-il pas d'un test d'intégration ?

Nous définissons les tests unitaires comme des tests qui opèrent sur des composants architecturaux de manière isolée, ce qui n'empêche pas les tests d'inclure des composants susceptibles d'appeler d'autres services ou d'utiliser certaines communications réseau.

De nombreuses applications sans serveur possèdent des composants architecturaux qui peuvent être testés de manière isolée, même dans le cloud. Par exemple, une fonction Lambda prend des entrées, traite les données et envoie un message à une file d'attente AmazonSQS. Un test unitaire de cette fonction devrait permettre de vérifier si les valeurs d'entrée entraînent la présence de certaines valeurs dans le message en file d'attente.

Prenons l'exemple d'un test écrit en utilisant le modèle organiser, agir, affirmer :

- Organiser : allouez des ressources (une file d'attente pour recevoir des messages et la fonction en cours de test).
- Agir : appelez la fonction en cours de test.
- Affirmer : récupérez le message envoyé par la fonction et validez la sortie.

Une approche de test simulé consiste à simuler la file d'attente avec un objet simulé en cours de traitement et à créer une instance en cours de traitement de la classe ou du module contenant le code de la fonction Lambda. Pendant la phase d'assertion, le message en file d'attente est récupéré dans l'objet simulé.

Dans une approche basée sur le cloud, le test créerait une SQS file d'attente Amazon aux fins du test et déploierait la fonction Lambda avec des variables d'environnement configurées pour utiliser la SQS file d'attente Amazon isolée comme destination de sortie. Après avoir exécuté la fonction Lambda, le test récupérait le message de la file d'attente AmazonSQS.

Le test basé sur le cloud exécute le même code, confirme le même comportement et valide l'exactitude fonctionnelle de l'application. Cependant, cela aurait l'avantage supplémentaire de pouvoir valider les paramètres de la fonction Lambda : le IAM rôle, les IAM politiques, le délai d'expiration et les paramètres de mémoire de la fonction.

# Prochaines étapes et ressources

Les ressources suivantes vous permettront d'en savoir plus et de découvrir des exemples pratiques de tests.

Exemples d'implémentations

Le référentiel Serverless Test Samples GitHub contient des exemples concrets de tests qui suivent les modèles et les meilleures pratiques décrits dans ce guide. Le référentiel contient des exemples de code et des descriptions guidées des processus de simulation, d'émulation et de test dans le cloud décrits dans les sections précédentes. Utilisez ce référentiel pour vous familiariser avec les derniers conseils de test sans serveur publiés par AWS.

Suggestions de lecture

Visitez Serverless Land pour accéder aux derniers blogs, vidéos et formations sur les technologies AWS sans serveur.

Il est également recommandé de lire les articles de AWS blog suivants :

- Accélérer le développement sans serveur avec AWS SAM Accelerate (article de AWS blog)
- Accroître la vitesse de développement avec CDK Watch (article de AWS blog)
- Simulation d'intégrations de services avec AWS Step Functions Local (AWS article de blog)
- Commencer à tester des applications sans serveur (article de AWS blog)

Outils

- AWS SAM — Tester et déboguer des applications sans serveur
- AWS SAM — Intégration aux tests automatisés
- Lambda : test des fonctions Lambda dans la console Lambda

# Invoquer Lambda avec des événements provenant d'autres services AWS

Certains AWS services peuvent appeler directement des fonctions Lambda à l'aide de déclencheurs. Ces services transmettent les événements à Lambda, et la fonction est invoquée immédiatement lorsque l'événement spécifié se produit. Les déclencheurs sont adaptés aux événements discrets et au traitement en temps réel. Lorsque vous créez un déclencheur à l'aide de la console Lambda, celle-ci interagit avec le AWS service correspondant pour configurer la notification d'événement sur ce service. Le déclencheur est en fait stocké et géré par le service qui génère les événements, et non par Lambda.

Les événements sont des données structurées dans JSON un format. La JSON structure varie en fonction du service qui la génère et du type d'événement, mais elles contiennent toutes les données dont la fonction a besoin pour traiter l'événement.

Une fonction peut avoir plusieurs déclencheurs. Chaque déclencheur agit comme un client qui invoque votre fonction indépendamment, et chaque événement que Lambda transmet à votre fonction contient des données provenant d'un seul déclencheur. Lambda convertit le document d'événement en objet et le transmet à votre gestionnaire de fonctions.

Selon le service, l'appel piloté par un événement peut être synchrone ou asynchrone.

- Pour une invocation synchrone, l'autre service qui génère l'événement attend la réponse de votre fonction. Ce service définit les données que la fonction doit renvoyer dans la réponse. Le service contrôle la stratégie d'erreur, par exemple s'il faut réessayer les erreurs.
- Pour l'invocation asynchrone, Lambda place l'événement dans une file d'attente avant de le transmettre à votre fonction. Lorsque Lambda met l'événement en file d'attent, il envoie immédiatement une réponse de réussite au service qui a généré l'événement. Après que la fonction ait traité l'événement, Lambda ne renvoie pas de réponse au service qui a généré l'événement.

## Création d'un déclencheur

Le moyen le plus simple de créer un déclencheur est d'utiliser la console Lambda. Lorsque vous créez un déclencheur à l'aide de la console, Lambda ajoute automatiquement les autorisations requises à la politique basée sur les ressources de la fonction.

Pour créer un déclencheur à l'aide de la console Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction pour laquelle vous souhaitez créer un déclencheur.
3. Dans le volet de Présentation de la fonction, choisissez Ajouter un déclencheur.
4. Sélectionnez le AWS service pour lequel vous souhaitez appeler votre fonction.
5. Renseignez les options dans le volet de configuration du déclencheur et choisissez Ajouter. En fonction de la fonction Service AWS que vous choisissez d'invoquer, les options de configuration du déclencheur seront différentes.

## Services pouvant invoquer des fonctions Lambda

Le tableau suivant répertorie les services qui peuvent invoquer des fonctions Lambda.

| Service | Méthode d'invocation |
| --- | --- |
| Streaming géré par Amazon pour Apache Kafka | Cartographie des sources d'événements |
| Self-managed Apache Kafka | Cartographie des sources d'événements |
| APIPasserelle Amazon | Pilotée par les événements ; invocation synchrone |
| AWS CloudFormation | Pilotée par les événements ; invocation asynchrone |
| Amazon CloudWatch Logs | Pilotée par les événements ; invocation asynchrone |
| AWS CodeCommit | Pilotée par les événements ; invocation asynchrone |
| AWS CodePipeline | Pilotée par les événements ; invocation asynchrone |
| Amazon Cognito | Pilotée par les événements ; invocation synchrone |
| AWS Config | Pilotée par les événements ; invocation asynchrone |
| Amazon Connect | Pilotée par les événements ; invocation synchrone |
| Amazon DynamoDB | Cartographie des sources d'événements |

| Service | Méthode d'invocation |
| --- | --- |
| Amazon Elastic File System | Intégration spéciale |
| Elastic Load Balancing (Application Load Balancer) | Pilotée par les événements ; invocation synchrone |
| Amazon EventBridge (CloudWatch Événements) | Axé sur les événements ; appel asynchrone (bus d'événements), appel synchrone ou asynchrone (canaux et plannings) |
| AWS IoT | Pilotée par les événements ; invocation asynchrone |
| Amazon Kinesis | Cartographie des sources d'événements |
| Amazon Data Firehose | Pilotée par les événements ; invocation synchrone |
| Amazon Lex | Pilotée par les événements ; invocation synchrone |
| Amazon MQ | Cartographie des sources d'événements |
| Amazon Simple Email Service | Pilotée par les événements ; invocation asynchrone |
| Amazon Simple Notification Service | Pilotée par les événements ; invocation asynchrone |
| Amazon Simple Queue Service | Cartographie des sources d'événements |
| Amazon Simple Storage Service (Amazon S3) | Pilotée par les événements ; invocation asynchrone |
| Amazon Simple Storage Service Batch | Pilotée par les événements ; invocation synchrone |
| Secrets Manager | Intégration spéciale |
| AWS Step Functions | Invocation basée sur les événements ; synchrone ou asynchrone |
| Amazon VPC Lattice | Pilotée par les événements ; invocation synchrone |
| AWS X-Ray | Intégration spéciale |

# Utilisation de Lambda avec Apache Kafka autogéré

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Lambda prend en charge Apache Kafka en tant que source d'événement. Apache Kafka est une plateforme open source de streaming d'événements qui prend en charge des charges de travail telles que les pipelines de données et l'analytique de streaming.

Vous pouvez utiliser le service Kafka AWS géré Amazon Managed Streaming for Apache Kafka (MSKAmazon) ou un cluster Kafka autogéré. Pour en savoir plus sur l'utilisation de Lambda avec AmazonMSK, consultez. Utilisation de Lambda avec Amazon MSK

Cette rubrique décrit comment utiliser Lambda avec un cluster Kafka autogéré. En AWS termes terminologiques, un cluster autogéré inclut les clusters Kafka non AWS hébergés. Par exemple, vous pouvez héberger votre cluster Kafka avec un fournisseur de cloud tel que Confluent Cloud.

Apache Kafka en tant que source d'événements fonctionne de la même manière qu'Amazon Simple Queue Service (AmazonSQS) ou Amazon Kinesis. Lambda interroge en interne les nouveaux messages de la source d'événement, puis invoque de manière synchrone la fonction Lambda cible. Lambda lit les messages par lot et les fournit à votre fonction en tant que charge utile d'événement. La taille de lot maximale est configurable. (par défaut, 100 messages).

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Pour les sources d'événements basées sur Kafka, Lambda prend en charge les paramètres de contrôle du traitement par lots, tels que les fenêtres de traitement par lots et la taille des lots. Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

Pour un exemple d'utilisation de Kafka autogéré comme source d'événements, consultez la section Utilisation d'Apache Kafka auto-hébergée comme source d'événements AWS Lambda sur le blog Compute. AWS

Rubriques


- Exemple d’évènement
- Configuration de sources d'événements Apache Kafka autogérées pour Lambda
- Traitement des messages Apache Kafka autogérés avec Lambda
- Utilisation du filtrage des événements avec une source d'événements Apache Kafka autogérée
- Capture des lots rejetés pour une source d'événements Apache Kafka autogérée
- Résolution des erreurs de mappage des sources d'événements Apache Kafka autogérées


## Exemple d’évènement

Lambda envoie le lot de messages dans le paramètre d’événement quand il invoque votre fonction Lambda. La charge utile d’un événement contient un tableau de messages. Chaque élément de tableau contient des détails de la rubrique Kafka et l’identifiant de partition Kafka, ainsi qu’un horodatage et un message codé en base 64.


```
{
   "eventSource": "SelfManagedKafka",
   "bootstrapServers":"b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
   "records":{
      "mytopic-0":[
         {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "key":"abcDEFghiJKLmnoPQRstuVWXyz1234==",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[
               {
                  "headerKey":[
                     104,
```

```
                    101,
                    97,
                    100,
                    101,
                    114,
                    86,
                    97,
                    108,
                    117,
                    101
                ]
            }
        ]
    }
  ]
 }
}
```

## Configuration de sources d'événements Apache Kafka autogérées pour Lambda

Avant de créer un mappage de source d'événements pour votre cluster Apache Kafka autogéré, vous devez vous assurer que votre cluster et celui dans lequel VPC il réside sont correctement configurés. Vous devez également vous assurer que le rôle d'exécution de votre fonction Lambda dispose des autorisations nécessairesIAM.

Suivez les instructions des sections suivantes pour configurer votre cluster Apache Kafka autogéré et votre fonction Lambda. Pour savoir comment créer le mappage des sources d'événements, consultezthe section called “Ajout d’un cluster Kafka en tant que source d’événement”.

Rubriques

- Authentification de cluster Kafka
- Configuration réseau
- APIautorisations d'accès et de fonction Lambda

### Authentification de cluster Kafka

Lambda prend en charge plusieurs méthodes d’authentification auprès de votre cluster Apache Kafka autogéré. Veillez à configurer le cluster Kafka pour utiliser l’une des méthodes d’authentification

prises en charge suivantes : Pour de plus amples informations sur la sécurité, veuillez consulter la section Sécurité de la documentation Kafka.

VPCaccès

Si seuls les utilisateurs Kafka de votre entourage ont VPC accès à vos courtiers Kafka, vous devez configurer la source d'événements Kafka pour l'accès à Amazon Virtual Private Cloud (AmazonVPC).

SASL/SCRAMauthentification

Lambda prend en charge l'authentification simple et l'authentification par couche de sécurité/mécanisme d'authentification Salted Challenge Response (SASL/SCRAM) avec le chiffrement Transport Layer Security (TLS) (). `SASL_SSL` Lambda envoie les informations d'identification chiffrées pour s'authentifier auprès du cluster. Lambda ne supporte pasSASL/SCRAMavec plaintext (). `SASL_PLAINTEXT` Pour plus d'informations sur SCRAM l'authentificationSASL/, consultez RFC5802.

Lambda prend également en charge l'authentification SASL /. PLAIN Comme ce mécanisme utilise des informations d'identification en texte clair, la connexion au serveur doit être TLS cryptée pour garantir la protection des informations d'identification.

Pour SASL l'authentification, vous stockez les informations de connexion sous forme de code secret dans AWS Secrets Manager. Pour plus d'informations sur l'utilisation de Secrets Manager, consultez Didacticiel : création et récupération d'un secret dans le Guide de l'utilisateur AWS Secrets Manager .

> **Important**
>
> Pour utiliser Secrets Manager pour l'authentification, les secrets doivent être stockés dans la même AWS région que votre fonction Lambda.

TLSAuthentification mutuelle

Mutual TLS (mTLS) fournit une authentification bidirectionnelle entre le client et le serveur. Le client envoie un certificat au serveur pour que le serveur vérifie le client, et le serveur envoie un certificat au client pour que le client vérifie le serveur.

Dans Apache Kafka autogéré, Lambda agit en tant que client. Vous configurez un certificat client (en tant que secret dans Secrets Manager) pour authentifier Lambda auprès des courtiers Kafka. Le certificat client doit être signé par une autorité de certification dans le magasin d'approbations du serveur.

Le cluster Kafka envoie un certificat de serveur à Lambda pour authentifier les courtiers auprès de Lambda. Le certificat de serveur peut être un certificat d'autorité de certification public ou un certificat CA/auto-signé privé. Le certificat d'autorité de certification publique doit être signé par une autorité de certification située du magasin d'approbations Lambda. Pour un certificat CA/auto-signé privé, vous configurez le certificat d'autorité de certification racine du serveur (en tant que secret dans Secrets Manager). Lambda utilise le certificat racine pour vérifier les courtiers Kafka.

Pour plus d'informations sur mTLS, consultez Présentation de TLS l'authentification mutuelle pour Amazon MSK en tant que source d'événements.

Configuration du secret du certificat client

Le AUTH secret CLIENT CERTIFICATE _ TLS _ _ nécessite un champ de certificat et un champ de clé privée. Pour une clé privée chiffrée, le secret nécessite un mot de passe de clé privée. Le certificat et la clé privée doivent tous deux être au PEM format.

> **Note**
>
> Lambda prend en charge les algorithmes de chiffrement par clé privée PBES1(mais pasPBES2).

Le champ de certificat doit contenir une liste de certificats, commençant par le certificat client, suivi de tous les certificats intermédiaires et se terminant par le certificat racine. Chaque certificat doit commencer sur une nouvelle ligne avec la structure suivante :

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager prend en charge les secrets jusqu'à 65 536 octets, ce qui offre suffisamment d'espace pour de longues chaînes de certificats.

La clé privée doit être au format PKCS#8, avec la structure suivante :

```
-----BEGIN PRIVATE KEY-----
         <private key contents>
-----END PRIVATE KEY-----
```

Pour une clé privée chiffrée, utilisez la structure suivante :

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
          <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

L'exemple suivant montre le contenu d'un secret pour mon TLS authentification à l'aide d'une clé privée cryptée. Pour une clé privée chiffrée, incluez le mot de passe de clé privée dans le secret.

```
{"privateKeyPassword":"testpassword",
"certificate":"-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
"privateKey":"-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

Configuration du secret du certificat d’autorité de certification racine du serveur

Vous créez ce secret si vos courtiers Kafka utilisent le TLS chiffrement avec des certificats signés par une autorité de certification privée. Vous pouvez utiliser TLS le chiffrement pour l'TLSauthentification VPC SCRAMPLAIN, SASLSASL/,/ou m.

Le secret du certificat de l'autorité de certification racine du serveur nécessite un champ contenant le PEM format du certificat de l'autorité de certification racine du courtier Kafka. La structure du secret est présentée dans l’exemple suivant.

```
{"certificate":"-----BEGIN CERTIFICATE-----
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMCVVMx
EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4xOzA5BgNVBAMTMlN0YXJmaWVs
```

```
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXRlIEF1dG...
-----END CERTIFICATE-----"
}
```

## Configuration réseau

Pour que Lambda puisse utiliser votre cluster Kafka comme source d'événements, il doit avoir accès à l'Amazon dans lequel réside VPC votre cluster. Nous vous recommandons de déployer des AWS PrivateLink VPCpoints de terminaison pour que Lambda puisse accéder à votre. VPC Déployez des points de terminaison pour Lambda AWS Security Token Service et AWS STS(). Si le broker utilise l'authentification, déployez également un VPC point de terminaison pour Secrets Manager. Si vous avez configuré une destination en cas de défaillance, déployez également un VPC point de terminaison pour le service de destination.

Vous pouvez également vous assurer que le cluster VPC associé à votre cluster Kafka inclut une NAT passerelle par sous-réseau public. Pour de plus amples informations, veuillez consulter the section called “Accès à Internet pour les VPC fonctions”.

Si vous utilisez des VPC points de terminaison, vous devez également les configurer pour activer les DNS noms privés.

Lorsque vous créez un mappage de source d'événements pour un cluster Apache Kafka autogéré, Lambda vérifie si des interfaces réseau élastiques (ENIs) sont déjà présentes pour les sous-réseaux et les groupes de sécurité de votre cluster. VPC Si Lambda trouve des objets existantsENIs, il essaie de les réutiliser. Sinon, Lambda en crée un nouveau ENIs pour se connecter à la source de l'événement et appeler votre fonction.

> **Note**
>
> Les fonctions Lambda s'exécutent toujours au sein VPCs du service Lambda. Ils VPCs sont gérés automatiquement par le service et ne sont pas visibles pour les clients. Vous pouvez également connecter votre fonction à un AmazonVPC. Dans les deux cas, la VPC configuration de votre fonction n'affecte pas le mappage des sources d'événements. Seule la configuration des sources d'événements VPC détermine la manière dont Lambda se connecte à votre source d'événements.

Pour plus d'informations sur la configuration du réseau, voir Configuration AWS Lambda avec un cluster Apache Kafka au sein d'un blog VPC sur le AWS Compute.

VPCrègles du groupe de sécurité

Configurez les groupes de sécurité pour l'Amazon VPC contenant votre cluster avec les règles suivantes (au minimum) :

- Règles entrantes – Autorisent tout le trafic sur le port de l’agent Kafka pour le groupe de sécurité spécifié en tant que source d’évènement. Kafka utilise le port 9092 par défaut.
- Règles sortantes : autorisent tout le trafic sur le port 443 pour toutes les destinations. Autorisent tout le trafic sur le port de l’agent Kafka pour le groupe de sécurité spécifié en tant que source d’évènement. Kafka utilise le port 9092 par défaut.
- Si vous utilisez des VPC points de terminaison au lieu d'une NAT passerelle, les groupes de sécurité associés aux VPC points de terminaison doivent autoriser tout le trafic entrant sur le port 443 en provenance des groupes de sécurité de la source de l'événement.

Utilisation de points de terminaison VPC

Lorsque vous utilisez des VPC points de terminaison, API les appels pour appeler votre fonction sont acheminés via ces points de terminaison à l'aide du. ENIs Le principal du service Lambda doit appeler `sts:AssumeRole` et appeler tous `lambda:InvokeFunction` les rôles et fonctions qui les utilisent. ENIs

Par défaut, les VPC points de terminaison ont IAM des politiques ouvertes. La meilleure pratique consiste à restreindre ces politiques afin de n'autoriser que des principaux spécifiques à effectuer les actions nécessaires à l'aide de ce point de terminaison. Pour garantir que le mappage de votre source d'événements est en mesure d'appeler votre fonction Lambda, la politique du VPC point de terminaison doit autoriser le principe du service Lambda à appeler et. `sts:AssumeRole` `lambda:InvokeFunction` Le fait de restreindre les politiques de vos VPC terminaux pour autoriser uniquement les API appels provenant de votre organisation empêche le mappage des sources d'événements de fonctionner correctement.

Les exemples de politiques de point de VPC terminaison suivants montrent comment accorder l'accès requis au principal de service Lambda pour les points de terminaison Lambda et AWS STS Lambda.

Example VPCpolitique relative aux terminaux - AWS STS points de terminaison

```
{
    "Statement": [
```

```
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example VPCpolitique de point de terminaison - point de terminaison Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Si votre courtier Kafka utilise l'authentification, vous pouvez également restreindre la politique de point de VPC terminaison pour le point de terminaison Secrets Manager. Pour appeler le Secrets ManagerAPI, Lambda utilise votre rôle fonctionnel, et non le principal du service Lambda. L'exemple suivant montre une politique de point de terminaison de Secrets Manager.

Example VPCpolitique relative aux points de terminaison - Point de terminaison Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Si vous avez configuré une destination en cas de défaillance, Lambda utilise également le rôle de votre fonction pour appeler `s3:PutObject` soit`sns:Publish`, `sqs:sendMessage` soit en utilisant le Lambda Géré. ENIs

## APIautorisations d'accès et de fonction Lambda

En plus d'accéder à votre cluster Kafka autogéré, votre fonction Lambda a besoin d'autorisations pour effectuer diverses actions. API Ajoutez ces autorisations au rôle d'exécution de votre fonction. Si vos utilisateurs ont besoin d'accéder à des API actions, ajoutez les autorisations requises à la politique d'identité de l'utilisateur ou du rôle AWS Identity and Access Management (IAM).

### Autorisations de fonction Lambda requises

Pour créer et stocker des journaux dans un groupe de CloudWatch journaux dans Amazon Logs, votre fonction Lambda doit disposer des autorisations suivantes dans son rôle d'exécution :

- journaux : CreateLogGroup
- journaux : CreateLogStream
- journaux : PutLogEvents

### Autorisations de fonction Lambda facultatives

Votre fonction Lambda peut également nécessiter ces autorisations pour :

- Décrivez votre secret Secrets Manager.
- Accédez à votre AWS Key Management Service (AWS KMS) clé gérée par le client.
- Accédez à votre AmazonVPC.
- Envoyez les enregistrements des invocations ayant échoué vers une destination.

### Secrets Manager et AWS KMS autorisations

Selon le type de contrôle d'accès que vous configurez pour vos courtiers Kafka, votre fonction Lambda peut avoir besoin d'une autorisation pour accéder à votre secret Secrets Manager ou pour déchiffrer AWS KMS votre clé gérée par le client. Afin d'accéder à ces ressources, le rôle d'exécution de votre fonction doit disposer des autorisations suivantes :

- responsable des secrets : GetSecretValue
- kms:Decrypt

### VPCautorisations

Si seuls les utilisateurs d'un VPC peuvent accéder à votre cluster Apache Kafka autogéré, votre fonction Lambda doit être autorisée à accéder à vos ressources Amazon. VPC Ces ressources incluent vos sous-réseauxVPC, vos groupes de sécurité et vos interfaces réseau. Afin d'accéder à ces ressources, le rôle d'exécution de votre fonction doit disposer des autorisations suivantes :

- EC2 : CreateNetworkInterface
- EC2 : DescribeNetworkInterfaces
- EC2 : DescribeVpcs
- EC2 : DeleteNetworkInterface
- EC2 : DescribeSubnets
- EC2 : DescribeSecurityGroups

### Ajout d'autorisations à votre rôle d'exécution

Pour accéder aux autres AWS services utilisés par votre cluster Apache Kafka autogéré, Lambda utilise les politiques d'autorisation que vous définissez dans le rôle d'exécution de votre fonction Lambda.

Par défaut, Lambda n'est pas autorisé à exécuter les actions obligatoires ou facultatives pour un cluster Apache Kafka autogéré. Vous devez créer et définir ces actions dans une politique de IAM confiance pour votre rôle d'exécution. Cet exemple montre comment créer une politique permettant à Lambda d'accéder à vos ressources AmazonVPC.

```
{
        "Version":"2012-10-17",
```

```
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "ec2:CreateNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
                 "ec2:DescribeVpcs",
                 "ec2:DeleteNetworkInterface",
                 "ec2:DescribeSubnets",
                 "ec2:DescribeSecurityGroups"
              ],
              "Resource":"*"
           }
        ]
     }
```

Accorder l'accès aux utilisateurs avec une IAM politique

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à effectuer des APIopérations sur les sources d'événements. Pour accorder l'accès aux utilisateurs de votre organisation ou de votre compte, vous pouvez ajouter ou mettre à jour une stratégie basée sur l'identité. Pour plus d'informations, consultez la section Contrôle de l'accès aux AWS ressources à l'aide de politiques dans le Guide de IAM l'utilisateur.

## Traitement des messages Apache Kafka autogérés avec Lambda

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Rubriques

- Ajout d'un cluster Kafka en tant que source d'événement
- Paramètres de configuration Apache Kafka autogérés
- Utilisation d'un cluster Kafka en tant que source d'événement
- Positions de départ des interrogations et des flux
- Mise à l'échelle automatique de la source d'événement Kafka
- CloudWatch Métriques Amazon

## Ajout d’un cluster Kafka en tant que source d’événement

Pour créer un mappage de source d'événements, ajoutez votre cluster Kafka en tant que déclencheur de fonction Lambda à l'aide de la console Lambda, AWS SDKd'un ou du ().AWS Command Line InterfaceAWS CLI

Cette section explique comment créer un mappage de source d'événement à l'aide de la console Lambda et de l' AWS CLI.

Prérequis

- Cluster Apache Kafka autogéré. Lambda prend en charge Apache Kafka versions 0.10.1.0 et ultérieures.
- Rôle d'exécution autorisé à accéder aux AWS ressources utilisées par votre cluster Kafka autogéré.

Identifiant de groupe de consommateurs personnalisable

Lorsque vous configurez Kafka comme source d’événements, vous pouvez spécifier un identifiant de groupe de consommateurs. Cet identifiant de groupe de consommateurs est un identifiant existant pour le groupe de clients Kafka auquel vous souhaitez rattacher votre fonction Lambda. Vous pouvez utiliser cette fonction pour migrer facilement toutes les configurations de traitement d’enregistrements Kafka en cours depuis d’autres clients vers Lambda.

Si vous spécifiez un identifiant de groupe de consommateurs et qu’il existe d’autres sondeurs actifs au sein de ce groupe de consommateurs, Kafka distribue des messages à tous les consommateurs. En d’autres termes, Lambda ne reçoit pas l’intégralité du message relatif au sujet Kafka. Si vous souhaitez que Lambda gère tous les messages de la rubrique, désactivez tous les autres sondeurs de ce groupe de consommateurs.

De plus, si vous spécifiez un identifiant de groupe de consommateurs et que Kafka trouve un groupe de consommateurs existant valide avec le même identifiant, Lambda ignore le paramètre `StartingPosition` pour le mappage des sources d’événements. Lambda commence plutôt à traiter les enregistrements en fonction de la compensation engagée par le groupe de consommateurs. Si vous spécifiez un identifiant de groupe de consommateurs et que Kafka ne trouve aucun groupe de consommateurs existant, Lambda configure votre source d’événement avec le `StartingPosition` spécifié.

L’identifiant du groupe de consommateurs que vous spécifiez doit être unique parmi toutes vos sources d’événements Kafka. Après avoir créé un mappage de sources d'événements Kafka avec l'identifiant de groupe de consommateurs spécifié, vous ne pouvez plus mettre à jour cette valeur.

Ajout d’un cluster Kafka autogéré (console)

Pour ajouter votre cluster Apache Kafka autogéré et une rubrique Kafka en tant que déclencheur pour votre fonction Lambda, procédez comme suit.

Pour ajouter un déclencheur Apache Kafka à votre fonction Lambda (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de votre fonction Lambda.
3. Sous Function overview (Vue d’ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Sous Trigger configuration (Configuration du déclencheur), procédez comme suit :
    a. Choisissez le type de déclencheur Apache Kafka.
    b. Pour Bootstrap servers (Serveurs d’amorçage), entrez l’adresse de paire hôte et port d’un agent Kafka dans votre cluster, puis choisissez Add (Ajouter). Répétez l’opération pour chaque agent Kafka dans le cluster.
    c. Pour Topic name (Nom de rubrique), entrez le nom de la rubrique Kafka utilisée pour stocker les registres dans le cluster.
    d. (Facultatif) Pour Batch size (Taille de lot), entrez le nombre maximal de registres à recevoir dans un même lot.
    e. Pour Batch window, veuillez saisir l’intervalle de temps maximal en secondes nécessaire à Lambda pour collecter des enregistrements avant d’invoquer la fonction.
    f. (Facultatif) Pour l’identifiant de groupe de consommateurs, entrez l’identifiant d’un groupe de consommateurs Kafka à rejoindre.
    g. (Facultatif) Pour Position de départ, choisissez Dernier pour commencer à lire le flux à partir du dernier enregistrement, Supprimer l’horizon pour commencer au premier enregistrement disponible ou À l’horodatage pour spécifier un horodatage à partir duquel commencer la lecture.
    h. (Facultatif) Pour VPC, choisissez l'Amazon VPC pour votre cluster Kafka. Choisissez ensuite les VPCsous-réseaux et les groupes VPC de sécurité.

Ce paramètre est obligatoire si seuls les utilisateurs de votre entreprise VPC accèdent à vos courtiers.

i. (Facultatif) Pour Authentication (Authentification), choisissez Add (Ajouter), puis procédez comme suit :

   i. Choisissez le protocole d'accès ou d'authentification des courtiers Kafka dans votre cluster.

      - Si votre courtier Kafka utilise PLAIN l'authentificationSASL/, choisissez BASIC_ AUTH.
      - Si votre courtier utilise SCRAM l'authentificationSASL/, choisissez l'un des SCRAM protocoles SASL_.
      - Si vous configurez TLS l'authentification m, choisissez le AUTH protocole CLIENTCERTIFICATE_ TLS _ _.

   ii. Pour l'TLSauthentificationSASL/SCRAMou m, choisissez la clé secrète du Gestionnaire de Secrets Manager qui contient les informations d'identification de votre cluster Kafka.

j. (Facultatif) Pour le chiffrement, choisissez le secret du Gestionnaire de secrets contenant le certificat de CA racine que vos courtiers Kafka utilisent pour le TLS chiffrement, si vos courtiers Kafka utilisent des certificats signés par une autorité de certification privée.

   Ce paramètre s'applique au TLS chiffrement pourSASL/SCRAMouSASL/PLAIN, ainsi qu'à TLS l'authentification m.

k. Pour créer le déclencheur dans un état désactivé pour le test (recommandé), désactivez Enable trigger (Activer le déclencheur). Ou, pour activer le déclencheur immédiatement, sélectionnezActiver un déclencheur.

5. Pour créer le déclencheur, choisissez Add (Ajouter).

Ajout d'un cluster Kafka autogéré (AWS CLI)

Utilisez les exemples de AWS CLI commandes suivants pour créer et afficher un déclencheur Apache Kafka autogéré pour votre fonction Lambda.

Utilisation deSASL/SCRAM

Si les utilisateurs de Kafka accèdent à vos courtiers Kafka via Internet, spécifiez le secret du Gestionnaire de Secrets que vous avez créé pour l'authentificationSASL//SCRAM. L'exemple suivant

utilise la create-event-source-mapping AWS CLI commande pour mapper une fonction Lambda nommée `my-kafka-function` à une rubrique Kafka nommée. `AWSKafkaTopic`

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-
east-1:111122223333:secret:MyBrokerSecretName \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

À l'aide d'un VPC

Si seuls les utilisateurs Kafka de votre entourage ont VPC accès à vos courtiers Kafka, vous devez spécifier vos VPC sous-réseaux et VPC votre groupe de sécurité. L'exemple suivant utilise la create-event-source-mapping AWS CLI commande pour mapper une fonction Lambda nommée `my-kafka-function` à une rubrique Kafka nommée. `AWSKafkaTopic`

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration '[{"Type": "VPC_SUBNET", "URI":
 "subnet:subnet-0011001100"}, {"Type": "VPC_SUBNET", "URI":
 "subnet:subnet-0022002200"}, {"Type": "VPC_SECURITY_GROUP", "URI":
 "security_group:sg-0123456789"}]' \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

Affichage de l'état à l'aide du AWS CLI

L'exemple suivant utilise la get-event-source-mapping AWS CLI commande pour décrire l'état du mappage des sources d'événements que vous avez créé.

```
aws lambda get-event-source-mapping
          --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

## Paramètres de configuration Apache Kafka autogérés

Tous les types de sources d'événements Lambda partagent les mêmes opérations CreateEventSourceMapping. UpdateEventSourceMappingAPI Cependant, seuls certains paramètres s'appliquent à Apache Kafka.

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| BatchSize | N | 100 | Maximum : 10 000. |
| Activé | N | Activées | none |
| FunctionName | Y | N/A | none |
| FilterCriteria | N | N/A | Contrôlez les événements que Lambda envoie à votre fonction |
| MaximumBatchingWindowInSeconds | N | 500 ms | Comportement de traitement par lots |
| SelfManagedEventSource | Y | N/A | Liste des agents Kafka. Peut définir uniquement sur Create (Créer) |
| SelfManagedKafkaEventSourceConfig | N | Contient le ConsumerGroupId champ qui prend par défaut une valeur unique. | Peut définir uniquement sur Create (Créer) |
| SourceAccessConfigurations | N | Pas d'informations d'identification | VPCinformations ou informations d'identification pour le cluster<br><br>Pour SASL _PLAIN, définissez sur BASIC _ AUTH |
| StartingPosition | Y | N/A | AT_TIMESTAMP, TRIM _HORIZON, ou LATEST |

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| | | | Peut définir uniquement sur Create (Créer) |
| StartingPositionTimestamp | N | N/A | Obligatoire s' StartingPosition il est défini sur AT_TIMESTAMP |
| Rubriques | Y | N/A | Nom de la rubrique<br><br>Peut définir uniquement sur Create (Créer) |

## Utilisation d’un cluster Kafka en tant que source d’événement

Lorsque vous ajoutez votre MSK cluster Apache Kafka ou Amazon comme déclencheur pour votre fonction Lambda, le cluster est utilisé comme source d'événements.

Lambda lit les données d'événements des sujets Kafka que vous spécifiez `Topics` dans une CreateEventSourceMappingdemande, en fonction de `StartingPosition` ce que vous spécifiez. Lorsque le traitement a réussi, votre rubrique Kafka est validée dans votre cluster Kafka.

Si vous spécifiez `StartingPosition` comme `LATEST`, Lambda commence à lire à partir du dernier message dans chaque partition de la rubrique. Un certain temps pouvant s’écouler après la configuration du déclencheur avant que Lambda commence à lire les messages, Lambda ne lit aucun message produit pendant cette fenêtre de temps.

Lambda traite les enregistrements d'une ou de plusieurs partitions de rubrique Kafka que vous spécifiez et envoie une JSON charge utile à votre fonction. Lorsque d'autres enregistrements sont disponibles, Lambda continue de traiter les enregistrements par lots, en fonction de la `BatchSize` valeur que vous spécifiez dans une CreateEventSourceMappingdemande, jusqu'à ce que votre fonction aborde le sujet.

Si votre fonction renvoie une erreur pour l’un des messages d’un lot, Lambda réessaie le lot de messages complet jusqu’à ce que le traitement réussisse ou que les messages expirent. Vous

pouvez envoyer les enregistrements qui échouent à toutes les tentatives vers une destination en cas d'échec pour un traitement ultérieur.

> **Note**
>
> Alors que les fonctions Lambda ont généralement un délai d'expiration maximal de 15 minutes, les mappages de sources d'événements pour Amazon, Apache Kafka autogéré, MSK Amazon DocumentDB et Amazon MQ pour ActiveMQ et RabbitMQ ne prennent en charge que les fonctions dont le délai d'expiration maximal est de 14 minutes. Cette contrainte garantit que le mappage des sources d'événements peut gérer correctement les erreurs de fonction et effectuer de nouvelles tentatives.

## Positions de départ des interrogations et des flux

Sachez que l'interrogation des flux lors des mises à jour et de la création du mappage des sources d'événements est finalement cohérente.

- Lors de la création du mappage des sources d'événements, le démarrage de l'interrogation des événements depuis le flux peut prendre plusieurs minutes.
- Lors des mises à jour du mappage des sources d'événements, l'arrêt et le redémarrage de l'interrogation des événements depuis le flux peuvent prendre plusieurs minutes.

Ce comportement signifie que si vous spécifiez `LATEST` comme position de départ du flux, le mappage des sources d'événements peut manquer des événements lors de la création ou des mises à jour. Pour vous assurer de ne manquer aucun événement, spécifiez la position de départ du flux comme `TRIM_HORIZON` ou `AT_TIMESTAMP`.

## Mise à l'échelle automatique de la source d'événement Kafka

Lorsque vous créez initialement une source d'événement Apache Kafka, Lambda alloue un consommateur pour traiter toutes les partitions de la rubrique Kafka. Chaque consommateur dispose de plusieurs processeurs exécutés en parallèle pour gérer des charges de travail accrues. Lambda augmente ou diminue automatiquement le nombre de consommateurs, en fonction de la charge de travail. Pour préserver l'ordre des messages dans chaque partition, le nombre maximum de consommateurs est de un par partition dans la rubrique.

Toutes les minutes, Lambda évalue le décalage de consommateurs de toutes les partitions dans la rubrique. Si le décalage est trop élevé, la partition reçoit des messages plus rapidement que Lambda ne peut les traiter. Si nécessaire, Lambda ajoute ou supprime des consommateurs dans la rubrique. Le processus de mise à l'échelle consistant à ajouter ou à supprimer des consommateurs a lieu dans les trois minutes suivant l'évaluation.

Si votre fonction Lambda cible est surchargée, Lambda réduit le nombre de consommateurs. Cette action réduit la charge de travail de la fonction en diminuant le nombre de messages que les consommateurs peuvent échanger avec la fonction.

Pour surveiller le débit de votre rubrique Kafka, vous pouvez afficher les métriques de consommateurs Apache Kafka, telles que `consumer_lag` et `consumer_offset`. Pour vérifier le nombre d'invocations de fonctions qui se produisent en parallèle, vous pouvez également surveiller les métriques de simultanéité pour votre fonction.

### CloudWatch Métriques Amazon

Lambda émet la métrique `OffsetLag` pendant que votre fonction traite les registres. La valeur de cette métrique est la différence de décalage entre le dernier enregistrement inscrit dans la rubrique source de l'événement Kafka et le dernier enregistrement traité par le groupe de consommateurs de votre fonction. Vous pouvez utiliser `OffsetLag` pour estimer la latence entre le moment où un enregistrement est ajouté et celui où votre groupe de consommateurs le traite.

Une tendance à la hausse de `OffsetLag` peut indiquer des problèmes liés aux sondeurs dans le groupe de consommateurs de votre fonction. Pour de plus amples informations, veuillez consulter Afficher les métriques des fonctions Lambda.

## Utilisation du filtrage des événements avec une source d'événements Apache Kafka autogérée

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources d'événements Apache Kafka autogérées.

Rubriques

- Notions de base du filtrage d'événements autogéré par Apache Kafka

## Notions de base du filtrage d'événements autogéré par Apache Kafka

Supposons qu'un producteur rédige des messages dans un sujet de votre cluster Apache Kafka autogéré, soit dans un JSON format valide, soit sous forme de chaînes simples. Un exemple d'enregistrement ressemblerait à ce qui suit, avec le message converti en chaîne encodée en Base64 dans le champ `value`.

```
{
    "mytopic-0":[
        {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[]
        }
    ]
}
```

Supposons que votre producteur Apache Kafka rédige des messages pour votre sujet dans le JSON format suivant.

```
{
    "device_ID": "AB1234",
    "session":{
        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}
```

Vous pouvez utiliser la clé `value` pour filtrer les enregistrements. Supposons que vous vouliez filtrer uniquement les enregistrements où `device_ID` commence par les lettres AB. L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
        }
```

```
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "value": {
        "device_ID": [ { "prefix": "AB" } ]
      }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

AWS SAM

Pour ajouter ce filtre à l'aide de AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Avec Apache Kafka autogéré, vous pouvez également filtrer les enregistrements dont le message est une chaîne simple. Supposons que vous vouliez ignorer les messages dont la chaîne est « error ». L'objet `FilterCriteria` se présente comme suit.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "value": [
        {
        "anything-but": [ "error" ]
        }
    ]
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d’événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Pour ajouter ce filtre à l'aide de AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

Les messages Apache Kafka autogérés doivent être des chaînes codées en UTF -8, soit des chaînes simples, soit au format. JSON C'est parce que Lambda décode les tableaux d'octets de Kafka en UTF -8 avant d'appliquer des critères de filtre. Si vos messages utilisent un autre encodage, tel que UTF -16 ouASCII, ou si le format du message ne correspond pas au `FilterCriteria` format, Lambda traite uniquement les filtres de métadonnées. Le tableau suivant résume le comportement spécifique :

| Format du du message entrant | Modèle de filtre de format pour les propriétés des messages | Action obtenue. |
|---|---|---|
| Chaîne de texte brut | Chaîne de texte brut | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne de texte brut | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Chaîne de texte brut | Valide JSON | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Chaîne de texte brut | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne non codée en UTF -8 | JSON, chaîne simple ou sans motif | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |

# Capture des lots rejetés pour une source d'événements Apache Kafka autogérée

Pour retenir les enregistrements des invocations de mappage de sources d'événements qui ont échoué, ajoutez une destination au mappage des sources d'événements de votre fonction. Chaque enregistrement envoyé à la destination est un JSON document contenant des métadonnées relatives à l'échec de l'invocation. Vous pouvez configurer n'importe quel SNS sujet Amazon, SQS file d'attente Amazon ou compartiment S3 comme destination. Votre rôle d'exécution doit disposer d'autorisations pour la destination :

- Pour les SQS destinations : sqs : SendMessage
- Pour les SNS destinations : SNS:Publish
- Pour les destinations du compartiment S3 : s3 : PutObject et s3 : ListBuckets

Vous devez déployer un VPC point de terminaison pour votre service de destination en cas de défaillance dans votre cluster Apache Kafka. VPC

En outre, si vous avez configuré une KMS clé sur votre destination, Lambda a besoin des autorisations suivantes en fonction du type de destination :

- Si vous avez activé le chiffrement avec votre propre KMS clé pour une destination S3, kms : GenerateDataKey est obligatoire. Si la KMS clé et la destination du compartiment S3 se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser kms :. GenerateDataKey
- Si vous avez activé le chiffrement avec votre propre KMS clé de SQS destination, KMS:Decrypt et kms : sont obligatoires. GenerateDataKey Si la KMS clé et la destination de la SQS file d'attente se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser KMS:Decrypt, kms :GenerateDataKey, kms : et kms :. DescribeKey ReEncrypt
- Si vous avez activé le chiffrement avec votre propre KMS clé de SNS destination, KMS:Decrypt et kms : sont obligatoires. GenerateDataKey Si la KMS clé et la destination du SNS sujet se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser KMS:Decrypt, kms :GenerateDataKey, kms : et kms :. DescribeKey ReEncrypt

## Configuration de destinations en cas de panne pour un mappage de sources d'événements Apache Kafka autogéré

Pour configurer une destination en cas de panne à l'aide de la console, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Invocation du mappage des sources d'événements.
5. Pour le mappage des sources d'événements, choisissez une source d'événements configurée pour cette fonction.
6. Pour Condition, sélectionnez En cas d'échec. Pour les invocations de mappage des sources d'événements, il s'agit de la seule condition acceptée.
7. Pour Type de destination, choisissez le type de destination auquel Lambda envoie les enregistrements d'invocation.
8. Pour Destination, choisissez une ressource.
9. Choisissez Save (Enregistrer).

Vous pouvez également configurer une destination en cas de panne à l'aide du AWS CLI. Par exemple, la create-event-source-mappingcommande suivante ajoute un mappage de source d'événement avec une destination SQS en cas de défaillance à `MyFunction` :

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

La update-event-source-mappingcommande suivante ajoute une destination S3 en cas de défaillance à la source d'événements associée à l'entrée `uuid` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
```

```
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Pour supprimer une destination, entrez une chaîne vide comme argument du paramètre `destination-config` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

SNSet SQS exemple d'enregistrement d'invocation

L'exemple suivant montre ce que Lambda envoie à un SNS sujet ou à une destination de SQS file d'attente en cas d'échec d'un appel de source d'événement Kafka. Chacune des clés sous `recordsInfo` contient à la fois le sujet et la partition Kafka, séparés par un trait d'union. Par exemple, pour la clé`"Topic-0"`, `Topic` est la rubrique Kafka, et `0` est la partition. Pour chaque sujet et chaque partition, vous pouvez utiliser les décalages et les données d'horodatage pour trouver les enregistrements d'invocation d'origine.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
```

```
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    }
}
```

Exemple d'enregistrement d'appel de destination S3

Pour les destinations S3, Lambda envoie l'intégralité de l'enregistrement d'invocation ainsi que les métadonnées à la destination. L'exemple suivant montre que Lambda envoie vers une destination de compartiment S3 en cas d'échec d'une invocation de source d'événement Kafka. Outre tous les champs de l'exemple précédent pour SQS et pour les SNS destinations, le `payload` champ contient l'enregistrement d'appel d'origine sous forme de JSON chaîne échappée.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
```

```
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    },
    "payload": "<Whole Event>" // Only available in S3
}
```

> **Tip**
>
> Nous vous recommandons d’activer la gestion des versions S3 sur votre compartiment de destination.

## Résolution des erreurs de mappage des sources d'événements Apache Kafka autogérées

Les rubriques suivantes fournissent des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation d'Apache Kafka autogéré avec Lambda. Si vous rencontrez un problème qui n’est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Pour obtenir de l'aide supplémentaire en matière de résolution des problèmes, consultez le centre de AWS connaissances.

## Authentification et erreurs d’autorisation

Si l'une des autorisations requises pour consommer les données du cluster Kafka est manquante, Lambda affiche l'un des messages d'erreur suivants dans le mappage des sources d'événements ci-dessous. LastProcessingResult

Messages d’erreur

- Le cluster n’a pas pu autoriser Lambda
- Échec de l’authentification SASL
- Le serveur n’a pas réussi à authentifier Lambda
- Lambda n’a pas réussi à authentifier le serveur
- Le certificat ou la clé privée fournis n’est pas valide

Le cluster n’a pas pu autoriser Lambda

Pour SASL/SCRAM ou mTLS, cette erreur indique que l’utilisateur fourni ne dispose pas de toutes les autorisations de liste de contrôle d’accès (ACL) Kafka requises suivantes :

- DescribeConfigs Cluster
- Décrire un groupe
- Lire le groupe
- Décrire la rubrique
- Lire la rubrique

Lorsque vous créez des listes ACL Kafka avec les autorisations `kafka-cluster` requises, vous devez spécifier la rubrique et le groupe en tant que ressources. Le nom de la rubrique doit correspondre à la rubrique dans le mappage des sources d’événements. Le nom du groupe doit correspondre à l’UUID du mappage des sources d’événements.

Une fois que vous avez ajouté les autorisations requises au rôle d’exécution, il peut y avoir un délai de plusieurs minutes avant que les modifications ne prennent effet.

Échec de l’authentification SASL

Pour SASL/SCRAM ou SASL/PLAIN, cette erreur indique que les informations d’identification fournies ne sont pas valides.

Le serveur n’a pas réussi à authentifier Lambda

Cette erreur indique que l’agent Kafka n’a pas réussi à authentifier Lambda. Cette erreur se produit dans les conditions suivantes :

- Vous n’avez pas fourni de certificat client pour l’authentification mTLS.
- Vous avez fourni un certificat client, mais les courtiers ne sont pas configurés pour utiliser l’authentification mTLS.
- Les courtiers Kafka ne font pas confiance à un certificat client.

Lambda n’a pas réussi à authentifier le serveur

Cette erreur indique que Lambda n’a pas réussi à authentifier l’agent Kafka. Cette erreur se produit dans les conditions suivantes :

- Les courtiers Kafka utilisent des certificats auto-signés ou une autorité de certification privée, mais n’ont pas fourni le certificat d’autorité de certification racine du serveur.
- Le certificat d’autorité de certification racine du serveur ne correspond pas à l’autorité de certification racine qui a signé le certificat du courtier.
- La validation du nom d’hôte a échoué, car le certificat du courtier ne contient pas le nom DNS ou l’adresse IP du courtier comme autre nom de sujet.

Le certificat ou la clé privée fournis n’est pas valide

Cette erreur indique que le consommateur Kafka n’a pas pu utiliser le certificat ou la clé privée fournis. Assurez-vous que le certificat et la clé utilisent le format PEM et que le chiffrement de la clé privée utilise un algorithme PBES1.

## API de mappage des sources d’événements

Lorsque vous ajoutez votre cluster Apache Kafka en tant que source d’événement pour votre fonction Lambda, si votre fonction rencontre une erreur, votre consommateur Kafka cesse de traiter les registres. Les consommateurs d’une partition de rubrique sont ceux qui s’abonnent à vos registres et qui les lisent et traitent. Vos autres consommateurs Kafka peuvent continuer à traiter les registres, à condition qu’ils ne rencontrent pas la même erreur.

Afin d'identifier les raisons pour lesquelles un consommateur cesse son traitement, vérifiez le champ `StateTransitionReason` dans la réponse de `EventSourceMapping`. La liste suivante décrit les erreurs de source d'événement que vous pouvez recevoir :

**`ESM_CONFIG_NOT_VALID`**

La configuration de mappage de source d'événement n'est pas valide.

**`EVENT_SOURCE_AUTHN_ERROR`**

Lambda n'a pas pu authentifier la source d'événement.

**`EVENT_SOURCE_AUTHZ_ERROR`**

Lambda ne dispose pas des autorisations requises pour accéder à la source d'événement.

**`FUNCTION_CONFIG_NOT_VALID`**

La configuration de la fonction n'est pas valide.

> **Note**
>
> Si vos registres d'événement Lambda dépassent la limite de taille autorisée de 6 Mo, il se peut qu'ils ne soient pas traités.

# Invocation d'une fonction Lambda à l'aide d'un point de terminaison Amazon Gateway API

Vous pouvez créer un site Web API avec un HTTP point de terminaison pour votre fonction Lambda à l'aide d'Amazon API Gateway. APIGateway fournit des outils pour créer et documenter des sites Web APIs qui acheminent les HTTP demandes vers les fonctions Lambda. Vous pouvez sécuriser l'accès à votre compte API grâce à des contrôles d'authentification et d'autorisation. APIsVous pouvez diffuser du trafic sur Internet ou n'être accessible qu'au sein de votreVPC.

Les ressources que vous API utilisez définissent une ou plusieurs méthodes, telles que GET ouPOST. Les méthodes ont une intégration qui achemine les requêtes vers une fonction Lambda ou un autre type d'intégration. Vous pouvez définir chaque ressource et méthode individuellement, ou utiliser des types de ressource et de méthode spéciaux pour correspondre à toutes les demandes adaptées à un modèle. Une ressource proxy attrape tous les chemins sous une ressource. La `ANY` méthode capture toutes les HTTP méthodes.

Sections


- Choix d'un API type
- Ajout d'un point de terminaison public à votre fonction Lambda
- Intégration de proxy
- Format des événements
- Format de la réponse
- Autorisations
- Exemple d'application
- Tutoriel : Utilisation de Lambda avec Gateway API
- Gestion des erreurs Lambda avec une API API Gateway


## Choix d'un API type

APIGateway prend en charge trois types de fonctions APIs qui invoquent des fonctions Lambda :

- HTTPAPI: Léger et à faible latence RESTfulAPI.
- RESTAPI: Un outil personnalisable et riche en fonctionnalités RESTfulAPI.
- WebSocket API: un site Web API qui maintient des connexions permanentes avec les clients pour une communication en duplex intégral.

HTTPAPIset REST APIs sont à la fois RESTful APIs chargés de traiter HTTP les demandes et de renvoyer des réponses. HTTPAPIssont plus récents et sont construits avec la version 2 de API GatewayAPI. Les fonctionnalités suivantes sont nouvelles pour HTTP APIs :

HTTPAPIfonctionnalités

- Déploiements automatiques – Lorsque vous modifiez des routages ou des intégrations, les modifications se déploient automatiquement sur des étapes pour lesquelles le déploiement automatique est activé.
- Étape par défaut : vous pouvez créer une étape par défaut (`$default`) pour traiter les demandes sur le chemin racine API du vôtreURL. Pour les étapes nommées, vous devez inclure le nom de l'étape au début du chemin d'accès.
- CORSconfiguration — Vous pouvez configurer votre système API pour ajouter CORS des en-têtes aux réponses sortantes, au lieu de les ajouter manuellement dans votre code de fonction.

RESTAPIssont les classiques RESTful APIs pris en charge par API Gateway depuis son lancement. RESTAPIsdisposent actuellement de davantage de fonctionnalités de personnalisation, d'intégration et de gestion.

RESTAPIfonctionnalités

- Types d'intégration : REST APIs prend en charge les intégrations Lambda personnalisées. Avec une intégration personnalisée, vous pouvez envoyer uniquement le corps de la requête à la fonction, ou appliquer un modèle de transformation au corps de requête avant de l'envoyer à la fonction.
- Contrôle d'accès : REST APIs prend en charge davantage d'options d'authentification et d'autorisation.
- Surveillance et suivi : REST APIs prise en charge du AWS X-Ray suivi et options de journalisation supplémentaires.

Pour une comparaison détaillée, voir Choisir entre HTTP APIs et REST APIs dans le Guide du développeur de API Gateway.

WebSocket APIsutilise également la version 2 de API Gateway API et prend en charge un ensemble de fonctionnalités similaire. Utilisez un WebSocket API pour les applications qui bénéficient d'une connexion permanente entre le client etAPI. WebSocket APIsfournissent une communication en

duplex intégral, ce qui signifie que le client et le API peuvent envoyer des messages en continu sans attendre de réponse.

HTTPAPIssupporte un format d'événement simplifié (version 2.0). L'exemple suivant montre un événement issu d'un HTTPAPI.

Example APIÉvénement du proxy Gateway (HTTPAPI)

```
{
    "version": "2.0",
    "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
    "rawPath": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
    "rawQueryString": "",
    "cookies": [
        "s_fid=7AABXMPL1AFD9BBF-0643XMPL09956DE2",
        "regStatus=pre-register"
    ],
    "headers": {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
        "accept-encoding": "gzip, deflate, br",
        ...
    },
    "requestContext": {
        "accountId": "123456789012",
        "apiId": "r3pmxmplak",
        "domainName": "r3pmxmplak.execute-api.us-east-2.amazonaws.com",
        "domainPrefix": "r3pmxmplak",
        "http": {
            "method": "GET",
            "path": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
            "protocol": "HTTP/1.1",
            "sourceIp": "205.255.255.176",
            "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
        },
        "requestId": "JKJaXmPLvHcESHA=",
        "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
        "stage": "default",
        "time": "10/Mar/2020:05:16:23 +0000",
        "timeEpoch": 1583817383220
    },
    "isBase64Encoded": true
```

```
}
```

Pour plus d'informations, voir Créer des intégrations de AWS Lambda proxy pour HTTP APIs API Gateway.

## Ajout d’un point de terminaison public à votre fonction Lambda

Ajouter un point de terminaison public à votre fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Sélectionnez APIGateway.
5. Choisissez Créer un API ou Utiliser un existant API.
    a. Nouveau API : Pour le APItype, sélectionnez HTTPAPI. Pour de plus amples informations, veuillez consulter Choix d'un API type.
    b. Existant API : sélectionnez-le dans API la liste déroulante ou entrez l'APIID (par exemple, r3pmxmplak).
6. Pour Sécurité, choisissez Ouvrir.
7. Choisissez Ajouter.

## Intégration de proxy

APIAPIsLes passerelles comprennent des étapes, des ressources, des méthodes et des intégrations. L’étape et la ressource déterminent le chemin du point de terminaison :

APIformat de chemin

- `/prod/` – Etape `prod` et ressource racine.
- `/prod/user` – Etape `prod` et ressource `user`.
- `/dev/{proxy+}` – Routage quelconque à l’étape `dev`.
- `/` — (HTTPAPIs) Le stage par défaut et la ressource racine.

Une intégration Lambda fait correspondre une combinaison de chemins et de HTTP méthodes à une fonction Lambda. Vous pouvez configurer API Gateway pour transmettre le corps de la HTTP demande tel quel (intégration personnalisée) ou pour encapsuler le corps de la demande dans un document contenant toutes les informations de la demande, y compris les en-têtes, la ressource, le chemin et la méthode.

Pour plus d'informations, consultez la section Intégrations de proxy Lambda dans Gateway. API

## Format des événements

Amazon API Gateway appelle votre fonction de manière synchrone avec un événement contenant une JSON représentation de la HTTP demande. Pour une intégration personnalisée, l'événement est le corps de la requête. Pour une intégration par proxy, l'événement a une structure définie. L'exemple suivant montre un événement proxy provenant d'une API passerelle RESTAPI.

Example APIÉvénement du proxy Gateway (RESTAPI)

```
{
      "resource": "/",
      "path": "/",
      "httpMethod": "GET",
      "requestContext": {
          "resourcePath": "/",
          "httpMethod": "GET",
          "path": "/Prod/",
          ...
      },
      "headers": {
          "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
          "accept-encoding": "gzip, deflate, br",
          "Host": "70ixmpl4fl.execute-api.us-east-2.amazonaws.com",
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36",
          "X-Amzn-Trace-Id": "Root=1-5e66d96f-7491f09xmpl79d18acf3d050",
          ...
      },
      "multiValueHeaders": {
          "accept": [
              "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
          ],
```

```
          "accept-encoding": [
              "gzip, deflate, br"
          ],
          ...
      },
      "queryStringParameters": null,
      "multiValueQueryStringParameters": null,
      "pathParameters": null,
      "stageVariables": null,
      "body": null,
      "isBase64Encoded": false
  }
```

## Format de la réponse

APIGateway attend une réponse de votre fonction et transmet le résultat à l'appelant. Pour une intégration personnalisée, vous définissez une réponse d'intégration et une réponse de méthode pour convertir le résultat de la fonction en HTTP réponse. Pour une intégration par proxy, la fonction doit répondre avec une représentation de la réponse dans un format spécifique.

L’exemple suivant montre un objet de réponse d’une fonction Node.js. L'objet de réponse représente une HTTP réponse réussie contenant un JSON document.

Example index.js : objet de réponse d'intégration de proxy (Node.js)

```
var response = {
      "statusCode": 200,
      "headers": {
        "Content-Type": "application/json"
      },
      "isBase64Encoded": false,
      "multiValueHeaders": {
        "X-Custom-Header": ["My value", "My other value"],
      },
      "body": "{\n  \"TotalCodeSize\": 104330022,\n  \"FunctionCount\": 26\n}"
    }
```

Le moteur d'exécution Lambda sérialise l'objet de réponse JSON et l'envoie au. API Il API analyse la réponse et l'utilise pour créer une HTTP réponse, qu'il envoie ensuite au client qui a fait la demande initiale.

Example HTTPréponse

```
< HTTP/1.1 200 OK
  < Content-Type: application/json
  < Content-Length: 55
  < Connection: keep-alive
  < x-amzn-RequestId: 32998fea-xmpl-4268-8c72-16138d629356
  < X-Custom-Header: My value
  < X-Custom-Header: My other value
  < X-Amzn-Trace-Id: Root=1-5e6aa925-ccecxmplbae116148e52f036
  <
  {
    "TotalCodeSize": 104330022,
    "FunctionCount": 26
  }
```

## Autorisations

Amazon API Gateway obtient l'autorisation d'appeler votre fonction conformément à la politique basée sur les ressources de la fonction. Vous pouvez accorder une autorisation d'appel à un ensemble API ou accorder un accès limité à une étape, une ressource ou une méthode.

Lorsque vous ajoutez un API à votre fonction à l'aide de la console Lambda, de la console API Gateway ou d'un AWS SAM modèle, la politique basée sur les ressources de la fonction est automatiquement mise à jour. Voici un exemple de politique de fonction.

Example stratégie de fonction

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLXE2F",
      "Effect": "Allow",
      "Principal": {
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:nodejs-apig-
function-1G3MXMPLXVXYI",
      "Condition": {
```

```
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:execute-api:us-east-2:111122223333:ktyvxmpls1/*/
GET/"
        }
      }
    }
  ]
}
```

Vous pouvez gérer les autorisations relatives aux politiques fonctionnelles manuellement en effectuant les API opérations suivantes :

- AddPermission
- RemovePermission
- GetPolicy

Pour accorder l'autorisation d'invocation à un objet existantAPI, utilisez la `add-permission` commande. Exemple :

```
aws lambda add-permission \
  --function-name my-function \
  --statement-id apigateway-get --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET/"
```

Vous devriez voir la sortie suivante :

```
{
    "Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-east-2:123456789012:function:my-function
\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-
east-2:123456789012:mnh1xmpli7/default/GET\"}}}"
}
```

> **Note**
>
> Si votre fonction est différente Régions AWS, l'identifiant de région dans la source ARN doit correspondre à la région de la fonction, et non à la région duAPI. API Lorsque API Gateway ARN invoque une fonction, il utilise une ressource basée sur le ARNAPI, mais modifiée pour correspondre à la région de la fonction.

ARNDans cet exemple, la source autorise une intégration sur la GET méthode de la ressource racine dans l'étape par défaut d'unAPI, avec IDmnh1xmpli7. Vous pouvez utiliser un astérisque dans la source ARN pour accorder des autorisations à plusieurs étapes, méthodes ou ressources.

Modèles de ressources

- `mnh1xmpli7/*/GET/*`— GET méthode sur toutes les ressources à toutes les étapes.
- `mnh1xmpli7/prod/ANY/user`— ANY méthode appliquée à la `user` ressource dans le `prod` stage.
- `mnh1xmpli7/*/*/*` – Toute méthode sur toutes les ressources à toutes les étapes.

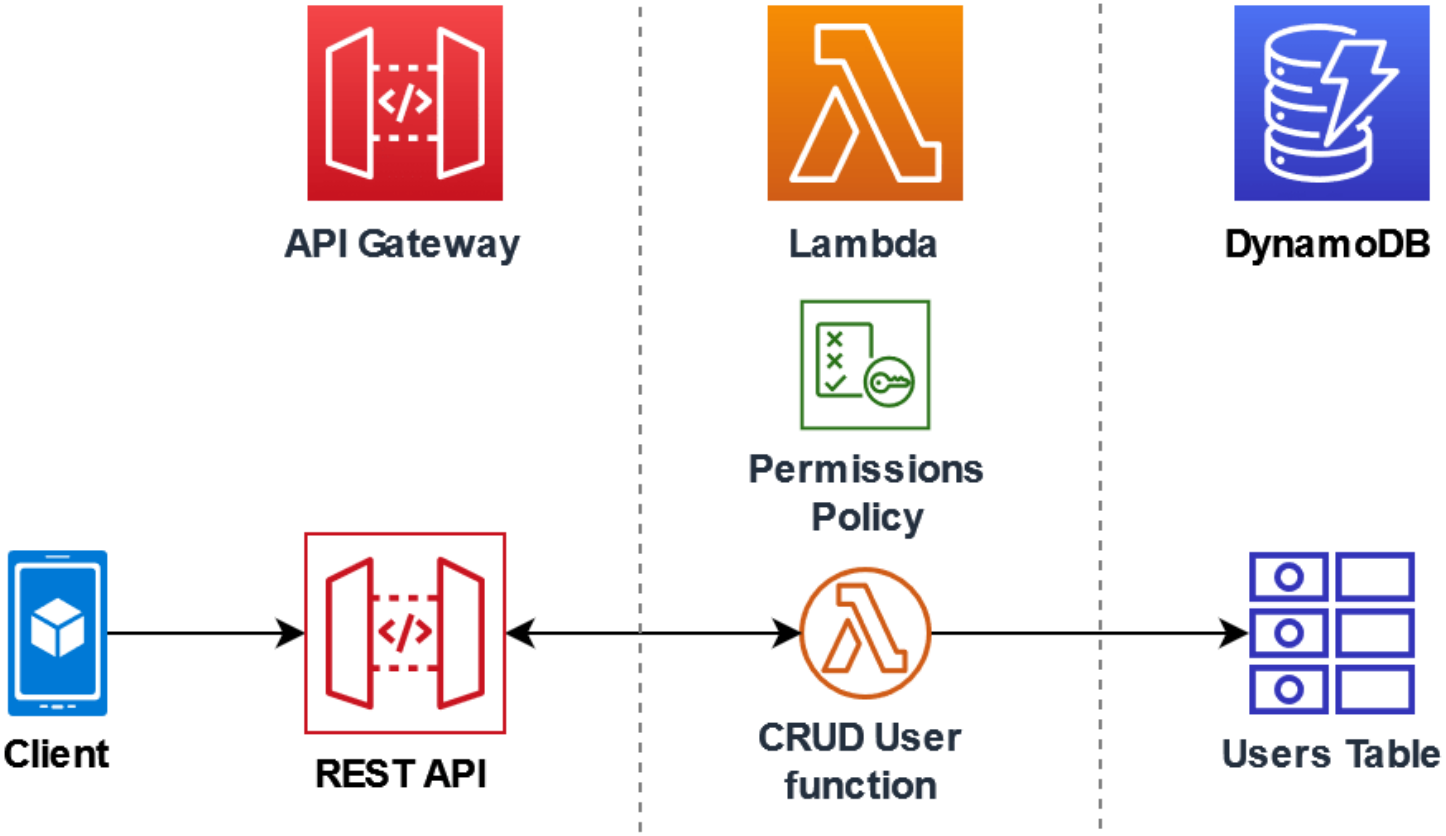
Pour de plus amples informations sur l'affichage de la politique et la suppression des instructions, veuillez consulter Utilisation de IAM politiques basées sur les ressources dans Lambda.

## Exemple d'application

L'exemple d'application APIGateway with Node.js inclut une fonction avec un AWS SAM modèle qui crée un fichier REST API dont le AWS X-Ray suivi est activé. Il inclut également des scripts pour le déploiement, l'appel de la fonctionAPI, le test et le nettoyage.

## Tutoriel : Utilisation de Lambda avec Gateway API

Dans ce didacticiel, vous allez créer un REST API via lequel vous appelez une fonction Lambda à l'aide d'une HTTP requête. Votre fonction Lambda effectuera des opérations de création, de lecture, de mise à jour et de suppression (CRUD) sur une table DynamoDB. Cette fonction est fournie ici à titre de démonstration, mais vous apprendrez à configurer une API passerelle REST API capable d'invoquer n'importe quelle fonction Lambda.




L'utilisation de API Gateway fournit aux utilisateurs un point de HTTP terminaison sécurisé pour appeler votre fonction Lambda et peut aider à gérer de gros volumes d'appels vers votre fonction en limitant le trafic et en validant et en autorisant automatiquement les appels. API APIGateway fournit également des contrôles de sécurité flexibles à l'aide de AWS Identity and Access Management (IAM) et Amazon Cognito. Ceci est utile pour les cas d'utilisation où une autorisation préalable est requise pour les appels à votre application.

Pour réaliser ce tutoriel, vous passerez par les étapes suivantes :

1. Créer et configurer une fonction Lambda en Python ou Node.js pour effectuer des opérations sur une table DynamoDB.
2. Créez une API passerelle REST API intégrée pour vous connecter à votre fonction Lambda.
3. Créer une table DynamoDB et la tester avec votre fonction Lambda dans la console.
4. Déployez votre installation API et testez-la complète à l'aide de curl dans un terminal.

En effectuant ces étapes, vous apprendrez à utiliser API Gateway pour créer un HTTP point de terminaison capable d'invoquer en toute sécurité une fonction Lambda à n'importe quelle échelle.

Vous apprendrez également comment déployer le vôtreAPI, comment le tester dans la console et en envoyant une HTTP demande à l'aide d'un terminal.

Sections


- Prérequis
- Créer une stratégie d’autorisations
- Créer un rôle d’exécution
- Créer la fonction
- Invoquez la fonction à l'aide du AWS CLI
- Création d'une API passerelle REST API d'utilisation
- Créez une ressource sur votre REST API
- Création d'une HTTP POST méthode
- Créez une table DynamoDB
- Testez l'intégration de API Gateway, Lambda et DynamoDB
- Déployez le API
- Utilisez curl pour appeler votre fonction à l'aide de requêtes HTTP
- Nettoyer vos ressources (facultatif)


## Prérequis

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l‘utilisateur racine a accès à l‘ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un

utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu‘utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d’un ensemble d’autorisations dans le Guide de l’utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l’utilisateur AWS IAM Identity Center .

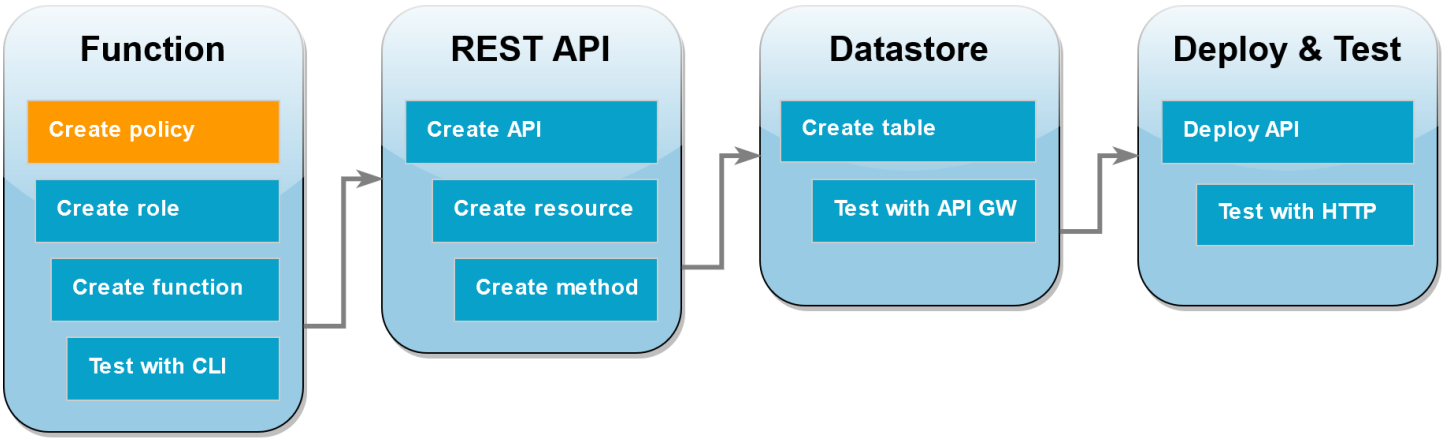
Installez le AWS Command Line Interface

Si vous ne l'avez pas encore installé AWS Command Line Interface, suivez les étapes décrites dans la section Installation ou mise à jour de la dernière version du AWS CLI pour l'installer.

Ce tutoriel nécessite un terminal de ligne de commande ou un shell pour exécuter les commandes. Sous Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d’obtenir une version intégrée à Windows d’Ubuntu et Bash.

## Créer une stratégie d’autorisations




Avant de créer un rôle d'exécution pour votre fonction Lambda, vous devez d'abord créer une politique d'autorisation pour autoriser votre fonction à accéder aux ressources requises AWS . Pour ce didacticiel, la politique permet à Lambda d'effectuer des CRUD opérations sur une table DynamoDB et d'écrire dans Amazon Logs. CloudWatch

Pour créer la politique

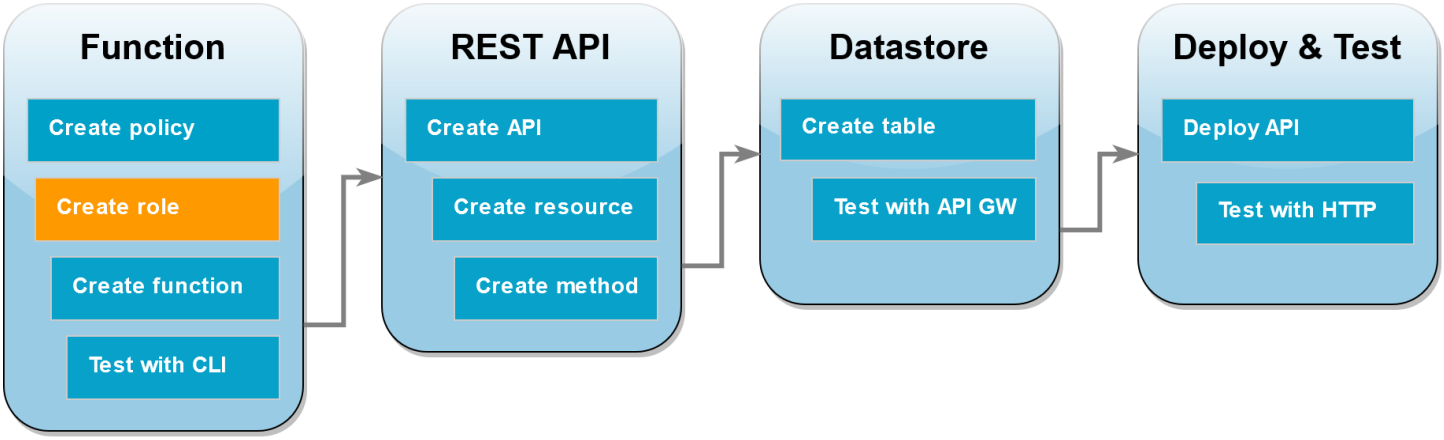
1. Ouvrez la page Politiques de la IAM console.
2. Choisissez Create Policy (Créer une politique).
3. Choisissez l'JSONonglet, puis collez la politique personnalisée suivante dans l'JSONéditeur.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1428341300017",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
```

```
      "Sid": "",
      "Resource": "*",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow"
    }
  ]
}
```

4. Choisissez Suivant : Balises.
5. Choisissez Suivant : Vérification.
6. Sous Examiner une stratégie, pour le Nom de la stratégie, saisissez **`lambda-apigateway-policy`**.
7. Choisissez Créer une stratégie.

## Créer un rôle d’exécution




Un rôle d'exécution est un rôle AWS Identity and Access Management (IAM) qui accorde à une fonction Lambda l'autorisation d'accéder aux AWS services et aux ressources. Pour permettre à votre fonction d’effectuer des opérations sur une table DynamoDB, vous attachez la politique d’autorisation que vous avez créée à l’étape précédente.

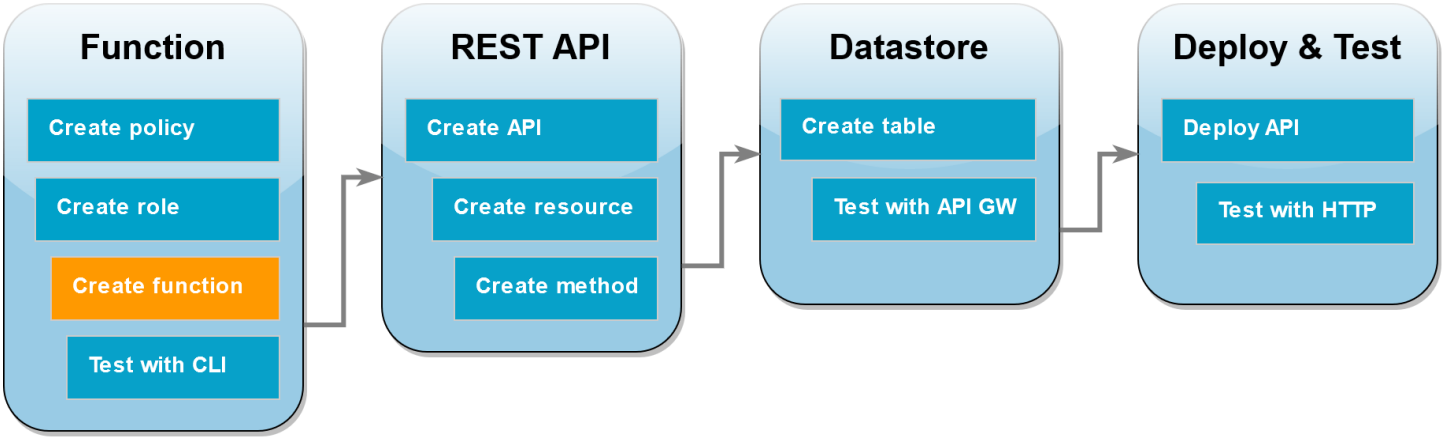
Pour créer un rôle d’exécution et attacher votre politique d’autorisations personnalisée

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez Créer un rôle.

3. Pour le type d'entité de confiance, choisissez Service AWS , puis pour le cas d'utilisation, choisissez Lambda.
4. Choisissez Suivant.
5. Dans la zone de recherche de stratégie, entrez **`lambda-apigateway-policy`**.
6. Dans les résultats de la recherche, sélectionnez la stratégie que vous avez créée (`lambda-apigateway-policy`), puis choisissez Suivant.
7. Sous Role details (Détails du rôle), pour Role name (Nom du rôle), saisissez **`lambda-apigateway-role`**, puis sélectionnez Create role (Créer un rôle).

Plus loin dans le didacticiel, vous aurez besoin du nom de ressource Amazon (ARN) du rôle que vous venez de créer. Sur la page Rôles de la IAM console, choisissez le nom de votre rôle (`lambda-apigateway-role`) et copiez le rôle ARN affiché sur la page Résumé.

## Créer la fonction




L'exemple de code suivant reçoit une entrée d'événement de API Gateway spécifiant une opération à effectuer sur la table DynamoDB que vous allez créer et certaines données de charge utile. Si les paramètres que la fonction reçoit sont valides, elle exécute l'opération demandée sur la table.

Node.js

Example index.mjs

Notez le `region` réglage. Cela doit correspondre à l' Région AWS endroit où vous déployez la fonction et créez la table DynamoDB.

```
import { DynamoDBDocumentClient, PutCommand, GetCommand,
         UpdateCommand, DeleteCommand} from "@aws-sdk/lib-dynamodb";
```

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const ddbClient = new DynamoDBClient({ region: "us-east-2" });
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);

// Define the name of the DDB table to perform the CRUD operations on
const tablename = "lambda-apigateway";

/**
 * Provide an event that contains the following keys:
 *
 *   - operation: one of 'create,' 'read,' 'update,' 'delete,' or 'echo'
 *   - payload: a JSON object containing the parameters for the table item
 *     to perform the operation on
 */
export const handler = async (event, context) => {

     const operation = event.operation;

     if (operation == 'echo'){
          return(event.payload);
     }

    else {
        event.payload.TableName = tablename;
        let response;

        switch (operation) {
          case 'create':
               response = await ddbDocClient.send(new PutCommand(event.payload));
               break;
          case 'read':
               response = await ddbDocClient.send(new GetCommand(event.payload));
               break;
          case 'update':
               response = ddbDocClient.send(new UpdateCommand(event.payload));
               break;
          case 'delete':
               response = ddbDocClient.send(new DeleteCommand(event.payload));
               break;
          default:
            response = 'Unknown operation: ${operation}';
          }
        console.log(response);
```

```
        return response;
    }
};
```

> **Note**
>
> Dans cet exemple, le nom de la table DynamoDB est défini comme une variable dans le code de votre fonction. Dans une application réelle, passer ce paramètre comme une variable d'environnement et éviter de coder en dur le nom de la table constitue une bonne pratique. Pour plus d'informations, consultez la section Utilisation de variables d' AWS Lambda environnement.

Pour créer la fonction

1. Enregistrez l'exemple de code sous forme de fichier nommé `index.mjs` et, si nécessaire, modifiez la AWS région spécifiée dans le code. La région spécifiée dans le code doit être la même que celle dans laquelle vous créez votre table DynamoDB plus loin dans le tutoriel.
2. Créez un package de déploiement à l'aide de la commande `zip` suivante.

   ```
   zip function.zip index.mjs
   ```

3. Créez une fonction Lambda à l'aide de la `create-function` AWS CLI commande. Pour le `role` paramètre, entrez le nom de ressource Amazon (ARN) du rôle d'exécution que vous avez copié précédemment.

   ```
   aws lambda create-function \
   --function-name LambdaFunctionOverHttps \
   --zip-file fileb://function.zip \
   --handler index.handler \
   --runtime nodejs20.x \
   --role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
   ```

Python 3

Example LambdaFunctionOverHttps.py

```
import boto3
```

```
# Define the DynamoDB table that Lambda will connect to
table_name = "lambda-apigateway"

# Create the DynamoDB resource
dynamo = boto3.resource('dynamodb').Table(table_name)

# Define some functions to perform the CRUD operations
def create(payload):
    return dynamo.put_item(Item=payload['Item'])

def read(payload):
    return dynamo.get_item(Key=payload['Key'])

def update(payload):
    return dynamo.update_item(**{k: payload[k] for k in ['Key', 'UpdateExpression',
    'ExpressionAttributeNames', 'ExpressionAttributeValues'] if k in payload})

def delete(payload):
    return dynamo.delete_item(Key=payload['Key'])

def echo(payload):
    return payload

operations = {
    'create': create,
    'read': read,
    'update': update,
    'delete': delete,
    'echo': echo,
}

def lambda_handler(event, context):
    '''Provide an event that contains the following keys:
      - operation: one of the operations in the operations dict below
      - payload: a JSON object containing parameters to pass to the
        operation being performed
    '''

    operation = event['operation']
    payload = event['payload']

    if operation in operations:
        return operations[operation](payload)
```

```
    else:
        raise ValueError(f'Unrecognized operation "{operation}"')
```

> **Note**
>
> Dans cet exemple, le nom de la table DynamoDB est défini comme une variable dans le code de votre fonction. Dans une application réelle, passer ce paramètre comme une variable d'environnement et éviter de coder en dur le nom de la table constitue une bonne pratique. Pour plus d'informations, consultez la section Utilisation de variables d' AWS Lambda environnement.

Pour créer la fonction

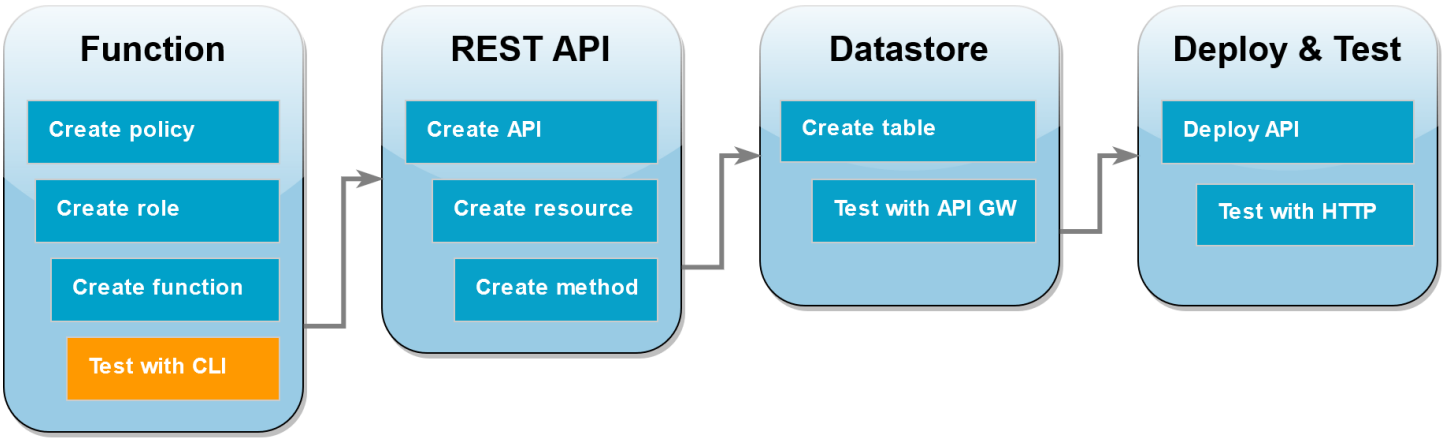
1. Enregistrez l'exemple de code en tant que fichier nommé `LambdaFunctionOverHttps.py`.
2. Créez un package de déploiement à l'aide de la commande `zip` suivante.

```
zip function.zip LambdaFunctionOverHttps.py
```

3. Créez une fonction Lambda à l'aide de la `create-function` AWS CLI commande. Pour le `role` paramètre, entrez le nom de ressource Amazon (ARN) du rôle d'exécution que vous avez copié précédemment.

```
aws lambda create-function \
--function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip \
--handler LambdaFunctionOverHttps.lambda_handler \
--runtime python3.12 \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

## Invoquez la fonction à l'aide du AWS CLI




Avant d'intégrer votre fonction à API Gateway, vérifiez que vous l'avez déployée avec succès. Créez un événement de test contenant les paramètres que votre API passerelle API enverra à Lambda et utilisez la AWS CLI `invoke` commande pour exécuter votre fonction.

Pour appeler la fonction Lambda avec AWS CLI

1. Enregistrez le fichier suivant JSON sous forme de fichier nomméinput.txt.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Exécutez la commande suivante `invoke` AWS CLI .

```
aws lambda invoke \
--function-name LambdaFunctionOverHttps \
--payload file://input.txt outputfile.txt \
--cli-binary-format raw-in-base64-out
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

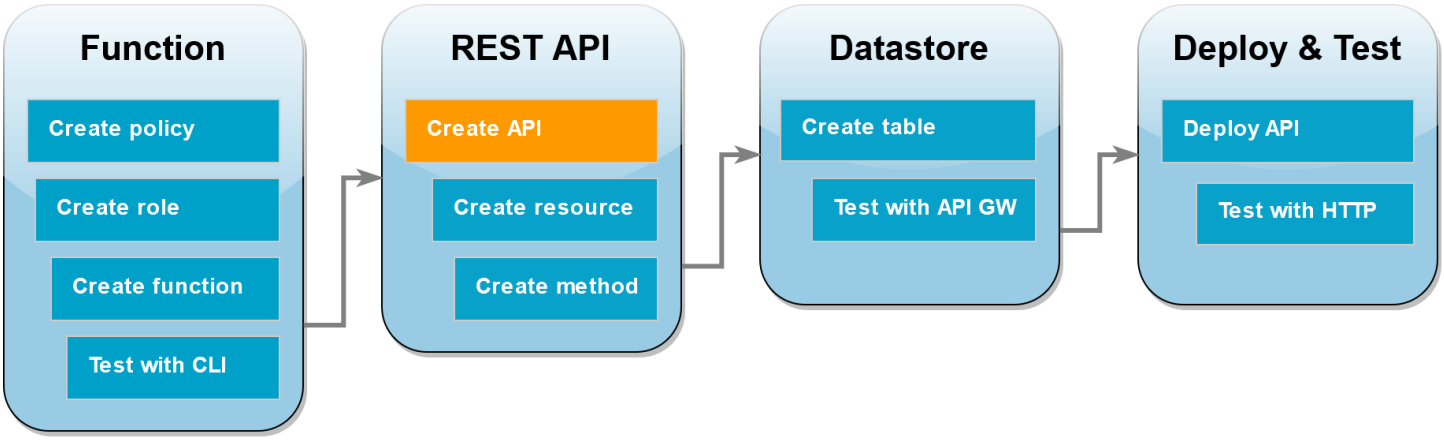
Vous devriez voir la réponse suivante :

```
{
"StatusCode": 200,
"ExecutedVersion": "LATEST"
}
```

3. Vérifiez que votre fonction a effectué l'`echo`opération que vous avez spécifiée lors de l'événement de JSON test. Inspectez le fichier `outputfile.txt` et vérifiez qu'il contient les éléments suivants :

```
{"somekey1": "somevalue1", "somekey2": "somevalue2"}
```

## Création d'une API passerelle REST API d'utilisation

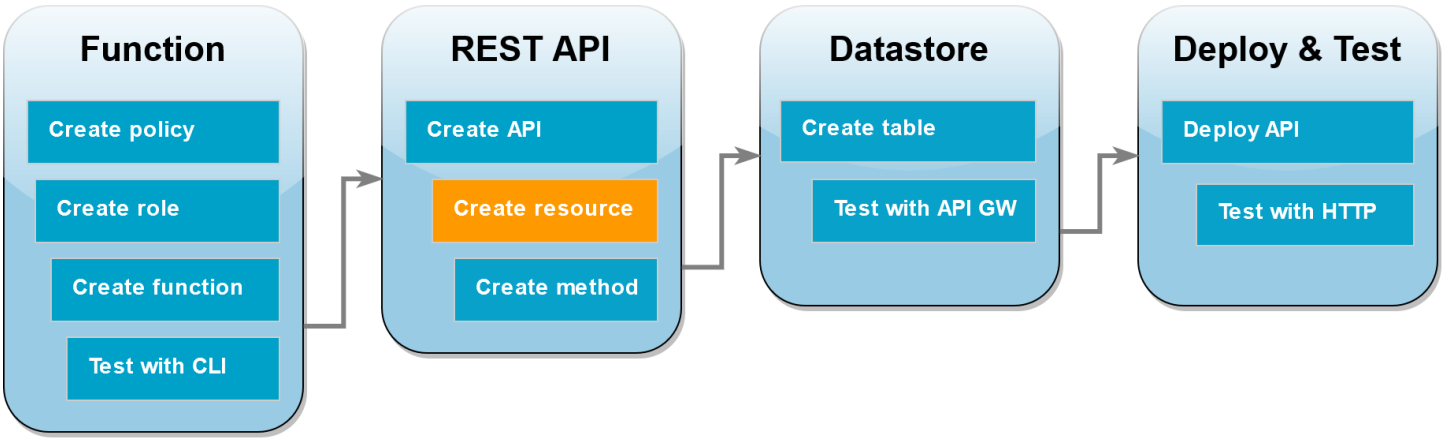



Au cours de cette étape, vous créez la API passerelle REST API que vous utiliserez pour appeler votre fonction Lambda.

Pour créer le API

1. Ouvrez la console API Gateway.
2. Choisissez Create API.
3. Dans le RESTAPIchamp, choisissez Build.
4. Dans APIDétails, laissez Nouveau API sélectionné, et dans APINom, entrez**`DynamoDBOperations`**.
5. Choisissez Create API.

## Créez une ressource sur votre REST API

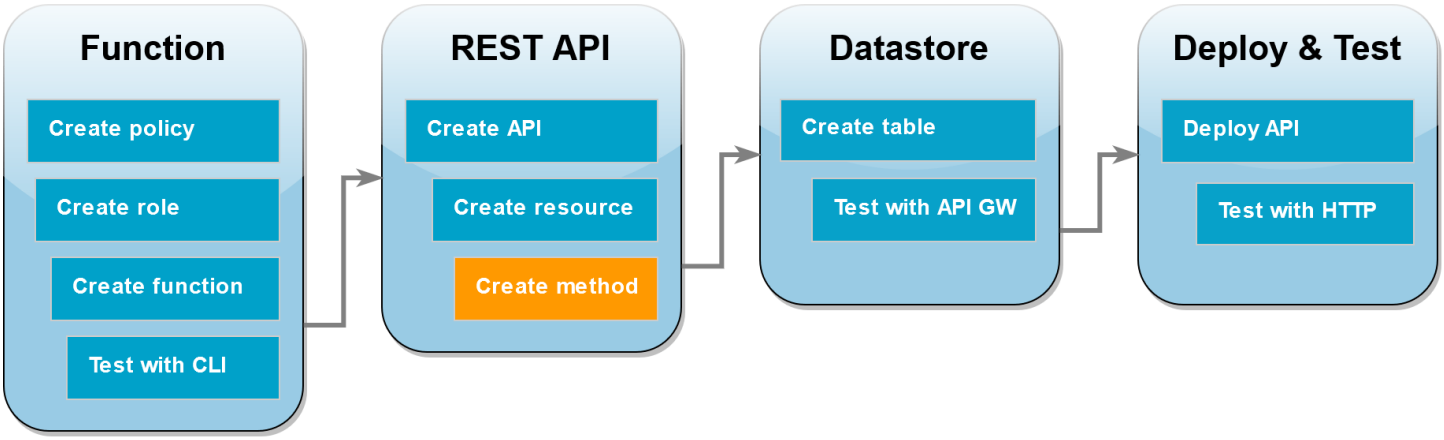



Pour ajouter une HTTP méthode à votreAPI, vous devez d'abord créer une ressource pour que cette méthode puisse fonctionner. Ici, vous créez la ressource pour gérer votre table DynamoDB.

Pour créer la ressource

1. Dans la console API Gateway, sur la page Ressources de votreAPI, choisissez Create Resource.
2. Dans Détails de la ressource, pour Nom de la ressource saisissez **`DynamoDBManager`**.
3. Choisissez Create Resource.

## Création d'une HTTP POST méthode



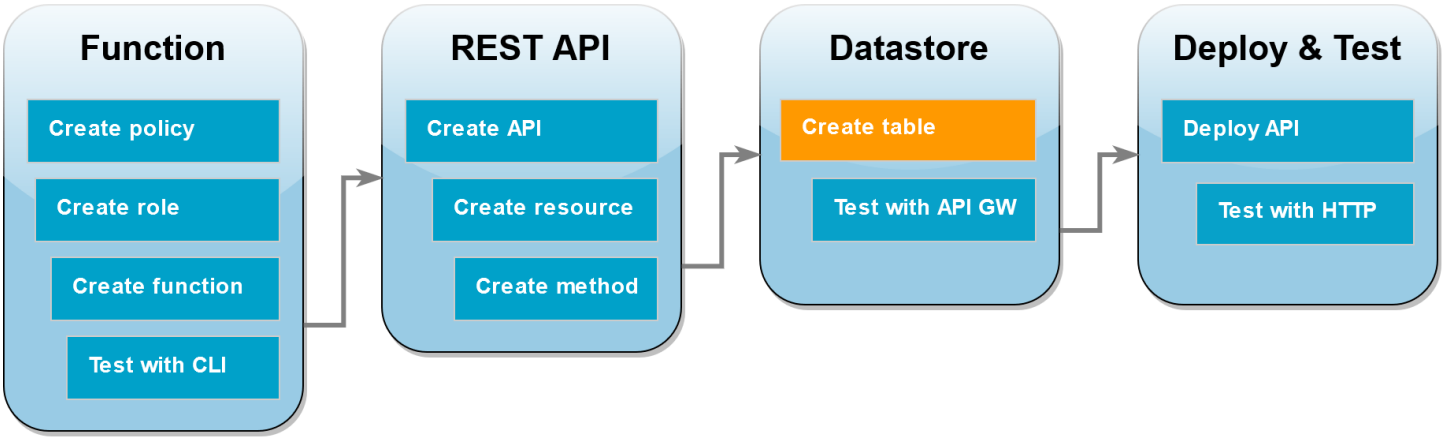

Dans cette étape, vous créez une méthode (`POST`) pour votre ressource `DynamoDBManager`. Vous liez cette `POST` méthode à votre fonction Lambda de sorte que lorsque la méthode reçoit une HTTP demande, API Gateway invoque votre fonction Lambda.

> **Note**
>
> Dans le cadre de ce didacticiel, une HTTP méthode (`POST`) est utilisée pour appeler une seule fonction Lambda qui exécute toutes les opérations sur votre table DynamoDB. Dans une application réelle, la meilleure pratique consiste à utiliser une fonction et une HTTP méthode Lambda différentes pour chaque opération. Pour plus d'informations, consultez Le monolithe Lambda dans Serverless Land.

Pour créer la méthode POST

1. Sur la page Ressources de votre choixAPI, assurez-vous que la `/DynamoDBManager` ressource est surlignée. Ensuite, dans le volet Méthodes, choisissez Create Method (Créer une méthode).
2. Dans Type de méthode, sélectionnez POST.
3. Pour Type d'intégration, laissez la Fonction Lambda sélectionnée.
4. Pour la fonction Lambda, choisissez le nom de ressource Amazon (ARN) pour votre fonction (`LambdaFunctionOverHttps`).
5. Choisissez Créer une méthode.

## Créez une table DynamoDB



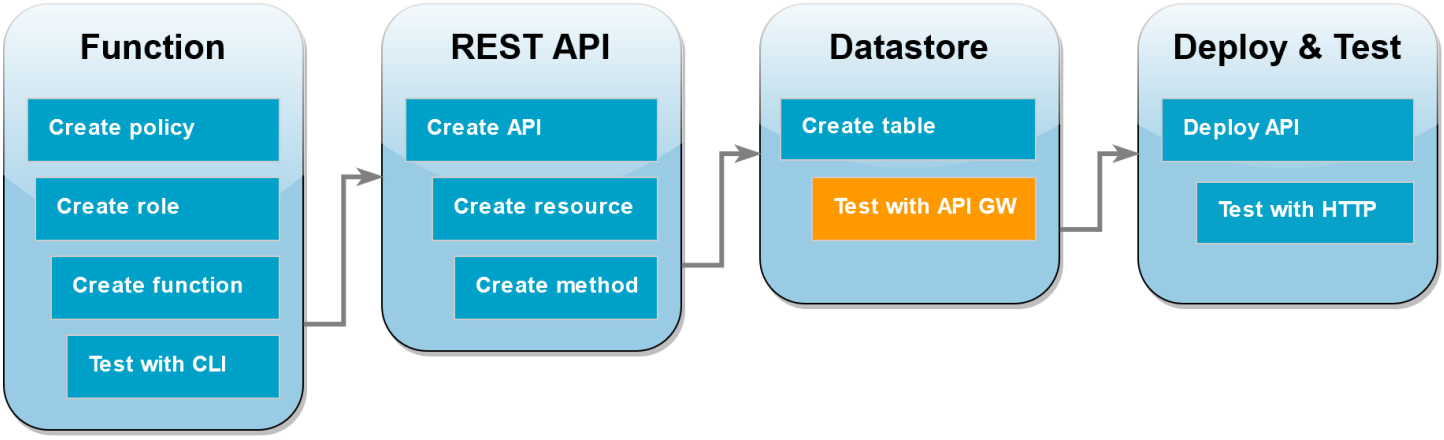

Créez une table DynamoDB vide sur laquelle votre fonction Lambda effectuera des opérations. CRUD

Créer le tableau DynamoDB

1. Ouvrez la page Tables (Tables) de la console DynamoDB.

2. Choisissez Créer un tableau.
3. Sous Détails du tableau, procédez comme suit :
    1. Sous Nom du tableau, saisissez **`lambda-apigateway`**.
    2. Pour Clé de partition, saisissez **`id`**, et conservez le type de données défini comme Chaîne.
4. Sous Table settings (Paramètres de la table), conservez les Default settings (Paramètres par défaut).
5. Choisissez Créer un tableau.

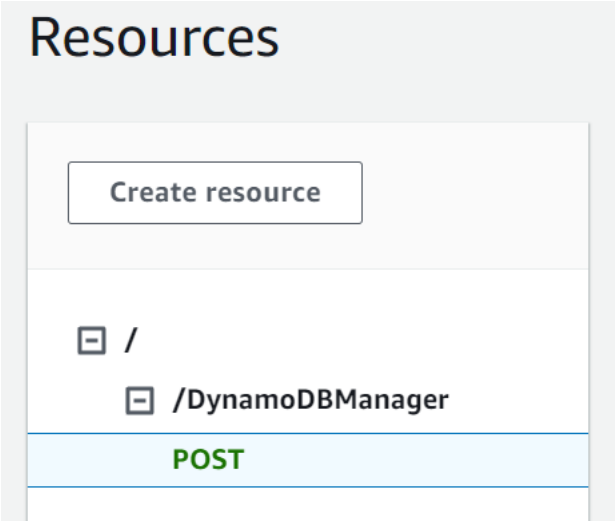
## Testez l'intégration de API Gateway, Lambda et DynamoDB




Vous êtes maintenant prêt à tester l'intégration de votre API méthode API Gateway à votre fonction Lambda et à votre table DynamoDB. À l'aide de la console API Gateway, vous envoyez des demandes directement à votre `POST` méthode à l'aide de la fonction de test de la console. Dans cette étape, vous utilisez d'abord une opération `create` pour ajouter un nouvel élément à votre table DynamoDB, puis vous utilisez une opération `update` pour modifier l'élément.

Test 1 : Pour créer un nouvel élément dans votre table DynamoDB

1. Dans la console API Gateway, choisissez votre API (`DynamoDBOperations`).
2. Choisissez la `POST`méthode sous la `DynamoDBManager` ressource.




3. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
4. Sous Méthode de test, laissez les Chaînes de requête et les En-têtes vides. Pour le corps de la demande, collez ce qui suit JSON :

```
{
  "operation": "create",
  "payload": {
    "Item": {
      "id": "1234ABCD",
      "number": 5
    }
  }
}
```

5. Sélectionnez Tester).

   Les résultats qui s'affichent à la fin du test doivent indiquer le statut `200`. Ce code d'état indique que l'opération `create` a réussi.

   Pour confirmer, vérifiez que votre table DynamoDB contient maintenant le nouvel élément.
6. Ouvrez la page Tables de la console DynamoDB et choisissez la table `lambda-apigateway`.
7. Choisissez Explore table items (Explorer les éléments de la table). Dans le volet Items returned (Éléments retournés), vous devriez voir un élément avec l'id `1234ABCD` et le numéro `5`.

Test 2 : Pour mettre à jour l'élément dans votre table DynamoDB

1. Dans la console API Gateway, retournez à l'onglet Test de votre POST méthode.

2. Sous Méthode de test, laissez les Chaînes de requête et les En-têtes vides. Pour le corps de la demande, collez ce qui suit JSON :

```
{
    "operation": "update",
    "payload": {
        "Key": {
            "id": "1234ABCD"
        },
        "UpdateExpression": "SET #num = :newNum",
        "ExpressionAttributeNames": {
            "#num": "number"
        },
        "ExpressionAttributeValues": {
            ":newNum": 10
        }
    }
}
```

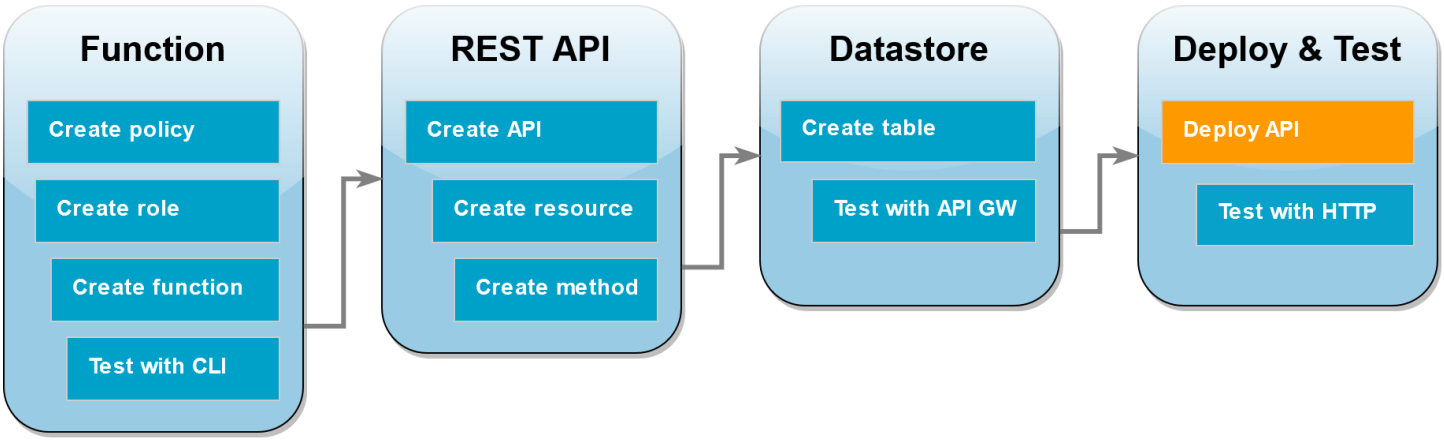
3. Sélectionnez Tester).

   Les résultats qui s'affichent à la fin du test devraient montrer l'état `200`. Ce code d'état indique que l'opération `update` a réussi.

   Pour confirmer, vérifiez que l'élément dans votre table DynamoDB a été modifié.

4. Ouvrez la page Tables de la console DynamoDB et choisissez la table `lambda-apigateway`.
5. Choisissez Explore table items (Explorer les éléments de la table). Dans le volet Items returned (Éléments retournés), vous devriez voir un élément avec l'id `1234ABCD` et le numéro `10`.

## Déployez le API

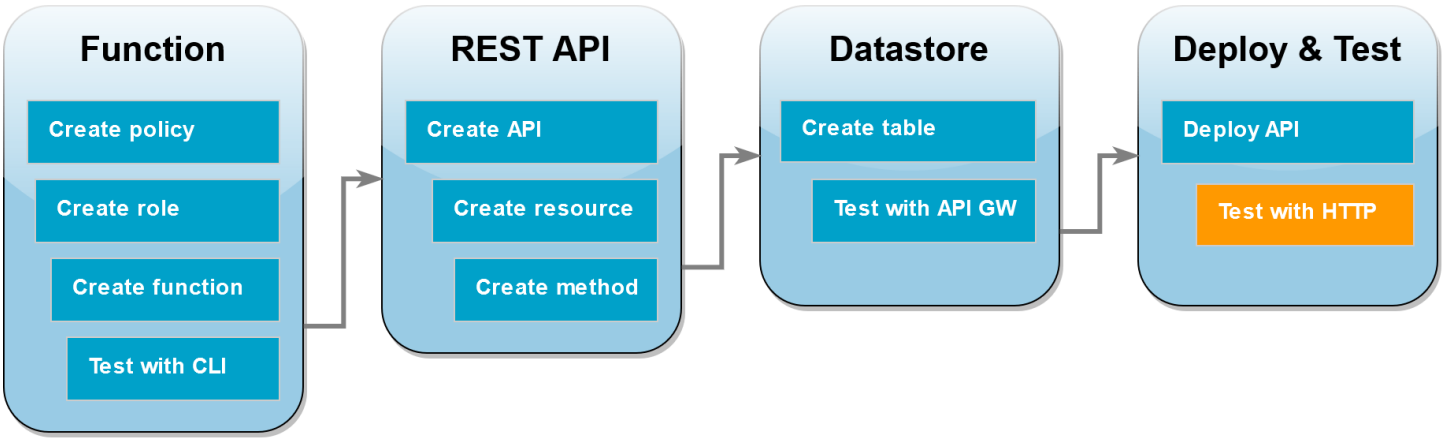



Pour qu'un client puisse appeler leAPI, vous devez créer un déploiement et une étape associée. Une étape représente un aperçu de vous, API y compris de ses méthodes et de ses intégrations.

Pour déployer l'API

1. Ouvrez la APIspage de la console API Gateway et choisissez le `DynamoDBOperations`API.
2. Sur la page Ressources, API choisissez Déployer API.
3. Pour Stage (Étape), choisissez *New stage* (Nouvelle étape), puis pour Stage name (Nom de l'étape), saisissez **`test`**.
4. Choisissez Deploy (Déployer).
5. Dans le volet des détails de l'étape, copiez l'Invoke URL. Vous l'utiliserez à l'étape suivante pour appeler votre fonction à l'aide d'une HTTP requête.

## Utilisez curl pour appeler votre fonction à l'aide de requêtes HTTP



Vous pouvez désormais appeler votre fonction Lambda en envoyant une HTTP demande à votre. API Au cours de cette étape, vous allez créer un nouvel élément dans votre table DynamoDB, puis effectuer des opérations de lecture, de mise à jour et de suppression sur cet élément.

Pour créer un élément dans votre table DynamoDB à l'aide de curl

1. Exécutez la `curl` commande suivante en utilisant l'appel que URL vous avez copié à l'étape précédente. Lorsque vous utilisez curl avec l'option `-d` (data), la HTTP POST méthode est automatiquement utilisée.

   ```
   curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
   -d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
   ```

   Si l'opération a réussi, vous devriez voir une réponse renvoyée avec un code d'HTTPétat de 200.
2. Vous pouvez également utiliser la console DynamoDB pour vérifier que le nouvel élément figure dans votre tableau en procédant comme suit :

   1. Ouvrez la page Tables de la console DynamoDB et choisissez la table `lambda-apigateway`.
   2. Sélectionnez Explore table items (Explorer les éléments de la table). Dans le volet Items returned (Éléments retournés), vous devriez voir un élément avec l'id `5678EFGH` et le numéro `15`.

Pour lire l'élément de votre table DynamoDB à l'aide de curl

- Exécutez la `curl` commande suivante pour lire la valeur de l'élément que vous venez de créer. Utilisez votre propre invocationURL.

  ```
  curl https://avos4dr2rk.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager -d
   \
  '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
  ```

  Vous devriez obtenir un résultat semblable à l'un des suivants selon que vous avez choisi le code de fonction Node.js ou Python :

Node.js

```
{"$metadata":
{"httpStatusCode":200,"requestId":"7BP3G5Q0C0O1E50FBQI9NS099JVV4KQNSO5AEMVJF66Q9ASUAAJG"
"attempts":1,"totalRetryDelay":0},"Item":{"id":"5678EFGH","number":15}}
```

Python

```
{"Item":{"id":"5678EFGH","number":15},"ResponseMetadata":
{"RequestId":"QNDJICE52E86B82VETR6RKBE5BVV4KQNSO5AEMVJF66Q9ASUAAJG",
"HTTPStatusCode":200,"HTTPHeaders":{"server":"Server","date":"Wed, 31 Jul 2024
 00:37:01 GMT","content-type":"application/x-amz-json-1.0",
"content-length":"52","connection":"keep-alive","x-amzn-
requestid":"QNDJICE52E86B82VETR6RKBE5BVV4KQNSO5AEMVJF66Q9ASUAAJG","x-amz-
crc32":"2589610852"},
"RetryAttempts":0}}
```

Pour mettre à jour l'élément de votre table DynamoDB à l'aide de curl

1. Exécutez la `curl` commande suivante pour mettre à jour l'élément que vous venez de créer en modifiant sa `number` valeur. Utilisez votre propre invocationURL.

```
curl https://avos4dr2rk.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "update", "payload": {"Key": {"id": "5678EFGH"},
 "UpdateExpression": "SET #num = :new_value", "ExpressionAttributeNames": {"#num":
 "number"}, "ExpressionAttributeValues": {":new_value": 42}}}'
```

2. Pour confirmer que la valeur de `number` pour l'élément a été mise à jour, exécutez une autre commande de lecture :

```
curl https://avos4dr2rk.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

Pour supprimer l'élément de votre table DynamoDB à l'aide de curl

1. Exécutez la commande `curl` suivante pour supprimer l'élément que vous venez de créer. Utilisez votre propre invocationURL.

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
```

2. Confirmez que l'opération de suppression a réussi. Dans le volet Items returned (Éléments retournés) de la page Explore items (Explorer les éléments) de la console DynamoDB, vérifiez que l'élément avec l'id `5678EFGH` n'est plus dans la table.

## Nettoyer vos ressources (facultatif)

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le API

1. Ouvrez la APIspage de la console API Gateway.
2. Sélectionnez le API que vous avez créé.
3. Sélectionnez Actions, Supprimer.
4. Sélectionnez Delete (Supprimer).

Pour supprimer la table DynamoDB

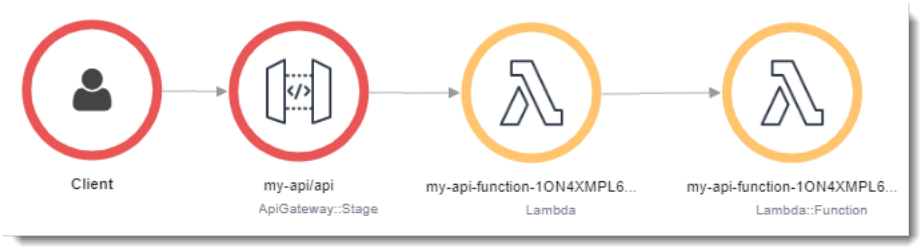
1. Ouvrez la page Tables (Tables) de la console DynamoDB.
2. Sélectionnez la table que vous avez créée.
3. Choisissez Supprimer.
4. Saisissez **`delete`** dans la zone de texte.
5. Choisissez Supprimer la table.

## Gestion des erreurs Lambda avec une API API Gateway

API Gateway traite toutes les erreurs d'invocation et de fonction comme des erreurs internes. Si l'API Lambda rejette la demande d'invocation, API Gateway renvoie un code d'erreur 500. Si la fonction s'exécute mais renvoie une erreur ou une réponse dans un format inadéquat, API Gateway renvoie un code d'erreur 502. Dans les deux cas, le corps de la réponse d'API Gateway est `{"message": "Internal server error"}`.

> **Note**
>
> API Gateway ne fait aucune nouvelle tentative d'appel Lambda. Si Lambda renvoie une erreur, API Gateway renvoie une réponse d'erreur au client.

L'exemple suivant montre une cartographie de suivi X-Ray pour une demande qui a entraîné une erreur de fonction et un message d'erreur 502 d'API Gateway. Le client reçoit le message d'erreur générique.



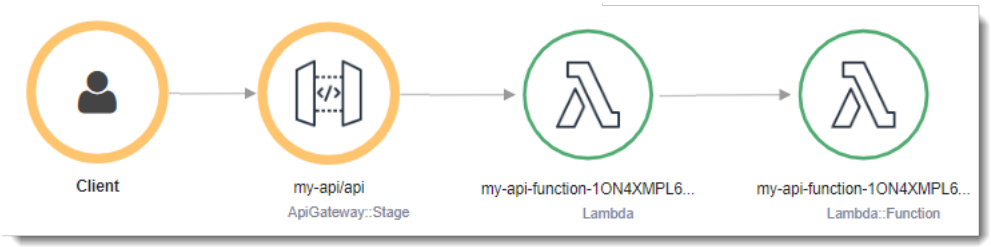

Pour personnaliser la réponse d'erreur, vous devez attraper les erreurs dans votre code et formater une réponse dans le format requis.

Example index.js : formatage des erreurs

```
var formatError = function(error){
```

```
  var response = {
    "statusCode": error.statusCode,
    "headers": {
      "Content-Type": "text/plain",
      "x-amzn-ErrorType": error.code
    },
    "isBase64Encoded": false,
    "body": error.code + ": " + error.message
  }
  return response
}
```

API Gateway convertit cette réponse en une erreur HTTP avec un code d'état et un corps personnalisés. Dans la carte de trace, le nœud de fonction est vert car il a géré l'erreur.




# Utilisation AWS Lambda avec AWS Infrastructure Composer

AWS Infrastructure Composer est un constructeur visuel permettant de concevoir des applications modernes sur AWS. Vous concevez l'architecture de votre application en faisant glisser, en regroupant et Services AWS en connectant dans un canevas visuel. Infrastructure Composer crée des modèles d'infrastructure sous forme de code (IaC) à partir de votre conception que vous pouvez déployer à l'aide de AWS SAMou AWS CloudFormation.

## Exportation d'une fonction Lambda vers Infrastructure Composer

Vous pouvez commencer à utiliser Infrastructure Composer en créant un nouveau projet basé sur la configuration d'une fonction Lambda existante à l'aide de la console Lambda. Pour exporter la configuration et le code de votre fonction vers Infrastructure Composer afin de créer un nouveau projet, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous souhaitez utiliser comme base pour votre projet Infrastructure Composer.

3. Dans le volet de présentation des fonctions, choisissez Exporter vers Infrastructure Composer.

   Pour exporter la configuration et le code de votre fonction vers Infrastructure Composer, Lambda crée un compartiment Amazon S3 dans votre compte pour stocker temporairement ces données.

4. Dans la boîte de dialogue, choisissez Confirmer et créez un projet pour accepter le nom par défaut de ce compartiment et exporter la configuration et le code de votre fonction vers Infrastructure Composer.

5. (Facultatif) Pour choisir un autre nom pour le compartiment Amazon S3 créé par Lambda, entrez un nouveau nom et choisissez Confirmer et créer un projet. Les noms de compartiment Amazon S3 doivent être uniques et respecter les règles de dénomination de compartiment.

6. Pour enregistrer vos fichiers de projet et de fonction dans Infrastructure Composer, activez le mode de synchronisation locale.

> **Note**
>
> Si vous avez déjà utilisé la fonctionnalité Exporter vers Application Composer et créé un compartiment Amazon S3 en utilisant le nom par défaut, Lambda peut réutiliser ce compartiment s'il existe toujours. Acceptez le nom du compartiment par défaut dans la boîte de dialogue pour réutiliser le compartiment existant.

## Configuration des compartiments de transfert Amazon S3 Transfer AccelAccelAccel

Le compartiment Amazon S3 créé par Lambda pour transférer la configuration de votre fonction chiffre automatiquement les objets à l'aide de la norme de chiffrement AES 256. Lambda configure également le bucket pour utiliser la condition de propriétaire du bucket afin de garantir que vous Compte AWS êtes le seul à pouvoir ajouter des objets au bucket.

Lambda configure le compartiment pour qu'il supprime automatiquement les objets 10 jours après leur téléchargement. Toutefois, Lambda ne supprime pas automatiquement le compartiment lui-même. Pour supprimer le bucket de votre Compte AWS, suivez les instructions de la section Supprimer un bucket. Le nom du compartiment par défaut utilise le préfixe`lambdasam`, une chaîne alphanumérique à 10 chiffres, et Région AWS vous avez créé votre fonction dans :

```
lambdasam-06f22da95b-us-east-1
```

Pour éviter que des frais supplémentaires ne soient ajoutés à votre compte Compte AWS, nous vous recommandons de supprimer le compartiment Amazon S3 dès que vous avez terminé d'exporter votre fonction vers Infrastructure Composer.

La tarification standard d'Amazon S3 s'applique.

## Autorisations nécessaires

Pour utiliser la fonctionnalité d'intégration de Lambda à Infrastructure Composer, vous devez disposer de certaines autorisations pour télécharger un AWS SAM modèle et écrire la configuration de votre fonction sur Amazon S3.

Pour télécharger un AWS SAM modèle, vous devez être autorisé à effectuer les API actions suivantes :

- GetPolicy
- iam : GetPolicyVersion
- iam : GetRole
- iam : GetRolePolicy
- iam : ListAttachedRolePolicies
- iam : ListRolePolicies
- iam : ListRoles

Vous pouvez autoriser l'utilisation de toutes ces actions en ajoutant la politique `AWSLambda_ReadOnlyAccess` AWS gérée à votre rôle IAM d'utilisateur.

Pour que Lambda puisse écrire la configuration de votre fonction sur Amazon S3, vous devez être autorisé à effectuer les actions suivantes : API

- S3 : PutObject
- S3 : CreateBucket
- S3 : PutBucketEncryption
- S3 : PutBucketLifecycleConfiguration

Si vous ne parvenez pas à exporter la configuration de votre fonction vers Infrastructure Composer, vérifiez que votre compte dispose des autorisations requises pour ces opérations. Si vous avez

les autorisations requises mais que vous ne pouvez toujours pas exporter la configuration de votre fonction, vérifiez les politiques basées sur les ressources qui pourraient limiter l'accès Amazon S3.

## Autres ressources

Pour un didacticiel plus détaillé sur la conception d'une application sans serveur dans Infrastructure Composer sur la base d'une fonction Lambda existante, voir. Infrastructure en tant que code (IaC)

Pour utiliser Infrastructure Composer et concevoir et déployer une application sans serveur complète AWS SAM à l'aide de Lambda, vous pouvez également suivre AWS Infrastructure Composer le didacticiel du Serverless Patterns AWS Workshop.

# Utilisation AWS Lambda avec AWS CloudFormation

Dans un AWS CloudFormation modèle, vous pouvez spécifier une fonction Lambda comme cible d'une ressource personnalisée. Utilisez des ressources personnalisées pour traiter les paramètres, récupérer les valeurs de configuration ou appeler d'autres AWS services lors d'événements liés au cycle de vie d'une pile.

L'exemple suivant appelle une fonction qui est définie ailleurs dans le modèle.

Example – Définition de ressource personnalisée

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

Le jeton de service est le nom de ressource Amazon (ARN) de la fonction AWS CloudFormation invoquée lorsque vous créez, mettez à jour ou supprimez la pile. Vous pouvez également inclure des propriétés supplémentaires telles que`FunctionName`, qui AWS CloudFormation sont transmises telles quelles à votre fonction.

AWS CloudFormation invoque votre fonction Lambda de manière asynchrone avec un événement incluant un rappel. URL

Example — événement de AWS CloudFormation message

```
{
    "RequestType": "Create",
    "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-
processor-primer-14ROR2T3JKU66",
    "ResponseURL": "https://cloudformation-custom-resource-response-useast1.s3-us-
east-1.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-1%3A123456789012%3Astack/
lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456%7Cprimerinvoke
%7C5d478078-13e9-baf0-464a-7ef285ecc786?
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1555451971&Signature=28UijZePE5I4dvukKQqM
%2F9Rf1o4%3D",
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-
processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
```

```
    "LogicalResourceId": "primerinvoke",
    "ResourceType": "AWS::CloudFormation::CustomResource",
    "ResourceProperties": {
        "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-
processor-primer-14ROR2T3JKU66",
        "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"
    }
}
```

La fonction est chargée de renvoyer une réponse au rappel URL indiquant le succès ou l'échec. Pour obtenir la syntaxe de réponse complète, consultez Objets de réponse des ressources personnalisées.

Example — réponse AWS CloudFormation personnalisée aux ressources

```
{
    "Status": "SUCCESS",
    "PhysicalResourceId": "2019/04/18/[$LATEST]b3d1bfc65f19ec610654e4d9b9de47a0",
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-
processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
    "LogicalResourceId": "primerinvoke"
}
```

AWS CloudFormation fournit une bibliothèque appelée `cfn-response` qui gère l'envoi de la réponse. Si vous définissez votre fonction dans un modèle, vous pouvez demander le nom de la bibliothèque. AWS CloudFormation ajoute ensuite la bibliothèque au package de déploiement qu'elle crée pour la fonction.

Si la fonction utilisée par une ressource personnalisée est associée à une interface réseau élastique, ajoutez les ressources suivantes à la VPC politique dans laquelle se **`region`** trouve la région dans laquelle se trouve la fonction, sans les tirets. Par exemple, `us-east-1` est `useast1`. Cela permettra à la ressource personnalisée de répondre au rappel URL qui renvoie un signal à la AWS CloudFormation pile.

```
arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",
```

L'exemple de fonction suivant appelle une deuxième fonction. Si l'appel aboutit, la fonction envoie une réponse de confirmation à AWS CloudFormation, et la mise à jour de la pile se poursuit.

Le modèle utilise le type de ressource AWS::Serverless : :Function fourni par. AWS Serverless Application Model

Example — Fonction de ressource personnalisée

```
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs16.x
      InlineCode: |
        var aws = require('aws-sdk');
        var response = require('cfn-response');
        exports.handler = function(event, context) {
            // For Delete requests, immediately send a SUCCESS response.
            if (event.RequestType == "Delete") {
                response.send(event, context, "SUCCESS");
                return;
            }
            var responseStatus = "FAILED";
            var responseData = {};
            var functionName = event.ResourceProperties.FunctionName
            var lambda = new aws.Lambda();
            lambda.invoke({ FunctionName: functionName }, function(err, invokeResult) {
                if (err) {
                    responseData = {Error: "Invoke call failed"};
                    console.log(responseData.Error + ":\n", err);
                }
                else responseStatus = "SUCCESS";
                response.send(event, context, responseStatus, responseData);
            });
        };
      Description: Invoke a function to create a log stream.
      MemorySize: 128
      Timeout: 8
      Role: !GetAtt role.Arn
      Tracing: Active
```

Si la fonction invoquée par la ressource personnalisée n'est pas définie dans un modèle, vous pouvez obtenir le code source à `cfn-response` partir du module cfn-response dans le guide de l' AWS CloudFormation utilisateur.

Pour plus d'informations sur les ressources personnalisées, consultez Ressources personnalisées dans le Guide de l'utilisateur AWS CloudFormation .

# Traiter les événements Amazon DocumentDB avec Lambda

Vous pouvez utiliser une fonction Lambda pour traiter les événements dans un flux de modifications Amazon DocumentDB (compatible avec MongoDB) en configurant un cluster Amazon DocumentDB comme source d'événements. Ensuite, vous pouvez automatiser les charges de travail orientées événements en invoquant votre fonction Lambda chaque fois que les données changent avec votre cluster Amazon DocumentDB.

> **Note**
>
> Lambda prend en charge uniquement les versions 4.0 et 5.0 d'Amazon DocumentDB. Lambda ne prend pas en charge la version 3.6.
> De plus, pour les mappages des sources d'événements, Lambda ne prend en charge que les clusters basés sur des instances et les clusters régionaux. Lambda ne prend pas en charge les clusters élastiques ni les clusters globaux. Cette limitation ne s'applique pas lorsque vous utilisez Lambda en tant que client pour vous connecter à Amazon DocumentDB. Lambda peut se connecter à tous les types de clusters pour effectuer des opérationsCRUD.

Lambda traite les événements des flux de modifications Amazon DocumentDB de manière séquentielle dans l'ordre de leur arrivée. De ce fait, votre fonction ne peut gérer qu'un seul appel simultané depuis Amazon DocumentDB à la fois. Pour surveiller votre fonction, vous pouvez suivre ses métriques de simultanéité.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Rubriques


- Exemple d'événement Amazon DocumentDB
- Conditions préalables et autorisations
- Configuration réseau

- Création d'un mappage des sources d'événements Amazon DocumentDB (console)
- Création d'un mappage de source d'événements Amazon DocumentDB (ou) SDK CLI
- Positions de départ des interrogations et des flux
- Surveillance de votre source d'événements Amazon DocumentDB
- Tutoriel : Utilisation AWS Lambda avec Amazon DocumentDB Streams

## Exemple d'événement Amazon DocumentDB


```
{
    "eventSourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:canaryclusterb2a659a2-qo5tcmqkcl03",
    "events": [
        {
            "event": {
                "_id": {
                    "_data": "0163eeb6e7000000090100000009000041e1"
                },
                "clusterTime": {
                    "$timestamp": {
                        "t": 1676588775,
                        "i": 9
                    }
                },
                "documentKey": {
                    "_id": {
                        "$oid": "63eeb6e7d418cd98afb1c1d7"
                    }
                },
                "fullDocument": {
                    "_id": {
                        "$oid": "63eeb6e7d418cd98afb1c1d7"
                    },
                    "anyField": "sampleValue"
                },
                "ns": {
                    "db": "test_database",
                    "coll": "test_collection"
                },
                "operationType": "insert"
            }
        }
```

```
    ],
    "eventSource": "aws:docdb"
}
```

Pour plus d'informations sur les événements de cet exemple et leurs formes, consultez la page Événements de modification sur le site Web de la documentation MongoDB.

## Conditions préalables et autorisations

Avant de pouvoir utiliser Amazon DocumentDB comme source d'événements pour votre fonction Lambda, veuillez prendre note des conditions préalables suivantes. Vous devez :

- Disposez d'un cluster Amazon DocumentDB existant dans le même Compte AWS cluster Région AWS que votre fonction. Si vous n'avez pas de cluster existant, vous pouvez en créer un en suivant les étapes de la section Prise en main d'Amazon DocumentDB dans le Guide du développeur Amazon DocumentDB. Vous pouvez également suivre la première série d'étapes pour créer un cluster Amazon DocumentDB avec tous les prérequis nécessaires. Tutoriel : Utilisation AWS Lambda avec Amazon DocumentDB Streams
- Autorisez Lambda à accéder aux ressources Amazon Virtual Private Cloud VPC (Amazon) associées à votre cluster Amazon DocumentDB. Pour de plus amples informations, veuillez consulter Configuration réseau.
- Activez TLS sur votre cluster Amazon DocumentDB. Il s'agit du paramètre par défaut. Si vous le désactivezTLS, Lambda ne pourra pas communiquer avec votre cluster.
- Activez les flux de modifications sur votre cluster Amazon DocumentDB. Pour plus d'informations, veuillez consulter la rubrique Utilisation des flux de modifications avec Amazon DocumentDB dans le Guide du développeur Amazon DocumentDB.
- Fournissez à Lambda les informations d'identification pour accéder à votre cluster Amazon DocumentDB. Lors de la configuration de la source d'événement, fournissez la clé AWS Secrets Manager qui contient les informations d'authentification (nom d'utilisateur et mot de passe) requises pour accéder à votre cluster. Pour fournir cette clé lors de la configuration, procédez de l'une des manières suivantes :
  - Si vous utilisez la console Lambda pour la configuration, saisissez cette clé dans le champ Clé du gestionnaire de secrets.
  - Si vous utilisez le AWS Command Line Interface (AWS CLI) pour la configuration, fournissez cette clé dans l'`source-access-configuration`option. Vous pouvez inclure cette option avec la commande `create-event-source-mapping` ou la commande `update-event-source-mapping`. Par exemple :

```
aws lambda create-event-source-mapping \
    ...
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
west-2:123456789012:secret:DocDBSecret-AbC4E6"}]' \
    ...
```

- Accordez des autorisations à Lambda pour gérer les ressources liées à votre flux Amazon DocumentDB. Ajoutez manuellement les autorisations suivantes au rôle d'exécution de votre fonction :
  - RDS : D escribeDBClusters
  - Paramètres RDS:D escribeDBCluster
  - RDS:D Groupes escribeDBSubnet
  - EC2 : CreateNetworkInterface
  - EC2 : DescribeNetworkInterfaces
  - EC2 : DescribeVpcs
  - EC2 : DeleteNetworkInterface
  - EC2 : DescribeSubnets
  - EC2 : DescribeSecurityGroups
  - kms:Decrypt
  - responsable des secrets : GetSecretValue
- La taille des événements de flux de modifications Amazon DocumentDB que vous envoyez à Lambda doit être inférieure à 6 Mo. Lambda prend en charge des charges utiles d'une taille maximale de 6 Mo. Si votre flux de modifications essaie d'envoyer à Lambda un événement supérieur à 6 Mo, Lambda supprime le message et émet la métrique `OversizedRecordCount`. Lambda émet toutes les métriques dans la mesure du possible.

> **Note**
>
> Alors que les fonctions Lambda ont généralement un délai d'expiration maximal de 15 minutes, les mappages de sources d'événements pour Amazon, Apache Kafka autogéré, MSK Amazon DocumentDB et Amazon MQ pour ActiveMQ et RabbitMQ ne prennent en charge que les fonctions dont le délai d'expiration maximal est de 14 minutes. Cette

contrainte garantit que le mappage des sources d'événements peut gérer correctement les erreurs de fonction et effectuer de nouvelles tentatives.

## Configuration réseau

Pour que Lambda puisse utiliser votre cluster Amazon DocumentDB comme source d'événements, il doit avoir accès à l'VPCAmazon dans lequel réside votre cluster. Nous vous recommandons de déployer des AWS PrivateLink VPCpoints de terminaison pour que Lambda puisse accéder à votre. VPC Déployez un VPC point de terminaison pour Lambda et, si le cluster utilise l'authentification, déployez également un VPC point de terminaison pour Secrets Manager.

Vous pouvez également vous assurer que le cluster VPC associé à votre cluster Amazon DocumentDB inclut une NAT passerelle par sous-réseau public. Pour de plus amples informations, veuillez consulter the section called "Accès à Internet pour les VPC fonctions".

Si vous utilisez des VPC points de terminaison, vous devez également les configurer pour activer les DNS noms privés.

Lorsque vous créez un mappage de source d'événements pour un cluster Amazon DocumentDB, Lambda vérifie si des interfaces réseau élastiques (ENIs) sont déjà présentes pour les sous-réseaux et les groupes de sécurité de votre cluster. VPC Si Lambda trouve des objets existantsENIs, il essaie de les réutiliser. Sinon, Lambda en crée un nouveau ENIs pour se connecter à la source de l'événement et appeler votre fonction.

> **Note**
>
> Les fonctions Lambda s'exécutent toujours au sein VPCs du service Lambda. Ils VPCs sont gérés automatiquement par le service et ne sont pas visibles pour les clients. Vous pouvez également connecter votre fonction à un AmazonVPC. Dans les deux cas, la VPC configuration de votre fonction n'affecte pas le mappage des sources d'événements. Seule la configuration des sources d'événements VPC détermine la manière dont Lambda se connecte à votre source d'événements.

### VPCrègles du groupe de sécurité

Configurez les groupes de sécurité pour l'Amazon VPC contenant votre cluster avec les règles suivantes (au minimum) :

- Règles de trafic entrant : autorisez tout le trafic sur le port du cluster Amazon DocumentDB pour les groupes de sécurité spécifiés pour votre source d'événement. Amazon DocumentDB utilise le port 27017 par défaut.
- Règles sortantes : autorisent tout le trafic sur le port 443 pour toutes les destinations. Autorisez tout le trafic sur le port du cluster Amazon DocumentDB. Amazon DocumentDB utilise le port 27017 par défaut.
- Si vous utilisez des VPC points de terminaison au lieu d'une NAT passerelle, les groupes de sécurité associés aux VPC points de terminaison doivent autoriser tout le trafic entrant sur le port 443 en provenance des groupes de sécurité de la source de l'événement.

## Utilisation de points de terminaison VPC

Lorsque vous utilisez des VPC points de terminaison, API les appels pour appeler votre fonction sont acheminés via ces points de terminaison à l'aide du. ENIs Le principal du service Lambda doit faire appel à toutes `lambda:InvokeFunction` les fonctions qui les utilisent. ENIs

Par défaut, les VPC points de terminaison ont IAM des politiques ouvertes. La meilleure pratique consiste à restreindre ces politiques afin de n'autoriser que des principaux spécifiques à effectuer les actions nécessaires à l'aide de ce point de terminaison. Pour garantir que le mappage de votre source d'événements est en mesure d'appeler votre fonction Lambda, la politique du VPC point de terminaison doit autoriser le principe du service Lambda à appeler. `lambda:InvokeFunction` Le fait de restreindre les politiques de vos VPC terminaux pour n'autoriser que les API appels provenant de votre organisation empêche le mappage des sources d'événements de fonctionner correctement.

Les exemples de politiques de point de VPC terminaison suivants montrent comment accorder l'accès requis aux points de terminaison Lambda.

Example VPCpolitique de point de terminaison - point de terminaison Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
```

```
            },
            "Resource": "*"
        }
    ]
}
```

Si votre cluster Amazon DocumentDB utilise l'authentification, vous pouvez également restreindre la politique de point de VPC terminaison pour le point de terminaison Secrets Manager. Pour appeler le Secrets ManagerAPI, Lambda utilise votre rôle fonctionnel, et non le principal du service Lambda. L'exemple suivant montre une politique de point de terminaison de Secrets Manager.

Example VPCpolitique relative aux points de terminaison - Point de terminaison Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

## Création d'un mappage des sources d'événements Amazon DocumentDB (console)

Pour qu'une fonction Lambda puisse lire le flux de modifications d'un cluster Amazon DocumentDB, créez un mappage des sources d'événements. Cette section explique comment procéder à partir de la console Lambda. Pour les AWS CLI instructions AWS SDK et les instructions, voirthe section called “Création d'un mappage de source d'événements Amazon DocumentDB (ou) SDK CLI”.

Pour créer un mappage des sources d'événements Amazon DocumentDB (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d’une fonction.

3. Sous Function overview (Présentation de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Sous Configuration du déclencheur, dans la liste déroulante, choisissez DocumentDB.
5. Configurez les options requises, puis choisissez Add (Ajouter).

Lambda prend en charge les options suivantes pour les sources d'événement Amazon DocumentDB :

- Cluster DocumentDB : sélectionnez un cluster Amazon DocumentDB.
- Activer le déclencheur : choisissez si vous voulez activer le déclencheur maintenant. Si vous cochez cette case, votre fonction commence immédiatement à recevoir du trafic provenant du flux de modifications Amazon DocumentDB spécifié lors de la création du mappage des sources d'événements. Nous vous recommandons de décocher la case pour créer le mappage des sources d'événements dans un état désactivé à des fins de test. Après la création, vous pouvez activer le mappage des sources d'événements à tout moment.
- Nom de la base de données – Saisissez le nom de la base de données du cluster à utiliser.
- (Facultatif) Nom de la collection : saisissez le nom d'une collection de la base de données à utiliser. Si vous n'indiquez pas de collection, Lambda écoute tous les événements de chaque collection de la base de données.
- Taille de lot – Définissez le nombre maximum de messages à extraire dans un lot, jusqu'à 10 000. La taille du lot par défaut est de 100.
- Position de départ – Choisissez la position dans le flux à partir de laquelle commencer la lecture des enregistrements.
  - Derniers – Traiter uniquement les nouveaux enregistrements qui sont ajoutés au flux. Votre fonction ne commence à traiter les enregistrements que lorsque Lambda a fini de créer votre source d'événements. Cela signifie que certains enregistrements peuvent être supprimés jusqu'à ce que la source de votre événement soit correctement créée.
  - Trim horizon (Supprimer l’horizon) – Traiter tous les enregistrements figurant dans le flux. Lambda utilise la durée de conservation des journaux de votre cluster pour déterminer par où commencer la lecture des événements. Plus précisément, Lambda commence à lire à partir de `current_time - log_retention_duration`. Votre flux de modifications doit déjà être actif avant cet horodatage pour que Lambda puisse lire correctement tous les événements.

  - At timestamp (À l'horodatage) – Traitez les enregistrements à partir d'une heure spécifique. Votre flux de modifications doit déjà être actif avant l'horodatage spécifié pour que Lambda puisse lire correctement tous les événements.
- Authentication – Choisissez la méthode d'authentification pour accéder aux agents de votre cluster.
  - BASIC_ AUTH — Avec l'authentification de base, vous devez fournir la clé Secrets Manager qui contient les informations d'identification pour accéder à votre cluster.
- Clé Secrets Manager : choisissez la clé Secrets Manager qui contient les informations d'authentification (nom d'utilisateur et mot de passe) requises pour accéder à votre cluster Amazon DocumentDB.
- (Facultatif) Fenêtre de traitement par lot : définissez l'intervalle de temps maximum (en secondes) pour collecter des enregistrements avant d'invoquer votre fonction, jusqu'à 300.
- (Facultatif) Configuration complète du document : pour les opérations de mise à jour des documents, choisissez ce que vous voulez envoyer au flux. La valeur par défaut est `Default`, ce qui signifie que pour chaque événement de flux de modifications, Amazon DocumentDB envoie uniquement un delta décrivant les modifications apportées. Pour plus d'informations sur ce champ, consultez la FullDocumentdocumentation API Javadoc de MongoDB.
  - Par défaut – Lambda n'envoie qu'un document partiel décrivant les modifications apportées.
  - UpdateLookup— Lambda envoie un delta décrivant les modifications, ainsi qu'une copie de l'intégralité du document.

## Création d'un mappage de source d'événements Amazon DocumentDB (ou) SDK CLI

Pour créer ou gérer un mappage de source d'événements Amazon DocumentDB avec un AWS SDK, vous pouvez utiliser les opérations suivantes : API

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Pour créer le mappage des sources d'événements avec le AWS CLI, utilisez la `create-event-source-mapping`commande. L'exemple suivant utilise cette commande pour mapper une fonction

nommée `my-function` à un flux de modifications Amazon DocumentDB. La source de l'événement est spécifiée par un Amazon Resource Name (ARN), avec une taille de lot de 500, à partir de l'horodatage en heure Unix. La commande spécifie également la clé Secrets Manager que Lambda utilise pour se connecter à Amazon DocumentDB. De plus, elle inclut des paramètres `document-db-event-source-config` qui spécifient la base de données et la collection à partir de laquelle lire.

```
aws lambda create-event-source-mapping --function-name my-function \
    --event-source-arn arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy
    --batch-size 500 \
    --starting-position AT_TIMESTAMP \
    --starting-position-timestamp 1541139109 \
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
east-1:123456789012:secret:DocDBSecret-BAtjxi"}]' \
    --document-db-event-source-config '{"DatabaseName":"test_database",
 "CollectionName": "test_collection"}' \
```

Vous devriez obtenir un résultat du type suivant :

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "DocumentDBEventSourceConfig": {
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "MaximumBatchingWindowInSeconds": 0,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541348195.412,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "User action"
}
```

Après la création, vous pouvez utiliser la commande `update-event-source-mapping` pour mettre à jour les paramètres de votre source d'événements Amazon DocumentDB. L'exemple suivant met à

jour la taille du lot à 1 000 et la fenêtre de traitement par lots à 10 secondes. Pour cette commande, vous avez besoin du mappage UUID de la source de votre événement, que vous pouvez récupérer à l'aide de la `list-event-source-mapping` commande ou de la console Lambda.

```
aws lambda update-event-source-mapping --function-name my-function \
    --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
    --batch-size 1000 \
    --batch-window 10
```

Vous devriez obtenir un résultat du type suivant :

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "DocumentDBEventSourceConfig": {
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "MaximumBatchingWindowInSeconds": 0,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541359182.919,
    "LastProcessingResult": "OK",
    "State": "Updating",
    "StateTransitionReason": "User action"
}
```

Lambda met à jour les paramètres de façon asynchrone, il se peut donc que vous ne voyiez pas ces modifications dans la sortie tant que le processus n'est pas terminé. Pour afficher les paramètres actuels de votre mappage des sources d'événements, utilisez la commande `get-event-source-mapping`.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Vous devriez obtenir un résultat du type suivant :

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "DocumentDBEventSourceConfig": {
```

```
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "BatchSize": 1000,
    "MaximumBatchingWindowInSeconds": 10,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541359182.919,
    "LastProcessingResult": "OK",
    "State": "Enabled",
    "StateTransitionReason": "User action"
}
```

Pour supprimer le mappage des sources d'événements Amazon DocumentDB, utilisez la commande `delete-event-source-mapping`.

```
aws lambda delete-event-source-mapping \
    --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

## Positions de départ des interrogations et des flux

Sachez que l'interrogation des flux lors des mises à jour et de la création du mappage des sources d'événements est finalement cohérente.

- Lors de la création du mappage des sources d'événements, le démarrage de l'interrogation des événements depuis le flux peut prendre plusieurs minutes.
- Lors des mises à jour du mappage des sources d'événements, l'arrêt et le redémarrage de l'interrogation des événements depuis le flux peuvent prendre plusieurs minutes.

Ce comportement signifie que si vous spécifiez `LATEST` comme position de départ du flux, le mappage des sources d'événements peut manquer des événements lors de la création ou des mises à jour. Pour vous assurer de ne manquer aucun événement, spécifiez la position de départ du flux comme `TRIM_HORIZON` ou `AT_TIMESTAMP`.

## Surveillance de votre source d'événements Amazon DocumentDB

Pour vous aider à surveiller votre source d'événements Amazon DocumentDB, Lambda émet la métrique `IteratorAge` lorsque votre fonction termine le traitement d'un lot d'enregistrements. L'âge

de l'itérateur est la différence entre l'horodatage de l'événement le plus récent et l'horodatage actuel. La métrique `IteratorAge` indique essentiellement l'ancienneté du dernier enregistrement traité dans le lot. Si votre fonction traite actuellement de nouveaux événements, vous pouvez utiliser l'âge de l'itérateur pour estimer la latence entre le moment où un enregistrement est ajouté et celui où votre fonction le traite. Une tendance à la hausse de `IteratorAge` peut indiquer des problèmes liés à votre fonction. Pour de plus amples informations, veuillez consulter Afficher les métriques des fonctions Lambda.

Les flux de modifications d'Amazon DocumentDB ne sont pas optimisés pour gérer les intervalles de temps importants entre les événements. Si votre source d'événements Amazon DocumentDB ne reçoit aucun événement pendant une période prolongée, Lambda peut désactiver le mappage de la source d'événements. La durée de cette période peut varier de quelques semaines à quelques mois en fonction de la taille du cluster et des autres charges de travail.

Lambda prend en charge des charges utiles allant jusqu'à 6 Mo. Cependant, les événements du flux de modification d'Amazon DocumentDB peuvent avoir une taille allant jusqu'à 16 Mo. Si votre flux de modifications tente d'envoyer à Lambda un événement d'une taille supérieure à 6 Mo, Lambda supprime le message et émet la métrique `OversizedRecordCount`. Lambda émet toutes les métriques dans la mesure du possible.

## Tutoriel : Utilisation AWS Lambda avec Amazon DocumentDB Streams

Dans ce tutoriel, vous créez une fonction Lambda de base qui consomme des événements à partir d'un flux de modifications Amazon DocumentDB (compatible avec MongoDB). Pour réaliser ce tutoriel, vous passerez par les étapes suivantes :

- Configurez votre cluster Amazon DocumentDB, connectez-vous-y, et activez les flux de modifications sur ce cluster.
- Créez votre fonction Lambda, et configurez votre cluster Amazon DocumentDB en tant que source d'événements pour votre fonction.
- Testez la end-to-end configuration en insérant des éléments dans votre base de données Amazon DocumentDB.

Rubriques


- Prérequis
- Créez l' AWS Cloud9 environnement
- Création du groupe EC2 de sécurité Amazon

- Création du cluster Amazon DocumentDB
- Création d'un secret dans Secrets Manager
- Installez le shell Mongo
- Connectez-vous au cluster Amazon DocumentDB
- Activation des flux de modifications
- Création de points de VPC terminaison d'interface
- Créer le rôle d’exécution
- Créer la fonction Lambda
- Création du mappage des sources d'événements Lambda
- Test de votre fonction : invocation manuelle
- Test de votre fonction : insertion d'un enregistrement
- Test de votre fonction : mise à jour d'un enregistrement
- Test de votre fonction : suppression d'un enregistrement
- Nettoyage de vos ressources

## Prérequis

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l‘utilisateur racine a accès à l‘ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l‘utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu'utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d’un ensemble d’autorisations dans le Guide de l’utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l’utilisateur AWS IAM Identity Center .

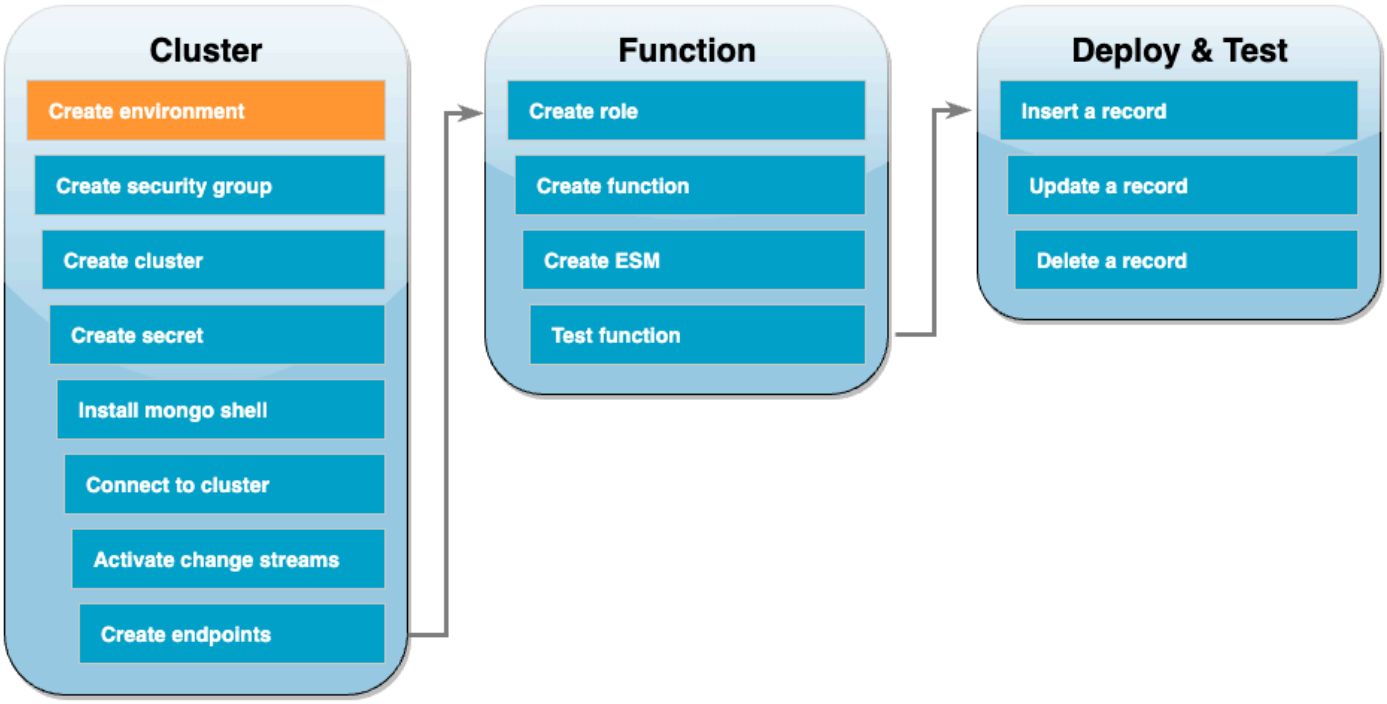
Installez le AWS Command Line Interface

Si vous ne l'avez pas encore installé AWS Command Line Interface, suivez les étapes décrites dans la section Installation ou mise à jour de la dernière version du AWS CLI pour l'installer.

Ce tutoriel nécessite un terminal de ligne de commande ou un shell pour exécuter les commandes. Sous Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d’obtenir une version intégrée à Windows d’Ubuntu et Bash.

## Créez l' AWS Cloud9 environnement




Avant de créer la fonction Lambda, vous devez créer et configurer votre cluster Amazon DocumentDB. Les étapes de configuration de votre cluster dans ce tutoriel sont basées sur la procédure de la section Démarrer avec Amazon DocumentDB (français non garanti).

> **Note**
>
> Si vous avez déjà configuré un cluster Amazon DocumentDB, assurez-vous d'activer les flux de modifications et de créer les points de terminaison d'interface VPC nécessaires. Ensuite, vous pouvez passer directement aux étapes de création de la fonction.

Tout d'abord, créez un AWS Cloud9 environnement. Vous utiliserez cet environnement tout au long de ce didacticiel pour vous connecter à votre cluster Amazon DocumentDB et l'interroger.

Pour créer un AWS Cloud9 environnement

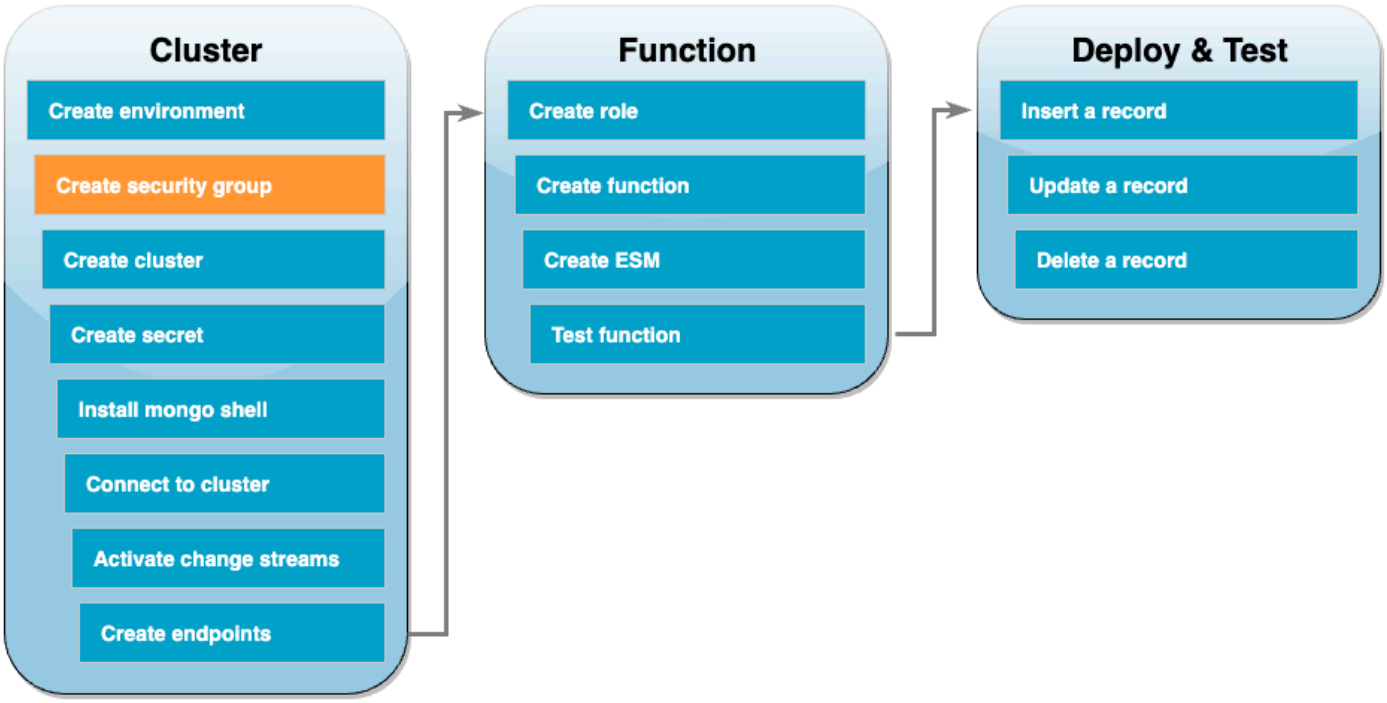
1. Ouvrez la AWS Cloud9 console et choisissez Create environment.
2. Créez un environnement avec la configuration suivante :
   - Sous Détails :

- Nom: `DocumentDBCloud9Environment`
- Type d'environnement : Nouvelle EC2 instance

- Sous Nouvelle EC2 instance :
  - Type d'instance : `t2.micro` (1 GiB RAM + 1 v) CPU
  - Plateforme : Amazon Linux 2
  - Délai d'attente : 30 minutes
- Sous Paramètres réseau :
  - Connexion : AWS Systems Manager (SSM)
  - Développez le menu déroulant des VPCparamètres.
  - Amazon Virtual Private Cloud (VPC) : Choisissez votre cloud par défaut. VPC
  - Sous-réseau : aucune préférence
- Conservez tous les autres paramètres par défaut.

3. Sélectionnez Create (Créer). Le provisionnement de votre nouvel AWS Cloud9 environnement peut prendre plusieurs minutes.

## Création du groupe EC2 de sécurité Amazon

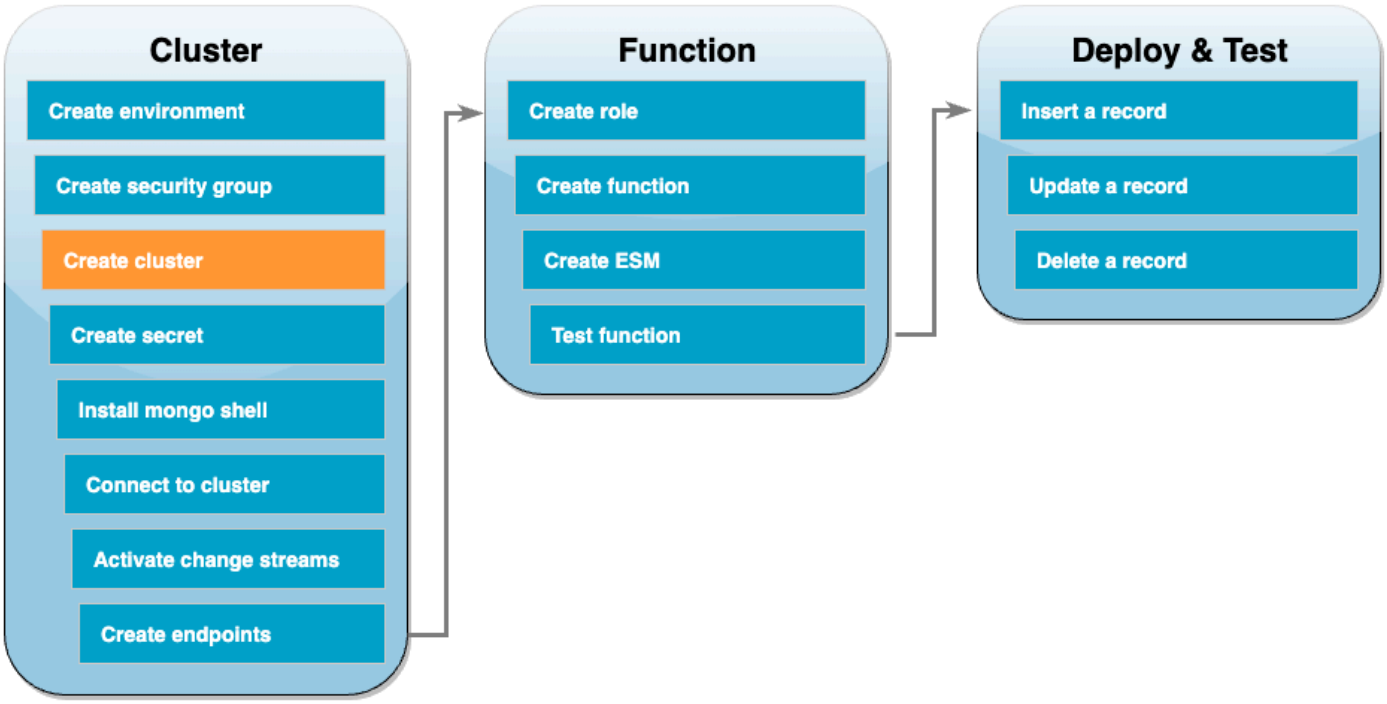


Créez ensuite un groupe de EC2 sécurité Amazon avec des règles qui autorisent le trafic entre votre cluster Amazon DocumentDB et votre AWS Cloud9 environnement.

Pour créer un groupe EC2 de sécurité

1. Ouvrez la EC2console. Sous Réseau et sécurité, choisissez Groupes de sécurité.
2. Sélectionnez Create security group (Créer un groupe de sécurité).
3. Créez un groupe de sécurité avec la configuration suivante :
   - Sous Informations de base :
     - Nom du groupe de sécurité : `DocDBTutorial`
     - Description : groupe de sécurité pour le trafic entre AWS Cloud9 et Amazon DocumentDB.
     - VPC: Choisissez votre valeur par défaut VPC.
   - Pour Inbound rules (Règles entrantes), choisissez Add rule (Ajouter une règle). Créez une règle avec la configuration suivante :
     - Type : Personnalisé TCP
     - Portée de ports : 27017
     - Source : Personnalisé
     - Dans le champ de recherche situé à côté de Source, choisissez le groupe de sécurité pour l' AWS Cloud9 environnement que vous avez créé à l'étape précédente. Pour afficher la liste des groupes de sécurité disponibles, saisissez `cloud9` dans le champ de recherche. Choisissez le groupe de sécurité portant le nom `aws-cloud9-<environment_name>`.
   - Conservez tous les autres paramètres par défaut.
4. Sélectionnez Create security group (Créer un groupe de sécurité).

# Création du cluster Amazon DocumentDB




Au cours de cette étape, vous allez créer un cluster Amazon DocumentDB en utilisant le groupe de sécurité de l'étape précédente.

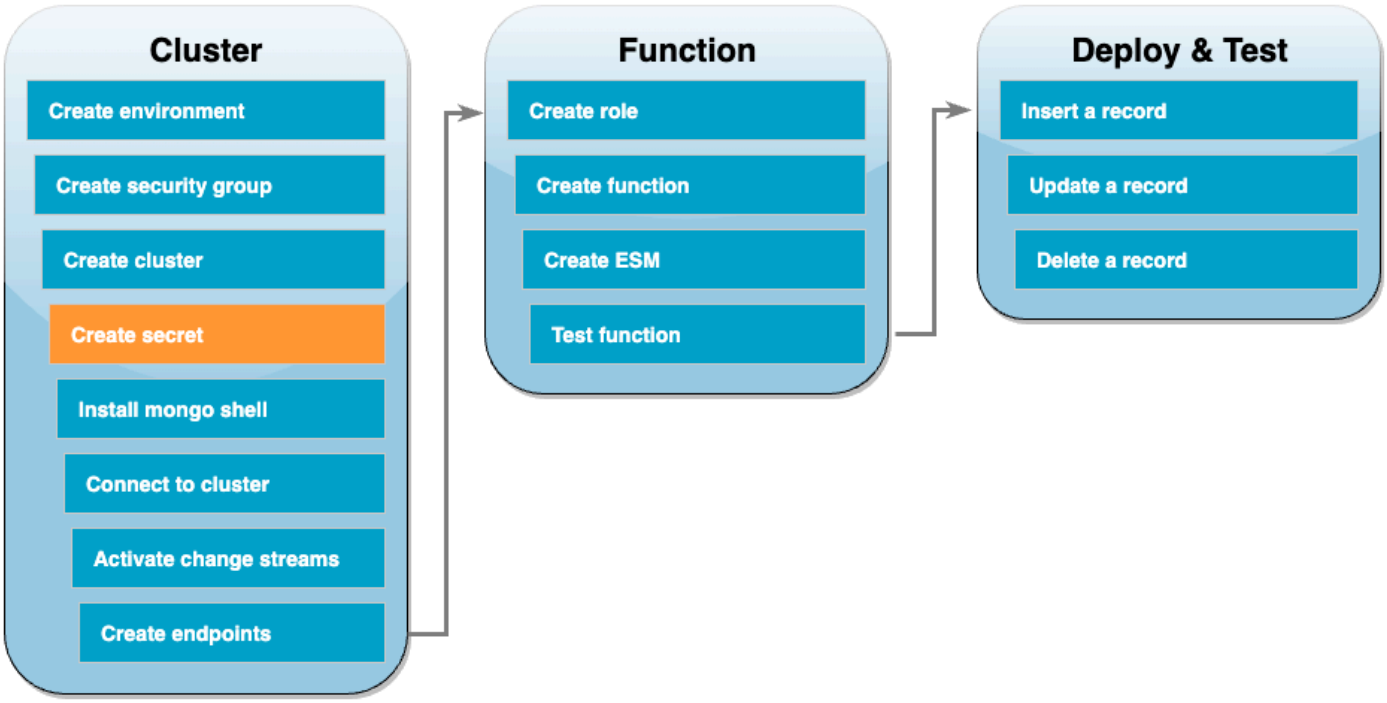
Pour créer un cluster Amazon DocumentDB

1. Ouvrez la console Amazon DocumentDB. Sous Clusters, sélectionnez Créer.
2. Créez un cluster avec la configuration suivante :
   - Pour Type de cluster, choisissez Cluster basé sur l'instance.
   - Sous Configuration :
     - Version du moteur : 5.0.0
     - Classe d'instance : db.t3.medium (essai gratuit éligible)
     - Nombre d'instances : 1
   - Sous Authentification :
     - Entrez le nom d'utilisateur et le mot de passe nécessaires pour vous connecter à votre cluster (mêmes informations d'identification que celles que vous avez utilisées pour créer le secret à l'étape précédente). Dans Confirmer le mot de passe, confirmez votre mot de passe.

- Activez Afficher les paramètres avancés.
- Sous Paramètres réseau :
  - Virtual Private Cloud (VPC) : Choisissez votre cloud par défaut VPC.
  - Groupe de sous-réseaux : par défaut
  - VPCgroupes de sécurité : choisissez également le groupe de `DocDBTutorial (VPC)` sécurité que vous avez créé à l'étape précédente. `default (VPC)`
- Conservez tous les autres paramètres par défaut.

3. Choisissez Créer un cluster. Le provisionnement de votre cluster Amazon DocumentDB peut prendre plusieurs minutes.

## Création d'un secret dans Secrets Manager



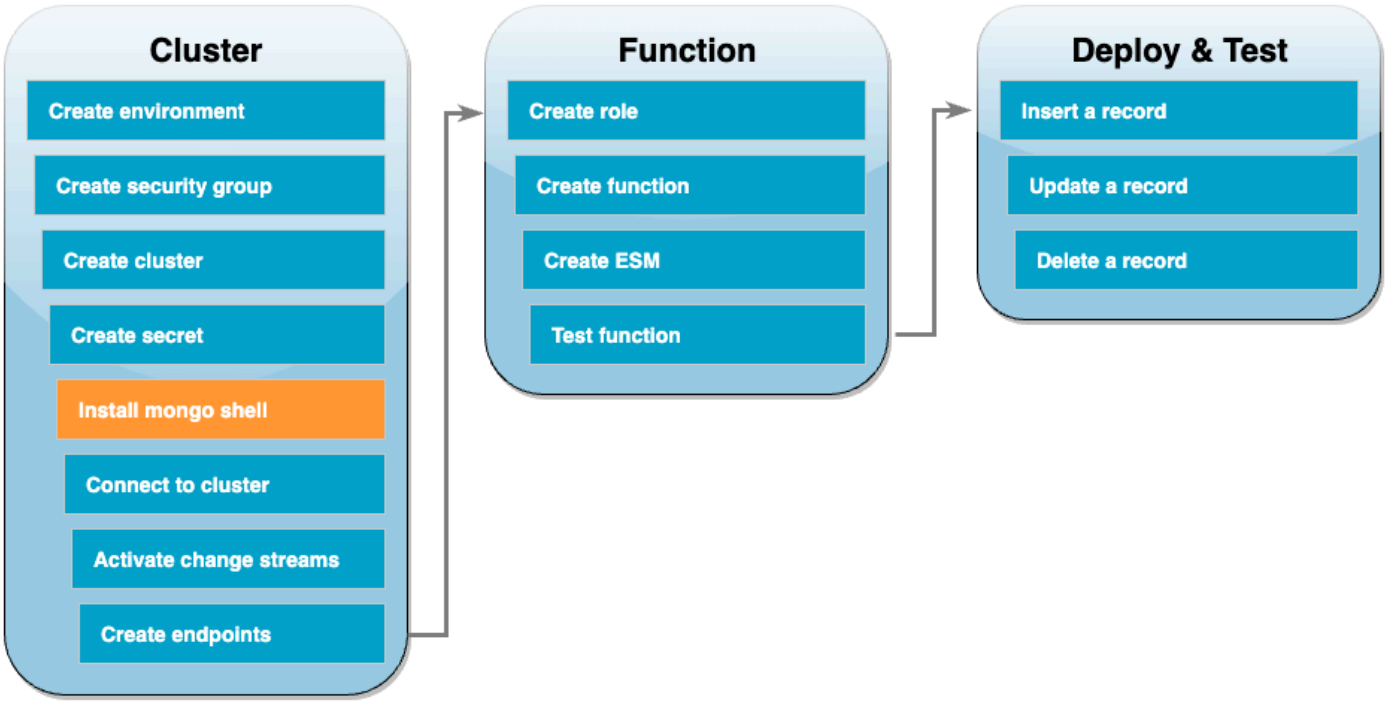

Pour accéder manuellement à votre cluster Amazon DocumentDB, vous devez fournir un nom d'utilisateur et un mot de passe. Pour que Lambda puisse accéder à votre cluster, vous devez fournir un secret Secrets Manager qui contient ces mêmes informations d'identification d'accès lors de la configuration de votre mappage des sources d'événements. Dans cette étape, vous allez créer ce secret.

Pour créer le secret dans Secrets Manager

1. Ouvrez la console Secrets Manager et choisissez Stocker un nouveau secret.
2. Pour Choisir le type de secret, sélectionnez les options suivantes :
   - Sous Informations de base :
     - Type de secret : informations d'identification pour votre base de données Amazon DocumentDB
     - Sous Informations d'identification, entrez le nom d'utilisateur et le mot de passe que vous utiliserez pour accéder à votre cluster Amazon DocumentDB.
     - Base de données : choisissez votre cluster Amazon DocumentDB.
     - Choisissez Suivant.
3. Pour Configurer le secret, choisissez les options suivantes :
   - Nom secret — `DocumentDBSecret`
   - Choisissez Suivant.
4. Choisissez Suivant.
5. Choisissez Stocker.
6. Actualisez la console pour vérifier que vous avez correctement enregistré le secret `DocumentDBSecret`.

Notez le secret ARN de votre secret. Vous en aurez besoin dans une étape ultérieure.

## Installez le shell Mongo




Au cours de cette étape, vous allez installer le shell mongo dans votre AWS Cloud9 environnement. Le shell mongo est un utilitaire de ligne de commande que vous utilisez pour vous connecter à votre cluster Amazon DocumentDB et l'interroger.

Pour installer le shell mongo sur votre AWS Cloud9 environnement

1. Ouvrez la AWS Cloud9 console. À côté de l'`DocumentDBCloud9Environment`environnement que vous avez créé précédemment, cliquez sur le lien Ouvrir situé sous la AWS Cloud9 IDEcolonne.
2. Dans la fenêtre du terminal, créez le fichier de référentiel MongoDB avec la commande suivante :

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc" | sudo tee /etc/
yum.repos.d/mongodb-org-5.0.repo
```

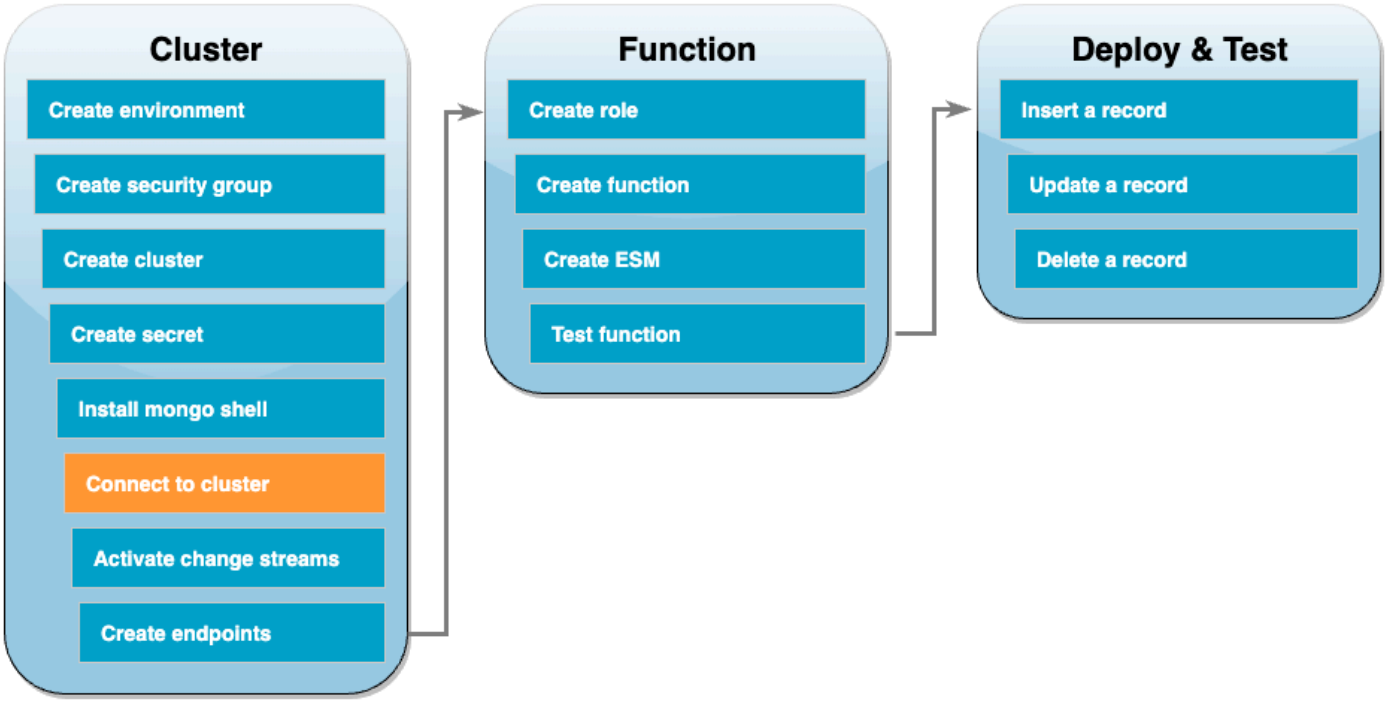
3. Installez ensuite le shell mongo avec la commande suivante :

```
sudo yum install -y mongodb-org-shell
```

4. Pour chiffrer les données en transit, téléchargez la clé publique d'amazon DocumentDB. La commande suivante télécharge un fichier nommé `global-bundle.pem` :

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

## Connectez-vous au cluster Amazon DocumentDB



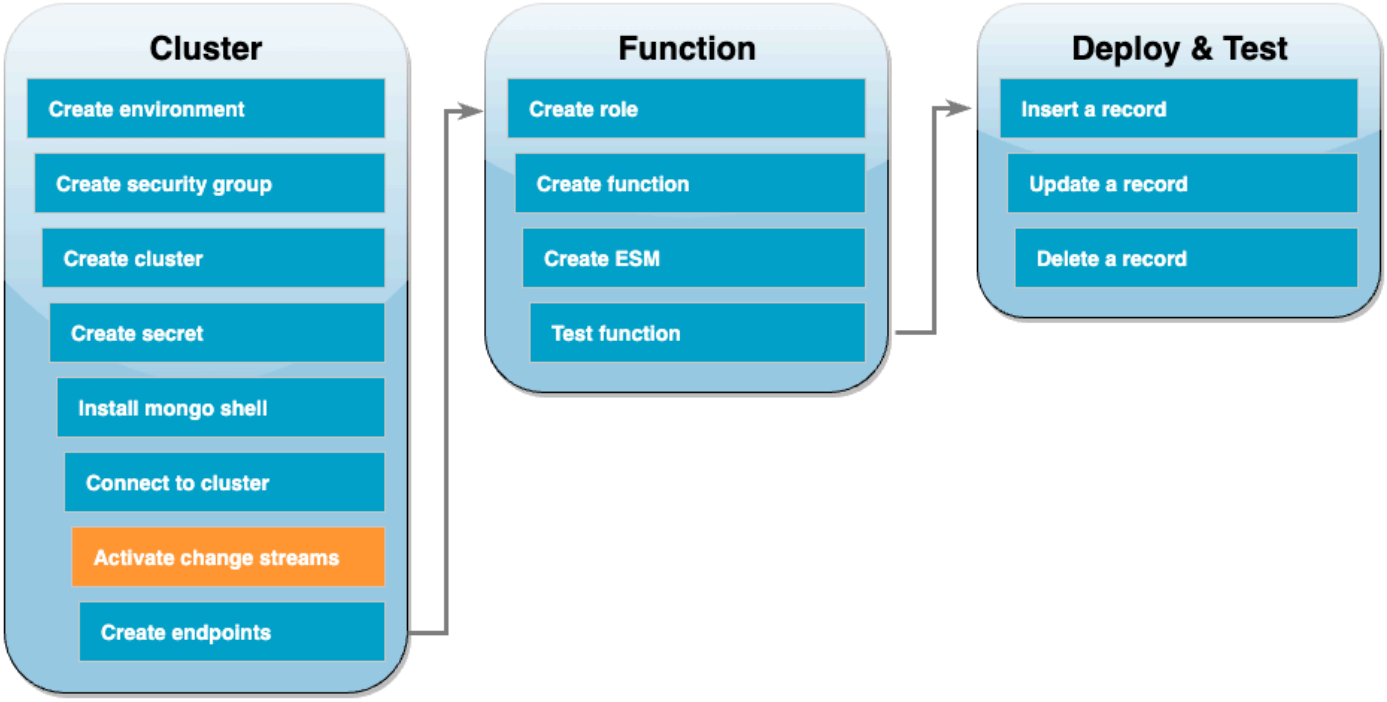

Vous êtes maintenant prêt à vous connecter à votre cluster Amazon DocumentDB à l'aide du shell mongo.

Pour vous connecter à votre cluster Amazon DocumentDB

1. Ouvrez la console Amazon DocumentDB. Sous Clusters, choisissez votre cluster en sélectionnant son identifiant.
2. Dans l'onglet Connectivité et sécurité, sous Connect to this cluster with the mongo shell, choisissez Copier.
3. Dans votre AWS Cloud9 environnement, collez cette commande dans le terminal. Remplacez `<insertYourPassword>` par le mot de passe correct.

Après avoir saisi cette commande, si l'invite de commande devient `rs0:PRIMARY>`, c'est que vous êtes connecté à votre cluster Amazon DocumentDB.

## Activation des flux de modifications




Dans le cadre de ce didacticiel, vous allez suivre les modifications apportées à la `products` collection de la `docdbdemo` base de données dans votre cluster Amazon DocumentDB. Pour ce faire, vous activez les flux de modifications. Tout d'abord, créez la base de données `docdbdemo` et testez-la en insérant un enregistrement.

Pour créer une nouvelle base de données dans votre cluster

1. Dans votre AWS Cloud9 environnement, assurez-vous que vous êtes toujours connecté à votre cluster Amazon DocumentDB.
2. Dans la fenêtre du terminal, utilisez la commande suivante pour créer une nouvelle base de données appelée `docdbdemo` :

   ```
   use docdbdemo
   ```

3. Utilisez ensuite la commande suivante pour insérer un enregistrement dans `docdbdemo` :

   ```
   db.products.insert({"hello":"world"})
   ```

Vous devriez obtenir un résultat du type suivant :

```
WriteResult({ "nInserted" : 1 })
```

4. Utilisez la commande suivante pour répertorier toutes les bases de données :

```
show dbs
```

Assurez-vous que votre sortie contient la base de données `docdbdemo` :

```
docdbdemo  0.000GB
```

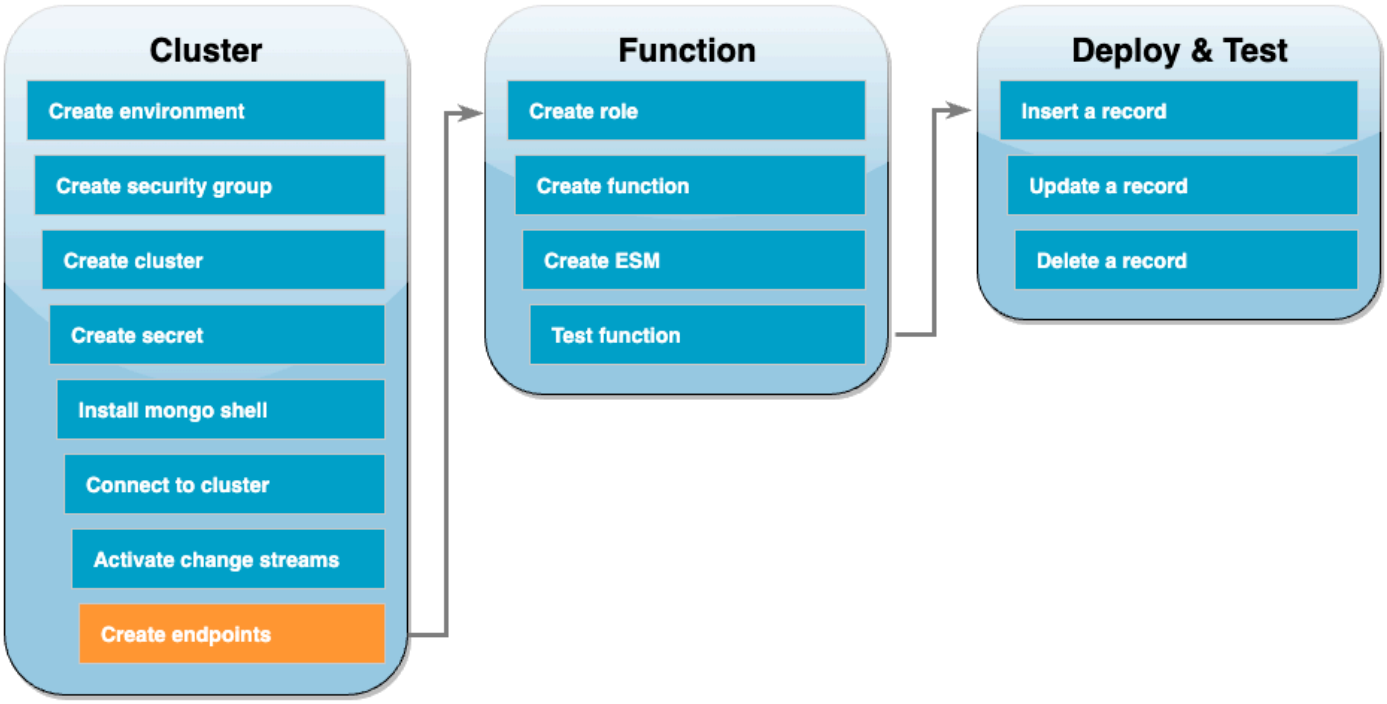
Activez ensuite les flux de modifications sur la collection `products` de la base de données `docdbdemo` à l'aide de la commande suivante :

```
db.adminCommand({modifyChangeStreams: 1,
    database: "docdbdemo",
    collection: "products",
    enable: true});
```

Vous devriez obtenir un résultat du type suivant :

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

## Création de points de VPC terminaison d'interface




Créez ensuite des VPCpoints de terminaison d'interface pour vous assurer que Lambda et Secrets Manager (utilisés ultérieurement pour stocker nos identifiants d'accès au cluster) puissent se connecter à votre serveur par défaut. VPC

Pour créer des points de VPC terminaison d'interface

1. Ouvrez la VPCconsole. Dans le menu de gauche, sous Cloud privé virtuel, choisissez Points de terminaison.
2. Choisissez Créer un point de terminaison. Créez un point de terminaison avec la configuration suivante :

   - Pour Balise de nom, saisissez `lambda-default-vpc`.
   - Pour la catégorie de service, sélectionnez AWS services.
   - Pour Services, saisissez `lambda` dans la zone de recherche. Choisissez le service au format `com.amazonaws.<region>.lambda`.
   - Pour VPC, choisissez votre valeur par défaut VPC.
   - Pour Sous-réseaux, cochez les cases à côté de chaque zone de disponibilité. Choisissez l'ID de sous-réseau correct pour chaque zone de disponibilité.

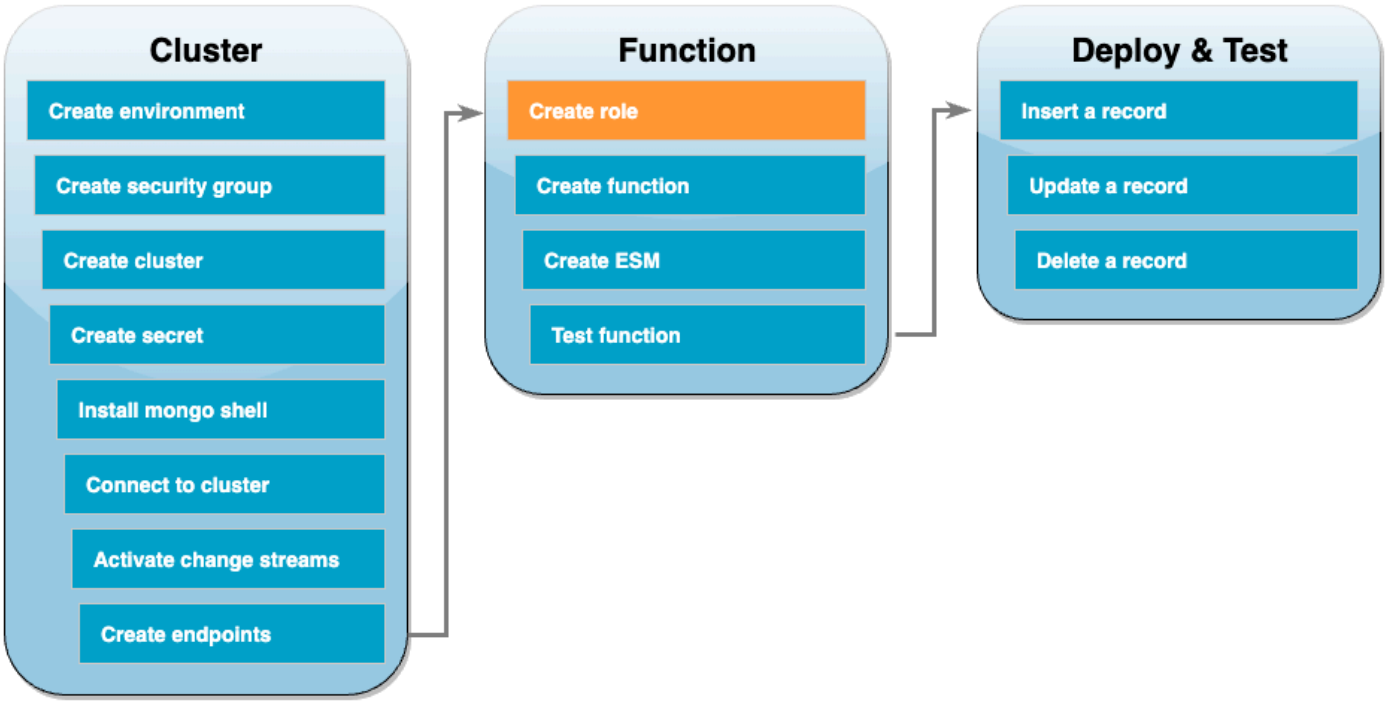
- Pour le type d'adresse IP, sélectionnezIPv4.
- Pour les groupes de sécurité, choisissez le groupe VPC de sécurité par défaut (nom du groupe de`default`) et le groupe de sécurité que vous avez créé précédemment (nom du groupe de`DocDBTutorial`).
- Conservez tous les autres paramètres par défaut.
- Choisissez Créer un point de terminaison.

3. Choisissez à nouveau Créer un point de terminaison. Créez un point de terminaison avec la configuration suivante :

   - Pour Balise de nom, saisissez `secretsmanager-default-vpc`.
   - Pour la catégorie de service, sélectionnez AWS services.
   - Pour Services, saisissez `secretsmanager` dans la zone de recherche. Choisissez le service au format `com.amazonaws.<region>.secretsmanager`.
   - Pour VPC, choisissez votre valeur par défaut VPC.
   - Pour Sous-réseaux, cochez les cases à côté de chaque zone de disponibilité. Choisissez l'ID de sous-réseau correct pour chaque zone de disponibilité.
   - Pour le type d'adresse IP, sélectionnezIPv4.
   - Pour les groupes de sécurité, choisissez le groupe VPC de sécurité par défaut (nom du groupe de`default`) et le groupe de sécurité que vous avez créé précédemment (nom du groupe de`DocDBTutorial`).
   - Conservez tous les autres paramètres par défaut.
   - Choisissez Créer un point de terminaison.

Ceci termine la partie de ce tutoriel concernant la configuration du cluster.

## Créer le rôle d'exécution




Dans les étapes suivantes, vous allez créer votre fonction Lambda. Tout d'abord, vous devez créer le rôle d'exécution qui donne à votre fonction l'autorisation d'accéder à votre cluster. Pour ce faire, vous devez d'abord créer une IAM stratégie, puis l'associer à un IAM rôle.

Pour créer une IAM politique

1. Ouvrez la page Politiques dans la IAM console et choisissez Create policy.
2. Choisissez l'JSONonglet. Dans la politique suivante, remplacez la ressource Secrets Manager ARN dans la dernière ligne de la déclaration par votre code secret ARN antérieur, et copiez la politique dans l'éditeur.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LambdaESMNetworkingAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "kms:Decrypt"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaDocDBESMAccess",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBSubnetGroups"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaDocDBESMGetSecretValueAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:DocumentDBSecret"
        }
    ]
}
```

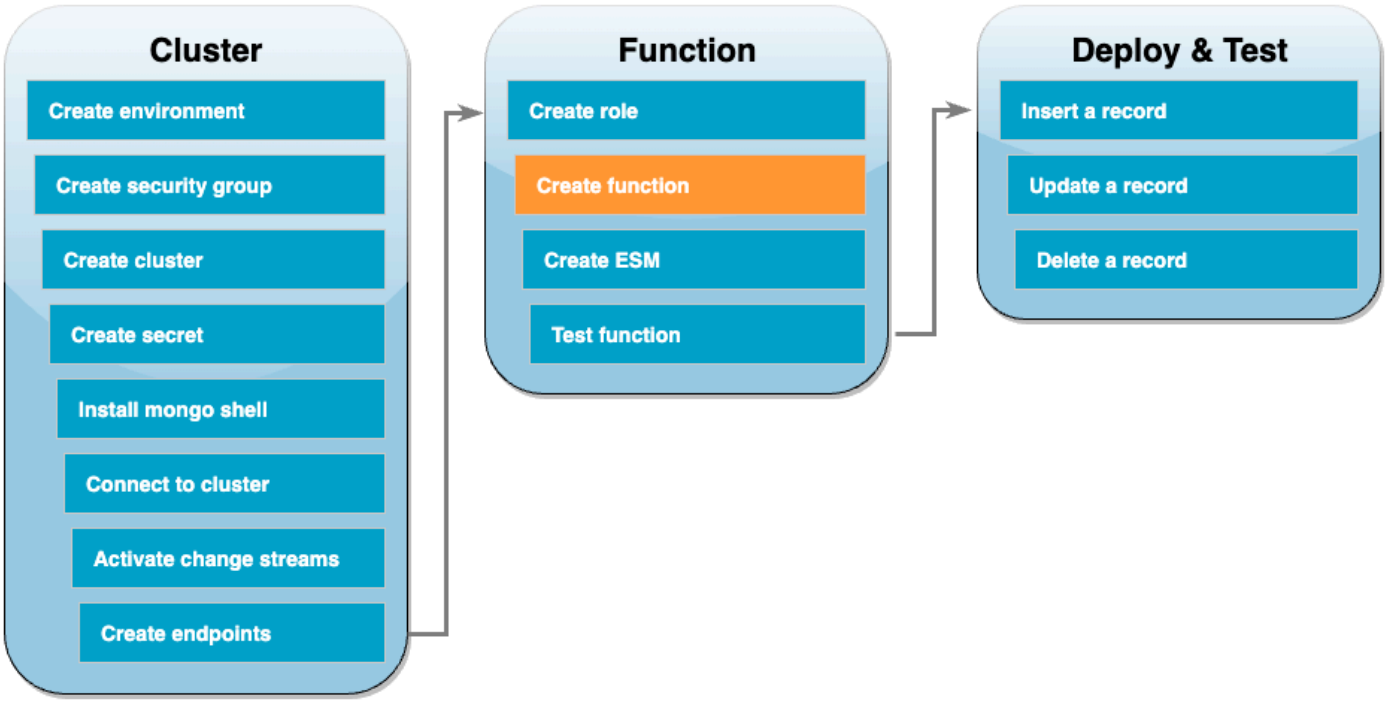
3. Choisissez Suivant : Balises, puis Suivant : Vérification.
4. Pour Name (Nom), saisissez `AWSDocumentDBLambdaPolicy`.
5. Sélectionnez Create policy (Créer une politique).

Pour créer le rôle IAM

1. Ouvrez la page Rôles dans la IAM console et choisissez Create role.
2. Pour Sélectionner une entité de confiance, choisissez les options suivantes :
   - Type d'entité de confiance : AWS service
   - Cas d'utilisation — Lambda
   - Choisissez Suivant.

3. Pour Ajouter des autorisations, choisissez la `AWSDocumentDBLambdaPolicy` politique que vous venez de créer, ainsi que celle permettant `AWSLambdaBasicExecutionRole` à votre fonction d'écrire sur Amazon CloudWatch Logs.
4. Choisissez Suivant.
5. Pour le Nom du rôle, saisissez `AWSDocumentDBLambdaExecutionRole`.
6. Sélectionnez Créer un rôle.

## Créer la fonction Lambda




L'exemple de code suivant reçoit une entrée d'événement Amazon DocumentDB et traite le message qu'elle contient.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
```

```
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';
```

```
console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. PHP

```
<?php

require __DIR__.'/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
```

```
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
 "\n";
    }
}
return new DocumentDBEventHandler();
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
```

```
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
 Error> {
```

```
    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())

}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Pour créer la fonction Lambda

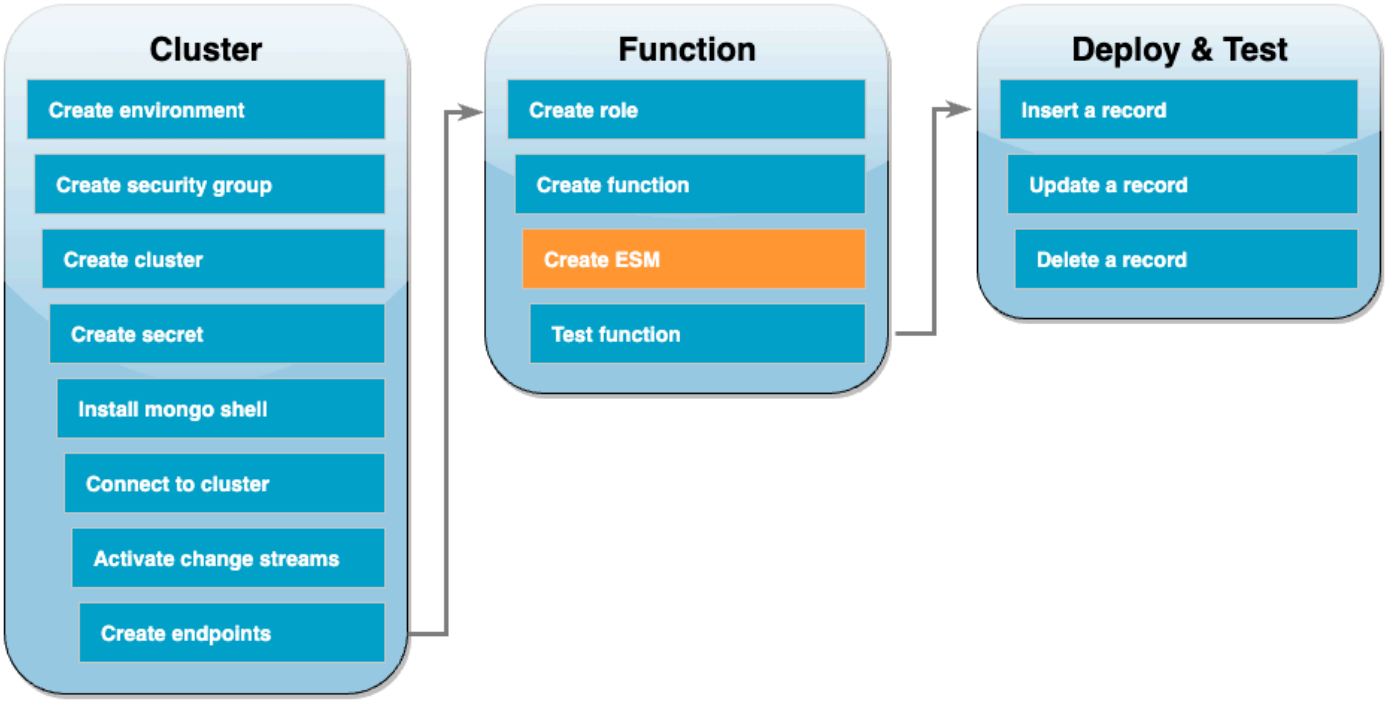
1. Copiez l'exemple de code dans un fichier nommé `index.js`.
2. Créez un package de déploiement avec la commande suivante.

```
zip function.zip index.js
```

3. Utilisez la CLI commande suivante pour créer la fonction. Remplacez `us-east-1` par le Région AWS, et `123456789012` par votre identifiant de compte.

```
aws lambda create-function \
  --function-name ProcessDocumentDBRecords \
  --zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \
  --region us-east-1 \
  --role arn:aws:iam::123456789012:role/AWSDocumentDBLambdaExecutionRole
```

## Création du mappage des sources d'événements Lambda

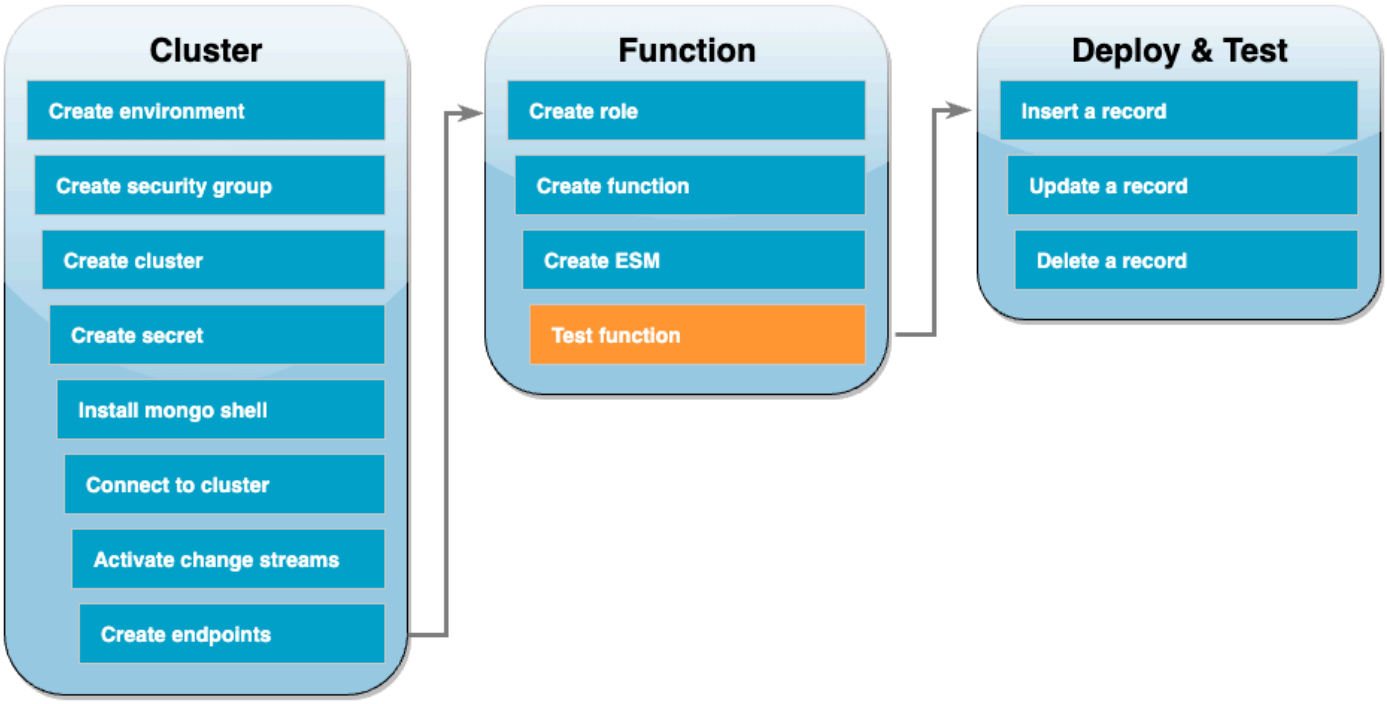



Créez le mappage des sources d'événements qui associe votre flux de modifications Amazon DocumentDB à votre fonction Lambda. Une fois que vous avez créé ce mappage des sources d'événements, commence AWS Lambda immédiatement à interroger le flux.

Pour créer le mappage des sources d'événements

1. Ouvrez la page Fonctions de la console Lambda.
2. Choisissez la fonction `ProcessDocumentDBRecords` que vous avez créée précédemment.
3. Choisissez l'onglet Configuration, puis choisissez Déclencheurs dans le menu de gauche.
4. Choisissez Add trigger (Ajouter déclencheur).
5. Sous Configuration du déclencheur, pour la source, sélectionnez Amazon DocumentDB.
6. Créez le mappage des sources d'événements avec la configuration suivante :
   - Cluster Amazon DocumentDB : choisissez le cluster que vous avez créé précédemment.
   - Nom de la base de données — `docdbdemo`
   - Nom de la collection — produits
   - Taille du lot — 1
   - Position de départ — Dernière
   - Authentification — BASIC _ AUTH
   - Clé Secrets Manager — Choisissez celle `DocumentDBSecret` que vous venez de créer.
   - Fenêtre Batch — 1
   - Configuration complète du document — UpdateLookup
7. Choisissez Ajouter. La création de votre mappage des sources d'événements peut prendre quelques minutes.

## Test de votre fonction : invocation manuelle




Pour tester que vous avez créé correctement votre fonction et le mappage des sources d'événements, invoquez votre fonction à l'aide de la commande `invoke`. Pour ce faire, copiez d'abord l'événement suivant JSON dans un fichier appelé `input.txt` :

```
{
  "eventSourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:canaryclusterb2a659a2-
qo5tcmqkcl03",
  "events": [
    {
      "event": {
        "_id": {
          "_data": "0163eeb6e7000000090100000009000041e1"
        },
        "clusterTime": {
          "$timestamp": {
            "t": 1676588775,
            "i": 9
          }
        },
        "documentKey": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
```

```
          }
        },
        "fullDocument": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          },
          "anyField": "sampleValue"
        },
        "ns": {
          "db": "docdbdemo",
          "coll": "products"
        },
        "operationType": "insert"
      }
    }
  ],
  "eventSource": "aws:docdb"
}
```

Ensuite, utilisez la commande suivante pour invoquer votre fonction avec cet événement :

```
aws lambda invoke \
  --function-name ProcessDocumentDBRecords \
  --cli-binary-format raw-in-base64-out \
  --region us-east-1 \
  --payload file://input.txt out.txt
```

Vous devriez obtenir une réponse qui ressemble à ce qui suit :

```
{
   "StatusCode": 200,
   "ExecutedVersion": "$LATEST"
}
```

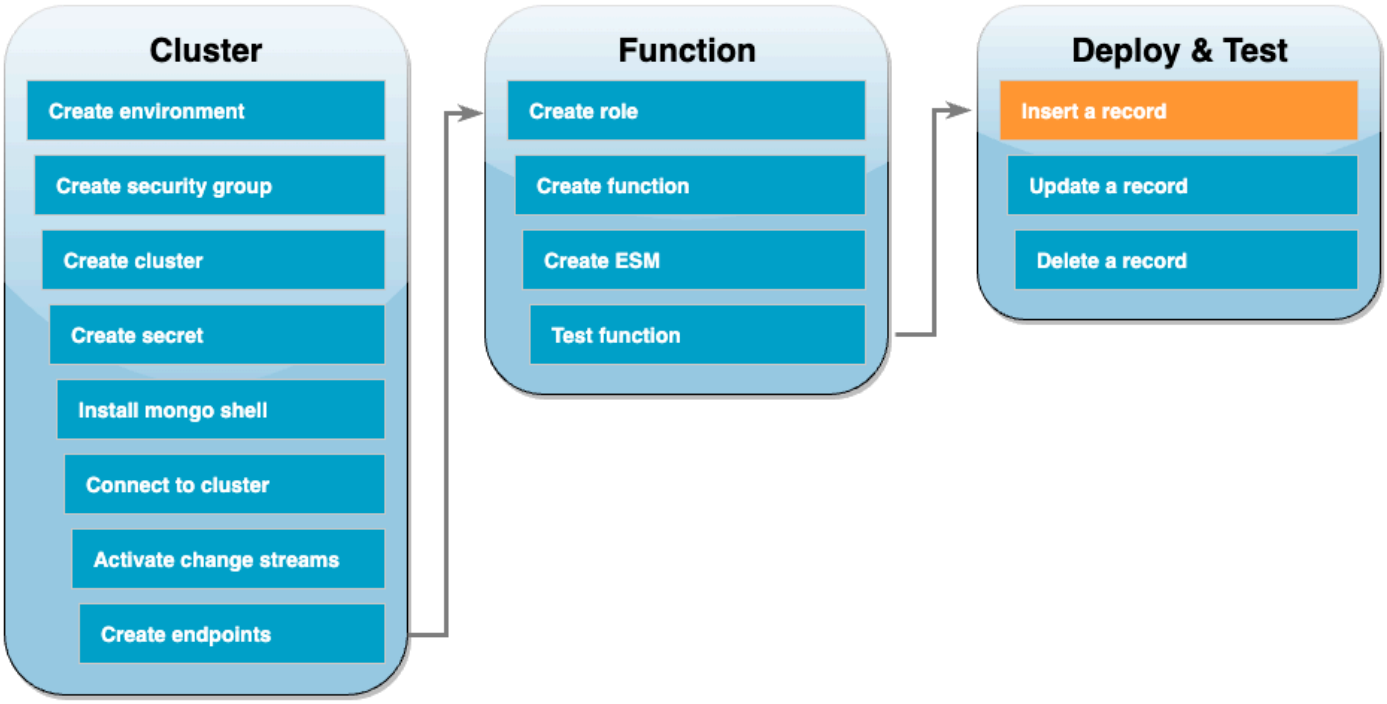
Vous pouvez vérifier que votre fonction a bien traité l'événement en consultant CloudWatch Logs.

Pour vérifier l'invocation manuelle via CloudWatch les journaux

1. Ouvrez la page Fonctions de la console Lambda.
2. Choisissez l'onglet Surveiller, puis sélectionnez Afficher CloudWatch les journaux. Cela vous amène au groupe de journaux spécifique associé à votre fonction dans la CloudWatch console.

3. Choisissez le flux de journaux le plus récent. Dans les messages du journal, vous devriez voir l'événementJSON.

## Test de votre fonction : insertion d'un enregistrement




Testez votre end-to-end configuration en interagissant directement avec votre base de données Amazon DocumentDB. Dans les étapes suivantes, vous allez insérer un enregistrement, le mettre à jour, puis le supprimer.

Pour insérer un enregistrement

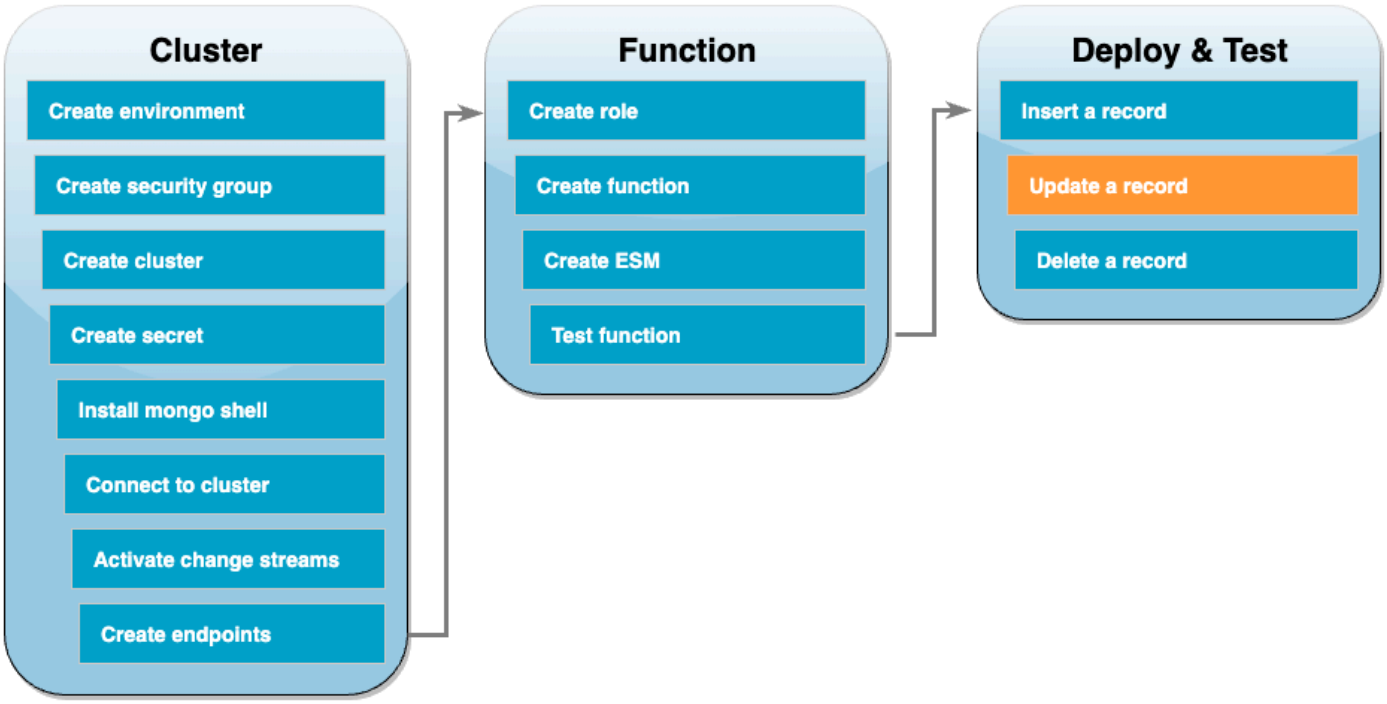
1. Reconnectez-vous à votre cluster Amazon DocumentDB dans AWS Cloud9 votre environnement.
2. Utilisez cette commande pour vous assurer que vous utilisez actuellement la base de données `docdbdemo` :

```
use docdbdemo
```

3. Insérez un enregistrement dans la collection `products` de la base de donnée `docdbdemo` :

```
db.products.insert({"name":"Pencil", "price": 1.00})
```

## Test de votre fonction : mise à jour d'un enregistrement

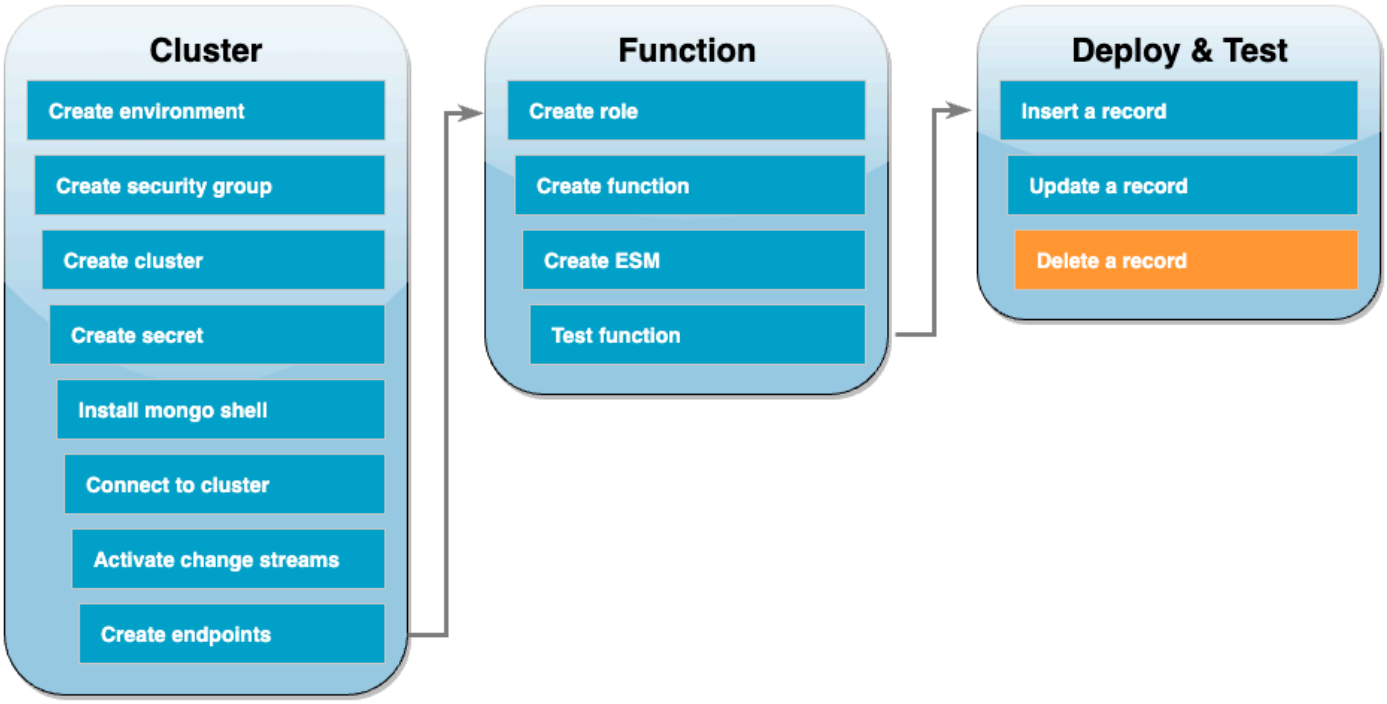



Ensuite, mettez à jour l'enregistrement que vous venez d'insérer à l'aide de la commande suivante :

```
db.products.update(
    { "name": "Pencil" },
    { $set: { "price": 0.50 }}
)
```

Vérifiez que votre fonction a correctement traité cet événement en vérifiant CloudWatch les journaux.

## Test de votre fonction : suppression d'un enregistrement




Enfin, supprimez l'enregistrement que vous venez de mettre à jour à l'aide de la commande suivante :

```
db.products.remove( { "name": "Pencil" } )
```

Vérifiez que votre fonction a correctement traité cet événement en vérifiant CloudWatch les journaux.

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant des ressources AWS que vous n'utilisez plus, vous évitez les frais superflus pour votre Compte AWS.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer les points de VPC terminaison

1. Ouvrez la VPCconsole. Dans le menu de gauche, sous Cloud privé virtuel, choisissez Points de terminaison.
2. Sélectionnez les points de terminaison que vous avez créés.
3. Choisissez Actions, puis Supprimer les VPC points de terminaison.
4. Saisissez **`delete`** dans le champ de saisie de texte.
5. Sélectionnez Delete (Supprimer).

Pour supprimer le cluster Amazon DocumentDB

1. Ouvrez la console Amazon DocumentDB.
2. Choisissez le cluster Amazon DocumentDB que vous avez créé pour ce didacticiel et désactivez la protection contre la suppression.
3. Sur la page principale des clusters, sélectionnez à nouveau votre cluster Amazon DocumentDB.
4. Sélectionnez Actions, Supprimer.
5. Pour Créer un instantané final du cluster, sélectionnez Non.
6. Saisissez **`delete`** dans le champ de saisie de texte.
7. Sélectionnez Delete (Supprimer).

Pour supprimer le secret dans Secrets Manager

1. Ouvrez la console Secrets Manager.
2. Sélectionnez le secret que vous avez créé pour ce tutoriel.
3. Choisissez Actions, Supprimer le secret.
4. Choisissez Schedule deletion (Planifier la suppression).

Pour supprimer le groupe EC2 de sécurité Amazon

1. Ouvrez la EC2console. Sous Réseau et sécurité, choisissez Groupes de sécurité.
2. Sélectionnez le groupe de sécurité que vous avez créé pour ce tutoriel.
3. Choisissez Actions, Supprimer les groupes de sécurité.
4. Sélectionnez Delete (Supprimer).

Pour supprimer l' AWS Cloud9 environnement

1. Ouvrez la AWS Cloud9 console.
2. Sélectionnez l'environnement que vous avez créé pour ce tutoriel.
3. Sélectionnez Delete (Supprimer).
4. Saisissez **`delete`** dans le champ de saisie de texte.
5. Sélectionnez Supprimer.

# Utilisation AWS Lambda avec Amazon DynamoDB

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Vous pouvez utiliser une AWS Lambda fonction pour traiter les enregistrements d'un flux Amazon DynamoDB. DynamoDB Streams vous permet de déclencher une fonction Lambda pour effectuer un travail supplémentaire chaque fois qu'une table DynamoDB est mise à jour.

Rubriques


- Flux d'interrogation et de mise en lots
- Positions de départ des interrogations et des flux
- Lecteurs simultanés d'une partition dans DynamoDB Streams
- Exemple d'évènement
- Traiter les enregistrements DynamoDB avec Lambda
- Configuration d'une réponse par lots partielle avec DynamoDB et Lambda
- Conserver les enregistrements supprimés pour une source d'événements DynamoDB dans Lambda
- Surveillance à l'aide de CloudWatch journaux
- Implémentation du traitement dynamique des flux DynamoDB dans Lambda
- Paramètres Lambda pour les mappages de sources d'événements Amazon DynamoDB
- Utilisation du filtrage des événements avec une source d'événements DynamoDB
- Tutoriel : Utilisation AWS Lambda avec les flux Amazon DynamoDB


## Flux d'interrogation et de mise en lots

Lambda interroge les partitions de votre flux DynamoDB en quête d'enregistrements à une fréquence de base de quatre fois par seconde. Lorsque des enregistrements sont disponibles, Lambda invoque votre fonction et attend le résultat. Si le traitement réussit, Lambda reprend l'interrogation jusqu'à recevoir plus d'enregistrements.

Par défaut, Lambda invoque votre fonction dès que des enregistrements sont disponibles. Si le lot que Lambda lit à partir de la source d'événements ne comprend qu'un seul enregistrement, Lambda envoie un seul registre à la fonction. Pour éviter d'invoquer la fonction avec un petit nombre de registres, vous pouvez indiquer à la source d'événement de les mettre en mémoire tampon pendant 5 minutes en configurant une fenêtre de traitement par lots. Avant d'invoquer la fonction, Lambda continue de lire les registres de la source d'événements jusqu'à ce qu'il ait rassemblé un lot complet, que la fenêtre de traitement par lot expire ou que le lot atteigne la limite de charge utile de 6 Mo. Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Lambda n'attend pas que les extensions configurées soient terminées avant d'envoyer le lot suivant pour traitement. En d'autres termes, vos extensions peuvent continuer à s'exécuter pendant que Lambda traite le prochain lot d'enregistrements. Cela peut causer des problèmes de limitation si vous enfreignez l'un des paramètres ou l'une des limites de simultanéité de votre compte. Pour détecter s'il s'agit d'un problème éventuel, surveillez vos fonctions et vérifiez si vous observez des métriques de simultanéité plus élevées que prévu pour votre mappage des sources d'événements. En raison de la brièveté des intervalles entre les invocations, Lambda peut brièvement signaler une utilisation simultanée supérieure au nombre de partitions. Cela peut être vrai même pour les fonctions Lambda sans extensions.

Configurez le ParallelizationFactorparamètre pour traiter une partition d'un flux DynamoDB avec plusieurs invocations Lambda simultanément. Vous pouvez spécifier le nombre de lots simultanés que Lambda interroge à partir d'une partition via un facteur de parallélisation de 1 (par défaut) à 10. Par exemple, lorsque vous définissez `ParallelizationFactor` la valeur 2, vous pouvez avoir 200 appels Lambda simultanés au maximum pour traiter 100 fragments de flux DynamoDB (bien qu'en pratique, vous puissiez voir des valeurs différentes pour la métrique). `ConcurrentExecutions` Cela permet d'augmenter le débit de traitement quand le volume de données est volatil et que la valeur du paramètre `IteratorAge` est élevée. Lorsque vous augmentez le nombre de lots simultanés par partition, Lambda garantit toujours le traitement dans l'ordre au niveau de l'élément (partition et clé de tri).

## Positions de départ des interrogations et des flux

Sachez que l'interrogation des flux lors des mises à jour et de la création du mappage des sources d'événements est finalement cohérente.

- Lors de la création du mappage des sources d'événements, le démarrage de l'interrogation des événements depuis le flux peut prendre plusieurs minutes.
- Lors des mises à jour du mappage des sources d'événements, l'arrêt et le redémarrage de l'interrogation des événements depuis le flux peuvent prendre plusieurs minutes.

Ce comportement signifie que si vous spécifiez `LATEST` comme position de départ du flux, le mappage des sources d'événements peut manquer des événements lors de la création ou des mises à jour. Pour vous assurer de ne manquer aucun événement, spécifiez la position de départ du flux comme `TRIM_HORIZON`.

## Lecteurs simultanés d'une partition dans DynamoDB Streams

Pour les tables à région unique qui ne sont pas des tables globales, vous pouvez concevoir jusqu'à deux fonctions Lambda pour lire la même partition DynamoDB Streams simultanément. Le dépassement de cette limite peut se traduire par une limitation de la demande. Pour les tables globales, nous vous recommandons de limiter le nombre de fonctions simultanées à une seule pour éviter la limitation des demandes.

## Exemple d'évènement

Example

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
```

```
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "SizeBytes": 26
      },
      "awsRegion": "us-west-2",
      "eventName": "INSERT",
      "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
      "eventSource": "aws:dynamodb"
    },
    {
      "eventID": "2",
      "eventVersion": "1.0",
      "dynamodb": {
        "OldImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "222",
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "SizeBytes": 59,
        "NewImage": {
          "Message": {
            "S": "This item has changed"
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
```

```
      },
      "awsRegion": "us-west-2",
      "eventName": "MODIFY",
      "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
      "eventSource": "aws:dynamodb"
    }
  ]}
```

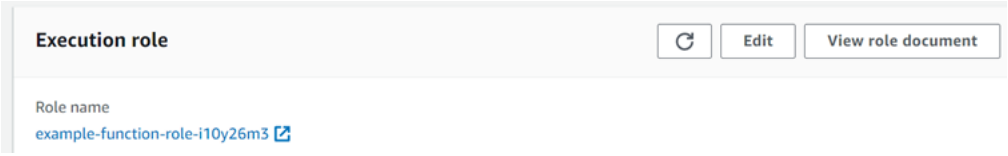
## Traiter les enregistrements DynamoDB avec Lambda

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des enregistrements de votre flux à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter les mêmes données avec plusieurs fonctions Lambda, ou traiter des éléments de plusieurs flux avec une seule fonction.

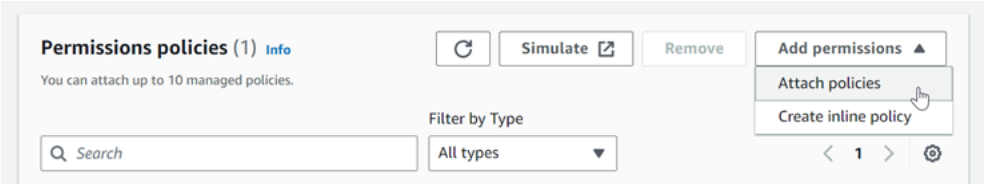
Pour configurer votre fonction de manière à lire depuis DynamoDB Streams, associez la politique AWSLambdaDynamoDBExecutionRole AWS gérée à votre rôle d'exécution, puis créez un déclencheur DynamoDB.

Pour ajouter des autorisations et créer un déclencheur

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Choisissez l'onglet Configuration, puis Permissions (Autorisations).
4. Sous Nom du rôle, choisissez le lien vers votre rôle d'exécution. Ce lien ouvre le rôle dans la IAM console.



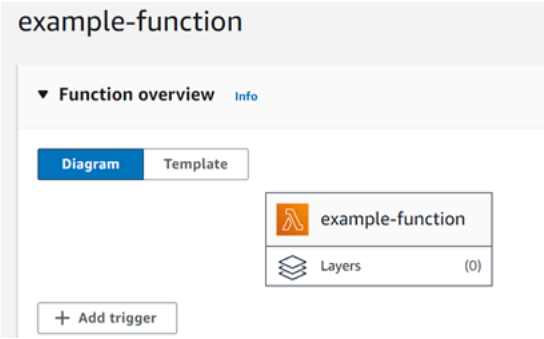

5. Choisissez Ajouter des autorisations, puis Attacher des politiques.




6. Dans le champ de recherche, entrez `AWSLambdaDynamoDBExecutionRole`. Ajoutez cette politique à votre rôle d'exécution. Il s'agit d'une politique AWS gérée qui contient les autorisations

dont votre fonction a besoin pour lire le flux DynamoDB. Pour plus d'informations sur cette politique, consultez AWSLambdaDynamoDBExecutionRolela référence des politiques AWS gérées.

7. Retournez à votre fonction dans la console Lambda. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).




8. Choisissez un type de déclencheur.
9. Configurez les options requises, puis choisissez Add (Ajouter).

Lambda prend en charge les options suivantes pour les sources d'événements DynamoDB :

Options de source d'événement

- DynamoDB table (Table DynamoDB) – Table DynamoDB à partir de laquelle lire les enregistrements.
- Batch size (Taille de lot) – Nombre d'enregistrements par lot à envoyer à la fonction, jusqu'à 10 000. Lambda transmet tous les enregistrements du lot à la fonction en une seule invocation, tant que la taille totale des événements ne dépasse pas la limite de charge utile pour une invocation synchrone (6 Mo).
- Batch window (Fenêtre de traitement par lots) – Intervalle de temps maximum (en secondes) pour collecter des enregistrements avant d'invoquer la fonction.
- Starting position (Position de départ) – Traiter uniquement les nouveaux enregistrements, ou tous enregistrement existants.
  - Latest (Derniers) – Traiter les nouveaux enregistrements ajoutés au flux.
  - Trim horizon (Supprimer l'horizon) – Traiter tous les enregistrements figurant dans le flux.

  Après le traitement de tous les enregistrements existants, la fonction est à jour et continue à traiter les nouveaux enregistrements.

- Destination en cas d'échec : SQS file d'attente standard ou SNS rubrique standard pour les enregistrements qui ne peuvent pas être traités. Quand Lambda écarte un lot d'enregistrements qui est trop ancien ou qui a épuisé toutes les tentatives, il envoie les détails du lot à la file d'attente ou à la rubrique.
- Retry attempts (Nouvelles tentatives) – Nombre maximum de nouvelles tentatives que Lambda effectue quand la fonction renvoie une erreur. Cela ne s'applique pas aux erreurs ou limitations de service où le lot n'a pas atteint la fonction.
- Maximum age of record (Âge maximum d'enregistrement) – Âge maximum d'un enregistrement que Lambda envoie à votre fonction.
- Split batch on error (Fractionner le lot en cas d'erreur) – Quand la fonction renvoie une erreur, diviser le lot en deux avant d'effectuer une nouvelle tentative. Le paramètre de taille de lot d'origine reste inchangé.
- Concurrent batches per shard (Lots simultanés par partition) – Traite simultanément plusieurs lots de la même partition.
- Enabled (Activé) – Valeur définie sur VRAI pour activer le mappage de source d'événement. Définissez la valeur sur « false » pour arrêter le traitement des enregistrements. Lambda garde une trace du dernier enregistrement traité et reprend le traitement à ce point lorsque le mappage est réactivé.

> **Note**
>
> Les GetRecords API appels invoqués par Lambda dans le cadre des déclencheurs DynamoDB ne vous sont pas facturés.

Pour gérer ultérieurement la configuration de la source d'événement, choisissez le déclencheur dans le concepteur.

## Configuration d'une réponse par lots partielle avec DynamoDB et Lambda

Lors de l'utilisation et du traitement de données de streaming à partir d'une source d'événement, par défaut, Lambda n'effectue un point de contrôle sur le numéro de séquence le plus élevé d'un lot que lorsque celui-ci est un succès complet. Lambda traite tous les autres résultats comme un échec complet et recommence à traiter le lot jusqu'à atteindre la limite de nouvelles tentatives. Pour autoriser des succès partiels lors du traitement des lots à partir d'un flux, activez `ReportBatchItemFailures`. Autoriser des succès partiels peut permettre de réduire le nombre

de nouvelles tentatives sur un enregistrement, mais cela n'empêche pas entièrement la possibilité de nouvelles tentatives dans un enregistrement réussi.

Pour l'activer`ReportBatchItemFailures`, incluez la valeur enum **`ReportBatchItemFailures`** dans la FunctionResponseTypesliste. Cette liste indique quels types de réponse sont activés pour votre fonction. Vous pouvez configurer cette liste lorsque vous créez ou mettez à jour un mappage de source d'événements.

### Syntaxe du rapport

Lors de la configuration des rapports d'échec d'articles de lot, la classe `StreamsEventResponse` est renvoyée avec une liste d'échecs d'articles de lot. Vous pouvez utiliser un objet `StreamsEventResponse` pour renvoyer le numéro de séquence du premier enregistrement ayant échoué dans le lot. Vous pouvez également créer votre classe personnalisée en utilisant la syntaxe de réponse correcte. La JSON structure suivante indique la syntaxe de réponse requise :

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Si le tableau `batchItemFailures` contient plusieurs éléments, Lambda utilise l'enregistrement portant le numéro de séquence le plus bas comme point de contrôle. Lambda réessaie ensuite tous les enregistrements à partir de ce point de contrôle.

### Conditions de réussite et d'échec

Lambda traite un lot comme un succès complet si vous renvoyez l'un des éléments suivants :

- Une liste `batchItemFailure` vide
- Une liste `batchItemFailure` nulle
- Une `EventResponse` vide
- Un nu `EventResponse`

Lambda traite un lot comme un échec complet si vous renvoyez l'un des éléments suivants :

- Une chaîne vid `itemIdentifier`
- Un `itemIdentifier` nul
- Un `itemIdentifier` avec un nom de clé incorrect

Lambda effectue des nouvelles tentatives en cas d'échec en fonction de votre stratégie de nouvelle tentative.

## Diviser un lot

Si votre invocation échoue et que `BisectBatchOnFunctionError` est activé, le lot est divisé en deux quel que soit votre paramètre `ReportBatchItemFailures`.

Quand une réponse de succès partiel de lot est reçue et que les paramètres `BisectBatchOnFunctionError` et `ReportBatchItemFailures` sont activés, le lot est divisé au numéro de séquence renvoyé, et Lambda n'effectue de nouvelle tentative que sur les enregistrements restants.

Voici quelques exemples de code de fonction qui renvoie la liste des messages ayant échoué IDs dans le lot :

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
```

```
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }
```

```
 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
```

```
public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
```

```
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. PHP

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler des défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
```

```
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## Conserver les enregistrements supprimés pour une source d'événements DynamoDB dans Lambda

La gestion des erreurs pour les mappages de sources d'événements DynamoDB dépend du fait que l'erreur se produit avant que la fonction ne soit invoquée ou pendant l'invocation de la fonction :

- Avant l'appel : si un mappage de source d'événement Lambda ne parvient pas à appeler la fonction en raison d'un ralentissement ou d'autres problèmes, il réessaie jusqu'à ce que les enregistrements

expirent ou dépassent l'âge maximum configuré sur le mappage de source d'événement (). MaximumRecordAgeInSeconds

- Pendant l'appel : si la fonction est invoquée mais renvoie une erreur, Lambda réessaie jusqu'à ce que les enregistrements expirent, dépassent l'âge maximum MaximumRecordAgeInSeconds() ou atteignent le quota de nouvelles tentatives configuré (). MaximumRetryAttempts Pour les erreurs de fonctionnement, vous pouvez également configurer BisectBatchOnFunctionError, ce qui divise un lot défaillant en deux lots plus petits, isolant ainsi les mauvais enregistrements et évitant les délais d'attente. Le fractionnement des lots ne consomme pas le quota de nouvelles tentatives.

Si les mesures de gestion des erreurs échouent, Lambda ignore les enregistrements et poursuit le traitement des lots du flux. Avec les paramètres par défaut, cela signifie qu'un enregistrement défectueux peut bloquer le traitement sur la partition affectée pendant jusqu'à une journée. Pour éviter cela, configurez le mappage de source d'événement de votre fonction avec un nombre raisonnable de nouvelles tentatives et un âge maximum d'enregistrement correspondant à votre cas d'utilisation.

## Configuration des destinations pour les appels échoués

Pour retenir les enregistrements des invocations de mappage de sources d'événements qui ont échoué, ajoutez une destination au mappage des sources d'événements de votre fonction. Chaque enregistrement envoyé à la destination est un JSON document contenant des métadonnées relatives à l'échec de l'invocation. Vous pouvez configurer n'importe quel SNS sujet Amazon ou SQS file d'attente Amazon comme destination. Votre rôle d'exécution doit disposer d'autorisations pour la destination :

- Pour les SQS destinations : sqs : SendMessage
- Pour les SNS destinations : SNS:Publish

Pour configurer une destination en cas de panne à l'aide de la console, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Invocation du mappage des sources d'événements.

5. Pour le mappage des sources d'événements, choisissez une source d'événements configurée pour cette fonction.
6. Pour Condition, sélectionnez En cas d'échec. Pour les invocations de mappage des sources d'événements, il s'agit de la seule condition acceptée.
7. Pour Type de destination, choisissez le type de destination auquel Lambda envoie les enregistrements d'invocation.
8. Pour Destination, choisissez une ressource.
9. Choisissez Save (Enregistrer).

Vous pouvez également configurer une destination en cas de panne à l'aide de AWS Command Line Interface (AWS CLI). Par exemple, la create-event-source-mappingcommande suivante ajoute un mappage de source d'événement avec une destination SQS en cas de défaillance à `MyFunction` :

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

La update-event-source-mappingcommande suivante met à jour le mappage d'une source d'événements afin d'envoyer les enregistrements d'invocation ayant échoué vers une SNS destination après deux tentatives, ou si les enregistrements datent de plus d'une heure.

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 \
--maximum-record-age-in-seconds 3600 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-
east-1:123456789012:dest-topic"}}'
```

Les paramètres mis à jour sont appliqués de façon asynchrone et ne sont pas reflétés dans la sortie tant que le processus n'est pas terminé. Utilisez la get-event-source-mappingcommande pour afficher l'état actuel.

Pour supprimer une destination, entrez une chaîne vide comme argument du paramètre `destination-config` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

L'exemple suivant illustre un enregistrement d'invocation pour un flux DynamoDB.

Example Enregistrement d'invocation

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:13:49.717Z",
    "DDBStreamBatchInfo": {
        "shardId": "shardId-00000001573689847184-864758bb",
        "startSequenceNumber": "800000000003126276362",
        "endSequenceNumber": "800000000003126276362",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",
        "batchSize": 1,
        "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/
stream/2019-11-14T00:04:06.388"
    }
}
```

Vous pouvez utiliser ces informations pour récupérer les enregistrements concernés à partir du flux à des fins de résolution de problèmes. Les enregistrements réels n'étant pas inclus, vous devez les récupérer du flux avant qu'ils expirent et soient perdus.

## Surveillance à l'aide de CloudWatch journaux

Lambda émet la métrique `IteratorAge` lorsque votre fonction termine le traitement d'un lot d'enregistrements. Cette métrique indique l'âge du dernier enregistrement du lot à la fin du traitement.

Si votre fonction traite de nouveaux événements, vous pouvez utiliser l'âge de l'itérateur pour estimer la latence entre le moment où un enregistrement est ajouté et celui où la fonction le traite.

Une tendance à la hausse de l'âge de l'itérateur peut indiquer des problèmes liés à votre fonction. Pour de plus amples informations, veuillez consulter Afficher les métriques des fonctions Lambda.

## Implémentation du traitement dynamique des flux DynamoDB dans Lambda

Les fonctions Lambda peuvent exécuter des applications de traitement de flux continu. Un flux représente des données illimitées qui circulent en continu dans votre application. Pour analyser les informations provenant de cette entrée de mise à jour continue, vous pouvez lier les enregistrements inclus à l'aide d'une fenêtre de temps définie.

Les fenêtres bascules sont des fenêtres temporelles distinctes qui s'ouvrent et se ferment à intervalles réguliers. Par défaut, les invocations Lambda sont sans état : vous ne pouvez pas les utiliser pour traiter des données sur plusieurs invocations continues sans base de données externe. Cependant, avec les fenêtres bascules, vous pouvez maintenir votre état au long des invocations. Cet état contient le résultat global des messages précédemment traités pour la fenêtre actuelle. Votre état peut être d'un maximum de 1 Mo par partition. S'il dépasse cette taille, Lambda met fin précocement à la fenêtre de traitement.

Chaque enregistrement d'un flux appartient à une fenêtre spécifique. La fonction Lambda traitera chaque enregistrement au moins une fois. Toutefois, elle ne garantit pas un seul traitement pour chaque enregistrement. Dans de rares cas, tels que pour la gestion des erreurs, certains enregistrements peuvent être sujet à de multiples traitements. Les dossiers sont toujours traités dans l'ordre dès la première fois. Si les enregistrements sont traités plusieurs fois, ils peuvent être traités dans le désordre.

### Regroupement et traitement

Votre fonction gérée par l'utilisateur est invoquée tant pour l'agrégation que pour le traitement des résultats finaux de celle-ci. Lambda regroupe tous les enregistrements reçus dans la fenêtre. Vous pouvez recevoir ces enregistrements en plusieurs lots, chacun sous forme d'invocation séparée. Chaque invocation reçoit un état. Ainsi, lorsque vous utilisez des fenêtres bascules, votre réponse de fonction Lambda doit contenir une propriété `state`. Si la réponse ne contient pas de propriété `state`, Lambda considère qu'il s'agit d'une invocation ayant échoué. Pour satisfaire à cette condition, votre fonction peut renvoyer un `TimeWindowEventResponse` objet ayant la JSON forme suivante :

Example **`TimeWindowEventResponse`** values

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Pour les fonctions Java, nous vous recommandons d'utiliser `Map<String, String>` pour représenter l'état.

À la fin de la fenêtre, l'indicateur `isFinalInvokeForWindow` est défini sur `true` pour indiquer qu'il s'agit de l'état final et qu'il est prêt pour le traitement. Après le traitement, la fenêtre et votre invocation final se terminent, puis l'état est supprimé.

À la fin de votre fenêtre, Lambda applique un traitement final pour les actions sur les résultats de l'agrégation. Votre traitement final est invoqué de manière synchrone. Une fois l'invocation réussie, votre fonction contrôle le numéro de séquence et le traitement du flux continue. Si l'invocation échoue, votre fonction Lambda suspend le traitement ultérieur jusqu'à ce que l'invocation soit réussie.

Example DynamodbTimeWindowEvent

```
{
   "Records":[
      {
         "eventID":"1",
         "eventName":"INSERT",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
```

```
            },
            "NewImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"111",
            "SizeBytes":26,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"2",
         "eventName":"MODIFY",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"222",
```

```
            "SizeBytes":59,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"3",
         "eventName":"REMOVE",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"333",
            "SizeBytes":38,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      }
   ],
    "window": {
        "start": "2020-07-30T17:00:00Z",
        "end": "2020-07-30T17:05:00Z"
    },
    "state": {
        "1": "state1"
    },
    "shardId": "shard123456789",
    "eventSourceARN": "stream-ARN",
    "isFinalInvokeForWindow": false,
    "isWindowTerminatedEarly": false
```

```
}
```

## Configuration

Vous pouvez configurer des fenêtres bascule lorsque vous créez ou mettez à jour un mappage de source d'événement. Pour configurer une fenêtre variable, spécifiez la fenêtre en secondes (TumblingWindowInSeconds). L'exemple de commande suivant AWS Command Line Interface (AWS CLI) crée un mappage des sources d'événements de streaming dont la fenêtre de basculement est de 120 secondes. La fonction Lambda définie pour l'agrégation et le traitement est nommée `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525 \
--function-name tumbling-window-example-function \
--starting-position TRIM_HORIZON \
--tumbling-window-in-seconds 120
```

Lambda détermine les limites des fenêtres bascule en fonction de l'heure à laquelle les enregistrements ont été insérés dans le flux. Tous les enregistrements ont un horodatage approximatif disponible que Lambda utilise pour déterminer des limites.

Les agrégations de fenêtres bascule ne prennent pas en charge le repartitionnement. Quand la partition prend fin, Lambda considère que la fenêtre de traitement est fermée, et les partitions enfants entament leur propre fenêtre de traitement dans un nouvel état.

Les fenêtres bascule prennent complètement en charge les stratégies de nouvelle tentative existantes `maxRetryAttempts` et `maxRecordAge`.

Example Handler.py – Agrégation et traitement

La fonction Python suivante montre comment regrouper et traiter votre état final :

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

#Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
```

```
    else:
        print('Aggregate invoke')

#Check for early terminations
    if event['isWindowTerminatedEarly']:
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['dynamodb']['NewImage']['Id']] = state.get(record['dynamodb']
['NewImage']['Id'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

## Paramètres Lambda pour les mappages de sources d'événements Amazon DynamoDB

Tous les types de sources d'événements Lambda partagent les mêmes opérations CreateEventSourceMapping. UpdateEventSourceMappingAPI Cependant, seuls certains paramètres s'appliquent à DynamoDB Streams.

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| BatchSize | N | 100 | Maximum : 10 000. |
| BisectBatchOnFunctionError | N | FAUX | none |
| DestinationConfig | N | N/A | SQSFile d'attente Amazon standard ou destination de SNS rubrique Amazon standard pour les enregistrements supprimés |
| Activées | N | VRAI | none |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| EventSourceArn | Y | N/A | ARNdu flux de données ou d'un consommateur de flux |
| FilterCriteria | N | N/A | Contrôlez les événements que Lambda envoie à votre fonction |
| FunctionName | Y | N/A | none |
| FunctionResponseTypes | N | N/A | Pour permettre à votre fonction de signaler des échecs spécifiques dans un lot, incluez la valeur `ReportBatchItemFailures` dans `FunctionResponseTypes` . Pour de plus amples informations, veuillez consulter Configuration d'une réponse par lots partielle avec DynamoDB et Lambda. |
| MaximumBatchingWindowInSeconds | N | 0 | none |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| MaximumRecordAgeInSeconds | N | -1 | -1 signifie infini : les enregistrements ayant échoué sont réessayés jusqu'à leur expiration. La limite de conservation des données pour DynamoDB Streams est de 24 heures.<br><br>Minimum : -1<br><br>Maximum : 604 800 |
| MaximumRetryAttempts | N | -1 | -1 signifie infini : les registres qui ont échoué sont réessayés jusqu'à ce que le registre expire.<br><br>Minimum : 0<br><br>Maximum : 10 000. |
| ParallelizationFactor | N | 1 | Maximum : 10 |
| StartingPosition | Y | N/A | TRIM_ HORIZON ou LATEST |
| TumblingWindowInSeconds | N | N/A | Minimum : 0<br><br>Maximum : 900 |

# Utilisation du filtrage des événements avec une source d'événements DynamoDB

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources d'événements DynamoDB.

Rubriques


- Evénement DynamoDB
- Filtrage à l'aide des attributs de table
- Filtrage à l'aide d'expressions booléennes
- Utilisation de l'opérateur Exists
- JSONformat pour le filtrage DynamoDB


## Evénement DynamoDB

Supposons que vous ayez une table DynamoDB avec la clé primaire `CustomerName` et les attributs `AccountManager` et `PaymentTerms`. La figure suivante montre un exemple d'enregistrement provenant du flux de votre table DynamoDB.

```
{
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
          "ApproximateCreationDateTime": "1678831218.0",
          "Keys": {
              "CustomerName": {
                  "S": "AnyCompany Industries"
              },
              "NewImage": {
                  "AccountManager": {
                      "S": "Pat Candella"
                  },
                  "PaymentTerms": {
                      "S": "60 days"
                  },
```

```
                "CustomerName": {
                    "S": "AnyCompany Industries"
                }
            },
            "SequenceNumber": "111",
            "SizeBytes": 26,
            "StreamViewType": "NEW_IMAGE"
        }
    }
}
```

Pour filtrer sur la base des valeurs de clé et d'attribut de votre table DynamoDB, utilisez la clé `dynamodb` dans l'enregistrement. Les sections suivantes fournissent des exemples de différents types de filtres.

Filtrage à l'aide des touches de table

Supposons que vous souhaitiez que votre fonction traite uniquement les enregistrements dont la clé primaire `CustomerName` est « AnyCompany  Industries ». L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" :
 [ \"AnyCompany Industries\" ] } } } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "dynamodb": {
        "Keys": {
            "CustomerName": {
                "S": [ "AnyCompany Industries" ]
                }
            }
        }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
   Filters:
     - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }'
```

## Filtrage à l'aide des attributs de table

Avec DynamoDB, vous pouvez également utiliser les clés `NewImage` et `OldImage` pour filtrer les valeurs d'attributs. Supposons que vous vouliez filtrer les enregistrements où l'attribut `AccountManager` de la dernière image de la table est « Pat Candella » ou « Shirley Rodriguez ». L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "dynamodb": {
        "NewImage": {
            "AccountManager": {
                "S": [ "Pat Candella", "Shirley Rodriguez" ]
            }
        }
    }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
 "Shirley Rodriguez" ] } } } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella", "Shirley Rodriguez" ] } } } }'
```

## Filtrage à l'aide d'expressions booléennes

Vous pouvez également créer des filtres à l'aide d'expressions booléennesAND. Ces expressions peuvent inclure les paramètres de clé et d'attribut de votre table. Supposons que vous souhaitiez filtrer les enregistrements dont la valeur `NewImage` d'`AccountManager` est « Pat Candella » et la valeur `OldImage` est « Terry Whitlock ». L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
```

```
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager
\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "dynamodb" : {
        "NewImage" : {
            "AccountManager" : {
                "S" : [
                    "Pat Candella"
                ]
            }
        }
    },
    "dynamodb": {
        "OldImage": {
            "AccountManager": {
                "S": [
                    "Terry Whitlock"
                ]
            }
        }
    }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l’aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d’événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }
```

### AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" :
 { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }
 "}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d’événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" :
 { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }
 "}]}'
```

### AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }'
```

> **Note**
>
> Le filtrage d’événements DynamoDB ne prend pas en charge l’utilisation d’opérateurs numériques (égalité numérique et plage numérique). Même si les éléments de votre table

sont stockés sous forme de nombres, ces paramètres sont convertis en chaînes dans l'objet d'JSONenregistrement.

## Utilisation de l'opérateur Exists

En raison de la structure des objets d'JSONévénements de DynamoDB, l'utilisation de l'opérateur Exists nécessite une attention particulière. L'opérateur Exists ne fonctionne que sur les nœuds foliaires de l'événement. Par conséquentJSON, si votre modèle de filtre utilise Exists pour tester un nœud intermédiaire, il ne fonctionnera pas. Tenez compte de l'élément de table DynamoDB suivant :

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
      {"S":"Sales"},
      {"S":"Marketing"},
      {"S":"Support"}
    ]
  }
}
```

Vous souhaiterez peut-être créer un modèle de filtre comme le suivant pour tester les événements contenant `"Organizations"` :

```
{ "dynamodb" : { "NewImage" : { "Organizations" : [ { "exists": true } ] } } }
```

Cependant, ce modèle de filtre ne renverra jamais de correspondance car il ne `"Organizations"` s'agit pas d'un nœud feuille. L'exemple suivant montre comment utiliser correctement l'opérateur Exists pour créer le modèle de filtre souhaité :

```
{ "dynamodb" : { "NewImage" : {"Organizations": {"L": {"S": [ {"exists":
 true } ] } } } } }
```

## JSONformat pour le filtrage DynamoDB

Pour filtrer correctement les événements provenant de sources DynamoDB, le champ de données et vos critères de filtre pour le champ de données `dynamodb` () doivent être au format valide. JSON Si

l'un des champs n'est pas dans un JSON format valide, Lambda supprime le message ou lance une exception. Le tableau suivant résume le comportement spécifique :

| Format des données entrantes | Pas de modèle de filtre pour les propriétés des données | Action obtenue. |
|---|---|---|
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Non- JSON | Lambda lance une exception au moment de la création ou de la mise à jour du mappage de sources d'événements. Le modèle de filtre pour les propriétés des données doit être dans un JSON format valide. |
| Non- JSON | Valide JSON | Lambda rejette l'enregis trement. |
| Non- JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Non- JSON | Non- JSON | Lambda lance une exception au moment de la création ou de la mise à jour du mappage de sources d'événements. Le modèle de filtre pour les propriétés des données doit |

| Format des données entrantes | Pas de modèle de filtre pour les propriétés des données | Action obtenue. |
| --- | --- | --- |
| | | être dans un JSON format valide. |

# Tutoriel : Utilisation AWS Lambda avec les flux Amazon DynamoDB

Dans ce didacticiel, vous allez créer une fonction Lambda afin d'utiliser des événements d'un flux Amazon DynamoDB.

## Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. CLILes exemples de commandes présentés dans

ce guide utilisent le formatage Linux. Les commandes qui incluent JSON des documents intégrés doivent être reformatées si vous utilisez Windows. CLI

## Créer le rôle d'exécution

Créez le rôle d'exécution qui autorise votre fonction à accéder aux AWS ressources.

Pour créer un rôle d'exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :
   - Entité de confiance – Lambda.
   - Autorisations — AWSLambdaDynamoDBExecutionRole.
   - Nom de rôle – **`lambda-dynamodb-role`**.

AWSLambdaDynamoDBExecutionRoleDispose des autorisations nécessaires à la fonction pour lire des éléments depuis DynamoDB et écrire des journaux dans Logs. CloudWatch

## Créer la fonction

Créez une fonction Lambda qui traite vos événements DynamoDB. Le code de fonction écrit certaines des données d'événements entrants dans CloudWatch Logs.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
```

```
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement DynamoDB avec Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
```

```
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consommation d'un événement DynamoDB avec Lambda en utilisant. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
```

```
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. PHP

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
```

```
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement DynamoDB avec Lambda à l'aide de Python.

```
import json
```

```
def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

### Ruby

#### SDKpour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");
```

```
    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Pour créer la fonction

1. Copiez l'exemple de code dans un fichier nommé `example.js`.
2. Créez un package de déploiement.

```
zip function.zip example.js
```

3. Créez une fonction Lambda à l'aide de la commande `create-function`.

```
aws lambda create-function --function-name ProcessDynamoDBRecords \
    --zip-file fileb://function.zip --handler example.handler --runtime nodejs18.x
 \
    --role arn:aws:iam::111122223333:role/lambda-dynamodb-role
```

## Test de la fonction Lambda

Au cours de cette étape, vous appelez votre fonction Lambda manuellement à l'aide de la `invoke` AWS Lambda CLI commande et de l'exemple d'événement DynamoDB suivant. Copiez ce qui suit dans un fichier nomméinput.txt.

Example input.txt

```
{
   "Records":[
      {
         "eventID":"1",
         "eventName":"INSERT",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"111",
            "SizeBytes":26,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"2",
         "eventName":"MODIFY",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
```

```
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"222",
            "SizeBytes":59,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"3",
         "eventName":"REMOVE",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
```

```
                    "N":"101"
                 }
              },
              "SequenceNumber":"333",
              "SizeBytes":38,
              "StreamViewType":"NEW_AND_OLD_IMAGES"
           },
           "eventSourceARN":"stream-ARN"
        }
    ]
}
```

Exécutez la commande suivante `invoke`.

```
aws lambda invoke --function-name ProcessDynamoDBRecords \
    --cli-binary-format raw-in-base64-out \
    --payload file://input.txt outputfile.txt
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

La fonction renvoie la chaîne `message` dans le corps de la réponse.

Vérifiez la sortie dans le fichier `outputfile.txt`.

## Créer une table DynamoDB avec un flux activé

Créez une table Amazon DynamoDB avec un flux activé.

Pour créer une table DynamoDB

1. Ouvrez la console DynamoDB.
2. Choisissez Créer un tableau.
3. Créez une table avec les paramètres suivants.
   - Nom de la table – **`lambda-dynamodb-stream`**
   - Clé primaire – **`id`** (chaîne)

4. Choisissez Create (Créer).

Pour activer les flux

1. Ouvrez la console DynamoDB.
2. Choisissez Tables.
3. Choisissez la table lambda-dynamodb-stream.
4. Sous la section Exports and streams (Exportations et flux), choisissez DynamoDB stream details (Détails du flux DynamoDB).
5. Choisissez Activer.
6. Pour Type de vue, sélectionnez Attributs clés uniquement.
7. Choisissez Activer le stream.

Notez le streamARN. Vous en aurez besoin à l'étape suivante pour associer le flux à votre fonction Lambda. Pour plus d'informations sur l'activation des flux, consultez Capture d'activité Table avec DynamoDB Streams.

## Ajouter une source d'événement dans AWS Lambda

Créez un mappage de source d'événement dans AWS Lambda. Ce mappage de source d'événement associe le flux DynamoDB avec votre fonction Lambda. Après avoir créé ce mappage des sources d'événements, AWS Lambda commence à interroger le flux.

Exécutez la commande suivante AWS CLI `create-event-source-mapping`. Une fois la commande exécutée, notez leUUID. Vous en aurez besoin UUID pour faire référence au mappage de source d'événement dans toutes les commandes, par exemple, lors de la suppression du mappage de source d'événement.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
    --batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Cela a pour effet de créer un mappage entre le flux DynamoDB spécifié et la fonction Lambda. Vous pouvez associer un flux diffuser à plusieurs fonctions Lambda, et associer la même fonction Lambda à plusieurs flux. Toutefois, les fonctions Lambda partagent le débit de lecture du flux qu'elles partagent.

Pour obtenir la liste des mappages de source d'événement, exécutez la commande suivante.

```
aws lambda list-event-source-mappings
```

Cette liste renvoie tous les mappages de source d'événement que vous avez créés et indique la valeur `LastProcessingResult` pour chacun d'eux, entre autres. Ce champ est utilisé pour fournir un message d'information en cas de problème. Des valeurs telles que `No records processed` (indique que le sondage n' AWS Lambda a pas commencé ou qu'il n'y a aucun enregistrement dans le flux) et `OK` (indique que les enregistrements du flux AWS Lambda ont été lus avec succès et que votre fonction Lambda a été invoquée) indiquent qu'il n'y a aucun problème. Dans le cas contraire, vous recevez un message d'erreur.

Si vous avez un grand nombre de mappages de source d'événement, utilisez le paramètre du nom de la fonction pour affiner les résultats.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

## Tester la configuration

Testez l' end-to-endexpérience. A mesure que vous mettez la table à jour, DynamoDB écrit les enregistrements d'événement dans le flux. Quand AWS Lambda interroge le flux, il y détecte les nouveaux enregistrements et invoque pour vous la fonction Lambda en transmettant les événements à la fonction.

1. Dans la console DynamoDB, vous pouvez ajouter, mettre à jour et supprimer des éléments dans la table. DynamoDB écrit des enregistrements de ces actions dans le flux.
2. AWS Lambda interroge le flux et lorsqu'il détecte des mises à jour du flux, il invoque votre fonction Lambda en transmettant les données d'événements qu'il trouve dans le flux.
3. Votre fonction s'exécute et crée des journaux sur Amazon CloudWatch. Vous pouvez vérifier les journaux signalés dans la CloudWatch console Amazon.

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la table DynamoDB

1. Ouvrez la page Tables (Tables) de la console DynamoDB.
2. Sélectionnez la table que vous avez créée.
3. Choisissez Supprimer.
4. Saisissez **`delete`** dans la zone de texte.
5. Choisissez Supprimer la table.

# Traitez les événements EC2 du cycle de vie d'Amazon avec une fonction Lambda

Vous pouvez l'utiliser AWS Lambda pour traiter les événements du cycle de vie depuis Amazon Elastic Compute Cloud et gérer les EC2 ressources Amazon. Amazon EC2 envoie des événements à Amazon EventBridge (CloudWatch Events) pour des événements du cycle de vie, tels que lorsqu'une instance change d'état, lorsqu'un instantané du volume Amazon Elastic Block Store est terminé ou lorsqu'il est prévu de mettre fin à une instance ponctuelle. Vous configurez EventBridge (CloudWatch Events) pour transmettre ces événements à une fonction Lambda pour traitement.

EventBridge (CloudWatch Events) invoque votre fonction Lambda de manière asynchrone avec le document d'événement d'Amazon. EC2

Example événement du cycle de vie d'une instance

```
{
    "version": "0",
    "id": "b6ba298a-7732-2226-xmpl-976312c1a050",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "111122223333",
    "time": "2019-10-02T17:59:30Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ec2:us-east-1:111122223333:instance/i-0c314xmplcd5b8173"
    ],
    "detail": {
        "instance-id": "i-0c314xmplcd5b8173",
        "state": "running"
    }
}
```

Pour plus de détails sur la configuration des événements, consultezInvoquer une fonction Lambda dans un planning. Pour un exemple de fonction qui traite les notifications EBS instantanées Amazon, consultez EventBridge Scheduler for Amazon. EBS

Vous pouvez également utiliser le AWS SDK pour gérer des instances et d'autres ressources avec Amazon EC2API.

# Octroi d'autorisations à EventBridge (CloudWatch événements)

Pour traiter les événements du cycle de vie provenant d'AmazonEC2, EventBridge (CloudWatch Events) a besoin d'une autorisation pour appeler votre fonction. Cette autorisation provient de la stratégie basée sur les ressourcesde la fonction. Si vous utilisez la console EventBridge (CloudWatch Events) pour configurer un déclencheur d'événement, la console met à jour la politique basée sur les ressources en votre nom. Sinon, ajoutez une instruction comme suit :

Example déclaration de politique basée sur les ressources pour les notifications relatives au cycle de vie d'Amazon EC2

```
{
  "Sid": "ec2-events",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:us-east-1:12456789012:function:my-function",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:events:us-east-1:12456789012:rule/*"
    }
  }
}
```

Pour ajouter une instruction, utilisez la `add-permission` AWS CLI commande.

```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
--principal events.amazonaws.com --function-name my-function --source-arn
 'arn:aws:events:us-east-1:12456789012:rule/*'
```

Si votre fonction utilise le AWS SDK pour gérer les EC2 ressources Amazon, ajoutez des EC2 autorisations Amazon au rôle d'exécution de la fonction.

# Traitez les demandes d'Application Load Balancer avec Lambda

Vous pouvez utiliser une fonction Lambda pour traiter les demandes d'un Application Load Balancer. Elastic Load Balancing prend désormais en charge les fonctions Lambda en tant que cibles pour un Application Load Balancer. Utilisez les règles de l'équilibreur de charge pour acheminer les HTTP demandes vers une fonction, en fonction des valeurs du chemin ou de l'en-tête. Traitez la demande et renvoyez une HTTP réponse depuis votre fonction Lambda.

Elastic Load Balancing appelle votre fonction Lambda de façon synchrone avec un événement qui contient le corps et les métadonnées de la demande.

Example Evénement de demande d'Application Load Balancer

```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-
east-1:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
        }
    },
    "httpMethod": "GET",
    "path": "/lambda",
    "queryStringParameters": {
        "query": "1234ABCD"
    },
    "headers": {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8",
        "accept-encoding": "gzip",
        "accept-language": "en-US,en;q=0.9",
        "connection": "keep-alive",
        "host": "lambda-alb-123578498.us-east-1.elb.amazonaws.com",
        "upgrade-insecure-requests": "1",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
        "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
        "x-forwarded-for": "72.12.164.125",
        "x-forwarded-port": "80",
        "x-forwarded-proto": "http",
        "x-imforwards": "20"
    },
    "body": "",
```

```
    "isBase64Encoded": False
}
```

Votre fonction traite l'événement et renvoie un document de réponse à l'équilibreur de charge enJSON. Elastic Load Balancing convertit le document en réponse de HTTP réussite ou d'erreur et le renvoie à l'utilisateur.

Example format du document de réponse

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "isBase64Encoded": False,
    "headers": {
        "Content-Type": "text/html"
    },
    "body": "<h1>Hello from Lambda!</h1>"
}
```

Pour configurer un Application Load Balancer comme déclencheur de fonction, accordez à Elastic Load Balancing l'autorisation d'exécuter la fonction, créez un groupe cible qui achemine les demandes vers la fonction, et ajoutez à l'équilibreur de charge une règle qui envoie les demandes au groupe cible.

Utilisez la commande `add-permission` pour ajouter une instruction d'autorisation à la stratégie basée sur les ressources de votre fonction.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

Vous devriez voir la sortie suivante:

```
{
    "Statement": "{\"Sid\":\"load-balancer\",\"Effect\":\"Allow\",\"Principal\":
{\"Service\":\"elasticloadbalancing.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:alb-function\"}"
}
```

Pour obtenir des instructions sur la configuration de l'écouteur de l'équilibreur de charge d'application et du groupe cible, consultez Fonctions Lambda en tant que cibles dans le Guide de l'utilisateur des Application Load Balancers.

# Invoquer une fonction Lambda dans un planning

Amazon EventBridge Scheduler est un planificateur sans serveur qui vous permet de créer, d'exécuter et de gérer des tâches à partir d'un service géré centralisé. Avec EventBridge Scheduler, vous pouvez créer des plannings à l'aide d'expressions cron et rate pour les modèles récurrents, ou configurer des appels ponctuels. Vous pouvez configurer des fenêtres temporelles flexibles pour la livraison, définir des limites de nouvelles tentatives, ainsi que la durée de rétention maximale pour les événements non traités.

Lorsque vous configurez le EventBridge planificateur avec Lambda, le planificateur appelle votre fonction Lambda de EventBridge manière asynchrone. Cette page explique comment utiliser le EventBridge Scheduler pour appeler une fonction Lambda dans un planning.

## Configurer le rôle d'exécution

Lorsque vous créez un nouveau calendrier, le EventBridge planificateur doit être autorisé à appeler son API opération cible en votre nom. Vous accordez ces autorisations au EventBridge Scheduler à l'aide d'un rôle d'exécution. La politique d'autorisation que vous associez au rôle d'exécution de votre planification définit les autorisations requises. Ces autorisations dépendent de la cible que API vous souhaitez que EventBridge Scheduler appelle.

Lorsque vous utilisez la console du EventBridge planificateur pour créer un calendrier, comme dans la procédure suivante, le EventBridge planificateur définit automatiquement un rôle d'exécution en fonction de la cible que vous avez sélectionnée. Si vous souhaitez créer un calendrier à l'aide de l'un des EventBridge planificateursSDKs, le ou AWS CLI AWS CloudFormation, vous devez disposer d'un rôle d'exécution existant qui accorde les autorisations dont le EventBridge planificateur a besoin pour appeler une cible. Pour plus d'informations sur la configuration manuelle d'un rôle d'exécution pour votre calendrier, consultez la section Configuration d'un rôle d'exécution dans le guide de l'utilisateur du EventBridge planificateur.

## Créer une planification

Pour créer une planification à l'aide de la console

1. Ouvrez la console Amazon EventBridge Scheduler à https://console.aws.amazon.com/scheduler/la maison.
2. Sur la page Planifications, choisissez Créer une planification.

3. Sur la page Spécifier le détail de la planification, dans la section Nom et description de la planification, procédez comme suit :

   a. Pour Nom de la planification, saisissez un nom à attribuer à votre planification. Par exemple, **`MyTestSchedule`**.

   b. (Facultatif) Dans le champ Description, saisissez une description de la planification. Par exemple, **`My first schedule`**.

   c. Pour Groupe de planifications, choisissez un groupe de planifications dans la liste déroulante. Si vous n'avez pas de groupe, choisissez par défaut. Pour créer un groupe de planifications, choisissez Crée votre propre planification.

      Vous utilisez des groupes de planifications pour leur ajouter des balises.

4. • Choisissez vos options de planification.

| Occurrence | Faites ceci... |
| --- | --- |
| Planification ponctuelle<br><br>Une planification ponctuelle n'invoque un objectif qu'une seule fois à la date et à l'heure que vous indiquez. | Pour Date et heure, procédez comme suit :<br>• Entrez une date valide au format `YYYY/MM/DD` .<br>• Entrez un horodatage au format `hh:mm` de 24 heures.<br>• Dans le champ Fuseau horaire, choisissez le fuseau horaire. |
| Planification récurrente<br><br>Une planification récurrente invoque un objectif à un taux que vous spécifiez à l'aide d'une expression cron ou d'une expression rate. | a. Pour Schedule type (Planifier le type), effectuez l'une des étapes suivantes :<br>• Pour utiliser une expression cron afin de définir la planification, choisissez |

| Occurrence | Faites ceci... | |
| --- | --- | --- |
| | Planification basée sur cron et entrez l'expression cron.<br>• Pour utiliser une expression de rythme pour définir la planification, choisissez Planification basée sur le rythme.<br>Pour plus d'informations sur les expressions cron et rate, consultez la section Types de planification sur EventBridge Scheduler dans le guide de l'utilisateur d'Amazon EventBridge Scheduler.<br>b. Pour Fenêtre temporelle flexible, choisissez Désactivé pour désactiver cette option ou choisir l'une des fenêtres temporelles prédéfinies. Par exemple, si vous choisissez15 minutes et que vous définissez une planification récurrente pour invoquer son objectif une fois par heure, la planification s'exécute dans les 15 | |

| Occurrence | Faites ceci... | |
|---|---|---|
| | minutes suivant le début de chaque heure. | |

5. (Facultatif) Si vous avez choisi Planification récurrente à l'étape précédente, dans la section Délai, procédez comme suit :

   a. Dans le champ Fuseau horaire, choisissez un fuseau horaire.

   b. Pour Date et heure de début, entrez une date valide au format `YYYY/MM/DD`, puis spécifiez un horodatage au format `hh:mm` de 24 heures.

   c. Pour Date et heure de fin, entrez une date valide au format `YYYY/MM/DD`, puis spécifiez un horodatage au format `hh:mm` de 24 heures.

6. Choisissez Suivant.

7. Sur la page Sélectionner une cible, choisissez l' AWS APIopération invoquée par le EventBridge planificateur :

   a. Sélectionnez Invoquer AWS Lambda .

   b. Dans la section Invoquer, sélectionnez une fonction ou choisissez Créer une fonction Lambda.

   c. (Facultatif) Saisissez une charge utile JSON. Si vous n'entrez aucune charge utile, le EventBridge planificateur utilise un événement vide pour appeler la fonction.

8. Choisissez Suivant.

9. Sur la page Settings (Paramètres), procédez comme suit :

   a. Pour activer la planification, sous État de la planification, activez Activer la planification.

   b. Pour configurer une politique de nouvelles tentatives pour votre calendrier, sous Politique de nouvelles tentatives et file d'attente de lettres mortes (DLQ), procédez comme suit :

      - Activez Réessayer.
      - Pour Âge maximal de l'événement, entrez le nombre maximum d'heures et de minutes pendant lequel le EventBridge planificateur doit conserver un événement non traité.
      - La durée maximale est 24 heures.
      - Pour Nombre maximum de tentatives, entrez le nombre maximum de fois que le EventBridge planificateur réessaie le calendrier si la cible renvoie une erreur.

La valeur maximale est 185 nouvelles tentatives.

Avec les politiques de nouvelle tentative, si un calendrier ne parvient pas à invoquer sa cible, le EventBridge planificateur le réexécute. Si elle est configurée, vous devez définir la durée de rétention maximale et les nouvelles tentatives pour la planification.

c. Choisissez l'endroit où EventBridge Scheduler stocke les événements non livrés.

| Option file d'attente de lettres mortes () DLQ | Faites ceci... |
| --- | --- |
| Ne stockez pas | Sélectionnez Aucun. |
| Enregistrez l'événement dans le même Compte AWS endroit où vous créez le calendrier | a. Choisissez Select an Amazon SQS queue dans my Compte AWS as DLQ a.<br>b. Choisissez le nom de ressource Amazon (ARN) de la SQS file d'attente Amazon. |
| Stockez l'événement dans un endroit Compte AWS différent de celui dans lequel vous créez le calendrier | a. Choisissez Spécifier une SQS file d'attente Amazon dans un autre en Comptes AWS tant que DLQ.<br>b. Entrez le nom de ressource Amazon (ARN) de la SQS file d'attente Amazon. |

d. Pour utiliser une clé gérée par le client afin de chiffrer votre entrée cible, sous Chiffrement, choisissez Personnaliser les paramètres de chiffrement (avancé).

Si vous choisissez cette option, entrez une KMS clé existante ARN ou choisissez Create an AWS KMS key pour accéder à la AWS KMS console. Pour plus d'informations sur la

manière dont EventBridge Scheduler chiffre vos données au repos, consultez la section Chiffrement au repos dans le guide de l'utilisateur d'Amazon EventBridge Scheduler.

e. Pour que le EventBridge planificateur crée un nouveau rôle d'exécution pour vous, choisissez Créer un nouveau rôle pour ce calendrier. Ensuite, saisissez un nom pour Nom du rôle. Si vous choisissez cette option, le EventBridge planificateur associe au rôle les autorisations requises pour votre cible modélisée.

10. Choisissez Suivant.
11. Sur la page Examiner et créer une planification, examinez les détails de votre planification. Dans chaque section, choisissez Modifier pour revenir à cette étape et modifier ses détails.
12. Choisissez Créer une planification.

    Vous pouvez consulter la liste de vos planifications nouvelles et existantes sur la page Planifications. Sous la colonne État, vérifiez que votre nouvelle planification est activée.

Pour confirmer que EventBridge Scheduler a invoqué la fonction, consultez les journaux Amazon CloudWatch de la fonction.

## Ressources connexes

Pour plus d'informations sur le EventBridge planificateur, consultez les rubriques suivantes :

- EventBridge Guide de l'utilisateur du planificateur
- EventBridge Référence du planificateur API
- EventBridge Tarification du planificateur

# Utilisation AWS Lambda avec AWS IoT

AWS IoT fournit une communication sécurisée entre les appareils connectés à Internet (tels que les capteurs) et le AWS cloud. Vous pouvez ainsi collecter les données de télémétrie de plusieurs périphériques, les stocker et les analyser.

Vous pouvez créer des AWS IoT règles pour que vos appareils interagissent avec les AWS services. Le moteur de AWS IoT règles fournit un langage SQL basé pour sélectionner les données des charges utiles des messages et les envoyer à d'autres services, tels qu'Amazon S3, Amazon DynamoDB AWS Lambda et. Vous définissez une règle pour appeler une fonction Lambda lorsque vous souhaitez appeler un autre AWS service ou un service tiers.

Lorsqu'un message IoT entrant déclenche la règle, AWS IoT appelle votre fonction Lambda de manière asynchrone et transmet les données du message IoT à la fonction.

L'exemple suivant montre une mesure de l'humidité à partir d'un capteur de serre. Les valeurs row et pos identifient l'emplacement du capteur. Cet exemple d'événement est basé sur le type de serre indiqué dans les didacticiels sur les règles AWS IoT.

Example AWS IoT événement de message

```
{
    "row" : "10",
    "pos" : "23",
    "moisture" : "75"
}
```

Pour l'appel asynchrone, Lambda met le message en file d'attente et fait une nouvelle tentative si votre fonction renvoie une erreur. Configurez votre fonction avec une destination pour conserver les événements que votre fonction n'a pas pu traiter.

Vous devez autoriser le AWS IoT service à appeler votre fonction Lambda. Utilisez la commande `add-permission` pour ajouter une instruction d'autorisation à la stratégie basée sur les ressources de votre fonction.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
 iot.amazonaws.com
```

Vous devriez voir la sortie suivante:

```
{
    "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":
{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":
\"arn:aws:lambda:us-east-1:123456789012:function:my-function\"}"
}
```

Pour plus d'informations sur l'utilisation de Lambda avec AWS IoT, consultez Création d'une AWS Lambda règle.

# Comment Lambda traite les enregistrements issus d'Amazon Kinesis Data Streams

Vous pouvez utiliser une fonction Lambda pour traiter les enregistrements d'un flux de données Amazon Kinesis. Vous pouvez associer une fonction Lambda à un consommateur de débit partagé Kinesis Data Streams (itérateur standard) ou à un consommateur de débit dédié doté d'un ventilateur amélioré. Pour les itérateurs standard, Lambda interroge chaque partition de votre flux Kinesis à la recherche d'enregistrements à l'aide du protocole. HTTP Le mappage de source d'événement partage le débit de lecture avec d'autres utilisateurs de la partition.

Pour plus d'informations sur les flux de données Kinesis, consultez Lecture de données à partir d'Amazon Kinesis Data Streams.

> **Note**
>
> Kinesis facture chaque partition et, pour les diffusions améliorées, les données lues à partir du flux. Pour obtenir des informations de tarification, consultez Tarification Amazon Kinesis.

Rubriques


- Flux d'interrogation et de mise en lots
- Exemple d'évènement
- Traitez les enregistrements Amazon Kinesis Data Streams avec Lambda
- Configuration d'une réponse par lots partielle avec Kinesis Data Streams et Lambda
- Conserver les enregistrements de lots supprimés pour une source d'événements Kinesis Data Streams dans Lambda
- Implémentation du traitement dynamique de Kinesis Data Streams dans Lambda
- Paramètres Lambda pour les mappages de sources d'événements Amazon Kinesis Data Streams
- Utilisation du filtrage des événements avec une source d'événements Kinesis
- Tutoriel : Utilisation de Lambda avec Kinesis Data Streams


## Flux d'interrogation et de mise en lots

Lambda lit les enregistrements du flux de données et invoque votre fonction de manière synchrone avec un événement contenant des enregistrements de flux. Lambda lit les registres par lots et

invoque votre fonction pour les traiter. Chaque lot contient des enregistrements provenant d'un seul flux de données/partition.

Pour les flux de données Kinesis standard, Lambda interroge les partitions de votre flux à la recherche d'enregistrements au rythme d'une fois par seconde pour chaque partition. Pour le fan-out amélioré par Kinesis, Lambda utilise HTTP une connexion /2 pour écouter les enregistrements envoyés par Kinesis. Lorsque des enregistrements sont disponibles, Lambda invoque votre fonction et attend le résultat.

Par défaut, Lambda invoque votre fonction dès que des enregistrements sont disponibles. Si le lot que Lambda lit à partir de la source d'événements ne comprend qu'un seul enregistrement, Lambda envoie un seul registre à la fonction. Pour éviter d'invoquer la fonction avec un petit nombre de registres, vous pouvez indiquer à la source d'événement de les mettre en mémoire tampon pendant 5 minutes en configurant une fenêtre de traitement par lots. Avant d'invoquer la fonction, Lambda continue de lire les registres de la source d'événements jusqu'à ce qu'il ait rassemblé un lot complet, que la fenêtre de traitement par lot expire ou que le lot atteigne la limite de charge utile de 6 Mo. Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Lambda n'attend pas que les extensions configurées soient terminées avant d'envoyer le lot suivant pour traitement. En d'autres termes, vos extensions peuvent continuer à s'exécuter pendant que Lambda traite le prochain lot d'enregistrements. Cela peut causer des problèmes de limitation si vous enfreignez l'un des paramètres ou l'une des limites de simultanéité de votre compte. Pour détecter s'il s'agit d'un problème éventuel, surveillez vos fonctions et vérifiez si vous observez des métriques de simultanéité plus élevées que prévu pour votre mappage des sources d'événements. En raison de la brièveté des intervalles entre les invocations, Lambda peut brièvement signaler une utilisation simultanée supérieure au nombre de partitions. Cela peut être vrai même pour les fonctions Lambda sans extensions.

Configurez le ParallelizationFactorparamètre pour traiter une partition d'un flux de données Kinesis avec plusieurs invocations Lambda simultanément. Vous pouvez spécifier le nombre de lots simultanés que Lambda interroge à partir d'une partition via un facteur de parallélisation de 1 (par défaut) à 10. Par exemple, lorsque vous définissez la `ParallelizationFactor` valeur 2, vous pouvez avoir 200 appels Lambda simultanés au maximum pour traiter 100 fragments de données Kinesis (bien qu'en pratique, vous puissiez voir des valeurs différentes pour la métrique). `ConcurrentExecutions` Cela permet d'augmenter le débit de traitement quand le volume de données est volatil et que la valeur du paramètre `IteratorAge` est élevée. Lorsque vous augmentez le nombre de lots simultanés par partition, Lambda garantit toujours un traitement dans l'ordre au niveau de la clé de partition.

Vous pouvez également utiliser l'`ParallelizationFactor`agrégation Kinesis. Le comportement du mappage des sources d'événements varie selon que vous utilisez ou non un ventilateur amélioré :

- Sans ventilation améliorée : tous les événements d'un événement agrégé doivent avoir la même clé de partition. La clé de partition doit également correspondre à celle de l'événement agrégé. Si les événements contenus dans l'événement agrégé ont des clés de partition différentes, Lambda ne peut garantir le traitement ordonné des événements par clé de partition.
- Avec un fan-out amélioré : Lambda décode d'abord l'événement agrégé en événements individuels. L'événement agrégé peut avoir une clé de partition différente de celle des événements qu'il contient. Cependant, les événements qui ne correspondent pas à la clé de partition sont supprimés et perdus. Lambda ne traite pas ces événements et ne les envoie pas vers une destination de défaillance configurée.

## Exemple d'évènement

Example

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1545084650.987
            },
```

```
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
        },
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692540925702759324208523137515618",
                "data": "VGhpcyBpcyBvbmx5IGEgdGVzdC4=",
                "approximateArrivalTimestamp": 1545084711.166
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
        }
    ]
}
```

# Traitez les enregistrements Amazon Kinesis Data Streams avec Lambda

Pour traiter les enregistrements Amazon Kinesis Data Streams avec Lambda, créez un consommateur pour votre flux, puis créez un mappage des sources d'événements Lambda.

## Configuration de votre fonction et de votre flux de données

Votre fonction Lambda est une application consommateur pour votre flux de données. Elle traite un lot d’enregistrements à la fois à partir de chaque partition. Vous pouvez mapper une fonction Lambda à un consommateur à débit partagé (itérateur standard) ou à un consommateur à débit dédié avec diffusion améliorée.

- Itérateur standard : Lambda interroge chaque partition de votre flux Kinesis à la recherche d'enregistrements à une fréquence de base d'une fois par seconde. Lorsque d'autres enregistrements sont disponibles, Lambda continue de traiter les lots jusqu'à ce que la fonction rattrape le flux. Le mappage de source d'événement partage le débit de lecture avec d'autres utilisateurs de la partition.
- Ventilateur amélioré : pour minimiser la latence et optimiser le débit de lecture, créez un consommateur de flux de données avec un ventilateur amélioré. Les consommateurs avec diffusion améliorée obtiennent une connexion dédiée pour chaque partition qui n'a pas d'impact sur les autres applications lisant sur le flux. Les consommateurs de flux utilisent HTTP/2 afin de réduire la latence en transférant les enregistrements à Lambda via une connexion longue durée et en comprimant les en-têtes de requête. Vous pouvez créer un consommateur de flux à l'aide de l'API Kinesis RegisterStreamConsumer.

```
aws kinesis register-stream-consumer \
--consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Vous devriez voir la sortie suivante :

```
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

Pour augmenter la vitesse à laquelle votre fonction traite les enregistrements, ajoutez des fragments à votre flux de données. Lambda traite les enregistrements de chaque partition dans l'ordre. Il arrête de traiter les enregistrements supplémentaires d'une partition si votre fonction renvoie une erreur. Plus de partitions signifient plus de lots traités en une seule fois, ce qui réduit l'impact des erreurs sur la simultanéité.

Si votre fonction ne peut pas augmenter sa capacité pour traiter le nombre total de lots simultanés, demandez une augmentation de quota ou réservez de la simultanéité pour votre fonction.

## Créez un mappage de source d'événements pour appeler une fonction Lambda

Pour appeler votre fonction Lambda avec les enregistrements de votre flux de données, créez un mappage des sources d'événements. Vous pouvez créer plusieurs mappages de source d'événement pour traiter les mêmes données avec plusieurs fonctions Lambda, ou pour traiter des éléments en provenance de plusieurs flux de données avec une seule fonction. Lors du traitement d'éléments provenant de plusieurs flux, chaque lot contient des enregistrements provenant d'une seule partition ou d'un seul flux.

Vous pouvez configurer des mappages de sources d'événements pour traiter les enregistrements d'un flux dans un autre Compte AWS. Pour en savoir plus, veuillez consulter la section the section called “Mappages entre comptes”.

Avant de créer un mappage de source d'événements, vous devez autoriser votre fonction Lambda à lire un flux de données Kinesis. Lambda a besoin des autorisations suivantes pour gérer les ressources liées à votre flux de données Kinesis :

- kinésie : DescribeStream
- kinesis : Résumé DescribeStream
- kinésie : GetRecords
- kinesis : Itérateur GetShard
- kinésie : ListShards
- kinésie : ListStreams
- kinésie : Shard SubscribeTo

La politique AWS gérée AWSLambdaKinesisExecutionRoleinclut ces autorisations. Ajoutez cette politique gérée à votre fonction comme décrit dans la procédure suivante.

AWS Management Console

Pour ajouter des autorisations Kinesis à votre fonction

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Dans l'onglet Configuration, sélectionnez Autorisations.
3. Dans le volet Rôle d'exécution, sous Nom du rôle, choisissez le lien vers le rôle d'exécution de votre fonction. Ce lien ouvre la page correspondant à ce rôle dans la console IAM.

4. Dans le volet Politiques d'autorisations, choisissez Ajouter des autorisations, puis sélectionnez Joindre des politiques.
5. Dans le champ de recherche, entrez **`AWSLambdaKinesisExecutionRole`**.
6. Cochez la case à côté de la politique et choisissez Ajouter une autorisation.

AWS CLI

Pour ajouter des autorisations Kinesis à votre fonction

- Exécutez la commande CLI suivante pour ajouter la `AWSLambdaKinesisExecutionRole` politique au rôle d'exécution de votre fonction :

```
aws iam attach-role-policy \
--role-name MyFunctionRole \
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

AWS SAM

Pour ajouter des autorisations Kinesis à votre fonction

- Dans la définition de votre fonction, ajoutez la `Policies` propriété comme indiqué dans l'exemple suivant :

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.x
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Après avoir configuré les autorisations requises, créez le mappage des sources d'événements.

AWS Management Console

Pour créer le mappage des sources d'événements Kinesis

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction.
2. Dans le volet de Présentation de la fonction, choisissez Ajouter un déclencheur.
3. Sous Configuration du déclencheur, pour la source, sélectionnez Kinesis.
4. Sélectionnez le flux Kinesis pour lequel vous souhaitez créer le mappage des sources d'événements et, éventuellement, un consommateur de votre flux.
5. (Facultatif) modifiez la taille du lot, la position de départ et la fenêtre Batch pour le mappage de la source de votre événement.
6. Choisissez Ajouter.

Lorsque vous créez le mappage de vos sources d'événements depuis la console, votre rôle IAM doit disposer des autorisations kinesis : ListStreams et kinesis : ListStream Consumers.

AWS CLI

Pour créer le mappage des sources d'événements Kinesis

- Exécutez la commande CLI suivante pour créer un mappage des sources d'événements Kinesis. Choisissez votre propre taille de lot et votre position de départ en fonction de votre cas d'utilisation.

```
aws lambda create-event-source-mapping \
--function-name MyFunction \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \
--starting-position LATEST \
--batch-size 100
```

Pour spécifier une fenêtre de traitement par lots, ajoutez l'`--maximum-batching-window-in-seconds`option. Pour plus d'informations sur l'utilisation de ce paramètre et d'autres paramètres, consultez la section create-event-source-mapping dans le manuel Command Reference.AWS CLI

AWS SAM

Pour créer le mappage des sources d'événements Kinesis

- Dans la définition de votre fonction, ajoutez la `KinesisEvent` propriété comme indiqué dans l'exemple suivant :

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.x
      Policies:
        - AWSLambdaKinesisExecutionRole
      Events:
        KinesisEvent:
          Type: Kinesis
          Properties:
            Stream: !GetAtt MyKinesisStream.Arn
            StartingPosition: LATEST
            BatchSize: 100

  MyKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
```

Pour en savoir plus sur la création d'un mappage des sources d'événements pour Kinesis Data Streams AWS SAM in, consultez Kinesis dans AWS Serverless Application Model le manuel du développeur.

## Position de départ du sondage et du stream

Sachez que l’interrogation des flux lors des mises à jour et de la création du mappage des sources d’événements est finalement cohérente.

- Lors de la création du mappage des sources d’événements, le démarrage de l’interrogation des événements depuis le flux peut prendre plusieurs minutes.

- Lors des mises à jour du mappage des sources d'événements, l'arrêt et le redémarrage de l'interrogation des événements depuis le flux peuvent prendre plusieurs minutes.

Ce comportement signifie que si vous spécifiez `LATEST` comme position de départ du flux, le mappage des sources d'événements peut manquer des événements lors de la création ou des mises à jour. Pour vous assurer de ne manquer aucun événement, spécifiez la position de départ du flux comme `TRIM_HORIZON` ou `AT_TIMESTAMP`.

### Création d'un mappage des sources d'événements entre comptes

Amazon Kinesis Data Streams prend en charge les politiques basées sur les ressources. De ce fait, vous pouvez traiter les données ingérées dans un flux à l' Compte AWS aide d'une fonction Lambda dans un autre compte.

Pour créer un mappage de source d'événements pour votre fonction Lambda à l'aide d'un flux Kinesis dans un autre Compte AWS, vous devez configurer le flux à l'aide d'une politique basée sur les ressources afin d'autoriser votre fonction Lambda à lire des éléments. Pour savoir comment configurer votre stream afin d'autoriser l'accès entre comptes, consultez la section Partage de l'accès avec des AWS Lambda fonctions multicomptes dans le guide du développeur Amazon Kinesis Streams.

Une fois que vous avez configuré votre flux avec une politique basée sur les ressources qui donne à votre fonction Lambda les autorisations requises, créez le mappage des sources d'événements à l'aide de l'une des méthodes décrites dans la section précédente.

Si vous choisissez de créer le mappage de votre source d'événements à l'aide de la console Lambda, collez l'ARN de votre flux directement dans le champ de saisie. Si vous souhaitez spécifier un consommateur pour votre flux, le fait de coller l'ARN du consommateur remplit automatiquement le champ du flux.

## Configuration d'une réponse par lots partielle avec Kinesis Data Streams et Lambda

Lors de l'utilisation et du traitement de données de streaming à partir d'une source d'événement, par défaut, Lambda n'effectue un point de contrôle sur le numéro de séquence le plus élevé d'un lot que lorsque celui-ci est un succès complet. Lambda traite tous les autres résultats comme un échec complet et recommence à traiter le lot jusqu'à atteindre la limite de nouvelles tentatives. Pour autoriser des succès partiels lors du traitement des lots à partir d'un flux, activez `ReportBatchItemFailures`. Autoriser des succès partiels peut permettre de réduire le nombre

de nouvelles tentatives sur un enregistrement, mais cela n'empêche pas entièrement la possibilité de nouvelles tentatives dans un enregistrement réussi.

Pour l'activer`ReportBatchItemFailures`, incluez la valeur enum **`ReportBatchItemFailures`** dans la FunctionResponseTypesliste. Cette liste indique quels types de réponse sont activés pour votre fonction. Vous pouvez configurer cette liste lorsque vous créez ou mettez à jour un mappage de source d'événements.

## Syntaxe du rapport

Lors de la configuration des rapports d'échec d'articles de lot, la classe `StreamsEventResponse` est renvoyée avec une liste d'échecs d'articles de lot. Vous pouvez utiliser un objet `StreamsEventResponse` pour renvoyer le numéro de séquence du premier enregistrement ayant échoué dans le lot. Vous pouvez également créer votre classe personnalisée en utilisant la syntaxe de réponse correcte. La JSON structure suivante indique la syntaxe de réponse requise :

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Si le tableau `batchItemFailures` contient plusieurs éléments, Lambda utilise l'enregistrement portant le numéro de séquence le plus bas comme point de contrôle. Lambda réessaie ensuite tous les enregistrements à partir de ce point de contrôle.

## Conditions de réussite et d'échec

Lambda traite un lot comme un succès complet si vous renvoyez l'un des éléments suivants :

- Une liste `batchItemFailure` vide
- Une liste `batchItemFailure` nulle
- Une `EventResponse` vide
- Un nu `EventResponse`

Lambda traite un lot comme un échec complet si vous renvoyez l'un des éléments suivants :

- Une chaîne vid `itemIdentifier`
- Un `itemIdentifier` nul
- Un `itemIdentifier` avec un nom de clé incorrect

Lambda effectue des nouvelles tentatives en cas d'échec en fonction de votre stratégie de nouvelle tentative.

## Diviser un lot

Si votre invocation échoue et que `BisectBatchOnFunctionError` est activé, le lot est divisé en deux quel que soit votre paramètre `ReportBatchItemFailures`.

Quand une réponse de succès partiel de lot est reçue et que les paramètres `BisectBatchOnFunctionError` et `ReportBatchItemFailures` sont activés, le lot est divisé au numéro de séquence renvoyé, et Lambda n'effectue de nouvelle tentative que sur les enregistrements restants.

Voici quelques exemples de code de fonction qui renvoie la liste des messages ayant échoué IDs dans le lot :

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
```

```
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
```

```
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
```

```
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
```

```
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
```

```
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end
```

```
def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
```

```
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
```

```
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## Conserver les enregistrements de lots supprimés pour une source d'événements Kinesis Data Streams dans Lambda

La gestion des erreurs pour les mappages de sources d'événements Kinesis dépend du fait que l'erreur se produit avant que la fonction ne soit invoquée ou pendant l'invocation de la fonction :

- Avant l'appel : si un mappage de source d'événement Lambda ne parvient pas à appeler la fonction en raison d'un ralentissement ou d'autres problèmes, il réessaie jusqu'à ce que les enregistrements expirent ou dépassent l'âge maximum configuré sur le mappage de source d'événement (). MaximumRecordAgeInSeconds
- Pendant l'appel : si la fonction est invoquée mais renvoie une erreur, Lambda réessaie jusqu'à ce que les enregistrements expirent, dépassent l'âge maximum MaximumRecordAgeInSeconds() ou atteignent le quota de tentatives configuré (). MaximumRetryAttempts Pour les erreurs de fonctionnement, vous pouvez également configurer BisectBatchOnFunctionError, ce qui divise un lot défaillant en deux lots plus petits, isolant ainsi les mauvais enregistrements et évitant les délais d'attente. Le fractionnement des lots ne consomme pas le quota de nouvelles tentatives.

Si les mesures de gestion des erreurs échouent, Lambda ignore les enregistrements et poursuit le traitement des lots du flux. Avec les paramètres par défaut, cela signifie qu'un enregistrement défectueux peut bloquer le traitement sur la partition affectée pendant jusqu'à une semaine. Pour éviter cela, configurez le mappage de source d'événement de votre fonction avec un nombre raisonnable de nouvelles tentatives et un âge maximum d'enregistrement correspondant à votre cas d'utilisation.

## Configuration des destinations pour les appels échoués

Pour retenir les enregistrements des invocations de mappage de sources d'événements qui ont échoué, ajoutez une destination au mappage des sources d'événements de votre fonction. Chaque enregistrement envoyé à la destination est un JSON document contenant des métadonnées relatives à l'échec de l'invocation. Vous pouvez configurer n'importe quel SNS sujet Amazon ou SQS file d'attente Amazon comme destination. Votre rôle d'exécution doit disposer d'autorisations pour la destination :

- Pour les SQS destinations : sqs : SendMessage
- Pour les SNS destinations : SNS:Publish

Pour configurer une destination en cas de panne à l'aide de la console, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Invocation du mappage des sources d'événements.
5. Pour le mappage des sources d'événements, choisissez une source d'événements configurée pour cette fonction.
6. Pour Condition, sélectionnez En cas d'échec. Pour les invocations de mappage des sources d'événements, il s'agit de la seule condition acceptée.
7. Pour Type de destination, choisissez le type de destination auquel Lambda envoie les enregistrements d'invocation.
8. Pour Destination, choisissez une ressource.
9. Choisissez Save (Enregistrer).

Vous pouvez également configurer une destination en cas de panne à l'aide de AWS Command Line Interface (AWS CLI). Par exemple, la create-event-source-mappingcommande suivante ajoute un mappage de source d'événement avec une destination SQS en cas de défaillance à `MyFunction` :

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \
```

```
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

La update-event-source-mappingcommande suivante met à jour le mappage d'une source d'événements afin d'envoyer les enregistrements d'invocation ayant échoué vers une SNS destination après deux tentatives, ou si les enregistrements datent de plus d'une heure.

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 \
--maximum-record-age-in-seconds 3600 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-
east-1:123456789012:dest-topic"}}'
```

Les paramètres mis à jour sont appliqués de façon asynchrone et ne sont pas reflétés dans la sortie tant que le processus n'est pas terminé. Utilisez la get-event-source-mappingcommande pour afficher l'état actuel.

Pour supprimer une destination, entrez une chaîne vide comme argument du paramètre `destination-config` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

L'exemple suivant montre ce que Lambda envoie à une SQS file d'attente ou à un SNS sujet en cas d'échec de l'appel d'une source d'événement Kinesis. Lambda n'envoyant que les métadonnées pour ces types de destination, utilisez les `endSequenceNumber` champs `streamArnshardId`,`startSequenceNumber`, et pour obtenir l'enregistrement original complet. Tous les champs affichés dans la `KinesisBatchInfo` propriété seront toujours présents.

```
{
    "requestContext": {
        "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
```

```
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KinesisBatchInfo": {
        "shardId": "shardId-000000000001",
        "startSequenceNumber":
 "49601189658422359378836298521827638475320189012309704722",
        "endSequenceNumber":
 "49601189658422359378836298522902373528957594348623495186",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",
        "batchSize": 500,
        "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"
    }
}
```

Vous pouvez utiliser ces informations pour récupérer les enregistrements concernés à partir du flux à des fins de résolution de problèmes. Les enregistrements réels n'étant pas inclus, vous devez les récupérer du flux avant qu'ils expirent et soient perdus.

## Implémentation du traitement dynamique de Kinesis Data Streams dans Lambda

Les fonctions Lambda peuvent exécuter des applications de traitement de flux continu. Un flux représente des données illimitées qui circulent en continu dans votre application. Pour analyser les informations provenant de cette entrée de mise à jour continue, vous pouvez lier les enregistrements inclus à l'aide d'une fenêtre de temps définie.

Les fenêtres bascules sont des fenêtres temporelles distinctes qui s'ouvrent et se ferment à intervalles réguliers. Par défaut, les invocations Lambda sont sans état : vous ne pouvez pas les utiliser pour traiter des données sur plusieurs invocations continues sans base de données externe. Cependant, avec les fenêtres bascules, vous pouvez maintenir votre état au long des invocations. Cet état contient le résultat global des messages précédemment traités pour la fenêtre actuelle. Votre état peut être d'un maximum de 1 Mo par partition. S'il dépasse cette taille, Lambda met fin précocement à la fenêtre de traitement.

Chaque enregistrement d'un flux appartient à une fenêtre spécifique. La fonction Lambda traitera chaque enregistrement au moins une fois. Toutefois, elle ne garantit pas un seul traitement

pour chaque enregistrement. Dans de rares cas, tels que pour la gestion des erreurs, certains enregistrements peuvent être sujet à de multiples traitements. Les dossiers sont toujours traités dans l'ordre dès la première fois. Si les enregistrements sont traités plusieurs fois, ils peuvent être traités dans le désordre.

## Regroupement et traitement

Votre fonction gérée par l'utilisateur est invoquée tant pour l'agrégation que pour le traitement des résultats finaux de celle-ci. Lambda regroupe tous les enregistrements reçus dans la fenêtre. Vous pouvez recevoir ces enregistrements en plusieurs lots, chacun sous forme d'invocation séparée. Chaque invocation reçoit un état. Ainsi, lorsque vous utilisez des fenêtres bascules, votre réponse de fonction Lambda doit contenir une propriété `state`. Si la réponse ne contient pas de propriété `state`, Lambda considère qu'il s'agit d'une invocation ayant échoué. Pour satisfaire à cette condition, votre fonction peut renvoyer un objet `TimeWindowEventResponse` ayant la forme JSON suivante :

Example **`TimeWindowEventResponse`** values

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Pour les fonctions Java, nous vous recommandons d'utiliser `Map<String, String>` pour représenter l'état.

À la fin de la fenêtre, l'indicateur `isFinalInvokeForWindow` est défini sur `true` pour indiquer qu'il s'agit de l'état final et qu'il est prêt pour le traitement. Après le traitement, la fenêtre et votre invocation final se terminent, puis l'état est supprimé.

À la fin de votre fenêtre, Lambda applique un traitement final pour les actions sur les résultats de l'agrégation. Votre traitement final est invoqué de manière synchrone. Une fois l'invocation réussie, votre fonction contrôle le numéro de séquence et le traitement du flux continue. Si l'invocation échoue, votre fonction Lambda suspend le traitement ultérieur jusqu'à ce que l'invocation soit réussie.

Example KinesisTimeWindowEvent

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1607497475.000
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",
            "awsRegion": "us-east-1",
            "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-
stream"
        }
    ],
    "window": {
        "start": "2020-12-09T07:04:00Z",
        "end": "2020-12-09T07:06:00Z"
    },
    "state": {
        "1": 282,
        "2": 715
    },
    "shardId": "shardId-000000000006",
    "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream",
    "isFinalInvokeForWindow": false,
    "isWindowTerminatedEarly": false
}
```

## Configuration

Vous pouvez configurer des fenêtres bascule lorsque vous créez ou mettez à jour un mappage de source d'événement. Pour configurer une fenêtre variable, spécifiez la fenêtre en secondes

(TumblingWindowInSeconds). L'exemple de commande suivant AWS Command Line Interface (AWS CLI) crée un mappage des sources d'événements de streaming dont la fenêtre de basculement est de 120 secondes. La fonction Lambda définie pour l'agrégation et le traitement est nommée `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream \
--function-name tumbling-window-example-function \
--starting-position TRIM_HORIZON \
--tumbling-window-in-seconds 120
```

Lambda détermine les limites des fenêtres bascule en fonction de l'heure à laquelle les enregistrements ont été insérés dans le flux. Tous les enregistrements ont un horodatage approximatif disponible que Lambda utilise pour déterminer des limites.

Les agrégations de fenêtres bascule ne prennent pas en charge le repartitionnement. Lorsqu'une partition se termine, Lambda considère que la fenêtre actuelle est fermée, et toute partition enfant ouvre sa propre fenêtre dans un nouvel état. Lorsqu'aucun nouvel enregistrement n'est ajouté à la fenêtre actuelle, Lambda attend jusqu'à 2 minutes avant de supposer que la fenêtre est terminée. Cela permet de garantir que la fonction lit tous les enregistrements de la fenêtre actuelle, même si les enregistrements sont ajoutés par intermittence.

Les fenêtres bascule prennent complètement en charge les stratégies de nouvelle tentative existantes `maxRetryAttempts` et `maxRecordAge`.

Example Handler.py – Agrégation et traitement

La fonction Python suivante montre comment regrouper et traiter votre état final :

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

#Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
    else:
        print('Aggregate invoke')

#Check for early terminations
    if event['isWindowTerminatedEarly']:
```

```
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['kinesis']['partitionKey']] = state.get(record['kinesis']
['partitionKey'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

## Paramètres Lambda pour les mappages de sources d'événements Amazon Kinesis Data Streams

Tous les mappages de sources d'événements Lambda partagent les mêmes CreateEventSourceMappingopérations. UpdateEventSourceMappingAPI Cependant, seuls certains paramètres s'appliquent à Kinesis.

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| BatchSize | N | 100 | Maximum : 10 000. |
| BisectBatchOnFunctionError | N | FAUX | none |
| DestinationConfig | N | N/A | SQSFile d'attente Amazon ou destination de SNS rubrique Amazon pour les enregistrements supprimés. Pour de plus amples informations, veuillez consulter Configuration des destinations pour les appels échoués. |
| Enabled | N | VRAI | none |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| EventSourceArn | Y | N/A | ARNdu flux de données ou d'un consommateur de flux |
| FunctionName | Y | N/A | none |
| FunctionResponseTypes | N | N/A | Pour permettre à votre fonction de signaler des échecs spécifiques dans un lot, incluez la valeur `ReportBatchItemFailures` dans `FunctionResponseTypes` . Pour de plus amples informations, veuillez consulter Configuration d'une réponse par lots partielle avec Kinesis Data Streams et Lambda. |
| MaximumBatchingWindowInSeconds | N | 0 | none |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| MaximumRecordAgeInSeconds | N | -1 | -1 signifie infini : Lambda ne supprime pas les enregistrements (les paramètres de conservation des données de Kinesis Data Streams s'appliquent toujours)<br><br>Minimum : -1<br><br>Maximum : 604 800 |
| MaximumRetryAttempts | N | -1 | -1 signifie infini : les registres qui ont échoué sont réessayés jusqu'à ce que le registre expire.<br><br>Minimum : -1<br><br>Maximum : 10 000. |
| ParallelizationFactor | N | 1 | Maximum : 10 |
| StartingPosition | Y | N/A | AT_TIMESTAMP, TRIM _HORIZON, ou LATEST |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| StartingPositionTimestamp | N | N/A | Valable uniquement s'il StartingPosition est défini sur AT_TIMESTAMP. L'heure à partir de laquelle commencer la lecture, en secondes au format horaire Unix. |
| TumblingWindowInSeconds | N | N/A | Minimum : 0<br><br>Maximum : 900 |

## Utilisation du filtrage des événements avec une source d'événements Kinesis

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources d'événements Kinesis.

Rubriques

- Principes de base du filtrage des événements Kinesis
- Filtrage des enregistrements agrégés de Kinesis

### Principes de base du filtrage des événements Kinesis

Supposons qu'un producteur insère des données JSON formatées dans votre flux de données Kinesis. Un exemple d'enregistrement ressemblerait à ce qui suit, avec les JSON données converties en une chaîne codée Base64 dans le `data` champ.

```
{
    "kinesis": {
```

```
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
        "data":
 "eyJSZWNvcmROdW1iZXIiOiAiMDAwMSIsICJUaW1lU3RhbXAiOiAieXl5eS1tbS1kZFRoaDptbTpzcyIsICJSZXF1ZXN0Q
        "approximateArrivalTimestamp": 1545084650.987
        },
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
}
```

Tant que les données que le producteur introduit dans le flux sont validesJSON, vous pouvez utiliser le filtrage des événements pour filtrer les enregistrements à l'aide de la `data` clé. Supposons qu'un producteur insère des enregistrements dans votre flux Kinesis au format suivantJSON.

```
{
    "record": 12345,
    "order": {
        "type": "buy",
        "stock": "ANYCO",
        "quantity": 1000
        }
}
```

Pour filtrer uniquement les enregistrements dont le type d'ordre est « buy », l'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "data": {
        "order": {
            "type": [ "buy" ]
            }
      }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/my-stream \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Pour filtrer correctement les événements provenant des sources Kinesis, le champ de données et vos critères de filtrage pour le champ de données doivent être au format valideJSON. Si l'un des champs n'est pas dans un JSON format valide, Lambda supprime le message ou lance une exception. Le tableau suivant résume le comportement spécifique :

| Format des données entrantes | Pas de modèle de filtre pour les propriétés des données | Action obtenue. |
|---|---|---|
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Non- JSON | Lambda lance une exception au moment de la création ou de la mise à jour du mappage de sources d'événements. Le modèle de filtre pour les propriétés des données doit être dans un JSON format valide. |
| Non- JSON | Valide JSON | Lambda rejette l'enregis trement. |
| Non- JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |

| Format des données entrantes | Pas de modèle de filtre pour les propriétés des données | Action obtenue. |
| --- | --- | --- |
| Non- JSON | Non- JSON | Lambda lance une exception au moment de la création ou de la mise à jour du mappage de sources d'événements. Le modèle de filtre pour les propriétés des données doit être dans un JSON format valide. |

### Filtrage des enregistrements agrégés de Kinesis

Avec Kinesis, vous pouvez agréger plusieurs enregistrements en un seul enregistrement Kinesis Data Streams pour augmenter le débit de vos données. Lambda ne peut appliquer des critères de filtrage aux enregistrements agrégés que lorsque vous utilisez la distribution ramifiée améliorée de Kinesis. Le filtrage des enregistrements agrégés avec Kinesis standard n'est pas pris en charge. Lorsque vous utilisez la distribution ramifiée améliorée, vous configurez un consommateur Kinesis à débit dédié pour qu'il serve de déclencheur à votre fonction Lambda. Lambda filtre alors les enregistrements agrégés et ne transmet que les enregistrements qui répondent à vos critères de filtrage.

Pour en savoir plus sur l'agrégation des enregistrements Kinesis, reportez-vous à la section Agrégation de la page Concepts clés de la Kinesis Producer Library (KPL). Pour en savoir plus sur l'utilisation de Lambda avec la fonction de ventilation améliorée de Kinesis, consultez la section Augmenter les performances de traitement des flux en temps réel grâce à la fonction de ventilation améliorée Amazon Kinesis Data Streams et à Lambda sur le blog consacré au calcul. AWS AWS

## Tutoriel : Utilisation de Lambda avec Kinesis Data Streams

Dans ce didacticiel, vous allez créer une fonction Lambda pour utiliser les événements d'un flux de données Amazon Kinesis.

1. L'application personnalisée écrit les enregistrements dans le flux.
2. AWS Lambda interroge le flux et, lorsqu'il détecte de nouveaux enregistrements dans le flux, invoque votre fonction Lambda.

3. AWS Lambda exécute la fonction Lambda en assumant le rôle d'exécution que vous avez spécifié lors de la création de la fonction Lambda.

## Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. CLILes exemples de commandes présentés dans ce guide utilisent le formatage Linux. Les commandes qui incluent JSON des documents intégrés doivent être reformatées si vous utilisez Windows. CLI

## Créer le rôle d'exécution

Créez le rôle d'exécution qui autorise votre fonction à accéder aux AWS ressources.

Pour créer un rôle d'exécution

1. Ouvrez la page des rôles dans la IAM console.
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :
   - Entité de confiance – AWS Lambda.
   - Autorisations — AWSLambdaKinesisExecutionRole.
   - Nom de rôle – **`lambda-kinesis-role`**.

La AWSLambdaKinesisExecutionRolepolitique dispose des autorisations dont la fonction a besoin pour lire des éléments provenant de Kinesis et écrire des journaux dans Logs. CloudWatch

## Créer la fonction

Créez une fonction Lambda qui traite vos messages Kinesis. Le code de fonction enregistre l'ID d'événement et les données d'événement de l'enregistrement Kinesis dans Logs. CloudWatch

Ce didacticiel utilise l'exécution Node.js 18.x, mais nous avons également fourni des exemples de code dans d'autres langages d'exécution. Vous pouvez sélectionner l'onglet dans la zone suivante pour voir le code de l'exécution qui vous intéresse. Le JavaScript code que vous allez utiliser dans cette étape se trouve dans le premier exemple présenté dans l'JavaScriptonglet.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
```

```
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }
```

```
    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
```

```
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
```

```
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
```

```
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Utilisation d'un événement Kinesis avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
```

```
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Utilisation d'un événement Kinesis avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';
```

```
class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
```

```
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un événement Kinesis avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour créer la fonction

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir kinesis-tutorial
cd kinesis-tutorial
```

2. Copiez l'exemple de JavaScript code dans un nouveau fichier nomméindex.js.
3. Créez un package de déploiement.

```
zip function.zip index.js
```

4. Créez une fonction Lambda à l'aide de la commande create-function.

```
aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-kinesis-role
```

## Test de la fonction Lambda

Appelez votre fonction Lambda manuellement à l'aide de la invoke AWS Lambda CLI commande et d'un exemple d'événement Kinesis.

Pour tester la fonction Lambda

1. Copiez ce qui suit JSON dans un fichier et enregistrez-le sousinput.txt.

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1545084650.987
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
```

```
            "invokeIdentityArn": "arn:aws:iam::111122223333:role/lambda-kinesis-
role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:111122223333:stream/
lambda-stream"
        }
    ]
}
```

2. Utilisez la commande `invoke` pour envoyer l'événement à la fonction.

```
aws lambda invoke --function-name ProcessKinesisRecords \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

La réponse est enregistrée dans `out.txt`.

## Créez un flux Kinesis

Pour créer un flux, utilisez la commande `create-stream` .

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Exécutez la `describe-stream` commande suivante pour obtenir le fluxARN.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Vous devriez voir la sortie suivante :

```
{
    "StreamDescription": {
        "Shards": [
            {
```

```
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "340282366920746074317682119384634633455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "49591073947768692513481539594623130411957558361251844610"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream",
        "StreamName": "lambda-stream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1544828156.0
    }
}
```

ARNÀ l'étape suivante, vous utiliserez le flux pour l'associer à votre fonction Lambda.

## Ajouter une source d’événement dans AWS Lambda

Exécutez la commande suivante AWS CLI `add-event-source`.

```
aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \
--event-source  arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream \
--batch-size 100 --starting-position LATEST
```

Notez l’ID de mappage pour une utilisation ultérieure. Pour obtenir une liste des mappages de source d’événement, exécutez la commande suivante `list-event-source-mappings`.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

Dans la réponse, vous pouvez vérifier que la valeur d'état indique `enabled`. Les mappages de source d'événement peuvent être désactivés pour suspendre temporairement l'interrogation, ce qui entraîne la perte d'enregistrements.

## Tester la configuration

Pour tester le mappage de source d'événement, ajoutez des enregistrements d'événements à votre flux Kinesis. La `--data` valeur est une chaîne CLI encodée en base64 avant de l'envoyer à Kinesis. Vous pouvez exécuter la même commande plus d'une fois pour ajouter plusieurs enregistrements dans le flux.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

Lambda utilise le rôle d'exécution pour lire les enregistrements du flux. Ensuite, il invoque votre fonction Lambda en transmettant des lots d'enregistrements. La fonction décode les données de chaque enregistrement et les enregistre, en envoyant le résultat à CloudWatch Logs. Affichez les journaux dans la console CloudWatch .

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le flux Kinesis

1. Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à https://console.aws.amazon.com l'adresse /kinesis.
2. Sélectionnez le flux que vous avez créé.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans le champ de saisie de texte.
5. Sélectionnez Supprimer.

# Utilisation de Lambda avec Amazon MQ

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Amazon MQ est un service d'agent de messages géré pour Apache ActiveMQ et RabbitMQ. Un agent de messages permet à des applications et composants logiciels de communiquer à l'aide de divers langages de programmation, systèmes d'exploitation et autres protocoles de messagerie formels par le biais de rubriques ou d'événements en file d'attente.

Amazon MQ peut également gérer des instances Amazon Elastic Compute Cloud EC2 (Amazon) en votre nom en installant des courtiers ActiveMQ ou RabbitMQ et en fournissant différentes topologies de réseau et d'autres besoins en infrastructure.

Vous pouvez utiliser une fonction Lambda pour traiter des enregistrements de votre agent de messages Amazon MQ. Lambda invoque votre fonction via un mappage de source d'événement, une ressource Lambda qui lit les messages de votre agent et invoque la fonction de manière synchrone.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Le mappage de source d'événement Amazon MQ est sujet aux restrictions de configuration suivantes :

- Simultanéité : les fonctions Lambda qui utilisent un mappage des sources d'événements Amazon MQ disposent d'un paramètre de simultanéité maximale par défaut. Pour ActiveMQ, le service Lambda limite le nombre d'environnements d'exécution simultanés à cinq par mappage de source d'événements Amazon MQ. Pour RabbitMQ, le nombre d'environnements d'exécution simultanés est limité à 1 par mappage de source d'événements Amazon MQ. Même si vous modifiez les

paramètres de simultanéité réservés ou fournis de votre fonction, le service Lambda ne mettra pas plus d'environnements d'exécution à disposition. Pour demander une augmentation de la simultanéité maximale par défaut pour un seul mappage de source d'événement Amazon MQ, AWS Support contactez le mappage de source UUID d'événement, ainsi que la région. Étant donné que les augmentations sont appliquées au niveau du mappage de source d'événement spécifique, et non au niveau du compte ou de la région, vous devez demander manuellement une augmentation de l'échelle pour chaque mappage de source d'événement.

- Traitement entre comptes – Lambda ne prend pas en charge le traitement entre comptes. Vous ne pouvez pas utiliser Lambda pour traiter des enregistrements d'un agent de messages Amazon MQ se trouvant dans un autre Compte AWS.
- Authentification — Pour ActiveMQ, seul ActiveMQ est pris en charge. SimpleAuthenticationPlugin Pour RabbitMQ, seul le mécanisme PLAINd'authentification est pris en charge. Les utilisateurs doivent utiliser AWS Secrets Manager pour gérer leurs informations d'identification. Pour plus d'informations sur l'authentification ActiveMQ, consultez la section Intégration des courtiers ActiveMQ LDAP dans le guide du développeur Amazon MQ.
- Quota de connexion – Les agents ont un nombre maximum de connexions autorisées par protocole de niveau filaire. Ce quota est basé sur le type d'instance de l'agent. Pour plus d'informations, consultez la section Agents de Quotas dans Amazon dans le Manuel du développeur Amazon MQ.
- Connectivité — Vous pouvez créer des courtiers dans un cloud privé virtuel public ou privé (VPC). Pour le mode privéVPCs, votre fonction Lambda doit accéder au pour VPC recevoir des messages. Pour plus d'informations, voir the section called “Configuration de l'accès au réseau” plus loin dans cette section.
- Destinations d'événements – Seules les destinations en file d'attente sont prises en charge. Toutefois, vous pouvez utiliser une rubrique virtuelle qui se comporte comme une rubrique en interne tout en interagissant avec Lambda en tant que file d'attente. Pour plus d'informations, consultez Virtual Destinations (Destinations virtuelles) sur le site web Apache ActiveMQ, et Virtual Hosts (Hôtes virtuels) sur le site web RabbitMQ.
- Topologie réseau – Pour ActiveMQ, un mappage de source d'événement prend en charge une seule instance ou un seul agent en veille. Pour RabbitMQ, un mappage de source d'événement prend en charge un seul agent d'instance ou un seul déploiement de cluster. Les agents à instance unique nécessitent un point de terminaison de basculement. Pour plus d'informations sur ces modes de déploiement d'agent, consultez Architecture d'agent ActiveMQ et Architecture d'agent RabbitMQ dans le Guide du développeur Amazon MQ.
- Protocoles – Les protocoles pris en charge dépendent du type d'intégration d'Amazon MQ.

- Pour les intégrations ActiveMQ, Lambda consomme les messages à l'aide OpenWire du protocole /Java Message Service (). JMS Aucun autre protocole n'est pris en charge pour la consommation de messages. Dans le cadre JMS du protocole, uniquement `TextMessage`et `BytesMessage`sont pris en charge. Lambda prend également en charge les propriétés JMS personnalisées. Pour plus d'informations sur le OpenWire protocole, consultez OpenWirele site Web d'Apache ActiveMQ.
- Pour les intégrations avec RabbitMQ, Lambda consomme les messages en utilisant le protocole 0-9-1. AMQP Aucun autre protocole n'est pris en charge pour la consommation de messages. Pour plus d'informations sur l'implémentation du protocole 0-9-1 par RabbitMQ, consultez AMQP le guide de référence complet AMQP 0-9-1 sur le site Web de RabbitMQ.

Lambda prend automatiquement en charge les dernières versions d'ActiveMQ et de RabbitMQ qu'Amazon MQ prend en charge. Pour connaître les dernières versions prises en charge, consultez Notes de mise à jour Amazon MQ dans le Guide du développeur Amazon MQ.

> **Note**
>
> Par défaut, Amazon MQ comporte une fenêtre de maintenance hebdomadaire pour les agents. Pendant cette période, les agents ne sont pas disponibles. Pendant ce temps, Lambda ne peut pas traiter les messages des agents sans veille.

Rubriques


- Comprendre le groupe de consommateurs Lambda pour Amazon MQ
- Traiter les messages Amazon MQ avec Lambda
- Paramètres de mappage des sources d'événements
- Filtrer les événements provenant d'une source d'événements Amazon MQ
- Résoudre les erreurs de mappage des sources d'événements Amazon MQ


## Comprendre le groupe de consommateurs Lambda pour Amazon MQ

Pour interagir avec Amazon MQ, Lambda crée un groupe de consommateurs qui peut être lu à partir de vos agents Amazon MQ. Le groupe de consommateurs est créé avec le même identifiant que le mappage des sources d'événementsUUID.

Pour les sources d'événements Amazon MQ, Lambda réunit les enregistrements par lots et les envoie à votre fonction dans une seule charge utile. Pour contrôler le comportement, vous pouvez configurer la fenêtre de traitement par lots et la taille du lot. Lambda extrait les messages jusqu'à ce qu'il traite la taille maximale de la charge utile de 6 Mo, que la fenêtre de traitement par lots expire ou que le nombre d'enregistrements atteigne la taille totale du lot. Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

Le groupe de consommateurs récupère les messages sous forme BLOB d'octets, les code en base64 en une seule JSON charge utile, puis invoque votre fonction. Si votre fonction renvoie une erreur pour l'un des messages d'un lot, Lambda réessaie le lot de messages complet jusqu'à ce que le traitement réussisse ou que les messages expirent.

> **Note**
>
> Alors que les fonctions Lambda ont généralement un délai d'expiration maximal de 15 minutes, les mappages de sources d'événements pour Amazon, Apache Kafka autogéré, MSK Amazon DocumentDB et Amazon MQ pour ActiveMQ et RabbitMQ ne prennent en charge que les fonctions dont le délai d'expiration maximal est de 14 minutes. Cette contrainte garantit que le mappage des sources d'événements peut gérer correctement les erreurs de fonction et effectuer de nouvelles tentatives.

Vous pouvez surveiller l'utilisation simultanée d'une fonction donnée à l'aide de la `ConcurrentExecutions` métrique d'Amazon CloudWatch. Pour de plus amples informations sur la simultanéité, consultez the section called "Configuration de la simultanéité réservée".

Example Événements d'enregistrement Amazon MQ

ActiveMQ

```
{
   "eventSource": "aws:mq",
   "eventSourceArn": "arn:aws:mq:us-
east-2:111122223333:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
   "messages": [
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/text-message",
        "deliveryMode": 1,
```

```
        "replyTo": null,
        "type": null,
        "expiration": "60000",
        "priority": 1,
        "correlationId": "myJMSCoID",
        "redelivered": false,
        "destination": {
          "physicalName": "testQueue"
        },
        "data":"QUJDOkFBQUE=",
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959,
        "properties": {
          "index": "1",
          "doAlarm": "false",
          "myCustomProperty": "value"
        }
      },
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/bytes-message",
        "deliveryMode": 1,
        "replyTo": null,
        "type": null,
        "expiration": "60000",
        "priority": 2,
        "correlationId": "myJMSCoID1",
        "redelivered": false,
        "destination": {
          "physicalName": "testQueue"
        },
        "data":"LQaGQ82S48k=",
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959,
        "properties": {
          "index": "1",
          "doAlarm": "false",
          "myCustomProperty": "value"
        }
      }
   ]
```

```
}
```

### RabbitMQ

```
{
  "eventSource": "aws:rmq",
  "eventSourceArn": "arn:aws:mq:us-
east-2:111122223333:broker:pizzaBroker:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "pizzaQueue::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                50
              ]
            },
            "numberInHeader": 10
          },
          "deliveryMode": 1,
          "priority": 34,
          "correlationId": null,
          "replyTo": null,
          "expiration": "60000",
```

```
          "messageId": null,
          "timestamp": "Jan 1, 1970, 12:33:41 AM",
          "type": null,
          "userId": "AIDACKCEVSQ6C2EXAMPLE",
          "appId": null,
          "clusterId": null,
          "bodySize": 80
        },
        "redelivered": false,
        "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
      }
    ]
  }
}
```

> **Note**
>
> Dans l'exemple RabbitMQ, `pizzaQueue` est le nom de la file d'attente RabbitMQ, et `/` est le nom de l'hôte virtuel. Lors de la réception de messages, la source d'événement répertorie les messages sous `pizzaQueue::/`.

## Traiter les messages Amazon MQ avec Lambda

Rubriques

- Configuration de l'accès au réseau
- Création du mappage des sources d'événements

### Configuration de l'accès au réseau

Pour donner à Lambda un accès complet à votre courtier via le mappage de votre source d'événements, soit votre courtier doit utiliser un point de terminaison public (adresse IP publique), soit vous devez fournir un accès à l'Amazon dans lequel VPC vous avez créé le courtier.

Par défaut, lorsque vous créez un agent Amazon MQ, l'indicateur `PubliclyAccessible` défini sur false. Pour que votre agent reçoive une adresse IP publique, vous devez définir l'indicateur `PubliclyAccessible` sur true.

La meilleure pratique pour utiliser Amazon MQ avec Lambda consiste à utiliser des AWS PrivateLink VPCpoints de terminaison et à donner à votre fonction Lambda l'accès à celle de votre courtier. VPC Déployez un point de terminaison pour Lambda et, pour ActiveMQ uniquement, un point de terminaison pour (). AWS Security Token Service AWS STS Si votre courtier utilise l'authentification, déployez également un point de terminaison pour AWS Secrets Manager. Pour en savoir plus, consultez the section called “Configurer les points de VPC terminaison Amazon”.

Vous pouvez également configurer une NAT passerelle sur chaque sous-réseau public du sous-réseau VPC contenant votre courtier Amazon MQ. Pour de plus amples informations, veuillez consulter the section called “Accès à Internet pour les VPC fonctions”.

Lorsque vous créez un mappage de source d'événements pour un courtier Amazon MQ, Lambda vérifie si des interfaces réseau élastiques (ENIs) sont déjà présentes pour les sous-réseaux et les groupes de sécurité de votre courtier. VPC Si Lambda trouve des objets existantsENIs, il essaie de les réutiliser. Sinon, Lambda en crée un nouveau ENIs pour se connecter à la source de l'événement et appeler votre fonction.

> **Note**
>
> Les fonctions Lambda s'exécutent toujours au sein VPCs du service Lambda. Ils VPCs sont gérés automatiquement par le service et ne sont pas visibles pour les clients. Vous pouvez également connecter votre fonction à un AmazonVPC. Dans les deux cas, la VPC configuration de votre fonction n'affecte pas le mappage des sources d'événements. Seule la configuration des sources d'événements VPC détermine la manière dont Lambda se connecte à votre source d'événements.

Configuration des règles du groupe de sécurité

Configurez les groupes de sécurité pour l'Amazon VPC contenant votre Amazon MQ.

> **Note**
>
> Par défaut, Amazon MQ pour ActiveMQ utilise le port, et 61617 Amazon MQ pour le port RabbitMQ. 5671

- Règles de trafic entrant : autorisez tout le trafic sur le port du courtier pour le groupe de sécurité associé à votre source d'événement à partir de son propre groupe de sécurité.

- Règles de sortie : autorisez tout le trafic sur le port 443 pour toutes les destinations. Autorisez tout le trafic sur le port du broker à partir de son propre groupe de sécurité.
- Règles relatives aux points de VPC terminaison Amazon : pour le groupe de sécurité associé à votre point de VPC terminaison Amazon, autorisez le trafic entrant sur le port en 443 provenance du groupe de sécurité du courtier

### Configurer les points de VPC terminaison Amazon

Lorsque vous utilisez des VPC points de terminaison Amazon, AWS achemine vos API appels pour appeler une fonction à l'aide de l'interface réseau élastique (ENI) du point de terminaison. Le principal du service Lambda doit faire appel à toutes `lambda:InvokeFunction` les fonctions qui les utilisent. ENIs En outre, pour Amazon MQ pour ActiveMQ, le principal du service Lambda doit exécuter les rôles qui utilisent le. `sts:AssumeRole` ENIs

Par défaut, les VPC points de terminaison Amazon ont des IAM politiques ouvertes qui permettent un large accès aux ressources. Pour utiliser Amazon MQ avec Lambda en production, vous pouvez restreindre ces politiques afin de n'autoriser que des principaux à accéder à des rôles et fonctions spécifiques.

Les exemples de politiques de point de VPC terminaison suivants montrent comment accorder l'accès requis aux points de terminaison Lambda AWS STS et aux points de terminaison Lambda.

Example Politique de point de AWS STS terminaison — point de terminaison (ActiveMQ uniquement)

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "arn:aws::iam::123456789012:role/my-role"
          }
      ]
  }
```

Example Politique relative aux terminaux — Point de terminaison Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "arn:aws::lambda:us-west-2:123456789012:function:my-function"
          }
      ]
  }
```

Si votre courtier Amazon MQ utilise l'authentification, vous pouvez également restreindre la politique de point de terminaison pour le point de terminaison Secrets Manager. Pour appeler le Secrets ManagerAPI, Lambda utilise votre rôle fonctionnel, et non le principal du service Lambda. L'exemple suivant montre une politique de point de terminaison de Secrets Manager.

Example VPCpolitique relative aux terminaux — point de terminaison Secrets Manager

```
{
      "Statement": [
          {
              "Action": "secretsmanager:GetSecretValue",
              "Effect": "Allow",
              "Principal": {
                  "AWS": [
                      "arn:aws::iam::123456789012:role/my-role"
                  ]
              },
              "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-
secret"
          }
      ]
  }
```

> **Warning**
>
> Le fait de restreindre les politiques de vos terminaux pour n'autoriser que les API appels provenant de votre organisation empêche le mappage des sources d'événements de fonctionner correctement.

## Création du mappage des sources d'événements

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des enregistrements d'un agent Amazon MQ à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter les mêmes données avec plusieurs fonctions, ou pour traiter des éléments à partir de plusieurs sources avec une seule fonction.

Pour configurer votre fonction de manière à lire depuis Amazon MQ, ajoutez les autorisations requises et créez un déclencheur MQ dans la console Lambda.

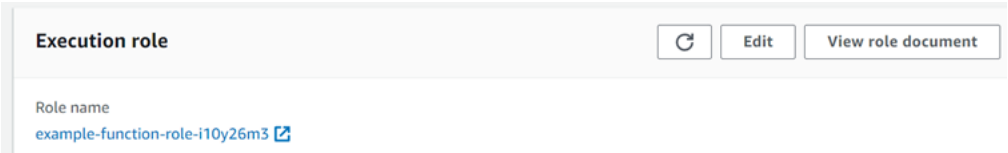
Pour lire les enregistrements d'un courtier Amazon MQ, votre fonction Lambda a besoin des autorisations suivantes. Vous autorisez Lambda à interagir avec votre courtier Amazon MQ et ses ressources sous-jacentes en ajoutant des instructions d'autorisation à votre rôle d'exécution de fonction :

- mq : DescribeBroker
- responsable des secrets : GetSecretValue
- EC2 : CreateNetworkInterface
- EC2 : DeleteNetworkInterface
- EC2 : DescribeNetworkInterfaces
- EC2 : DescribeSecurityGroups
- EC2 : DescribeSubnets
- EC2 : DescribeVpcs
- journaux : CreateLogGroup
- journaux : CreateLogStream
- journaux : PutLogEvents

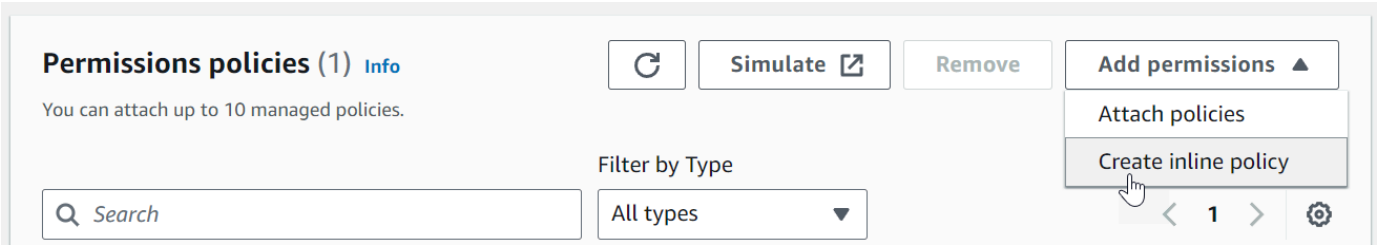
> **Note**
>
> Lorsque vous utilisez une clé gérée par le client chiffrée, ajoutez également l'autorisation `kms:Decrypt`.

Pour ajouter des autorisations et créer un déclencheur

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Choisissez l'onglet Configuration, puis Permissions (Autorisations).
4. Sous Nom du rôle, choisissez le lien vers votre rôle d'exécution. Ce lien ouvre le rôle dans la IAM console.




5. Choisissez Ajouter des autorisations, puis choisissez Créer une politique intégrée.



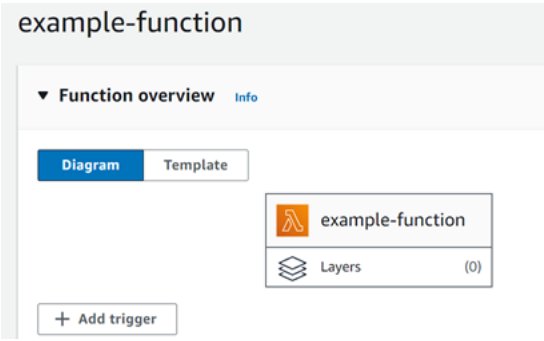

6. Dans l'éditeur de politiques, choisissez JSON. Saisissez la stratégie suivante. Votre fonction a besoin de ces autorisations pour lire depuis un courtier Amazon MQ.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "mq:DescribeBroker",
          "secretsmanager:GetSecretValue",
          "ec2:CreateNetworkInterface",
          "ec2:DeleteNetworkInterface",
          "ec2:DescribeNetworkInterfaces",
          "ec2:DescribeSecurityGroups",
```

```
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Lorsque vous utilisez une clé cryptée gérée par le client, vous devez également ajouter l'`kms:Decrypt`autorisation.

7. Choisissez Suivant. Entrez un nom de stratégie, puis choisissez Créer une politique.
8. Retournez à votre fonction dans la console Lambda. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).




9. Choisissez le type de déclencheur MQ.
10. Configurez les options requises, puis choisissez Add (Ajouter).

Lambda prend en charge les options suivantes pour les sources d'événement Amazon MQ.

- MQ broker (Agent MQ) – Sélectionnez un agent Amazon MQ.
- Batch size (Taille de lot) – Définissez le nombre maximum de messages à extraire dans un lot.
- Queue name (Nom de file d'attente) – Entrez la file d'attente Amazon MQ à consommer.
- Source access configuration (Configuration de l'accès source) – Entrez les informations d'hôte virtuel et le secret Secrets Manager qui stocke vos informations d'identification d'agent.

- Enable trigger (Activer le déclencheur) – Désactivez le déclencheur pour arrêter le traitement des enregistrements.

Pour activer ou désactiver le déclencheur (ou le supprimer), choisissez le déclencheur MQ dans le concepteur. Pour reconfigurer le déclencheur, utilisez les API opérations de mappage des sources d'événements.

## Paramètres de mappage des sources d'événements

Tous les types de sources d'événements Lambda partagent les mêmes opérations CreateEventSourceMapping. UpdateEventSourceMappingAPI Cependant, seuls certains paramètres s'appliquent à Amazon MQ et à RabbitMQ.

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| BatchSize | N | 100 | Maximum : 10 000. |
| Activé | N | VRAI | none |
| FunctionName | Y | N/A | none |
| FilterCriteria | N | N/A | Contrôlez les événements que Lambda envoie à votre fonction |
| MaximumBatchingWindowInSeconds | N | 500 ms | Comportement de traitement par lots |
| Files d'attente | N | N/A | Le nom de la file d'attente de destination de l'agent Amazon MQ à consommer. |
| SourceAccessConfigurations | N | N/A | Pour ActiveMQB ASIC, _ identifiants. AUTH Pour RabbitMQ, il peut contenir à la fois les |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| | | | AUTH informations d'identification BASIC _ et VIRTUAL les informations _. HOST |

## Filtrer les événements provenant d'une source d'événements Amazon MQ

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources d'événements Amazon MQ.

Rubriques

- Principes de base du filtrage des événements Amazon MQ

### Principes de base du filtrage des événements Amazon MQ

Supposons que votre file d'attente de messages Amazon MQ contienne des messages dans un JSON format valide ou sous forme de chaînes simples. Un exemple d'enregistrement ressemblerait à ce qui suit, avec les données converties en chaîne encodée en Base64 dans le champ `data`.

ActiveMQ

```
{
    "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
    "messageType": "jms/text-message",
    "deliveryMode": 1,
    "replyTo": null,
    "type": null,
    "expiration": "60000",
    "priority": 1,
    "correlationId": "myJMSCoID",
    "redelivered": false,
    "destination": {
```

```
      "physicalName": "testQueue"
    },
    "data":"QUJDOkFBQUE=",
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959,
    "properties": {
      "index": "1",
      "doAlarm": "false",
      "myCustomProperty": "value"
    }
}
```

### RabbitMQ

```
{
    "basicProperties": {
        "contentType": "text/plain",
        "contentEncoding": null,
        "headers": {
            "header1": {
                "bytes": [
                  118,
                  97,
                  108,
                  117,
                  101,
                  49
                ]
            },
            "header2": {
                "bytes": [
                  118,
                  97,
                  108,
                  117,
                  101,
                  50
                ]
            },
            "numberInHeader": 10
        },
        "deliveryMode": 1,
```

```
        "priority": 34,
        "correlationId": null,
        "replyTo": null,
        "expiration": "60000",
        "messageId": null,
        "timestamp": "Jan 1, 1970, 12:33:41 AM",
        "type": null,
        "userId": "AIDACKCEVSQ6C2EXAMPLE",
        "appId": null,
        "clusterId": null,
        "bodySize": 80
        },
    "redelivered": false,
    "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}
```

Pour les agents Active MQ et Rabbit MQ, vous pouvez utiliser le filtrage d'événements pour filtrer les enregistrements à l'aide de la clé `data`. Supposons que votre file d'attente Amazon MQ contienne des messages au format suivantJSON.

```
{
    "timeout": 0,
    "IPAddress": "203.0.113.254"
}
```

Pour filtrer uniquement les enregistrements dont le champ `timeout` est supérieur à 0, l'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\": [ \">\",
 0] } } ] } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "data": {
```

```
        "timeout": [ { "numeric": [ ">", 0 ] } ]
        }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Avec Amazon MQ, vous pouvez également filtrer les enregistrements dont le message est une chaîne simple. Supposons que vous vouliez traiter uniquement les enregistrements dont le message commence par « Result:  ». L'objet `FilterCriteria` se présente comme suit.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "data": [
        {
        "prefix": "Result: "
        }
    ]
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d’événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

Les messages Amazon MQ doivent être des chaînes codées en UTF -8, soit des chaînes simples, soit au format. JSON Cela est dû au fait que Lambda décode les tableaux d'octets Amazon MQ en UTF -8 avant d'appliquer des critères de filtre. Si vos messages utilisent un autre encodage, tel que UTF -16 ouASCII, ou si le format du message ne correspond pas au `FilterCriteria` format, Lambda traite uniquement les filtres de métadonnées. Le tableau suivant résume le comportement spécifique :

| Format du du message entrant | Modèle de filtre de format pour les propriétés des messages | Action obtenue. |
|---|---|---|
| Chaîne de texte brut | Chaîne de texte brut | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne de texte brut | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Chaîne de texte brut | Valide JSON | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Chaîne de texte brut | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne non codée en UTF -8 | JSON, chaîne simple ou sans motif | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |

# Résoudre les erreurs de mappage des sources d'événements Amazon MQ

Quand une fonction Lambda rencontre une erreur irrécupérable, votre consommateur Amazon MQ arrête le traitement des enregistrements. D'autres consommateurs peuvent continuer le traitement, à condition qu'ils ne rencontrent pas la même erreur. Pour déterminer la cause potentielle d'un consommateur arrêté, vérifiez le champ `StateTransitionReason` dans les détails de retour de votre `EventSourceMapping` pour l'un des codes suivants :

**`ESM_CONFIG_NOT_VALID`**

La configuration de mappage de source d'événement n'est pas valide.

**`EVENT_SOURCE_AUTHN_ERROR`**

Lambda n'a pas réussi à authentifier la source de l'événement.

**`EVENT_SOURCE_AUTHZ_ERROR`**

Lambda ne dispose pas des autorisations requises pour accéder à la source d'événement.

**`FUNCTION_CONFIG_NOT_VALID`**

La configuration de la fonction n'est pas valide.

Les enregistrements ne sont pas non plus traités si Lambda les abandonne en raison de leur taille. La taille limite pour les enregistrements Lambda est de 6 Mo. Pour redistribuer les messages en cas d'erreur de fonctionnement, vous pouvez utiliser une file d'attente contenant des lettres mortes (). DLQ Pour plus d'informations, consultez la section Remise et DLQ gestion des messages sur le site Web d'Apache ActiveMQ et le Guide de fiabilité sur le site Web de RabbitMQ.

> **Note**
>
> Lambda ne prend pas en charge les stratégies de relivraison personnalisées. Lambda utilise plutôt une politique dont les valeurs par défaut sont celles de la page Politique de remise du site Web Apache ActiveMQ, définies sur 6. `maximumRedeliveries`

# Utilisation de Lambda avec Amazon MSK

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Amazon Managed Streaming for Apache Kafka MSK (Amazon) est un service entièrement géré que vous pouvez utiliser pour créer et exécuter des applications utilisant Apache Kafka pour traiter des données de streaming. Amazon MSK simplifie la configuration, le dimensionnement et la gestion des clusters exécutant Kafka. Amazon facilite MSK également la configuration de votre application pour plusieurs zones de disponibilité et pour des raisons de sécurité avec AWS Identity and Access Management (IAM). Amazon MSK prend en charge plusieurs versions open source de Kafka.

AmazonMSK, en tant que source d'événements, fonctionne de la même manière qu'Amazon Simple Queue Service (AmazonSQS) ou Amazon Kinesis. Lambda interroge en interne les nouveaux messages de la source d'événement, puis invoque de manière synchrone la fonction Lambda cible. Lambda lit les messages par lot et les fournit à votre fonction en tant que charge utile d'événement. La taille maximale du lot est configurable (la valeur par défaut est de 100 messages). Pour de plus amples informations, veuillez consulter Comportement de traitement par lots.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Pour un exemple de configuration d'Amazon en MSK tant que source d'événements, consultez la section Utilisation d'Amazon en MSK tant que source d'événements AWS Lambda sur le blog AWS Compute. Pour un didacticiel complet, consultez Amazon MSK Lambda Integration dans les Amazon MSK Labs.

Rubriques

- Exemple d'évènement

- Configuration des sources d'MSKévénements Amazon pour Lambda
- Traitement des MSK messages Amazon avec Lambda
- Utilisation du filtrage des événements avec une source d'MSKévénements Amazon
- Capture des lots rejetés pour une source d'événements Amazon MSK
- Tutoriel : Utilisation d'un mappage de source d'MSKévénements Amazon pour appeler une fonction Lambda

## Exemple d’évènement

Lambda envoie le lot de messages dans le paramètre d’événement quand il invoque votre fonction. La charge utile d’un événement contient un tableau de messages. Chaque élément du tableau contient les détails du MSK sujet Amazon et l'identifiant de la partition, ainsi qu'un horodatage et un message codé en base64.


```
{
   "eventSource":"aws:kafka",
   "eventSourceArn":"arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
   "bootstrapServers":"b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
   "records":{
      "mytopic-0":[
         {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "key":"abcDEFghiJKLmnoPQRstuVWXyz1234==",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[
               {
                  "headerKey":[
                     104,
                     101,
                     97,
                     100,
                     101,
                     114,
```

```
                    86,
                    97,
                    108,
                    117,
                    101
                ]
            }
        ]
    }
  ]
 }
}
```

## Configuration des sources d'MSKévénements Amazon pour Lambda

Avant de créer un mappage des sources d'événements pour votre MSK cluster Amazon, vous devez vous assurer que votre cluster et celui dans lequel VPC il réside sont correctement configurés. Vous devez également vous assurer que le rôle d'exécution de votre fonction Lambda dispose des autorisations nécessairesIAM.

Suivez les instructions des sections suivantes pour configurer votre MSK cluster Amazon et votre VPC fonction Lambda. Pour savoir comment créer le mappage des sources d'événements, consultezthe section called “Ajouter Amazon MSK en tant que source d'événements”.

Rubriques

- MSKauthentification par cluster
- Gestion des API accès et des autorisations
- Authentification et erreurs d’autorisation
- Configuration réseau

### MSKauthentification par cluster

Lambda a besoin d'une autorisation pour accéder au MSK cluster Amazon, récupérer des enregistrements et effectuer d'autres tâches. Amazon MSK propose plusieurs options pour contrôler l'accès des clients au MSK cluster.

Options d’accès au cluster

- Accès non authentifié

- SASL/SCRAMauthentification
- Authentification basée sur les rôles IAM
- TLSAuthentification mutuelle
- Configuration du TLS secret m
- Comment Lambda choisit un courtier bootstrap

Accès non authentifié

Si aucun client n'accède au cluster via Internet, vous pouvez utiliser un accès non authentifié.

SASL/SCRAMauthentification

Amazon MSK prend en charge l'authentification simple et l'authentification par couche de sécurité/mécanisme d'authentification Salted Challenge Response (SASL/SCRAM) avec le chiffrement Transport Layer Security (TLS). Pour que Lambda puisse se connecter au cluster, vous devez stocker les informations d'authentification (nom d'utilisateur et mot de passe) dans un AWS Secrets Manager secret.

Pour plus d'informations sur l'utilisation de Secrets Manager, consultez Authentification par nom d'utilisateur et mot de passe avec AWS Secrets Manager dans le Guide du développeur Amazon Managed Streaming for Apache Kafka.

Amazon MSK ne prend pas en charge PLAIN l'authentificationSASL//.

Authentification basée sur les rôles IAM

Vous pouvez l'utiliser IAM pour authentifier l'identité des clients qui se connectent au MSK cluster. Si IAM l'authentification est active sur votre MSK cluster et que vous ne fournissez pas de secret pour l'authentification, Lambda utilise automatiquement l'authentification par défaut. IAM Pour créer et déployer des politiques basées sur les utilisateurs ou les rôles, utilisez la IAM console ou. API Pour plus d'informations, consultez la section sur le contrôle IAM d'accès dans le guide du développeur Amazon Managed Streaming for Apache Kafka.

Pour permettre à Lambda de se connecter au MSK cluster, de lire des enregistrements et d'effectuer d'autres actions requises, ajoutez les autorisations suivantes au rôle d'exécution de votre fonction.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:DescribeGroup",
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeClusterDynamicConfiguration"
            ],
            "Resource": [
                "arn:aws:kafka:region:account-id:cluster/cluster-name/cluster-uuid",
                "arn:aws:kafka:region:account-id:topic/cluster-name/cluster-uuid/topic-
name",
                "arn:aws:kafka:region:account-id:group/cluster-name/cluster-
uuid/consumer-group-id"
            ]
        }
    ]
}
```

Vous pouvez étendre ces autorisations à un cluster, une rubrique et un groupe spécifiques. Pour plus d'informations, consultez les actions Amazon MSK Kafka dans le guide du développeur Amazon Managed Streaming for Apache Kafka.

TLSAuthentification mutuelle

Mutual TLS (mTLS) fournit une authentification bidirectionnelle entre le client et le serveur. Le client envoie un certificat au serveur pour que le serveur vérifie le client, et le serveur envoie un certificat au client pour que le client vérifie le serveur.

Pour AmazonMSK, Lambda agit en tant que client. Vous configurez un certificat client (sous forme de secret dans Secrets Manager) pour authentifier Lambda auprès des courtiers de votre cluster. MSK Le certificat client doit être signé par une autorité de certification dans le magasin d'approbations du serveur. Le MSK cluster envoie un certificat de serveur à Lambda pour authentifier les courtiers auprès de Lambda. Le certificat du serveur doit être signé par une autorité de certification (CA) présente dans le AWS trust store.

Pour obtenir des instructions sur la façon de générer un certificat client, consultez la section Présentation de TLS l'authentification mutuelle pour Amazon MSK en tant que source d'événements.

Amazon MSK ne prend pas en charge les certificats de serveur auto-signés, car tous les courtiers d'Amazon MSK utilisent des certificats publics signés par Amazon Trust Services CAs, auxquels Lambda fait confiance par défaut.

Pour plus d'informations sur m TLS for AmazonMSK, consultez Mutual TLS Authentication dans le guide du développeur Amazon Managed Streaming for Apache Kafka.

Configuration du TLS secret m

Le AUTH secret CLIENT CERTIFICATE _ TLS _ _ nécessite un champ de certificat et un champ de clé privée. Pour une clé privée chiffrée, le secret nécessite un mot de passe de clé privée. Le certificat et la clé privée doivent tous deux être au PEM format.

> **Note**
>
> Lambda prend en charge les algorithmes de chiffrement par clé privée PBES1(mais pasPBES2).

Le champ de certificat doit contenir une liste de certificats, commençant par le certificat client, suivi de tous les certificats intermédiaires et se terminant par le certificat racine. Chaque certificat doit commencer sur une nouvelle ligne avec la structure suivante :

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager prend en charge les secrets jusqu'à 65 536 octets, ce qui offre suffisamment d'espace pour de longues chaînes de certificats.

La clé privée doit être au format PKCS#8, avec la structure suivante :

```
-----BEGIN PRIVATE KEY-----
         <private key contents>
-----END PRIVATE KEY-----
```

Pour une clé privée chiffrée, utilisez la structure suivante :

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
```

```
          <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

L'exemple suivant montre le contenu d'un secret pour mon TLS authentification à l'aide d'une clé privée cryptée. Pour une clé privée chiffrée, vous devez inclure le mot de passe de clé privée dans le secret.

```
{
 "privateKeyPassword": "testpassword",
 "certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
 "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

Comment Lambda choisit un courtier bootstrap

Lambda choisit un courtier d'amorçage en fonction des méthodes d'authentification disponibles sur votre cluster, et si vous fournissez un secret pour l'authentification. Si vous fournissez un secret pour m TLS ouSASL/SCRAM, Lambda choisit automatiquement cette méthode d'authentification. Si vous ne e faites pas, Lambda sélectionne la méthode d'authentification la plus puissante active sur votre cluster. Voici l'ordre de priorité dans lequel Lambda sélectionne un courtier, de l'autorisation la plus forte à la plus faible :

- m TLS (secret fourni pour mTLS)
- SASL/SCRAM(secret fourni pourSASL/SCRAM)
- SASLIAM(aucun secret fourni et IAM authentification active)

- Non authentifié TLS (aucun secret fourni et IAM authentification inactive)
- Texte en clair (aucun secret n'est fourni, et les fonctions d'IAMauthentification et de non-authentification ne TLS sont pas actives)

> **Note**
>
> Si Lambda ne parvient pas à se connecter au type de courtier le plus sécurisé, Lambda n'essaie pas de se connecter à un type de courtier différent (plus faible). Si vous souhaitez que Lambda choisisse un type de courtier plus faible, désactivez toutes les méthodes d'authentification plus puissantes sur votre cluster.

## Gestion des API accès et des autorisations

Outre l'accès au MSK cluster Amazon, votre fonction a besoin d'autorisations pour effectuer diverses MSK API actions Amazon. Ajoutez ces autorisations au rôle d'exécution de votre fonction. Si vos utilisateurs ont besoin d'accéder à l'une des MSK API actions Amazon, ajoutez les autorisations requises à la politique d'identité de l'utilisateur ou du rôle.

Vous pouvez ajouter manuellement chacune des autorisations suivantes à votre rôle d'exécution. Vous pouvez également associer la politique AWS gérée AWSLambdaMSKExecutionRoleà votre rôle d'exécution. La `AWSLambdaMSKExecutionRole` politique contient toutes les API actions et VPC autorisations requises répertoriées ci-dessous.

Autorisations de rôle d'exécution de fonction Lambda requises

Pour créer et stocker des journaux dans un groupe de CloudWatch journaux dans Amazon Logs, votre fonction Lambda doit disposer des autorisations suivantes dans son rôle d'exécution :

- journaux : CreateLogGroup
- journaux : CreateLogStream
- journaux : PutLogEvents

Pour que Lambda puisse accéder à votre MSK cluster Amazon en votre nom, votre fonction Lambda doit disposer des autorisations suivantes dans son rôle d'exécution :

- Kafka : DescribeCluster

- Source : V2 DescribeCluster
- Kafka : GetBootstrapBrokers
- kafka : DescribeVpcConnection : Nécessaire uniquement pour les mappages de sources d'événements entre comptes.
- kafka : ListVpcConnections : Non requis dans le rôle d'exécution, mais obligatoire pour un IAM principal qui crée un mappage de source d'événements entre comptes.

Vous n'avez besoin d'ajouter que `kafka:DescribeCluster` ou `kafka:DescribeClusterV2`. Pour les MSK clusters provisionnés, l'une ou l'autre des autorisations fonctionne. Pour les MSK clusters sans serveur, vous devez utiliser`kafka:DescribeClusterV2`.

> **Note**
>
> Lambda prévoit à terme de supprimer l'autorisation `kafka:DescribeCluster` de la politique gérée `AWSLambdaMSKExecutionRole` associée. Si vous utilisez cette politique, vous devez migrer toutes les applications utilisant `kafka:DescribeCluster` pour qu'elles utilisent `kafka:DescribeClusterV2` à la place.

VPCautorisations

Si seuls les utilisateurs d'un cluster VPC peuvent accéder à votre MSK cluster Amazon, votre fonction Lambda doit être autorisée à accéder à vos ressources AmazonVPC. Ces ressources incluent vos sous-réseauxVPC, vos groupes de sécurité et vos interfaces réseau. Pour accéder à ces ressources, le rôle d'exécution de votre fonction doit disposer des autorisations suivantes. Ces autorisations sont incluses dans la politique AWSLambdaMSKExecutionRole AWS gérée.

- EC2 : CreateNetworkInterface
- EC2 : DescribeNetworkInterfaces
- EC2 : DescribeVpcs
- EC2 : DeleteNetworkInterface
- EC2 : DescribeSubnets
- EC2 : DescribeSecurityGroups

### Autorisations de fonction Lambda facultatives

Votre fonction Lambda peut également nécessiter ces autorisations pour :

- Accédez à votre SCRAM secret, si vous utilisez l'SCRAMauthentificationSASL/.
- Décrivez votre secret Secrets Manager.
- Accédez à votre AWS Key Management Service (AWS KMS) clé gérée par le client.
- Envoyez les enregistrements des invocations ayant échoué vers une destination.

### Secrets Manager et AWS KMS autorisations

Selon le type de contrôle d'accès que vous configurez pour vos MSK courtiers Amazon, votre fonction Lambda peut avoir besoin d'une autorisation pour accéder à votre SCRAM secret (si vous utilisezSASL/SCRAMauthentication), ou au secret de Secrets Manager pour déchiffrer votre clé gérée par le AWS KMS client. Afin d'accéder à ces ressources, le rôle d'exécution de votre fonction doit disposer des autorisations suivantes :

- Kafka : ListScramSecrets
- responsable des secrets : GetSecretValue
- kms:Decrypt

### Ajout d'autorisations à votre rôle d'exécution

Procédez comme suit pour ajouter la politique AWS gérée AWSLambdaMSKExecutionRoleà votre rôle d'exécution à l'aide de la IAM console.

Pour ajouter une politique AWS gérée

1. Ouvrez la page Politiques de la IAM console.
2. Dans la zone de recherche, saisissez le nom de la stratégie (`AWSLambdaMSKExecutionRole`).
3. Sélectionnez la stratégie dans la liste, puis choisissez Policy actions (Actions de stratégie), Attach (Attacher).
4. Sur la page Attach policy (Attacher une politique), sélectionnez votre rôle d'exécution dans la liste, puis choisissez Attach policy (Attacher une politique).

Accorder l'accès aux utilisateurs avec une IAM politique

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à effectuer des MSK API opérations Amazon. Pour accorder l’accès aux utilisateurs de votre organisation ou de votre compte, vous pouvez ajouter ou mettre à jour une politique basée sur l’identité. Pour plus d'informations, consultez les exemples de politiques MSK basées sur l'identité Amazon dans le guide du développeur Amazon Managed Streaming for Apache Kafka.

## Authentification et erreurs d’autorisation

Si l'une des autorisations requises pour utiliser les données du MSK cluster Amazon est manquante, Lambda affiche l'un des messages d'erreur suivants dans le mappage des sources d'événements ci-dessous. LastProcessingResult

Messages d’erreur

- Le cluster n’a pas pu autoriser Lambda
- SASLéchec de l'authentification
- Le serveur n’a pas réussi à authentifier Lambda
- Le certificat ou la clé privée fournis n’est pas valide

Le cluster n’a pas pu autoriser Lambda

PourSASL/SCRAMou mTLS, cette erreur indique que l'utilisateur indiqué ne possède pas toutes les autorisations requises de la liste de contrôle d'accès Kafka (ACL) suivantes :

- DescribeConfigs Cluster
- Décrire un groupe
- Lire le groupe
- Décrire la rubrique
- Lire la rubrique

Pour le contrôle IAM d'accès, il manque au rôle d'exécution de votre fonction une ou plusieurs des autorisations requises pour accéder au groupe ou au sujet. Passez en revue la liste des autorisations requises dans the section called “Authentification basée sur les rôles IAM”.

Lorsque vous créez Kafka ACLs ou une IAM politique avec les autorisations de cluster Kafka requises, spécifiez le sujet et le groupe en tant que ressources. Le nom de la rubrique doit

correspondre à la rubrique dans le mappage des sources d'événements. Le nom du groupe doit correspondre à celui du mappage des sources d'événementsUUID.

Une fois que vous avez ajouté les autorisations requises au rôle d'exécution, il peut y avoir un délai de plusieurs minutes avant que les modifications ne prennent effet.

SASLéchec de l'authentification

PourSASL/SCRAM, cette erreur indique que le nom d'utilisateur et le mot de passe fournis ne sont pas valides.

Pour le contrôle IAM d'accès, le rôle d'exécution ne dispose pas de l'`kafka-cluster:Connect`autorisation pour le MSK cluster. Ajoutez cette autorisation au rôle et spécifiez le nom de ressource Amazon (ARN) du cluster en tant que ressource.

Cette erreur peut se produire de façon intermittente. Le cluster rejette les connexions lorsque le nombre de TCP connexions dépasse le quota MSK de service Amazon. Lambda fait marche arrière et tente de nouveau jusqu'à ce qu'une connexion soit réussie. Une fois que Lambda se connecte au cluster et interroge les enregistrements, le dernier résultat de traitement passe à `OK`.

Le serveur n'a pas réussi à authentifier Lambda

Cette erreur indique que les courtiers Amazon MSK Kafka n'ont pas réussi à s'authentifier auprès de Lambda. Cette erreur se produit dans les conditions suivantes :

- Vous n'avez pas fourni de certificat client pour mon TLS authentification.
- Vous avez fourni un certificat client, mais les courtiers ne sont pas configurés pour utiliser TLS m.
- Les courtiers ne font pas confiance à un certificat client.

Le certificat ou la clé privée fournis n'est pas valide

Cette erreur indique que le MSK consommateur Amazon n'a pas pu utiliser le certificat ou la clé privée fournis. Assurez-vous que le certificat et la clé utilisent le PEM format et que le chiffrement par clé privée utilise un PBES1 algorithme.

## Configuration réseau

Pour que Lambda puisse utiliser votre cluster Kafka comme source d'événements, il doit avoir accès à l'Amazon dans lequel réside VPC votre cluster. Nous vous recommandons de déployer des AWS

PrivateLink VPCpoints de terminaison pour que Lambda puisse accéder à votre. VPC Déployez des points de terminaison pour Lambda AWS Security Token Service et AWS STS(). Si le broker utilise l'authentification, déployez également un VPC point de terminaison pour Secrets Manager. Si vous avez configuré une destination en cas de défaillance, déployez également un VPC point de terminaison pour le service de destination.

Vous pouvez également vous assurer que le cluster VPC associé à votre cluster Kafka inclut une NAT passerelle par sous-réseau public. Pour de plus amples informations, veuillez consulter the section called “Accès à Internet pour les VPC fonctions”.

Si vous utilisez des VPC points de terminaison, vous devez également les configurer pour activer les DNS noms privés.

Lorsque vous créez un mappage de source d'événements pour un MSK cluster, Lambda vérifie si des interfaces réseau élastiques (ENIs) sont déjà présentes pour les sous-réseaux et les groupes de sécurité de votre cluster. VPC Si Lambda trouve des objets existantsENIs, il essaie de les réutiliser. Sinon, Lambda en crée un nouveau ENIs pour se connecter à la source de l'événement et appeler votre fonction.

> **Note**
>
> Les fonctions Lambda s'exécutent toujours au sein VPCs du service Lambda. Ils VPCs sont gérés automatiquement par le service et ne sont pas visibles pour les clients. Vous pouvez également connecter votre fonction à un AmazonVPC. Dans les deux cas, la VPC configuration de votre fonction n'affecte pas le mappage des sources d'événements. Seule la configuration des sources d'événements VPC détermine la manière dont Lambda se connecte à votre source d'événements.

Votre VPC configuration Amazon est consultable via Amazon MSK API. Vous n’avez pas besoin de le configurer pendant la configuration à l’aide de la commande create-event-source-mapping.

Pour plus d'informations sur la configuration du réseau, voir Configuration AWS Lambda avec un cluster Apache Kafka au sein d'un blog VPC sur le AWS Compute.

VPCrègles du groupe de sécurité

Configurez les groupes de sécurité pour l'Amazon VPC contenant votre cluster avec les règles suivantes (au minimum) :

- Règles de trafic entrant : autorisez tout le trafic sur le port Amazon MSK Broker (9092 pour le texte en clair, 9094 pour, 9096 pour TLSSASL, 9098 pourIAM) pour les groupes de sécurité spécifiés pour votre source d'événement.
- Règles sortantes : autorisent tout le trafic sur le port 443 pour toutes les destinations. Autorisez tout le trafic sur le port Amazon MSK Broker (9092 pour le texte en clair, 9094 pourTLS, 9096 pourSASL, 9098 pourIAM) pour les groupes de sécurité spécifiés pour votre source d'événement.
- Si vous utilisez des VPC points de terminaison au lieu d'une NAT passerelle, les groupes de sécurité associés aux VPC points de terminaison doivent autoriser tout le trafic entrant sur le port 443 en provenance des groupes de sécurité de la source de l'événement.

Utilisation de points de terminaison VPC

Lorsque vous utilisez des VPC points de terminaison, API les appels pour appeler votre fonction sont acheminés via ces points de terminaison à l'aide du. ENIs Le principal du service Lambda doit appeler `sts:AssumeRole` et appeler tous `lambda:InvokeFunction` les rôles et fonctions qui les utilisent. ENIs

Par défaut, les VPC points de terminaison ont IAM des politiques ouvertes. La meilleure pratique consiste à restreindre ces politiques afin de n'autoriser que des principaux spécifiques à effectuer les actions nécessaires à l'aide de ce point de terminaison. Pour garantir que le mappage de votre source d'événements est en mesure d'appeler votre fonction Lambda, la politique du VPC point de terminaison doit autoriser le principe du service Lambda à appeler et. `sts:AssumeRole` `lambda:InvokeFunction` Le fait de restreindre les politiques de vos VPC terminaux pour autoriser uniquement les API appels provenant de votre organisation empêche le mappage des sources d'événements de fonctionner correctement.

Les exemples de politiques de point de VPC terminaison suivants montrent comment accorder l'accès requis au principal de service Lambda pour les points de terminaison Lambda et AWS STS Lambda.

Example VPCpolitique relative aux terminaux - AWS STS points de terminaison

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
```

```
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example VPCpolitique de point de terminaison - point de terminaison Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Si votre courtier Kafka utilise l'authentification, vous pouvez également restreindre la politique de point de VPC terminaison pour le point de terminaison Secrets Manager. Pour appeler le Secrets ManagerAPI, Lambda utilise votre rôle fonctionnel, et non le principal du service Lambda. L'exemple suivant montre une politique de point de terminaison de Secrets Manager.

Example VPCpolitique relative aux terminaux - Point de terminaison Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
```

```
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Si vous avez configuré une destination en cas de défaillance, Lambda utilise également le rôle de votre fonction pour appeler `s3:PutObject` `sqs:sendMessage` ou `sns:Publish` utiliser le Lambda Géré. ENIs

## Traitement des MSK messages Amazon avec Lambda

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Rubriques

- Ajouter Amazon MSK en tant que source d'événements
- Paramètres MSK de configuration Amazon
- Création de mappages de sources d’événements entre comptes
- Utilisation d'un MSK cluster Amazon comme source d'événements
- Positions de départ des interrogations et des flux
- CloudWatch Métriques Amazon
- Mise à l'échelle automatique de la source MSK d'événements Amazon

### Ajouter Amazon MSK en tant que source d'événements

Pour créer un mappage de sources d'événements, ajoutez Amazon MSK en tant que déclencheur de fonction Lambda à l'aide de la console Lambda, de an AWS SDKou du ().AWS Command Line InterfaceAWS CLI Notez que lorsque vous ajoutez Amazon MSK en tant que déclencheur, Lambda prend en charge les VPC paramètres du MSK cluster Amazon, et non ceux de la fonction Lambda. VPC

Cette section explique comment créer un mappage de source d'événement à l'aide de la console Lambda et de l' AWS CLI.

Prérequis

- Un MSK cluster Amazon et un sujet Kafka. Pour plus d'informations, consultez Getting Started Using Amazon MSK dans le guide du développeur Amazon Managed Streaming for Apache Kafka.
- Rôle d'exécution autorisé à accéder aux AWS ressources utilisées par votre MSK cluster.

Identifiant de groupe de consommateurs personnalisable

Lorsque vous configurez Kafka comme source d'événements, vous pouvez spécifier un identifiant de groupe de consommateurs. Cet identifiant de groupe de consommateurs est un identifiant existant pour le groupe de clients Kafka auquel vous souhaitez rattacher votre fonction Lambda. Vous pouvez utiliser cette fonction pour migrer facilement toutes les configurations de traitement d'enregistrements Kafka en cours depuis d'autres clients vers Lambda.

Si vous spécifiez un identifiant de groupe de consommateurs et qu'il existe d'autres sondeurs actifs au sein de ce groupe de consommateurs, Kafka distribue des messages à tous les consommateurs. En d'autres termes, Lambda ne reçoit pas l'intégralité du message relatif au sujet Kafka. Si vous souhaitez que Lambda gère tous les messages de la rubrique, désactivez tous les autres sondeurs de ce groupe de consommateurs.

De plus, si vous spécifiez un identifiant de groupe de consommateurs et que Kafka trouve un groupe de consommateurs existant valide avec le même identifiant, Lambda ignore le paramètre `StartingPosition` pour le mappage des sources d'événements. Lambda commence plutôt à traiter les enregistrements en fonction de la compensation engagée par le groupe de consommateurs. Si vous spécifiez un identifiant de groupe de consommateurs et que Kafka ne trouve aucun groupe de consommateurs existant, Lambda configure votre source d'événement avec le `StartingPosition` spécifié.

L'identifiant du groupe de consommateurs que vous spécifiez doit être unique parmi toutes vos sources d'événements Kafka. Après avoir créé un mappage des sources d'événements Kafka avec l'identifiant de groupe de consommateurs spécifié, vous ne pouvez plus mettre à jour cette valeur.

Ajouter un MSK déclencheur Amazon (console)

Suivez ces étapes pour ajouter votre MSK cluster Amazon et un sujet Kafka comme déclencheur pour votre fonction Lambda.

Pour ajouter un MSK déclencheur Amazon à votre fonction Lambda (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.

2. Choisissez le nom de votre fonction Lambda.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).
4. Sous Trigger configuration (Configuration du déclencheur), procédez comme suit :
    a. Choisissez le type de déclencheur MSK.
    b. Pour MSKcluster, sélectionnez votre cluster.
    c. Pour Batch size (Taille de lot), entrez le nombre maximum d'enregistrements à recevoir dans un même lot.
    d. Pour Batch window, veuillez saisir l'intervalle de temps maximal en secondes nécessaire à Lambda pour collecter des enregistrements avant d'invoquer la fonction.
    e. Dans Topic name (Nom de la rubrique), saisissez le nom d'une rubrique Kafka.
    f. (Facultatif) Pour l'identifiant de groupe de consommateurs, entrez l'identifiant d'un groupe de consommateurs Kafka à rejoindre.
    g. (Facultatif) Pour Position de départ, choisissez Dernier pour commencer à lire le flux à partir du dernier enregistrement, Supprimer l'horizon pour commencer au premier enregistrement disponible ou À l'horodatage pour spécifier un horodatage à partir duquel commencer la lecture.
    h. (Facultatif) Pour Authentification, choisissez la clé secrète pour vous authentifier auprès des courtiers de votre MSK cluster.
    i. Pour créer le déclencheur dans un état désactivé pour le test (recommandé), désactivez Enable trigger (Activer le déclencheur). Ou, pour activer le déclencheur immédiatement, sélectionnezActiver un déclencheur.
5. Pour créer le déclencheur, choisissez Add (Ajouter).

Ajouter un MSK déclencheur Amazon (AWS CLI)

Utilisez les exemples de AWS CLI commandes suivants pour créer et afficher un MSK déclencheur Amazon pour votre fonction Lambda.

Création d'un déclencheur à l'aide du AWS CLI

Example — Crée un mappage des sources d'événements pour le cluster qui utilise IAM l'authentification

L'exemple suivant utilise la create-event-source-mapping AWS CLI commande pour mapper une fonction Lambda nommée `my-kafka-function` à une rubrique Kafka nommée. `AWSKafkaTopic` La position de départ de la rubrique est définie sur `LATEST`. Lorsque le cluster utilise l'authentification IAM basée sur les rôles, vous n'avez pas besoin d'SourceAccessConfigurationobjet. Exemple :

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
```

Example — Crée un mappage des sources d'événements pour le cluster qui utilise SCRAM l'authentificationSASL/

Si le cluster utilise l'SCRAMauthentificationSASL/, vous devez inclure un SourceAccessConfigurationobjet qui spécifie `SASL_SCRAM_512_AUTH` et un secret Secrets ManagerARN.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Example — Crée un mappage des sources d'événements pour le cluster qui utilise TLS l'authentification m

Si le cluster utilise l'TLSauthentification m, vous devez inclure un SourceAccessConfigurationobjet qui spécifie `CLIENT_CERTIFICATE_TLS_AUTH` et un secret Secrets ManagerARN.

```
aws lambda create-event-source-mapping \
```

```
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-
fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Pour plus d'informations, consultez la documentation CreateEventSourceMappingAPIde référence.

Affichage de l'état à l'aide du AWS CLI

L'exemple suivant utilise la get-event-source-mapping AWS CLI commande pour décrire l'état du mappage des sources d'événements que vous avez créé.

```
aws lambda get-event-source-mapping \
  --uuid 6d9bce8e-836b-442c-8070-74e77903c815
```

## Paramètres MSK de configuration Amazon

Tous les types de sources d'événements Lambda partagent les mêmes opérations CreateEventSourceMapping. UpdateEventSourceMappingAPI Toutefois, seuls certains paramètres s'appliquent à AmazonMSK.

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| AmazonManagedKafkaEventSourceConfig | N | Contient le ConsumerGroupId champ, dont la valeur par défaut est unique. | Peut définir uniquement sur Create (Créer) |
| BatchSize | N | 100 | Maximum : 10 000. |
| Activé | N | Activées | none |
| EventSourceArn | Y | N/A | Peut définir uniquement sur Create (Créer) |
| FunctionName | Y | N/A | none |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| FilterCriteria | N | N/A | Contrôlez les événements que Lambda envoie à votre fonction |
| MaximumBatchingWindowInSeconds | N | 500 ms | Comportement de traitement par lots |
| SourceAccessConfigurations | N | Pas d'informations d'identification | SASL/SCRA Mou CLIENT _ _ CERTIFICATE TLS _ AUTH (mutuellesTLS) identifiants d'authentification pour votre source d'événement |
| StartingPosition | Y | N/A | AT_TIMESTAMP, TRIM _HORIZON, ou LATEST<br><br>Peut définir uniquement sur Create (Créer) |
| StartingPositionTimestamp | N | N/A | Obligatoire s' StartingPosition il est défini sur AT_ TIMESTAMP |
| Rubriques | Y | N/A | Nom de rubrique Kafka<br><br>Peut définir uniquement sur Create (Créer) |

## Création de mappages de sources d'événements entre comptes

Vous pouvez utiliser la connectivité VPC multi-privée pour connecter une fonction Lambda à un MSK cluster provisionné dans un autre. Compte AWS Utilisations de VPC connectivité multiple AWS PrivateLink, ce qui permet de maintenir tout le trafic sur le AWS réseau.

> **Note**
>
> Vous ne pouvez pas créer de mappages de sources d'événements entre comptes pour les clusters sans serveurMSK.

Pour créer un mappage des sources d'événements entre comptes, vous devez d'abord configurer la VPC multiconnectivité pour le MSK cluster. Lorsque vous créez le mappage des sources d'événements, utilisez la VPC connexion gérée ARN plutôt que le clusterARN, comme indiqué dans les exemples suivants. Le CreateEventSourceMappingfonctionnement varie également en fonction du type d'authentification utilisé par le MSK cluster.

Example — Créez un mappage des sources d'événements entre comptes pour le cluster qui utilise l'authentification IAM

Lorsque le cluster utilise l'authentification IAM basée sur les rôles, vous n'avez pas besoin d'SourceAccessConfigurationobjet. Exemple :

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
```

Example — Crée un mappage des sources d'événements entre comptes pour le cluster qui utilise l'authentificationSASL/SCRAM

Si le cluster utilise l'SCRAMauthentificationSASL/, vous devez inclure un SourceAccessConfigurationobjet qui spécifie `SASL_SCRAM_512_AUTH` et un secret Secrets ManagerARN.

Il existe deux manières d'utiliser les secrets pour les mappages de sources d'MSKévénements Amazon entre comptes avec l'authentificationSASL/SCRAM:

- Créez un secret dans le compte de fonction Lambda et synchronisez-le avec le secret du cluster. Créez une rotation pour synchroniser les deux secrets. Cette option vous permet de contrôler le secret depuis le compte de fonction.
- Utilisez le secret associé au MSK cluster. Ce secret doit autoriser l'accès intercompte au compte de la fonction Lambda. Pour plus d'informations, consultez la section Autorisations relatives aux AWS Secrets Manager secrets pour les utilisateurs d'un autre compte.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

Example — Crée un mappage des sources d'événements entre comptes pour le cluster qui utilise l'authentification m TLS

Si le cluster utilise l'TLSauthentification m, vous devez inclure un SourceAccessConfigurationobjet qui spécifie `CLIENT_CERTIFICATE_TLS_AUTH` et un secret Secrets ManagerARN. Le secret peut être stocké dans le compte du cluster ou dans le compte de la fonction Lambda.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

## Utilisation d'un MSK cluster Amazon comme source d'événements

Lorsque vous ajoutez votre MSK cluster Apache Kafka ou Amazon comme déclencheur pour votre fonction Lambda, le cluster est utilisé comme source d'événements.

Lambda lit les données d'événements des sujets Kafka que vous spécifiez `Topics` dans une CreateEventSourceMappingdemande, en fonction de `StartingPosition` ce que vous spécifiez. Lorsque le traitement a réussi, votre rubrique Kafka est validée dans votre cluster Kafka.

Si vous spécifiez `StartingPosition` comme `LATEST`, Lambda commence à lire à partir du dernier message dans chaque partition de la rubrique. Un certain temps pouvant s'écouler après la configuration du déclencheur avant que Lambda commence à lire les messages, Lambda ne lit aucun message produit pendant cette fenêtre de temps.

Lambda lit les messages de manière séquentielle pour chaque partition de rubrique Kafka. Une seule charge utile Lambda peut contenir des messages provenant de plusieurs partitions. Lorsque d'autres enregistrements sont disponibles, Lambda continue de traiter les enregistrements par lots, en fonction de la `BatchSize` valeur que vous spécifiez dans une CreateEventSourceMappingdemande, jusqu'à ce que votre fonction aborde le sujet.

Après avoir traité chaque lot, Lambda valide les décalages des messages dans celui-ci. Si votre fonction renvoie une erreur pour l'un des messages d'un lot, Lambda réessaie le lot de messages complet jusqu'à ce que le traitement réussisse ou que les messages expirent. Vous pouvez envoyer les enregistrements qui échouent à toutes les tentatives vers une destination en cas d'échec pour un traitement ultérieur.

> **Note**
>
> Alors que les fonctions Lambda ont généralement un délai d'expiration maximal de 15 minutes, les mappages de sources d'événements pour Amazon, Apache Kafka autogéré, MSK Amazon DocumentDB et Amazon MQ pour ActiveMQ et RabbitMQ ne prennent en charge que les fonctions dont le délai d'expiration maximal est de 14 minutes. Cette contrainte garantit que le mappage des sources d'événements peut gérer correctement les erreurs de fonction et effectuer de nouvelles tentatives.

## Positions de départ des interrogations et des flux

Sachez que l'interrogation des flux lors des mises à jour et de la création du mappage des sources d'événements est finalement cohérente.

- Lors de la création du mappage des sources d'événements, le démarrage de l'interrogation des événements depuis le flux peut prendre plusieurs minutes.

- Lors des mises à jour du mappage des sources d'événements, l'arrêt et le redémarrage de l'interrogation des événements depuis le flux peuvent prendre plusieurs minutes.

Ce comportement signifie que si vous spécifiez `LATEST` comme position de départ du flux, le mappage des sources d'événements peut manquer des événements lors de la création ou des mises à jour. Pour vous assurer de ne manquer aucun événement, spécifiez la position de départ du flux comme `TRIM_HORIZON` ou `AT_TIMESTAMP`.

## CloudWatch Métriques Amazon

Lambda émet la métrique `OffsetLag` pendant que votre fonction traite les registres. La valeur de cette métrique est la différence de décalage entre le dernier enregistrement inscrit dans la rubrique source de l'événement Kafka et le dernier enregistrement traité par le groupe de consommateurs de votre fonction. Vous pouvez utiliser `OffsetLag` pour estimer la latence entre le moment où un enregistrement est ajouté et celui où votre groupe de consommateurs le traite.

Une tendance à la hausse de `OffsetLag` peut indiquer des problèmes liés aux sondeurs dans le groupe de consommateurs de votre fonction. Pour de plus amples informations, veuillez consulter Afficher les métriques des fonctions Lambda.

## Mise à l'échelle automatique de la source MSK d'événements Amazon

Lorsque vous créez initialement une source d'MSKévénements Amazon, Lambda alloue un consommateur pour traiter toutes les partitions du sujet Kafka. Chaque consommateur dispose de plusieurs processeurs exécutés en parallèle pour gérer des charges de travail accrues. Lambda augmente ou diminue automatiquement le nombre de consommateurs, en fonction de la charge de travail. Pour préserver l'ordre des messages dans chaque partition, le nombre maximum de consommateurs est de un par partition dans la rubrique.

Toutes les minutes, Lambda évalue le décalage de consommateurs de toutes les partitions dans la rubrique. Si le décalage est trop élevé, la partition reçoit des messages plus rapidement que Lambda ne peut les traiter. Si nécessaire, Lambda ajoute ou supprime des consommateurs dans la rubrique. Le processus de mise à l'échelle consistant à ajouter ou à supprimer des consommateurs a lieu dans les trois minutes suivant l'évaluation.

Si votre fonction Lambda cible est limitée, Lambda réduit le nombre de consommateurs. Cette action réduit la charge de travail de la fonction en diminuant le nombre de messages que les consommateurs peuvent échanger avec la fonction.

Pour contrôler le débit de votre rubrique Kafka, consultez la Métrique du décalage de délai Lambda émet pendant que votre fonction traite les enregistrements.

Pour vérifier le nombre d'invocations de fonctions qui se produisent en parallèle, vous pouvez également surveiller les métriques de simultanéité pour votre fonction.

## Utilisation du filtrage des événements avec une source d'MSKévénements Amazon

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources MSK d'événements Amazon.

Rubriques

- Principes de base du filtrage des MSK événements Amazon

### Principes de base du filtrage des MSK événements Amazon

Supposons qu'un producteur rédige des messages pour un sujet de votre MSK cluster Amazon, soit dans un JSON format valide, soit sous forme de chaînes simples. Un exemple d'enregistrement ressemblerait à ce qui suit, avec le message converti en chaîne encodée en Base64 dans le champ `value`.

```
{
    "mytopic-0":[
        {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[]
        }
    ]
}
```

Supposons que votre producteur Apache Kafka rédige des messages pour votre sujet dans le JSON format suivant.

```
{
    "device_ID": "AB1234",
    "session":{
        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}
```

Vous pouvez utiliser la clé `value` pour filtrer les enregistrements. Supposons que vous vouliez filtrer uniquement les enregistrements où `device_ID` commence par les lettres AB. L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "value": {
        "device_ID": [ { "prefix": "AB" } ]
      }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Avec AmazonMSK, vous pouvez également filtrer les enregistrements dont le message est une chaîne simple. Supposons que vous vouliez ignorer les messages dont la chaîne est « error ». L'objet `FilterCriteria` se présente comme suit.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
        }
    ]
```

```
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "value": [
        {
        "anything-but": [ "error" ]
        }
    ]
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Pour ajouter ce filtre AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

MSKLes messages Amazon doivent être des chaînes codées en UTF -8, soit des chaînes simples, soit au JSON format. C'est parce que Lambda décode les tableaux d'octets MSK Amazon UTF en -8 avant d'appliquer des critères de filtre. Si vos messages utilisent un autre encodage, tel que UTF -16 ouASCII, ou si le format du message ne correspond pas au `FilterCriteria` format, Lambda traite uniquement les filtres de métadonnées. Le tableau suivant résume le comportement spécifique :

| Format du du message entrant | Modèle de filtre de format pour les propriétés des messages | Action obtenue. |
|---|---|---|
| Chaîne de texte brut | Chaîne de texte brut | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne de texte brut | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Chaîne de texte brut | Valide JSON | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Chaîne de texte brut | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de |

| Format du du message entrant | Modèle de filtre de format pour les propriétés des messages | Action obtenue. |
| --- | --- | --- |
| | | métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne non codée en UTF -8 | JSON, chaîne simple ou sans motif | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |

## Capture des lots rejetés pour une source d'événements Amazon MSK

Pour retenir les enregistrements des invocations de mappage de sources d'événements qui ont échoué, ajoutez une destination au mappage des sources d'événements de votre fonction. Chaque enregistrement envoyé à la destination est un JSON document contenant des métadonnées relatives à l'échec de l'invocation. Vous pouvez configurer n'importe quel SNS sujet Amazon, SQS file d'attente Amazon ou compartiment S3 comme destination. Votre rôle d'exécution doit disposer d'autorisations pour la destination :

- Pour les SQS destinations : sqs : SendMessage
- Pour les SNS destinations : SNS:Publish
- Pour les destinations du compartiment S3 : s3 : PutObject et s3 : ListBuckets

Vous devez déployer un VPC point de terminaison pour votre service de destination en cas de défaillance au sein de votre MSK cluster VPC Amazon.

En outre, si vous avez configuré une KMS clé sur votre destination, Lambda a besoin des autorisations suivantes en fonction du type de destination :

- Si vous avez activé le chiffrement avec votre propre KMS clé pour une destination S3, kms : GenerateDataKey est obligatoire. Si la KMS clé et la destination du compartiment S3 se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser kms :. GenerateDataKey

- Si vous avez activé le chiffrement avec votre propre KMS clé de SQS destination, KMS:Decrypt et kms : sont obligatoires. GenerateDataKey Si la KMS clé et la destination de la SQS file d'attente se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser KMS:Decrypt, kms :GenerateDataKey, kms : et kms :. DescribeKey ReEncrypt
- Si vous avez activé le chiffrement avec votre propre KMS clé de SNS destination, KMS:Decrypt et kms : sont obligatoires. GenerateDataKey Si la KMS clé et la destination du SNS sujet se trouvent dans un compte différent de celui de votre fonction Lambda et de votre rôle d'exécution, configurez la KMS clé pour qu'elle approuve le rôle d'exécution afin d'autoriser KMS:Decrypt, kms :GenerateDataKey, kms : et kms :. DescribeKey ReEncrypt

## Configuration de destinations en cas de panne pour le mappage d'une source MSK d'événements Amazon

Pour configurer une destination en cas de panne à l'aide de la console, procédez comme suit :

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add destination (Ajouter une destination).
4. Pour Source, choisissez Invocation du mappage des sources d'événements.
5. Pour le mappage des sources d'événements, choisissez une source d'événements configurée pour cette fonction.
6. Pour Condition, sélectionnez En cas d'échec. Pour les invocations de mappage des sources d'événements, il s'agit de la seule condition acceptée.
7. Pour Type de destination, choisissez le type de destination auquel Lambda envoie les enregistrements d'invocation.
8. Pour Destination, choisissez une ressource.
9. Choisissez Save (Enregistrer).

Vous pouvez également configurer une destination en cas de panne à l'aide du AWS CLI. Par exemple, la create-event-source-mappingcommande suivante ajoute un mappage de source d'événement avec une destination SQS en cas de défaillance à `MyFunction` :

```
aws lambda create-event-source-mapping \
```

```
--function-name "MyFunction" \
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

La update-event-source-mappingcommande suivante ajoute une destination S3 en cas de défaillance à la source d'événements associée à l'entrée `uuid` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Pour supprimer une destination, entrez une chaîne vide comme argument du paramètre `destination-config` :

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

SNSet SQS exemple d'enregistrement d'invocation

L'exemple suivant montre ce que Lambda envoie à un SNS sujet ou à une destination de SQS file d'attente en cas d'échec d'un appel de source d'événement Kafka. Chacune des clés sous `recordsInfo` contient à la fois le sujet et la partition Kafka, séparés par un trait d'union. Par exemple, pour la clé`"Topic-0"`, `Topic` est la rubrique Kafka, et `0` est la partition. Pour chaque sujet et chaque partition, vous pouvez utiliser les décalages et les données d'horodatage pour trouver les enregistrements d'invocation d'origine.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
```

```
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    }
}
```

Exemple d'enregistrement d'appel de destination S3

Pour les destinations S3, Lambda envoie l’intégralité de l’enregistrement d’invocation ainsi que les métadonnées à la destination. L’exemple suivant montre que Lambda envoie vers une destination de compartiment S3 en cas d’échec d’une invocation de source d’événement Kafka. Outre tous les champs de l'exemple précédent pour SQS et pour les SNS destinations, le `payload` champ contient l'enregistrement d'appel d'origine sous forme de JSON chaîne échappée.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
```

```
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    },
    "payload": "<Whole Event>" // Only available in S3
}
```

> **Tip**
>
> Nous vous recommandons d’activer la gestion des versions S3 sur votre compartiment de destination.

# Tutoriel : Utilisation d'un mappage de source d'MSKévénements Amazon pour appeler une fonction Lambda

Dans ce didacticiel, vous allez effectuer les opérations suivantes :

- Créez une fonction Lambda dans le même AWS compte qu'un cluster Amazon MSK existant.
- Configurez le réseau et l'authentification pour que Lambda communique avec Amazon. MSK
- Configurez un mappage des sources d'MSKévénements Lambda Amazon, qui exécute votre fonction Lambda lorsque des événements apparaissent dans la rubrique.

Une fois ces étapes terminées, lorsque des événements seront envoyés à AmazonMSK, vous pourrez configurer une fonction Lambda pour traiter ces événements automatiquement avec votre propre code Lambda personnalisé.

Que pouvez-vous faire avec cette fonctionnalité ?

Exemple de solution : utilisez un mappage des sources d'MSKévénements pour fournir des scores en direct à vos clients.

Imaginons le scénario suivant : votre entreprise héberge une application Web dans laquelle vos clients peuvent consulter des informations sur des événements en direct, tels que des matchs de sport. Les informations mises à jour du jeu sont fournies à votre équipe via un sujet Kafka sur AmazonMSK. Vous souhaitez concevoir une solution qui utilise les mises à jour du MSK sujet pour fournir une vue actualisée de l'événement en direct aux clients au sein d'une application que vous développez. Vous avez opté pour l'approche de conception suivante : vos applications clientes communiqueront avec un backend sans serveur hébergé dans. AWS Les clients se connecteront via des sessions Websocket à l'aide d'Amazon API Gateway WebSocket API.

Dans cette solution, vous avez besoin d'un composant qui lit les MSK événements, exécute une logique personnalisée pour préparer ces événements pour la couche application, puis transmet ces informations à la API passerelleAPI. Vous pouvez implémenter ce composant en fournissant votre logique personnalisée dans une fonction Lambda, puis en l'appelant à l'aide d'un mappage de source d'MSKévénements AWS Lambda Amazon. AWS Lambda

Pour plus d'informations sur la mise en œuvre de solutions à l'aide d'Amazon API Gateway WebSocket API, consultez les WebSocket APIdidacticiels de la documentation de API Gateway.

## Prérequis

Un AWS compte avec les ressources préconfigurées suivantes :

Pour remplir ces conditions préalables, nous vous recommandons de suivre Getting started using Amazon MSK dans la MSK documentation Amazon.

- Un MSK cluster Amazon. Consultez Créer un MSK cluster Amazon dans Getting started using Amazon MSK.
- La configuration suivante :
  - Assurez-vous que l'authentification IAM basée sur les rôles est activée dans les paramètres de sécurité de votre cluster. Cela améliore votre sécurité en limitant votre fonction Lambda à accéder uniquement aux MSK ressources Amazon nécessaires. Ceci est activé par défaut sur les nouveaux MSK clusters Amazon.
  - Assurez-vous que l'accès public est désactivé dans les paramètres réseau de votre cluster. Restreindre l'accès à Internet de votre MSK cluster Amazon améliore votre sécurité en limitant le nombre d'intermédiaires qui traitent vos données. Ceci est activé par défaut sur les nouveaux MSK clusters Amazon.
- Une rubrique Kafka dans votre MSK cluster Amazon à utiliser pour cette solution. Consultez Créer un sujet dans Getting started using Amazon MSK.
- Un hôte administrateur Kafka configuré pour récupérer les informations de votre cluster Kafka et envoyer des événements Kafka à votre sujet à des fins de test, par exemple une EC2 instance Amazon sur laquelle l'administrateur Kafka et la bibliothèque CLI Amazon sont installés. MSK IAM Consultez la section Création d'un ordinateur client dans Getting started using Amazon MSK.

Une fois que vous avez configuré ces ressources, collectez les informations suivantes à partir de votre AWS compte pour confirmer que vous êtes prêt à continuer.

- Le nom de votre MSK cluster Amazon. Vous pouvez trouver ces informations dans la MSK console Amazon.
- Le clusterUUID, qui fait partie du ARN MSK cluster Amazon, que vous pouvez trouver dans la MSK console Amazon. Suivez les procédures décrites dans Listing clusters dans la MSK documentation Amazon pour trouver ces informations.
- Les groupes de sécurité associés à votre MSK cluster Amazon. Vous pouvez trouver ces informations dans la MSK console Amazon. Dans les étapes suivantes, appelez-les votre *`clusterSecurityGroups`*.

- L'identifiant de l'Amazon VPC contenant votre MSK cluster Amazon. Vous pouvez trouver ces informations en identifiant les sous-réseaux associés à votre MSK cluster Amazon dans la MSK console Amazon, puis en identifiant l'Amazon VPC associé au sous-réseau dans la console AmazonVPC.
- Le nom du sujet Kafka utilisé dans votre solution. Vous pouvez trouver ces informations en appelant votre MSK cluster Amazon avec le Kafka `topics` CLI depuis votre hôte administrateur Kafka. Pour plus d'informations sur les sujetsCLI, consultez la section Ajouter et supprimer des sujets dans la documentation de Kafka.
- Le nom d'un groupe de consommateurs pour votre sujet Kafka, adapté à une utilisation par votre fonction Lambda. Ce groupe peut être créé automatiquement par Lambda, vous n'avez donc pas besoin de le créer avec le Kafka. CLI Si vous devez gérer vos groupes de consommateurs, pour en savoir plus sur les groupes de consommateursCLI, consultez la section Gestion des groupes de consommateurs dans la documentation de Kafka.

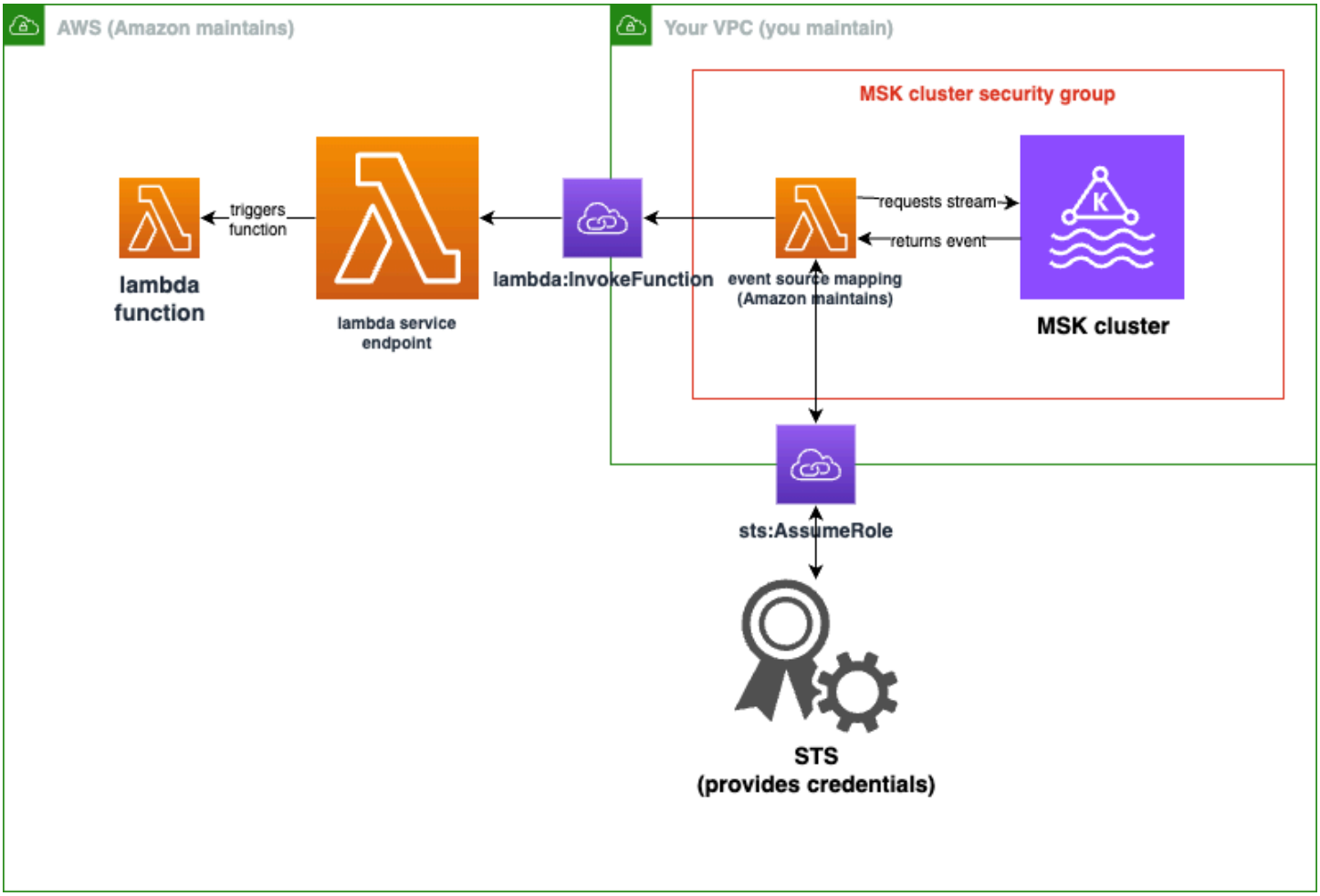
Les autorisations suivantes dans votre AWS compte :

- Autorisation de créer et de gérer une fonction Lambda.
- Autorisation de créer des IAM politiques et de les associer à votre fonction Lambda.
- Autorisation de créer des VPC points de terminaison Amazon et de modifier la configuration réseau dans l'Amazon VPC hébergeant votre MSK cluster Amazon.

## Configurer la connectivité réseau pour que Lambda communique avec Amazon MSK

AWS PrivateLink À utiliser pour connecter Lambda et Amazon. MSK Vous pouvez le faire en créant une interface Amazon VPC endpoints dans la VPC console Amazon. Pour plus d'informations sur la configuration réseau, consultezthe section called “Configuration réseau”.

Lorsqu'un mappage de source d'MSKévénements Amazon s'exécute pour le compte d'une fonction Lambda, il assume le rôle d'exécution de la fonction Lambda. Ce IAM rôle autorise le mappage à accéder à des ressources sécurisées parIAM, telles que votre MSK cluster Amazon. Bien que les composants partagent un rôle d'exécution, le MSK mappage Amazon et votre fonction Lambda ont des exigences de connectivité distinctes pour leurs tâches respectives, comme le montre le schéma suivant.




Le mappage de votre source d'événements appartient au groupe de sécurité de votre MSK cluster Amazon. Au cours de cette étape de mise en réseau, créez des VPC points de terminaison Amazon à partir de votre MSK cluster Amazon VPC pour connecter le mappage des sources d'événements au Lambda STS et aux services. Sécurisez ces points de terminaison pour accepter le trafic provenant du groupe de sécurité de votre MSK cluster Amazon. Ajustez ensuite les groupes de sécurité du MSK cluster Amazon pour permettre au mappage des sources d'événements de communiquer avec le MSK cluster Amazon.

Vous pouvez configurer les étapes suivantes à l'aide du AWS Management Console.

Pour configurer les VPC points de terminaison Amazon de l'interface afin de connecter Lambda et Amazon MSK

1. Créez un groupe de sécurité pour votre interface Amazon VPC endpoints, *`endpointSecurityGroup`*, qui autorise le TCP trafic entrant sur 443 depuis *`clusterSecurityGroups`*. Suivez la procédure décrite dans Créer un groupe de sécurité dans

la EC2 documentation Amazon pour créer un groupe de sécurité. Suivez ensuite la procédure décrite dans Ajouter des règles à un groupe de sécurité dans la EC2 documentation Amazon pour ajouter les règles appropriées.

Créez un groupe de sécurité avec les informations suivantes :

Lorsque vous ajoutez vos règles de trafic entrant, créez une règle pour chaque groupe de sécurité dans *clusterSecurityGroups*. Pour chaque règle :

- Pour Type, sélectionnez HTTPS.
- Pour Source, sélectionnez l'une des *clusterSecurityGroups*.

2. Créez un point de terminaison connectant le service Lambda à l'Amazon VPC contenant votre cluster AmazonMSK. Suivez la procédure décrite dans Créer un point de terminaison d'interface.

   Créez un point de terminaison d'interface avec les informations suivantes :

   - Pour le nom du service, sélectionnez`com.amazonaws.`*regionName*`.lambda`, où *regionName* héberge votre fonction Lambda.
   - Pour VPC, sélectionnez l'Amazon VPC contenant votre MSK cluster Amazon.
   - Pour les groupes de sécurité, sélectionnez *endpointSecurityGroup*, que vous avez créé précédemment.
   - Pour les sous-réseaux, sélectionnez les sous-réseaux qui hébergent votre cluster AmazonMSK.
   - Pour Policy, fournissez le document de politique suivant, qui sécurise le point de terminaison afin qu'il soit utilisé par le principal de service Lambda pour `lambda:InvokeFunction` l'action.

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
```

```
    ]
}
```

- Assurez-vous que le DNS nom Enable reste défini.

3. Créez un point de terminaison connectant le AWS STS service à l'Amazon VPC contenant votre MSK cluster Amazon. Suivez la procédure décrite dans Créer un point de terminaison d'interface.

   Créez un point de terminaison d'interface avec les informations suivantes :

   - Pour Nom du service, sélectionnez AWS STS.
   - Pour VPC, sélectionnez l'Amazon VPC contenant votre MSK cluster Amazon.
   - Pour les groupes de sécurité, sélectionnez *endpointSecurityGroup*.
   - Pour les sous-réseaux, sélectionnez les sous-réseaux qui hébergent votre cluster AmazonMSK.
   - Pour Policy, fournissez le document de politique suivant, qui sécurise le point de terminaison afin qu'il soit utilisé par le principal de service Lambda pour `sts:AssumeRole` l'action.

   ```
   {
       "Statement": [
           {
               "Action": "sts:AssumeRole",
               "Effect": "Allow",
               "Principal": {
                   "Service": [
                       "lambda.amazonaws.com"
                   ]
               },
               "Resource": "*"
           }
       ]
   }
   ```

   - Assurez-vous que le DNS nom Enable reste défini.

4. Pour chaque groupe de sécurité associé à votre MSK cluster Amazon, c'est-à-dire dans *clusterSecurityGroups*, autorisez ce qui suit :

   - Autoriser tout le TCP trafic entrant et sortant sur le 9098 à tous *clusterSecurityGroups*, y compris à l'intérieur de lui-même.

- Autoriser tout le TCP trafic sortant sur le 443.

Une partie de ce trafic est autorisée par les règles des groupes de sécurité par défaut. Par conséquent, si votre cluster est attaché à un seul groupe de sécurité et que ce groupe possède des règles par défaut, des règles supplémentaires ne sont pas nécessaires. Pour ajuster les règles des groupes de sécurité, suivez les procédures décrites dans Ajouter des règles à un groupe de sécurité dans la EC2 documentation Amazon.

Ajoutez des règles à vos groupes de sécurité avec les informations suivantes :

- Pour chaque règle entrante ou sortante pour le port 9098, fournissez
  - Pour Type, sélectionnez Personnalisé TCP.
  - Pour la plage de ports, indiquez 9098.
  - Pour Source, indiquez l'un des *clusterSecurityGroups*.
- Pour chaque règle entrante pour le port 443, sélectionnez HTTPSType.

## Créez un IAM rôle pour que Lambda puisse lire un extrait de votre rubrique Amazon MSK

Identifiez les exigences d'authentification pour que Lambda puisse lire dans votre rubrique MSK Amazon, puis définissez-les dans une politique. Créez un rôle, *lambdaAuthRole*, qui autorise Lambda à utiliser ces autorisations. Autorisez les actions sur votre MSK cluster Amazon à l'aide d'`kafka-cluster`IAMactions. Autorisez ensuite Lambda à effectuer les EC2 actions Amazon et MSK `kafka` Amazon nécessaires pour découvrir votre MSK cluster Amazon et s'y connecter, ainsi que des CloudWatch actions afin que Lambda puisse enregistrer ce qu'il a fait.

Pour décrire les exigences d'authentification pour que Lambda puisse lire depuis Amazon MSK

1. Rédiger un document de IAM politique (un JSON document), *clusterAuthPolicy*, qui permet à Lambda de lire un extrait de votre sujet Kafka dans votre MSK cluster Amazon en utilisant votre groupe de consommateurs Kafka. Lambda nécessite qu'un groupe de consommateurs Kafka soit défini lors de la lecture.

   Modifiez le modèle suivant pour l'aligner sur vos prérequis :

   ```
   {
       "Version": "2012-10-17",
   ```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:DescribeGroup",
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeClusterDynamicConfiguration"
            ],
            "Resource": [
                "arn:aws:kafka:region:account-id:cluster/mskClusterName/cluster-
uuid",
                "arn:aws:kafka:region:account-id:topic/mskClusterName/cluster-
uuid/mskTopicName",
                "arn:aws:kafka:region:account-id:group/mskClusterName/cluster-
uuid/mskGroupName"
            ]
        }
    ]
}
```

Pour plus d'informations, consultezthe section called “Authentification basée sur les rôles IAM”. Lorsque vous rédigez votre politique :

- Dans *region* and *account-id*, indiquez ceux qui hébergent votre MSK cluster Amazon.
- Dans *mskClusterName*, indiquez le nom de votre MSK cluster Amazon.
- Dans *cluster-uuid*, fournissez le UUID dans le ARN pour votre MSK cluster Amazon.
- Dans *mskTopicName*, indiquez le nom de votre sujet Kafka.
- Dans *mskGroupName*, indiquez le nom de votre groupe de consommateurs Kafka.

2. Identifiez AmazonMSK, Amazon EC2 et CloudWatch les autorisations requises pour que Lambda découvre et connecte votre MSK cluster Amazon, puis enregistrez ces événements.

   La politique `AWSLambdaMSKExecutionRole` gérée définit de manière permissive les autorisations requises. Utilisez-le dans les étapes suivantes.

   Dans un environnement de production, déterminez si vous `AWSLambdaMSKExecutionRole` souhaitez restreindre votre politique de rôle d'exécution sur la base du principe du moindre privilège, puis rédigez une politique pour votre rôle qui remplace cette politique gérée.

Pour plus de détails sur le langage de IAM politique, consultez la IAMdocumentation.

Maintenant que vous avez rédigé votre document de stratégie, créez-en une IAM afin de pouvoir l'associer à votre rôle. Vous pouvez le faire à l'aide de la console en suivant la procédure suivante.

Pour créer une IAM politique à partir de votre document de stratégie

1. Connectez-vous à la IAM console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation de gauche, choisissez Politiques.
3. Choisissez Create Policy (Créer une politique).
4. Dans la section Éditeur de politiques, choisissez l'JSONoption.
5. Coller *clusterAuthPolicy*.
6. Lorsque vous avez fini d'ajouter des autorisations à la politique, choisissez Suivant.
7. Sur la page Vérifier et créer, tapez un Nom de politique et une Description (facultative) pour la politique que vous créez. Vérifiez les Autorisations définies dans cette politique pour voir les autorisations accordées par votre politique.
8. Choisissez Create policy (Créer une politique) pour enregistrer votre nouvelle politique.

Pour plus d'informations, consultez la section Création IAM de politiques dans la IAM documentation.

Maintenant que vous disposez des IAM politiques appropriées, créez un rôle et associez-le à celui-ci. Vous pouvez le faire à l'aide de la console en suivant la procédure suivante.

Pour créer un rôle d'exécution dans la IAM console

1. Ouvrez la page Rôles dans la IAM console.
2. Sélectionnez Créer un rôle.
3. Sous Trusted entity type (Type d'entité approuvée), choisissez service AWS .
4. Sous Cas d'utilisation, choisissez Lambda.
5. Choisissez Suivant.
6. Sélectionnez les stratégies suivantes :
   - *clusterAuthPolicy*
   - `AWSLambdaMSKExecutionRole`

7. Choisissez Suivant.
8. Pour Nom du rôle, entrez *lambdaAuthRole* puis choisissez Create role.

Pour de plus amples informations, veuillez consulter the section called “Rôle d'exécution (autorisations pour les fonctions d'accéder à d'autres ressources)”.

## Créez une fonction Lambda pour lire un extrait de votre rubrique Amazon MSK

Créez une fonction Lambda configurée pour utiliser votre IAM rôle. Vous pouvez créer votre fonction Lambda à l'aide de la console.

Pour créer une fonction Lambda à l'aide de votre configuration d'authentification

1. Ouvrez la console Lambda et sélectionnez Create function dans l'en-tête.
2. Sélectionnez Créer à partir de zéro.
3. Pour Nom de la fonction, indiquez le nom approprié de votre choix.
4. Pour Runtime, choisissez la dernière version prise en charge de `Node.js` pour utiliser le code fourni dans ce didacticiel.
5. Choisissez Modifier le rôle d'exécution par défaut.
6. Sélectionnez Utiliser un rôle existant.
7. Pour Rôle existant, sélectionnez *lambdaAuthRole*.

Dans un environnement de production, vous devez généralement ajouter des politiques supplémentaires au rôle d'exécution de votre fonction Lambda afin de traiter de manière significative vos événements Amazon. MSK Pour plus d'informations sur l'ajout de politiques à votre rôle, consultez la section Ajouter ou supprimer des autorisations d'identité dans la IAM documentation.

## Créez une source d'événement mappée à votre fonction Lambda

Le mappage de votre source d'MSKévénements Amazon fournit au service Lambda les informations nécessaires pour appeler votre Lambda lorsque des événements Amazon appropriés se produisent. MSK Vous pouvez créer un MSK mappage Amazon à l'aide de la console. Créez un déclencheur Lambda, puis le mappage des sources d'événements est automatiquement configuré.

Pour créer un déclencheur Lambda (et un mappage des sources d'événements)

1. Accédez à la page de présentation de votre fonction Lambda.

2. Dans la section de présentation des fonctions, choisissez Ajouter un déclencheur en bas à gauche.
3. Dans le menu déroulant Sélectionnez une source, sélectionnez Amazon MSK.
4. Ne configurez pas l'authentification.
5. Pour le MSKcluster, sélectionnez le nom de votre cluster.
6. Pour Taille du lot, entrez 1. Cette étape facilite le test de cette fonctionnalité et ne constitue pas une valeur idéale en production.
7. Dans le champ Nom du sujet, indiquez le nom de votre sujet Kafka.
8. Pour l'identifiant du groupe de consommateurs, indiquez l'identifiant de votre groupe de consommateurs Kafka.

## Mettez à jour votre fonction Lambda pour lire vos données de streaming

Lambda fournit des informations sur les événements Kafka via le paramètre de méthode d'événement. Pour un exemple de structure d'un MSK événement Amazon, consultezthe section called “ Exemple d’évènement”. Après avoir compris comment interpréter les MSK événements Amazon transférés par Lambda, vous pouvez modifier le code de votre fonction Lambda pour utiliser les informations qu'ils fournissent.

Fournissez le code suivant à votre fonction Lambda pour enregistrer le contenu d'un MSK événement Lambda Amazon à des fins de test :

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda en utilisant. NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to
 process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
 logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }


}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
 event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda en utilisant. JavaScript

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

### Utilisation d'un MSK événement Amazon avec Lambda en utilisant. PHP

```
<?php
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

// using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kafka\KafkaEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }
```

```
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

### Python

#### SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Python.

```
import base64

def lambda_handler(event, context):
    # Iterate through keys
    for key in event['records']:
        print('Key:', key)
        # Iterate through records
        for record in event['records'][key]:
            print('Record:', record)
            # Decode base64
            msg = base64.b64decode(record['value']).decode('utf-8')
            print('Message:', msg)
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
  # Iterate through keys
  event['records'].each do |key, records|
    puts "Key: #{key}"

    # Iterate through records
    records.each do |record|
      puts "Record: #{record}"

      # Decode base64
      msg = Base64.decode64(record['value'])
      puts "Message: #{msg}"
    end
  end
end
```

Vous pouvez fournir un code de fonction à votre Lambda à l'aide de la console.

Pour mettre à jour le code de votre fonction Lambda

1. Accédez à la page de présentation de votre fonction Lambda.
2. Cliquez sur l'onglet Code.
3. Entrez le code fourni dans le code sourceIDE.
4. Dans la barre de navigation du code source, choisissez Deploy.

## Testez votre fonction Lambda pour vérifier qu'elle est connectée à votre rubrique Amazon MSK

Vous pouvez désormais vérifier si votre Lambda est invoqué par la source d'événements en consultant les journaux d'événements. CloudWatch

Pour vérifier si votre fonction Lambda est invoquée

1. Utilisez votre hôte d'administration Kafka pour générer des événements Kafka à l'aide du. `kafka-console-producer` CLI Pour plus d'informations, voir Écrire des événements dans le sujet dans la documentation de Kafka. Envoyez suffisamment d'événements pour remplir le lot défini en fonction de la taille du lot pour le mappage de votre source d'événements défini à l'étape précédente, sinon Lambda attendra d'autres informations pour appeler.
2. Si votre fonction s'exécute, Lambda écrit ce qui s'est passé à. CloudWatch Dans la console, accédez à la page détaillée de votre fonction Lambda.
3. Sélectionnez l'onglet Configuration.
4. Dans la barre latérale, sélectionnez Outils de surveillance et d'exploitation.
5. Identifiez le groupe de CloudWatch journaux sous Configuration de la journalisation. Le groupe de journaux doit commencer par`/aws/lambda`. Choisissez le lien vers le groupe de journaux.
6. Dans la CloudWatch console, examinez les événements du journal pour les événements du journal que Lambda a envoyés au flux de journal. Identifiez s'il existe des événements de journal contenant le message de votre événement Kafka, comme dans l'image suivante. Si tel est le cas, vous avez connecté avec succès une fonction Lambda à Amazon à l'MSKaide d'un mappage de source d'événements Lambda.

# Utilisation AWS Lambda avec Amazon RDS

Vous pouvez connecter une fonction Lambda à une base de données Amazon Relational Database Service (AmazonRDS) directement et via un proxy Amazon. RDS Les connexions directes sont utiles dans les scénarios simples, et les proxys sont recommandés pour la production. Un proxy de base de données gère un groupe de connexions partagées à la base de données qui permet à votre fonction d'atteindre des niveaux de simultanéité élevés sans épuiser les connexions de base de données.

Nous vous recommandons d'utiliser Amazon RDS Proxy pour les fonctions Lambda qui établissent fréquemment de courtes connexions à des bases de données ou qui ouvrent et ferment un grand nombre de connexions de base de données. Pour plus d'informations, consultez la section Connexion automatique d'une fonction Lambda à une instance de base de données dans le manuel Amazon Relational Database Service Developer Guide.

## Configuration de votre fonction pour qu'elle fonctionne avec les RDS ressources

Dans la console Lambda, vous pouvez approvisionner et configurer des instances de RDS base de données Amazon et des ressources proxy. Vous pouvez le faire en accédant aux RDSbases de données sous l'onglet Configuration. Vous pouvez également créer et configurer des connexions aux fonctions Lambda dans la console AmazonRDS. Lorsque vous configurez une instance RDS de base de données à utiliser avec Lambda, tenez compte des critères suivants :

- Pour vous connecter à une base de données, votre fonction doit se trouver sur le même Amazon que celui sur VPC lequel votre base de données s'exécute.
- Vous pouvez utiliser les RDS bases de données Amazon avec les SQL moteurs My, MariaDB, SQL Postgre ou SQL Microsoft Server.
- Vous pouvez également utiliser des clusters de base de données Aurora avec les SQL moteurs My SQL ou Postgre.
- Vous devez fournir un secret Secrets Manager pour l'authentification de la base de données.
- Un IAM rôle doit autoriser l'utilisation du secret, et une politique de confiance doit autoriser Amazon RDS à assumer ce rôle.
- Le IAM principal qui utilise la console pour configurer la RDS ressource Amazon et la connecter à votre fonction doit disposer des autorisations suivantes :

> **Note**
>
> Vous n'avez besoin des autorisations Amazon RDS Proxy que si vous configurez un RDS proxy Amazon pour gérer un pool de connexions à votre base de données.

Example politique d'autorisation

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds-db:connect",
        "rds:CreateDBProxy",
        "rds:CreateDBInstance",
        "rds:CreateDBSubnetGroup",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeDBProxies",
        "rds:DescribeDBProxyTargets",
        "rds:DescribeDBProxyTargetGroups",
        "rds:RegisterDBProxyTargets",
```

```
        "rds:ModifyDBInstance",
        "rds:ModifyDBProxy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:ListFunctions",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:AttachPolicy",
        "iam:CreateRole",
        "iam:CreatePolicy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds",
        "secretsmanager:CreateSecret"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon RDS facture un tarif horaire pour les proxys en fonction de la taille de l'instance de base de données. Consultez la section Tarification des RDS proxys pour plus de détails. Pour plus d'informations sur les connexions proxy en général, consultez la section Utilisation d'Amazon RDS Proxy dans le guide de RDS l'utilisateur Amazon.

> **Configuration de Lambda et Amazon RDS**
>
> Les RDS consoles Lambda et Amazon vous aideront à configurer automatiquement certaines des ressources requises pour établir une connexion entre Lambda et Amazon. RDS

## Connexion à une RDS base de données Amazon dans une fonction Lambda

L'exemple de code suivant montre comment implémenter une fonction Lambda qui se connecte à une base de données AmazonRDS. La fonction effectue une simple demande de base de données et renvoie le résultat.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
```

```
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
```

```
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
 RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
 event, Context context) {
        APIGatewayProxyResponseEvent response = new
 APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                    System.getenv("ProxyHostName"),
                    System.getenv("Port"),
                    System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
 DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
 token);
                 PreparedStatement statement =
 connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
```

```
            }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
                .region(Region.of(System.getenv("AWS_REGION")))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .resourceArn(System.getenv("ProxyHostName"))
                .secretArn(System.getenv("DBUserName"))
                .database(System.getenv("DBName"))
                .sql("SELECT 'RDS IAM Authentication'")
                .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
 rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {
```

```
  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
 the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
```

```
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
 body: string }> => {
    // Execute database flow
    const result = await dbOps();
```

```
    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. PHP

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
```

```
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']


# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
```

```
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'}  # Ensure you have the CA bundle for SSL
 connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}")  # Return an error message if an exception
 occurs
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
  port = ENV['Port']           # 3306
  user = ENV['DBUser']
  region = ENV['DBRegion']     # 'us-east-1'
```

```
  db_name = ENV['DBName']

  credentials = Aws::Credentials.new(
    ENV['AWS_ACCESS_KEY_ID'],
    ENV['AWS_SECRET_ACCESS_KEY'],
    ENV['AWS_SESSION_TOKEN']
  )
  rds_client = Aws::RDS::AuthTokenGenerator.new(
    region: region,
    credentials: credentials
  )

  token = rds_client.auth_token(
    endpoint: endpoint+ ':' + port,
    user_name: user,
    region: region
  )

  begin
    conn = Mysql2::Client.new(
      host: endpoint,
      username: user,
      password: token,
      port: port,
      database: db_name,
      sslca: '/var/task/global-bundle.pem',
      sslverify: true,
      enable_cleartext_plugin: true
    )
    a = 3
    b = 2
    result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
    puts result
    conn.close
    {
      statusCode: 200,
      body: result.to_json
    }
  rescue => e
    puts "Database connection failed due to #{e}"
  end
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
```

```
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
 aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
 SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
```

```
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    }))
}
```

## Traitement des notifications d'événements provenant d'Amazon RDS

Vous pouvez utiliser Lambda pour traiter les notifications d'événements provenant d'une base de données AmazonRDS. Amazon RDS envoie des notifications à une rubrique Amazon Simple

Notification Service (AmazonSNS), que vous pouvez configurer pour appeler une fonction Lambda. Amazon SNS intègre le message d'Amazon RDS dans son propre document d'événement et l'envoie à votre fonction.

Pour plus d'informations sur la configuration d'une RDS base de données Amazon pour envoyer des notifications, consultez Utiliser les notifications RDS d'événements Amazon.

Example RDSMessage Amazon lors d'un SNS événement Amazon


```
{
        "Records": [
          {
            "EventVersion": "1.0",
            "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:rds-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
            "EventSource": "aws:sns",
            "Sns": {
              "SignatureVersion": "1",
              "Timestamp": "2023-01-02T12:45:07.000Z",
              "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi
+tE/1+82j...65r==",
              "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
              "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
              "Message": "{\"Event Source\":\"db-instance\",\"Event Time\":\"2023-01-02
 12:45:06.000\",\"Identifier Link\":\"https://console.aws.amazon.com/rds/home?
region=eu-west-1#dbinstance:id=dbinstanceid\",\"Source ID\":\"dbinstanceid\",\"Event ID
\":\"http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_Events.html#RDS-
EVENT-0002\",\"Event Message\":\"Finished DB Instance backup\"}",
              "MessageAttributes": {},
              "Type": "Notification",
              "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?
Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
              "TopicArn":"arn:aws:sns:us-east-2:123456789012:sns-lambda",
              "Subject": "RDS Notification Message"
            }
          }
        ]
      }
```

# Tutoriel complet sur Lambda et Amazon RDS

- Utilisation d'une fonction Lambda pour accéder à une RDS base de données Amazon — Dans le guide de RDS l'utilisateur Amazon, découvrez comment utiliser une fonction Lambda pour écrire des données dans une base de données Amazon RDS via un proxy Amazon. RDS Votre fonction Lambda lit les enregistrements d'une SQS file d'attente Amazon et écrit de nouveaux éléments dans une table de votre base de données chaque fois qu'un message est ajouté.

# Traiter les notifications d'événements Amazon S3 avec Lambda

Vous pouvez utiliser Lambda pour traiter les notifications d'événement d'Amazon Simple Storage Service. Amazon S3 peut envoyer un événement à une fonction Lambda lors de la création ou de la suppression d'un objet. Vous configurez des paramètres de notification sur un compartiment et accordez à Amazon S3 l'autorisation d'appeler une fonction sur la stratégie d'autorisations basée sur une ressource de la fonction.

> **Warning**
>
> Si votre fonction Lambda utilise le même compartiment que celui qui la déclenche, la fonction risque de s'exécuter en boucle. Par exemple, si le compartiment déclenche une fonction chaque fois qu'un objet est chargé et que la fonction charge un objet dans le compartiment, la fonction se déclenche elle-même indirectement. Afin d'éviter cela, utilisez deux compartiments ou configurez le déclencheur pour qu'il s'applique uniquement à un préfixe utilisé pour les objets entrants.

Amazon S3 appelle votre fonction de manière asynchrone avec un événement contenant des détails sur l'objet. L'exemple suivant montre un événement envoyé par Amazon S3 lors du chargement d'un package de déploiement vers Amazon S3.

Example Evénement de notification Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-2",
      "eventTime": "2019-09-03T19:37:27.192Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
      },
      "requestParameters": {
        "sourceIPAddress": "205.255.255.255"
      },
      "responseElements": {
        "x-amz-request-id": "D82B88E5F771F645",
```

```
        "x-amz-id-2":
 "vlR7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "A3I5XTEXAMAI3E"
          },
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
        },
        "object": {
          "key": "b21b84d653bb07b05b1e6b33684dc11b",
          "size": 1305107,
          "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
          "sequencer": "0C0F6F405D6ED209E1"
        }
      }
    }
  ]
}
```

Pour appeler votre fonction, Amazon S3 a besoin d'une autorisation de la stratégie basée sur une ressource. Lorsque vous configurez un déclencheur Amazon S3 dans la console Lambda, cette dernière modifie la stratégie basée sur une ressource pour permettre à Amazon S3 d'appeler la fonction si le nom du compartiment et l'ID de compte correspondent. Si vous configurez la notification dans Amazon S3, vous utilisez le Lambda API pour mettre à jour la politique. Vous pouvez également utiliser le Lambda API pour accorder une autorisation à un autre compte ou restreindre l'autorisation à un alias désigné.

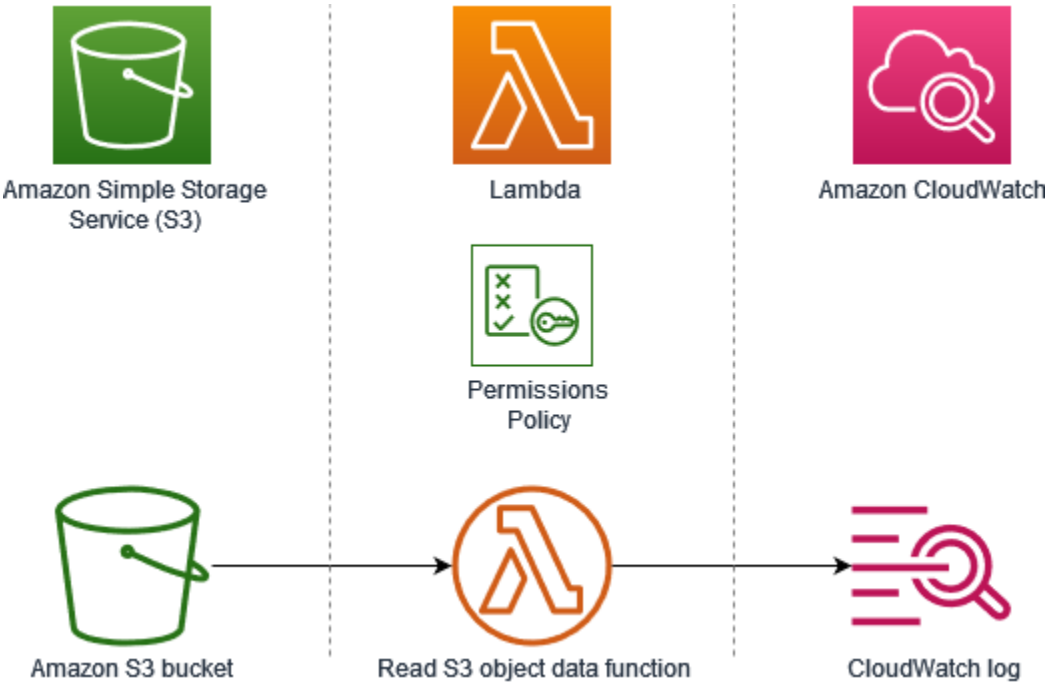
Si votre fonction utilise le AWS SDK pour gérer les ressources Amazon S3, elle a également besoin des autorisations Amazon S3 dans son rôle d'exécution.

Rubriques

- Didacticiel : utilisation d’un déclencheur Amazon S3 pour invoquer une fonction Lambda
- Didacticiel : Utilisation d'un déclencheur Amazon S3 pour créer des images miniatures

# Didacticiel : utilisation d'un déclencheur Amazon S3 pour invoquer une fonction Lambda

Dans ce didacticiel, vous allez utiliser la console pour créer une fonction Lambda et configurer un déclencheur pour un compartiment Amazon Simple Storage Service (Amazon S3). Chaque fois que vous ajoutez un objet à votre compartiment Amazon S3, votre fonction s'exécute et affiche le type d'objet dans Amazon CloudWatch Logs.




Ce didacticiel explique comment :

1. Créez un compartiment Amazon S3.
2. Créez une fonction Lambda qui renvoie le type d'objet des objets dans un compartiment Amazon S3.
3. Configurez un déclencheur Lambda qui invoque votre fonction lorsque des objets sont chargés dans votre compartiment.
4. Testez votre fonction, d'abord avec un événement fictif, puis en utilisant le déclencheur.

En suivant ces étapes, vous apprendrez à configurer une fonction Lambda pour qu'elle s'exécute chaque fois que des objets sont ajoutés ou supprimés d'un compartiment Amazon S3. Vous pouvez compléter ce didacticiel en n'utilisant que la AWS Management Console.

## Prérequis

### Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

### Création d'un utilisateur doté d'un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu'utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

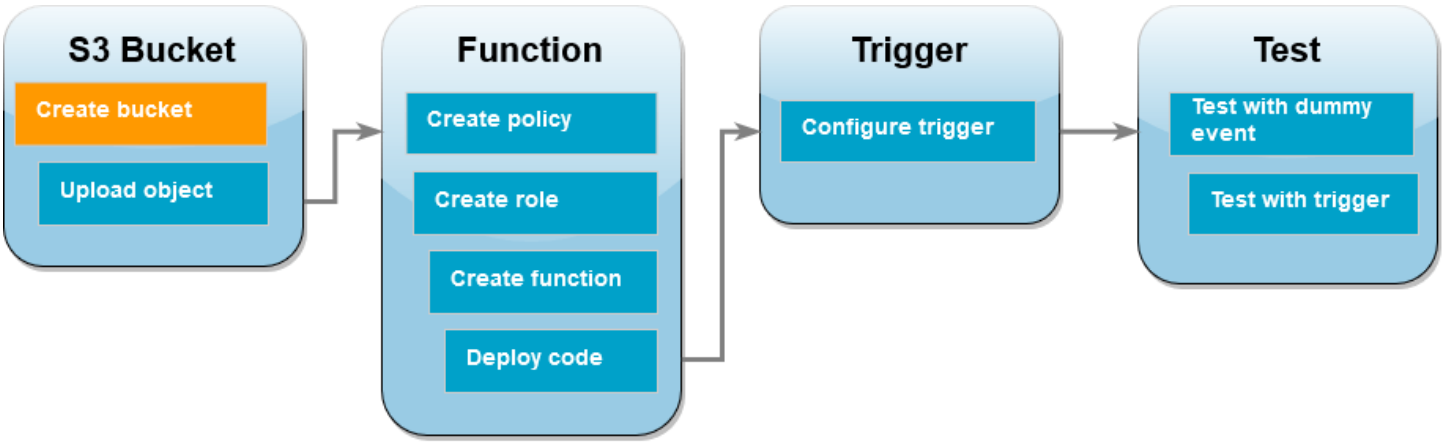
Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d'un ensemble d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l'utilisateur AWS IAM Identity Center .

## Créer un compartiment Amazon S3



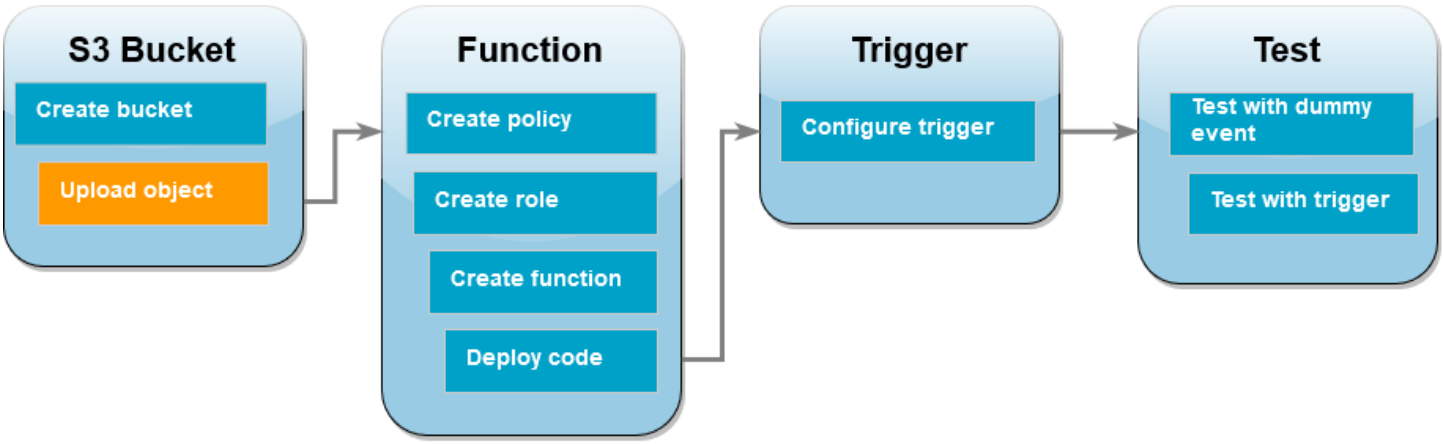

Pour créer un compartiment Amazon S3

1. Ouvrez la console Amazon S3 et sélectionnez la page Compartiments.
2. Choisissez Create bucket (Créer un compartiment).
3. Sous Configuration générale, procédez comme suit :

   a. Pour le nom du compartiment, saisissez un nom unique au monde qui respecte les règles de dénomination du compartiment Amazon S3. Les noms de compartiment peuvent contenir uniquement des lettres minuscules, des chiffres, de points (.) et des traits d'union (-).

   b. Choisissez une région dans Région AWS . Plus loin dans le didacticiel, vous devez créer votre fonction Lambda dans la même région.
4. Conservez les valeurs par défaut de toutes les autres options et choisissez Créer un compartiment.

## Charger un objet de test dans votre compartiment

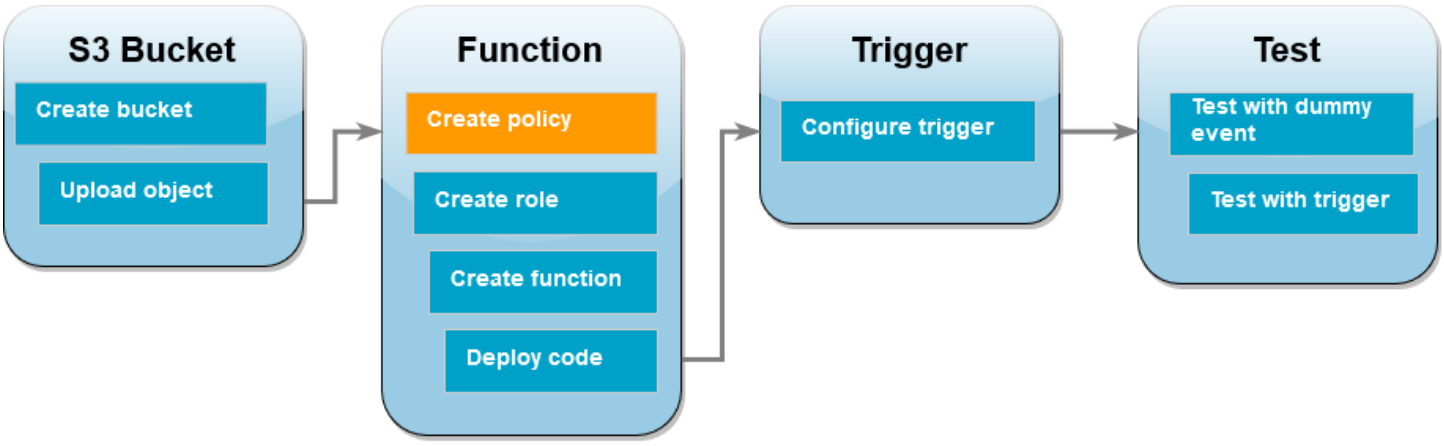


Pour charger un objet de test

1. Ouvrez la page Compartiments de la console Amazon S3 et choisissez le compartiment que vous avez créé à l'étape précédente.
2. Sélectionnez Charger.
3. Choisissez Ajouter des fichiers et sélectionnez l'objet que vous souhaitez télécharger. Vous pouvez sélectionner n'importe quel fichier (par exemple,`HappyFace.jpg`).
4. Choisissez Ouvrir, puis Charger.

Plus loin dans le didacticiel, vous testerez votre fonction Lambda à l'aide de cet objet.

## Créer une stratégie d'autorisations




Créez une politique d'autorisation qui permet à Lambda d'obtenir des objets depuis un compartiment Amazon S3 et d'écrire dans Amazon CloudWatch Logs.

Pour créer la politique

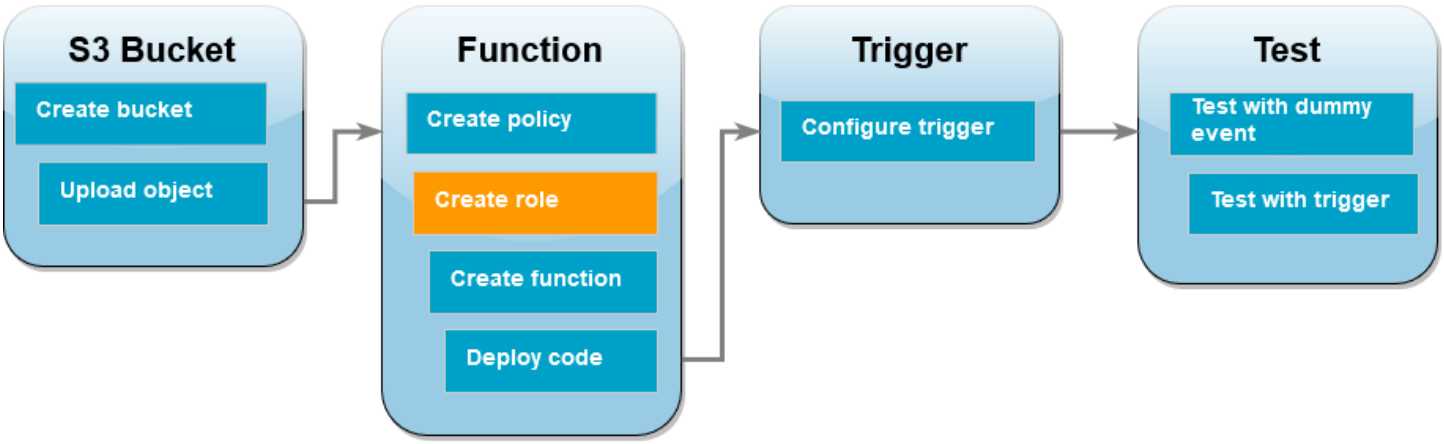
1. Ouvrez la page Politiques de la IAM console.
2. Choisissez Create Policy (Créer une politique).
3. Choisissez l'JSONonglet, puis collez la politique personnalisée suivante dans l'JSONéditeur.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

4. Choisissez Suivant : Balises.
5. Choisissez Suivant : Vérification.
6. Sous Examiner une stratégie, pour le Nom de la stratégie, saisissez **`s3-trigger-tutorial`**.
7. Choisissez Créer une stratégie.

## Créer un rôle d'exécution

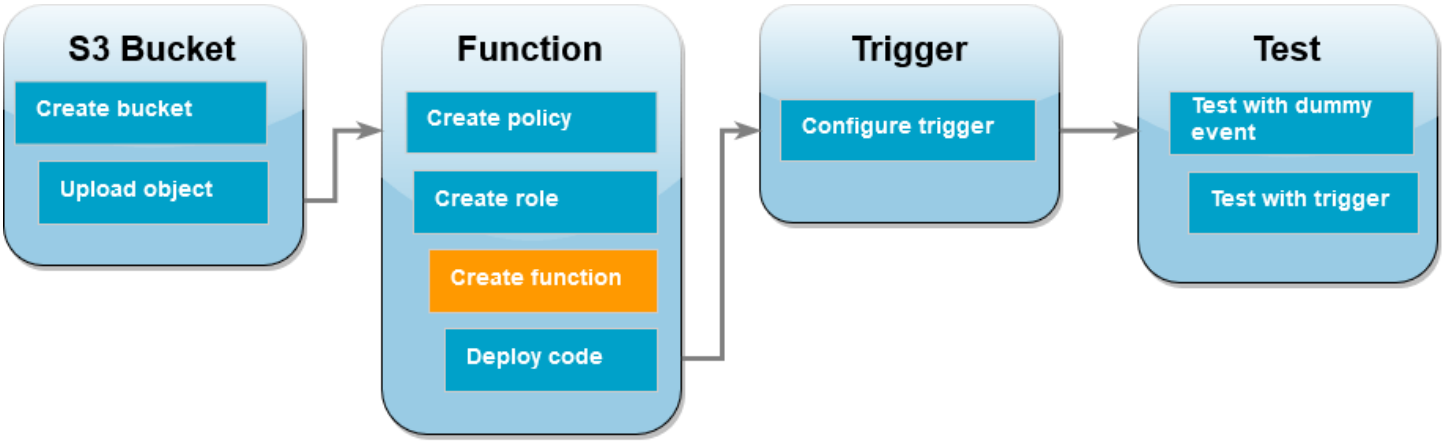



Un rôle d'exécution est un rôle AWS Identity and Access Management (IAM) qui accorde à une fonction Lambda l'autorisation d'accéder aux AWS services et aux ressources. Au cours de cette étape, créez un rôle d'exécution à l'aide de la politique d'autorisation que vous avez créée à l'étape précédente.

Pour créer un rôle d'exécution et attacher votre politique d'autorisations personnalisée

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez Créer un rôle.
3. Pour le type d'entité de confiance, choisissez Service AWS , puis pour le cas d'utilisation, choisissez Lambda.
4. Choisissez Suivant.
5. Dans la zone de recherche de stratégie, entrez **`s3-trigger-tutorial`**.
6. Dans les résultats de la recherche, sélectionnez la stratégie que vous avez créée (`s3-trigger-tutorial`), puis choisissez Suivant.
7. Sous Role details (Détails du rôle), pour Role name (Nom du rôle), saisissez **`lambda-s3-trigger-role`**, puis sélectionnez Create role (Créer un rôle).

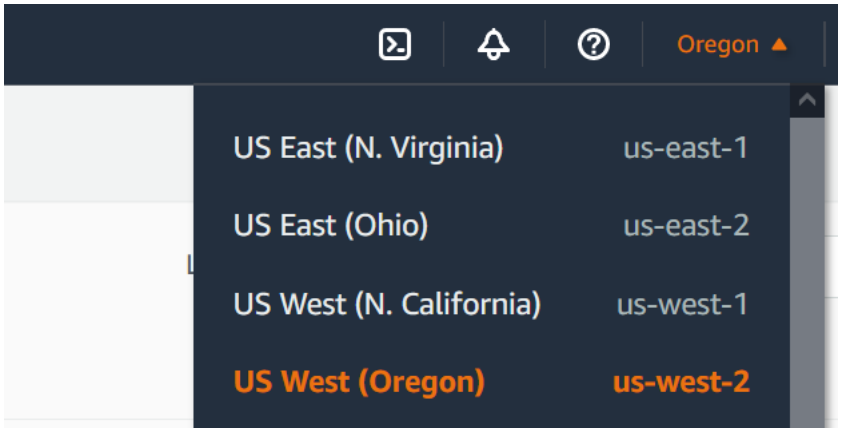
## Créer la fonction Lambda




Créez une fonction Lambda dans la console à l'aide de l'environnement d'exécution Python 3.12.

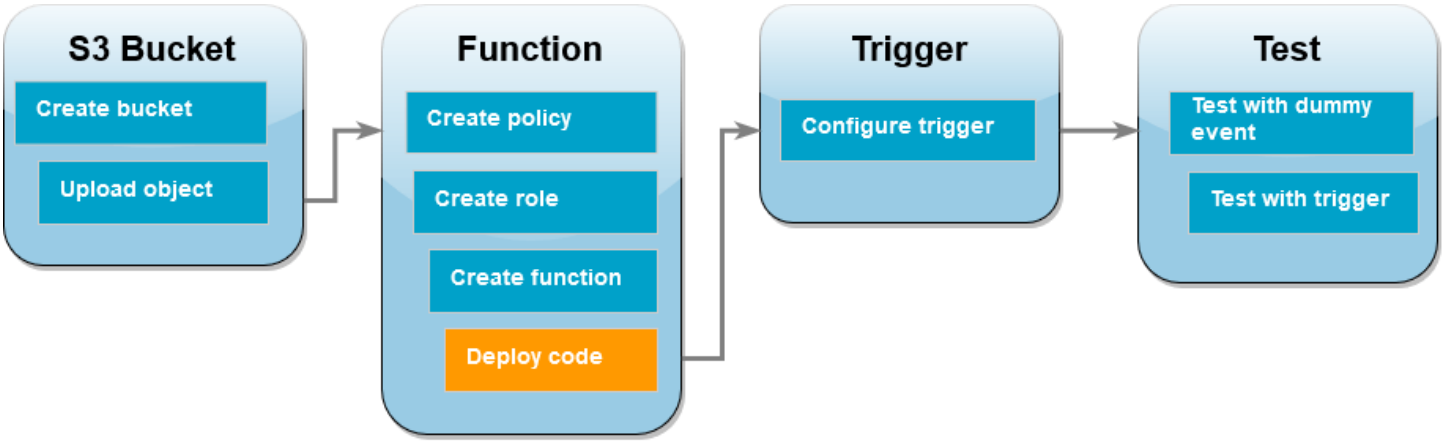
Pour créer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Assurez-vous de travailler dans le même environnement que celui dans Région AWS lequel vous avez créé votre compartiment Amazon S3. Vous pouvez modifier votre région à l'aide de la liste déroulante en haut de l'écran.




3. Sélectionnez Create function (Créer une fonction).
4. Choisissez Créer à partir de zéro.
5. Sous Basic information (Informations de base), procédez comme suit :
   a. Sous Nom de la fonction, saisissez `s3-trigger-tutorial`.
   b. Pour Runtime, choisissez Python 3.12.
   c. Pour Architecture, choisissez x86_64.
6. Dans l'onglet Modifier le rôle d'exécution par défaut, procédez comme suit :
   a. Ouvrez l'onglet, puis choisissez Utiliser un rôle existant.
   b. Sélectionnez le `lambda-s3-trigger-role` que vous avez créé précédemment.
7. Sélectionnez Create function (Créer une fonction).

## Déployer le code de la fonction



Ce didacticiel utilise le moteur d'exécution Python 3.12, mais nous avons également fourni des exemples de fichiers de code pour d'autres environnements d'exécution. Vous pouvez sélectionner l'onglet dans la zone suivante pour voir le code d'exécution qui vous intéresse.

La fonction Lambda récupère le nom clé de l'objet chargé et le nom du compartiment à partir du `event` paramètre qu'il reçoit d'Amazon S3. La fonction utilise ensuite la méthode get_object du AWS SDK for Python (Boto3) pour récupérer les métadonnées de l'objet, y compris le type de contenu (MIMEtype) de l'objet téléchargé.

Pour déployer le code de la fonction

1. Choisissez l'onglet Python dans la zone suivante et copiez le code.

   .NET

   AWS SDK for .NET

   > **Note**
   >
   > Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

   Consommation d'un événement S3 avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be
 converted into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson

namespace S3Integration
{
```

```
   public class Function
   {
       private static AmazonS3Client _s3Client;
       public Function() : this(null)
       {
       }

       internal Function(AmazonS3Client s3Client)
       {
           _s3Client = s3Client ?? new AmazonS3Client();
       }

       public async Task<string> Handler(S3Event evt, ILambdaContext
context)
       {
           try
           {
               if (evt.Records.Count <= 0)
               {
                   context.Logger.LogLine("Empty S3 Event received");
                   return string.Empty;
               }

               var bucket = evt.Records[0].S3.Bucket.Name;
               var key =
HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

               context.Logger.LogLine($"Request is for {bucket} and {key}");

               var objectResult = await _s3Client.GetObjectAsync(bucket,
key);

               context.Logger.LogLine($"Returning {objectResult.Key}");

               return objectResult.Key;
           }
           catch (Exception e)
           {
               context.Logger.LogLine($"Error processing request -
{e.Message}");

               return string.Empty;
           }
       }
```

```
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
```

```
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
 com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNo

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger =
 LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
          S3EventNotificationRecord record = s3event.getRecords().get(0);
          String srcBucket = record.getS3().getBucket().getName();
          String srcKey = record.getS3().getObject().getUrlDecodedKey();

          S3Client s3Client = S3Client.builder().build();
          HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
 srcKey);

          logger.info("Successfully retrieved " + srcBucket + "/" + srcKey +
 " of type " + headObject.contentType());

          return "Ok";
        } catch (Exception e) {
          throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String
 bucket, String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
                .bucket(bucket)
                .key(key)
                .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. JavaScript

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\
+/g, ' '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}.
 Make sure they exist and your bucket is in the same region as this
 function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Consommation d'un événement S3 avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> =>
 {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/
g, ' '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. PHP

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';


class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());
```

```
            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
 $bucket . '. Make sure they exist and your bucket is in the same region as
 this function.' . "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')
```

```
def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']
['key'], encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they
 exist and your bucket is in the same region as this function.'.format(key,
 bucket))
        raise e
```

### Ruby

#### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda à l'aide de Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
```

```
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']
['key'], Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they
 exist and your bucket is in the same region as this function."
    raise e
  end
end
```

### Rust

#### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

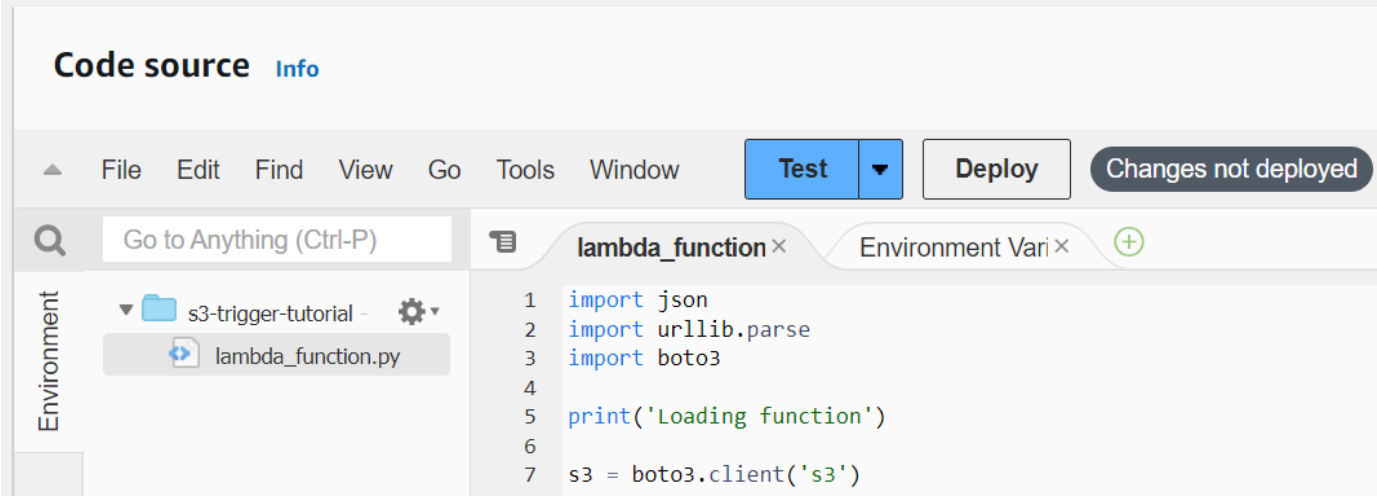
Utilisation d'un événement S3 avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};


/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request
 from SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket =
 evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name to
 exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object
 key to exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
```

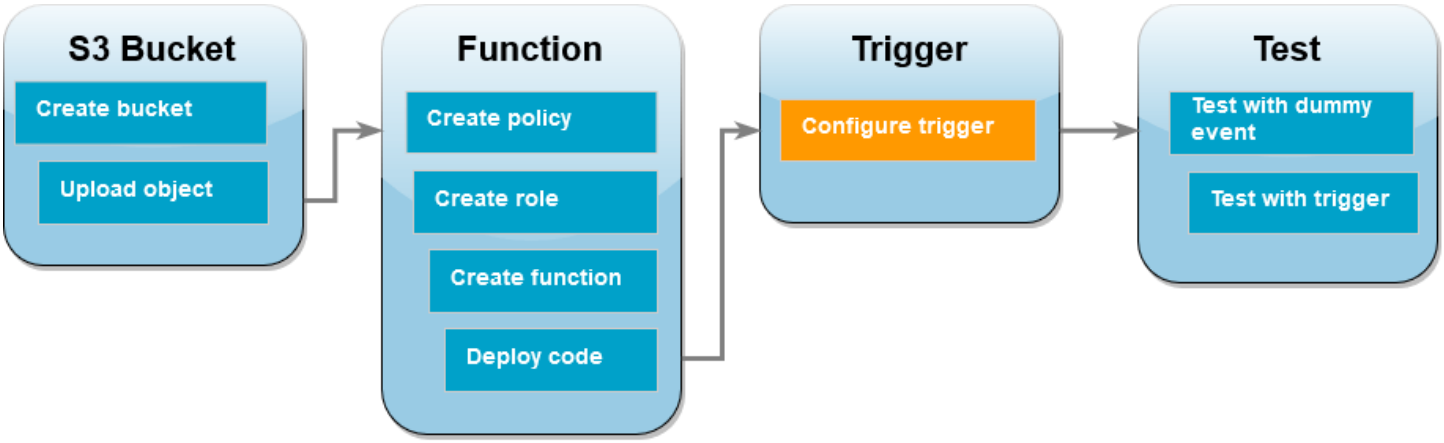
```
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

2. Dans le volet Source du code de la console Lambda, collez le code dans le fichier lambda_function.py.



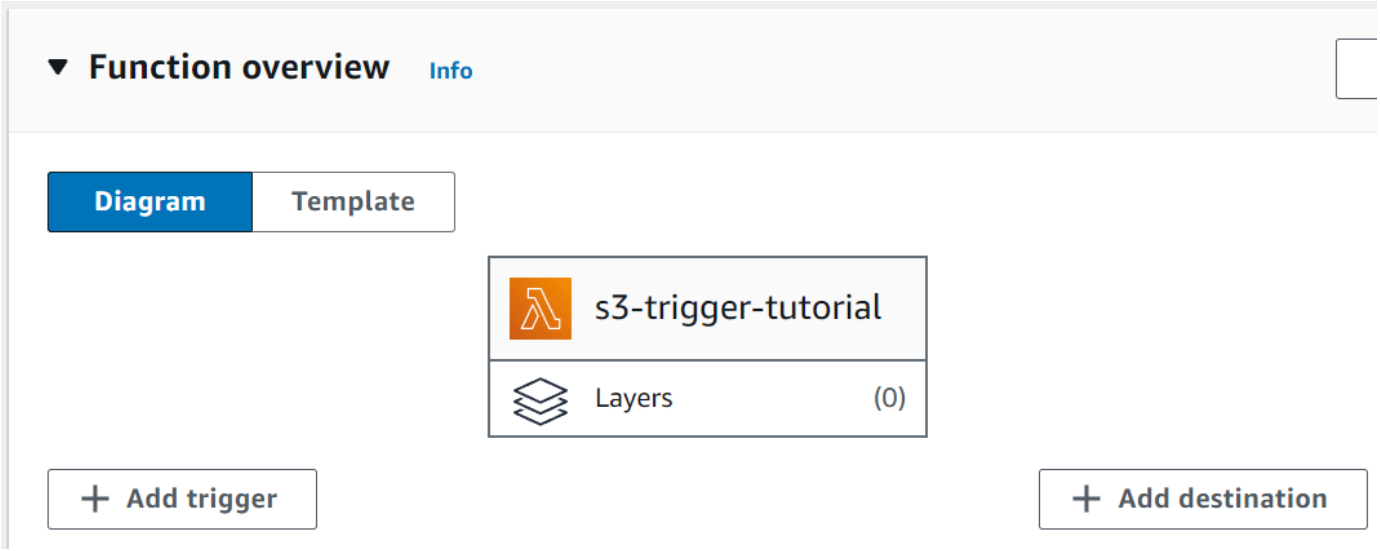

3. Choisissez Deploy (Déployer).

## Création d'un déclencheur Amazon S3




Pour créer le déclencheur Amazon S3

1. Dans le volet de Présentation de la fonction, choisissez Ajouter un déclencheur.



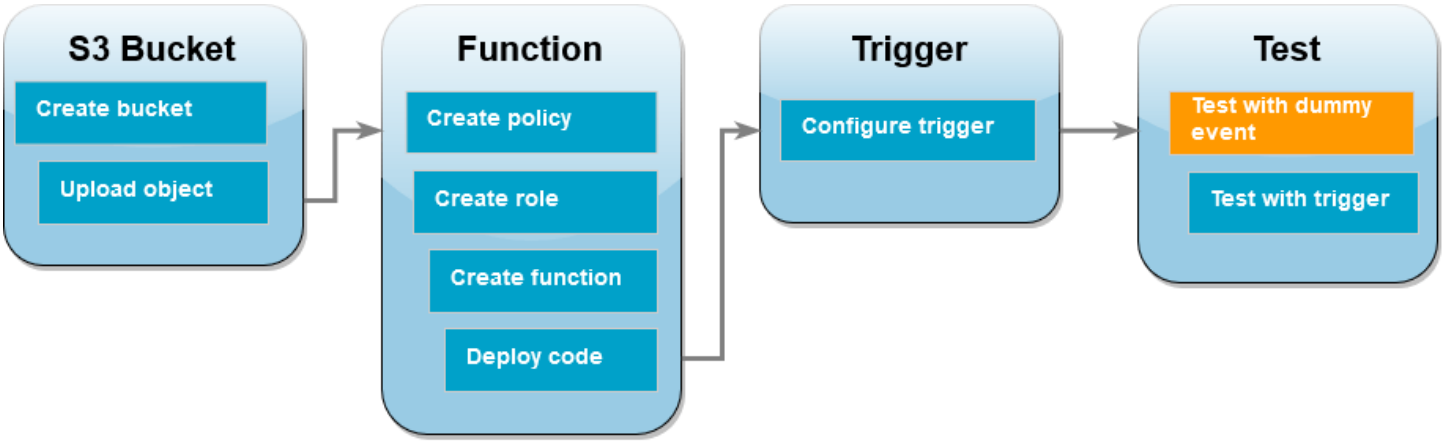

2. Sélectionnez S3.
3. Sous Compartiment, sélectionnez le compartiment que vous avez créé précédemment dans le didacticiel.
4. Sous Types d'événements, assurez-vous que l'option Tous les événements de création d'objets est sélectionnée.
5. Sous Invocation récursive, cochez la case pour confirmer qu'il n'est pas recommandé d'utiliser le même compartiment Amazon S3 pour les entrées et les sorties.
6. Choisissez Ajouter.

> **Note**
>
> Lorsque vous créez un déclencheur Amazon S3 pour une fonction Lambda à l'aide de la console Lambda, Amazon S3 configure une notification d'événement sur le compartiment que vous spécifiez. Avant de configurer cette notification d'événement, Amazon S3 effectue une série de vérifications pour confirmer que la destination de l'événement existe et dispose des IAM politiques requises. Amazon S3 effectue également ces tests sur toutes les autres notifications d'événements configurées pour ce compartiment.
> En raison de cette vérification, si le compartiment a déjà configuré des destinations d'événements pour des ressources qui n'existent plus ou pour des ressources qui ne disposent pas des politiques d'autorisation requises, Amazon S3 ne sera pas en mesure de créer la nouvelle notification d'événement. Vous verrez le message d'erreur suivant indiquant que votre déclencheur n'a pas pu être créé :

```
An error occurred when creating the trigger: Unable to validate the following
 destination configurations.
```

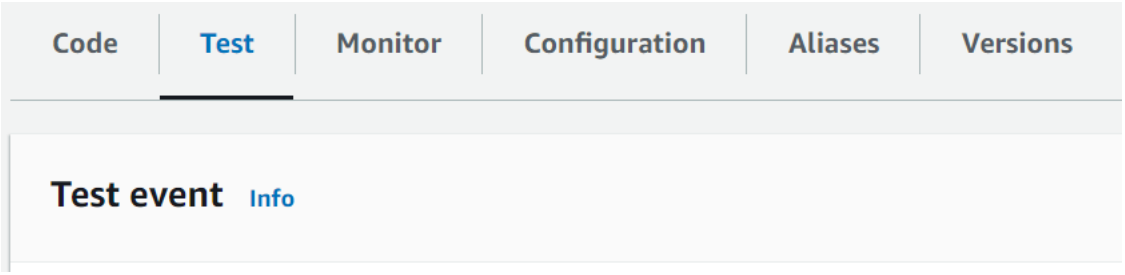
Vous pouvez voir cette erreur si vous avez précédemment configuré un déclencheur pour une autre fonction Lambda utilisant le même compartiment, et si vous avez depuis supprimé la fonction ou modifié ses politiques d'autorisation.

## Test de votre fonction Lambda à l'aide d'un événement fictif




Pour tester la fonction Lambda à l'aide d'un événement fictif

1. Sur la page de console Lambda de votre fonction, choisissez l'onglet Test.




2. Dans Event name (Nom de l'événement), saisissez `MyTestEvent`.
3. Dans l'événement JSON, collez l'événement de test suivant. Assurez-vous de remplacer les valeurs suivantes :

   - Remplacez `us-east-1` par la région dans laquelle vous avez créé votre compartiment Amazon S3.

- Remplacez les deux instances de `amzn-s3-demo-bucket` par le nom de votre propre compartiment Amazon S3.
- Remplacez `test%2FKey` par le nom de l'objet de test que vous avez chargé précédemment dans votre compartiment (par exemple, `HappyFace.jpg`).

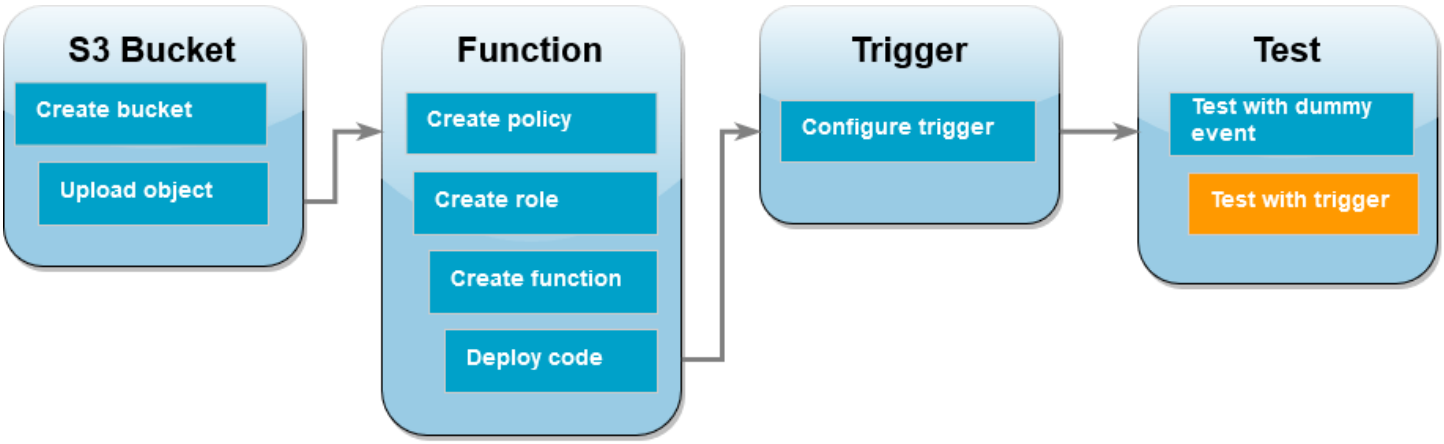
```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
```

```
    }
  ]
}
```

4. Choisissez Save (Enregistrer).
5. Sélectionnez Tester).
6. Si votre fonction s'exécute correctement, vous obtiendrez un résultat similaire à celui qui suit dans l'onglet Résultats de l'exécution.

```
Response
"image/jpeg"

Function Logs
START RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6 Version: $LATEST
2021-02-18T21:40:59.280Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    INPUT
 BUCKET AND KEY:  { Bucket: 'amzn-s3-demo-bucket', Key: 'HappyFace.jpg' }
2021-02-18T21:41:00.215Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    CONTENT
 TYPE: image/jpeg
END RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6
REPORT RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6    Duration: 976.25 ms
 Billed Duration: 977 ms    Memory Size: 128 MB    Max Memory Used: 90 MB    Init
 Duration: 430.47 ms

Request ID
12b3cae7-5f4e-415e-93e6-416b8f8b66e6
```

Test de la fonction Lambda avec le déclencheur Amazon S3



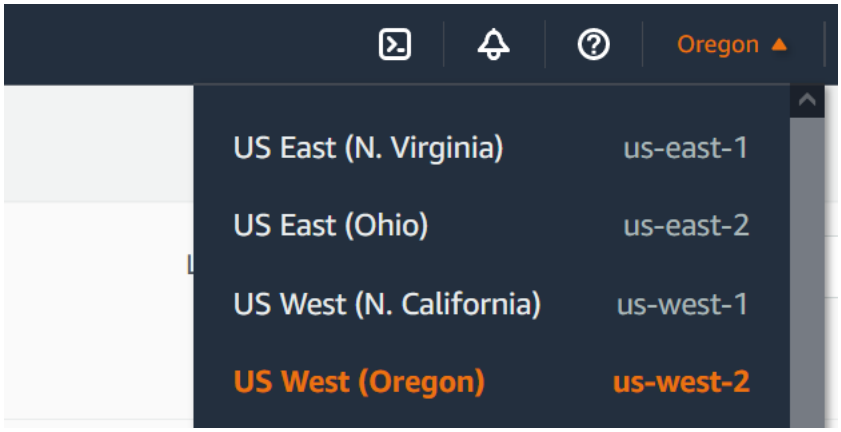
Pour tester votre fonction avec le déclencheur configuré, téléchargez un objet dans votre compartiment Amazon S3 à l'aide de la console. Pour vérifier que votre fonction Lambda s'est exécutée comme prévu, utilisez CloudWatch Logs pour afficher le résultat de votre fonction.

Pour charger un objet dans votre compartiment Amazon S3

1. Ouvrez la page Buckets de la console Amazon S3 et choisissez le bucket que vous avez créé précédemment.
2. Sélectionnez Charger.
3. Choisissez Ajouter des fichiers et utilisez le sélecteur de fichiers pour choisir l'objet que vous souhaitez charger. Cet objet peut être n'importe quel fichier que vous choisissez.
4. Choisissez Ouvrir, puis Charger.

Pour vérifier l'invocation de la fonction à l'aide CloudWatch de Logs

1. Ouvrez la CloudWatchconsole.
2. Assurez-vous de travailler de la même manière que celle dans laquelle Région AWS vous avez créé votre fonction Lambda. Vous pouvez modifier votre région à l'aide de la liste déroulante en haut de l'écran.




3. Choisissez Journaux, puis Groupes de journaux.
4. Choisissez le groupe de journaux de votre fonction (`/aws/lambda/s3-trigger-tutorial`).
5. Sous Flux de journaux, sélectionnez le flux de journaux le plus récent.
6. Si votre fonction a été invoquée correctement en réponse à votre déclencheur Amazon S3, vous verrez un résultat similaire à ce qui suit. Le `CONTENT TYPE` que vous voyez dépend du type de fichier que vous avez chargé dans votre compartiment.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT
 TYPE: image/jpeg
```

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le compartiment S3

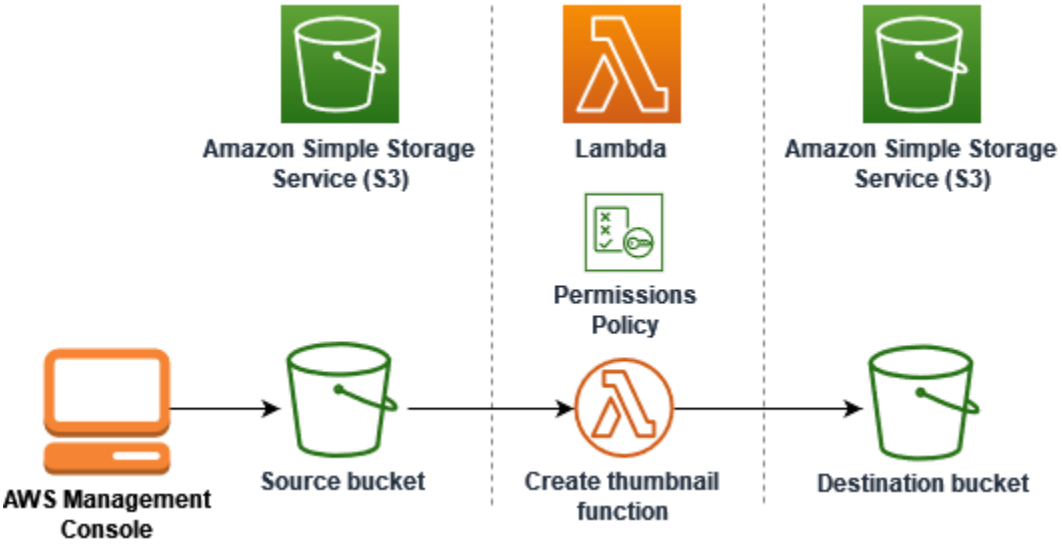
1. Ouvrez la console Amazon S3.
2. Sélectionnez le compartiment que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du compartiment dans le champ de saisie de texte.
5. Choisissez Supprimer le compartiment.

## Étapes suivantes

DansDidacticiel : Utilisation d'un déclencheur Amazon S3 pour créer des images miniatures, le déclencheur Amazon S3 invoque une fonction qui crée une image miniature pour chaque fichier image chargé dans un compartiment. Ce didacticiel nécessite un niveau modéré de connaissance du AWS domaine Lambda. Il montre comment créer des ressources à l'aide de AWS Command Line Interface (AWS CLI) et comment créer un package de déploiement d'archives de fichiers .zip pour la fonction et ses dépendances.

# Didacticiel : Utilisation d'un déclencheur Amazon S3 pour créer des images miniatures

Dans ce tutoriel, vous allez créer et configurer une fonction Lambda qui redimensionne les images ajoutées à un compartiment Amazon Simple Storage Service (Amazon S3). Lorsque vous ajoutez un fichier image à votre compartiment, Amazon S3 invoque votre fonction Lambda. La fonction crée ensuite une version miniature de l'image et l'envoie vers un autre compartiment Amazon S3.




Pour compléter ce didacticiel, effectuez les tâches suivantes :

1. Créez des compartiments Amazon S3 source et de destination et chargez un exemple d'image.
2. Créez une fonction Lambda qui redimensionne une image et envoie une miniature vers un compartiment Amazon S3.
3. Configurez un déclencheur Lambda qui invoque votre fonction lorsque des objets sont chargés dans votre compartiment source.

4. Testez votre fonction, d'abord avec un événement fictif, puis en chargeant une image dans votre compartiment source.

En suivant ces étapes, vous apprendrez à utiliser Lambda pour effectuer une tâche de traitement de fichiers sur des objets ajoutés à un compartiment Amazon S3. Vous pouvez suivre ce didacticiel en utilisant le AWS Command Line Interface (AWS CLI) ou le AWS Management Console.

Si vous recherchez un exemple plus simple pour apprendre à configurer un déclencheur Amazon S3 pour Lambda, vous pouvez consulter le Didacticiel : utilisation d'un déclencheur Amazon S3 pour invoquer une fonction Lambda.

Rubriques


- Prérequis
- Création de deux compartiments Amazon S3
- Charger une image de test dans votre compartiment source
- Créer une stratégie d’autorisations
- Créer un rôle d’exécution
- Créer le package de déploiement de la fonction
- Créer la fonction Lambda
- Configurer Amazon S3 pour invoquer la fonction
- Test de votre fonction Lambda à l’aide d’un événement fictif
- Tester votre fonction à l'aide du déclencheur Amazon S3
- Nettoyage de vos ressources


## Prérequis

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et le gérer en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .

2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu'utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

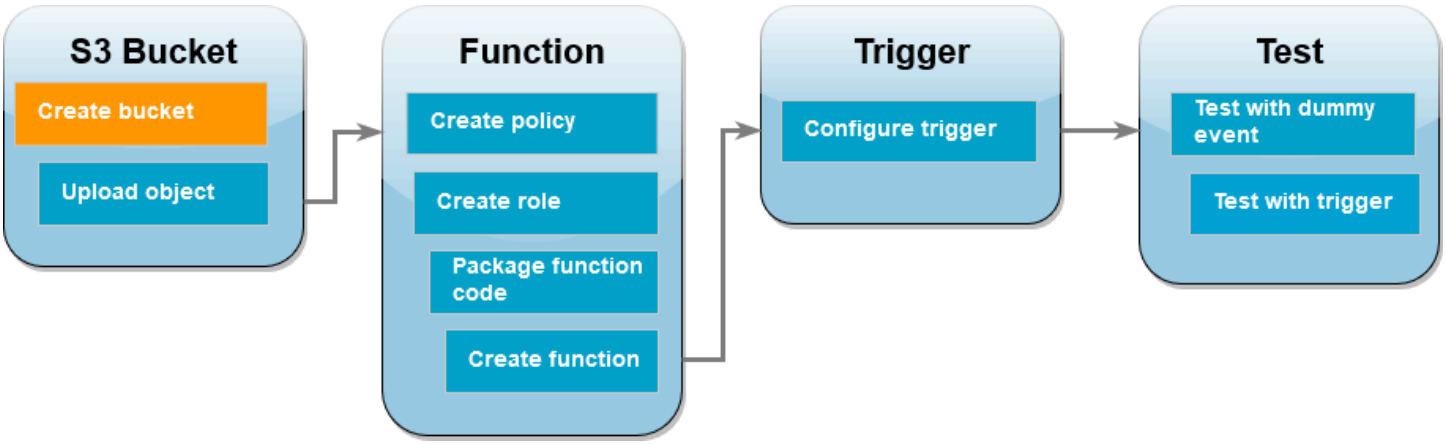
   Pour obtenir des instructions, consultez Création d'un ensemble d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l'utilisateur AWS IAM Identity Center .

Si vous souhaitez utiliser le AWS CLI pour terminer le didacticiel, installez la dernière version du AWS Command Line Interface.

Pour le code de votre fonction Lambda, vous pouvez utiliser Python ou Node.js. Installez les outils de prise en charge linguistique et un gestionnaire de packages pour le langage que vous souhaitez utiliser.

## Création de deux compartiments Amazon S3




Créez d'abord deux compartiments Amazon S3. Le premier compartiment est le compartiment source dans lequel vous allez charger vos images. Le second compartiment est utilisé par Lambda pour enregistrer la miniature redimensionnée lorsque vous invoquez votre fonction.

AWS Management Console

Pour créer les compartiments Amazon S3 (console)

1. Ouvrez la page Compartiments de la console Amazon S3.
2. Choisissez Create bucket (Créer un compartiment).
3. Sous Configuration générale, procédez comme suit :
   a. Pour le nom du compartiment, saisissez un nom unique au monde qui respecte les règles de dénomination du compartiment Amazon S3. Les noms de compartiment peuvent contenir uniquement des lettres minuscules, des chiffres, de points (.) et des traits d'union (-).
   b. Pour Région AWS, choisissez la Région AWS la plus proche de votre emplacement géographique. Plus loin dans le didacticiel, vous devez créer votre fonction Lambda dans celui-ci Région AWS. Notez donc la région que vous avez choisie.
4. Conservez les valeurs par défaut de toutes les autres options et choisissez Créer un compartiment.
5. Répétez les étapes 1 à 4 pour créer votre compartiment de destination. Pour Nom du compartiment, saisissez **`amzn-s3-demo-source-bucket-resized`**, où **`amzn-s3-demo-source-bucket`** est le nom du compartiment source que vous venez de créer.

AWS CLI

Pour créer les compartiments Amazon S3 (AWS CLI)

1. Exécutez la CLI commande suivante pour créer votre compartiment source. Le nom que vous choisissez pour votre compartiment doit être unique au monde et respecter les Règles de dénomination du compartiment Amazon S3. Les noms peuvent contenir uniquement des lettres minuscules, des chiffres, de points (.) et des traits d'union (-). Pour `region` et `LocationConstraint`, choisissez la Région AWS la plus proche de votre emplacement géographique.

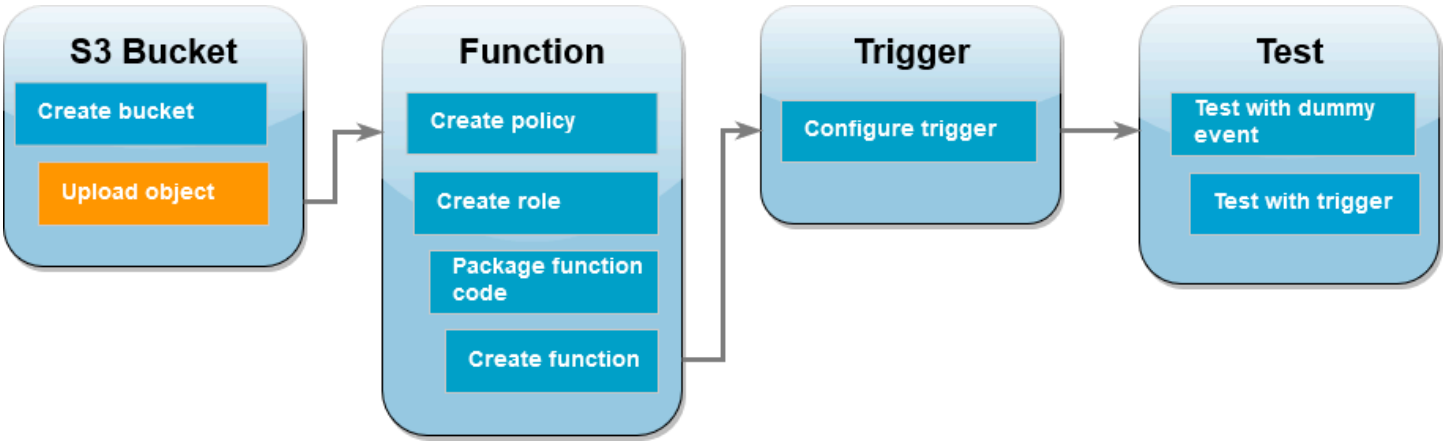
   ```
   aws s3api create-bucket --bucket amzn-s3-demo-source-bucket --region us-east-1 \
   --create-bucket-configuration LocationConstraint=us-east-1
   ```

   Plus loin dans le didacticiel, vous devez créer votre fonction Lambda dans le même Région AWS emplacement que votre compartiment source. Notez donc la région que vous avez choisie.

2. Exécutez la commande suivante pour créer votre compartiment de destination. Pour le nom du compartiment, vous devez utiliser **`amzn-s3-demo-source-bucket-resized`**, où **`amzn-s3-demo-source-bucket`** est le nom du compartiment source que vous avez créé à l'étape 1. Pour `region` et`LocationConstraint`, choisissez le même Région AWS que celui que vous avez utilisé pour créer votre bucket source.

   ```
   aws s3api create-bucket --bucket amzn-s3-demo-source-bucket-resized --region us-
   east-1 \
   --create-bucket-configuration LocationConstraint=us-east-1
   ```

## Charger une image de test dans votre compartiment source




Plus loin dans le didacticiel, vous testerez votre fonction Lambda en l'invoquant à l'aide de la console Lambda ou de AWS CLI la console Lambda. Pour confirmer que votre fonction fonctionne correctement, votre compartiment source doit contenir une image de test. Cette image peut être n'importe laquelle JPG ou n'importe quel PNG fichier de votre choix.

AWS Management Console

Pour charger une image de test dans votre compartiment source (console)

1. Ouvrez la page Compartiments de la console Amazon S3.
2. Sélectionnez le compartiment source que vous avez créé à l'étape précédente.
3. Sélectionnez Charger.
4. Choisissez Ajouter des fichiers et utilisez le sélecteur de fichiers pour sélectionner l'objet que vous souhaitez charger.
5. Choisissez Ouvrir, puis Charger.

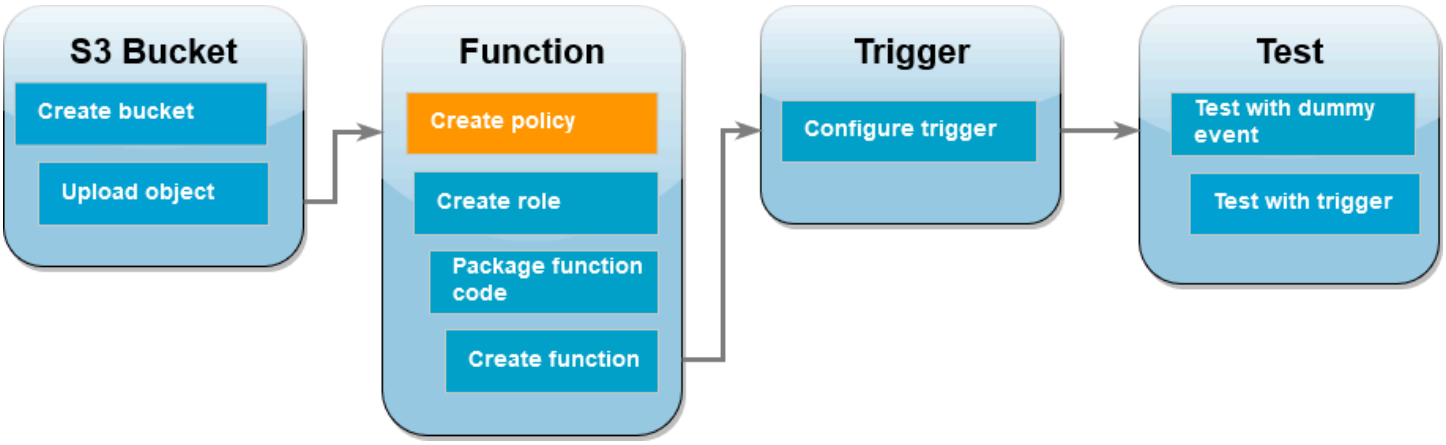
AWS CLI

Pour charger une image de test dans votre compartiment source (AWS CLI)

- Dans le répertoire contenant l'image que vous souhaitez télécharger, exécutez la CLI commande suivante. Remplacez le paramètre `--bucket` par le nom de votre compartiment source. Pour les paramètres `--key` et `--body`, utilisez le nom de fichier de votre image de test.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key HappyFace.jpg --
body ./HappyFace.jpg
```

## Créer une stratégie d'autorisations




La première étape de la création de votre fonction Lambda consiste à créer une politique d'autorisations. Cette politique donne à votre fonction les autorisations dont elle a besoin pour accéder à d'autres AWS ressources. Pour ce didacticiel, la politique donne à Lambda des autorisations de lecture et d'écriture pour les compartiments Amazon S3 et lui permet d'écrire dans Amazon Logs. CloudWatch

AWS Management Console

Pour créer une politique (console)

1. Ouvrez la page Politiques de la console AWS Identity and Access Management (IAM).
2. Choisissez Create Policy (Créer une politique).
3. Choisissez l'JSONonglet, puis collez la politique personnalisée suivante dans l'JSONéditeur.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
```

```
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

4. Choisissez Suivant.
5. Sous Détails de la politique, pour le Nom de la politique, saisissez **`LambdaS3Policy`**.
6. Choisissez Create Policy (Créer une politique).

AWS CLI

Pour créer la politique (AWS CLI)

1. Enregistrez ce qui suit JSON dans un fichier nommé`policy.json`.

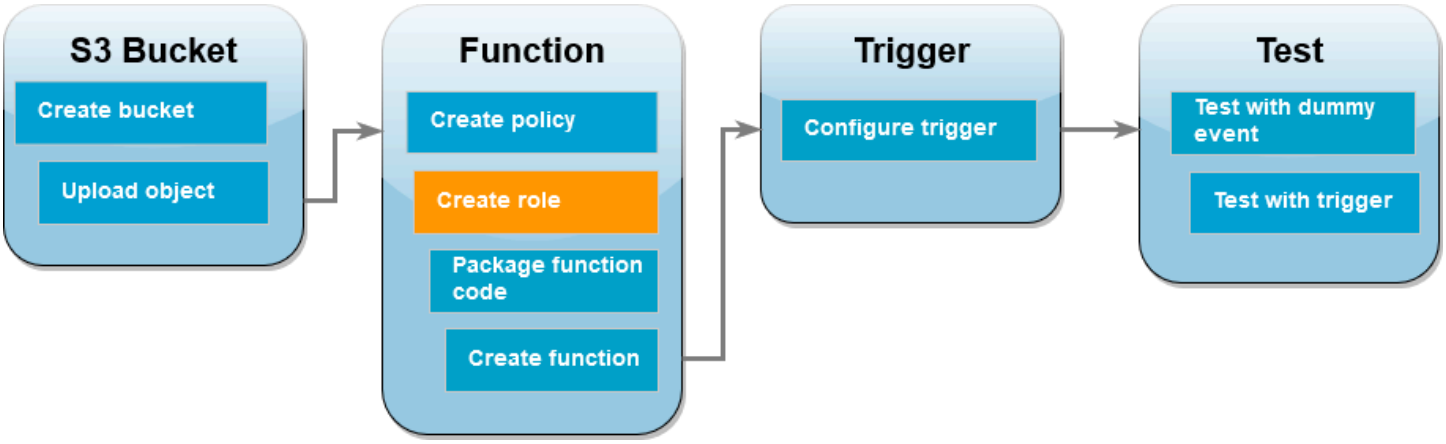
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

2. Dans le répertoire dans lequel vous avez enregistré le document de JSON politique, exécutez la CLI commande suivante.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://
policy.json
```

## Créer un rôle d'exécution




Un rôle d'exécution est un IAM rôle qui accorde à une fonction Lambda l'autorisation d'accès Services AWS et de ressources. Pour donner à votre fonction un accès en lecture et en écriture à un compartiment Amazon S3, vous attachez la politique d'autorisation que vous avez créée à l'étape précédente.

AWS Management Console

Pour créer un rôle d'exécution et attacher votre politique d'autorisations (console)

1. Ouvrez la page Rôles de la console (IAM).
2. Sélectionnez Créer un rôle.
3. Pour Type d'entité fiable, sélectionnez Service AWS, et pour Cas d'utilisation, sélectionnez Lambda.
4. Choisissez Suivant.
5. Ajoutez la politique d'autorisations que vous avez créée à l'étape précédente en procédant comme suit :
   a. Dans la zone de recherche de stratégie, entrez **`LambdaS3Policy`**.
   b. Dans les résultats de la recherche, cochez la case pour `LambdaS3Policy`.
   c. Choisissez Suivant.
6. Sous Détails du rôle, pour le Nom du rôle entrez **`LambdaS3Role`**.
7. Sélectionnez Créer un rôle.

AWS CLI

Pour créer un rôle d'exécution et attacher votre politique d'autorisations (AWS CLI)

1. Enregistrez ce qui suit JSON dans un fichier nommé`trust-policy.json`. Cette politique de confiance permet à Lambda d'utiliser les autorisations du rôle en autorisant le principal `lambda.amazonaws.com` du service à appeler l'action AWS Security Token Service (AWS STS)`AssumeRole`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

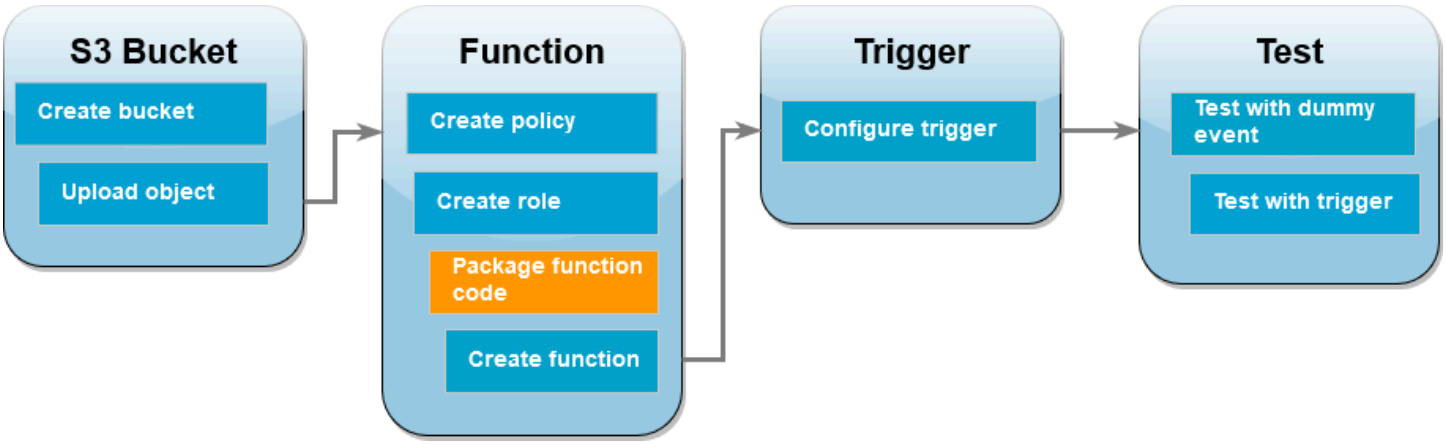
2. Dans le répertoire dans lequel vous avez enregistré le document de politique de JSON confiance, exécutez la CLI commande suivante pour créer le rôle d'exécution.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
 file://trust-policy.json
```

3. Pour joindre la politique d'autorisation que vous avez créée à l'étape précédente, exécutez la CLI commande suivante. Remplacez le Compte AWS numéro indiqué dans le contrat ARN par votre propre numéro de compte.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::123456789012:policy/LambdaS3Policy
```

## Créer le package de déploiement de la fonction




Pour créer votre fonction, vous créez un package de déploiement contenant le code de votre fonction et ses dépendances. Pour cette fonction `CreateThumbnail`, votre code de fonction utilise une bibliothèque distincte pour le redimensionnement de l'image. Suivez les instructions correspondant à la langue de votre choix pour créer un package de déploiement contenant la bibliothèque requise.

Node.js

Pour créer le package de déploiement (Node.js)

1. Créez un répertoire nommé `lambda-s3` pour votre code de fonction et vos dépendances et accédez-y.

```
mkdir lambda-s3
cd lambda-s3
```

2. Créez un nouveau projet Node.js avec `npm`. Appuyez sur `Enter` pour accepter les options par défaut fournies dans l'expérience interactive.

```
npm init
```

3. Enregistrez le code de fonction suivant dans un fichier nommé `index.mjs`. Assurez-vous de les `us-east-1` remplacer par celui Région AWS dans lequel vous avez créé vos propres compartiments source et de destination.

```
// dependencies
import { S3Client, GetObjectCommand, PutObjectCommand } from '@aws-sdk/client-
s3';

import { Readable } from 'stream';

import sharp from 'sharp';
import util from 'util';


// create S3 client
const s3 = new S3Client({region: 'us-east-1'});

// define the handler function
export const handler = async (event, context) => {

// Read options from the event parameter and get the source bucket
console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
  const srcBucket = event.Records[0].s3.bucket.name;

// Object key may have spaces or unicode non-ASCII characters
const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/
g, " "));
const dstBucket = srcBucket + "-resized";
const dstKey    = "resized-" + srcKey;

// Infer the image type from the file suffix
const typeMatch = srcKey.match(/\.([^.]*)$/);
if (!typeMatch) {
  console.log("Could not determine the image type.");
```

```
  return;
}

// Check that the image type is supported
const imageType = typeMatch[1].toLowerCase();
if (imageType != "jpg" && imageType != "png") {
  console.log(`Unsupported image type: ${imageType}`);
  return;
}

// Get the image from the source bucket. GetObjectCommand returns a stream.
try {
  const params = {
    Bucket: srcBucket,
    Key: srcKey
  };
  var response = await s3.send(new GetObjectCommand(params));
  var stream = response.Body;

// Convert stream to buffer to pass to sharp resize function.
  if (stream instanceof Readable) {
    var content_buffer = Buffer.concat(await stream.toArray());

  } else {
    throw new Error('Unknown object stream type');
  }


} catch (error) {
  console.log(error);
  return;
}


// set thumbnail width. Resize will set the height automatically to maintain
 aspect ratio.
const width  = 200;

// Use the sharp module to resize the image and save in a buffer.
try {
  var output_buffer = await sharp(content_buffer).resize(width).toBuffer();

} catch (error) {
  console.log(error);
```

```
  return;
}

// Upload the thumbnail image to the destination bucket
try {
  const destparams = {
    Bucket: dstBucket,
    Key: dstKey,
    Body: output_buffer,
    ContentType: "image"
  };

  const putResult = await s3.send(new PutObjectCommand(destparams));

  } catch (error) {
    console.log(error);
    return;
  }

  console.log('Successfully resized ' + srcBucket + '/' + srcKey +
    ' and uploaded to ' + dstBucket + '/' + dstKey);
  };
```

4. Dans votre répertoire `lambda-s3`, installez la bibliothèque sharp en utilisant npm. Notez que la dernière version de sharp (0.33) n'est pas compatible avec Lambda. Installez la version 0.32.6 pour terminer ce didacticiel.

```
npm install sharp@0.32.6
```

La commande npm `install` crée un répertoire `node_modules` pour vos modules. Après cette étape, votre structure de répertoire doit se présenter comme suit.

```
lambda-s3
|- index.mjs
|- node_modules
|  |- base64js
|  |- bl
|  |- buffer
...
|- package-lock.json
|- package.json
```

5. Créez un package de déploiement .zip contenant le code de votre fonction et ses dépendances. Sous macOS ou Linux, exécutez les commandes suivantes.

```
zip -r function.zip .
```

Sous Windows, utilisez l'utilitaire zip de votre choix pour créer un fichier .zip. Assurez-vous que vos fichiers `index.mjs`, `package.json`, et `package-lock.json` ainsi que votre répertoire `node_modules` se trouvent tous à la racine de votre fichier .zip.

Python

Pour créer le package de déploiement (Python)

1. Enregistrez l'exemple de code en tant que fichier nommé `lambda_function.py`.

```
import boto3
import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
  with Image.open(image_path) as image:
    image.thumbnail(tuple(x / 2 for x in image.size))
    image.save(resized_path)

def lambda_handler(event, context):
  for record in event['Records']:
    bucket = record['s3']['bucket']['name']
    key = unquote_plus(record['s3']['object']['key'])
    tmpkey = key.replace('/', '')
    download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
    upload_path = '/tmp/resized-{}'.format(tmpkey)
    s3_client.download_file(bucket, key, download_path)
    resize_image(download_path, upload_path)
    s3_client.upload_file(upload_path, '{}-resized'.format(bucket), 'resized-
{}'.format(key))
```

2. Dans le répertoire dans lequel vous avez créé votre `lambda_function.py` fichier, créez un nouveau répertoire nommé `package` et installez la bibliothèque Pillow (PIL) et le AWS SDK for Python (Boto3). Bien que le runtime Lambda Python inclue une version du Boto3SDK, nous vous recommandons d'ajouter toutes les dépendances de votre fonction à votre package de déploiement, même si elles sont incluses dans le runtime. Pour plus d'informations, consultez la Dépendances d'exécution en Python.

   ```
   mkdir package
   pip install \
   --platform manylinux2014_x86_64 \
   --target=package \
   --implementation cp \
   --python-version 3.12 \
   --only-binary=:all: --upgrade \
   pillow boto3
   ```

   La bibliothèque Pillow contient du code C/C++. En utilisant les options `--platform manylinux_2014_x86_64` et `--only-binary=:all:`, pip téléchargera et installera une version de Pillow contenant des fichiers binaires précompilés compatibles avec le système d'exploitation Amazon Linux 2. Cela garantit que votre package de déploiement fonctionnera dans l'environnement d'exécution Lambda, quels que soient le système d'exploitation et l'architecture de votre ordinateur de création local.

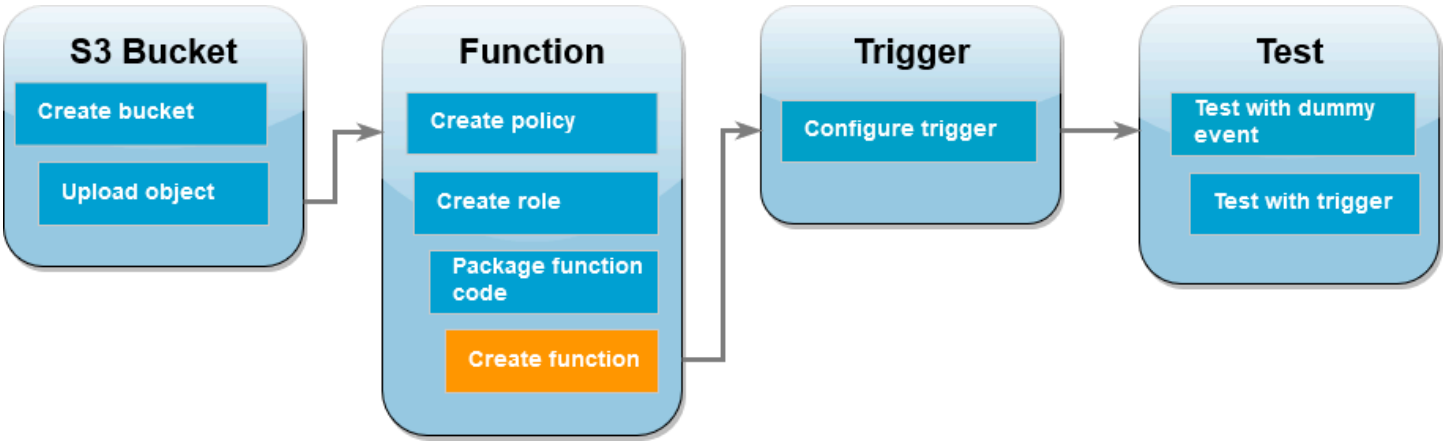
3. Créez un fichier .zip contenant le code de votre application et les bibliothèques Pillow et Boto3. Sous Linux ou macOS, exécutez les commandes suivantes depuis votre interface de ligne de commande.

   ```
   cd package
   zip -r ../lambda_function.zip .
   cd ..
   zip lambda_function.zip lambda_function.py
   ```

   Sous Windows, utilisez l'outil zip de votre choix pour créer le fichier `lambda_function.zip`. Assurez-vous que votre fichier `lambda_function.py` et les dossiers contenant vos dépendances sont installés à la racine du fichier .zip.

Vous pouvez également créer votre package de déploiement à l'aide d'un environnement virtuel Python. Consultez Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python.

## Créer la fonction Lambda



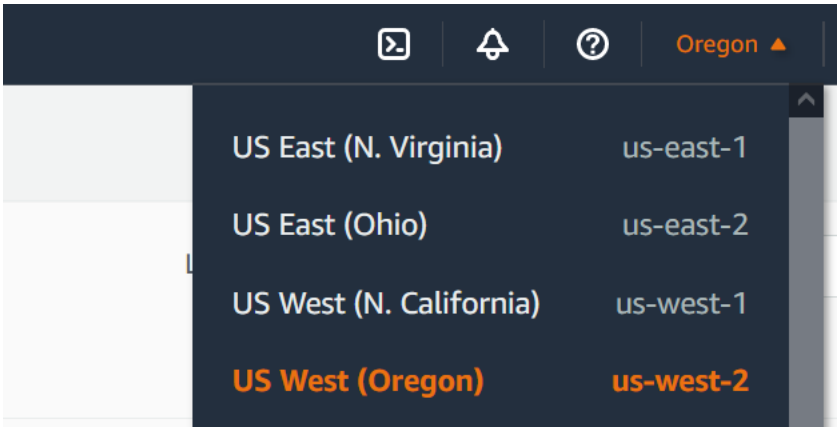

Vous pouvez créer votre fonction Lambda à l'aide de la console Lambda AWS CLI ou de la console Lambda. Suivez les instructions correspondant à la langue de votre choix pour créer la fonction.

AWS Management Console

Pour créer une fonction (console)

Pour créer votre fonction Lambda à l'aide de la console, vous devez d'abord créer une fonction de base contenant du code « Hello world ». Vous remplacez ensuite ce code par votre propre code de fonction en téléchargeant le fichier the.zip ou JAR le fichier que vous avez créé à l'étape précédente.

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Assurez-vous de travailler dans le même environnement que celui dans Région AWS lequel vous avez créé votre compartiment Amazon S3. Vous pouvez modifier votre région à l'aide de la liste déroulante en haut de l'écran.




3. Sélectionnez Create function (Créer une fonction).

4. Choisissez Créer à partir de zéro.
5. Sous Informations de base, procédez comme suit :
   a. Sous Nom de la fonction, saisissez **`CreateThumbnail`**.
   b. Pour Runtime, choisissez Node.js 20.x ou Python 3.12 en fonction du langage que vous avez choisi pour votre fonction.
   c. Pour Architecture, choisissez x86_64.
6. Dans l'onglet Modifier le rôle d'exécution par défaut, procédez comme suit :
   a. Ouvrez l'onglet, puis choisissez Utiliser un rôle existant.
   b. Sélectionnez le `LambdaS3Role` que vous avez créé précédemment.
7. Sélectionnez Create function (Créer une fonction).

Pour charger le code de fonction (console)

1. Dans le volet Source du code, choisissez Charger à partir de.
2. Choisissez Fichier .zip.
3. Sélectionnez Charger.
4. Dans le sélecteur de fichiers, sélectionnez votre fichier .zip et choisissez Ouvrir.
5. Choisissez Save (Enregistrer).

AWS CLI

Pour créer la fonction (AWS CLI)

- Exécutez la CLI commande correspondant à la langue que vous avez choisie. Pour le `role` paramètre, assurez-vous de le remplacer `123456789012` par votre propre Compte AWS identifiant. Pour le paramètre `region`, remplacez `us-east-1` par la région dans laquelle vous avez créé vos compartiments Amazon S3.
  - Pour Node.js, exécutez la commande suivante depuis le répertoire contenant votre fichier `function.zip`.

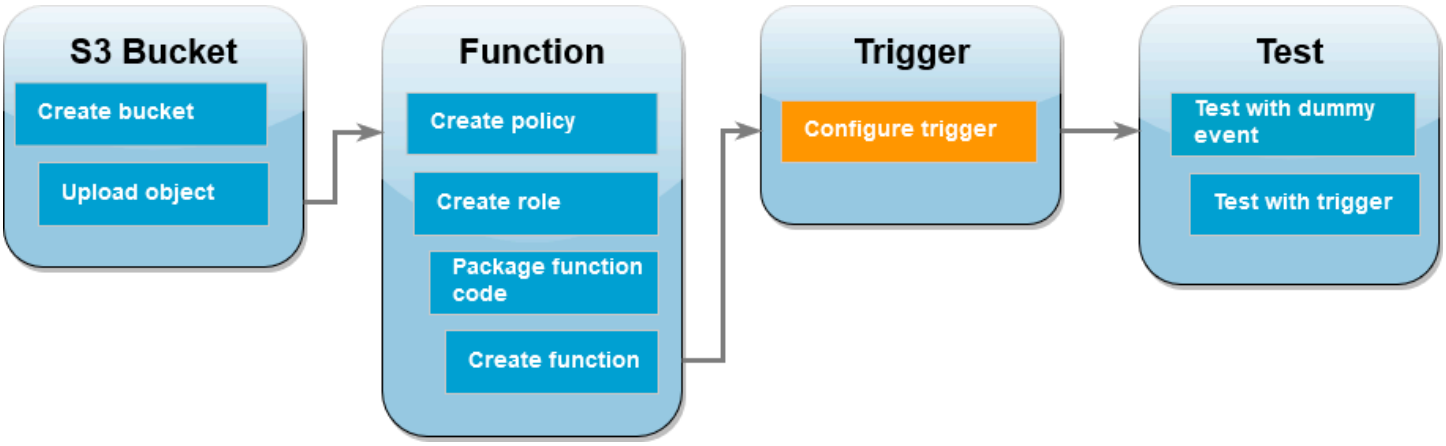
```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \
--timeout 10 --memory-size 1024 \
```

```
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

- Pour Python, exécutez la commande suivante depuis le répertoire contenant votre fichier `lambda_function.zip`.

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://lambda_function.zip --handler
 lambda_function.lambda_handler \
--runtime python3.12 --timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

## Configurer Amazon S3 pour invoquer la fonction




Pour que votre fonction Lambda s'exécute lorsque vous chargez une image dans votre compartiment source, vous devez configurer un déclencheur pour votre fonction. Vous pouvez configurer le déclencheur Amazon S3 à l'aide de la console ou de la AWS CLI.

> **Important**
>
> Cette procédure configure le compartiment Amazon S3 pour qu'il invoque votre fonction chaque fois qu'un objet est créé dans le compartiment. Veillez à configurer cela uniquement pour le compartiment source. Si votre fonction Lambda crée des objets dans le même compartiment que celui qui l'invoque, votre fonction peut être invoquée en continu dans une boucle. Cela peut entraîner la facturation de frais imprévus à votre Compte AWS charge.

AWS Management Console

Pour configurer le déclencheur Amazon S3 (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction (`CreateThumbnail`).
2. Choisissez Add trigger (Ajouter déclencheur).
3. Sélectionnez S3.
4. Sous Compartiment, sélectionnez votre compartiment source.
5. Sous Types d'événements, sélectionnez Tous les événements de création d'objets.
6. Sous Invocation récursive, cochez la case pour confirmer qu'il n'est pas recommandé d'utiliser le même compartiment Amazon S3 pour les entrées et les sorties. Vous pouvez en savoir plus sur les modèles d'invocation récursive dans Lambda en lisant la rubrique Modèles récursifs qui provoquent des fonctions Lambda incontrôlables dans Serverless Land.
7. Choisissez Ajouter.

   Lorsque vous créez un déclencheur à l'aide de la console Lambda, ce dernier crée automatiquement une politique basée sur une ressource pour donner au service que vous sélectionnez l'autorisation d'invoquer votre fonction.

AWS CLI

Pour configurer le déclencheur Amazon S3 (AWS CLI)

1. Pour que votre compartiment source Amazon S3 invoque votre fonction lorsque vous ajoutez un fichier image, vous devez d'abord configurer les autorisations pour votre fonction à l'aide d'une politique basée sur une ressource. Une déclaration de politique basée sur les ressources donne à d'autres personnes Services AWS l'autorisation d'invoquer votre fonction. Pour autoriser Amazon S3 à appeler votre fonction, exécutez la CLI commande suivante. Assurez-vous de remplacer le `source-account` paramètre par votre propre Compte AWS identifiant et d'utiliser votre propre nom de compartiment source.

```
aws lambda add-permission --function-name CreateThumbnail \
--principal s3.amazonaws.com --statement-id s3invoke --action
 "lambda:InvokeFunction" \
--source-arn arn:aws:s3:::amzn-s3-demo-source-bucket \
--source-account 123456789012
```

La politique que vous définissez à l'aide de cette commande permet à Amazon S3 d'invoquer votre fonction uniquement lorsqu'une action a lieu sur votre compartiment source.

> **Note**
>
> Bien que les noms des compartiments Amazon S3 soient globalement uniques, il est préférable de spécifier que le compartiment doit appartenir à votre compte lorsque vous utilisez des politiques basées sur les ressources. En effet, si vous supprimez un bucket, il est possible qu'un autre Compte AWS en crée un avec le même Amazon Resource Name (ARN).

2. Enregistrez ce qui suit JSON dans un fichier nomménotification.json. Lorsqu'il est appliqué à votre compartiment source, cela JSON configure le compartiment pour qu'il envoie une notification à votre fonction Lambda chaque fois qu'un nouvel objet est ajouté. Remplacez le Compte AWS numéro et Région AWS dans la fonction Lambda par votre propre numéro ARN de compte et votre propre région.

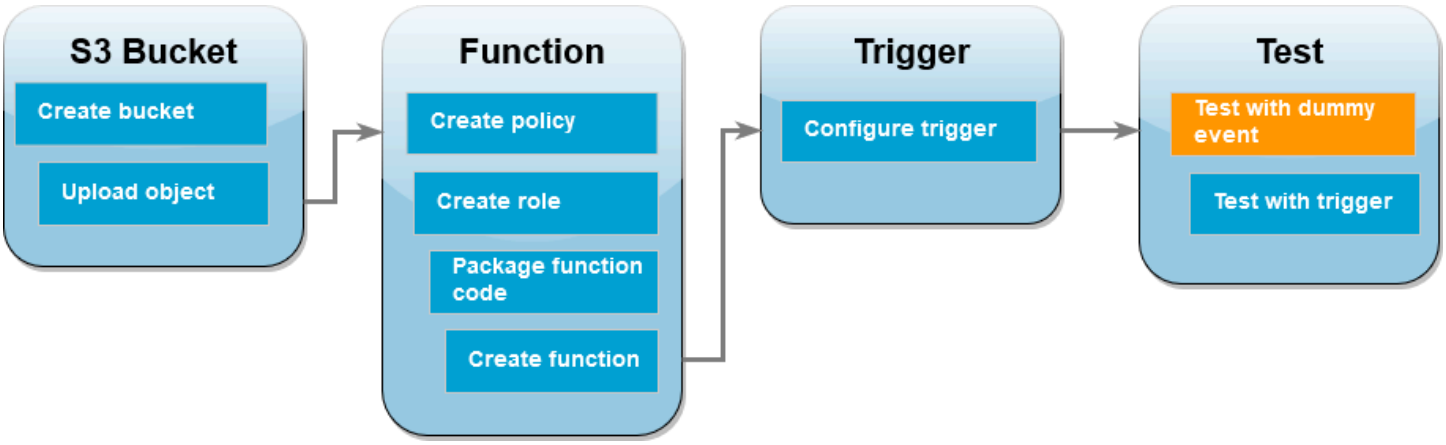
```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Exécutez la CLI commande suivante pour appliquer les paramètres de notification du JSON fichier que vous avez créé à votre compartiment source. Remplacez `amzn-s3-demo-source-bucket` par le nom de votre propre compartiment source.

```
aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-source-
bucket \
--notification-configuration file://notification.json
```

Pour en savoir plus sur la `put-bucket-notification-configuration` commande et l'`notification-configuration`option, consultez put-bucket-notification-configurationla référence des AWS CLI commandes.

## Test de votre fonction Lambda à l'aide d'un événement fictif




Avant de tester l'ensemble de votre configuration en ajoutant un fichier image à votre compartiment source Amazon S3, vous devez vérifier que votre fonction Lambda fonctionne correctement en l'invoquant avec un événement fictif. Dans Lambda, un événement est un document JSON formaté qui contient des données à traiter par votre fonction. Lorsque votre fonction est invoquée par Amazon S3, l'événement envoyé à votre fonction contient des informations telles que le nom du compartiment, le compartiment ARN et la clé de l'objet.

AWS Management Console

Pour tester votre fonction Lambda à l'aide d'un événement fictif (console)

1. Ouvrez la page Fonctions de la console Lambda et choisissez votre fonction (`CreateThumbnail`).
2. Choisissez l'onglet Test.
3. Pour créer votre événement de test, dans le volet Événement de test, procédez comme suit :
   a. Sous Action d'événement de test, sélectionnez Créer un nouvel événement.
   b. Dans Event name (Nom de l'événement), saisissez **`myTestEvent`**.
   c. Pour Modèle, sélectionnez S3 Put.
   d. Remplacez les valeurs des paramètres suivants par vos propres valeurs.
      - Pour `awsRegion`, remplacez-le `us-east-1` par celui dans Région AWS lequel vous avez créé vos compartiments Amazon S3.
      - Pour `name`, remplacez `amzn-s3-demo-bucket` par le nom de votre propre compartiment source Amazon S3.

- Pour `key`, remplacez `test%2Fkey` par le nom de fichier de l'objet de test que vous avez chargé dans votre compartiment source à l'étape Charger une image de test dans votre compartiment source.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
```

```
}
```

e. Choisissez Save (Enregistrer).

4. Dans le volet Événement de test, choisissez Test.
5. Pour vérifier que votre fonction a créé une version redimensionnée de votre image et l'a stockée dans votre compartiment Amazon S3 cible, procédez comme suit :

   a. Ouvrez la page Compartiments de la console Amazon S3.

   b. Choisissez votre compartiment cible et vérifiez que votre fichier redimensionné est répertorié dans le volet Objets.

AWS CLI

Pour tester de votre fonction Lambda à l'aide d'un événement fictif (AWS CLI)

1. Enregistrez ce qui suit JSON dans un fichier nommé`dummyS3Event.json`. Remplacez les valeurs des paramètres suivants par vos propres valeurs :

   - Pour`awsRegion`, remplacez-le `us-east-1` par celui dans Région AWS lequel vous avez créé vos compartiments Amazon S3.
   - Pour `name`, remplacez `amzn-s3-demo-bucket` par le nom de votre propre compartiment source Amazon S3.
   - Pour `key`, remplacez `test%2Fkey` par le nom de fichier de l'objet de test que vous avez chargé dans votre compartiment source à l'étape Charger une image de test dans votre compartiment source.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
```

```
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

2. Dans le répertoire dans lequel vous avez enregistré votre `dummyS3Event.json` fichier, appelez la fonction en exécutant la CLI commande suivante. Cette commande invoque votre fonction Lambda de manière synchrone en la spécifiant `RequestResponse` comme valeur du paramètre de type d'invocation. Pour en savoir plus sur l'invocation synchrone et asynchrone, consultez Invocation de fonctions Lambda.

```
aws lambda invoke --function-name CreateThumbnail \
--invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
--payload file://dummyS3Event.json outputfile.txt
```

cli-binary-formatCette option est obligatoire si vous utilisez la version 2 du AWS CLI. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, veuillez consulter les AWS CLI options de ligne de commande.

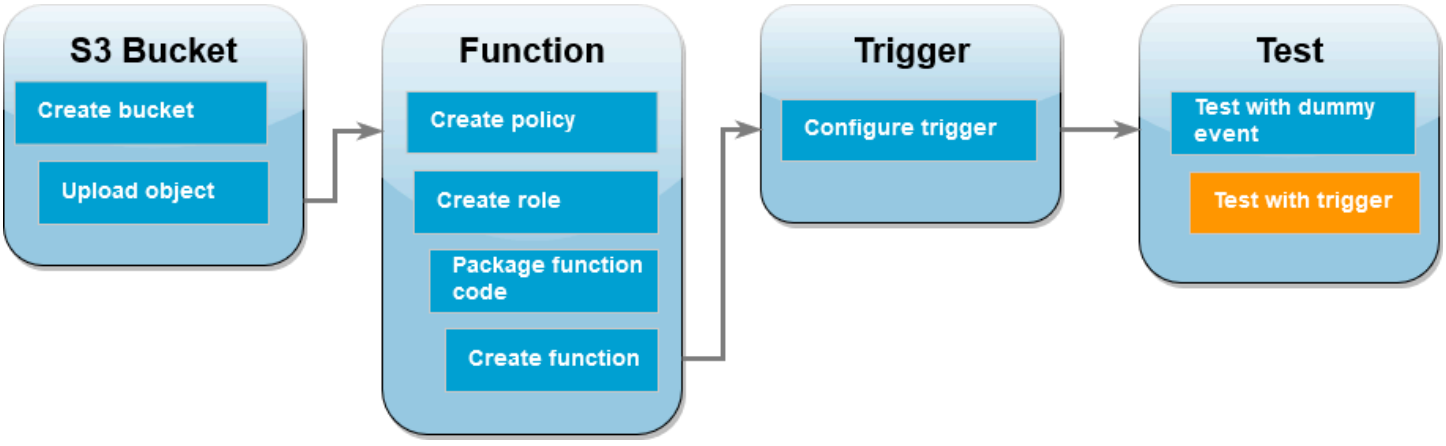
3. Vérifiez que votre fonction a créé une version miniature de votre image et l'a enregistrée dans votre compartiment Amazon S3 cible. Exécutez la CLI commande suivante en la `amzn-s3-demo-source-bucket-resized` remplaçant par le nom de votre propre compartiment de destination.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Vous devez visualiser des résultats similaires à ce qui suit. Le paramètre `Key` indique le nom de fichier de votre fichier image redimensionné.

```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-06T21:40:07+00:00",
            "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",
            "Size": 2633,
            "StorageClass": "STANDARD"
        }
    ]
}
```

## Tester votre fonction à l'aide du déclencheur Amazon S3




Maintenant que vous avez confirmé que votre fonction Lambda fonctionne correctement, vous êtes prêt à tester votre configuration complète en ajoutant un fichier image à votre compartiment source Amazon S3. Lorsque vous ajoutez votre image au compartiment source, votre fonction Lambda doit

être automatiquement invoquée. Votre fonction crée une version redimensionnée du fichier et la stocke dans votre compartiment cible.

AWS Management Console

Pour tester votre fonction Lambda à l'aide du déclencheur Amazon S3 (console)

1. Pour charger une image dans votre compartiment Amazon S3, procédez comme suit :
   a. Ouvrez la page Compartiments de la console Amazon S3 et choisissez votre compartiment source.
   b. Sélectionnez Charger.
   c. Choisissez Ajouter des fichiers et utilisez le sélecteur de fichiers pour sélectionner le fichier image que vous souhaitez charger. Votre objet image peut être n'importe quel fichier .jpg ou .png.
   d. Choisissez Ouvrir, puis Charger.
2. Vérifiez que Lambda a enregistré une version redimensionnée de votre fichier image dans votre compartiment cible en procédant comme suit :
   a. Retournez à la page Compartiments de la console Amazon S3 et choisissez votre compartiment de destination.
   b. Dans le volet Objets, vous devriez maintenant voir deux fichiers image redimensionnés, un pour chaque test de votre fonction Lambda. Pour télécharger votre image redimensionnée, sélectionnez le fichier, puis choisissez Télécharger.

AWS CLI

Pour tester votre fonction Lambda à l'aide du déclencheur Amazon S3 (AWS CLI)

1. Dans le répertoire contenant l'image que vous souhaitez télécharger, exécutez la CLI commande suivante. Remplacez le paramètre `--bucket` par le nom de votre compartiment source. Pour les paramètres `--key` et `--body`, utilisez le nom de fichier de votre image de test. Votre image de test peut être n'importe quel fichier .jpg ou .png.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key SmileyFace.jpg --
body ./SmileyFace.jpg
```

2. Vérifiez que votre fonction a créé une version miniature de votre image et l'a enregistrée dans votre compartiment Amazon S3 cible. Exécutez la CLI commande suivante en la `amzn-s3-demo-source-bucket-resized` remplaçant par le nom de votre propre compartiment de destination.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Si votre fonction s'exécute correctement, vous obtiendrez un résultat similaire à celui qui suit. Votre compartiment cible doit désormais contenir deux fichiers redimensionnés.

```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-07T00:15:50+00:00",
            "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
            "Size": 2763,
            "StorageClass": "STANDARD"
        },
        {
            "Key": "resized-SmileyFace.jpg",
            "LastModified": "2023-06-07T00:13:18+00:00",
            "ETag": "\"ca536e5a1b9e32b22cd549e18792cdbc\"",
            "Size": 1245,
            "StorageClass": "STANDARD"
        }
    ]
}
```

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.

3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Suppression de la stratégie que vous avez créée

1. Ouvrez la page Politiques de la IAM console.
2. Sélectionnez la politique que vous avez créée (AWSLambdaS3Policy).
3. Choisissez Actions de stratégie, Supprimer.
4. Sélectionnez Delete (Supprimer).

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d’exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer le compartiment S3

1. Ouvrez la console Amazon S3.
2. Sélectionnez le compartiment que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du compartiment dans le champ de saisie de texte.
5. Choisissez Supprimer le compartiment.

# Utilisation de Lambda avec Amazon SQS

> **Note**
>
> Si vous souhaitez envoyer des données à une cible autre qu'une fonction Lambda ou enrichir les données avant de les envoyer, consultez Amazon EventBridge Pipes.

Vous pouvez utiliser une fonction Lambda pour traiter les messages dans une file d'attente Amazon Simple Queue Service SQS (Amazon). Lambda prend en charge à la fois les files d'attente standard et les files d'attentepremier entré, premier sorti (FIFO) pour les mappages de sources d'événements. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques Région AWS, bien qu'elles puissent être différentes. Comptes AWS

Rubriques


- Comprendre le comportement d'interrogation et de traitement par lots pour les mappages de sources SQS d'événements Amazon
- Exemple d’événement de message de file d’attente standard
- Exemple FIFO d'événement de message de file d'attente
- Création et configuration d'un mappage des sources d'SQSévénements Amazon
- Configuration du comportement de dimensionnement pour les mappages de sources d'SQSévénements
- Gestion des erreurs pour une source d'SQSévénements dans Lambda
- Paramètres Lambda pour les mappages de sources d'SQSévénements Amazon
- Utilisation du filtrage des événements avec une source d'SQSévénements Amazon
- Tutoriel : Utilisation de Lambda avec Amazon SQS
- Tutoriel : Utilisation d'une SQS file d'attente Amazon multi-comptes comme source d'événements


## Comprendre le comportement d'interrogation et de traitement par lots pour les mappages de sources SQS d'événements Amazon

Avec les mappages de sources d'SQSévénements Amazon, Lambda interroge la file d'attente et appelle votre fonction de manière synchrone avec un événement. Chaque événement peut contenir

un lot de plusieurs messages provenant de la file d'attente. Lambda reçoit ces événements un lot à la fois et appelle votre fonction une fois pour chaque lot. Après que la fonction a traité un lot avec succès, Lambda supprime ses messages de la file d'attente.

Lorsque Lambda reçoit un lot, les messages restent dans la file d'attente mais sont masqués pendant la durée du délai de visibilité de la file d'attente. Si votre fonction traite correctement tous les messages du lot, Lambda les supprime de la file d'attente. Par défaut, si votre fonction rencontre une erreur lors du traitement d'un lot, tous les messages de ce lot redeviennent visibles dans la file d'attente une fois le délai de visibilité expiré. Pour cette raison, le code de votre fonction peut traiter le même message plusieurs fois sans effets secondaires involontaires.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Pour empêcher Lambda de traiter un message plusieurs fois, vous pouvez soit configurer le mappage de votre source d'événements pour inclure les défaillances d'éléments de lot dans la réponse de votre fonction, soit utiliser le DeleteMessageAPIpour supprimer les messages de la file d'attente lorsque votre fonction Lambda les traite correctement.

Pour plus d'informations sur les paramètres de configuration pris en charge par Lambda pour les mappages de sources d'SQSévénements, consultez. the section called “Création d'un mappage des sources d'SQSévénements”

## Exemple d’événement de message de file d’attente standard

Example Événement de SQS message Amazon (file d'attente standard)

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "Test message.",
```

```
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        },
        {
            "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
            "receiptHandle": "AQEBzWwaftRI0KuVm4tP+/7q1rGgNqicHq...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082650636",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082650649"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

Par défaut, Lambda interroge jusqu'à 10 messages à la fois dans votre file d'attente, et envoie ce lot à votre fonction. Pour éviter d'invoquer la fonction avec un petit nombre d'enregistrements, vous pouvez configurer la source d'événements pour qu'elle mette en mémoire tampon les enregistrements pendant 5 minutes maximum en configurant une fenêtre de traitement par lots. Avant d'appeler la fonction, Lambda continue d'interroger les messages de la file d'attente standard jusqu'à ce que la fenêtre de traitement par lots expire, que le quota de charge utile d'appel soit atteint ou que la taille de lot maximale configurée soit atteinte.

Si vous utilisez une fenêtre de traitement par lots et que votre SQS file d'attente contient un trafic très faible, Lambda peut attendre jusqu'à 20 secondes avant d'appeler votre fonction. C'est le cas même si vous définissez une fenêtre de lot inférieure à 20 secondes.

> **Note**
>
> En Java, vous pouvez rencontrer des erreurs de pointeur nul lors de la désérialisationJSON. Cela peut être dû à la façon dont les cases « Records » et « eventSource ARN » sont converties par le mappeur JSON d'objets.

## Exemple FIFO d'événement de message de file d'attente

Pour les FIFO files d'attente, les enregistrements contiennent des attributs supplémentaires liés à la déduplication et au séquençage.

Example Événement de SQS message Amazon (FIFOfile d'attente)

```
{
    "Records": [
        {
            "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
            "receiptHandle": "AQEBBX8nesZEXmkhsmZeyIE8iQAMig7qw...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1573251510774",
                "SequenceNumber": "18849496460467696128",
                "MessageGroupId": "1",
                "SenderId": "AIDAIO23YVJENQZJOL4VO",
                "MessageDeduplicationId": "1",
                "ApproximateFirstReceiveTimestamp": "1573251510774"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo.fifo",
            "awsRegion": "us-east-2"
        }
    ]
}
```

# Création et configuration d'un mappage des sources d'SQSévénements Amazon

Pour traiter les SQS messages Amazon avec Lambda, configurez votre file d'attente avec les paramètres appropriés, puis créez un mappage des sources d'événements Lambda.

## Configuration d’une file d’attente à utiliser avec Lambda

Si vous n'avez pas encore de SQS file d'attente Amazon, créez-en une qui servira de source d'événements pour votre fonction Lambda. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques Région AWS, bien qu'elles puissent être différentes. Comptes AWS

Pour laisser le temps à votre fonction de traiter chaque lot d'enregistrements, définissez le délai de visibilité de la file d'attente source à au moins six fois le délai de configuration de votre fonction. Ce délai supplémentaire permet à Lambda de réessayer si votre fonction est limitée lors du traitement d'un lot précédent.

Par défaut, si Lambda rencontre une erreur à un moment quelconque lors du traitement d'un lot, tous les messages de ce lot retournent dans la file d'attente. Après l'expiration du délai de visibilité, les messages redeviennent visibles pour Lambda. Vous pouvez configurer le mappage de votre source d'événements pour utiliser des réponses par lots partielles afin de renvoyer uniquement les messages ayant échoué dans la file d'attente. En outre, si votre fonction ne parvient pas à traiter un message plusieurs fois, Amazon SQS peut l'envoyer dans une file d'attente de lettres mortes. Nous vous recommandons de définir `maxReceiveCount` la politique de redrive de votre file d'attente source sur au moins 5. Cela donne à Lambda quelques chances de réessayer avant d'envoyer les messages ayant échoué directement à la file d'attente des lettres mortes.

## Configuration des autorisations du rôle d'exécution Lambda

La politique AWSLambdaSQSQueueExecutionRole AWS gérée inclut les autorisations dont Lambda a besoin pour lire les informations de votre file d'attente AmazonSQS. Vous pouvez ajouter cette politique gérée au rôle d'exécution de votre fonction.

En option, si vous utilisez une file d’attente chiffrée, vous devez également ajouter l’autorisation suivante à votre rôle d’exécution :

- kms:Decrypt

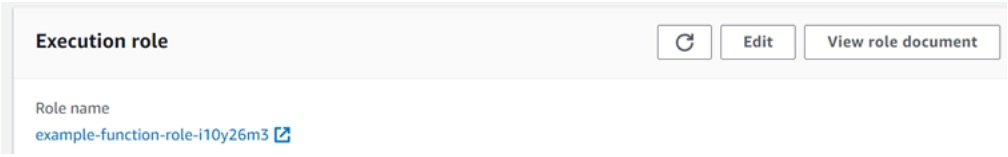
## Création d'un mappage des sources d'SQSévénements

Créez un mappage de source d'événement pour indiquer à Lambda d'envoyer des éléments de votre file d'attente à une fonction Lambda. Vous pouvez créer plusieurs mappages de source d'événement pour traiter des éléments de plusieurs files d'attente avec une seule fonction. Quand Lambda invoque la fonction cible, l'événement peut contenir plusieurs éléments, jusqu'à une taille de lot maximale configurable.

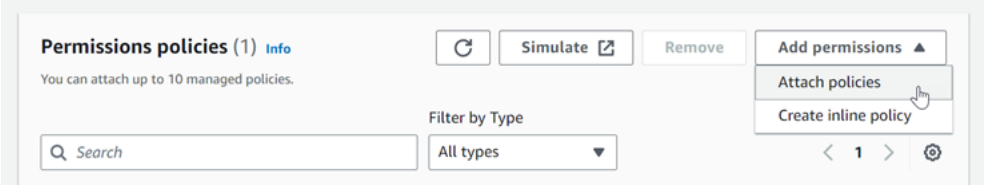
Pour configurer votre fonction afin qu'elle puisse lire depuis AmazonSQS, associez la politique AWSLambdaSQSQueueExecutionRole AWS gérée à votre rôle d'exécution. Créez ensuite un mappage des sources d'SQSévénements à partir de la console en suivant les étapes suivantes.

Pour ajouter des autorisations et créer un déclencheur

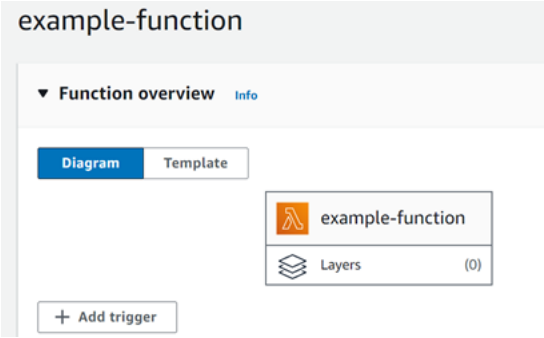
1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Choisissez l'onglet Configuration, puis Permissions (Autorisations).
4. Sous Nom du rôle, choisissez le lien vers votre rôle d'exécution. Ce lien ouvre le rôle dans la IAM console.




5. Choisissez Ajouter des autorisations, puis Attacher des politiques.




6. Dans le champ de recherche, entrez `AWSLambdaSQSQueueExecutionRole`. Ajoutez cette politique à votre rôle d'exécution. Il s'agit d'une politique AWS gérée qui contient les autorisations dont votre fonction a besoin pour lire des informations depuis une SQS file d'attente Amazon. Pour plus d'informations sur cette politique, consultez AWSLambdaSQSQueueExecutionRolela référence des politiques AWS gérées.
7. Retournez à votre fonction dans la console Lambda. Sous Function overview (Vue d'ensemble de la fonction), choisissez Add trigger (Ajouter un déclencheur).




8. Choisissez un type de déclencheur.
9. Configurez les options requises, puis choisissez Add (Ajouter).

Lambda prend en charge les options de configuration suivantes pour les sources d'SQSévénements Amazon :

SQSfile d'attente

La SQS file d'attente Amazon à partir de laquelle lire les enregistrements. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques Région AWS, bien qu'elles puissent être différentes. Comptes AWS

Activation du déclencheur

L'état du mappage des sources d'événements. Activez le déclencheur est sélectionné par défaut.

Taille de lot

Le nombre maximum d'enregistrements à envoyer à la fonction dans chaque lot. Pour une file d'attente standard, cela peut aller jusqu'à 10 000 registres. Pour une FIFO file d'attente, le maximum est de 10. Pour une taille de lot supérieure à 10, vous devez également définir la fenêtre de lot (`MaximumBatchingWindowInSeconds`) sur au moins 1 seconde.

Configurez le délai d'expiration de votre fonction afin de disposer de suffisamment de temps pour traiter un lot complet d'articles. Si les éléments sont longs à traiter, choisissez une taille de lot plus petite. Une grande taille de lot peut améliorer l'efficacité pour des charges de travail qui sont très rapides ou qui induisent beaucoup d'efforts supplémentaires. Si vous configurez une simultanéité réservée sur votre fonction, définissez un minimum de cinq exécutions simultanées pour réduire le risque d'erreurs de limitation lorsque Lambda invoque votre fonction.

Lambda transmet tous les enregistrements du lot à la fonction en un seul appel, à condition que la taille totale des événements ne dépasse pas le quota de charge utile d'appel pour un

appel synchrone (6 Mo). Lambda et Amazon SQS génèrent des métadonnées pour chaque enregistrement. Ces métadonnées supplémentaires sont comptabilisées dans la taille de charge utile totale, ce qui peut entraîner l'envoi dans un lot d'un nombre total d'enregistrements inférieur à la taille du lot configuré. Les champs de métadonnées SQS envoyés par Amazon peuvent être de longueur variable. Pour plus d'informations sur les champs de SQS métadonnées Amazon, consultez la documentation relative aux ReceiveMessageAPIopérations dans le manuel Amazon Simple Queue Service API Reference.

Fenêtre de lot

Intervalle de temps maximal (en secondes) pour collecter des enregistrements avant d'invoquer la fonction. Cela s'applique uniquement aux files d'attente standards.

Si vous utilisez une fenêtre de traitement par lots supérieure à 0 seconde, vous devez tenir compte de l'augmentation du temps de traitement dans le délai de visibilité de votre file d'attente. Nous vous recommandons de paramétrer votre délai de visibilité de file d'attente à six fois le délai d'expiration de la fonction, en plus de la valeur de `MaximumBatchingWindowInSeconds`. Cela permet à votre fonction Lambda de traiter chaque lot d'événements et de réessayer en cas d'erreur de limitation.

Lorsque les messages sont disponibles, Lambda commence à les traiter par lots. Lambda commence à traiter cinq lots à la fois avec cinq invocations simultanés de votre fonction. Si les messages sont toujours disponibles, Lambda ajoute jusqu'à 300 instances supplémentaires de votre fonction par minute, jusqu'à un maximum de 1 000 instances de fonction supplémentaires. Pour en savoir plus sur la mise à l'échelle et la simultanéité de la fonction, consultez Mise à l'échelle de fonction Lambda.

Pour traiter plus de messages, vous pouvez optimiser votre fonction Lambda pour un débit plus élevé. Pour plus d'informations, consultez Comprendre comment s' AWS Lambda adapte aux files d'attente SQS standard d'Amazon.

Simultanéité maximum

Le nombre maximum de fonctions simultanées que la source d'événement peut invoquer. Pour de plus amples informations, veuillez consulter Configuration de la simultanéité maximale pour les sources d'SQSévénements Amazon.

Critères de filtrage

Ajoutez des critères de filtrage pour contrôler les événements que Lambda envoie à votre fonction pour traitement. Pour de plus amples informations, veuillez consulter Contrôlez les événements que Lambda envoie à votre fonction.

# Configuration du comportement de dimensionnement pour les mappages de sources d'SQSévénements

Pour les files d'attente standard, Lambda utilise de longues interrogations pour interroger une file d'attente jusqu'à ce qu'elle devienne active. Lorsque les messages sont disponibles, Lambda commence à traiter cinq lots à la fois avec cinq invocations simultanées de votre fonction. Si les messages sont toujours disponibles, Lambda augmente le nombre de processus de lecture de lots jusqu'à 300 instances supplémentaires par minute. Le nombre maximum de lots qui peut être traité simultanément par un mappage de source d'événement est de 1 000.

Pour les FIFO files d'attente, Lambda envoie des messages à votre fonction dans l'ordre dans lequel elle les reçoit. Lorsque vous envoyez un message à une FIFO file d'attente, vous spécifiez un ID de groupe de messages. Amazon SQS veille à ce que les messages d'un même groupe soient transmis à Lambda dans l'ordre. Lorsque Lambda lit vos messages par lots, chaque lot peut contenir des messages provenant de plusieurs groupes de messages, mais l'ordre des messages est conservé. Si la fonction renvoie une erreur, toutes les nouvelles tentatives sont effectuées sur les messages concernés avant que Lambda reçoive des messages supplémentaires du même groupe.

## Configuration de la simultanéité maximale pour les sources d'SQSévénements Amazon

Vous pouvez utiliser le paramètre de simultanéité maximale pour contrôler le comportement de dimensionnement de vos sources SQS d'événements. Le paramètre de simultanéité maximale limite le nombre d'instances simultanées de la fonction qu'une source d'SQSévénements Amazon peut invoquer. La simultanéité maximale est un paramètre de niveau source d'événement. Si plusieurs sources d'SQSévénements Amazon sont mappées à une fonction, chaque source d'événements peut avoir un paramètre de simultanéité maximale distinct. Vous pouvez utiliser la simultanéité maximale pour empêcher une file d'attente d'utiliser toute la simultanéité réservée de la fonction ou le reste du quota de simultanéité du compte. La configuration d'une simultanéité maximale sur une source d'SQSévénements Amazon est gratuite.

Il est important de noter que la simultanéité maximale et la simultanéité réservée sont deux paramètres indépendants. Ne définissez pas une simultanéité maximale supérieure à la simultanéité réservée de la fonction. Si vous configurez la simultanéité maximale, assurez-vous que la simultanéité réservée à votre fonction est supérieure ou égale à la simultanéité maximale totale pour toutes les sources d'SQSévénements Amazon associées à la fonction. Sinon, Lambda peut limiter vos messages.

Lorsque le quota de simultanéité de votre compte est défini sur la valeur par défaut de 1 000, un mappage de source d'SQSévénements Amazon peut être redimensionné pour invoquer des instances de fonction jusqu'à cette valeur, sauf si vous spécifiez une simultanéité maximale.

Si vous recevez une augmentation du quota de simultanéité par défaut de votre compte, Lambda risque de ne pas être en mesure d'invoquer des instances de fonctions concurrentes jusqu'à votre nouveau quota. Par défaut, Lambda peut évoluer pour invoquer jusqu'à 1 250 instances de fonctions simultanées pour un mappage de source d'SQSévénements Amazon. Si cela ne correspond pas à votre cas d'utilisation, contactez le AWS support pour discuter d'une augmentation de la simultanéité du mappage des sources d'SQSévénements Amazon de votre compte.

> **Note**
>
> Pour les FIFO files d'attente, les appels simultanés sont plafonnés soit par le nombre de groupes de messages IDs (`messageGroupId`), soit par le paramètre de simultanéité maximal, selon le plus faible des deux. Par exemple, si vous avez six groupes de messages IDs et que la simultanéité maximale est fixée à 10, votre fonction peut avoir un maximum de six appels simultanés.

Vous pouvez configurer une simultanéité maximale sur les mappages de sources d'SQSévénements Amazon nouveaux et existants.

Configuration de la simultanéité maximale à l'aide de la console Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Sous Vue d'ensemble des fonctions, sélectionnez SQS. Cela ouvre l'onglet Configuration.
4. Sélectionnez le SQS déclencheur Amazon, puis choisissez Modifier.
5. Pour Maximum concurrency (Simultanéité maximale), saisissez un nombre compris entre 2 et 1 000. Pour désactiver la simultanéité maximale, laissez la case vide.
6. Choisissez Save (Enregistrer).

Configurez la simultanéité maximale à l'aide de AWS Command Line Interface ()AWS CLI

Utilisez la update-event-source-mappingcommande avec l'`--scaling-config`option. Exemple :

```
aws lambda update-event-source-mapping \
```

```
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config '{"MaximumConcurrency":5}'
```

Pour désactiver la simultanéité maximale, entrez une valeur vide pour `--scaling-config` :

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config "{}"
```

Configurer une simultanéité maximale à l'aide du Lambda API

Utilisez l'UpdateEventSourceMappingaction CreateEventSourceMappingou avec un ScalingConfigobjet.

## Gestion des erreurs pour une source d'SQSévénements dans Lambda

Pour gérer les erreurs liées à une source d'SQSévénement, Lambda utilise automatiquement une stratégie de réessai associée à une stratégie de temporisation. Vous pouvez également personnaliser le comportement de gestion des erreurs en configurant le mappage de votre source d'SQSévénements pour renvoyer des réponses partielles par lots.

### Stratégie d’interruption pour les invocations ayant échoué

Lorsqu’une invocation échoue, Lambda tente de réessayer l’invocation tout en mettant en œuvre une stratégie d’interruption. La stratégie d’interruption diffère légèrement selon que Lambda a rencontré l’échec à cause d’une erreur dans votre code de fonction, ou à cause de la limitation.

- Si votre code de fonction est à l'origine de l'erreur, Lambda arrêtera le traitement et réessaiera l'appel. Entre-temps, Lambda recule progressivement, réduisant ainsi le niveau de simultanéité alloué au mappage de votre source d'SQSévénements Amazon. Une fois le délai de visibilité de votre file d'attente expiré, le message réapparaîtra dans la file d'attente.
- Si l'invocation échoue en raison d'un ralentissement, Lambda annule progressivement les nouvelles tentatives en réduisant le niveau de simultanéité alloué au mappage de votre source d'événements Amazon. SQS Lambda continue à réessayer le message jusqu’à ce que l’horodatage du message dépasse le délai de visibilité de votre file d’attente, auquel cas Lambda abandonne le message.

## Mise en œuvre de réponses partielles de lot

Lorsque votre fonction Lambda rencontre une erreur lors du traitement d'un lot, tous les messages de ce lot redeviennent par défaut visibles dans la file d'attente, y compris les messages traités avec succès par Lambda. Par conséquent, votre fonction peut finir par traiter le même message plusieurs fois.

Pour éviter de retraiter les messages traités avec succès d'un lot en échec, vous pouvez configurer le mappage des sources d'événements pour que seuls les messages ayant échoué soient à nouveau visibles. C'est ce que l'on appelle une réponse partielle de lot. Pour activer les réponses partielles par lots, spécifiez `ReportBatchItemFailures` l'FunctionResponseTypesaction lors de la configuration du mappage des sources d'événements. Cela permet à votre fonction de renvoyer un succès partiel, ce qui peut contribuer à réduire le nombre de nouvelles tentatives inutiles sur les enregistrements.

Lorsque `ReportBatchItemFailures` est activé, Lambda ne réduit pas la taille de l'interrogation des messages lorsque les invocations de fonctions échouent. Si vous vous attendez à ce que certains messages échouent et que vous ne voulez pas que ces échecs aient un impact sur le taux de traitement des messages, utilisez `ReportBatchItemFailures`.

> **Note**
>
> Gardez les points suivants à l'esprit lorsque vous utilisez des réponses partielles de lot :
>
> - Si la fonction génère une exception, l'ensemble du lot est considéré comme un échec complet.
> - Si vous utilisez cette fonctionnalité avec une FIFO file d'attente, votre fonction doit arrêter de traiter les messages après le premier échec et renvoyer tous les messages ayant échoué ou non traités. `batchItemFailures` Cela permet de préserver l'ordre des messages dans votre file d'attente.

Pour activer les rapports partiels de lot

1. Consultez les bonnes pratiques pour la mise en œuvre des réponses partielles de lot.
2. Exécutez la commande suivante pour activer `ReportBatchItemFailures` pour votre fonction. Pour récupérer les mappages de vos sources d'événementsUUID, exécutez la list-event-source-mappings AWS CLI commande.

```
aws lambda update-event-source-mapping \
```

```
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
--function-response-types "ReportBatchItemFailures"
```

3. Mettez à jour votre code de fonction pour détecter toutes les exceptions et renvoyer les messages d'échec dans une `batchItemFailures` JSON réponse. La `batchItemFailures` réponse doit inclure une liste de messagesIDs, sous forme de `itemIdentifier` JSON valeurs.

   Supposons, par exemple, que vous disposiez d'un lot de cinq messages IDsid1, avec le message `id2id3`,`id4`,, et`id5`. Votre fonction traite avec succès `id1`, `id3` et `id5`. Pour rendre les messages `id2` et `id4` visibles de nouveau dans la file d'attente, votre fonction doit renvoyer la réponse suivante :

   ```
   {
     "batchItemFailures": [
           {
               "itemIdentifier": "id2"
           },
           {
               "itemIdentifier": "id4"
           }
       ]
   }
   ```

   Voici quelques exemples de code de fonction qui renvoie la liste des messages ayant échoué IDs dans le lot :

   .NET

   AWS SDK for .NET

   > **Note**
   >
   > Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

   Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. NET.

   ```
   // Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   // SPDX-License-Identifier: Apache-2.0
   ```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be
 converted into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
```

```
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
```

```
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

### Java

#### SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context)
 {

        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
```

```
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures
 list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(messageId));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. JavaScript

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
```

```
    }
    console.log(`Processed message: ${record.body}`);
}
```

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure,
 SQSRecord } from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
```

```
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS
 records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}
```

```
        for record in event["Records"]:
            try:
                # process message
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
```

```
  end
end
```

### Rust

#### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments SQS par lots avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
```

```
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

Si les événements ayant échoué ne retournent pas dans la file d'attente, voir Comment résoudre les problèmes liés à la fonction SQS ReportBatchItemFailures Lambda ? dans le AWS Knowledge Center.

Conditions de réussite et d'échec

Lambda traite un lot comme un succès complet si votre fonction renvoie l'un des éléments suivants :

- Une liste `batchItemFailures` vide
- Une liste `batchItemFailures` nulle
- Une `EventResponse` vide
- Une `EventResponse` nulle

Lambda traite un lot comme un échec complet si votre fonction renvoie l'un des éléments suivants :

- Une JSON réponse non valide
- Une chaîne `itemIdentifier` vide
- Un `itemIdentifier` nul
- Un `itemIdentifier` avec un nom de clé incorrect
- Une valeur `itemIdentifier` avec un ID de message inexistant

CloudWatch métriques

Pour déterminer si votre fonction signale correctement les défaillances d'articles par lots, vous pouvez surveiller les SQS statistiques `NumberOfMessagesDeleted` et `ApproximateAgeOfOldestMessage` Amazon sur Amazon CloudWatch.

- `NumberOfMessagesDeleted` suit le nombre de messages supprimés de votre file d'attente. Si cela tombe à 0, cela indique que la réponse de votre fonction ne renvoie pas correctement les messages d'échec.
- `ApproximateAgeOfOldestMessage` suit combien de temps le message le plus ancien est resté dans votre file d'attente. Une forte augmentation de cette métrique peut indiquer que votre fonction ne renvoie pas correctement les messages d'échec.

## Paramètres Lambda pour les mappages de sources d'SQSÉvénements Amazon

Tous les types de sources d'événements Lambda partagent les mêmes opérations CreateEventSourceMapping. UpdateEventSourceMappingAPI Toutefois, seuls certains paramètres s'appliquent à AmazonSQS.

| Paramètre | Obligatoire | Par défaut | Remarques |
|---|---|---|---|
| BatchSize | N | 10 | Pour les files d'attente standard, le maximum est de 10 000.<br>Pour les FIFO files d'attente, le maximum est de 10. |
| Activées | N | VRAI | none |
| EventSourceArn | Y | N/A | Le flux ARN de données ou le consommateur du flux |
| FunctionName | Y | N/A | none |
| FilterCriteria | N | N/A | Contrôlez les événements que Lambda envoie à votre fonction |

| Paramètre | Obligatoire | Par défaut | Remarques |
| --- | --- | --- | --- |
| FunctionResponseTypes | N | N/A | Pour permettre à votre fonction de signaler des échecs spécifiques dans un lot, incluez la valeur `ReportBatchItemFailures` dans `FunctionResponseTypes` . Pour de plus amples informations, veuillez consulter Mise en œuvre de réponses partielles de lot. |
| MaximumBatchingWindowInSeconds | N | 0 | none |
| ScalingConfig | N | N/A | Configuration de la simultanéité maximale pour les sources d'SQSévénements Amazon |

## Utilisation du filtrage des événements avec une source d'SQSévénements Amazon

Vous pouvez utiliser le filtrage d'événements pour contrôler les enregistrements d'un flux ou d'une file d'attente que Lambda envoie à votre fonction. Pour des informations générales sur le fonctionnement du filtrage des événements, consultezthe section called "Filtrage des événements".

Cette section se concentre sur le filtrage des événements pour les sources MSK d'événements Amazon.

Rubriques

- Principes de base du filtrage des SQS événements Amazon

## Principes de base du filtrage des SQS événements Amazon

Supposons que votre SQS file d'attente Amazon contienne des messages au JSON format suivant.

```
{
    "RecordNumber": 1234,
    "TimeStamp": "yyyy-mm-ddThh:mm:ss",
    "RequestCode": "AAAA"
}
```

Un exemple d'enregistrement pour cette file d'attente ressemblerait à ce qui suit.

```
{
    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
    "body": "{\n "RecordNumber": 1234,\n "TimeStamp": "yyyy-mm-ddThh:mm:ss",\n
 "RequestCode": "AAAA"\n}",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
        },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:my-queue",
    "awsRegion": "us-west-2"
}
```

Pour filtrer en fonction du contenu de vos SQS messages Amazon, utilisez la `body` clé figurant dans l'enregistrement des SQS messages Amazon. Supposons que vous souhaitiez traiter uniquement les enregistrements contenant le « » `RequestCode` dans votre SQS message AmazonBBBB. L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
```

```
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "body": {
        "RequestCode": [ "BBBB" ]
        }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

AWS SAM

Pour ajouter ce filtre à l'aide de AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Supposons que vous vouliez que votre fonction ne traite que les enregistrements où `RecordNumber` est supérieur à 9 999. L'objet `FilterCriteria` serait le suivant.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\": [ \">\",
 9999 ] } ] } }"
        }
    ]
}
```

Pour plus de clarté, voici la valeur du filtre `Pattern` expansé en clairJSON.

```
{
    "body": {
        "RecordNumber": [
            {
                "numeric": [ ">", 9999 ]
            }
        ]
    }
}
```

Vous pouvez ajouter votre filtre à l'aide de la console AWS CLI ou d'un AWS SAM modèle.

Console

Pour ajouter ce filtre à l'aide de la console, suivez les instructions de Attacher des critères de filtre à un mappage de sources d'événements (console) et saisissez la chaîne suivante comme critère de filtre.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

AWS CLI

Pour créer un nouveau mappage de source d'événements avec ces critères de filtre à l'aide de AWS Command Line Interface (AWS CLI), exécutez la commande suivante.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

Pour ajouter ces critères de filtre à un mappage des sources d'événements existant, exécutez la commande suivante.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

AWS SAM

Pour ajouter ce filtre à l'aide de AWS SAM, ajoutez l'extrait suivant au YAML modèle de votre source d'événement.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

Pour AmazonSQS, le corps du message peut être n'importe quelle chaîne. Toutefois, cela peut être problématique si vous vous `FilterCriteria` attendez `body` à ce que le JSON format soit valide. Le scénario inverse est également vrai : si le corps du message entrant est JSON formaté mais que vos critères de filtrage s'attendent `body` à être une chaîne simple, cela peut entraîner un comportement involontaire.

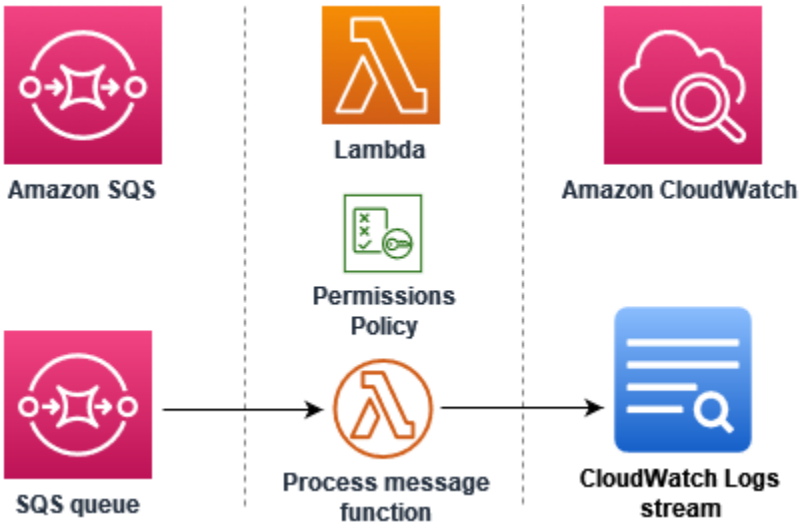
Pour éviter ce problème, assurez-vous que le format du corps dans vos `FilterCriteria` correspond au format attendu de `body` dans les messages que vous recevez de votre file d'attente.

Avant de filtrer vos messages, Lambda évalue automatiquement le format du corps de votre message entrant et de votre modèle de filtre pour `body`. En cas de décalage, Lambda abandonne le message. Le tableau suivant résume cette évaluation :

| Format du **body** du message entrant | Format du **body** du modèle de filtre | Action obtenue. |
|---|---|---|
| Chaîne de texte brut | Chaîne de texte brut | Lambda filtre en fonction de vos critères de filtre. |
| Chaîne de texte brut | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Chaîne de texte brut | Valide JSON | Lambda rejette le message. |
| Valide JSON | Chaîne de texte brut | Lambda rejette le message. |
| Valide JSON | Pas de modèle de filtre pour les propriétés des données | Lambda filtre (uniquement selon les autres propriétés de métadonnées) en fonction de vos critères de filtre. |
| Valide JSON | Valide JSON | Lambda filtre en fonction de vos critères de filtre. |

## Tutoriel : Utilisation de Lambda avec Amazon SQS

Dans ce didacticiel, vous allez créer une fonction Lambda qui consomme les messages d'une file d'attente Amazon Simple Queue Service SQS (Amazon). La fonction Lambda s’exécute chaque fois qu’un nouveau message est ajouté à la file d’attente. La fonction écrit les messages dans un flux Amazon CloudWatch Logs. Le diagramme suivant montre les ressources AWS utilisées pour compléter ce tutoriel.




Pour compléter ce didacticiel, effectuez les tâches suivantes :

1. Créez une fonction Lambda qui écrit des messages dans Logs. CloudWatch
2. Créez une SQS file d'attente Amazon.
3. Créez un mappage des sources d'événements Lambda. Le mappage de la source d'événements lit la SQS file d'attente Amazon et appelle votre fonction Lambda lorsqu'un nouveau message est ajouté.
4. Testez la configuration en ajoutant des messages à votre file d'attente et en surveillant les résultats dans CloudWatch Logs.

## Prérequis

### Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources

de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu‘utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d’un ensemble d’autorisations dans le Guide de l’utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l’utilisateur AWS IAM Identity Center .

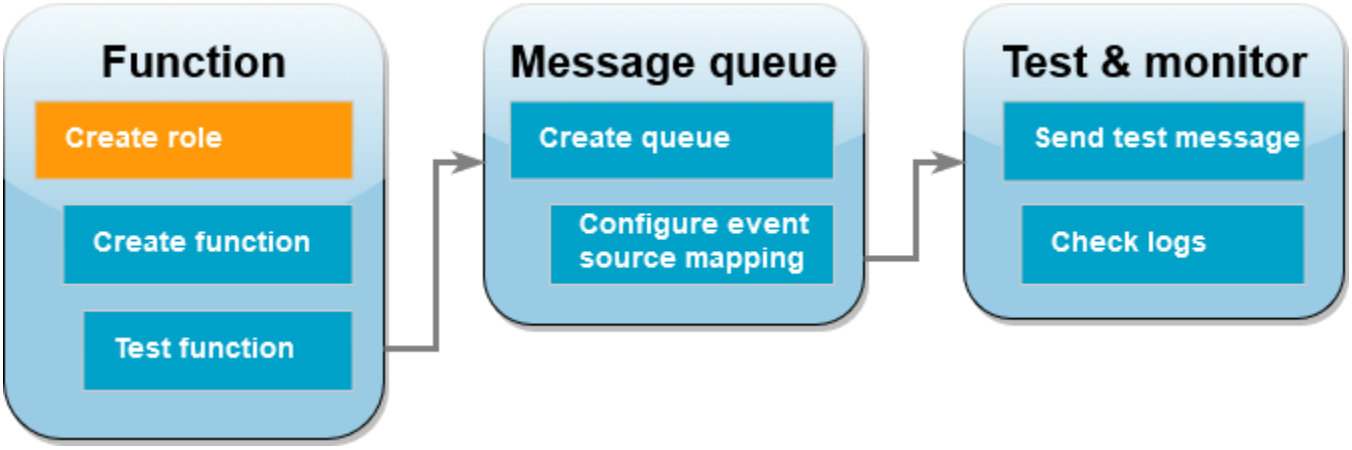
Installez le AWS Command Line Interface

Si vous ne l'avez pas encore installé AWS Command Line Interface, suivez les étapes décrites dans la section Installation ou mise à jour de la dernière version du AWS CLI pour l'installer.

Ce tutoriel nécessite un terminal de ligne de commande ou un shell pour exécuter les commandes. Sous Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d’obtenir une version intégrée à Windows d’Ubuntu et Bash.

## Créer le rôle d’exécution



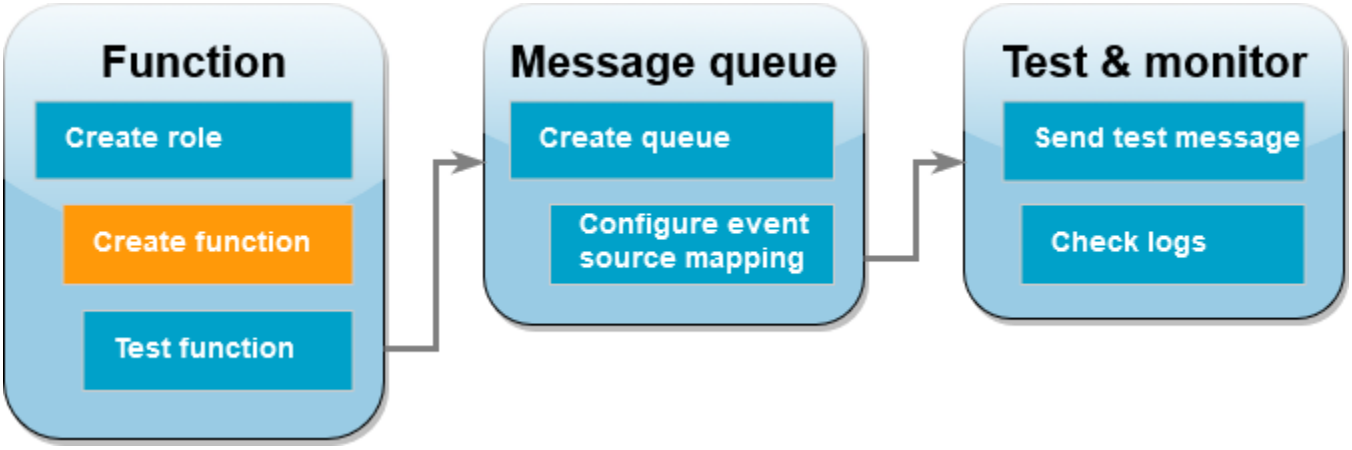

Un rôle d'exécution est un rôle AWS Identity and Access Management (IAM) qui accorde à une fonction Lambda l'autorisation d'accéder aux AWS services et aux ressources. Pour autoriser votre fonction à lire des articles provenant d'AmazonSQS, joignez la politique AWSLambdaSQSQueueExecutionRoled'autorisation.

Pour créer un rôle d'exécution et y associer une politique d'SQSautorisation Amazon

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez Créer un rôle.
3. Pour Type d’entité de confiance, choisissez Service AWS .
4. Pour Cas d’utilisation, choisissez Lambda.
5. Choisissez Suivant.
6. Dans le champ de recherche Politiques d’autorisations, saisissez **`AWSLambdaSQSQueueExecutionRole`**.
7. Sélectionnez la AWSLambdaSQSQueueExecutionRolepolitique, puis cliquez sur Suivant.
8. Sous Détails du rôle, pour Nom du rôle, saisissez **`lambda-sqs-role`**, puis sélectionnez Créer un rôle.

Après la création du rôle, notez le nom de ressource Amazon (ARN) de votre rôle d'exécution. Vous en aurez besoin dans les étapes suivantes.

## Créer la fonction




Créez une fonction Lambda qui traite vos messages AmazonSQS. Le code de fonction enregistre le corps du SQS message Amazon dans CloudWatch Logs.

Ce didacticiel utilise l'exécution Node.js 18.x, mais nous avons également fourni des exemples de code dans d'autres langages d'exécution. Vous pouvez sélectionner l'onglet dans la zone suivante pour voir le code de l'exécution qui vous intéresse. Le JavaScript code que vous allez utiliser dans cette étape se trouve dans le premier exemple présenté dans l'JavaScriptonglet.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
```

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un SQS événement avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
```

```
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consommation d'un SQS événement avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Consommation d'un SQS événement avec Lambda en utilisant. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
```

```
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
```

```
        .init();

    run(service_fn(function_handler)).await
}
```

Pour créer une fonction Lambda Node.js.

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

2. Copiez l'exemple de JavaScript code dans un nouveau fichier nomméindex.js.
3. Créez un package de déploiement à l'aide de la commande zip suivante.

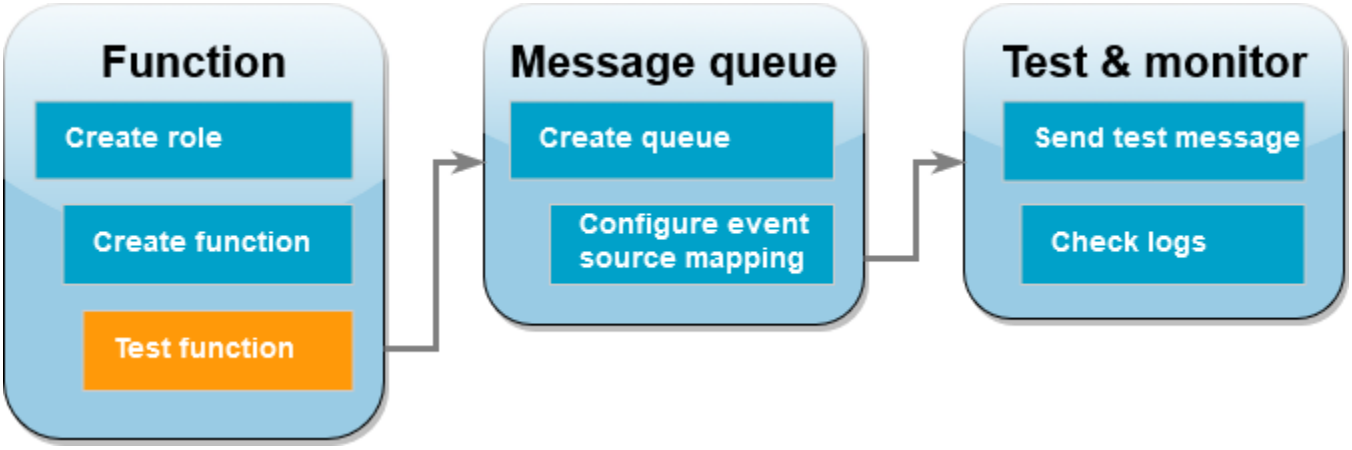
```
zip function.zip index.js
```

4. Créez une fonction Lambda à l'aide de la commande AWS CLI create-function. Pour le role paramètre, entrez le rôle ARN d'exécution que vous avez créé précédemment.

> **Note**
>
> La fonction Lambda et la SQS file d'attente Amazon doivent être identiques. Région AWS

```
aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-sqs-role
```

## Tester la fonction




Appelez votre fonction Lambda manuellement à l'aide de la `invoke` AWS CLI commande et d'un exemple d'événement AmazonSQS.

Pour invoquer la fonction Lambda avec un exemple d’événement

1. Enregistrez le fichier suivant JSON sous forme de fichier nomméinput.json. Cela JSON simule un événement qu'Amazon SQS pourrait envoyer à votre fonction Lambda, `"body"` qui contient le message réel provenant de la file d'attente. Dans cet exemple, le message est `"test"`.

   Example SQSÉvénement Amazon

   Il s’agit d’un événement de test : vous n’avez pas besoin de modifier le message ou le numéro de compte.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
```

```
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:my-queue",
            "awsRegion": "us-east-1"
        }
    ]
}
```

2. Exécutez la AWS CLI commande d'appel suivante. Cette commande renvoie CloudWatch les journaux dans la réponse. Pour de plus amples informations sur la récupération des journaux, veuillez consulter Accès aux journaux à l'aide du AWS CLI.

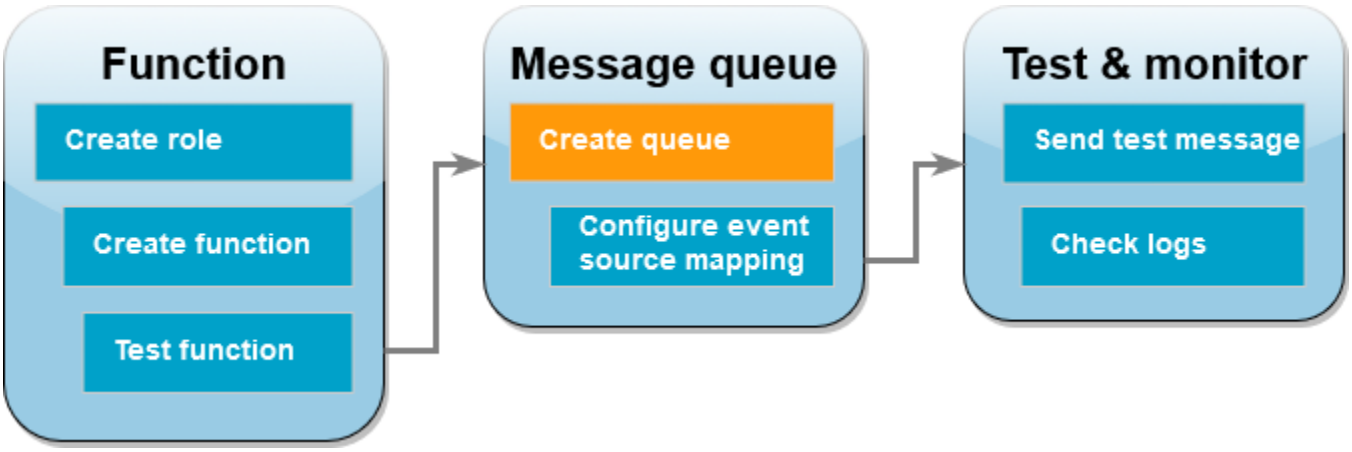
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
 --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

   L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales prises en charge par l'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface version 2.

3. Recherchez le journal `INFO` dans la réponse. C'est ici que la fonction Lambda enregistre le corps du message. Vous devriez voir des journaux qui ressemblent à ceci :

```
2023-09-11T22:45:04.271Z 348529ce-2211-4222-9099-59d07d837b60 INFO Processed
 message test
2023-09-11T22:45:04.288Z 348529ce-2211-4222-9099-59d07d837b60 INFO done
```

## Création d'une SQS file d'attente Amazon



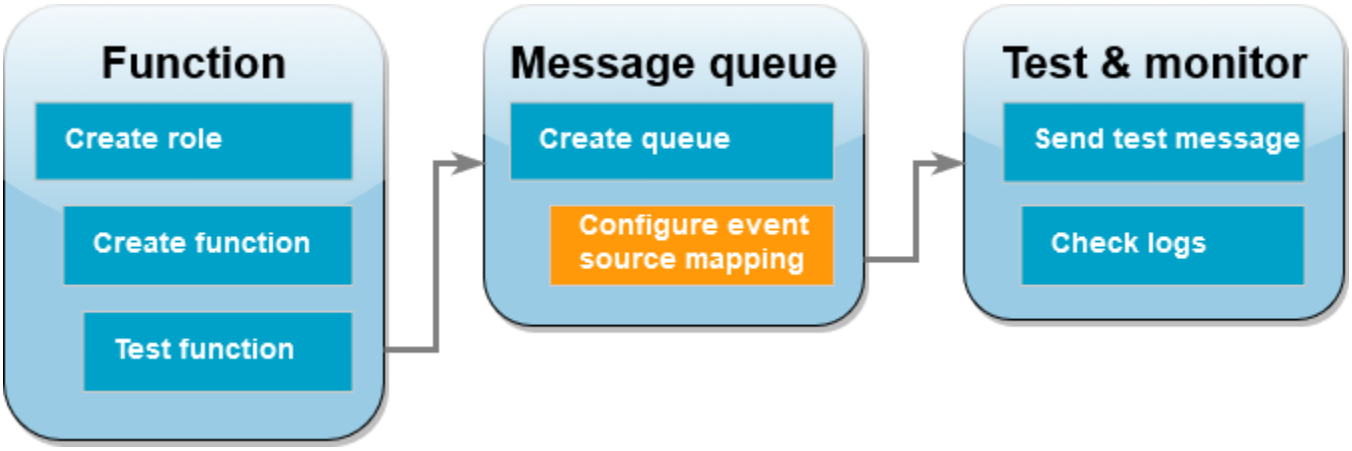

Créez une SQS file d'attente Amazon que la fonction Lambda peut utiliser comme source d'événements. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques. Région AWS

Pour créer une file d’attente

1. Ouvrez la SQSconsole Amazon.
2. Choisissez Créez une file d’attente.
3. Entrez un nom pour la queue. Conservez les paramètres par défaut de toutes les autres options.
4. Choisissez Créez une file d’attente.

Après avoir créé la file d'attente, notez saARN. Vous en aurez besoin à l’étape suivante lorsque vous associerez la file d’attente à votre fonction Lambda.

## Configurer la source de l’événement



Connectez la SQS file d'attente Amazon à votre fonction Lambda en créant un mappage des sources d'événements. Le mappage de la source d'événements lit la SQS file d'attente Amazon et appelle votre fonction Lambda lorsqu'un nouveau message est ajouté.

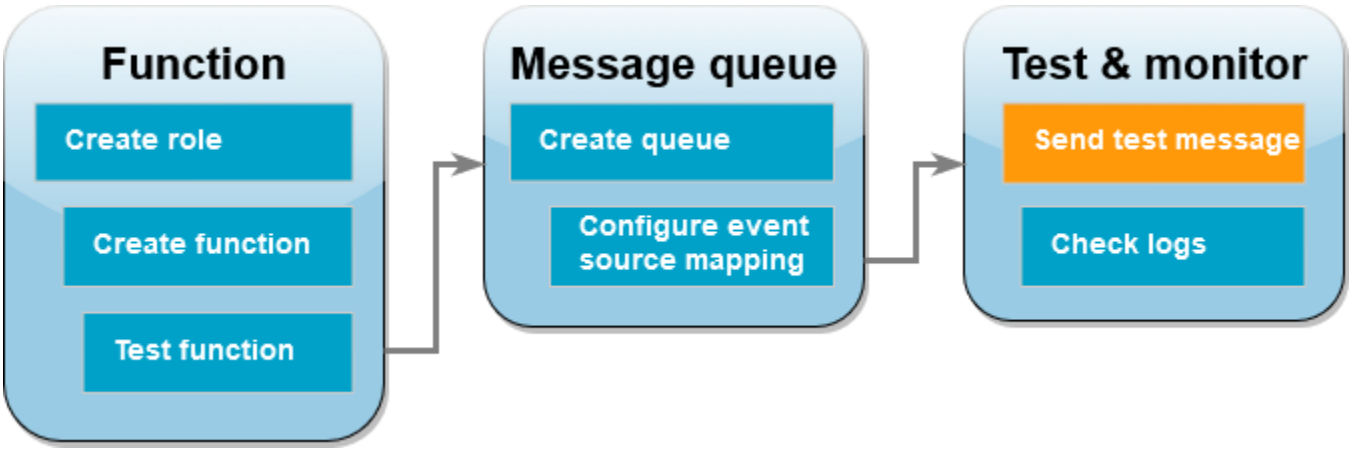
Pour créer un mappage entre votre SQS file d'attente Amazon et votre fonction Lambda, utilisez la create-event-source-mapping AWS CLI commande. Exemple :

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord  --batch-size
 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Pour obtenir la liste de vos mappages de sources d'événements, utilisez la list-event-source-mappingscommande. Exemple :

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

## Envoyer un message de test

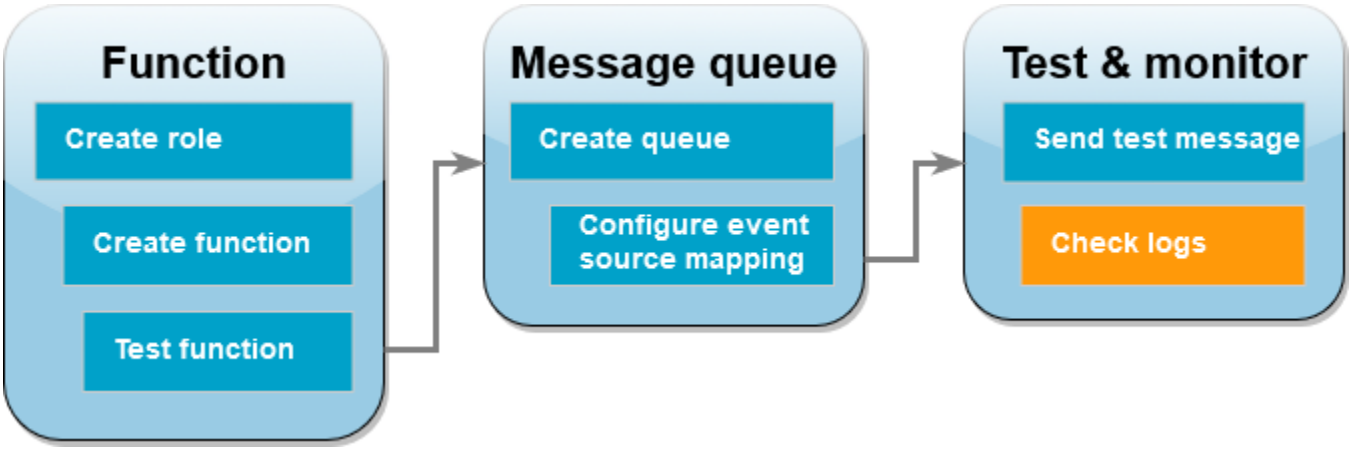



Pour envoyer un SQS message Amazon à la fonction Lambda

1. Ouvrez la SQSconsole Amazon.
2. Choisissez la queue que vous avez créée précédemment.
3. Choisissez Envoyer et recevoir des messages.
4. Sous Corps du message, entrez un message de test, tel que « ceci est un message de test ».
5. Choisissez Send Message (Envoyer un message).

Lambda interroge la file d'attente concernant les mises à jour. Lorsqu'il y a un nouveau message, Lambda invoque votre fonction avec ces nouvelles données d'événement de la file d'attente. Si le gestionnaire de la fonction revient sans exception, Lambda considère le message comme traité avec succès et commence à lire de nouveaux messages dans la file d'attente. Après avoir traité un message avec succès, Lambda le supprime automatiquement de la file d'attente. Si le gestionnaire renvoie une exception, Lambda considère que le traitement du lot de messages a échoué et invoque la fonction avec le même lot de messages.

## Consultez les CloudWatch journaux




Pour confirmer que la fonction a traité le message

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez la rocessSQSRecord fonction P.
3. Sélectionnez Monitor (Surveiller).
4. Choisissez Afficher CloudWatch les journaux.
5. Dans la CloudWatch console, choisissez le flux de log pour la fonction.
6. Recherchez le journal `INFO`. C'est ici que la fonction Lambda enregistre le corps du message. Vous devriez voir le message que vous avez envoyé depuis la SQS file d'attente Amazon. Exemple :

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed
 message this is a test message.
```

### Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la SQS file d'attente Amazon

1. Connectez-vous à la SQS console Amazon AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/sqs/.
2. Sélectionnez la file d'attente que vous avez créée.
3. Choisissez Supprimer.
4. Saisissez **`confirm`** dans le champ de saisie de texte.
5. Sélectionnez Supprimer.

## Tutoriel : Utilisation d'une SQS file d'attente Amazon multi-comptes comme source d'événements

Dans ce didacticiel, vous allez créer une fonction Lambda qui consomme les messages d'une file d'attente Amazon Simple Queue Service SQS (Amazon) d'un autre AWS compte. Ce didacticiel

implique deux AWS comptes : le compte A fait référence au compte qui contient votre fonction Lambda, et le compte B fait référence au compte qui contient la file d'attente AmazonSQS.

## Prérequis

Ce didacticiel suppose que vous avez quelques connaissances des opérations Lambda de base et de la console Lambda. Si ce n'est déjà fait, suivez les instructions fournies dans Créer une fonction Lambda à l'aide de la console pour créer votre première fonction Lambda.

Pour effectuer les étapes suivantes, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2. Les commandes et la sortie attendue sont répertoriées dans des blocs distincts :

```
aws --version
```

Vous devriez voir la sortie suivante:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

Sur Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. CLILes exemples de commandes présentés dans ce guide utilisent le formatage Linux. Les commandes qui incluent JSON des documents intégrés doivent être reformatées si vous utilisez Windows. CLI

## Création du rôle d'exécution (compte A)

Dans le compte A, créez un rôle d'exécution qui autorise votre fonction à accéder aux AWS ressources requises.

Pour créer un rôle d'exécution

1. Ouvrez la page Rôles dans la console AWS Identity and Access Management (IAM).
2. Choisissez Create role (Créer un rôle).
3. Créez un rôle avec les propriétés suivantes :
   - Trusted entity (Entité de confiance) – AWS Lambda.
   - Autorisations — AWSLambdaSQSQueueExecutionRole
   - Nom de rôle – **`cross-account-lambda-sqs-role`**

La AWSLambdaSQSQueueExecutionRolepolitique prévoit les autorisations dont la fonction a besoin pour lire des articles provenant d'Amazon SQS et pour écrire des journaux dans Amazon CloudWatch Logs.

## Créer la fonction (compte A)

Dans le compte A, créez une fonction Lambda qui traite vos messages AmazonSQS. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques. Région AWS

L'exemple de code Node.js 18 suivant écrit chaque message dans CloudWatch un journal de connexion.

Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Pour créer la fonction

> **Note**
>
> Suivre ces étapes permet de créer une fonction dans Node.js 18. Pour les autres langages, les étapes sont similaires mais certains détails sont différents.

1. Enregistrez l'exemple de code en tant que fichier nommé `index.mjs`.
2. Créez un package de déploiement.

```
zip function.zip index.mjs
```

3. Créez la fonction à l'aide de la commande `create-function` AWS Command Line Interface (AWS CLI).

```
aws lambda create-function --function-name CrossAccountSQSExample \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role
```

## Testez la fonction (compte A)

Dans le compte A, testez votre fonction Lambda manuellement à l'aide de la `invoke` AWS CLI commande et d'un exemple d'événement AmazonSQS.

Si le gestionnaire revient normalement sans exception, Lambda considère le message comme traité avec succès et commence à lire de nouveaux messages dans la file d'attente. Après avoir traité un message avec succès, Lambda le supprime automatiquement de la file d'attente. Si le gestionnaire renvoie une exception, Lambda considère que le traitement du lot de messages a échoué et invoque la fonction avec le même lot de messages.

1. Enregistrez le fichier suivant JSON sous forme de fichier nomméinput.txt.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
```

```
            "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:example-queue",
            "awsRegion": "us-east-1"
        }
    ]
}
```

Ce qui précède JSON simule un événement qu'Amazon SQS pourrait envoyer à votre fonction Lambda, `"body"` qui contient le message réel provenant de la file d'attente.

2. Exécutez la commande suivante `invoke` AWS CLI .

```
aws lambda invoke --function-name CrossAccountSQSExample \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

3. Vérifiez la sortie dans le fichier `outputfile.txt`.

## Création d'une SQS file d'attente Amazon (compte B)

Dans le compte B, créez une SQS file d'attente Amazon que la fonction Lambda du compte A peut utiliser comme source d'événements. La fonction Lambda et la SQS file d'attente Amazon doivent être identiques. Région AWS

Pour créer une file d'attente

1. Ouvrez la SQSconsole Amazon.
2. Choisissez Créez une file d'attente.
3. Créez une fille d'attente avec les propriétés suivantes.
   - Type–Standard
   - Nom — LambdaCrossAccountQueue
   - Configuration— Conservez les paramètres par défaut.
   - Stratégie d'accès – Choisissez Avancé. Collez la JSON politique suivante :

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue"
    }
  ]
}
```

Cette politique accorde au rôle d'exécution Lambda dans le compte A l'autorisation de consommer les messages de cette file d'attente AmazonSQS.

4. Après avoir créé la file d'attente, enregistrez son nom de ressource Amazon (ARN). Vous en aurez besoin à l'étape suivante lorsque vous associerez la file d'attente à votre fonction Lambda.

## Configurer la source de l'événement (Compte A)

Dans le compte A, créez un mappage de source d'événement entre la SQS file d'attente Amazon du compte B et votre fonction Lambda en exécutant la commande suivante `create-event-source-mapping` AWS CLI .

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

Pour obtenir une liste de vos mappages de source d'événements, exécutez la commande suivante.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

## Tester la configuration

Maintenant, vous pouvez tester la configuration comme suit :

1. Dans le compte B, ouvrez la SQSconsole Amazon.
2. Choisissez LambdaCrossAccountQueuecelui que vous avez créé précédemment.
3. Choisissez Send and receive messages (Envoyer et recevoir des messages).
4. Sous Message body (Corps du message), saisissez un message test.
5. Choisissez Send Message (Envoyer un message).

Votre fonction Lambda dansCompte Adoit recevoir le message. Lambda continuera d'interroger la file d'attente pour les mises à jour. Lorsqu'il y a un nouveau message, Lambda invoque votre fonction avec ces nouvelles données d'événement de la file d'attente. Votre fonction s'exécute et crée des journaux sur Amazon CloudWatch. Vous pouvez consulter les journaux dans la CloudWatch console.

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

DansCompte A, nettoyez votre rôle d'exécution et votre fonction Lambda.

Pour supprimer le rôle d'exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d'exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Dans le compte B, nettoyez la SQS file d'attente Amazon.

Pour supprimer la SQS file d'attente Amazon

1. Connectez-vous à la SQS console Amazon AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/sqs/.
2. Sélectionnez la file d’attente que vous avez créée.
3. Choisissez Supprimer.
4. Saisissez **`confirm`** dans le champ de saisie de texte.
5. Sélectionnez Supprimer.

# Invoquer une fonction Lambda avec des événements par lots Amazon S3

Vous pouvez utiliser des opérations par lot Amazon S3 pour appeler une fonction Lambda sur un grand ensemble d'objets Amazon S3. Amazon S3 effectue le suivi de la progression des opérations par lot, envoie des notifications et stocke un rapport d'achèvement indiquant le statut de chaque action.

Pour exécuter une opération par lot, vous créez une tâche d'opérations par lot Amazon S3. Lorsque vous créez le travail, vous fournissez un manifeste (la liste des objets) et configurez l'action à effectuer sur ces objets.

Lorsque la tâche par lots démarre, Amazon S3 appelle la fonction Lambda de manière synchrone pour chaque objet figurant dans le manifeste. Le paramètre d'événement inclut les noms du compartiment et de l'objet.

L'exemple suivant montre l'événement qu'Amazon S3 envoie à la fonction Lambda pour un objet nommé customerImage1.jpg dans le compartiment amzn-s3-demo-bucket.

Example Événement de demande de lot Amazon S3

```
{
"invocationSchemaVersion": "1.0",
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "job": {
        "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
    },
    "tasks": [
        {
            "taskId": "dGFza2lkZ29lc2hlcmUK",
            "s3Key": "customerImage1.jpg",
            "s3VersionId": "1",
            "s3BucketArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    ]
}
```

Votre fonction Lambda doit renvoyer un JSON objet avec les champs indiqués dans l'exemple suivant. Vous pouvez copier `invocationId` et `taskId` à partir du paramètre d'événement.

Vous pouvez renvoyer une chaîne dans `resultString`. Amazon S3 enregistre les valeurs `resultString` dans le rapport de fin de tâche.

Example Réponse à la demande de lot Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

## Appel de fonctions Lambda à partir d'opérations par lot Amazon S3

Vous pouvez appeler la fonction Lambda avec une fonction qualifiée ou non qualifiée. ARN Si vous souhaitez utiliser la même version de fonction pour l'ensemble du travail par lot, configurez une version de fonction spécifique dans le paramètre `FunctionARN` lorsque vous créez votre travail. Si vous configurez un alias ou le LATEST qualificatif $, le traitement par lots commence immédiatement à appeler la nouvelle version de la fonction si l'alias ou le $ LATEST est mis à jour pendant l'exécution du travail.

Notez que vous ne pouvez pas réutiliser une fonction Amazon S3 basée sur un événement existante pour des opérations par lot. Cela est dû au fait que l'opération par lots Amazon S3 transmet un paramètre d'événement différent à la fonction Lambda et attend un message de retour avec une structure spécifiqueJSON.

Dans la stratégie basée sur une ressource que vous créez pour la tâche par lot Amazon S3, veillez à définir une autorisation pour que la tâche appelle votre fonction Lambda.

Dans le rôle d'exécution pour la fonction, définissez une stratégie d'approbation pour qu'Amazon S3 endosse ce rôle lors de l'exécution de votre fonction.

Si votre fonction utilise le AWS SDK pour gérer les ressources Amazon S3, vous devez ajouter des autorisations Amazon S3 dans le rôle d'exécution.

Lorsque le travail s'exécute, Amazon S3 démarre plusieurs instances de fonction pour traiter les objets Amazon S3 en parallèle, jusqu'à la limite de simultanéité de la fonction. Amazon S3 limite la montée en puissance initiale des instances afin d'éviter des surcoûts pour les tâches de plus petite taille.

Si la fonction Lambda renvoie un code de réponse `TemporaryFailure`, Amazon S3 réessaie l'opération.

Pour plus d'informations les opérations par lot Amazon S3, consultez Exécution d'opérations par lot dans le Manuel du développeur Amazon S3.

Pour voir un exemple d'utilisation d'une fonction Lambda dans des opérations par lot Amazon S3, consultez Appel d'une fonction Lambda à partir d'opérations par lot Amazon S3 dans le Manuel du développeur Amazon S3.

# Invocation de fonctions Lambda avec les notifications Amazon SNS

Utilisez une fonction Lambda pour traiter les notifications Amazon Simple Notification Service (Amazon SNS). Amazon SNS prend en charge les fonctions Lambda en tant que cibles pour les messages envoyés à une rubrique. Vous pouvez abonner votre fonction à des rubriques du même compte ou d'autres comptes AWS . Pour voir une procédure, consultez the section called “Didacticiel”.

Lambda prend en charge les déclencheurs SNS pour les rubriques SNS standard uniquement. Les sujets FIFO ne sont pas pris en charge.

Pour les appels asynchrones, Lambda place le message en file d'attente et gère les nouvelles tentatives. Si Amazon SNS ne peut pas atteindre Lambda ou si le message est rejeté, Amazon SNS effectue de nouvelles tentatives à intervalles croissants pendant plusieurs heures. Pour plus de détails, consultes Fiabilité dans le FAQ sur Amazon SNS.

> **Warning**
>
> Les mappages de sources d'événements Lambda traitent chaque événement au moins une fois, et le traitement des enregistrements peut être dupliqué. Pour éviter les problèmes potentiels liés à des événements dupliqués, nous vous recommandons vivement de rendre votre code de fonction idempotent. Pour en savoir plus, consultez Comment rendre ma fonction Lambda idempotente dans le Knowledge Center. AWS

Rubriques

- Ajout d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda à l'aide de la console
- Ajout manuel d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda
- Exemple de forme d'événement SNS
- Tutoriel : Utilisation AWS Lambda avec Amazon Simple Notification Service

## Ajout d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda à l'aide de la console

Pour ajouter un sujet SNS comme déclencheur d'une fonction Lambda, le moyen le plus simple consiste à utiliser la console Lambda. Lorsque vous ajoutez le déclencheur via la console, Lambda

définit automatiquement les autorisations et les abonnements nécessaires pour commencer à recevoir des événements provenant de la rubrique SNS.

Pour ajouter une rubrique SNS comme déclencheur pour une fonction Lambda (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom de la fonction pour laquelle vous souhaitez ajouter le déclencheur.
3. Choisissez Configuration, puis Triggers.
4. Choisissez Add trigger (Ajouter déclencheur).
5. Sous Configuration du déclencheur, dans le menu déroulant, choisissez SNS.
6. Pour la rubrique SNS, choisissez la rubrique SNS à laquelle vous abonner.

## Ajout manuel d'un déclencheur de rubrique Amazon SNS pour une fonction Lambda

Pour configurer manuellement un déclencheur SNS pour une fonction Lambda, vous devez suivre les étapes suivantes :

- Définissez une politique basée sur les ressources pour votre fonction afin de permettre au SNS de l'invoquer.
- Abonnez votre fonction Lambda à la rubrique Amazon SNS.

> **Note**
>
> Si votre rubrique SNS et votre fonction Lambda se trouvent sur des comptes AWS différents, vous devez également accorder des autorisations supplémentaires pour autoriser les abonnements entre comptes à la rubrique SNS. Pour plus d'informations, consultez Accorder une autorisation multi-comptes pour l'abonnement Amazon SNS.

Vous pouvez utiliser le AWS Command Line Interface (AWS CLI) pour effectuer ces deux étapes. Tout d'abord, pour définir une politique basée sur les ressources pour une fonction Lambda qui autorise les appels SNS, utilisez la commande suivante. AWS CLI Assurez-vous de remplacer la valeur de par le nom `--function-name` de votre fonction Lambda et la valeur de par l'ARN `--source-arn` de votre rubrique SNS.

```
aws lambda add-permission --function-name example-function \
    --source-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com
```

Pour abonner votre fonction à la rubrique SNS, utilisez la AWS CLI commande suivante. Remplacez la valeur de `--topic-arn` par l'ARN de votre rubrique SNS et la valeur de par l'ARN `--notification-endpoint` de votre fonction Lambda.

```
aws sns subscribe --protocol lambda \
    --region us-east-1 \
    --topic-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --notification-endpoint arn:aws:lambda:us-east-1:123456789012:function:example-
function
```

## Exemple de forme d'événement SNS

Amazon SNS appelle votre fonction de façon asynchrone avec un événement contenant un message et des métadonnées.

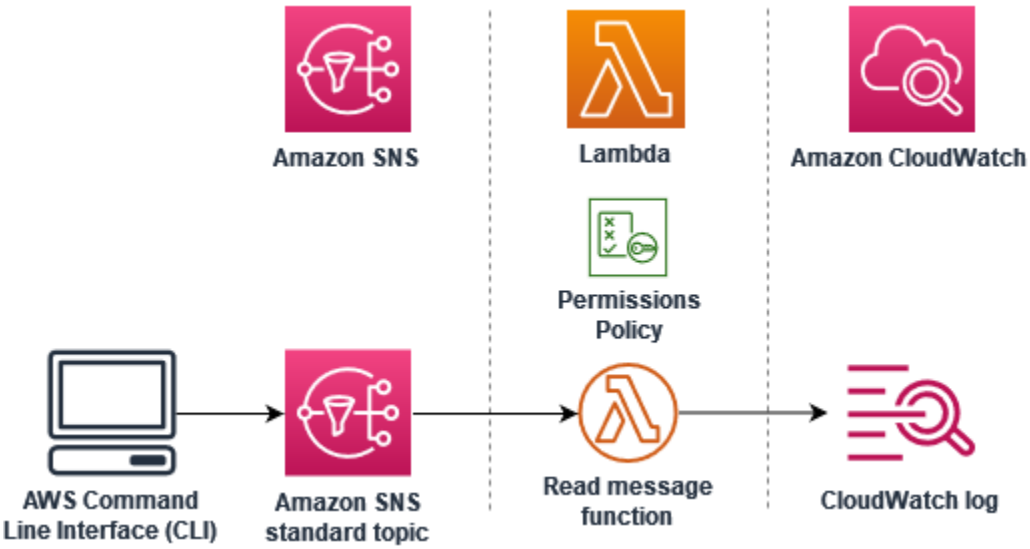
Example Événement de message Amazon SNS

```
{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda:21be56ed-
a058-49f5-8c98-aedd2564c486",
      "EventSource": "aws:sns",
      "Sns": {
        "SignatureVersion": "1",
        "Timestamp": "2019-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",
        "SigningCertURL": "https://sns.us-east-1.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        "MessageAttributes": {
          "Test": {
            "Type": "String",
            "Value": "TestString"
```

```
          },
          "TestBinary": {
            "Type": "Binary",
            "Value": "TestBinary"
          }
        },
        "Type": "Notification",
        "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
        "TopicArn":"arn:aws:sns:us-east-1:123456789012:sns-lambda",
        "Subject": "TestInvoke"
      }
    }
  ]
}
```

## Tutoriel : Utilisation AWS Lambda avec Amazon Simple Notification Service

Dans ce didacticiel, vous utiliserez une fonction Lambda pour vous abonner Compte AWS à une rubrique Amazon Simple Notification Service SNS (Amazon) dans une rubrique séparée. Compte AWS Lorsque vous publiez des messages sur votre SNS sujet Amazon, votre fonction Lambda lit le contenu du message et le transmet à Amazon CloudWatch Logs. Pour terminer ce didacticiel, vous devez utiliser le AWS Command Line Interface (AWS CLI).




Pour compléter ce tutoriel, effectuez les tâches suivantes :

- Dans le compte A, créez un SNS sujet Amazon.

- Dans le compte B, créez une fonction Lambda qui lira les messages de la rubrique.
- Dans le compte B, créez un abonnement à la rubrique.
- Publiez des messages sur le SNS sujet Amazon dans le compte A et vérifiez que la fonction Lambda du compte B les envoie dans Logs. CloudWatch

En suivant ces étapes, vous apprendrez à configurer une SNS rubrique Amazon pour appeler une fonction Lambda. Vous apprendrez également comment créer une politique AWS Identity and Access Management (IAM) qui autorise une ressource d'une autre Compte AWS à invoquer Lambda.

Dans le tutoriel, vous utilisez deux Comptes AWS différents. Les AWS CLI commandes illustrent cela en utilisant deux profils nommés appelés `accountA` et`accountB`, chacun étant configuré pour être utilisé avec un profil différent Compte AWS. Pour savoir comment configurer le AWS CLI pour utiliser différents profils, consultez la section Paramètres des fichiers de configuration et d'identification dans le guide de l'AWS Command Line Interface utilisateur de la version 2. Assurez-vous de configurer la même valeur par défaut Région AWS pour les deux profils.

Si les AWS CLI profils que vous créez pour les deux Comptes AWS utilisent des noms différents, ou si vous utilisez le profil par défaut et un profil nommé, modifiez les AWS CLI commandes dans les étapes suivantes selon vos besoins.

## Prérequis

### Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l‘utilisateur racine a accès à l‘ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l‘utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur root.

   Pour obtenir des instructions, voir Activer un MFA périphérique virtuel pour votre utilisateur Compte AWS root (console) dans le guide de IAM l'utilisateur.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu'utilisateur doté d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URLidentifiant envoyé à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAMIdentity Center, consultez la section Connexion au portail AWS d'accès dans le guide de Connexion à AWS l'utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme à la meilleure pratique consistant à appliquer les autorisations du moindre privilège.

   Pour obtenir des instructions, consultez Création d’un ensemble d’autorisations dans le Guide de l’utilisateur AWS IAM Identity Center .
2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l’utilisateur AWS IAM Identity Center .

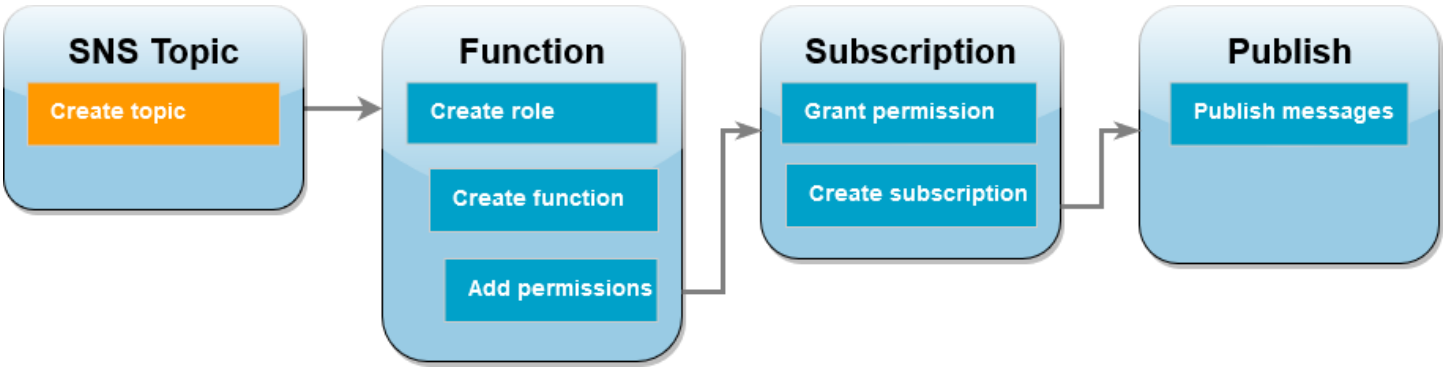
Installez le AWS Command Line Interface

Si vous ne l'avez pas encore installé AWS Command Line Interface, suivez les étapes décrites dans la section Installation ou mise à jour de la dernière version du AWS CLI pour l'installer.

Ce tutoriel nécessite un terminal de ligne de commande ou un shell pour exécuter les commandes. Sous Linux et macOS, utilisez votre gestionnaire de shell et de package préféré.

> **Note**
>
> Sous Windows, certaines CLI commandes Bash que vous utilisez couramment avec Lambda (`zip`telles que) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d’obtenir une version intégrée à Windows d’Ubuntu et Bash.

## Créer un SNS sujet Amazon (compte A)




Pour créer la rubrique

- Dans le compte A, créez un sujet SNS standard Amazon à l'aide de la AWS CLI commande suivante.

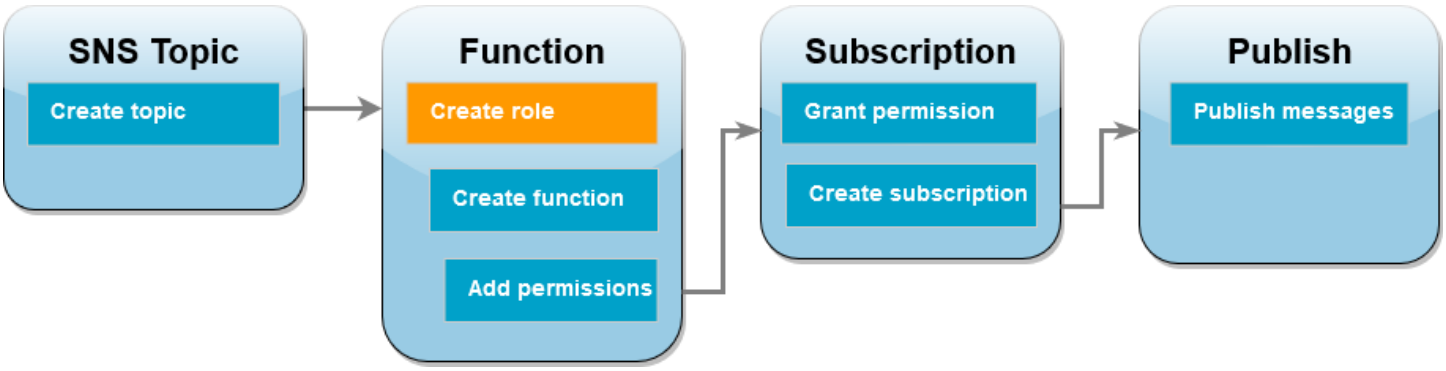
  ```
  aws sns create-topic --name sns-topic-for-lambda --profile accountA
  ```

  Vous devez visualiser des résultats similaires à ce qui suit.

  ```
  {
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
  }
  ```

  Notez le nom de ressource Amazon (ARN) de votre sujet. Vous en aurez besoin plus tard dans le tutoriel lorsque vous ajouterez des autorisations à votre fonction Lambda pour vous abonner à cette rubrique.

## Créer un rôle d'exécution de fonction (compte B)

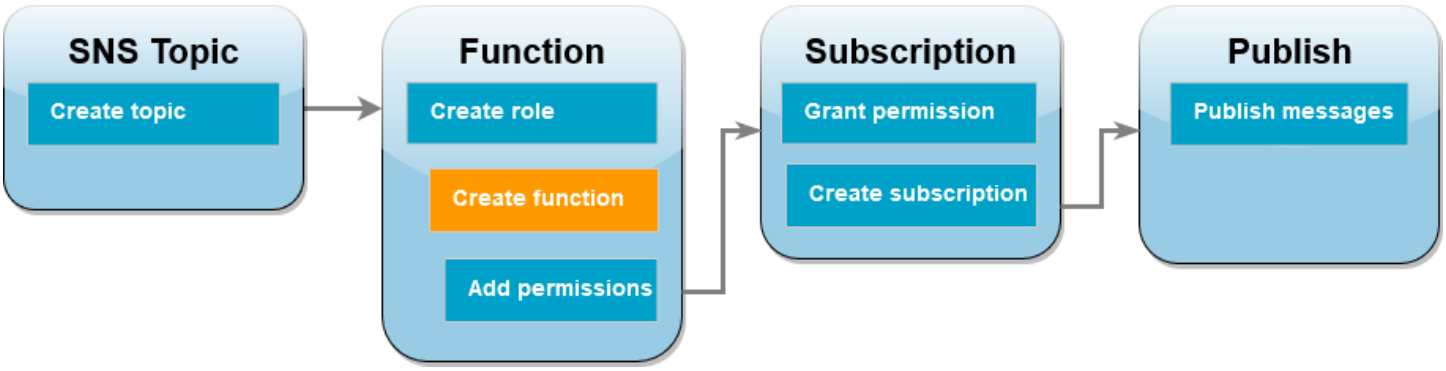


Un rôle d'exécution est un IAM rôle qui accorde à une fonction Lambda l'autorisation d'accéder aux AWS services et aux ressources. Avant de créer votre fonction dans le compte B, vous devez créer un rôle qui donne à la fonction les autorisations de base pour écrire des journaux dans CloudWatch Logs. Nous ajouterons les autorisations nécessaires pour lire un extrait de votre SNS rubrique Amazon ultérieurement.

Pour créer un rôle d'exécution

1. Dans le compte B, ouvrez la page des rôles dans la IAM console.
2. Sélectionnez Créer un rôle.
3. Pour Type d'entité de confiance, choisissez Service AWS .
4. Pour Cas d'utilisation, choisissez Lambda.
5. Choisissez Suivant.
6. Ajoutez une stratégie d'autorisations de base au rôle en procédant comme suit :
   a. Dans le champ de recherche Politiques d'autorisations, saisissez **`AWSLambdaBasicExecutionRole`**.
   b. Choisissez Suivant.
7. Finalisez la création du rôle en procédant comme suit :
   a. Sous Détails du rôle, saisissez **`lambda-sns-role`** pour Nom du rôle.
   b. Sélectionnez Créer un rôle.

## Créer une fonction Lambda (compte B)




Créez une fonction Lambda qui traite vos messages AmazonSNS. Le code de fonction enregistre le contenu des messages de chaque enregistrement dans Amazon CloudWatch Logs.

Ce didacticiel utilise l'exécution Node.js 18.x, mais nous avons également fourni des exemples de code dans d'autres langages d'exécution. Vous pouvez sélectionner l'onglet dans la zone suivante pour voir le code de l'exécution qui vous intéresse. Le JavaScript code que vous allez utiliser dans cette étape se trouve dans le premier exemple présenté dans l'JavaScriptonglet.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }
```

```
    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
 {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

## Go

### SDKpour Go V2

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un SNS événement avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;
```

```
import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consommation d'un SNS événement avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Consommation d'un SNS événement avec Lambda en utilisant. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
```

```
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
 marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
```

```
  raise
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus sur GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features
 = ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
```

```
    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour créer la fonction

1. Créez un répertoire pour le projet, puis passez à ce répertoire.

```
mkdir sns-tutorial
cd sns-tutorial
```

2. Copiez l'exemple de JavaScript code dans un nouveau fichier nomméindex.js.
3. Créez un package de déploiement à l'aide de la commande zip suivante.

```
zip function.zip index.js
```

4. Exécutez la AWS CLI commande suivante pour créer votre fonction Lambda dans le compte B.

```
aws lambda create-function --function-name Function-With-SNS \
    --zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
    --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role  \
    --timeout 60 --profile accountB
```

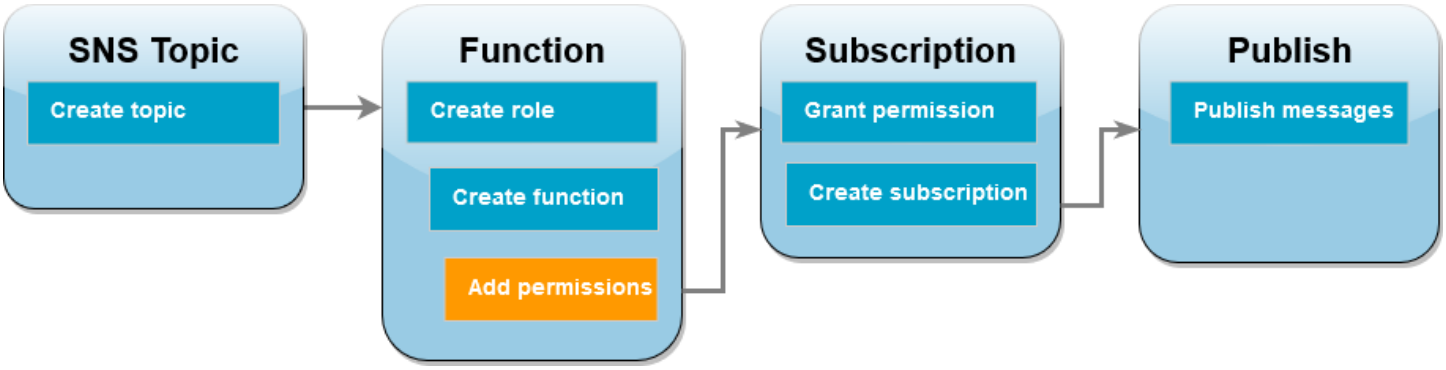
Vous devez visualiser des résultats similaires à ce qui suit.

```
{
    "FunctionName": "Function-With-SNS",
```

```
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:Function-With-
SNS",
    "Runtime": "nodejs18.x",
    "Role": "arn:aws:iam::123456789012:role/lambda_basic_role",
    "Handler": "index.handler",
    ...
    "RuntimeVersionConfig": {
        "RuntimeVersionArn": "arn:aws:lambda:us-
west-2::runtime:7d5f06b69c951da8a48b926ce280a9daf2e8bb1a74fc4a2672580c787d608206"
    }
}
```

5. Enregistrez le nom de ressource Amazon (ARN) de votre fonction. Vous en aurez besoin plus tard dans le didacticiel lorsque vous ajouterez des autorisations permettant SNS à Amazon d'appeler votre fonction.

## Ajouter des autorisations à la fonction (compte B)




Pour SNS qu'Amazon puisse invoquer votre fonction, vous devez lui accorder une autorisation dans une déclaration relative à une politique basée sur les ressources. Vous ajoutez cette instruction à l'aide de la AWS CLI `add-permission` commande.

Pour SNS autoriser Amazon à appeler votre fonction

- Dans le compte B, exécutez la AWS CLI commande suivante à l'aide ARN de la SNS rubrique Amazon que vous avez enregistrée précédemment.

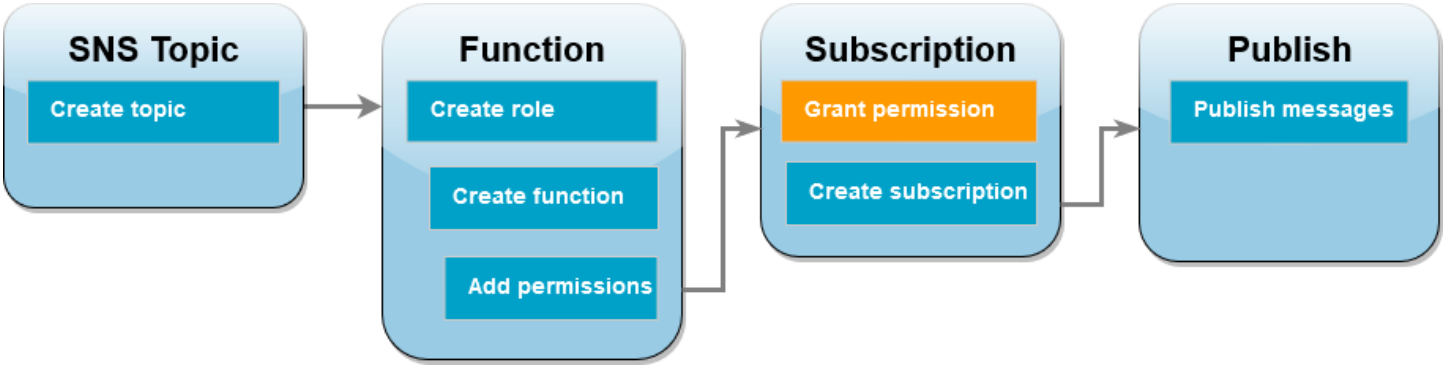
```
aws lambda add-permission --function-name Function-With-SNS \
    --source-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com --profile accountB
```

Vous devez visualiser des résultats similaires à ce qui suit.

```
{
    "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
      \"arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda\"}},
      \"Action\":[\"lambda:InvokeFunction\"],
      \"Resource\":\"arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-
SNS\",
      \"Effect\":\"Allow\",\"Principal\":{\"Service\":\"sns.amazonaws.com\"},
      \"Sid\":\"function-with-sns\"}"
}
```

> **Note**
>
> Si le compte associé au SNS sujet Amazon est hébergé dans le cadre d'un opt-in Région AWS, vous devez spécifier la région dans le principal. Par exemple, si vous travaillez sur un SNS sujet Amazon dans la région Asie-Pacifique (Hong Kong), vous devez le spécifier `sns.ap-east-1.amazonaws.com` plutôt que `sns.amazonaws.com` pour le principal.

## Accorder une autorisation multi-comptes pour l'SNSabonnement Amazon (compte A)



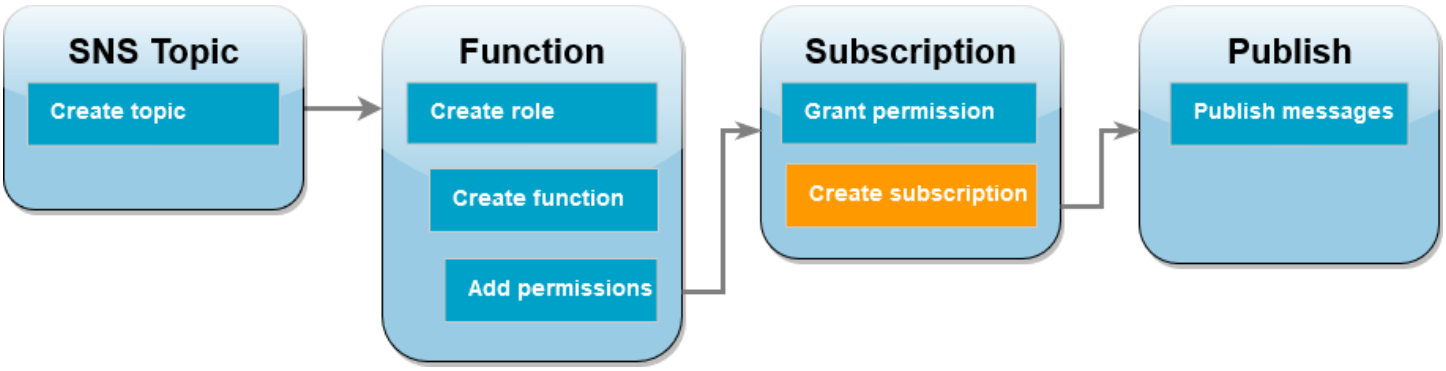

Pour que votre fonction Lambda du compte B puisse s'abonner au SNS sujet Amazon que vous avez créé dans le compte A, vous devez autoriser le compte B à s'abonner à votre sujet. Vous accordez cette autorisation à l'aide de la AWS CLI `add-permission` commande.

Pour accorder l'autorisation permettant au compte B de s'abonner à la rubrique

- Dans le compte A, exécutez la AWS CLI commande suivante. Utilisez le ARN pour le SNS sujet Amazon que vous avez enregistré précédemment.

```
aws sns add-permission --label lambda-access --aws-account-id <AccountB_ID> \
    --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --action-name Subscribe ListSubscriptionsByTopic --profile accountA
```

## Créer un abonnement (compte B)




Dans le compte B, vous abonnez désormais votre fonction Lambda à la SNS rubrique Amazon que vous avez créée au début du didacticiel dans le compte A. Lorsqu'un message est envoyé à cette rubrique (`sns-topic-for-lambda`), Amazon SNS invoque votre `Function-With-SNS` fonction Lambda dans le compte B.

Pour créer un abonnement

- Dans le compte B, exécutez la AWS CLI commande suivante. Utilisez la région par défaut dans laquelle vous avez créé votre sujet, ainsi que la région ARNs correspondant à votre sujet et la fonction Lambda.

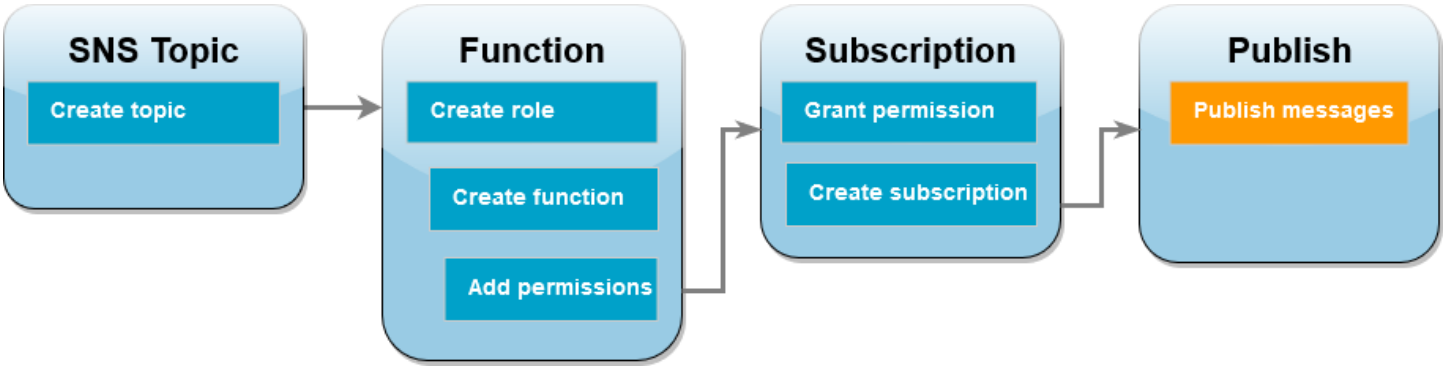
```
aws sns subscribe --protocol lambda \
    --region us-east-1 \
    --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --notification-endpoint arn:aws:lambda:us-
east-1:<AccountB_ID>:function:Function-With-SNS \
    --profile accountB
```

  Vous devez visualiser des résultats similaires à ce qui suit.

```
{
    "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-
lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
```

```
}
```

## Publier des messages sur la rubrique (compte A et compte B)




Maintenant que votre fonction Lambda du compte B est abonnée à votre SNS rubrique Amazon dans le compte A, il est temps de tester votre configuration en publiant des messages sur votre sujet. Pour confirmer qu'Amazon SNS a invoqué votre fonction Lambda, utilisez CloudWatch Logs pour afficher le résultat de votre fonction.

Pour publier un message sur votre rubrique et consulter le résultat de votre fonction

1. Saisissez `Hello World` dans un fichier texte et enregistrez-le sous `message.txt`.
2. À partir du répertoire dans lequel vous avez enregistré votre fichier texte, exécutez la AWS CLI commande suivante dans le compte A. Utilisez-le ARN pour votre propre sujet.

   ```
   aws sns publish --message file://message.txt --subject Test \
       --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
       --profile accountA
   ```

   Cela renverra un identifiant de message avec un identifiant unique, indiquant qu'Amazon SNS a accepté le message. Amazon tente SNS ensuite de transmettre le message aux abonnés du sujet. Pour vérifier qu'Amazon SNS a invoqué votre fonction Lambda, utilisez CloudWatch Logs pour afficher le résultat de votre fonction :
3. Dans le compte B, ouvrez la page Log groups de la CloudWatch console Amazon.
4. Choisissez le groupe de journaux de votre fonction (`/aws/lambda/Function-With-SNS`).
5. Choisissez le flux de journaux le plus récent.
6. Si votre fonction a été correctement invoquée, vous verrez une sortie similaire à la suivante montrant le contenu du message que vous avez publié dans votre rubrique.

```
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO Processed
 message Hello World
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO done
```

## Nettoyage de vos ressources

Vous pouvez maintenant supprimer les ressources que vous avez créées pour ce didacticiel, sauf si vous souhaitez les conserver. En supprimant AWS les ressources que vous n'utilisez plus, vous évitez des frais inutiles pour votre Compte AWS.

Dans le compte A, nettoyez votre SNS rubrique Amazon.

Pour supprimer le SNS sujet Amazon

1. Ouvrez la page Rubriques de la SNS console Amazon.
2. Sélectionnez la rubrique que vous avez créée.
3. Sélectionnez Delete (Supprimer).
4. Saisissez **`delete me`** dans le champ de saisie de texte.
5. Sélectionnez Delete (Supprimer).

Dans le compte B, nettoyez votre rôle d'exécution, votre fonction Lambda et votre abonnement AmazonSNS.

Pour supprimer le rôle d’exécution

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle d’exécution que vous avez créé.
3. Sélectionnez Delete (Supprimer).
4. Saisissez le nom du rôle dans le champ de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.

4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

Pour supprimer l'SNSabonnement Amazon

1. Ouvrez la page Abonnements de la SNS console Amazon.
2. Sélectionnez l'abonnement que vous avez créé.
3. Choisissez Delete (Supprimer), Delete (Supprimer).

# Gestion des autorisations dans AWS Lambda

Vous pouvez utiliser AWS Identity and Access Management (IAM) pour gérer les autorisations dans AWS Lambda. Il existe deux catégories principales d'autorisations que vous devez prendre en compte lorsque vous travaillez avec des fonctions Lambda :

- Autorisations dont vos fonctions Lambda ont besoin pour effectuer des API actions et accéder à d'autres ressources AWS
- Autorisations dont AWS les autres utilisateurs et entités ont besoin pour accéder à vos fonctions Lambda

Les fonctions Lambda ont souvent besoin d'accéder à d'autres AWS ressources et d'effectuer diverses API opérations sur ces ressources. Par exemple, vous pouvez avoir une fonction Lambda qui répond à un événement en mettant à jour les entrées d'une base de données Amazon DynamoDB. Dans ce cas, votre fonction a besoin d'autorisations pour accéder à la base de données, ainsi que d'autorisations pour mettre ou mettre à jour des éléments dans cette base de données.

Vous définissez les autorisations dont votre fonction Lambda a besoin dans un IAM rôle spécial appelé rôle d'exécution. Dans ce rôle, vous pouvez associer une politique qui définit toutes les autorisations dont votre fonction a besoin pour accéder à d'autres AWS ressources et lire des sources d'événements. Chaque fonction Lambda doit avoir un rôle d'exécution. Au minimum, votre rôle d'exécution doit avoir accès à Amazon CloudWatch car les fonctions Lambda se connectent par défaut à CloudWatch Logs. Vous pouvez associer la politique `AWSLambdaBasicExecutionRole` gérée à votre rôle d'exécution pour répondre à cette exigence.

Pour autoriser d'autres Comptes AWS organisations et services à accéder à vos ressources Lambda, plusieurs options s'offrent à vous :

- Vous pouvez utiliser des politiques basées sur l'identité pour autoriser d'autres utilisateurs à accéder à vos ressources Lambda. Les stratégies basées sur l'identité peuvent s'appliquer directement aux utilisateurs, ou aux groupes et aux rôles associés à un utilisateur.
- Vous pouvez utiliser des politiques basées sur les ressources pour autoriser d'autres comptes et AWS services à accéder à vos ressources Lambda. Lorsqu'un utilisateur essaie d'accéder à une ressource Lambda, Lambda prend en compte à la fois les stratégies basées sur l'identité appartenant à l'utilisateur et la stratégie basée sur une ressource appartenant aux ressources. Lorsqu'un AWS service tel qu'Amazon Simple Storage Service (Amazon S3) appelle votre fonction Lambda, Lambda ne prend en compte que la politique basée sur les ressources.

- Vous pouvez utiliser un modèle de contrôle d'accès basé sur les attributs (ABAC) pour contrôler l'accès à vos fonctions Lambda. Vous pouvez ainsi associer des balises à une fonction Lambda, les transmettre dans certaines API demandes ou les associer au IAM principal auteur de la demande. ABAC Spécifiez les mêmes balises dans l'élément condition d'une IAM politique pour contrôler l'accès aux fonctions.

Dans AWS, il est recommandé d'accorder uniquement les autorisations requises pour effectuer une tâche (autorisations du moindre privilège). Pour implémenter cela dans Lambda, nous vous recommandons de commencer par une politique AWS gérée. Vous pouvez utiliser ces stratégies gérées telles quelles ou comme point de départ pour créer des stratégies plus restrictives.

Pour vous aider à affiner vos autorisations d'accès avec le moindre privilège, Lambda fournit des conditions supplémentaires que vous pouvez inclure dans vos politiques. Pour de plus amples informations, veuillez consulter the section called “Ressources et conditions”.

Pour plus d'informationsIAM, consultez le guide de IAM l'utilisateur.

# Définition des autorisations d'une fonction Lambda avec un rôle d'exécution

Le rôle d'exécution d'une fonction Lambda est un rôle AWS Identity and Access Management (IAM) qui accorde à la fonction l'autorisation d'accéder aux AWS services et aux ressources. Par exemple, vous pouvez créer un rôle d'exécution autorisé à envoyer des journaux à Amazon CloudWatch et à y télécharger des données de suivi AWS X-Ray. Cette page fournit des informations sur la façon de créer, d'afficher et de gérer le rôle d'exécution d'une fonction Lambda.

Lambda assume automatiquement votre rôle d'exécution lorsque vous appelez votre fonction. Vous devez éviter d'appeler manuellement `sts:AssumeRole` pour assumer le rôle d'exécution dans votre code de fonction. Si votre cas d'utilisation nécessite que le rôle soit assumé par lui-même, vous devez inclure le rôle lui-même en tant que principal approuvé dans la politique d'approbation de votre rôle. Pour plus d'informations sur la façon de modifier une politique d'approbation des rôles, consultez la section Modification d'une politique d'approbation des rôles (console) dans le Guide de IAM l'utilisateur.

Pour que Lambda assume correctement votre rôle d'exécution, la politique de confiance du rôle doit spécifier le principal du service Lambda (`lambda.amazonaws.com`) en tant que service fiable.

Rubriques

- Création d'un rôle d'exécution dans la IAM console
- Création et gestion des rôles à l'aide du AWS CLI
- Accorder un accès assorti d'un privilège minimum à votre rôle d'exécution Lambda
- Afficher et mettre à jour les autorisations dans le rôle d'exécution
- Utilisation de politiques AWS gérées dans le rôle d'exécution
- Utilisation de l'ARN de la fonction source pour contrôler le comportement d'accès aux fonctions

## Création d'un rôle d'exécution dans la IAM console

Par défaut, Lambda crée un rôle d'exécution avec des autorisations minimales lorsque vous créez une fonction dans la console Lambda. Plus précisément, ce rôle d'exécution inclut la politique `AWSLambdaBasicExecutionRole` gérée, qui donne à votre fonction les autorisations de base pour consigner des événements sur Amazon CloudWatch Logs.

Vos fonctions ont généralement besoin d'autorisations supplémentaires pour effectuer des tâches plus pertinentes. Par exemple, vous pouvez avoir une fonction Lambda qui répond à un événement en mettant à jour les entrées d'une base de données Amazon DynamoDB. Vous pouvez créer un rôle d'exécution doté des autorisations nécessaires à l'aide de la IAM console.

Pour créer un rôle d'exécution dans la IAM console

1. Ouvrez la page Rôles dans la IAM console.
2. Sélectionnez Créer un rôle.
3. Sous Trusted entity type (Type d'entité approuvée), choisissez service AWS .
4. Sous Cas d'utilisation, choisissez Lambda.
5. Choisissez Suivant.
6. Sélectionnez les politiques AWS gérées que vous souhaitez associer à votre rôle. Par exemple, si votre fonction doit accéder à DynamoDB, sélectionnez AWSLambdaDynamoDBExecutionRolela politique gérée.
7. Choisissez Suivant.
8. Entrez un nom dans Role name, puis choisissez Create role.

Pour obtenir des instructions détaillées, consultez la section Création d'un rôle pour un AWS service (console) dans le Guide de IAM l'utilisateur.

Après avoir créé votre rôle d'exécution, associez-le à votre fonction. Lorsque vous créez une fonction dans la console Lambda, vous pouvez associer à la fonction n'importe quel rôle d'exécution que vous avez créé précédemment. Si vous souhaitez associer un nouveau rôle d'exécution à une fonction existante, suivez les étapes décrites dansthe section called “Mettre à jour le rôle d'exécution d'une fonction”.

## Création et gestion des rôles à l'aide du AWS CLI

Pour créer un rôle d'exécution avec le AWS Command Line Interface (AWS CLI), utilisez la create-role commande. Lorsque vous utilisez cette commande, vous pouvez préciser la politique d'approbation en ligne. La politique d'approbation d'un rôle accorde aux principaux spécifiés l'autorisation d'assumer le rôle. Dans l'exemple suivant, vous accordez au service Lambda l'autorisation principale d'assumer votre rôle. Notez que les exigences relatives à l'exclusion des guillemets dans la JSON chaîne peuvent varier en fonction de votre shell.

```
aws iam create-role \
```

```
  --role-name lambda-ex \
  --assume-role-policy-document '{"Version": "2012-10-17","Statement":
 [{ "Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action":
 "sts:AssumeRole"}]}'
```

Vous pouvez également définir la politique de confiance pour le rôle à l'aide d'un JSON fichier distinct. Dans l'exemple suivant, le fichier `trust-policy.json` se trouve dans le répertoire actuel.

Example trust-policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
aws iam create-role \
  --role-name lambda-ex \
  --assume-role-policy-document file://trust-policy.json
```

Vous devriez voir la sortie suivante :

```
{
    "Role": {
        "Path": "/",
        "RoleName": "lambda-ex",
        "RoleId": "AROAQFOXMPL6TZ6ITKWND",
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
        "CreateDate": "2020-01-17T23:19:12Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
```

```
                    "Service": "lambda.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }
}
```

Pour ajouter des autorisations au rôle, utilisez la commande attach-policy-to-role. La commande suivante ajoute la politique `AWSLambdaBasicExecutionRole` gérée au rôle `lambda-ex` d'exécution.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Après avoir créé votre rôle d'exécution, associez-le à votre fonction. Lorsque vous créez une fonction dans la console Lambda, vous pouvez associer à la fonction n'importe quel rôle d'exécution que vous avez créé précédemment. Si vous souhaitez associer un nouveau rôle d'exécution à une fonction existante, suivez les étapes décrites dansthe section called “Mettre à jour le rôle d'exécution d'une fonction”.

## Accorder un accès assorti d'un privilège minimum à votre rôle d'exécution Lambda

Lorsque vous créez un IAM rôle pour votre fonction Lambda pour la première fois pendant la phase de développement, vous pouvez parfois accorder des autorisations au-delà de ce qui est requis. Avant de publier votre fonction dans l'environnement de production, une bonne pratique consiste à ajuster la stratégie de manière à inclure uniquement les autorisations requises. Pour plus d'informations, voir Appliquer les autorisations de moindre privilège dans le Guide de l'IAMutilisateur.

Utilisez IAM Access Analyzer pour identifier les autorisations requises pour la politique de rôle IAM d'exécution. IAMAccess Analyzer examine vos AWS CloudTrail journaux pendant la période que vous spécifiez et génère un modèle de politique avec uniquement les autorisations utilisées par la fonction pendant cette période. Vous pouvez utiliser le modèle pour créer une politique gérée avec des autorisations détaillées, puis l'associer au IAM rôle. Ainsi, vous accordez uniquement les autorisations dont le rôle a besoin pour interagir avec les AWS ressources correspondant à votre cas d'utilisation spécifique.

Pour plus d'informations, voir Générer des politiques en fonction de l'activité d'accès dans le Guide de IAM l'utilisateur.

## Afficher et mettre à jour les autorisations dans le rôle d'exécution

Cette rubrique explique comment afficher et mettre à jour le rôle d'exécution de votre fonction.

Rubriques


- Afficher le rôle d'exécution d'une fonction
- Mettre à jour le rôle d'exécution d'une fonction


### Afficher le rôle d'exécution d'une fonction

Pour afficher le rôle d'exécution d'une fonction, utilisez la console Lambda.

Pour afficher le rôle d'exécution d'une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Choisissez Configuration (Configuration), puis Permissions (Autorisations).
4. Sous Rôle d'exécution, vous pouvez afficher le rôle actuellement utilisé comme rôle d'exécution de la fonction. Pour plus de commodité, vous pouvez consulter toutes les ressources et actions auxquelles la fonction peut accéder dans la section Récapitulatif des ressources. Vous pouvez également choisir un service dans la liste déroulante pour voir toutes les autorisations associées à ce service.

### Mettre à jour le rôle d'exécution d'une fonction

À tout moment, vous pouvez ajouter ou supprimer des autorisations à partir du rôle d'exécution d'une fonction ou configurer votre fonction afin d'utiliser un autre rôle. Si votre fonction a besoin d'accéder à d'autres services ou ressources, vous devez ajouter les autorisations nécessaires au rôle d'exécution.

Lorsque vous ajoutez des autorisations à votre fonction, effectuez également une mise à jour triviale de son code ou de sa configuration. Cela force l'arrêt et le remplacement des instances en cours d'exécution de votre fonction, qui ont des informations d'identification obsolètes.

Pour mettre à jour le rôle d'exécution d'une fonction, vous pouvez utiliser la console Lambda.

Pour mettre à jour le rôle d'exécution d'une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez le nom d'une fonction.
3. Choisissez Configuration (Configuration), puis Permissions (Autorisations).
4. Sous Rôle d'exécution, choisissez Modifier.
5. Si vous souhaitez mettre à jour votre fonction pour utiliser un rôle différent en tant que rôle d'exécution, choisissez le nouveau rôle dans le menu déroulant sous Rôle existant.

   > **Note**
   >
   > Si vous souhaitez mettre à jour les autorisations au sein d'un rôle d'exécution existant, vous ne pouvez le faire que dans la console AWS Identity and Access Management (IAM).

   Si vous souhaitez créer un nouveau rôle à utiliser comme rôle d'exécution, choisissez Créer un nouveau rôle à partir de modèles de AWS politique sous Rôle d'exécution. Entrez ensuite le nom de votre nouveau rôle sous Nom du rôle et spécifiez les politiques que vous souhaitez associer au nouveau rôle sous Modèles de politiques.
6. Choisissez Enregistrer.

## Utilisation de politiques AWS gérées dans le rôle d'exécution

Les politiques AWS gérées suivantes fournissent les autorisations requises pour utiliser les fonctionnalités Lambda.

| Modification | Description | Date |
| --- | --- | --- |
| AWSLambdaMSKExecutionRole— Lambda a ajouté l'autorisation Kafka : DescribeCluster V2 à cette politique. | `AWSLambdaMSKExecutionRole` accorde l'autorisation de lire et d'accéder aux enregistrements d'un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), de gérer les interface | 17 juin 2022 |

| Modification | Description | Date |
| --- | --- | --- |
| | s réseau élastiques (ENI) et d'écrire dans des journaux. CloudWatch | |
| AWSLambdaBasicExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaBasicExecutionRole` accorde des autorisations pour charger les journaux sur CloudWatch. | 14 février 2022 |
| AWSLambdaDynamoDBExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaDynamoDBExecutionRole` accorde l'autorisation de lire les enregistrements d'un flux Amazon DynamoDB et d'écrire dans Logs. CloudWatch | 14 février 2022 |
| AWSLambdaKinesisExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaKinesisExecutionRole` accorde l'autorisation de lire les événements d'un flux de données Amazon Kinesis et d'écrire dans Logs. CloudWatch | 14 février 2022 |
| AWSLambdaMSKExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaMSKExecutionRole` accorde l'autorisation de lire et d'accéder aux enregistrements d'un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), de gérer les interfaces réseau élastiques (ENI) et d'écrire dans des journaux. CloudWatch | 14 février 2022 |

| Modification | Description | Date |
| --- | --- | --- |
| AWSLambdaSQSQueueExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaSQSQueueExecutionRole` accorde l'autorisation de lire un message depuis une file d'attente Amazon Simple Queue Service (Amazon SQS) et d'écrire dans Logs. CloudWatch | 14 février 2022 |
| AWSLambdaVPCAccessExecutionRole— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSLambdaVPCAccessExecutionRole` accorde des autorisations pour gérer les ENI au sein d'un Amazon VPC et écrire CloudWatch dans Logs. | 14 février 2022 |
| AWSXRayDaemonWriteAccess— Lambda a commencé à suivre les modifications apportées à cette politique. | `AWSXRayDaemonWriteAccess` accorde des autorisations pour charger des données de suivi vers X-Ray. | 14 février 2022 |
| CloudWatchLambdaInsightsExecutionRolePolicy— Lambda a commencé à suivre les modifications apportées à cette politique. | `CloudWatchLambdaInsightsExecutionRolePolicy` accorde l'autorisation d'écrire des métriques d'exécution dans CloudWatch Lambda Insights. | 14 février 2022 |

| Modification | Description | Date |
| --- | --- | --- |
| ObjectLambdaExecutionRolePolitique Amazon S3 — Lambda a commencé à suivre les modifications apportées à cette politique. | `AmazonS3ObjectLambdaExecutionRolePolicy` accorde les autorisations nécessaires pour interagir avec l'objet Lambda d'Amazon Simple Storage Service (Amazon S3) et pour écrire dans Logs. CloudWatch | 14 février 2022 |

Pour certaines fonctionnalités, la console Lambda tente d'ajouter les autorisations manquantes au rôle d'exécution dans une stratégie gérée par le client. Ces stratégies peuvent être excessivement nombreuses. Afin d'éviter la création de stratégies supplémentaires, ajoutez les stratégies gérées AWS pertinentes au rôle d'exécution avant d'activer les fonctions.

Lorsque vous utilisez un mappage de source d'événement pour appeler votre fonction, Lambda utilise le rôle d'exécution pour lire les données d'événement. Par exemple, un mappage de source d'événement pour Kinesis lit les événements d'un flux de données et les envoie à votre fonction par lots.

Lorsqu'un service endosse un rôle dans votre compte, vous pouvez inclure les clés de contexte de condition globale `aws:SourceAccount` et `aws:SourceArn` dans votre politique d'approbation des rôles afin de limiter l'accès au rôle aux seules demandes générées par les ressources attendues. Pour plus d'informations, consultez Prévention du problème de l'adjoint confus entre services pour AWS Security Token Service.

Outre les politiques AWS gérées, la console Lambda fournit des modèles permettant de créer une politique personnalisée avec des autorisations pour des cas d'utilisation supplémentaires. Lorsque vous créez une fonction dans la console Lambda, vous pouvez choisir de créer un nouveau rôle d'exécution doté des autorisations d'un ou plusieurs modèles. Ces modèles s'appliquent également automatiquement lorsque vous créez une fonction à partir d'un plan ou lorsque vous configurez les options qui nécessitent l'accès à d'autres services. Des exemples de modèles sont disponibles dans le GitHubréférentiel de ce guide.

# Utilisation de l'ARN de la fonction source pour contrôler le comportement d'accès aux fonctions

Il est courant que le code de votre fonction Lambda envoie des requêtes d'API à d'autres AWS services. Pour effectuer ces requêtes, Lambda génère un ensemble éphémère d'informations d'identification en assumant le rôle d'exécution de votre fonction. Ces informations d'identification sont disponibles en tant que variables d'environnement lors de l'appel de votre fonction. Lorsque vous travaillez avec des kits SDK AWS , vous n'avez pas besoin de fournir des informations d'identification pour le kit SDK directement dans le code. Par défaut, la chaîne de fournisseurs d'informations d'identification vérifie séquentiellement chaque endroit où vous pouvez définir des informations d'identification et sélectionne le premier disponible, généralement les variables d'environnement (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` et `AWS_SESSION_TOKEN`).

Lambda injecte l'ARN de la fonction source dans le contexte des informations d'identification si la demande est une demande d' AWS API provenant de votre environnement d'exécution. Lambda injecte également l'ARN de la fonction source pour les demandes d'API AWS suivantes que Lambda effectue en votre nom en dehors de votre environnement d'exécution :

| Service | Action | Raison |
| --- | --- | --- |
| CloudWatch Journaux | `CreateLogGroup` , `CreateLogStream` , `PutLogEvents` | Pour stocker les journaux dans un groupe de CloudWatch journaux |
| X-Ray | `PutTraceSegments` | Pour envoyer des données de traçage à X-Ray |
| Amazon EFS | `ClientMount` | Pour connecter votre fonction à un système de fichiers Amazon Elastic File System (Amazon EFS) |

Les autres appels d' AWS API que Lambda effectue en dehors de votre environnement d'exécution en votre nom en utilisant le même rôle d'exécution ne contiennent pas l'ARN de la fonction source. Voici des exemples de tels appels d'API en dehors de l'environnement d'exécution :

- Appelle to AWS Key Management Service (AWS KMS) pour chiffrer et déchiffrer automatiquement vos variables d'environnement.
- Les appels à Amazon Elastic Compute Cloud (Amazon EC2) pour créer des interfaces réseau Elastic (ENI) pour une fonction compatible VPC.
- Appels à AWS des services, tels qu'Amazon Simple Queue Service (Amazon SQS), pour lire à partir d'une source d'événements configurée comme mappage de sources d'événements.

Avec l'ARN de la fonction source dans le contexte des informations d'identification, vous pouvez vérifier si un appel à votre ressource provient du code d'une fonction Lambda spécifique. Pour vérifier cela, utilisez la clé de `lambda:SourceFunctionArn` condition dans une politique basée sur l'identité IAM ou une politique de contrôle des services (SCP).

> **Note**
>
> Vous ne pouvez pas utiliser l'élément `lambda:SourceFunctionArn` dans les politiques basées sur les ressources.

Avec cette clé de condition dans vos politiques basées sur l'identité ou SCP, vous pouvez implémenter des contrôles de sécurité pour les actions d'API que votre code de fonction effectue sur d'autres services. AWS Cela comporte quelques applications de sécurité clés, telles que l'identification de la source d'une fuite d'informations d'identification.

> **Note**
>
> La clé de condition `lambda:SourceFunctionArn` est différente des clés de condition `lambda:FunctionArn` et `aws:SourceArn`. Le clé de condition `lambda:FunctionArn` s'applique uniquement aux Mappages de sources d'événement et aide à définir les fonctions que votre source d'événement peut invoquer. La clé de `aws:SourceArn` condition s'applique uniquement aux politiques dans lesquelles votre fonction Lambda est la ressource cible et permet de définir quels autres AWS services et ressources peuvent invoquer cette fonction. La clé de `lambda:SourceFunctionArn` condition peut s'appliquer à n'importe quelle politique basée sur l'identité ou à tout SCP afin de définir les fonctions Lambda spécifiques autorisées à effectuer des appels d' AWS API spécifiques vers d'autres ressources.

Pour utiliser `lambda:SourceFunctionArn` dans votre stratégie, incluez-la en tant que condition dans l'un des Opérateurs de condition ARN. La valeur de la clé doit être un ARN valide.

Par exemple, supposons que votre code de fonction Lambda émette un appel `s3:PutObject` qui cible un compartiment Amazon S3 spécifique. Vous pouvez vouloir n'autoriser qu'une fonction Lambda spécifique à avoir un accès `s3:PutObject` à ce compartiment. Dans ce cas, le rôle d'exécution de votre fonction doit être associé à une stratégie qui ressemble à ceci :

Example stratégie accordant à une fonction Lambda spécifique l'accès à une ressource Amazon S3

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleSourceFunctionArn",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Condition": {
                "ArnEquals": {
                    "lambda:SourceFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:source_lambda"
                }
            }
        }
    ]
}
```

Cette politique permet uniquement un accès `s3:PutObject` si la source est la fonction Lambda avec ARN `arn:aws:lambda:us-east-1:123456789012:function:source_lambda`. Cette politique n'autorise pas un accès `s3:PutObject` à une autre identité d'appel. C'est le cas même si une fonction ou une entité différente émet un appel `s3:PutObject` avec le même rôle d'exécution.

> **Note**
>
> La clé de condition `lambda:SourceFunctionARN` ne prend pas en charge les versions des fonctions Lambda ni les alias de fonction. Si vous utilisez l'ARN pour une version ou un alias de fonction en particulier, votre fonction n'est pas autorisée à effectuer l'action que vous spécifiez. Veillez à utiliser l'ARN non qualifié pour votre fonction sans suffixe de version ou d'alias.

Vous pouvez également l'utiliser `lambda:SourceFunctionArn` dans les SCP. Supposons, par exemple, que vous souhaitiez limiter l'accès à votre compartiment à un code de fonction Lambda unique ou à des appels issus d'un cloud privé virtuel (VPC) Amazon. Le SCP suivant illustre ce processus.

Example politique refusant l'accès à Amazon S3 dans des conditions spécifiques

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Effect": "Deny",
            "Condition": {
                "StringNotEqualsIfExists": {
                    "aws:SourceVpc": [
                        "vpc-12345678"
                    ]
                }
            }
        },
        {
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Effect": "Deny",
            "Condition": {
                "ArnNotEqualsIfExists": {
                    "lambda:SourceFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:source_lambda"
                }
            }
        }
    ]
}
```

Cette stratégie refuse toutes les actions S3 sauf si elles proviennent d'une fonction Lambda spécifique avec ARN `arn:aws:lambda:*:123456789012:function:source_lambda`,

ou à moins qu'ils ne proviennent du VPC spécifié. L'opérateur `StringNotEqualsIfExists` indique à IAM de traiter cette condition uniquement si la clé `aws:SourceVpc` est présente dans la demande. De la même façon, IAM considère l'opérateur `ArnNotEqualsIfExists` uniquement si le `lambda:SourceFunctionArn` existe.

# Accorder à d'autres AWS entités l'accès à vos fonctions Lambda

Pour autoriser d'autres Comptes AWS organisations et services à accéder à vos ressources Lambda, plusieurs options s'offrent à vous :

- Vous pouvez utiliser des politiques basées sur l'identité pour autoriser d'autres utilisateurs à accéder à vos ressources Lambda. Les stratégies basées sur l'identité peuvent s'appliquer directement aux utilisateurs, ou aux groupes et aux rôles associés à un utilisateur.
- Vous pouvez utiliser des politiques basées sur les ressources pour autoriser d'autres comptes et AWS services à accéder à vos ressources Lambda. Lorsqu'un utilisateur essaie d'accéder à une ressource Lambda, Lambda prend en compte à la fois les stratégies basées sur l'identité appartenant à l'utilisateur et la stratégie basée sur une ressource appartenant aux ressources. Lorsqu'un AWS service tel qu'Amazon Simple Storage Service (Amazon S3) appelle votre fonction Lambda, Lambda ne prend en compte que la politique basée sur les ressources.
- Vous pouvez utiliser un modèle de contrôle d'accès basé sur les attributs (ABAC) pour contrôler l'accès à vos fonctions Lambda. Vous pouvez ainsi associer des balises à une fonction Lambda, les transmettre dans certaines API demandes ou les associer au IAM principal auteur de la demande. ABAC Spécifiez les mêmes balises dans l'élément condition d'une IAM politique pour contrôler l'accès aux fonctions.

Pour vous aider à affiner vos autorisations d'accès avec le moindre privilège, Lambda fournit des conditions supplémentaires que vous pouvez inclure dans vos politiques. Pour de plus amples informations, veuillez consulter the section called “Ressources et conditions”.

## IAMPolitiques basées sur l'identité pour Lambda

Vous pouvez utiliser des politiques basées sur l'identité dans AWS Identity and Access Management (IAM) pour autoriser les utilisateurs de votre compte à accéder à Lambda. Les politiques basées sur l'identité peuvent s'appliquer aux utilisateurs, aux groupes d'utilisateurs ou aux rôles. Vous pouvez également accorder aux utilisateurs d'un autre compte l'autorisation d'endosser un rôle de votre compte et d'accéder à vos ressources Lambda.

Lambda fournit des politiques AWS gérées qui accordent l'accès aux API actions Lambda et, dans certains cas, à d'autres AWS services utilisés pour développer et gérer les ressources Lambda. Lambda met à jour ces stratégies gérées si nécessaire pour s'assurer que vos utilisateurs ont accès aux nouvelles fonctionnalités quand elles sont disponibles.

- AWSLambda_ FullAccess — Accorde un accès complet aux actions Lambda et aux autres AWS services utilisés pour développer et gérer les ressources Lambda.
- AWSLambda_ ReadOnlyAccess — Accorde un accès en lecture seule aux ressources Lambda.
- AWSLambdaRole— Accorde l'autorisation d'invoquer des fonctions Lambda.

AWS les politiques gérées autorisent les API actions sans restreindre les fonctions ou les couches Lambda qu'un utilisateur peut modifier. Pour bénéficier d'un contrôle plus précis, vous pouvez créer vos propres stratégies qui limitent la portée des autorisations d'un utilisateur.

Rubriques

- Accorder aux utilisateurs l'accès à une fonction Lambda
- Accorder aux utilisateurs l'accès à une couche Lambda

## Accorder aux utilisateurs l'accès à une fonction Lambda

Utilisez des politiques basées sur l'identité pour permettre aux utilisateurs, aux groupes d'utilisateurs ou aux rôles d'effectuer des opérations sur les fonctions Lambda.

> **Note**
>
> Pour une fonction définie comme une image de conteneur, l'autorisation utilisateur d'accéder à l'image doit être configurée dans Amazon Elastic Container Registry (AmazonECR). Pour un exemple, consultez les politiques relatives aux ECR référentiels Amazon.

Voici un exemple de stratégie d'autorisations avec une portée limitée. Elle permet à un utilisateur de créer et gérer des fonctions Lambda nommées avec un préfixe distinct (`intern-`) et configurées avec un rôle d'exécution désigné.

Example Stratégie de développement des fonctions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyPermissions",
            "Effect": "Allow",
            "Action": [
```

```
                "lambda:GetAccountSettings",
                "lambda:GetEventSourceMapping",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunctionCodeSigningConfig",
                "lambda:GetFunctionConcurrency",
                "lambda:ListEventSourceMappings",
                "lambda:ListFunctions",
                "lambda:ListTags",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DevelopFunctions",
            "Effect": "Allow",
            "NotAction": [
                "lambda:AddPermission",
                "lambda:PutFunctionConcurrency"
            ],
            "Resource": "arn:aws:lambda:*:*:function:intern-*"
        },
        {
            "Sid": "DevelopEventSourceMappings",
            "Effect": "Allow",
            "Action": [
                "lambda:DeleteEventSourceMapping",
                "lambda:UpdateEventSourceMapping",
                "lambda:CreateEventSourceMapping"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
                }
            }
        },
        {
            "Sid": "PassExecutionRole",
            "Effect": "Allow",
            "Action": [
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:GetRole",
```

```
                "iam:GetRolePolicy",
                "iam:PassRole",
                "iam:SimulatePrincipalPolicy"
            ],
            "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
        },
        {
            "Sid": "ViewLogs",
            "Effect": "Allow",
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
        }
    ]
}
```

Les autorisations de la stratégie sont organisées en déclarations reposant sur les ressources et conditions qu'elles prennent en charge.

- `ReadOnlyPermissions` – La console Lambda utilise ces autorisations lorsque vous parcourez et affichez les fonctions. Les conditions ou modèles de ressources ne sont pas pris en charge.

```
            "Action": [
                "lambda:GetAccountSettings",
                "lambda:GetEventSourceMapping",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunctionCodeSigningConfig",
                "lambda:GetFunctionConcurrency",
                "lambda:ListEventSourceMappings",
                "lambda:ListFunctions",
                "lambda:ListTags",
                "iam:ListRoles"
            ],
            "Resource": "*"
```

- `DevelopFunctions`— Utilisez n'importe quelle action Lambda qui agit sur des fonctions préfixées par`intern-`, sauf et. `AddPermission` `PutFunctionConcurrency` `AddPermission`modifie la politique basée sur les ressources appliquée à la fonction et peut avoir des implications en termes

de sécurité. `PutFunctionConcurrency`réserve la capacité de mise à l'échelle d'une fonction et peut réduire la capacité d'autres fonctions.

```
"NotAction": [
    "lambda:AddPermission",
    "lambda:PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*:*:function:intern-*"
```

- `DevelopEventSourceMappings` – Gérez les mappages de source d'événement sur les fonctions qui sont préfixées avec `intern-`. Ces actions s'exécutent sur des mappages de source d'événement, mais vous pouvez les restreindre par fonction à l'aide d'une condition.

```
"Action": [
    "lambda:DeleteEventSourceMapping",
    "lambda:UpdateEventSourceMapping",
    "lambda:CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
    "StringLike": {
        "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
    }
}
```

- `PassExecutionRole`— Affichez et transmettez uniquement un rôle nomméintern-lambda-execution-role, qui doit être créé et géré par un utilisateur IAM autorisé. `PassRole`est utilisé lorsque vous attribuez un rôle d'exécution à une fonction.

```
"Action": [
    "iam:ListRolePolicies",
    "iam:ListAttachedRolePolicies",
    "iam:GetRole",
    "iam:GetRolePolicy",
    "iam:PassRole",
    "iam:SimulatePrincipalPolicy"
],
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- `ViewLogs`— Utilisez CloudWatch les journaux pour afficher les journaux des fonctions préfixées par`intern-`.

```
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Cette stratégie permet à un utilisateur de démarrer avec Lambda, sans nuire aux ressources des autres utilisateurs. Il ne permet pas à un utilisateur de configurer une fonction pour qu'elle soit déclenchée par d'autres services ou d'appeler d'autres AWS services, ce qui nécessite IAM des autorisations plus étendues. Cela n'inclut pas non plus l'autorisation d'accéder à des services qui ne sont pas compatibles avec les politiques à portée limitée, tels que CloudWatch X-Ray. Pour ces services, utilisez les stratégies en lecture seule afin d'accorder à l'utilisateur l'accès aux métriques et aux données de suivi.

Lorsque vous configurez des déclencheurs pour votre fonction, vous devez y accéder pour utiliser le AWS service qui appelle votre fonction. Par exemple, pour configurer un déclencheur Amazon S3, vous vous devez être autorisé à utiliser les actions Amazon S3 qui gèrent les notifications de compartiment. Un grand nombre de ces autorisations sont incluses dans la politique FullAccess gérée AWSLambda_.

## Accorder aux utilisateurs l'accès à une couche Lambda

Utilisez des politiques basées sur l'identité pour permettre aux utilisateurs, aux groupes d'utilisateurs ou aux rôles d'effectuer des opérations sur les couches Lambda. La stratégie suivante accorde à un utilisateur l'autorisation de créer des couches et de les utiliser avec des fonctions. Les modèles de ressources permettent à l'utilisateur de travailler dans n' Région AWS importe quelle version de couche, à condition que le nom de la couche commence par`test-`.

Example stratégie de développement des couches

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublishLayers",
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:PublishLayerVersion"
            ],
            "Resource": "arn:aws:lambda:*:*:layer:test-*"
        },
        {
            "Sid": "ManageLayerVersions",
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": "arn:aws:lambda:*:*:layer:test-*:*"
        }
    ]
}
```

Vous pouvez également appliquer l'utilisation des couches durant la création et la configuration de la fonction avec la condition `lambda:Layer`. Par exemple, vous pouvez empêcher les utilisateurs d'utiliser les couches publiées par d'autres comptes. La stratégie ci-dessous ajoute une condition aux actions `CreateFunction` et `UpdateFunctionConfiguration` afin d'exiger que toutes les couches spécifiées proviennent du compte `123456789012`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureFunctions",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "lambda:Layer": [
                        "arn:aws:lambda:*:123456789012:layer:*:*"
                    ]
                }
            }
        }
    ]
```

```
}
```

Pour que la condition s'applique, vérifiez qu'aucune autre déclaration n'accorde d'autorisation utilisateur à ces actions.

## Utilisation de IAM politiques basées sur les ressources dans Lambda

Lambda prend en charge les stratégies d'autorisations basées sur une ressource pour les fonctions et les couches Lambda. Vous pouvez utiliser des politiques basées sur les ressources pour accorder l'accès à d'autres AWS comptes, organisations ou services. Les stratégies basées sur les ressources s'appliquent à une seule fonction, version ou version de couche ou à un seul alias.

Console

Pour afficher la stratégie basée sur les ressources d'une fonction

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez Configuration (Configuration), puis Permissions (Autorisations).
4. Faites défiler la page vers le bas, jusqu'à Resource-based policy (Stratégie basée sur les ressources), puis choisissez View policy document (Afficher le document de stratégie). La politique basée sur les ressources indique les autorisations appliquées lorsqu'un autre compte ou AWS service tente d'accéder à la fonction. L'exemple suivant montre une déclaration qui autorise Amazon S3 à invoquer une fonction nommée `my-function` pour un compartiment nommé `amzn-s3-demo-bucket` dans le compte `123456789012`.

   Example Stratégie basée sur une ressource

   ```
   {
       "Version": "2012-10-17",
       "Id": "default",
       "Statement": [
           {
               "Sid": "lambda-allow-s3-my-function",
               "Effect": "Allow",
               "Principal": {
                 "Service": "s3.amazonaws.com"
               },
               "Action": "lambda:InvokeFunction",
   ```

```
            "Resource":  "arn:aws:lambda:us-east-2:123456789012:function:my-
function",
            "Condition": {
              "StringEquals": {
                "AWS:SourceAccount": "123456789012"
              },
              "ArnLike": {
                "AWS:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
              }
            }
        }
    ]
}
```

AWS CLI

Pour afficher une stratégie basée sur les ressources d'une fonction, utilisez la commande `get-policy`.

```
aws lambda get-policy \
  --function-name my-function \
  --output text
```

Vous devriez voir la sortie suivante:

```
{"Version":"2012-10-17","Id":"default","Statement":
[{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"s3.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:u
east-2:123456789012:function:my-function","Condition":{"ArnLike":
{"AWS:SourceArn":"arn:aws:sns:us-east-2:123456789012:lambda*"}}}]}     7c681fc9-
b791-4e91-acdf-eb847fdaa0f0
```

Pour les versions et les alias, ajoutez le numéro de version ou l'alias au nom de la fonction.

```
aws lambda get-policy --function-name my-function:PROD
```

Pour supprimer les autorisations de votre fonction, utilisez `remove-permission`.

```
aws lambda remove-permission \
  --function-name example \
```

```
  --statement-id sns
```

Utilisez la commande `get-layer-version-policy` pour afficher les autorisations sur une couche.

```
aws lambda get-layer-version-policy \
  --layer-name my-layer \
  --version-number 3 \
  --output text
```

Vous devriez voir la sortie suivante:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236    {"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:
west-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

Utilisez `remove-layer-version-permission` pour supprimer des déclarations de la stratégie.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3
 --statement-id engineering-org
```

## APIActions soutenues

Les API actions Lambda suivantes prennent en charge les politiques basées sur les ressources :

- CreateAlias
- DeleteAlias
- DeleteFunction
- DeleteFunctionConcurrency
- DeleteFunctionEventInvokeConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias
- GetFunction
- GetFunctionConcurrency

- GetFunctionConfiguration
- GetFunctionEventInvokeConfig
- GetPolicy
- GetProvisionedConcurrencyConfig
- Invoke
- ListAliases
- ListFunctionEventInvokeConfigs
- ListProvisionedConcurrencyConfigs
- ListTags
- ListVersionsByFunction
- PublishVersion
- PutFunctionConcurrency
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- TagResource
- UntagResource
- UpdateAlias
- UpdateFunctionCode
- UpdateFunctionEventInvokeConfig

## Octroi de l'accès aux services à la fonction Lambda AWS

Lorsque vous utilisez un AWS service pour appeler votre fonction, vous accordez l'autorisation dans une déclaration relative à une politique basée sur les ressources. Vous pouvez appliquer l'instruction à l'ensemble de la fonction ou la limiter à une seule version ou à un seul alias.

> **Note**
>
> Lorsque vous ajoutez un déclencheur à la fonction à l'aide de la console Lambda, celle-ci met à jour la stratégie basée sur une ressource de la fonction afin d'autoriser le service à l'invoquer. Pour accorder des autorisations à d'autres comptes ou services non disponibles dans la console Lambda, vous pouvez utiliser l' AWS CLI.

Ajoutez une instruction à l'aide de la commande add permission. La déclaration de stratégie basée sur les ressources la plus simple autorise un service à invoquer une fonction. La commande suivante autorise Amazon Simple Notification Service à appeler une fonction nomméemy-function.

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns \
  --principal sns.amazonaws.com \
  --output text
```

Vous devriez voir la sortie suivante :

```
{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function"}
```

Cela permet à Amazon API d'SNSappeler l'action Invoke sur la fonction, mais cela ne limite pas le SNS sujet Amazon qui déclenche l'invocation. Pour vous assurer que votre fonction n'est invoquée que par une ressource spécifique, spécifiez le nom de ressource Amazon (ARN) de la ressource avec l'source-arnoption. La commande suivante permet uniquement SNS à Amazon d'appeler la fonction pour les abonnements à une rubrique nomméemy-topic.

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns-my-topic \
  --principal sns.amazonaws.com \
  --source-arn arn:aws:sns:us-east-2:123456789012:my-topic
```

Certains services peuvent invoquer des fonctions dans d'autres comptes. Si vous spécifiez une source ARN contenant votre identifiant de compte, cela ne pose aucun problème. Pour Amazon S3, toutefois, la source est un compartiment qui ARN ne contient pas d'ID de compte. Il est possible que vous puissiez supprimer le compartiment et qu'un autre compte crée un compartiment du même nom. Utilisez l'option `source-account` avec votre ID de compte pour vous assurer que seules les ressources de votre compte peuvent invoquer la fonction.

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
```

```
  --statement-id s3-account \
  --principal s3.amazonaws.com \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
  --source-account 123456789012
```

## Octroi de l'accès à la fonction à une organisation

Pour accorder des autorisations à une organisation dans AWS Organizations, spécifiez l'ID de l'organisation en tant que `principal-org-id`. La commande add permission suivante accorde un accès d'invocation à tous les utilisateurs de l'organisation. `o-a1b2c3d4e5f`

```
aws lambda add-permission \
  --function-name example \
  --statement-id PrincipalOrgIDExample \
  --action lambda:InvokeFunction \
  --principal * \
  --principal-org-id o-a1b2c3d4e5f
```

> **Note**
>
> Dans cette commande, `Principal` est `*`. Cela signifie que tous les utilisateurs de l'organisation `o-a1b2c3d4e5f` obtiennent des autorisations d'invocation de fonction. Si vous spécifiez un rôle Compte AWS ou comme étant le`Principal`, seul ce principal obtient les autorisations d'invocation des fonctions, mais uniquement s'il fait également partie de l'`o-a1b2c3d4e5f`organisation.

Cette commande crée une politique basée sur les ressources qui ressemble à ce qui suit :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalOrgIDExample",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:example",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
```

```
                }
            }
        }
    ]
}
```

Pour plus d'informations, consultez aws : PrincipalOrg ID dans le guide de IAM l'utilisateur.

## Accorder l'accès à la fonction Lambda à d'autres comptes

Pour partager une fonction avec une autre Compte AWS, ajoutez une déclaration d'autorisations entre comptes à la politique basée sur les ressources de la fonction. Exécutez la commande add permission et spécifiez l'ID du compte sous la `principal` forme. L'exemple suivant accorde au compte `111122223333` l'autorisation d'invoquer la fonction `my-function` avec l'alias `prod`.

```
aws lambda add-permission \
  --function-name my-function:prod \
  --statement-id xaccount \
  --action lambda:InvokeFunction \
  --principal 111122223333 \
  --output text
```

Vous devriez voir la sortie suivante:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:l
east-1:123456789012:function:my-function"}
```

La stratégie basée sur les ressources accorde à l'autre compte l'autorisation d'accéder à la fonction, mais n'autorise pas les utilisateurs de ce compte à dépasser leurs autorisations. Les utilisateurs de l'autre compte doivent disposer des autorisations utilisateur correspondantes pour utiliser le LambdaAPI.

Pour limiter l'accès à un utilisateur ou à un rôle dans un autre compte, spécifiez ARN l'identité complète en tant que principal. Par exemple, `arn:aws:iam::123456789012:user/developer`.

L'alias limite la version pouvant être invoquée par l'autre compte. L'autre compte doit inclure l'alias dans la fonctionARN.

```
aws lambda invoke \
  --function-name arn:aws:lambda:us-east-2:123456789012:function:my-function:prod out
```

Vous devriez voir la sortie suivante :

```
{
    "StatusCode": 200,
    "ExecutedVersion": "1"
}
```

Le propriétaire de la fonction peut alors mettre à jour l'alias afin qu'il pointe vers une nouvelle version sans que l'appelant ait besoin de modifier la façon dont il invoque votre fonction. Cela permet de s'assurer que l'autre compte n'a pas besoin de changer son code pour utiliser la nouvelle version et qu'il est uniquement autorisé à invoquer la version de la fonction associée à l'alias.

Vous pouvez accorder un accès entre comptes pour la plupart API des actions effectuées sur une fonction existante. Par exemple, vous pouvez accorder l'accès à `lambda:ListAliases` afin qu'un compte puisse obtenir une liste des alias ou à `lambda:GetFunction` pour qu'il puisse télécharger le code de votre fonction. Ajoutez chaque autorisation séparément ou utilisez `lambda:*` pour accorder l'accès à toutes les actions pour la fonction spécifiée.

Pour autoriser d'autres comptes à accéder à plusieurs fonctions ou à effectuer des actions qui ne s'appliquent pas à une fonction, nous vous recommandons d'utiliser IAMdes rôles.

## Octroi de l'accès à la couche Lambda à d'autres comptes

Pour partager une couche avec une autre Compte AWS, ajoutez une déclaration d'autorisations entre comptes à la politique basée sur les ressources de la couche. Exécutez la add-layer-version-permissioncommande et spécifiez l'ID du compte sous la formeprincipal. Dans chaque relevé, vous pouvez autoriser un seul compte, tous les comptes ou une organisation dans AWS Organizations.

L'exemple suivant autorise le compte 111122223333 à accéder à la version 2 de la couche `bash-runtime`.

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Vous devez voir des résultats similaires à ce qui suit :

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Les autorisations ne s'appliquent qu'à une seule version de couche. Répétez le processus chaque fois que vous créez une nouvelle version de la couche.

Pour accorder l'autorisation à tous les comptes d'une AWS Organizationsorganisation, utilisez l'`organization-id`option. L'exemple suivant autorise tous les comptes de l'organisation `o-t194hfs8cz` à utiliser la version 3 de`my-layer`.

```
aws lambda add-layer-version-permission \
  --layer-name my-layer \
  --version-number 3 \
  --statement-id engineering-org \
  --principal '*' \
  --action lambda:GetLayerVersion \
  --organization-id o-t194hfs8cz \
  --output text
```

Vous devriez voir la sortie suivante :

```
{"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam
east-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

Pour autoriser plusieurs comptes ou organisations, vous devez ajouter plusieurs relevés.

## Utilisation du contrôle d'accès basé sur les attributs dans Lambda

Avec le contrôle d'accès basé sur les attributs (ABAC), vous pouvez utiliser des balises pour contrôler l'accès à vos ressources Lambda. Vous pouvez associer des tags à certaines ressources Lambda, à certaines API demandes ou au principal AWS Identity and Access Management (IAM) auteur de la demande. Pour plus d'informations sur la manière dont l'accès est AWS accordé en fonction des attributs, consultez la section Contrôle de l'accès aux AWS ressources à l'aide de balises dans le Guide de l'IAMutilisateur.

Vous pouvez l'utiliser ABAC pour accorder le moindre privilège sans spécifier de nom de ressource Amazon (ARN) ou de ARN modèle dans la IAM politique. Au lieu de cela, vous pouvez spécifier une balise dans l'élément condition d'une IAM politique pour contrôler l'accès. La mise à l'échelle est plus facile, ABAC car vous n'avez pas à mettre à jour vos IAM politiques lorsque vous créez de nouvelles ressources. Ajoutez plutôt des balises aux nouvelles ressources pour contrôler l'accès.

Dans Lambda, les balises fonctionnent sur les ressources suivantes :

- Fonctions : pour plus d'informations sur les fonctions de balisage, voir,. the section called “Balises”
- Configurations de signature de code : pour plus d'informations sur le balisage des configurations de signature de code, consultez. the section called “Balises de configuration de signature de code”
- Mappages de sources d'événements : pour plus d'informations sur le balisage des mappages de sources d'événements, consultez. the section called “Balises de mappage des sources d'événements”

Les balises ne sont pas prises en charge pour les couches.

Vous pouvez utiliser les clés de condition suivantes pour écrire des règles IAM de politique basées sur des balises :

- aws : ResourceTag /tag-key : contrôlez l'accès en fonction des balises associées à une ressource Lambda.
- aws : RequestTag /tag-key : exige la présence de balises dans une demande, par exemple lors de la création d'une nouvelle fonction.
- aws : PrincipalTag /tag-key : contrôlez ce que le IAM principal (la personne qui fait la demande) est autorisé à faire en fonction des balises associées à son IAM utilisateur ou à son rôle.
- aws : TagKeys : Contrôle si des clés de balise spécifiques peuvent être utilisées dans une requête.

Vous ne pouvez définir des conditions que pour les actions qui les soutiennent. Pour obtenir la liste des conditions prises en charge par chaque action Lambda, consultez la section Actions, ressources et clés de condition AWS Lambda dans la référence d'autorisation de service. Pour la prise en charge d'aws : ResourceTag /tag-key, reportez-vous à « Types de ressources définis par ». AWS Lambda Pour aws : RequestTag /tag-key et aws : TagKeys support, reportez-vous à « Actions définies par ». AWS Lambda

Rubriques

- Sécurisez vos fonctions par tag

## Sécurisez vos fonctions par tag

Les étapes suivantes montrent une méthode de configuration des autorisations pour les fonctions utilisantABAC. Dans cet exemple de scénario, vous allez créer quatre politiques IAM d'autorisation. Vous allez ensuite associer ces politiques à un nouveau IAM rôle. Enfin, vous allez créer un IAM utilisateur et lui donner l'autorisation d'assumer le nouveau rôle.

Rubriques


- Prérequis
- Étape 1 : Exiger des balises pour les nouvelles fonctions
- Étape 2 : Autoriser les actions basées sur les balises associées à une fonction Lambda et à un principal IAM
- Étape 3 : Accorder des permissions de liste
- Étape 4 : accorder des autorisations IAM
- Étape 5 : Créer le rôle IAM
- Étape 6 : Création d'un utilisateur IAM
- Étape 7 : Tester les autorisations
- Étape 8 : Nettoyez vos ressources


### Prérequis

Assurez-vous que vous disposez d'un rôle d'exécution Lambda. Vous utiliserez ce rôle lorsque vous accorderez IAM des autorisations et lorsque vous créerez une fonction Lambda.

### Étape 1 : Exiger des balises pour les nouvelles fonctions

Lors de l'utilisation ABAC avec Lambda, il est recommandé d'exiger que toutes les fonctions possèdent des balises. Cela permet de garantir que vos politiques ABAC d'autorisation fonctionnent comme prévu.

Créez une IAM politique similaire à l'exemple suivant. Cette politique utilise les clés de TagKeys condition aws : RequestTag /tag-key, aws : ResourceTag /tag-key et aws : pour exiger que les nouvelles fonctions et le IAM principal qui les crée possèdent tous deux la balise. `project` Le `ForAllValues` modificateur garantit que `project` est la seule balise autorisée. Si vous n'incluez pas`ForAllValues`, les utilisateurs peuvent ajouter d'autres balises à la fonction tant qu'ils transmettent également `project`.

Example — Exiger des balises sur les nouvelles fonctions

```
{
    "Version": "2012-10-17",
    "Statement": {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:TagResource"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/project": "${aws:PrincipalTag/project}",
          "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": "project"
        }
      }
    }
  }
```

Étape 2 : Autoriser les actions basées sur les balises associées à une fonction Lambda et à un principal IAM

Créez une deuxième IAM politique à l'aide de la clé de condition aws : ResourceTag /tag-key pour exiger que la balise du principal corresponde à la balise attachée à la fonction. L'exemple de stratégie suivant autorise les principaux avec la balise `project` pour appeler des fonctions avec la balise `project`. Si une fonction possède d'autres balises, l'action est refusée.

Example — Exige des balises correspondantes sur la fonction et le IAM principal

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunction"
        ],
```

```
        "Resource": "arn:aws:lambda:*:*:function:*",
        "Condition": {
          "StringEquals": {
            "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
          }
        }
      }
    ]
  }
```

Étape 3 : Accorder des permissions de liste

Créez une politique qui permet au principal de répertorier les fonctions et IAM les rôles Lambda. Cela permet au directeur de voir toutes les fonctions et tous les IAM rôles Lambda sur la console et lors de l'APIappel des actions.

Example — Accorder des autorisations Lambda et de liste IAM

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "AllResourcesLambdaNoTags",
        "Effect": "Allow",
        "Action": [
          "lambda:GetAccountSettings",
          "lambda:ListFunctions",
          "iam:ListRoles"
        ],
        "Resource": "*"
      }
    ]
  }
```

Étape 4 : accorder des autorisations IAM

Créez une politique qui autorise iam : PassRole. Cette autorisation est requise lorsque vous attribuez un rôle d'exécution à une fonction. Dans l'exemple de politique suivant, remplacez l'exemple ARN par celui de votre ARN rôle d'exécution Lambda.

> **Note**
>
> N'utilisez pas la clé de condition `ResourceTag` dans une politique avec l'action `iam:PassRole`. Vous ne pouvez pas utiliser le tag d'un IAM rôle pour contrôler l'accès à qui peut transmettre ce rôle. Pour plus d'informations sur les autorisations requises pour transmettre un rôle à un service, consultez la section Octroi à un utilisateur des autorisations pour transmettre un rôle à un AWS service.

Example — Accordez l'autorisation de transmettre le rôle d'exécution

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
          "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
      }
    ]
  }
```

Étape 5 : Créer le rôle IAM

C'est une bonne pratique d'utilisation des rôles pour déléguer des autorisations. Créez un rôle IAM appelé `abac-project-role` :

- À l'Étape 1 : Sélection d'une entité de confiance : Choisissez :AWS compte, puis Ce compte.
- À l'étape 2 : Ajouter des autorisations : Joignez les quatre IAM politiques que vous avez créées lors des étapes précédentes.
- À l'Étape 3 : Nommer, vérifier et créer : Choisissez :Ajouter une balise. Pour Key (Clé), saisissez `project`. Ne saisissez pas Valeur.

Étape 6 : Création d'un utilisateur IAM

Création d'un utilisateur IAM appelé `abac-test-user`. Sur la page Définir des autorisations, choisissez Attacher directement les stratégies existantes, puis choisissez Créer une stratégie. Entrez la définition de politique suivante. Remplacez *111122223333* avec votre identifiant de AWS compte. Cette stratégie permet à `abac-test-user` d'assumer `abac-project-role`.

Example — Autoriser IAM l'utilisateur à assumer ABAC un rôle

```
{
    "Version": "2012-10-17",
    "Statement": {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
    }
  }
```

Étape 7 : Tester les autorisations

1. Connectez-vous à la AWS console en tant que`abac-test-user`. Pour plus d'informations, voir Se connecter en tant qu'IAMutilisateur.
2. Basculez vers le rôle `abac-project-role`. Pour plus d'informations, consultez Changement de rôles (console).
3. Création d'une fonction Lambda.
   - Sous Autorisations Choisissez Modifier le rôle d'exécution par défaut, puis dans le champ Rôle d'exécution, choisissez Utiliser un rôle existant. Choisissez le même rôle d'exécution que celui que vous avez utilisé dansÉtape 4 : accorder des autorisations IAM.
   - Sous Paramètres avancés, choisissez Activer les balises, puis Ajouter un nouveau tag. Pour Key (Clé), saisissez `project`. Ne saisissez pas Valeur.
4. Test de la fonction.
5. Créez une deuxième fonction Lambda et ajoutez une balise différente, telle que`environment`. Cette opération devrait échouer car la ABAC politique que vous avez créée permet Étape 1 : Exiger des balises pour les nouvelles fonctions uniquement au principal de créer des fonctions avec la `project` balise.

6. Créez une troisième fonction sans balises. Cette opération devrait échouer car la ABAC politique que vous avez créée Étape 1 : Exiger des balises pour les nouvelles fonctions n'autorise pas le principal à créer des fonctions sans balises.

Cette stratégie d'autorisation vous permet de contrôler l'accès sans créer de nouvelles politiques pour chaque nouvel utilisateur. Pour accorder l'accès à de nouveaux utilisateurs, il suffit de leur donner l'autorisation d'assumer le rôle qui correspond au projet qui leur a été attribué.

Étape 8 : Nettoyez vos ressources

Pour supprimer le IAM rôle

1. Ouvrez la page Rôles de la IAM console.
2. Sélectionnez le rôle que vous avez créé à l'étape 5.
3. Sélectionnez Delete (Supprimer).
4. Pour confirmer la suppression, entrez le nom du rôle dans le champ de saisie de texte.
5. Sélectionnez Delete (Supprimer).

Pour supprimer l'IAMutilisateur

1. Ouvrez la page Utilisateurs de la IAM console.
2. Sélectionnez l'IAMutilisateur que vous avez créé à l'étape 6.
3. Sélectionnez Delete (Supprimer).
4. Pour confirmer la suppression, entrez le nom d'utilisateur dans le champ de saisie de texte.
5. Choisissez Delete user (Supprimer l'utilisateur).

Pour supprimer la fonction Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction que vous avez créée.
3. Sélectionnez Actions, Supprimer.
4. Saisissez **`delete`** dans la zone de saisie de texte et choisissez Delete (Supprimer).

# Affiner les sections Ressources et Conditions des politiques

Vous pouvez restreindre l'étendue des autorisations d'un utilisateur en spécifiant les ressources et les conditions dans une politique AWS Identity and Access Management (IAM). Chaque action d'une politique prend en charge une combinaison de types de ressources et de conditions qui varie en fonction du comportement de l'action.

Chaque déclaration de IAM politique autorise une action exécutée sur une ressource. Lorsque l'action n'agit pas sur une ressource nommée, ou lorsque vous accordez l'autorisation d'effectuer l'action sur toutes les ressources, la valeur de la ressource de la stratégie est un caractère générique (*). Pour de nombreuses actions, vous pouvez restreindre les ressources qu'un utilisateur peut modifier en spécifiant le nom de ressource Amazon (ARN) d'une ressource ou en utilisant un ARN modèle correspondant à plusieurs ressources.

Par type de ressource, la conception générale de la manière de limiter la portée d'une action est la suivante :

- Fonctions : les actions qui agissent sur une fonction peuvent être limitées à une fonction spécifique par fonction, version ou alias. ARN
- Mappages de sources d'événements : les actions peuvent être limitées à des ressources de mappage de sources d'événements spécifiques par. ARN Les mappages de sources d'événements sont toujours associés à une fonction. Vous pouvez également utiliser `lambda:FunctionArn` cette condition pour restreindre les actions par fonction associée.
- Couches : les actions liées à l'utilisation des couches et aux autorisations agissent sur une version d'une couche.
- Configuration de signature de code : les actions peuvent être limitées à des ressources de configuration de signature de code spécifiques par. ARN
- Balises : utilisez les conditions de balise standard. Pour de plus amples informations, veuillez consulter the section called “Contrôle d'accès basé sur les attributs”.

Pour restreindre les autorisations par ressource, spécifiez la ressource parARN.

Format de ressource Lambda ARN

- Fonction – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*
- Version de la fonction – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*`:`*`1`*

- Alias de la fonction – `arn:aws:lambda:us-west-2:123456789012:function:my-function:TEST`
- Mappage de source d'événement – `arn:aws:lambda:us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47`
- Couche – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer`
- Version de la couche – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer:1`
- Configuration de signature de code — `arn:aws:lambda:us-west-2:123456789012:code-signing-config:my-csc`

Par exemple, la politique suivante permet Compte AWS `123456789012` à un utilisateur d'appeler une fonction nommée `my-function` dans la AWS région USA Ouest (Oregon).

Example appeler une stratégie de fonction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

Il s'agit d'un cas particulier où l'identifiant d'action (`lambda:InvokeFunction`) est différent de l'APIopération (Invoke). Pour les autres actions, l'identificateur de l'action est le nom de l'opération avec le préfixe `lambda:`.

Sections

- Comprendre la section Condition des politiques
- Fonctions de référencement dans la section Ressources des politiques
- IAMActions prises en charge et comportements fonctionnels

## Comprendre la section Condition des politiques

Les conditions sont facultatives dans la stratégie, et appliquent une logique supplémentaire pour déterminer si une action est autorisée. Outre les conditions courantes prises en charge par toutes les actions, Lambda définit les types de condition que vous pouvez utiliser pour restreindre les valeurs des paramètres supplémentaires sur certaines actions.

Par exemple, la condition `lambda:Principal` permet de restreindre le service ou le compte auquel un utilisateur peut accorder l'accès selon la stratégie basée sur les ressources d'une fonction. La politique suivante permet à un utilisateur d'autoriser les rubriques Amazon Simple Notification Service (AmazonSNS) à invoquer une fonction nomméetest.

Example gestion des autorisations d'une stratégie de fonction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:test:*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "sns.amazonaws.com"
                }
            }
        }
    ]
}
```

La condition exige que le mandant soit Amazon SNS et non un autre service ou compte. Le modèle de ressource exige que le nom de la fonction soit `test` et inclue un numéro de version ou un alias. Par exemple, `test:v1`.

Pour plus d'informations sur les ressources et les conditions de Lambda et d'autres AWS services, consultez la section Actions, ressources et clés de condition pour les AWS services dans la référence d'autorisation de service.

## Fonctions de référencement dans la section Ressources des politiques

Vous référencez une fonction Lambda dans une déclaration de politique à l'aide d'un Amazon Resource Name ()ARN. Le format d'une fonction ARN varie selon que vous faites référence à l'ensemble de la fonction (non qualifiée) ou à une version de fonction ou à un alias (qualifié).

Lorsqu'ils effectuent des API appels Lambda, les utilisateurs peuvent spécifier une version ou un alias en transmettant une version ARN ou un alias ARN dans le GetFunction`FunctionName`paramètre, ou en définissant une valeur dans le GetFunction`Qualifier`paramètre. Lambda prend des décisions d'autorisation en comparant l'élément de ressource de la IAM politique avec les appels `FunctionName` et les appels `Qualifier` transmis. API En cas d'incompatibilité, Lambda refuse la demande.

Que vous autorisiez ou refusiez une action sur votre fonction, vous devez utiliser les ARN types de fonctions appropriés dans votre déclaration de politique pour obtenir les résultats escomptés. Par exemple, si votre politique fait référence au non qualifiéARN, Lambda accepte les demandes qui font référence au ARN non qualifié mais refuse les demandes qui font référence à un qualifié. ARN

> **Note**
>
> Vous ne pouvez pas utiliser un caractère générique (*) pour établir une correspondance avec l'ID du compte. Pour plus d'informations sur la syntaxe acceptée, consultez la référence IAM JSON aux politiques dans le Guide de IAM l'utilisateur.

Example autoriser l'invocation d'une personne non qualifiée ARN

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction"
        }
    ]
}
```

Si votre politique fait référence à une personne qualifiée spécifiqueARN, Lambda accepte les demandes qui y font référence, ARN mais refuse les demandes faisant référence à une personne non qualifiée ARN ou à une qualification différenteARN, par exemple. `myFunction:2`

Example permettant l'invocation d'un qualifié spécifique ARN

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:1"
        }
    ]
}
```

Si votre politique fait référence à une ARN utilisation qualifiée`:*`, Lambda accepte toute demande qualifiée ARN mais refuse les demandes faisant référence à une utilisation non qualifiée. ARN

Example permettant l'invocation de toute personne qualifiée ARN

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:*"
        }
    ]
}
```

Si votre politique fait référence à une ARN utilisation`*`, Lambda accepte toute utilisation qualifiée ou non qualifiée. ARN

Example permettant l'invocation de toute personne qualifiée ou non qualifiée ARN

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction*"
        }
    ]
}
```

## IAMActions prises en charge et comportements fonctionnels

Les actions définissent ce qui peut être autorisé par le biais IAM de politiques. Pour obtenir la liste des actions prises en charge dans Lambda, consultez la section Actions, ressources et clés de condition AWS Lambda dans la référence d'autorisation de service. Dans la plupart des cas, lorsqu'une IAM action autorise une API action Lambda, le nom de l'IAMaction est identique à celui de l'action Lambda, avec les API exceptions suivantes :

| APIaction | IAMaction |
| --- | --- |
| Invoke | `lambda:InvokeFunction` |
| GetLayerVersion<br><br>GetLayerVersionByArn | `lambda:GetLayerVersion` |

Outre les ressources et les conditions définies dans la référence d'autorisation de service, Lambda prend en charge les ressources et conditions suivantes pour certaines actions. Nombre d'entre elles sont liées aux fonctions de référencement dans la section des ressources des politiques. Les actions qui agissent sur une fonction peuvent être limitées à une fonction spécifique par fonction, version ou aliasARN, comme décrit dans le tableau suivant.

| Action | Ressource | Condition |
| --- | --- | --- |
| AddPermission<br><br>RemovePermission<br><br>Invoquer — Autorisation : `lambda:InvokeFunction` | Version de la fonction<br><br>Alias de la fonction | N/A |

| Action | Ressource | Condition |
| --- | --- | --- |
| UpdateFunctionConfiguration | N/A | `lambda:CodeSigningConfigArn` |
| CreateFunctionUrlConfig<br>DeleteFunctionUrlConfig<br>GetFunctionUrlConfig<br>UpdateFunctionUrlConfig | Alias de la fonction | N/A |

# Sécurité dans AWS Lambda

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de conformitéAWS. Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS Lambda, consultez la section AWS Services concernés par programme de conformité.
- Sécurité dans le cloud : votre responsabilité est déterminée par le service AWS que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre entreprise,et la législation et la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de Lambda. Les rubriques suivantes vous montrent comment configurer Lambda pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos ressources Lambda.

Pour plus d'informations sur l'application de principes de sécurité aux applications Lambda, Consultez Sécurité dans Serverless Land.

Rubriques


- Protection des données dans AWS Lambda
- Identity and Access Management pour AWS Lambda
- Création d'une stratégie de gouvernance pour les fonctions et les couches Lambda
- Validation de la conformité pour AWS Lambda
- Résilience dans AWS Lambda
- Sécurité de l'infrastructure dans AWS Lambda
- Utilisation de la signature de code pour vérifier l'intégrité du code avec Lambda

# Protection des données dans AWS Lambda

Le modèle de responsabilité AWS partagée de s'applique à la protection des données dans AWS Lambda. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez la section Confidentialité des données FAQ. Pour plus d'informations sur la protection des données en Europe, consultez le modèle de responsabilitéAWS partagée et le billet de GDPR blog sur le blog sur la AWS sécurité.

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) pour chaque compte.
- UtilisezSSL/TLSpour communiquer avec les AWS ressources. Nous avons besoin de la TLS version 1.2 et recommandons la TLS version 1.3.
- Configuration API et journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section Utilisation des CloudTrail sentiers dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de FIPS 140 à 3 modules cryptographiques validés pour accéder AWS via une interface de ligne de commande ou unAPI, utilisez un point de terminaison. FIPS Pour plus d'informations sur les FIPS points de terminaison disponibles, voir Federal Information Processing Standard (FIPS) 140-3.

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec Lambda ou une autre personne à Services AWS l'aide de la console, API AWS CLI, ou. AWS SDKs Toutes les données que vous entrez dans

des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez un URL à un serveur externe, nous vous recommandons vivement de ne pas inclure d'informations d'identification dans le URL afin de valider votre demande auprès de ce serveur.

Sections


- Chiffrement en transit
- Chiffrement des données au repos pour AWS Lambda


## Chiffrement en transit

Les API points de terminaison Lambda ne prennent en charge que les connexions sécurisées. HTTPS Lorsque vous gérez les ressources Lambda avec le AWS Management Console, ou le Lambda AWS SDKAPI, toutes les communications sont chiffrées avec Transport Layer Security (). TLS Pour une liste complète des API points de terminaison, voir AWS Régions et points de terminaison dans le. Références générales AWS

Lorsque vous connectez votre fonction à un système de fichiers, Lambda utilise le chiffrement pendant le transit pour toutes les connexions. Pour plus d'informations, consultez la section Chiffrement des données dans Amazon EFS dans le manuel Amazon Elastic File System User Guide.

Lorsque vous utilisez variables d’environnement, vous pouvez activer les assistants de chiffrement de la console afin qu’ils utilisent le chiffrement côté client pour protéger les variables d’environnement en transit. Pour de plus amples informations, veuillez consulter Sécurisation des variables d'environnement Lambda.

## Chiffrement des données au repos pour AWS Lambda

Lambda chiffre toujours les variables d’environnement au repos. Par défaut, Lambda utilise une AWS KMS key que Lambda crée dans votre compte pour chiffrer vos variables d’environnement. C' Clé gérée par AWS est ce qu'on appelle`aws/lambda`.

Pour chaque fonction, vous pouvez éventuellement configurer Lambda pour qu'il utilise une clé gérée par le client au lieu de la clé Clé gérée par AWS par défaut pour chiffrer vos variables d'environnement. Pour de plus amples informations, veuillez consulter Sécurisation des variables d'environnement Lambda.

Si le mappage des sources d'événements de votre fonction comporte un objet de critères de filtre, Lambda chiffre également les critères de filtre à l'aide d'une AWS KMS clé par défaut. Vous pouvez éventuellement utiliser une clé gérée par le client pour chiffrer les critères de filtre.

Lambda chiffre toujours les fichiers que vous chargez sur Lambda, y compris les packages de déploiement et les archives de couches.

Amazon CloudWatch Logs chiffre AWS X-Ray également les données par défaut et peut être configuré pour utiliser une clé gérée par le client. Pour plus de détails, voir Chiffrer les données des CloudWatch journaux dans les journaux et Protection des données dans AWS X-Ray.

Sections

- Surveillance de vos clés de chiffrement pour Lambda

## Surveillance de vos clés de chiffrement pour Lambda

Lorsque vous utilisez une clé gérée par AWS KMS le client avec Lambda, vous pouvez utiliser. AWS CloudTrail Les exemples suivants sont CloudTrail des événements et des `GenerateDataKey` appels effectués par Lambda pour `Decrypt` accéder à des données chiffrées par votre clé gérée par le client. `DescribeKey`

Decrypt

Si vous avez utilisé une clé gérée par le AWS KMS client pour chiffrer votre objet de critères de filtrage, Lambda envoie `Decrypt` une demande en votre nom lorsque vous essayez d'y accéder en texte clair (par exemple, depuis `ListEventSourceMappings` un appel). L'exemple d'événement suivant enregistre l'opération `Decrypt` :

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
```

```
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:45:23Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2024-05-30T01:05:46Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
            "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
            "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
            "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
```

```
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

DescribeKey

Si vous avez utilisé une clé gérée par le AWS KMS client pour chiffrer votre objet de critères de filtre, Lambda envoie `DescribeKey` une demande en votre nom lorsque vous essayez d'y accéder (par exemple, depuis `GetEventSourceMapping` un appel). L'exemple d'événement suivant enregistre l'opération `DescribeKey` :

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:45:23Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-05-30T01:09:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "54.240.197.238",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36",
    "requestParameters": {
        "keyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    },
```

```
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.eu-west-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

### GenerateDataKey

Lorsque vous utilisez une clé gérée par le AWS KMS client pour chiffrer l'objet de vos critères de filtre dans un `UpdateEventSourceMapping` appel `CreateEventSourceMapping` OR, Lambda envoie `GenerateDataKey` une demande en votre nom pour générer une clé de données afin de chiffrer les critères de filtre (chiffrement d'enveloppe). L'exemple d'événement suivant enregistre l'opération `GenerateDataKey` :

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:06:07Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2024-05-30T01:04:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 32,
        "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
            "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
            "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
            "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
        },
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

# Identity and Access Management pour AWS Lambda

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. IAMles administrateurs contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser les ressources Lambda. IAMest un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques


- Public ciblé
- Authentification par des identités
- Gestion des accès à l’aide de politiques
- Comment AWS Lambda fonctionne avec IAM
- Exemples de politiques basées sur l’identité pour AWS Lambda
- Politiques gérées par AWS pour AWS Lambda
- Résolution des problèmes AWS Lambda d'identité et d'accès


## Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez dans Lambda.

Utilisateur du service – Si vous utilisez le service Lambda pour accomplir votre tâche, votre administrateur vous fournit les informations d’identification et les autorisations dont vous avez besoin. Plus vous utilisez de fonctions Lambda pour accomplir votre travail, plus vous avez besoin d’autorisations. Comprendre la gestion des accès peut vous aider à demander à votre administrateur les autorisations appropriées. Si vous ne pouvez pas accéder à une fonction dans Lambda, consultez Résolution des problèmes AWS Lambda d'identité et d'accès.

Administrateur du service – Si vous êtes responsable des ressources Lambda dans votre entreprise, vous bénéficiez probablement d’un accès total à Lambda. C’est à vous de déterminer les fonctions et

les fonctions et ressources Lambda auxquelles vos utilisateurs des services pourront accéder. Vous devez ensuite envoyer des demandes à votre IAM administrateur pour modifier les autorisations des utilisateurs de votre service. Consultez les informations de cette page pour comprendre les concepts de base deIAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM Lambda, consultez. Comment AWS Lambda fonctionne avec IAM

IAMadministrateur — Si vous êtes IAM administrateur, vous souhaiterez peut-être en savoir plus sur la manière dont vous pouvez rédiger des politiques pour gérer l'accès à Lambda. Pour consulter des exemples de politiques basées sur l'identité Lambda que vous pouvez utiliser dans, consultez. IAM Exemples de politiques basées sur l’identité pour AWS Lambda

## Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié (connecté à AWS) en tant que Utilisateur racine d'un compte AWS, en tant qu'IAMutilisateur ou en assumant un IAM rôle.

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAMIdentity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez en tant qu'identité fédérée, votre administrateur a préalablement configuré la fédération d'identité à l'aide de IAM rôles. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer vous-même les demandes, consultez la section Signature des AWS API demandes dans le guide de IAM l'utilisateur.

Quelle que soit la méthode d’authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour en savoir

plus, consultez Authentification multifactorielle dans le guide de AWS IAM Identity Center l'utilisateur et Utilisation de l'authentification multifactorielle (MFA) AWS dans le guide de l'IAMutilisateur.

### Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui nécessitent que vous vous connectiez en tant qu'utilisateur root, consultez la section Tâches nécessitant des informations d'identification utilisateur root dans le Guide de IAM l'utilisateur.

### Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles assument des rôles, qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour plus d'informations sur IAM Identity Center, consultez Qu'est-ce qu'IAMIdentity Center ? dans le guide de AWS IAM Identity Center l'utilisateur.

### Utilisateurs et groupes IAM

Un IAMutilisateur est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou une seule application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des IAM utilisateurs dotés d'informations d'identification à long terme, telles que des mots de

passe et des clés d'accès. Toutefois, si vous avez des cas d'utilisation spécifiques qui nécessitent des informations d'identification à long terme auprès des IAM utilisateurs, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, voir Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification à long terme dans le Guide de IAM l'utilisateur.

Un IAMgroupe est une identité qui définit un ensemble d'IAMutilisateurs. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez nommer un groupe IAMAdminset lui donner les autorisations nécessaires pour administrer IAM des ressources.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, voir Quand créer un IAM utilisateur (au lieu d'un rôle) dans le Guide de IAM l'utilisateur.

### IAMrôles

Un IAMrôle est une identité au sein de Compte AWS vous dotée d'autorisations spécifiques. Il est similaire à un IAM utilisateur, mais n'est pas associé à une personne en particulier. Vous pouvez assumer temporairement un IAM rôle dans le en AWS Management Console changeant de rôle. Vous pouvez assumer un rôle en appelant une AWS API opération AWS CLI or ou en utilisant une option personnaliséeURL. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez la section Méthodes pour assumer un rôle dans le Guide de IAM l'utilisateur.

IAMles rôles dotés d'informations d'identification temporaires sont utiles dans les situations suivantes :

- Accès utilisateur fédéré : pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour plus d'informations sur les rôles pour la fédération, voir Création d'un rôle pour un fournisseur d'identité tiers dans le guide de IAM l'utilisateur. Si vous utilisez IAM Identity Center, vous configurez un ensemble d'autorisations. Pour contrôler les accès auxquels vos identités peuvent accéder après leur authentification, IAM Identity Center met en corrélation l'ensemble d'autorisations avec un rôle dans. IAM Pour plus d'informations sur les jeux d'autorisations, consultez Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .

- Autorisations IAM utilisateur temporaires : un IAM utilisateur ou un rôle peut assumer un IAM rôle afin d'obtenir temporairement différentes autorisations pour une tâche spécifique.
- Accès entre comptes : vous pouvez utiliser un IAM rôle pour autoriser une personne (un mandant fiable) d'un autre compte à accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour connaître la différence entre les rôles et les politiques basées sur les ressources pour l'accès entre comptes, voir Accès aux ressources entre comptes IAM dans le guide de l'IAMutilisateur.
- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, un rôle de service ou un rôle lié au service.
  - Sessions d'accès transmises (FAS) — Lorsque vous utilisez un IAM utilisateur ou un rôle pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FASutilise les autorisations du principal appelant an Service AWS, combinées à la demande Service AWS pour adresser des demandes aux services en aval. FASles demandes ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur les politiques relatives FAS aux demandes, consultez la section Transférer les sessions d'accès.
  - Rôle de service — Un rôle de service est un IAMrôle qu'un service assume pour effectuer des actions en votre nom. Un IAM administrateur peut créer, modifier et supprimer un rôle de service de l'intérieurIAM. Pour plus d'informations, consultez la section Création d'un rôle auquel déléguer des autorisations Service AWS dans le Guide de IAM l'utilisateur.
  - Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un IAM administrateur peut consulter, mais pas modifier les autorisations pour les rôles liés à un service.
- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un IAM rôle pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une EC2 instance et qui AWS CLI soumettent des AWS API demandes. Cela est préférable au stockage des clés d'accès dans l'EC2instance. Pour attribuer un AWS rôle à une EC2 instance et le rendre disponible pour toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil

d'instance contient le rôle et permet aux programmes exécutés sur l'EC2instance d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez la section Utilisation d'un IAM rôle pour accorder des autorisations aux applications exécutées sur des EC2 instances Amazon dans le Guide de IAM l'utilisateur.

Pour savoir s'il faut utiliser IAM des rôles ou des IAM utilisateurs, voir Quand créer un IAM rôle (au lieu d'un utilisateur) dans le guide de IAM l'utilisateur.

## Gestion des accès à l’aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de JSON documents. Pour plus d'informations sur la structure et le contenu des documents de JSON politique, voir Présentation des JSON politiques dans le guide de IAM l'utilisateur.

Les administrateurs peuvent utiliser AWS JSON des politiques pour spécifier qui a accès à quoi. C’est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d’aucune autorisation. Pour autoriser les utilisateurs à effectuer des actions sur les ressources dont ils ont besoin, un IAM administrateur peut créer des IAM politiques. L'administrateur peut ensuite ajouter les IAM politiques aux rôles, et les utilisateurs peuvent assumer les rôles.

IAMles politiques définissent les autorisations pour une action, quelle que soit la méthode que vous utilisez pour effectuer l'opération. Par exemple, supposons que vous disposiez d’une politique qui autorise l’action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle auprès du AWS Management Console AWS CLI, ou du AWS API.

### Politiques basées sur l’identité

Les politiques basées sur l'identité sont JSON des documents de politique d'autorisation que vous pouvez joindre à une identité, telle qu'un IAM utilisateur, un groupe d'utilisateurs ou un rôle. Ces politiques contrôlent quel type d’actions des utilisateurs et des rôles peuvent exécuter, sur quelles

ressources et dans quelles conditions. Pour savoir comment créer une politique basée sur l'identité, consultez la section Création de IAM politiques dans le Guide de l'IAMutilisateur.

Les politiques basées sur l’identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour savoir comment choisir entre une politique gérée ou une politique intégrée, voir Choisir entre des politiques gérées et des politiques intégrées dans le Guide de l'IAMutilisateur.

## Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents JSON de stratégie que vous attachez à une ressource. Les politiques de confiance dans les IAM rôles et les politiques relatives aux compartiments Amazon S3 sont des exemples de politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l’accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d’actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser de politiques AWS gérées depuis une IAM stratégie basée sur les ressources.

## Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLssont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format du document JSON de stratégie.

Amazon S3 et Amazon VPC sont des exemples de services compatiblesACLs. AWS WAF Pour en savoir plusACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

## Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- Limites d'autorisations — Une limite d'autorisations est une fonctionnalité avancée dans laquelle vous définissez le maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une IAM entité (IAMutilisateur ou rôle). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations en résultant représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez la section Limites d'autorisations pour les IAM entités dans le Guide de IAM l'utilisateur.
- Politiques de contrôle des services (SCPs) : SCPs JSON politiques qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans AWS Organizations. AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctionnalités d'une organisation, vous pouvez appliquer des politiques de contrôle des services (SCPs) à l'un ou à l'ensemble de vos comptes. Les SCP limites d'autorisations pour les entités présentes dans les comptes des membres, y compris chacune d'entre elles Utilisateur racine d'un compte AWS. Pour plus d'informations sur les Organizations et consultez SCPs les politiques de contrôle des services dans le Guide de AWS Organizations l'utilisateur.
- Politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de séance en résultant sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations, consultez la section Politiques de session dans le guide de IAM l'utilisateur.

## Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser

une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de IAM l'utilisateur.

## Comment AWS Lambda fonctionne avec IAM

Avant de commencer IAM à gérer l'accès à Lambda, découvrez quelles IAM fonctionnalités peuvent être utilisées avec Lambda.

| IAMfonctionnalité | Assistance Lambda |
|---|---|
| Politiques basées sur l’identité | Oui |
| Politiques basées sur les ressources | Oui |
| Actions de politique | Oui |
| Ressources de politique | Oui |
| Clés de condition de politique (spécifiques au service) | Oui |
| ACLs | Non |
| ABAC(balises dans les politiques) | Partielle |
| Informations d’identification temporaires | Oui |
| Transférer les sessions d'accès (FAS) | Non |
| Rôles de service | Oui |
| Rôles liés à un service | Partielle |

Pour obtenir une vue d'ensemble de la façon dont Lambda et les autres AWS services fonctionnent avec la plupart des IAM fonctionnalités, consultez la section AWS Services compatibles IAM dans le Guide de l'IAMutilisateur.

### Politiques basées sur l'identité pour Lambda

Prend en charge les politiques basées sur l’identité : oui

Les politiques basées sur l'identité sont JSON des documents de politique d'autorisation que vous pouvez joindre à une identité, telle qu'un IAM utilisateur, un groupe d'utilisateurs ou un rôle. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour savoir comment créer une politique basée sur l'identité, consultez la section Création de IAM politiques dans le Guide de l'IAMutilisateur.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier les actions et les ressources autorisées ou refusées ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Vous ne pouvez pas spécifier le principal dans une politique basée sur une identité, car celle-ci s'applique à l'utilisateur ou au rôle auquel elle est attachée. Pour en savoir plus sur tous les éléments que vous pouvez utiliser dans une JSON politique, consultez la référence aux éléments de IAM JSON politique dans le Guide de IAM l'utilisateur.

Exemples de politiques basées sur l'identité pour Lambda

Pour consulter des exemples de politiques basées sur l'identité Lambda, consultez. Exemples de politiques basées sur l'identité pour AWS Lambda

## Politiques basées sur les ressources au sein de Lambda

Prend en charge les politiques basées sur les ressources : Oui

Les politiques basées sur les ressources sont des documents JSON de stratégie que vous attachez à une ressource. Les politiques de confiance dans les IAM rôles et les politiques relatives aux compartiments Amazon S3 sont des exemples de politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour activer l'accès entre comptes, vous pouvez spécifier un compte entier ou IAM des entités d'un autre compte comme principal dans une politique basée sur les ressources. L'ajout d'un principal entre comptes à une politique basée sur les ressources ne représente qu'une partie de l'instauration de la relation d'approbation. Lorsque le principal et la ressource sont différents Comptes AWS, un IAM administrateur du compte de confiance doit également accorder à l'entité principale (utilisateur ou rôle) l'autorisation d'accéder à la ressource. Pour ce faire, il attache une politique basée sur une identité à l'entité. Toutefois, si une politique basée sur des ressources accorde l'accès à un

principal dans le même compte, aucune autre politique basée sur l'identité n'est requise. Pour plus d'informations, voir Accès aux ressources entre comptes IAM dans le Guide de IAM l'utilisateur.

Vous pouvez associer une politique basée sur les ressources à une fonction ou à une couche Lambda. Cette politique définit les principaux autorisés à effectuer des actions sur la fonction ou la couche.

Pour savoir comment associer une politique basée sur les ressources à une fonction ou à une couche, consultez. Utilisation de IAM politiques basées sur les ressources dans Lambda

## Actions politiques pour Lambda

Prend en charge les actions de politique : oui

Les administrateurs peuvent utiliser AWS JSON des politiques pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'`Action`élément d'une JSON politique décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès dans une politique. Les actions de stratégie portent généralement le même nom que l' AWS APIopération associée. Il existe certaines exceptions, telles que les actions avec autorisation uniquement qui n'ont pas d'opération correspondante. API Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour consulter la liste des actions Lambda, reportez-vous à la section Actions définies par AWS Lambda dans la référence d'autorisation de service.

Les actions de politique dans Lambda utilisent le préfixe suivant avant l'action :

```
lambda
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [
      "lambda:action1",
      "lambda:action2"
         ]
```

Pour consulter des exemples de politiques basées sur l'identité Lambda, consultez. Exemples de politiques basées sur l’identité pour AWS Lambda

## Ressources relatives aux politiques pour Lambda

Prend en charge les ressources de politique : oui

Les administrateurs peuvent utiliser AWS JSON des politiques pour spécifier qui a accès à quoi. C’est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Resource` JSON de stratégie indique le ou les objets auxquels s'applique l'action. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de spécifier une ressource en utilisant son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d’indiquer que l’instruction s’applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de ressources Lambda et leurs caractéristiquesARNs, consultez la section Types de ressources définis par AWS Lambda dans la référence d'autorisation de service. Pour savoir avec quelles actions vous pouvez spécifier pour chaque ressource, consultez la ARN section Actions définies par AWS Lambda.

Pour consulter des exemples de politiques basées sur l'identité Lambda, consultez. Exemples de politiques basées sur l’identité pour AWS Lambda

## Clés de conditions de politique pour Lambda

Prend en charge les clés de condition de politique spécifiques au service : oui

Les administrateurs peuvent utiliser AWS JSON des politiques pour spécifier qui a accès à quoi. C’est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une `OR` opération logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez autoriser un IAM utilisateur à accéder à une ressource uniquement si celle-ci est étiquetée avec son nom IAM d'utilisateur. Pour plus d'informations, consultez IAMla section Éléments de politique : variables et balises dans le Guide de IAM l'utilisateur.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques au service. Pour voir toutes les clés de condition AWS globales, voir les clés contextuelles de condition AWS globales dans le guide de IAM l'utilisateur.

Pour consulter la liste des clés de condition Lambda, voir Clés de condition pour AWS Lambda dans la référence d'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez la section Actions définies par AWS Lambda.

Pour consulter des exemples de politiques basées sur l'identité Lambda, consultez. Exemples de politiques basées sur l'identité pour AWS Lambda

## ACLsà Lambda

Supports ACLs : Non

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLssont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format du document JSON de stratégie.

## ABACavec Lambda

Supports ABAC (balises dans les politiques) : Partiel

Le contrôle d'accès basé sur les attributs (ABAC) est une stratégie d'autorisation qui définit les autorisations en fonction des attributs. Dans AWS, ces attributs sont appelés balises. Vous pouvez

associer des balises à IAM des entités (utilisateurs ou rôles) et à de nombreuses AWS ressources. Le balisage des entités et des ressources est la première étape deABAC. Vous concevez ensuite des ABAC politiques pour autoriser les opérations lorsque le tag du principal correspond à celui de la ressource à laquelle il essaie d'accéder.

ABACest utile dans les environnements qui se développent rapidement et aide dans les situations où la gestion des politiques devient fastidieuse.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informationsABAC, voir Qu'est-ce que c'est ABAC ? dans le guide de IAM l'utilisateur. Pour consulter un didacticiel présentant les étapes de configurationABAC, voir Utiliser le contrôle d'accès basé sur les attributs (ABAC) dans le guide de l'IAMutilisateur.

Pour plus d'informations sur le balisage des ressources Lambda, consultez. Utilisation du contrôle d'accès basé sur les attributs dans Lambda

### Utilisation d'informations d'identification temporaires avec Lambda

Prend en charge les informations d'identification temporaires : oui

Certains Services AWS ne fonctionnent pas lorsque vous vous connectez à l'aide d'informations d'identification temporaires. Pour plus d'informations, y compris celles qui Services AWS fonctionnent avec des informations d'identification temporaires, consultez Services AWS la section relative à l'utilisation IAM dans le Guide de IAM l'utilisateur.

Vous utilisez des informations d'identification temporaires si vous vous connectez à l' AWS Management Console aide d'une méthode autre qu'un nom d'utilisateur et un mot de passe. Par exemple, lorsque vous accédez à AWS l'aide du lien d'authentification unique (SSO) de votre entreprise, ce processus crée automatiquement des informations d'identification temporaires. Vous créez également automatiquement des informations d'identification temporaires lorsque vous vous connectez à la console en tant qu'utilisateur, puis changez de rôle. Pour plus d'informations sur le changement de rôle, consultez la section Passage à un rôle (console) dans le guide de IAM l'utilisateur.

Vous pouvez créer manuellement des informations d'identification temporaires à l'aide du AWS CLI ou AWS API. Vous pouvez ensuite utiliser ces informations d'identification temporaires pour y accéder AWS. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez la section Informations d'identification de sécurité temporaires dans IAM.

## Transférer les sessions d'accès pour Lambda

Prend en charge les sessions d'accès transféré (FAS) : Non

Lorsque vous utilisez un IAM utilisateur ou un rôle pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FASutilise les autorisations du principal appelant an Service AWS, combinées à la demande Service AWS pour adresser des demandes aux services en aval. FASles demandes ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur les politiques relatives FAS aux demandes, consultez la section Transférer les sessions d'accès.

## Rôles de service pour Lambda

Prend en charge les rôles de service : oui

Un rôle de service est un IAMrôle qu'un service assume pour effectuer des actions en votre nom. Un IAM administrateur peut créer, modifier et supprimer un rôle de service de l'intérieurIAM. Pour plus d'informations, consultez la section Création d'un rôle auquel déléguer des autorisations Service AWS dans le Guide de IAM l'utilisateur.

Dans Lambda, un rôle de service est appelé rôle d'exécution.

> **Warning**
>
> La modification des autorisations pour un rôle d'exécution peut perturber les fonctionnalités Lambda.

## Rôles liés à un service pour Lambda

Prend en charge les rôles liés aux services : partiel

Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un IAM administrateur peut consulter, mais pas modifier les autorisations pour les rôles liés à un service.

Lambda n'a pas de rôles liés au service, contrairement à Lambda@Edge. Pour plus d'informations, consultez la section Rôles liés aux services pour Lambda @Edge dans le manuel Amazon Developer Guide. CloudFront

Pour plus de détails sur la création ou la gestion des rôles liés à un service, consultez la section AWS Services compatibles avec. IAM Recherchez un service dans le tableau qui inclut un `Yes` dans la colonne Rôle lié à un service. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

## Exemples de politiques basées sur l'identité pour AWS Lambda

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier des ressources Lambda. Ils ne peuvent pas non plus effectuer de tâches en utilisant le AWS Management Console, AWS Command Line Interface (AWS CLI) ou AWS API. Pour autoriser les utilisateurs à effectuer des actions sur les ressources dont ils ont besoin, un IAM administrateur peut créer des IAM politiques. L'administrateur peut ensuite ajouter les IAM politiques aux rôles, et les utilisateurs peuvent assumer les rôles.

Pour savoir comment créer une politique IAM basée sur l'identité à l'aide de ces exemples de documents de JSON stratégie, consultez la section Création de IAM politiques dans le guide de l'IAMutilisateur.

Pour plus de détails sur les actions et les types de ressources définis par Lambda, y compris le format de ARNs pour chacun des types de ressources, voir Actions, ressources et clés de condition AWS Lambda dans la référence d'autorisation de service.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la console Lambda
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

## Bonnes pratiques en matière de politiques

Les stratégies basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources Lambda dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez les politiques AWS gérées ou les politiques AWS gérées pour les fonctions professionnelles dans le Guide de IAM l'utilisateur.
- Appliquer les autorisations du moindre privilège : lorsque vous définissez des autorisations à IAM l'aide de politiques, accordez uniquement les autorisations nécessaires à l'exécution d'une tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation IAM pour appliquer des autorisations, consultez la section Politiques et autorisations IAM dans le guide de IAM l'utilisateur.
- Utilisez des conditions dans IAM les politiques pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques pour limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez rédiger une condition de politique pour spécifier que toutes les demandes doivent être envoyées en utilisantSSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que AWS CloudFormation. Pour plus d'informations, voir Éléments IAM JSON de politique : Condition dans le guide de IAM l'utilisateur.
- Utilisez IAM Access Analyzer pour valider vos IAM politiques afin de garantir des autorisations sécurisées et fonctionnelles. IAM Access Analyzer valide les politiques nouvelles et existantes afin qu'elles soient conformes au langage des IAM politiques (JSON) et IAM aux meilleures pratiques. IAMAccess Analyzer fournit plus de 100 vérifications des politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez la section Validation des politiques d'IAMAccess Analyzer dans le guide de IAM l'utilisateur.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des IAM utilisateurs ou un utilisateur root Compte AWS, activez-le MFA pour une sécurité supplémentaire.

Pour exiger le MFA moment où les API opérations sont appelées, ajoutez MFA des conditions à vos politiques. Pour plus d'informations, consultez la section Configuration de l'APIaccès MFA protégé dans le Guide de l'IAMutilisateur.

Pour plus d'informations sur les meilleures pratiques en matière de sécuritéIAM, consultez la section Bonnes pratiques en matière de sécurité IAM dans le Guide de IAM l'utilisateur.

## Utilisation de la console Lambda

Pour accéder à la AWS Lambda console, vous devez disposer d'un ensemble minimal d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher des informations détaillées sur les ressources Lambda de votre. Compte AWS Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement le AWS CLI ou le AWS API. Au lieu de cela, autorisez uniquement l'accès aux actions correspondant à l'APIopération qu'ils tentent d'effectuer.

Pour obtenir un exemple de stratégie qui accorde un accès minimal pour le développement de fonctions, veuillez consulter Accorder aux utilisateurs l'accès à une fonction Lambda. Outre LambdaAPIs, la console Lambda utilise d'autres services pour afficher la configuration des déclencheurs et vous permettre d'ajouter de nouveaux déclencheurs. Si vos utilisateurs utilisent Lambda avec d'autres services, ils ont également besoin d'accéder à ces services. Pour plus d'informations sur la configuration d'autres services avec Lambda, consultez Invoquer Lambda avec des événements provenant d'autres services AWS.

## Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux IAM utilisateurs de consulter les politiques intégrées et gérées associées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide du AWS CLI ou. AWS API

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

## Politiques gérées par AWS pour AWS Lambda

Une politique gérée par AWS est une politique autonome créée et administrée par AWS. Les politiques gérées par AWS sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

Gardez à l'esprit que les politiques gérées par AWS peuvent ne pas accorder les autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont disponibles pour tous les clients AWS. Nous vous recommandons de réduire encore les autorisations en définissant des politiques gérées par le client qui sont spécifiques à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les politiques gérées par AWS. Si AWS met à jour les autorisations définies dans une politique gérée par AWS, la mise à jour affecte toutes les identités de principal (utilisateurs, groupes et rôles) auxquelles la politique est associée. AWS est plus susceptible de mettre à jour une politique gérée par AWS lorsqu'un nouveau Service AWS est lancé ou que de nouvelles opérations API deviennent accessibles pour les services existants.

Pour plus d'informations, consultez Politiques gérées par AWS dans le Guide de l'utilisateur IAM.

Rubriques

- AWSpolitique gérée : AWSLambda_FullAccess
- AWSpolitique gérée : AWSLambda_ReadOnlyAccess
- AWSpolitique gérée : AWSLambdaBasicExecutionRole
- AWSpolitique gérée : AWSLambdaDynamoDBExecutionRole
- AWSpolitique gérée : AWSLambdaENIManagementAccess
- AWSpolitique gérée : AWSLambdaExecute
- AWSpolitique gérée : AWSLambdaInvocation -DynamoDB
- AWSpolitique gérée : AWSLambdaKinesisExecutionRole
- AWSpolitique gérée : AWSLambdaMSKExecutionRole
- AWSpolitique gérée : AWSLambdaRole
- AWSpolitique gérée : AWSLambdaSQSQueueExecutionRole
- AWSpolitique gérée : AWSLambdaVPCAccessExecutionRole
- Mises à jour de Lambda concernant les stratégies gérées par AWS

## AWSpolitique gérée : AWSLambda_FullAccess

Cette stratégie accorde un accès complet aux actions Lambda. Elle accorde également des autorisations à d'autres services AWS utilisés pour développer et gérer les ressources Lambda.

Vous pouvez attacher la stratégie `AWSLambda_FullAccess` à vos utilisateurs, groupes et rôles.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `lambda` : donne aux principaux un accès complet à Lambda.

- `cloudformation` : permet aux principaux de décrire les piles AWS CloudFormation et de répertorier les ressources qu'elles contiennent.
- `cloudwatch`— Permet aux principaux de répertorier les CloudWatch métriques Amazon et d'obtenir des données métriques.
- `ec2` : permet aux principaux de décrire les groupes de sécurité, les sous-réseaux et les VPC.
- `iam` : permet aux principaux d'obtenir les stratégies, les versions des stratégies, les rôles, les stratégies de rôle, les stratégies de rôles associées et la liste des rôles. Cette stratégie permet également aux principaux de transmettre des rôles à Lambda. L'autorisation `PassRole` est utilisée lorsque vous attribuez un rôle d'exécution à une fonction.
- `kms` : permet aux principaux de répertorier les alias.
- `logs`— Permet aux principaux de décrire les groupes de CloudWatch journaux Amazon. Pour les groupes de journaux associés à une fonction Lambda, cette stratégie permet au principal de décrire les flux de journaux, d'obtenir les événements du journal et de filtrer les événements du journal.
- `states` : permet aux principaux de décrire et de répertorier machines d'état AWS Step Functions.
- `tag` : permet aux principaux d'obtenir des ressources en fonction de leurs balises.
- `xray` : permet aux principaux d'obtenir des résumés de traces AWS X-Ray et de récupérer une liste de traces spécifiées par ID.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambda_FullAccessle Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambda_ReadOnlyAccess

Cette stratégie accorde un accès en lecture seule aux ressources Lambda et à d'autres services AWS utilisés pour développer et gérer les ressources Lambda.

Vous pouvez attacher la stratégie `AWSLambda_ReadOnlyAccess` à vos utilisateurs, groupes et rôles.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `lambda` : permet aux principaux d'obtenir et de répertorier toutes les ressources.
- `cloudformation` : permet aux principaux de décrire et de répertorier les piles AWS CloudFormation et de répertorier les ressources qu'elles contiennent.

- `cloudwatch`— Permet aux principaux de répertorier les CloudWatch métriques Amazon et d'obtenir des données métriques.
- `ec2` : permet aux principaux de décrire les groupes de sécurité, les sous-réseaux et les VPC.
- `iam` : permet aux principaux d'obtenir les stratégies, les versions des stratégies, les rôles, les stratégies de rôle, les stratégies de rôles associées et la liste des rôles.
- `kms` : permet aux principaux de répertorier les alias.
- `logs`— Permet aux principaux de décrire les groupes de CloudWatch journaux Amazon. Pour les groupes de journaux associés à une fonction Lambda, cette stratégie permet au principal de décrire les flux de journaux, d'obtenir les événements du journal et de filtrer les événements du journal.
- `states` : permet aux principaux de décrire et de répertorier machines d'état AWS Step Functions.
- `tag` : permet aux principaux d'obtenir des ressources en fonction de leurs balises.
- `xray` : permet aux principaux d'obtenir des résumés de traces AWS X-Ray et de récupérer une liste de traces spécifiées par ID.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambda_ReadOnlyAccesslle Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaBasicExecutionRole

Cette politique accorde des autorisations pour télécharger des CloudWatch journaux dans Logs.

Vous pouvez attacher la stratégie `AWSLambdaBasicExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaBasicExecutionRolele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaDynamoDBExecutionRole

Cette politique accorde des autorisations pour lire les enregistrements d'un flux Amazon DynamoDB et pour écrire dans Logs. CloudWatch

Vous pouvez attacher la stratégie `AWSLambdaDynamoDBExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaDynamoDBExecutionRolele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaENIManagementAccess

Cette stratégie accorde les autorisations nécessaires pour créer, décrire et supprimer des interfaces réseau Elastic utilisées par une fonction Lambda compatible VPC.

Vous pouvez attacher la stratégie `AWSLambdaENIManagementAccess` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaENIManagementAccessle Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaExecute

Cette politique accorde `PUT` `GET` un accès à Amazon Simple Storage Service et un accès complet aux CloudWatch journaux.

Vous pouvez attacher la stratégie `AWSLambdaExecute` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaExecutele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaInvocation -DynamoDB

Cette stratégie accorde un accès en lecture à Amazon DynamoDB Streams.

Vous pouvez attacher la stratégie `AWSLambdaInvocation-DynamoDB` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, voir AWSLambdaInvocation-DynamoDB dans le Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaKinesisExecutionRole

Cette politique accorde l'autorisation de lire les événements d'un flux de données Amazon Kinesis et d'écrire dans Logs. CloudWatch

Vous pouvez attacher la stratégie `AWSLambdaKinesisExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaKinesisExecutionRolele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaMSKExecutionRole

Cette politique autorise la lecture et l'accès aux enregistrements d'un cluster Amazon Managed Streaming for Apache Kafka, la gestion des interfaces réseau élastiques et l'écriture dans les CloudWatch journaux.

Vous pouvez attacher la stratégie `AWSLambdaMSKExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaMSKExecutionRolele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaRole

Cette stratégie accorde des autorisations pour invoquer les fonctions Lambda.

Vous pouvez attacher la stratégie `AWSLambdaRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaRolele Guide de référence des politiques AWS gérées.

## AWSpolitique gérée : AWSLambdaSQSQueueExecutionRole

Cette politique accorde des autorisations pour lire et supprimer des messages d'une file d'attente Amazon Simple Queue Service, et accorde des autorisations d'écriture aux CloudWatch journaux.

Vous pouvez attacher la stratégie `AWSLambdaSQSQueueExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaSQSQueueExecutionRolele Guide de référence des politiques AWS gérées.

### AWSpolitique gérée : AWSLambdaVPCAccessExecutionRole

Cette politique accorde des autorisations pour gérer des interfaces réseau élastiques au sein d'un Amazon Virtual Private Cloud et pour écrire dans CloudWatch Logs.

Vous pouvez attacher la stratégie `AWSLambdaVPCAccessExecutionRole` à vos utilisateurs, groupes et rôles.

Pour plus d'informations sur cette politique, y compris le document de politique JSON et les versions de politique, consultez AWSLambdaVPCAccessExecutionRolele Guide de référence des politiques AWS gérées.

### Mises à jour de Lambda concernant les stratégies gérées par AWS

| Modification | Description | Date |
| --- | --- | --- |
| AWSLambdaVPCAccess ExecutionRole— Modification | Lambda a mis à jour la `AWSLambdaVPCAccess ExecutionRole` politique pour autoriser l'action. `ec2:DescribeSubnets` | 5 janvier 2024 |
| AWSLambda_ReadOnly Access— Modification | Lambda a mis à jour la stratégie `AWSLambda _ReadOnlyAccess` pour permettre aux principaux de répertorier les piles AWS CloudFormation. | 27 juillet 2023 |
| AWS Lambda a démarré le suivi des modifications | AWS Lambda a commencé à suivre les modifications pour ses politiques gérées par AWS. | 27 juillet 2023 |

## Résolution des problèmes AWS Lambda d'identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec Lambda et. IAM

Rubriques


- Je ne suis pas autorisé à effectuer une action dans Lambda
- Je ne suis pas autorisé à effectuer iam : PassRole
- Je souhaite permettre à des personnes extérieures à moi d'accéder Compte AWS à mes ressources Lambda


## Je ne suis pas autorisé à effectuer une action dans Lambda

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit lorsque l'`mateojackson`IAMutilisateur essaie d'utiliser la console pour afficher les détails d'une *`my-example-widget`* ressource fictive mais ne dispose pas des `lambda:`*`GetWidget`* autorisations fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lambda:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `lambda:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

## Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter l'action `iam:PassRole`, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle à Lambda.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un IAM utilisateur nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans Lambda. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

## Je souhaite permettre à des personnes extérieures à moi d'accéder Compte AWS à mes ressources Lambda

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour en savoir plus, consultez les éléments suivants :

- Pour savoir si Lambda prend en charge ces fonctions, consultez Comment AWS Lambda fonctionne avec IAM.
- Pour savoir comment donner accès à vos ressources sur un site Comptes AWS qui vous appartient, consultez la section Fournir l'accès à un IAM utilisateur dans un autre site Compte AWS que vous possédez dans le Guide de IAM l'utilisateur.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le Guide de IAM l'utilisateur.
- Pour savoir comment fournir un accès via la fédération d'identité, consultez la section Fournir un accès aux utilisateurs authentifiés de manière externe (fédération d'identité) dans le guide de l'IAMutilisateur.
- Pour connaître la différence entre l'utilisation de rôles et l'utilisation de politiques basées sur les ressources pour l'accès entre comptes, voir Accès aux ressources entre comptes IAM dans le guide de l'IAMutilisateur.

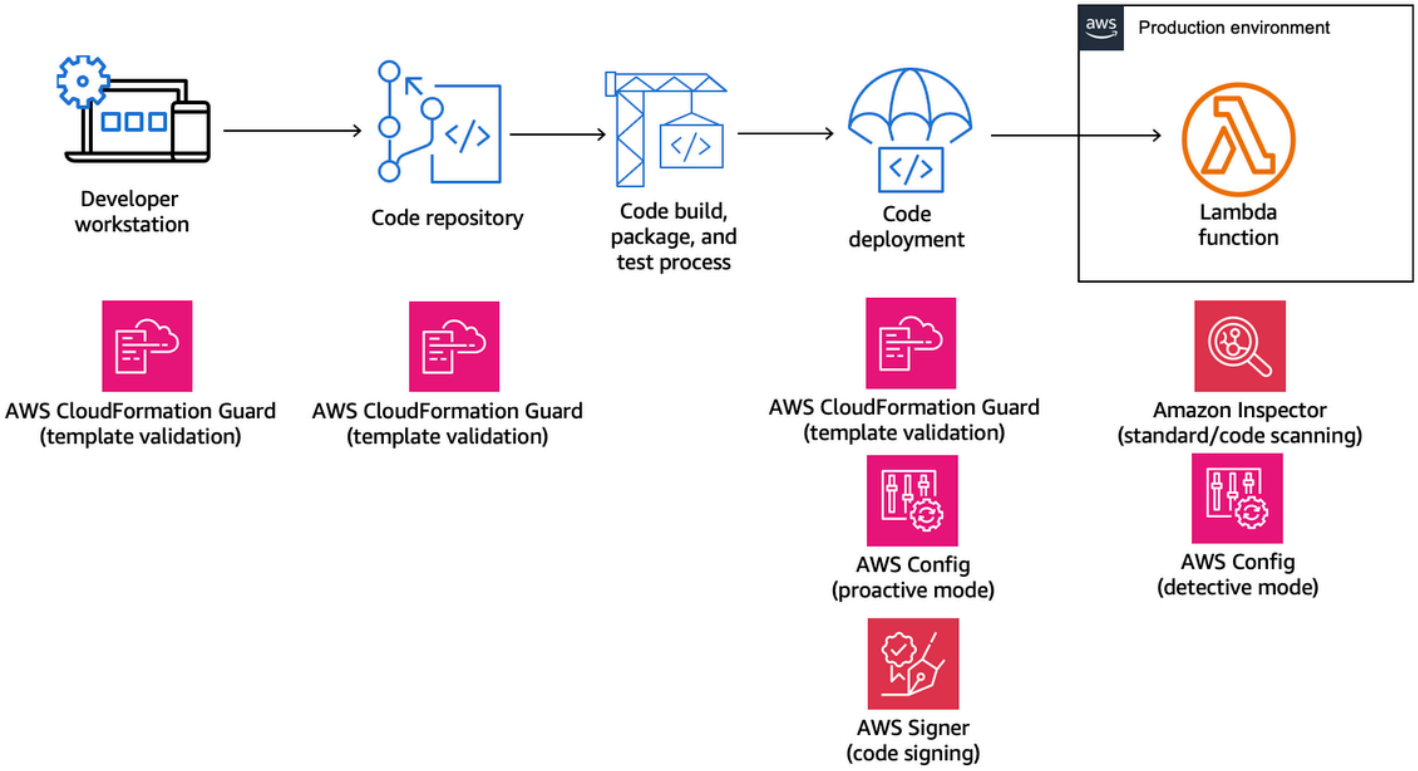
# Création d'une stratégie de gouvernance pour les fonctions et les couches Lambda

Pour créer et déployer des applications cloud natives sans serveur, vous devez garantir agilité et rapidité de mise sur le marché grâce à une gouvernance et à des barrières de protection appropriés. Vous définissez des priorités au niveau de l'entreprise, en mettant peut-être l'accent sur l'agilité comme priorité absolue, ou en insistant sur l'aversion au risque par le biais de la gouvernance, de barrières de protection et de contrôles. En réalité, vous n'aurez pas une stratégie « l'un ou l'autre », mais une stratégie « et » qui équilibre à la fois agilité et barrières de protection dans le cycle de vie de votre développement logiciel. Quelle que soit la place de ces exigences dans le cycle de vie de votre entreprise, les capacités de gouvernance sont susceptibles de devenir une exigence de mise en œuvre dans vos processus et chaînes d'outils.

Voici quelques exemples de contrôles de gouvernance qu'une organisation peut mettre en œuvre pour Lambda :

- Les fonctions Lambda ne doivent pas être publiquement accessibles.
- Les fonctions Lambda doivent être associées à un VPC.
- Les fonctions Lambda ne doivent pas utiliser d'environnements d'exécution obsolètes.
- Les fonctions Lambda doivent être étiquetées avec un ensemble de balises obligatoires.
- Les couches Lambda ne doivent pas être accessibles en dehors de l'organisation.
- Les fonctions Lambda associées à un groupe de sécurité doivent avoir des balises correspondantes entre la fonction et le groupe de sécurité.
- Les fonctions Lambda associées à une couche doivent utiliser une version approuvée
- Les variables d'environnement Lambda doivent être chiffrées au repos avec une clé gérée par le client.

Le schéma suivant est un exemple de stratégie de gouvernance approfondie qui met en œuvre des contrôles et des politiques tout au long du processus de développement et de déploiement du logiciel :




Les rubriques suivantes expliquent comment implémenter des contrôles pour développer et déployer des fonctions Lambda dans votre organisation, à la fois pour la start-up et pour l'entreprise. Votre organisation possède peut-être déjà des outils en place. Les rubriques suivantes adoptent une approche modulaire de ces contrôles, afin que vous puissiez sélectionner les composants dont vous avez réellement besoin.

Rubriques

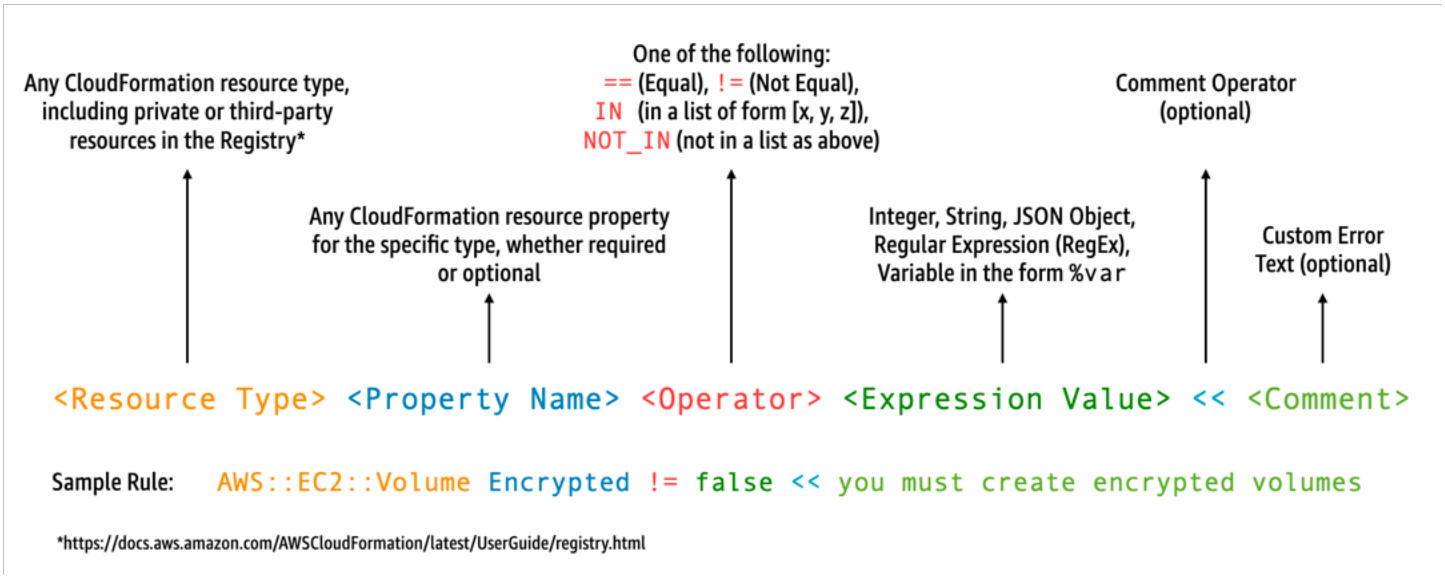
- Contrôles proactifs pour Lambda avec AWS CloudFormation Guard
- Mettez en œuvre des contrôles préventifs pour Lambda avec AWS Config
- Détectez les déploiements et les configurations Lambda non conformes avec AWS Config
- Signature de code Lambda avec AWS Signer
- Automatisez les évaluations de sécurité pour Lambda avec Amazon Inspector
- Mettre en œuvre l'observabilité pour la sécurité et la conformité Lambda

# Contrôles proactifs pour Lambda avec AWS CloudFormation Guard

AWS CloudFormation Guardest un outil d'évaluation open source à usage général. policy-as-code Cela peut être utilisé à des fins de gouvernance préventive et de conformité en validant les modèles d'infrastructure en tant que code (IaC) et les compositions de services par rapport aux règles de politique. Ces règles peuvent être personnalisées en fonction des exigences de votre équipe ou de votre organisation. Pour les fonctions Lambda, les règles Guard peuvent être utilisées pour contrôler la création de ressources et les mises à jour de configuration en définissant les paramètres de propriété requis lors de la création ou de la mise à jour d'une fonction Lambda.

Les administrateurs de conformité définissent la liste des contrôles et des politiques de gouvernance nécessaires au déploiement et à la mise à jour des fonctions Lambda. Les administrateurs de plateforme mettent en œuvre les contrôles dans les pipelines CI/CD, sous forme de webhooks de validation préalable à la validation avec des référentiels de code, et fournissent aux développeurs des outils en ligne de commande pour valider les modèles et le code sur les postes de travail locaux. Les développeurs créent du code, valident les modèles à l'aide d'outils de ligne de commande, puis valident le code dans des référentiels, qui sont ensuite automatiquement validés via les pipelines CI/CD avant le déploiement dans un environnement. AWS

Guard vous permet d'écrire vos règles et d'implémenter vos contrôles dans un langage spécifique au domaine comme suit.




Supposons, par exemple, que vous souhaitiez vous assurer que les développeurs choisissent uniquement les derniers environnements d'exécution. Vous pouvez spécifier deux politiques différentes, l'une pour identifier les environnements d'exécution déjà obsolètes et l'autre pour

identifier les environnements d'exécution qui le seront bientôt. Pour cela, vous pouvez écrire le fichier `etc/rules.guard` suivant :

```
let lambda_functions = Resources.*[
    Type == "AWS::Lambda::Function"
]

rule lambda_already_deprecated_runtime when %lambda_functions !empty {
    %lambda_functions {
        Properties {
            when Runtime exists {
                Runtime !in ["dotnetcore3.1", "nodejs12.x", "python3.6", "python2.7",
 "dotnet5.0", "dotnetcore2.1", "ruby2.5", "nodejs10.x", "nodejs8.10", "nodejs4.3",
 "nodejs6.10", "dotnetcore1.0", "dotnetcore2.0", "nodejs4.3-edge", "nodejs"] <<Lambda
 function is using a deprecated runtime.>>
            }
        }
    }
}

rule lambda_soon_to_be_deprecated_runtime when %lambda_functions !empty {
    %lambda_functions {
        Properties {
            when Runtime exists {
                Runtime !in ["nodejs16.x", "nodejs14.x", "python3.7", "java8",
 "dotnet7", "go1.x", "ruby2.7", "provided"] <<Lambda function is using a runtime that
 is targeted for deprecation.>>
            }
        }
    }
}
```

Supposons maintenant que vous écriviez le `iac/lambda.yaml` CloudFormation modèle suivant qui définit une fonction Lambda :

```
  Fn:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      CodeUri: src
      Handler: fn.handler
      Role: !GetAtt FnRole.Arn
      Layers:
```

```
        - arn:aws:lambda:us-east-1:111122223333:layer:LambdaInsightsExtension:35
```

Après avoir installé l'utilitaire Guard, validez votre modèle :

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

Le résultat se présente comme suit :

```
lambda.yaml Status = FAIL
FAILED rules
rules.guard/lambda_soon_to_be_deprecated_runtime
---
Evaluating data lambda.yaml against rules rules.guard
Number of non-compliant resources 1
Resource = Fn {
  Type      = AWS::Lambda::Function
  Rule = lambda_soon_to_be_deprecated_runtime {
    ALL {
      Check =  Runtime not IN
 ["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]
 {
       ComparisonError {
         Message          = Lambda function is using a runtime that is targeted for
 deprecation.
         Error            = Check was not compliant as property [/Resources/
Fn/Properties/Runtime[L:88,C:15]] was not present in [(resolved, Path=[L:0,C:0]
 Value=["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"])]
       }
         PropertyPath    = /Resources/Fn/Properties/Runtime[L:88,C:15]
         Operator        = NOT IN
         Value           = "python3.7"
         ComparedWith    =
 [["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]]
         Code:
              86.  Fn:
              87.    Type: AWS::Lambda::Function
              88.    Properties:
              89.      Runtime: python3.7
              90.      CodeUri: src
              91.      Handler: fn.handler

      }
    }
```

```
  }
}
```

Guard permet à vos développeurs de savoir depuis leur poste de travail local qu'ils doivent mettre à jour le modèle afin d'utiliser un environnement d'exécution autorisé par l'organisation. Cela se produit avant la validation dans un référentiel de code et avant l'échec des vérifications dans un pipeline CI/CD. Vos développeurs reçoivent ainsi des informations sur la manière de développer des modèles conformes et de consacrer leur temps à l'écriture de code apportant une valeur commerciale.
Ce contrôle peut être appliqué sur le poste de travail du développeur local, dans un webhook de validation préalable à la validation et/ou dans le pipeline CI/CD avant le déploiement.

## Mises en garde

Si vous utilisez des modèles AWS Serverless Application Model (AWS SAM) pour définir des fonctions Lambda, sachez que vous devez mettre à jour la règle Guard pour rechercher le type de `AWS::Serverless::Function` ressource comme suit.

```
let lambda_functions = Resources.*[
    Type == "AWS::Serverless::Function"
]
```

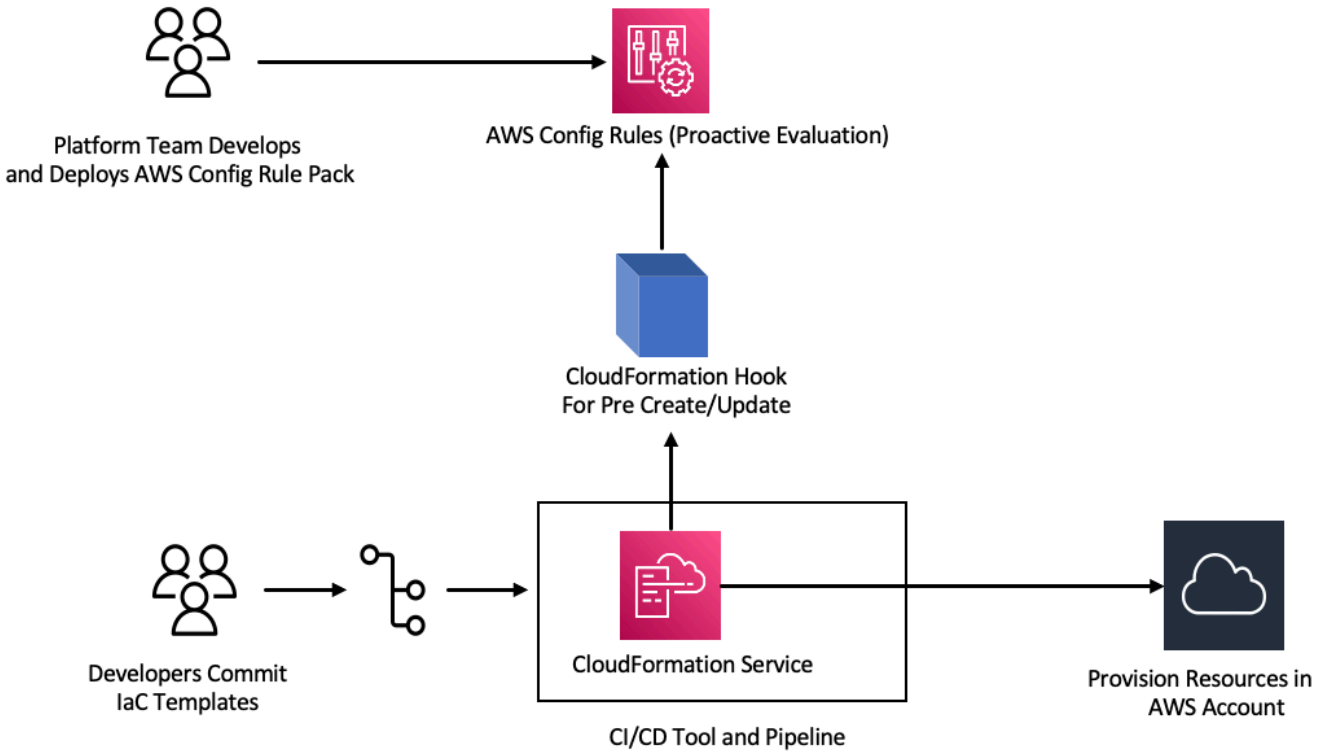
Guard s'attend également à ce que les propriétés soient incluses dans la définition de la ressource. Dans le même temps, les AWS SAM modèles permettent de spécifier les propriétés dans une section Globals distincte. Les propriétés définies dans la section Globals ne sont pas validées par vos règles Guard.

Comme indiqué dans la documentation de dépannage de Guard, sachez que Guard ne prend pas en charge les formulaires intrinsèques abrégés tels que `!GetAtt` ou `!Sub` et nécessite plutôt l'utilisation des formulaires étendus : `Fn::GetAtt` et. `Fn::Sub` (L'exemple précédent n'évaluant pas la propriété Role, la forme abrégée intrinsèque a été utilisée pour des raisons de simplicité.)

# Mettez en œuvre des contrôles préventifs pour Lambda avec AWS Config

Il est essentiel de garantir la conformité de vos applications sans serveur le plus tôt possible dans le processus de développement. Dans cette rubrique, nous expliquons comment mettre en œuvre des contrôles préventifs à l'aide de AWS Config. Cela vous permet de mettre en œuvre des contrôles de conformité plus tôt dans le processus de développement et de mettre en œuvre les mêmes contrôles dans vos pipelines CI/CD. Cela permet également de standardiser vos contrôles dans un référentiel de règles géré de manière centralisée afin que vous puissiez appliquer vos contrôles de manière cohérente sur l'ensemble de vos AWS comptes.

Supposons, par exemple, que vos administrateurs de conformité aient défini une exigence garantissant que toutes les fonctions Lambda incluent AWS X-Ray le traçage. Grâce AWS Config au mode proactif, vous pouvez effectuer des contrôles de conformité sur les ressources de vos fonctions Lambda avant le déploiement, ce qui réduit le risque de déploiement de fonctions Lambda mal configurées et fait gagner du temps aux développeurs en leur fournissant plus rapidement des informations sur l'infrastructure sous forme de modèles de code. Voici une visualisation du flux pour les contrôles préventifs avec AWS Config :



Envisagez d'exiger que le suivi soit activé pour toutes les fonctions Lambda. En réponse, l'équipe de la plateforme identifie la nécessité d'une AWS Config règle spécifique pour s'exécuter de manière proactive sur tous les comptes. Cette règle signale toute fonction Lambda dont la configuration de suivi X-Ray n'est pas configurée comme une ressource non conforme. L'équipe développe une règle, l'intègre dans un pack de conformité et déploie le pack de conformité sur tous les comptes afin de garantir que tous les AWS comptes de l'organisation appliquent ces contrôles de manière uniforme. Vous pouvez écrire la règle dans la syntaxe AWS CloudFormation Guard 2.x.x, qui prend la forme suivante :

```
rule name when condition { assertion }
```

Voici un exemple de règle Guard qui vérifie que le suivi est activé pour les fonctions Lambda :

```
rule lambda_tracing_check {
  when configuration.tracingConfig exists {
      configuration.tracingConfig.mode == "Active"
  }
}
```

L'équipe de la plateforme prend des mesures supplémentaires en imposant que chaque AWS CloudFormation déploiement invoque un hook de pré-création/mise à jour. Elle assume l'entière responsabilité du développement de ce hook et de la configuration du pipeline, du renforcement du contrôle centralisé des règles de conformité et du maintien de leur application cohérente dans tous les déploiements. Pour développer, empaqueter et enregistrer un hook, voir Developing AWS CloudFormation Hooks dans la documentation de l'interface de ligne de CloudFormation commande (CFN-CLI). Vous pouvez utiliser le CloudFormation CLIpour créer le projet de crochet :

```
cfn init
```

Cette commande vous demande des informations de base sur votre projet hook et crée un projet contenant les fichiers suivants :

```
README.md
<hook-name>.json
rpdk.log
src/handler.py
template.yml
hook-role.yaml
```

En tant que développeur de hooks, vous devez ajouter le type de ressource cible souhaité dans le fichier `<hook-name>.json` de configuration. Dans la configuration ci-dessous, un hook est configuré pour s'exécuter avant qu'une fonction Lambda ne soit créée à l'aide de. CloudFormation Vous pouvez également ajouter des gestionnaires `preUpdate` et des actions `preDelete` similaires.

```
    "handlers": {
        "preCreate": {
            "targetNames": [
                "AWS::Lambda::Function"
            ],
            "permissions": []
        }
    }
```

Vous devez également vous assurer que le CloudFormation hook dispose des autorisations appropriées pour appeler le AWS Config APIs. Vous pouvez le faire en mettant à jour le fichier de définition de rôle nommé `hook-role.yaml`. Le fichier de définition du rôle possède par défaut la politique de confiance suivante, qui permet CloudFormation d'assumer le rôle.

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - hooks.cloudformation.amazonaws.com
                - resources.cloudformation.amazonaws.com
```

Pour permettre à ce hook d'appeler configAPIs, vous devez ajouter les autorisations suivantes à la déclaration Policy. Ensuite, vous soumettez le projet hook à l'aide de la `cfn submit` commande, où vous CloudFormation créez un rôle avec les autorisations requises.

```
      Policies:
        - PolicyName: HookTypePolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - "config:Describe*"
                  - "config:Get*"
```

```
                - "config:List*"
                - "config:SelectResourceConfig"
              Resource: "*
```

Ensuite, vous devez écrire une fonction Lambda dans un `src/handler.py` fichier. Dans ce fichier, vous trouverez les méthodes nommées `preCreate`, `preUpdate` et `preDelete` déjà créées lorsque vous avez lancé le projet. Votre objectif est d'écrire une fonction commune et réutilisable qui appelle le AWS Config `StartResourceEvaluation` API en mode proactif à l'aide du AWS SDK for Python (Boto3). Cet API appel prend les propriétés de la ressource en entrée et évalue la ressource par rapport à la définition de la règle.

```
def validate_lambda_tracing_config(resource_type, function_properties:
 MutableMapping[str, Any]) -> ProgressEvent:
  LOG.info("Fetching proactive data")
  config_client = boto3.client('config')
  resource_specs = {
      'ResourceId': 'MyFunction',
      'ResourceType': resource_type,
      'ResourceConfiguration': json.dumps(function_properties),
      'ResourceConfigurationSchemaType': 'CFN_RESOURCE_SCHEMA'
  }
  LOG.info("Resource Specifications:", resource_specs)
  eval_response = config_client.start_resource_evaluation(EvaluationMode='PROACTIVE',
 ResourceDetails=resource_specs, EvaluationTimeout=60)
  ResourceEvaluationId = eval_response.ResourceEvaluationId
  compliance_response =
 config_client.get_compliance_details_by_resource(ResourceEvaluationId=ResourceEvaluationId)
  LOG.info("Compliance Verification:",
 compliance_response.EvaluationResults[0].ComplianceType)
  if "NON_COMPLIANT" == compliance_response.EvaluationResults[0].ComplianceType:
      return ProgressEvent(status=OperationStatus.FAILED, message="Lambda function
 found with no tracing enabled : FAILED", errorCode=HandlerErrorCode.NonCompliant)
  else:
      return ProgressEvent(status=OperationStatus.SUCCESS, message="Lambda function
 found with tracing enabled : PASS.")
```

Vous pouvez maintenant appeler la fonction commune depuis le gestionnaire pour le hook de pré-création. Voici un exemple de gestionnaire :

```
@hook.handler(HookInvocationPoint.CREATE_PRE_PROVISION)
def pre_create_handler(
        session: Optional[SessionProxy],
```

```
        request: HookHandlerRequest,
        callback_context: MutableMapping[str, Any],
        type_configuration: TypeConfigurationModel
) -> ProgressEvent:
    LOG.info("Starting execution of the hook")
    target_name = request.hookContext.targetName
    LOG.info("Target Name:", target_name)
    if "AWS::Lambda::Function" == target_name:
        return validate_lambda_tracing_config(target_name,
            request.hookContext.targetModel.get("resourceProperties")
        )
    else:
        raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")
```

Après cette étape, vous pouvez enregistrer le hook et le configurer pour écouter tous les événements de création de AWS Lambda fonctions.

Un développeur prépare le modèle d'infrastructure en tant que code (IaC) pour un microservice sans serveur à l'aide de Lambda. Cette préparation inclut le respect des normes internes, suivi de tests locaux et de validation du modèle dans le référentiel. Voici un exemple de modèle IaC :

```
  MyLambdaFunction:
  Type: 'AWS::Lambda::Function'
  Properties:
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    FunctionName: MyLambdaFunction
    Code:
      ZipFile: |
        import json

        def handler(event, context):
            return {
                'statusCode': 200,
                'body': json.dumps('Hello World!')
            }
    Runtime: python3.8
    TracingConfig:
        Mode: PassThrough
    MemorySize: 256
    Timeout: 10
```

Dans le cadre du processus CI/CD, lorsque le CloudFormation modèle est déployé, le CloudFormation service invoque le hook de pré-création/mise à jour juste avant le provisionnement du type de ressource. `AWS::Lambda::Function` Le hook utilise des AWS Config règles exécutées en mode proactif pour vérifier que la configuration de la fonction Lambda inclut la configuration de suivi obligatoire. La réponse du hook détermine l'étape suivante. S'il est conforme, le hook signale le succès et CloudFormation procède à la mise à disposition des ressources. Dans le cas contraire, le déploiement de la CloudFormation pile échoue, le pipeline s'arrête immédiatement et le système enregistre les détails pour un examen ultérieur. Les notifications de conformité sont envoyées vers les parties prenantes concernées.

Vous pouvez trouver les informations relatives à la réussite ou à l'échec du hook dans la CloudFormation console :

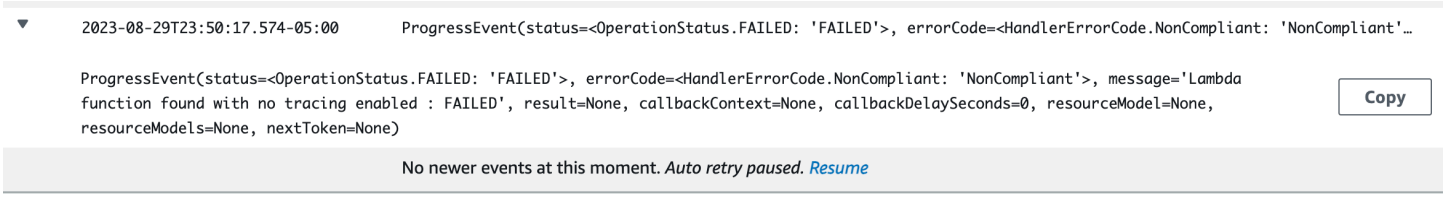
Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

**Events** (19)

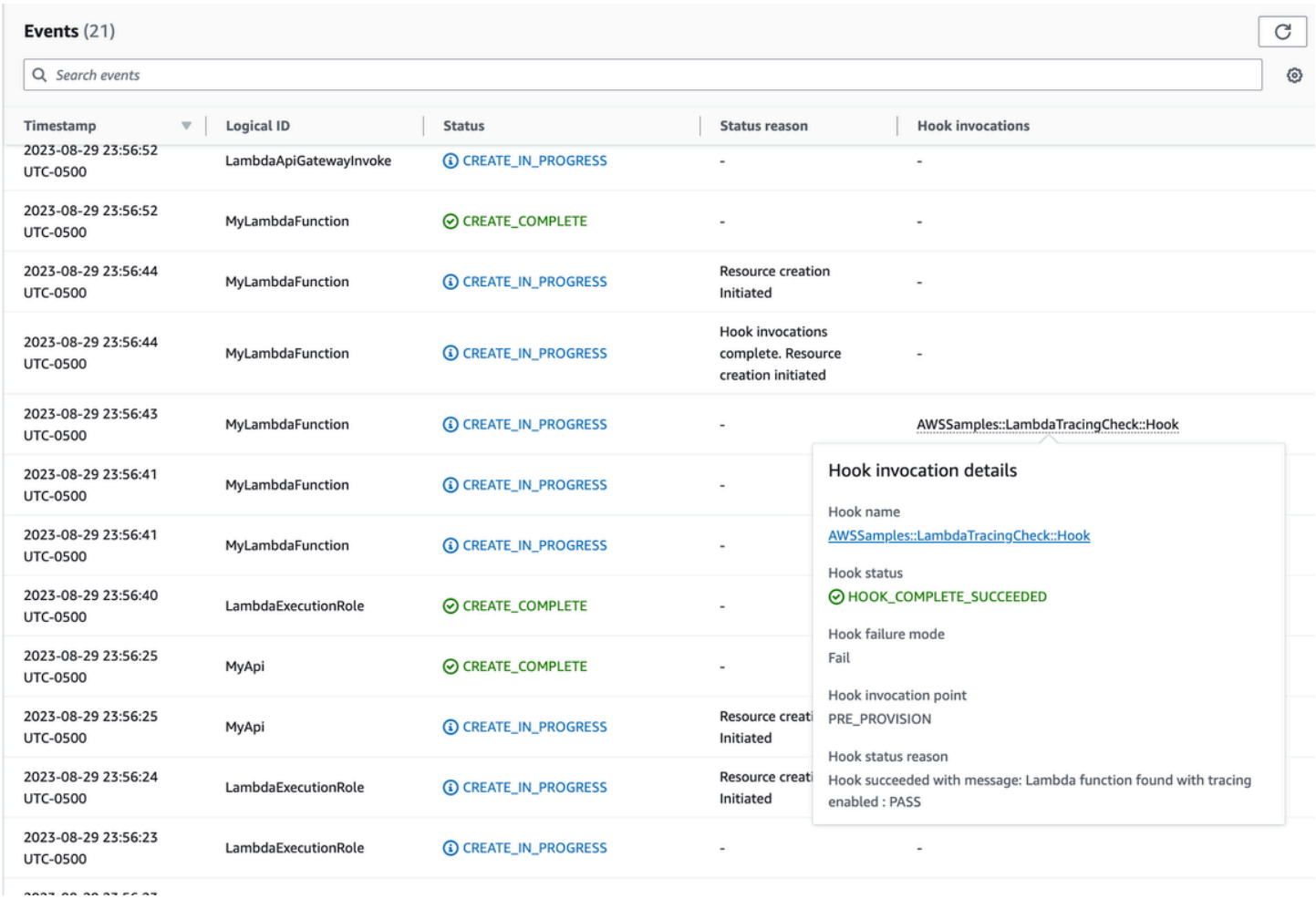
Search events

| Timestamp | Logical ID | Status | Status reason | Hook invocations |
|---|---|---|---|---|
| 2023-08-29 23:50:23 UTC-0500 | HookTestStack | ROLLBACK_COMPLETE | - | - |
| 2023-08-29 23:50:22 UTC-0500 | LambdaExecutionRole | DELETE_COMPLETE | - | - |
| 2023-08-29 23:50:21 UTC-0500 | MyApi | DELETE_COMPLETE | - | - |
| 2023-08-29 23:50:20 UTC-0500 | LambdaExecutionRole | DELETE_IN_PROGRESS | - | - |
| 2023-08-29 23:50:20 UTC-0500 | MyLambdaFunction | DELETE_COMPLETE | - | - |
| 2023-08-29 23:50:20 UTC-0500 | MyApi | DELETE_IN_PROGRESS | - | - |
| 2023-08-29 23:50:18 UTC-0500 | HookTestStack | ROLLBACK_IN_PROGRESS | The following resource(s) failed to create: [MyLambdaFunction]. Rollback requested by user. | - |
| 2023-08-29 23:50:17 UTC-0500 | MyLambdaFunction | CREATE_FAILED | The following hook(s) failed: [AWSSamples::LambdaTracingCheck::Hook] | - |
| 2023-08-29 23:50:17 UTC-0500 | MyLambdaFunction | CREATE_IN_PROGRESS | - | AWSSamples::LambdaTracingCheck::Hook |
| 2023-08-29 23:50:16 UTC-0500 | MyLambdaFunction | CREATE_IN_PROGRESS | - | AWSSamples::LambdaTracingCheck::Hook |
| 2023-08-29 23:50:15 UTC-0500 | MyLambdaFunction | CREATE_IN_PROGRESS | - | - |
| 2023-08-29 23:50:14 UTC-0500 | LambdaExecutionRole | CREATE_COMPLETE | - | - |
| 2023-08-29 23:49:59 UTC-0500 | MyApi | CREATE_COMPLETE | - | - |
| 2023-08-29 23:49:59 UTC-0500 | MyApi | CREATE_IN_PROGRESS | Resource creation Initiated | - |
| 2023-08-29 23:49:58 UTC-0500 | LambdaExecutionRole | CREATE_IN_PROGRESS | Resource creation Initiated | - |
| 2023-08-29 23:49:58 UTC-0500 | LambdaExecutionRole | CREATE_IN_PROGRESS | - | - |
| 2023-08-29 23:49:58 UTC-0500 | MyApi | CREATE_IN_PROGRESS | - | - |
| 2023-08-29 23:49:55 UTC-0500 | HookTestStack | CREATE_IN_PROGRESS | User Initiated | - |
| 2023-08-29 23:49:50 UTC-0500 | HookTestStack | REVIEW_IN_PROGRESS | User Initiated | - |

Si les journaux sont activés pour votre CloudFormation hook, vous pouvez capturer le résultat de l'évaluation du hook. Voici un exemple de journal pour un hook dont le statut a échoué, indiquant que X-Ray n'est pas activé sur la fonction Lambda :




Si le développeur choisit de modifier le modèle IaC pour mettre à jour la valeur `TracingConfig Mode`sur `Active` et le redéployer, le hook s'exécute correctement et la pile poursuit la création de la ressource Lambda.




Ainsi, vous pouvez mettre en œuvre des contrôles préventifs en mode proactif lors du développement et du déploiement de ressources sans serveur dans vos AWS comptes. AWS Config En intégrant des règles AWS Config dans le pipeline CI/CD, vous pouvez identifier et éventuellement bloquer

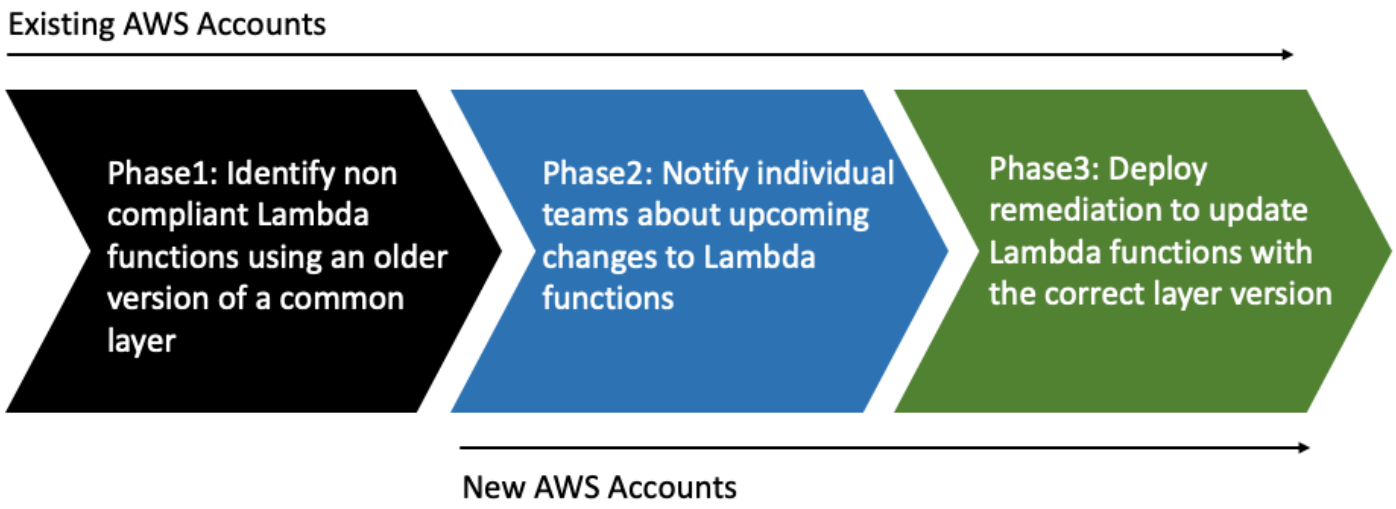
les déploiements de ressources non conformes, tels que les fonctions Lambda dépourvues de configuration de suivi active. Cela garantit que seules les ressources conformes aux dernières politiques de gouvernance sont déployées dans vos AWS environnements.

# Détectez les déploiements et les configurations Lambda non conformes avec AWS Config

Outre l'évaluation proactive, AWS Config vous pouvez également détecter de manière réactive les déploiements de ressources et les configurations non conformes à vos politiques de gouvernance. Cela est important car les politiques de gouvernance évoluent au fur et à mesure que votre organisation apprend et met en œuvre de nouvelles meilleures pratiques.

Imaginons un scénario dans lequel vous définissez une toute nouvelle politique lors du déploiement ou de la mise à jour des fonctions Lambda : toutes les fonctions Lambda doivent toujours utiliser une version de couche Lambda spécifique et approuvée. Vous pouvez configurer AWS Config pour surveiller les fonctions nouvelles ou mises à jour pour les configurations de couches. S'il AWS Config détecte une fonction qui n'utilise pas une version de couche approuvée, il signale la fonction comme une ressource non conforme. Vous pouvez éventuellement configurer AWS Config pour corriger automatiquement la ressource en spécifiant une action de correction à l'aide d'un document d' AWS Systems Manager automatisation. Par exemple, vous pouvez écrire un document d'automatisation en Python à l'aide de AWS SDK for Python (Boto3), qui met à jour la fonction non conforme pour qu'elle pointe vers la version de couche approuvée. Ainsi, il AWS Config sert à la fois de détection et de contrôle correctif, automatisant la gestion de la conformité.

Décomposons ce processus en trois phases de mise en œuvre importantes :




## Phase 1 : identification des ressources d'accès

Commencez par l'activer AWS Config sur tous vos comptes et par le configurer pour enregistrer les fonctions AWS Lambda. Cela permet d' AWS Config observer quand les fonctions Lambda sont

créées ou mises à jour. Vous pouvez ensuite configurer des règles de stratégie personnalisées pour vérifier les violations de politique spécifiques, en utilisant la syntaxe AWS CloudFormation Guard . Les règles de garde prennent la forme générale suivante :

```
rule name when condition { assertion }
```

Vous trouverez ci-dessous un exemple de règle qui vérifie qu'une couche n'est pas définie sur une ancienne version de couche :

```
rule desiredlayer when configuration.layers !empty {
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn
}
```

Découvrons la syntaxe et la structure des règles :

- Nom de la règle : le nom de la règle est `desiredlayer` dans l'exemple fourni.
- Condition : cette clause spécifie la condition dans laquelle la règle doit être vérifiée. Dans l'exemple fourni, la condition est `configuration.layers !empty`. Cela signifie que la ressource ne doit être évaluée que lorsque la propriété `layers` de la configuration n'est pas vide.
- Assertion : après la clause `when`, une assertion détermine ce que la règle vérifie. L'assertion `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` vérifie si l'un des ARN de la couche Lambda ne correspond pas à la valeur `OldLayerArn`. S'ils ne correspondent pas, l'assertion est vraie et la règle est acceptée ; sinon, elle échoue.

`CONFIG_RULE_PARAMETERS`est un ensemble spécial de paramètres configuré avec la AWS Config règle. Dans ce cas, `OldLayerArn` est un paramètre dans `CONFIG_RULE_PARAMETERS`. Cela permet aux utilisateurs de fournir une valeur d'ARN spécifique qu'ils considèrent comme ancienne ou obsolète, puis la règle vérifie si des fonctions Lambda utilisent cet ancien ARN.

### Phase 2 : Visualisation et conception

AWS Config collecte les données de configuration et les stocke dans des compartiments Amazon Simple Storage Service (Amazon S3). Vous pouvez utiliser Amazon Athena pour interroger ces données directement depuis vos compartiments S3. Avec Athena, vous pouvez agréger ces données au niveau de l'organisation, en générant une vue globale de la configuration de vos ressources sur l'ensemble de vos comptes. Pour configurer l'agrégation des données de configuration des ressources, consultez Visualiser les AWS Config données à l'aide d'Athena et d' QuickSightAmazon sur AWS le blog Cloud Operations and Management.

Voici un exemple de requête Athena pour identifier toutes les fonctions Lambda à l'aide d'un ARN de couche particulier :

```
WITH unnested AS (
    SELECT
      item.awsaccountid AS account_id,
      item.awsregion AS region,
      item.configuration AS lambda_configuration,
      item.resourceid AS resourceid,
      item.resourcename AS resourcename,
      item.configuration AS configuration,
      json_parse(item.configuration) AS lambda_json
    FROM
      default.aws_config_configuration_snapshot,
      UNNEST(configurationitems) as t(item)
    WHERE
      "dt" = 'latest'
      AND item.resourcetype = 'AWS::Lambda::Function'
  )

  SELECT DISTINCT
    region as Region,
    resourcename as FunctionName,
    json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
    json_extract_scalar(lambda_json, '$.timeout') AS timeout,
    json_extract_scalar(lambda_json, '$.version') AS version
  FROM
    unnested
  WHERE
    lambda_configuration LIKE '%arn:aws:lambda:us-
east-1:111122223333:layer:AnyGovernanceLayer:24%'
```

Voici les résultats de la requête :

Query results | Query stats

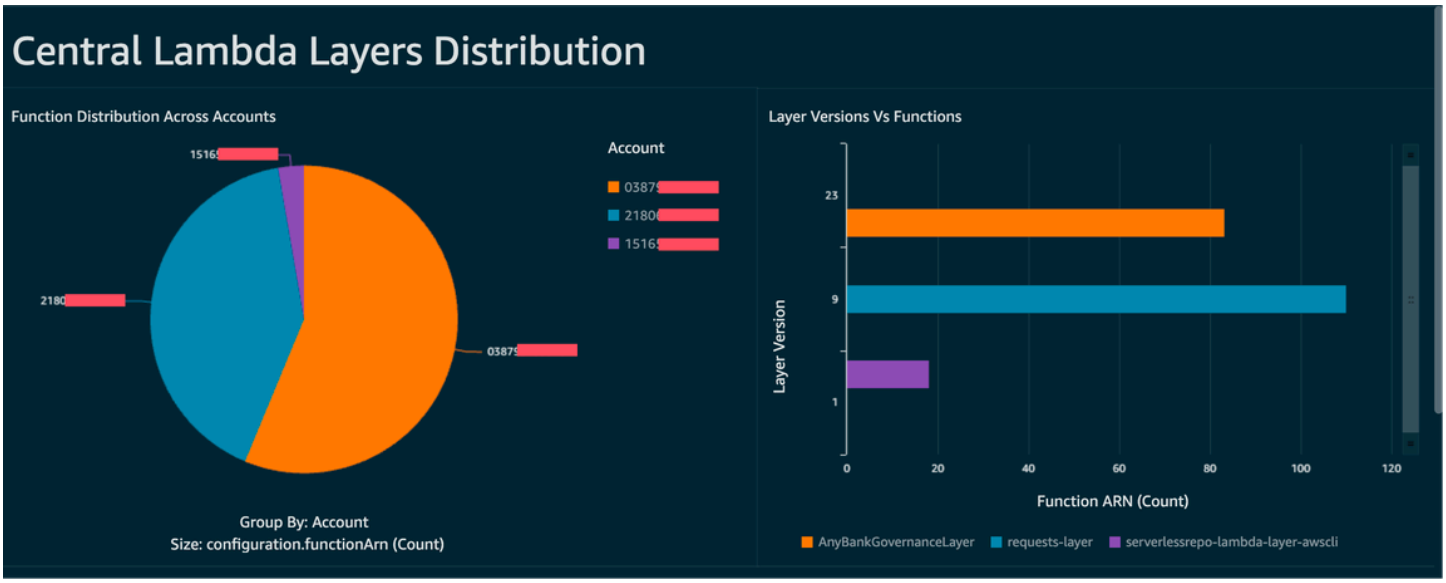
Completed — Time in queue: 127 ms — Run time: 1.803 sec — Data scanned: 239.40 KB

**Results** (27) — Copy — Download results

Search rows — 1

| # | Region | FunctionName | memory_size | timeout | version |
|---|---|---|---|---|---|
| 1 | us-east-1 | UpdateUIForPublishEvents | 128 | 18 | $LATEST |
| 2 | us-east-1 | SchedulerCLI-InstanceSchedulerMain | 128 | 300 | $LATEST |
| 3 | us-east-1 | my_functions_function10 | 128 | 3 | $LATEST |
| 4 | us-east-1 | lex-web-ui-CognitoIdentityP-CleanStackNameFunction-1TSORSH6L6YXQ | 128 | 300 | $LATEST |
| 5 | us-east-1 | GetLatestArn | 128 | 3 | $LATEST |
| 6 | us-east-1 | aws-python-http-api-project-dev-hello | 1024 | 6 | $LATEST |
| 7 | us-east-1 | cloud9-MyTest-MyTest-688JGPVYP37L | 128 | 15 | $LATEST |
| 8 | us-east-1 | my_functions_function1 | 128 | 3 | $LATEST |
| 9 | us-east-1 | my_functions_function25 | 128 | 3 | $LATEST |

Une fois les AWS Config données agrégées au sein de l'organisation, vous pouvez créer un tableau de bord à l'aide d'Amazon QuickSight. En important vos résultats Athena dans Amazon QuickSight, vous pouvez visualiser dans quelle mesure vos fonctions Lambda respectent la règle de version des couches. Ce tableau de bord peut mettre en évidence les ressources conformes et non conformes, ce qui vous aide à déterminer votre politique d'application, comme indiqué dans la section suivante. L'image suivante est un exemple de tableau de bord qui indique la distribution des versions de couches appliquées aux fonctions au sein de l'organisation.




## Phase 3 : mettre en œuvre et appliquer

Vous pouvez désormais éventuellement associer la règle de version de couche que vous avez créée lors de la phase 1 à une action de correction via un document d'automatisation de Systems Manager, que vous créez sous forme de script Python écrit avec AWS SDK for Python (Boto3). Le script

appelle l'action de l'API de UpdateFunctionconfiguration pour chaque fonction Lambda, mettant à jour la configuration de la fonction avec le nouvel ARN de couche. Vous pouvez également demander au script de soumettre une pull request au référentiel de code pour mettre à jour l'ARN de la couche. De cette façon, les futurs déploiements de code sont également mis à jour avec le bon ARN de couche.

# Signature de code Lambda avec AWS Signer

AWS Signer est un service de signature de code entièrement géré qui vous permet de valider votre code par le biais d'une signature numérique afin de confirmer que le code n'a pas été modifié et qu'il provient d'un diffuseur de publication. AWS Signer peut être utilisé conjointement avec AWS Lambda pour vérifier que les fonctions et les couches ne sont pas modifiées avant le déploiement dans vos environnements AWS. Cela protège votre organisation contre les acteurs malveillants susceptibles d'avoir obtenu des informations d'identification pour créer de nouvelles fonctions ou mettre à jour des fonctions existantes.

Pour configurer la signature de code pour vos fonctions Lambda, commencez par créer un compartiment S3 avec la gestion des versions activée. Ensuite, créez un profil de signature avec AWS Signer, spécifiez Lambda comme plate-forme, puis spécifiez une période de jours pendant laquelle le profil de signature est valide. Exemple :

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

Utilisez ensuite le profil de signature et créez une configuration de signature avec Lambda. Vous devez spécifier ce qu'il convient de faire lorsque la configuration de signature détecte un artefact qui ne correspond pas à la signature numérique attendue : avertir (mais autoriser le déploiement) ou appliquer (et bloquer le déploiement). L'exemple ci-dessous est configuré pour appliquer et bloquer les déploiements.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

AWS Signer est à présent configuré avec Lambda pour bloquer les déploiements non fiables. Supposons que vous avez terminé de coder une demande de fonctionnalité et que vous êtes

maintenant prêt à déployer la fonction. La première étape consiste à compresser le code avec les dépendances appropriées, puis à signer l'artefact à l'aide du profil de signature que vous avez créé. Vous pouvez le faire en téléchargeant l'artefact zip dans le compartiment S3, puis en lançant une tâche de signature.

```
aws signer start-signing-job \
--source 's3={bucketName=your-versioned-bucket,key=your-prefix/your-zip-
artifact.zip,version=QyaJ3c4qa50LXV.9VaZgXHlsGbvCXxpT}' \
--destination 's3={bucketName=your-versioned-bucket,prefix=your-prefix/}' \
--profile-name your-signer-id
```

Vous obtenez un résultat comme suit, où le `jobId` est l'objet créé dans le compartiment et le préfixe de destination et `jobOwner` l'identifiant Compte AWS à 12 chiffres où le travail a été exécuté.

```
{
    "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
    "jobOwner": "111122223333"
  }
```

Vous pouvez désormais déployer votre fonction à l'aide de l'objet S3 signé et de la configuration de signature de code que vous avez créée.

```
  Fn:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://your-versioned-bucket/your-prefix/87a3522b-5c0b-4d7d-
b4e0-4255a8e05388.zip
      Handler: fn.handler
      Role: !GetAtt FnRole.Arn
      CodeSigningConfigArn: !Ref pSigningConfigArn
```

Vous pouvez également tester le déploiement d'une fonction avec l'artefact zip source non signé d'origine. Le déploiement devrait échouer avec le message d'erreur suivant : .

```
Lambda cannot deploy the function. The function or layer might be signed using a
 signature that the client is not configured to accept. Check the provided signature
 for unsigned.
```

Si vous créez et déployez vos fonctions à l'aide de AWS Serverless Application Model (AWS SAM), la commande package gère le téléchargement de l'artefact zip vers S3, lance également le travail

de signature et obtient l'artefact signé. Pour ce faire, vous pouvez utiliser la commande avec les paramètres suivants :

```
sam package -t your-template.yaml \
--output-template-file your-output.yaml \
--s3-bucket your-versioned-bucket \
--s3-prefix your-prefix \
--signing-profiles your-signer-id
```

AWS Signer vous permet de vérifier que le déploiement des artefacts ZIP déployés dans vos comptes est fiable. Vous pouvez inclure le processus ci-dessus dans vos pipelines CI/CD et exiger que toutes les fonctions soient associées à une configuration de signature de code en utilisant les techniques décrites dans les rubriques précédentes. En utilisant la signature de code dans vos déploiements de fonctions Lambda, vous empêchez les acteurs malveillants susceptibles d'avoir obtenu des informations d'identification pour créer ou mettre à jour des fonctions d'injecter du code malveillant dans vos fonctions.

# Automatisez les évaluations de sécurité pour Lambda avec Amazon Inspector

Amazon Inspector est un service de gestion des vulnérabilités qui analyse continuellement les charges de travail pour détecter les vulnérabilités logicielles connues et l'exposition involontaire au réseau. Amazon Inspector crée une constatation qui décrit la vulnérabilité, identifie la ressource affectée, évalue la gravité de la vulnérabilité et fournit des conseils pour y remédier.

Le support d'Amazon Inspector fournit une évaluation automatique et continue des vulnérabilités de sécurité pour les fonctions et les couches Lambda. Amazon Inspector propose deux types de scan pour Lambda :

- Analyse standard Lambda (par défaut) : analyse les dépendances des applications au sein d'une fonction Lambda et de ses couches pour détecter les vulnérabilités des packages.
- Analyse du code Lambda : analyse le code d'application personnalisé dans vos fonctions et couches pour détecter les vulnérabilités du code. Vous pouvez activer l’analyse standard Lambda ou activer l’analyse standard Lambda conjointement avec l’analyse du code Lambda.

Pour activer Amazon Inspector, accédez à la console Amazon Inspector, développez la section Paramètres et choisissez Gestion des comptes. Dans l'onglet Comptes, choisissez Activer, puis sélectionnez l'une des options d’analyse.

Vous pouvez activer Amazon Inspector pour plusieurs comptes et déléguer les autorisations de gestion d'Amazon Inspector pour l'organisation à des comptes spécifiques lors de la configuration d'Amazon Inspector. Lors de l'activation, vous devez accorder des autorisations à Amazon Inspector en créant le rôle :`AWSServiceRoleForAmazonInspector2`. La console Amazon Inspector vous permet de créer ce rôle à l'aide d'une option en un clic.

Pour l'analyse standard Lambda, Amazon Inspector lance des analyses de vulnérabilité des fonctions Lambda dans les situations suivantes :

- Dès qu'Amazon Inspector découvre une fonction Lambda existante.
- Lorsque vous déployez une fonction Lambda.
- Lorsque vous déployez une mise à jour du code d'application ou des dépendances d'une fonction Lambda existante ou de ses couches.
- Chaque fois qu'Amazon Inspector ajoute un nouvel élément Common Vulnerabilities and Exposures (CVE) à sa base de données, et que ce CVE est pertinent pour votre fonction.

Pour l’analyse du code Lambda, Amazon Inspector évalue le code d'application de votre fonction Lambda à l'aide d'un raisonnement automatisé et d'un machine learning qui analyse le code de votre application pour vérifier sa conformité globale en matière de sécurité. Si Amazon Inspector détecte une vulnérabilité dans le code d'application de votre fonction Lambda, Amazon Inspector produit une recherche détaillée de vulnérabilité dans le code. Pour obtenir la liste des détections possibles, consultez la bibliothèque Amazon CodeGuru Detector.

Pour consulter les résultats, rendez-vous sur la console Amazon Inspector. Dans le menu Résultats, choisissez Par fonction Lambda pour afficher les résultats de l’analyse de sécurité effectuée sur les fonctions Lambda.

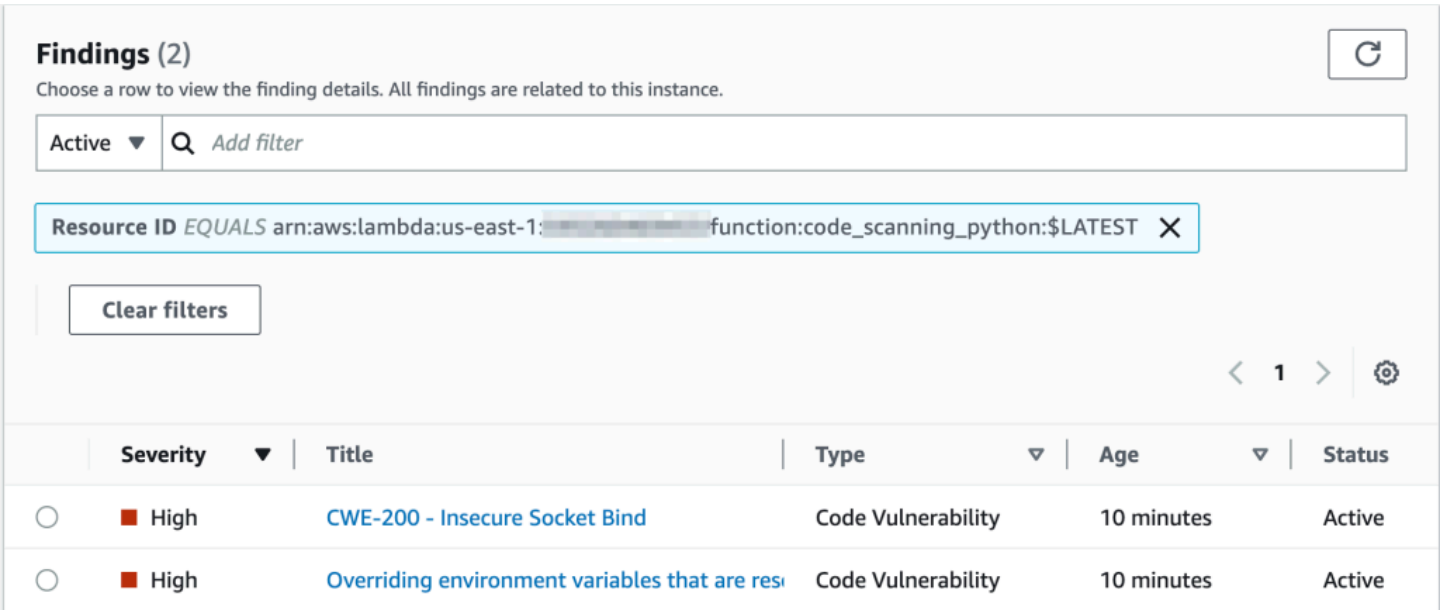
Pour exclure une fonction Lambda de l’analyse standard, balisez la fonction avec la paire clé-valeur suivante :

- `Key:InspectorExclusion`
- `Value:LambdaStandardScanning`

Pour exclure une fonction Lambda des analyses du code, balisez la fonction avec la paire clé-valeur suivante :

- `Key:InspectorCodeExclusion`
- `Value:LambdaCodeScanning`

Par exemple, comme le montre l'image suivante, Amazon Inspector détecte automatiquement les vulnérabilités et classe les résultats de type Vulnérabilité de code, ce qui indique que la vulnérabilité se trouve dans le code de la fonction, et non dans l'une des bibliothèques dépendantes du code. Vous pouvez vérifier ces informations pour une fonction spécifique ou pour plusieurs fonctions à la fois.




Vous pouvez approfondir chacun de ces résultats et découvrir comment remédier au problème.

## Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior.

Finding ID: arn:aws:inspector2:us-east-1:[redacted]:finding/[redacted]

Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior or failure of the Lambda function.

| Finding overview | |
|---|---|
| AWS account ID | [redacted] |
| Severity | High |
| Type | Code Vulnerability |
| Detector name | Override of reserved variable names in a Lambda function |
| Relevant CWE | -- |
| Rule ID | Rule-434311 |
| Detector tags | #availability, #aws-python-sdk, #aws-lambda, #data-integrity, #maintainability, #security, #security-context, #python |
| Fix available | Yes |
| Created at | March 29, 2023 10:08 AM (UTC-04:00) |

| Vulnerability details | |
|---|---|
| File path | lambda_function.py |

**Vulnerability location**

```
3  import socket
4
5  def lambda_handler(event, context):
6
7      # print("Scenario 1");
8      os.environ['_HANDLER'] = 'hello'
9      # print("Scenario 1 ends")
10
11     # print("Scenario 2");
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.bind(('',0))
```

**Suggested remediation**

Your code attempts to override an environment variable that is reserved by the Lambda runtime environment. This can lead to unexpected behavior and might break the execution of your Lambda function.

Lorsque vous travaillez avec vos fonctions Lambda, assurez-vous de respecter les conventions de dénomination de vos fonctions Lambda. Pour plus d'informations, consultez Utiliser les variables d'environnement Lambda pour configurer les valeurs dans le code.

Vous êtes responsable des suggestions de correction que vous acceptez. Passez toujours en revue les suggestions de correction avant de les accepter. Vous devrez peut-être apporter des modifications aux suggestions de correction pour vous assurer que votre code répond à vos attentes.

# Mettre en œuvre l'observabilité pour la sécurité et la conformité Lambda

AWS Config est un outil utile pour rechercher et corriger les ressources AWS sans serveur non conformes. Chaque modification que vous apportez à vos ressources sans serveur est enregistrée dans AWS Config. Vous AWS Config permet également de stocker des données de capture instantanée de configuration sur S3. Vous pouvez utiliser Amazon Athena et Amazon QuickSight pour créer des tableaux de bord et consulter les données. AWS Config Dans Détectez les déploiements et les configurations Lambda non conformes avec AWS Config, nous avons discuté de la manière dont nous pouvons visualiser une certaine configuration, comme les couches Lambda. Cette rubrique développe ces concepts.

## Visibilité sur les configurations Lambda

Vous pouvez utiliser des requêtes pour extraire des configurations importantes telles que l' Compte AWS ID, la région, la configuration de AWS X-Ray suivi, la VPC configuration, la taille de la mémoire, le temps d'exécution et les balises. Voici un exemple de requête que vous pouvez utiliser pour extraire ces informations d'Athena :

```
WITH unnested AS (
    SELECT
      item.awsaccountid AS account_id,
      item.awsregion AS region,
      item.configuration AS lambda_configuration,
      item.resourceid AS resourceid,
      item.resourcename AS resourcename,
      item.configuration AS configuration,
      json_parse(item.configuration) AS lambda_json
    FROM
      default.aws_config_configuration_snapshot,
      UNNEST(configurationitems) as t(item)
    WHERE
      "dt" = 'latest'
      AND item.resourcetype = 'AWS::Lambda::Function'
  )

  SELECT DISTINCT
    account_id,
    tags,
    region as Region,
    resourcename as FunctionName,
    json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
```

```
    json_extract_scalar(lambda_json, '$.timeout') AS timeout,
    json_extract_scalar(lambda_json, '$.runtime') AS version
    json_extract_scalar(lambda_json, '$.vpcConfig.SubnetIds') AS vpcConfig
    json_extract_scalar(lambda_json, '$.tracingConfig.mode') AS tracingConfig
  FROM
    unnested
```

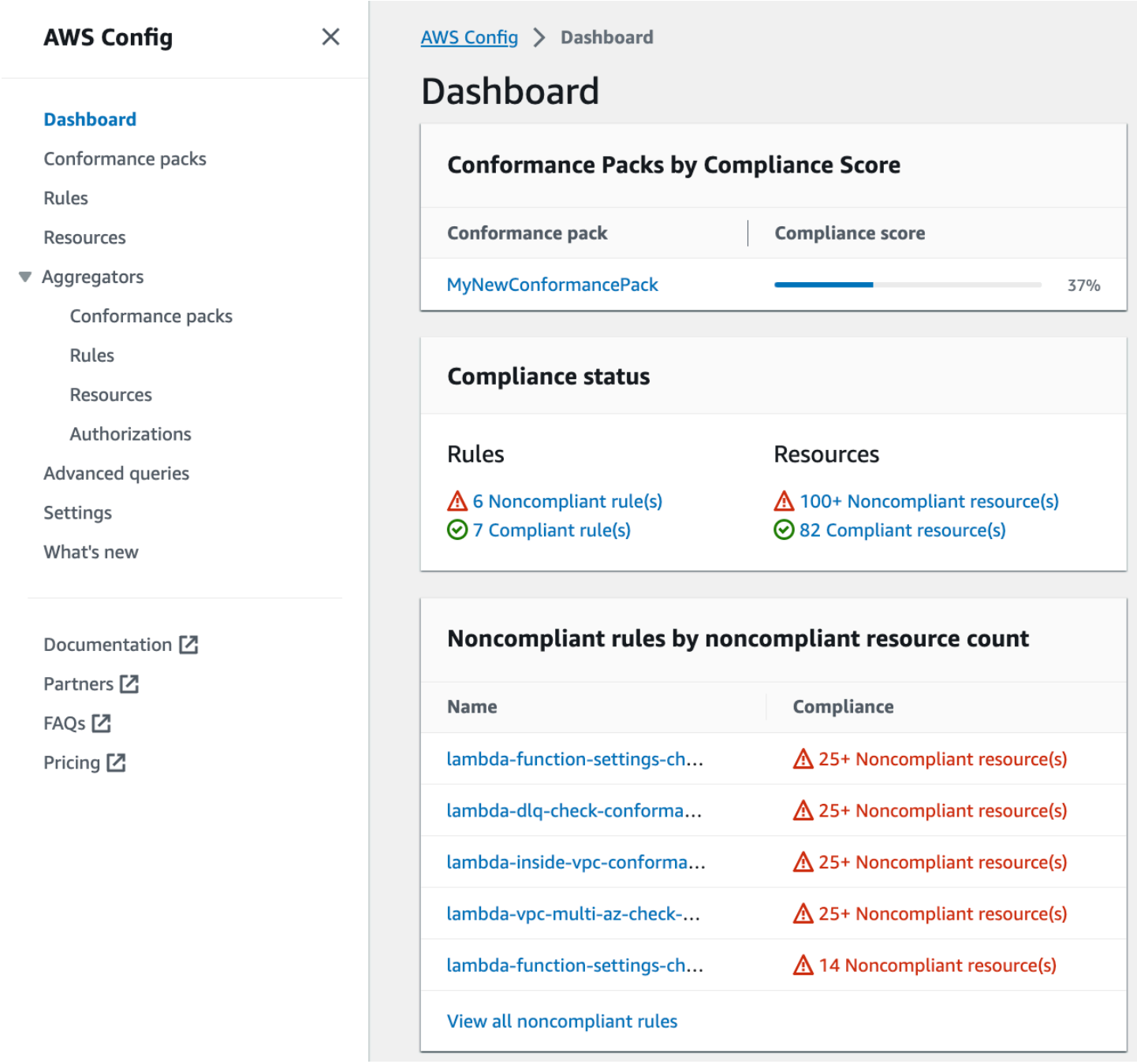
Vous pouvez utiliser la requête pour créer un QuickSight tableau de bord Amazon et visualiser les données. Pour agréger les données de configuration des AWS ressources, créer des tables dans Athena et créer des QuickSight tableaux de bord Amazon à partir des données d'Athena, consultez Visualiser les données à l'aide d' AWS Config Athena et d'Amazon QuickSight sur le blog Cloud Operations and Management. AWS Notamment, cette requête récupère également les informations de balise pour les fonctions. Cela permet de mieux comprendre vos charges de travail et vos environnements, en particulier si vous utilisez des balises personnalisées.

Pour plus d'informations sur les actions que vous pouvez entreprendre, consultez la section Prise en compte des constatations relatives à l'observabilité plus loin dans cette rubrique.

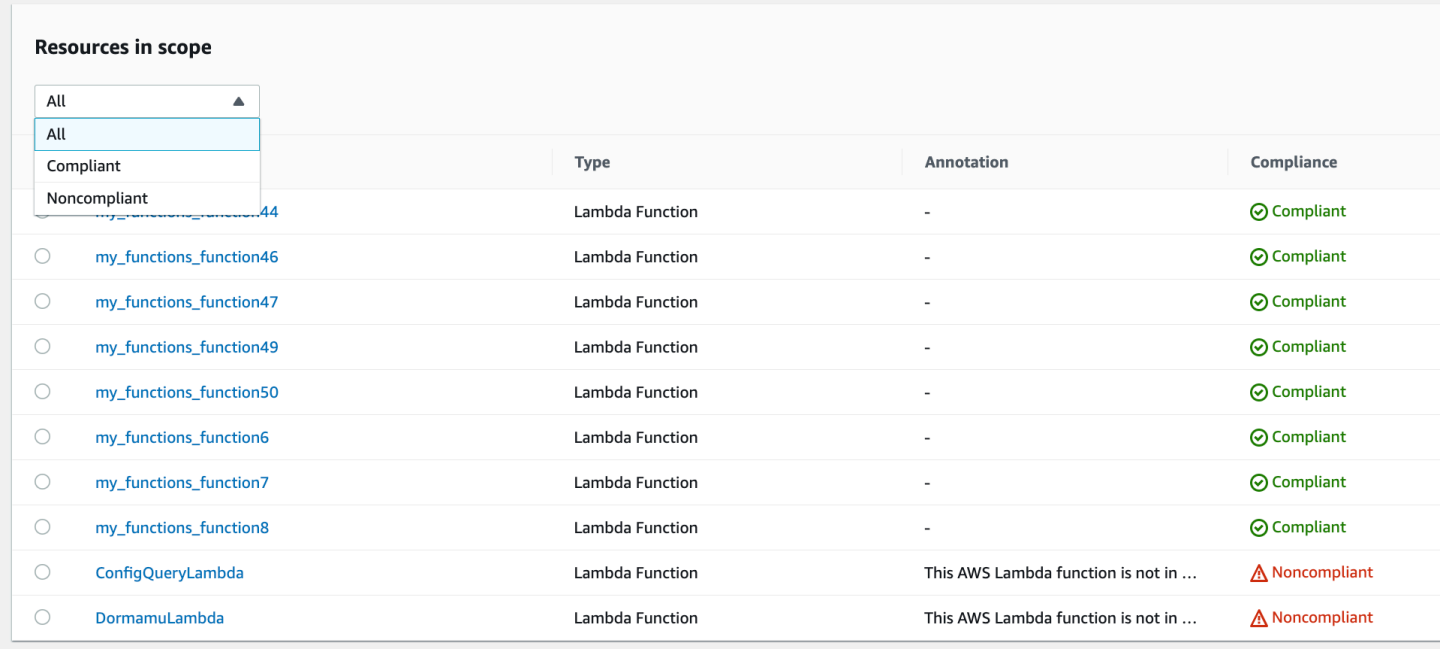
## Visibilité sur la conformité à Lambda

Avec les données générées par AWS Config, vous pouvez créer des tableaux de bord au niveau de l'organisation pour surveiller la conformité. Cela permet un suivi et une surveillance cohérents de :

- Packs de conformité par score de conformité
- Règles en fonction des ressources non conformes
- Statut de conformité

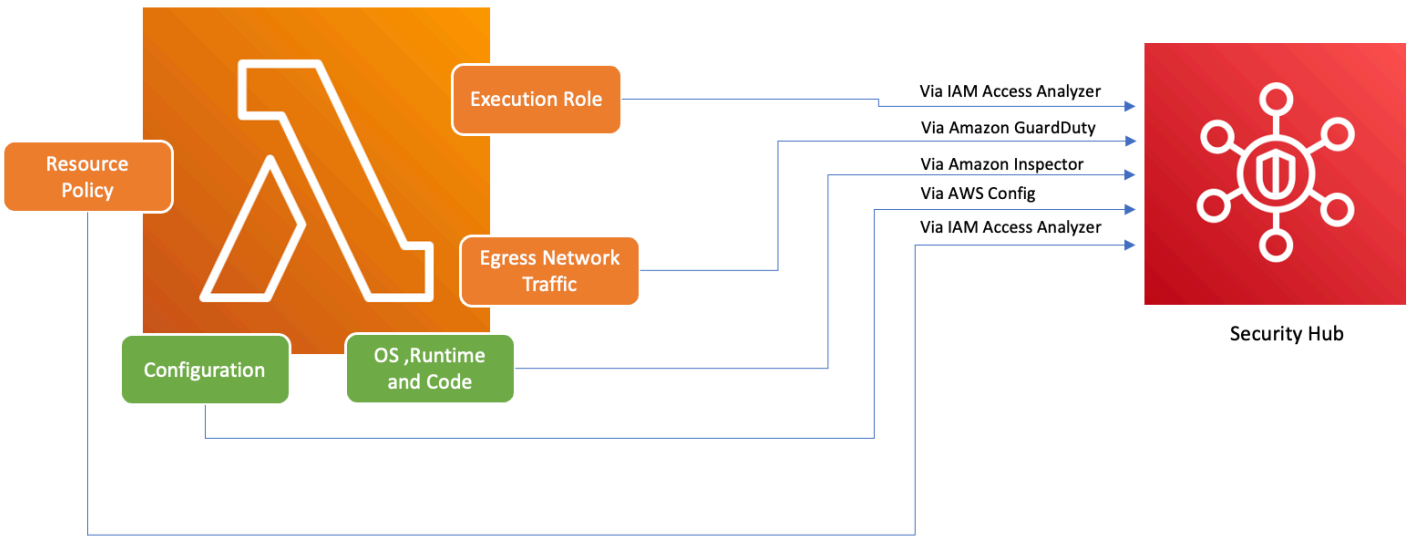



Vérifiez chaque règle pour identifier les ressources non conformes à cette règle. Par exemple, si votre organisation exige que toutes les fonctions Lambda soient associées à un VPC et si vous avez déployé une AWS Config règle pour identifier la conformité, vous pouvez sélectionner la `lambda-inside-vpc` règle dans la liste ci-dessus.




Pour plus d'informations sur les actions que vous pouvez entreprendre, consultez la section [Prise en compte des constatations relatives à l'observabilité] ci-dessous.

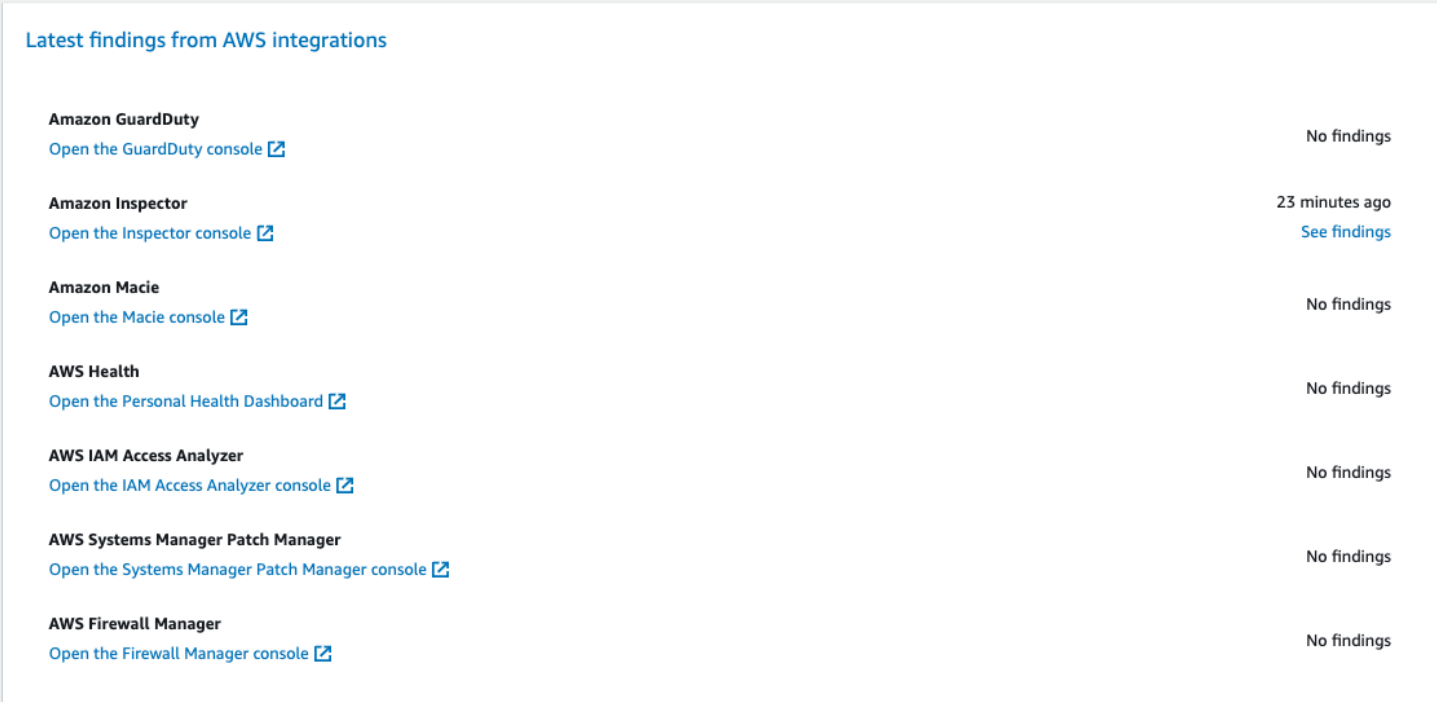
## Visibilité des limites des fonctions Lambda à l'aide de Security Hub



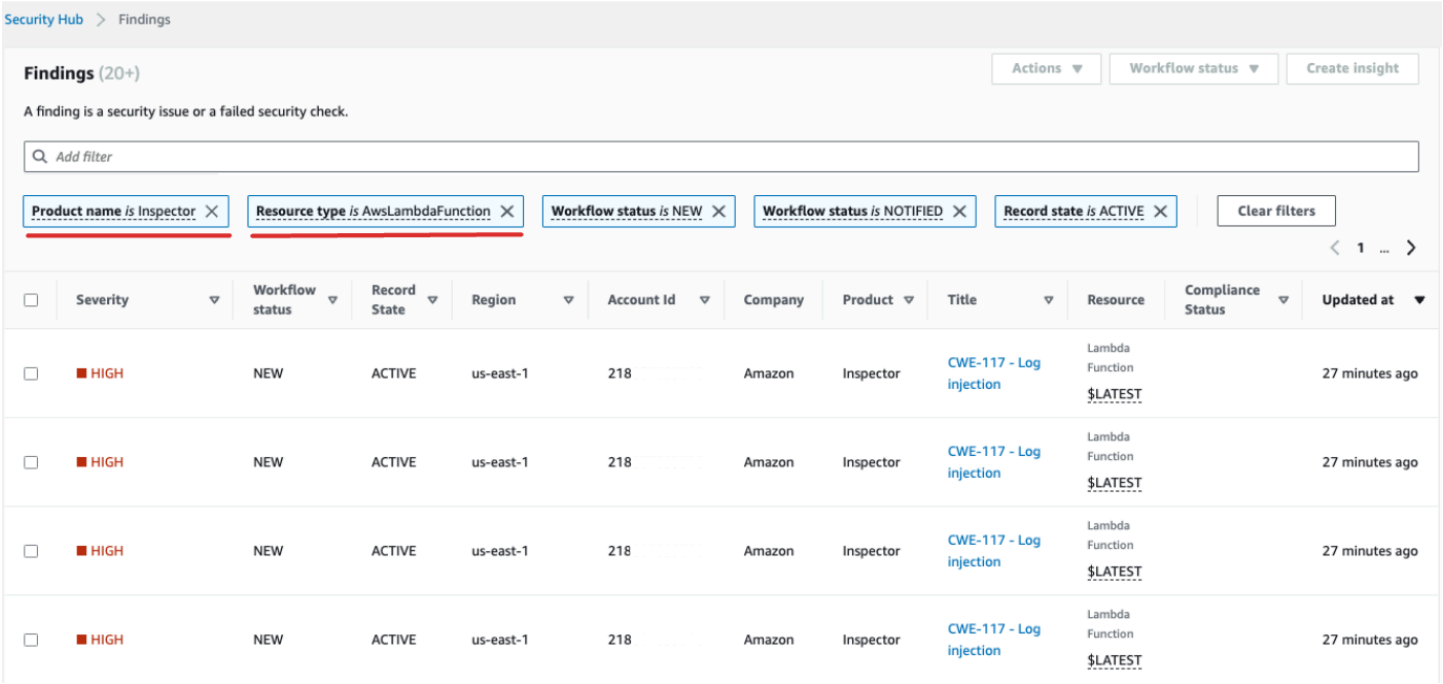

Pour garantir que les AWS services tels que Lambda sont utilisés en toute sécurité, nous avons AWS introduit les meilleures pratiques de sécurité fondamentales v1.0.0. Cet ensemble de bonnes pratiques fournit des directives claires pour sécuriser les ressources et les données dans l' AWS environnement, en soulignant l'importance de maintenir une posture de sécurité solide. Il AWS Security Hub complète cela en proposant un centre de sécurité et de conformité unifié. Il regroupe,

organise et hiérarchise les résultats de sécurité provenant de plusieurs AWS services tels qu'Amazon Inspector AWS Identity and Access Management Access Analyzer et Amazon. GuardDuty

Si Security Hub, Amazon Inspector, IAM Access Analyzer sont GuardDuty activés au sein de votre AWS organisation, Security Hub agrège automatiquement les résultats de ces services. Prenons par exemple Amazon Inspector. Security Hub vous permet d'identifier efficacement les vulnérabilités du code et des packages dans les fonctions Lambda. Dans la console Security Hub, accédez à la section inférieure intitulée Dernières découvertes issues AWS des intégrations. Ici, vous pouvez consulter et analyser les résultats provenant de divers AWS services intégrés.




Pour voir les détails, cliquez sur le lien Voir les résultats dans la deuxième colonne. Cela affiche une liste des résultats filtrés par produit, comme Amazon Inspector. Pour limiter votre recherche aux fonctions Lambda, définissez `ResourceType` sur `AwsLambdaFunction`. Cela affiche les résultats d'Amazon Inspector relatifs aux fonctions Lambda.




En GuardDuty effet, vous pouvez identifier les modèles de trafic réseau suspects. De telles anomalies peuvent suggérer l'existence d'un code potentiellement malveillant dans votre fonction Lambda.

IAMAccess Analyzer vous permet de vérifier les politiques, en particulier celles comportant des instructions de condition qui accordent l'accès aux fonctions à des entités externes. De plus, IAM Access Analyzer évalue les autorisations définies lors de l'utilisation de l'AddPermissionopération dans le API Lambda avec un. `EventSourceToken`

## Prise en compte des constatations relatives à l'observabilité

Compte tenu du large éventail de configurations possibles pour les fonctions Lambda et de leurs exigences distinctes, une solution d'automatisation standardisée pour la correction peut ne pas convenir à toutes les situations. De plus, les modifications sont mises en œuvre différemment selon les environnements. Si vous rencontrez une configuration qui ne semble pas conforme, tenez compte des directives suivantes :

1. Stratégie de balisage

   Nous vous recommandons de mettre en œuvre une stratégie de balisage complète. Chaque fonction Lambda doit être étiquetée avec des informations clés telles que :

   - Propriétaire : personne ou équipe responsable de la fonction.

- Environnement : production, mise en scène, développement ou bac à sable.
- Application : le contexte plus large auquel appartient cette fonction, le cas échéant.

2. Sensibilisation des propriétaires

   Au lieu d'automatiser les modifications majeures (comme le réglage de la VPC configuration), contactez de manière proactive les propriétaires des fonctions non conformes (identifiées par le tag du propriétaire) en leur laissant suffisamment de temps pour :

   - Ajuster les configurations non conformes sur les fonctions Lambda.
   - Fournir une explication et demander une exception, ou affiner les normes de conformité.

3. Maintenir une base de données de gestion de configuration (CMDB)

   Alors que les balises peuvent fournir un contexte immédiat, le maintien d'une approche centralisée CMDB peut fournir des informations plus approfondies. Il peut contenir des informations plus détaillées sur chaque fonction Lambda, ses dépendances et d'autres métadonnées critiques. A CMDB est une ressource inestimable pour l'audit, les contrôles de conformité et l'identification des responsables des fonctions.

Le paysage de l'infrastructure sans serveur étant en constante évolution, il est essentiel d'adopter une position proactive en matière de surveillance. Grâce à des outils tels que AWS Config Security Hub et Amazon Inspector, les anomalies potentielles ou les configurations non conformes peuvent être rapidement identifiées. Cependant, les outils ne peuvent à eux seuls garantir une conformité totale ou des configurations optimales. Il est essentiel d'associer ces outils à des processus bien documentés et aux meilleures pratiques.

- Boucle de rétroaction : une fois les étapes de correction entreprises, assurez-vous qu'il existe une boucle de rétroaction. Cela implique de revoir régulièrement les ressources non conformes pour vérifier si elles ont été mises à jour ou si elles présentent toujours les mêmes problèmes.
- Documentation : documentez toujours les observations, les mesures prises et les exceptions accordées. Une documentation appropriée aide non seulement lors des audits, mais contribue également à améliorer le processus afin d'améliorer la conformité et la sécurité à l'avenir.
- Formation et sensibilisation : assurez-vous que toutes les parties prenantes, en particulier les responsables des fonctions Lambda, sont régulièrement formées et informées des meilleures pratiques, des politiques organisationnelles et des mandats de conformité. Des ateliers, des webinaires ou des sessions de formation réguliers peuvent contribuer dans une large mesure

à garantir que tout le monde est sur la même longueur d'onde en matière de sécurité et de conformité.

En conclusion, alors que les outils et les technologies fournissent des capacités robustes pour détecter et signaler les problèmes potentiels, l'élément humain (compréhension, communication, formation et documentation) reste essentiel. Ensemble, ils forment une puissante combinaison qui garantit que vos fonctions Lambda et votre infrastructure globale restent conformes, sécurisées et optimisées pour répondre aux besoins de votre entreprise.

# Validation de la conformité pour AWS Lambda

Les auditeurs tiers évaluent la sécurité et la conformité de AWS Lambda dans le cadre de plusieurs programmes de conformité AWS. Il s'agit notamment des certifications SOC, PCI, FedRAMP, HIPAA et autres.

Pour obtenir la liste des services AWS concernés par des programmes de conformité spécifiques, consultez Services AWS concernés par les programmes de conformité. Pour obtenir des informations générales, veuillez consulter Programmes de conformité AWS.

Vous pouvez télécharger les rapports de l'audit externe avec AWS Artifact. Pour plus d'informations, consultez Téléchargement de rapports dans AWS Artifact.

Votre responsabilité de conformité lors de l'utilisation de Lambda est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise, ainsi que par la législation et la réglementation applicables. Vous pouvez mettre en œuvre des contrôles de gouvernance pour garantir que les fonctions Lambda de votre entreprise répondent à vos exigences de conformité. Pour plus d'informations, consultez Création d'une stratégie de gouvernance pour les fonctions et les couches Lambda.

# Résilience dans AWS Lambda

L'infrastructure mondiale d'AWS repose sur les régions AWS et les zones de disponibilité AWS. Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour de plus amples informations sur les régions et les zones de disponibilité AWS, veuillez consulter Infrastructure mondiale AWS.

Outre l'infrastructure globale d'AWS, Lambda propose plusieurs fonctions qui contribuent à la prise en charge de vos besoins en matière de résilience et de sauvegarde des données.

- Gestion des versions – Vous pouvez utiliser la gestion des versions dans Lambda pour enregistrer le code et la configuration de votre fonction à mesure que vous la développez. Avec les alias, vous

pouvez utiliser la gestion des versions pour effectuer des déploiements bleu/vert et de roulement. Pour plus d'informations, consultez Gérer les versions des fonctions Lambda.

- Mise à l'échelle – Lorsque votre fonction reçoit une demande tandis qu'elle traite une demande précédente, Lambda lance une autre instance de votre fonction pour gérer la charge accrue. Lambda adapte automatiquement son échelle pour gérer 1 000 exécutions simultanées par région, un quota qui peut être augmenté si nécessaire. Pour plus d'informations, consultez Comprendre la mise à l'échelle des fonctions Lambda.
- Haute disponibilité – Lambda exécute votre fonction dans plusieurs zones de disponibilité afin de vous assurer qu'elle est disponible pour traiter les événements en cas d'interruption de service dans une seule zone. Si vous configurez votre fonction pour vous connecter à un cloud privé virtuel (VPC) de votre compte, spécifiez les sous-réseaux dans plusieurs zones de disponibilité, pour une garantie de haute disponibilité. Pour plus d'informations, consultez Permettre aux fonctions Lambda d'accéder aux ressources d'un Amazon VPC.
- Simultanéité réservée – Pour vous assurer que votre fonction peut toujours adapter son échelle afin de gérer des demandes supplémentaires, vous pouvez réserver de la simultanéité. La définition de la simultanéité réservée pour une fonction permet de s'assurer qu'elle peut être mise à l'échelle, sans le dépasser, un certain nombre spécifié d'appels simultanés. Cela garantit que vous ne perdez pas les demandes en raison d'autres fonctions utilisant toute la simultanéité disponible. Pour plus d'informations, consultez Configuration de la simultanéité réservée pour une fonction.
- Nouvelles tentatives – Pour les appels asynchrones et un sous-ensemble d'appels déclenchés par d'autres services, en cas d'erreur, Lambda effectue automatiquement de nouvelles tentatives espacées. Les autres clients et les services AWSqui appellent des fonctions de manière synchrone sont responsables de l'exécution des nouvelles tentatives. Pour plus d'informations, consultez Comprendre le comportement des nouvelles tentatives dans Lambda.
- Files d'attente de lettres mortes – Pour les appels asynchrones, vous pouvez configurer Lambda de façon à envoyer à une file d'attente de lettres mortes les demandes qui ont échoué à toutes les tentatives. Une file d'attente de lettres mortes est une rubrique Amazon SNS ou une file d'attente Amazon SQS qui reçoit des événements en vue de leur dépannage ou d'un nouveau traitement. Pour de plus amples informations, veuillez consulter Ajouter une file d'attente de lettres mortes.

## Sécurité de l'infrastructure dans AWS Lambda

En tant que service géré, AWS Lambda il est protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section Sécurité du AWS cloud. Pour concevoir votre AWS environnement

en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le cadre  AWS bien architecturé du pilier de sécurité.

Vous utilisez des API appels AWS publiés pour accéder à Lambda via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Sécurité de la couche de transport (TLS). Nous avons besoin de la TLS version 1.2 et recommandons la TLS version 1.3.
- Des suites de chiffrement parfaitement confidentielles (PFS) telles que (Ephemeral Diffie-Hellman) ou DHE ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un identifiant de clé d'accès et d'une clé d'accès secrète associés à un IAM principal. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

## Utilisation de la signature de code pour vérifier l'intégrité du code avec Lambda

La signature de code pour AWS Lambda permet de garantir que seul du code fiable s'exécute dans vos fonctions Lambda. Lorsque vous activez la signature de code pour une fonction, Lambda contrôle chaque déploiement de code et vérifie que le package de code est signé par une source approuvée.

> **Note**
>
> Les fonctions définies en tant qu'images de conteneur ne prennent pas en charge la signature de code.

Pour vérifier l'intégrité du code, utilisezAWS  Signer pour créer des packages de code signés numériquement pour les fonctions et les couches. Lorsqu'un utilisateur tente de déployer un package de code, Lambda effectue des contrôles de validation sur le package de code avant d'accepter le déploiement. Étant donné que les contrôles de validation de signature de code s'exécutent au moment du déploiement, l'exécution de la fonction n'est aucunement affectée en matière de performances.

Vous pouvez également utiliser AWS Signer pour créer des profils de signature. Vous utilisez un profil de signature afin de créer le package de code signé. Utilisez AWS Identity and Access Management (IAM) pour contrôler qui peut signer les packages de code et créer des profils de signature. Pour de plus amples informations, consultez Authentification et contrôle d'accès dans le Guide du développeur AWS  Signer.

Les couches Lambda suivent le même format de package de code signé que les packages de code de fonction. Lorsque vous ajoutez une couche à une fonction pour laquelle la signature de code est activée, Lambda vérifie que la couche est signée par un profil de signature autorisé. Lorsque vous activez la signature de code pour une fonction, toutes les couches ajoutées à la fonction doivent également être signées par l'un des profils de signature autorisés.

Vous pouvez configurer la signature de code afin de journaliser les modifications dans AWS CloudTrail. Les déploiements réussis et bloqués vers les fonctions sont enregistrés CloudTrail avec des informations sur les contrôles de signature et de validation.

L'utilisation de AWS Signer ou la signature de code pour AWS Lambda.

## Validation de signature

Lambda effectue les contrôles de validation suivants lorsque vous déployez un package de code signé vers votre fonction :

1. Intégrité – Vérifie que le package de code n'a pas été modifié depuis sa signature. Lambda compare le hachage du package au hachage de la signature.
2. Expiration – Vérifie que la signature du package de code n'a pas expiré.
3. Discordance – Vérifie que le package de code est signé avec l'un des profils de signature autorisés pour la fonction Lambda. Une incompatibilité se produit aussi si une signature n'est pas présente.
4. Révocation – Vérifie que la signature du package de code n'a pas été révoquée.

La stratégie de validation de signature définie dans la configuration de signature de code détermine laquelle des actions suivantes Lambda doit accomplir en cas d'échec de l'un des contrôles de validation :

- Avertissement – Lambda autorise le déploiement du package de code, mais émet un avertissement. Lambda émet une nouvelle CloudWatch métrique Amazon et enregistre également l'avertissement dans le CloudTrail journal.

- Mise en application – Lambda émet un avertissement (identique à celui de l'action Avertissement) et bloque le déploiement du package de code.

Vous pouvez configurer la stratégie pour les contrôles de validation d'expiration, d'incompatibilité et de révocation. Notez que vous ne pouvez pas configurer une stratégie pour le contrôle d'intégrité. Si le contrôle d'intégrité échoue, Lambda bloque le déploiement.

## Configuration de la signature de code avec le Lambda API

Pour gérer les configurations de signature de code avec le AWS CLI ou AWS SDK, utilisez les API opérations suivantes :

- ListCodeSigningConfigs
- CreateCodeSigningConfig
- GetCodeSigningConfig
- UpdateCodeSigningConfig
- DeleteCodeSigningConfig

Pour gérer la configuration de signature de code pour une fonction, effectuez les API opérations suivantes :

- CreateFunction
- GetFunctionCodeSigningConfig
- PutFunctionCodeSigningConfig
- DeleteFunctionCodeSigningConfig
- ListFunctionsByCodeSigningConfig

Rubriques

- Création de configurations de signature de code pour Lambda
- Mise à jour d'une configuration de signature de code
- Configuration des IAM politiques pour les configurations de signature de code Lambda
- Utilisation de balises dans les configurations de signature de code

# Création de configurations de signature de code pour Lambda

Afin d'activer la signature de code pour une fonction, vous créez une configuration de signature de code et l'attachez à la fonction. Une configuration de signature de code définit une liste de profils de signature autorisés et l'action de stratégie à privilégier en cas d'échec de l'un des contrôles de validation.

Sections


- Conditions préalables à la configuration
- Création de configurations de signature de code
- Activation de la signature de code pour une fonction


## Conditions préalables à la configuration

Avant de configurer la signature de code pour une fonction Lambda, utilisez AWS Signer pour effectuer les opérations suivantes :

- Créer un ou plusieurs profils de signature.
- Utiliser un profil de signature afin de créer un package de code signé pour votre fonction.

Pour plus d'informations, reportez-vous à Création des profils de signature (Console) dans le Guide du développeur AWS  Signer.

## Création de configurations de signature de code

Une configuration de signature de code définit une liste des profils de signature autorisés et la stratégie de validation de signature.

Pour créer une configuration de signature de code (console)

1. Ouvrez la page des configurations de signature de code de la console Lambda.
2. Choisissez Create configuration (Créer une configuration).
3. Pour Description, saisissez un nom descriptif pour la configuration.
4. Sous Signing profiles (Profils de signature), ajoutez jusqu'à 20 profils de signature à la configuration.

a. Pour Signer la version du profilARN, choisissez le nom de ressource Amazon d'une version de profil (ARN) ou entrez leARN.

b. Pour ajouter un profil de signature supplémentaire, sélectionnez Add signing profiles (Ajouter des profils de signature).

5. Sous Signature validation policy (Stratégie de validation de signature), sélectionnez Warn (Avertissement) ou Enforce (Application).
6. Choisissez Create configuration (Créer une configuration).

### Activation de la signature de code pour une fonction

Afin d'activer la signature de code pour une fonction, vous associez une configuration de signature de code à la fonction.

Pour associer une configuration de signature de code à une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez la fonction pour laquelle vous souhaitez activer la signature de code.
3. Ouvrez l'onglet Configuration.
4. Faites défiler l'écran vers le bas et choisissez Signature par code.
5. Choisissez Modifier.
6. Sous Edit code signing (Modifier la signature de code), sélectionnez une configuration de signature de code pour cette fonction.
7. Choisissez Save (Enregistrer).

## Mise à jour d'une configuration de signature de code

Lorsque vous mettez à jour une configuration de signature de code, les modifications ont un impact sur les futurs déploiements de fonctions associées à la configuration de signature de code.

Pour mettre à jour une configuration de signature de code (console)

1. Ouvrez la page des configurations de signature de code de la console Lambda.
2. Sélectionnez une configuration de signature de code à mettre à jour, puis sélectionnez Edit (Modifier).
3. Pour Description, saisissez un nom descriptif pour la configuration.

4. Sous Signing profiles (Profils de signature), ajoutez jusqu'à 20 profils de signature à la configuration.

   a. Pour Signer la version du profilARN, choisissez le nom de ressource Amazon d'une version de profil (ARN) ou entrez leARN.

   b. Pour ajouter un profil de signature supplémentaire, sélectionnez Add signing profiles (Ajouter des profils de signature).

5. Sous Signature validation policy (Stratégie de validation de signature), sélectionnez Warn (Avertissement) ou Enforce (Application).

6. Sélectionnez Enregistrer les modifications.

## Configuration des IAM politiques pour les configurations de signature de code Lambda

Pour autoriser un utilisateur à accéder aux APIopérations de signature de code, joignez une ou plusieurs déclarations de politique à la politique utilisateur. Pour plus d'informations sur les stratégies utilisateur, consultez IAMPolitiques basées sur l'identité pour Lambda.

L'exemple d'instruction de stratégie suivant autorise la création, la mise à jour et la récupération de configurations de signature de code.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "lambda:CreateCodeSigningConfig",
          "lambda:UpdateCodeSigningConfig",
          "lambda:GetCodeSigningConfig"
        ],
      "Resource": "*"
    }
  ]
}
```

Les administrateurs peuvent utiliser la clé de condition `CodeSigningConfigArn` pour spécifier les configurations de signature de code que les développeurs doivent utiliser pour créer ou mettre à jour vos fonctions.

L'exemple de déclaration de stratégie suivant accorde l'autorisation de créer une fonction. La déclaration de stratégie comprend une condition `lambda:CodeSigningConfigArn` pour spécifier la configuration de signature de code autorisée. Lambda bloque toute `CreateFunction` API demande si son `CodeSigningConfigArn` paramètre est manquant ou ne correspond pas à la valeur de la condition.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReferencingCodeSigningConfig",
      "Effect": "Allow",
      "Action": [
          "lambda:CreateFunction",
        ],
      "Resource": "*",
      "Condition": {
          "StringEquals": {
              "lambda:CodeSigningConfigArn":
                  “arn:aws:lambda:us-west-2:123456789012:code-signing-
config:csc-0d4518bd353a0a7c6”
          }
      }
    }
  ]
}
```

## Utilisation de balises dans les configurations de signature de code

Vous pouvez baliser les configurations de signature de code pour organiser et gérer vos ressources. Les balises sont des paires clé-valeur de forme libre associées à vos ressources et prises en charge par l'ensemble des services. AWS Pour plus d'informations sur les cas d'utilisation des balises, consultez la section Stratégies de balisage courantes dans le guide des AWS ressources de balisage et de l'éditeur de balises.

Vous pouvez utiliser le AWS Lambda API pour afficher et mettre à jour les balises. Vous pouvez également afficher et mettre à jour les balises tout en gérant une configuration de signature de code spécifique dans la console Lambda.

Sections

- Autorisations requises pour l’utilisation des balises

- Utilisation des balises avec la console Lambda
- Utilisation de balises avec AWS CLI

## Autorisations requises pour l'utilisation des balises

Pour autoriser une AWS Identity and Access Management (IAM) identité (utilisateur, groupe ou rôle) à lire ou à définir des balises sur une ressource, accordez-lui les autorisations correspondantes :

- lambda : ListTags —Lorsqu' une ressource possède des balises, accordez cette autorisation à tous ceux qui ont besoin de `ListTags` l'utiliser. Pour les fonctions balisées, cette autorisation est également nécessaire pour`GetFunction`.
- lambda : TagResource —Accordez cette autorisation à toute personne ayant besoin d'appeler `TagResource` ou d'exécuter un tag lors de la création.

Pour de plus amples informations, veuillez consulter IAMPolitiques basées sur l'identité pour Lambda.

## Utilisation des balises avec la console Lambda

Vous pouvez utiliser la console Lambda pour créer des configurations de signature de code comportant des balises, ajouter des balises aux configurations de signature de code existantes et filtrer les configurations de signature de code par balise.

Pour ajouter une balise lors de la création d'une configuration de signature de code

1. Ouvrez les configurations de signature de code dans la console Lambda.
2. Dans l'en-tête du volet de contenu, choisissez Créer une configuration.
3. Dans la section en haut du volet de contenu, configurez votre configuration de signature de code. Pour plus d'informations sur la configuration des configurations de signature de code, consultezthe section called “Signature de code”.
4. Dans la section Balises, choisissez Ajouter une balise.
5. Dans le champ Clé, entrez la clé de votre tag. Pour plus d'informations sur les restrictions relatives au balisage, consultez la section Limites et exigences relatives au nommage des balises dans le Guide des  AWS ressources de balisage et de l'éditeur de balises.
6. Choisissez Create configuration (Créer une configuration).

Pour ajouter une balise à une configuration de signature de code existante

1. Ouvrez les configurations de signature de code dans la console Lambda.
2. Choisissez le nom de votre configuration de signature de code.
3. Dans les onglets situés sous le volet Détails, sélectionnez Tags.
4. Choisissez Gérer les balises.
5. Choisissez Add new tag (Ajouter une nouvelle balise).
6. Dans le champ Clé, entrez la clé de votre tag. Pour plus d'informations sur les restrictions relatives au balisage, consultez la section Limites et exigences relatives au nommage des balises dans le Guide des  AWS ressources de balisage et de l'éditeur de balises.
7. Choisissez Save (Enregistrer).

Pour filtrer les configurations de signature de code par balise

1. Ouvrez les configurations de signature de code dans la console Lambda.
2. Choisissez le champ de recherche.
3. Dans la liste déroulante, sélectionnez votre tag sous le sous-titre Tags.
4. Sélectionnez Utiliser : « tag-name » pour voir toutes les configurations de signature de code étiquetées avec cette clé, ou choisissez un opérateur pour filtrer davantage par valeur.
5. Sélectionnez la valeur de votre balise pour filtrer par une combinaison de clé et de valeur de balise.

Le champ de recherche permet également de rechercher des clés de balise. Entrez le nom d'une clé pour la retrouver dans la liste.

## Utilisation de balises avec AWS CLI

Vous pouvez ajouter et supprimer des balises sur les ressources Lambda existantes, y compris les configurations de signature de code, avec le Lambda. API Vous pouvez également ajouter des balises lors de la création d'une configuration de signature de code, ce qui vous permet de conserver une ressource étiquetée tout au long de son cycle de vie.

Mise à jour des balises avec la balise Lambda APIs

Vous pouvez ajouter et supprimer des balises pour les ressources Lambda prises en charge par le biais des opérations TagResourceet UntagResourceAPI.

Vous pouvez appeler ces opérations à l'aide du AWS CLI. Pour ajouter des balises à une ressource existante, utilisez la `tag-resource` commande. Cet exemple ajoute deux balises, dont une avec la clé *`Department`* et un avec la clé *`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pour supprimer des balises, utilisez la commande `untag-resource`. Cet exemple supprime le tag contenant la clé *`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

Ajout de balises lors de la création d'une configuration de signature de code

Pour créer une nouvelle configuration de signature de code Lambda avec des balises, utilisez l'CreateCodeSigningConfigAPIopération. Spécifiez le paramètre `Tags`. Vous pouvez appeler cette opération à l'aide de la `create-code-signing-config` AWS CLI commande et de l'`--tags`option. Pour plus d'informations sur la CLI commande, reportez-vous create-code-signing-configà la référence des AWS CLI commandes.

Avant d'utiliser le `Tags` paramètre with`CreateCodeSigningConfig`, assurez-vous que votre rôle est autorisé à étiqueter les ressources en plus des autorisations habituelles nécessaires pour cette opération. Pour plus d'informations sur les autorisations relatives au balisage, consultezthe section called “Autorisations requises pour l’utilisation des balises”.

Afficher les tags avec le tag Lambda APIs

Pour afficher les balises appliquées à une ressource Lambda spécifique, utilisez l'`ListTags`APIopération. Pour plus d'informations, consultez ListTags.

Vous pouvez appeler cette opération à l'aide de la `list-tags` AWS CLI commande en fournissant un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier
```

### Filtrer les ressources par tag

Vous pouvez utiliser cette AWS Resource Groups Tagging API GetResourcesAPIopération pour filtrer vos ressources par balises. L'opération `GetResources` reçoit jusqu'à 10 filtres, chaque filtre contenant une clé de balise et jusqu'à 10 valeurs de balise. Vous fournissez `GetResources`avec un `ResourceType` pour filtrer par certains types de ressources.

Vous pouvez appeler cette opération à l'aide de la `get-resources` AWS CLI commande. Pour des exemples d'utilisation`get-resources`, voir get-resources dans la référence des AWS CLIcommandes.

# Surveillance et dépannage des fonctions Lambda

AWS Lambda s'intègre à d'autres AWS services pour vous aider à surveiller et à dépanner vos fonctions Lambda. Lambda surveille automatiquement les fonctions Lambda en votre nom et fournit des statistiques via Amazon. CloudWatch Pour vous aider à surveiller votre code lors de l'exécution de celui-ci, Lambda suit automatiquement le nombre de demandes, la durée d'invocation par demande et le nombre de demandes générant une erreur.

Vous pouvez utiliser d'autres AWS services pour résoudre les problèmes liés à vos fonctions Lambda. Cette section explique comment utiliser ces services AWS pour surveiller, suivre, déboguer et dépanner vos fonctions et applications Lambda. Pour plus de détails sur la journalisation des fonctions et les erreurs dans chaque environnement d'exécution, consultez les sections relatives à chaque environnement d'exécution.

Pour plus d'informations sur la surveillance d'applications Lambda, consultez Surveillance et observabilité dans Serverless Land.

Sections


- Tarification
- Afficher les métriques des fonctions Lambda
- Utilisation des CloudWatch journaux et journaux avec Lambda
- Enregistrement des appels AWS Lambda d'API à l'aide de AWS CloudTrail
- Visualisez les invocations de fonctions Lambda à l'aide de AWS X-Ray
- Surveillez les performances des fonctions avec Amazon CloudWatch Lambda Insights


## Tarification

CloudWatch dispose d'un niveau gratuit perpétuel. Au-delà du seuil du niveau gratuit, les CloudWatch frais pour les métriques, les tableaux de bord, les alarmes, les journaux et les informations. Pour plus d'informations, consultez les CloudWatch tarifs Amazon.

# Afficher les métriques des fonctions Lambda

Lorsque votre AWS Lambda fonction a fini de traiter un événement, Lambda envoie des métriques concernant l'invocation à Amazon. CloudWatch Ces métriques sont gratuites.

Sur la CloudWatch console, vous pouvez créer des graphiques et des tableaux de bord avec ces indicateurs. Vous pouvez définir des alarmes pour répondre aux changements d'utilisation, de performances ou de taux d'erreur. Lambda envoie des données métriques à intervalles d'une CloudWatch minute. Pour obtenir des informations plus immédiates sur votre fonction Lambda, vous pouvez créer des métriques personnalisées à haute résolution comme décrit dans Serverless Land. Des frais s'appliquent pour les mesures personnalisées et les CloudWatch alarmes. Pour plus d'informations, consultez Amazon CloudWatch Pricing.

Cette page décrit les métriques d'invocation, de performance et de simultanéité de la fonction Lambda disponibles sur la console. CloudWatch

Sections


- Afficher les métriques sur la CloudWatch console
- Types de métriques


## Afficher les métriques sur la CloudWatch console

Vous pouvez utiliser la CloudWatch console pour filtrer et trier les métriques des fonctions par nom de fonction, alias ou version.

Pour afficher les métriques sur la CloudWatch console

1. Ouvrez la page Metrics (espace de `AWS/Lambda` noms) de la CloudWatch console.
2. Dans l'onglet Parcourir, sous Métriques, choisissez l'une des dimensions suivantes :
   - By Function Name (Par nom de fonction) (`FunctionName`) – Affichez les métriques agrégées pour l'ensemble des versions et alias d'une fonction.
   - By Resource (Par ressource) (`Resource`) – Affichez les métriques pour une version ou un alias d'une fonction.
   - By Executed Version (Par version exécutée) (`ExecutedVersion`) – Affichez les métriques pour une combinaison d'alias et de version. Utilisez la dimension `ExecutedVersion` pour

comparer les taux d'erreur pour deux versions d'une fonction qui sont les cibles d'un alias pondéré.

- Dans toutes les fonctions (aucune) : affichez les métriques agrégées pour toutes les fonctions en cours Région AWS.

3. Choisissez une métrique, puis choisissez Ajouter au graphique ou une autre option de représentation graphique.

Par défaut, les graphiques utilisent la statistique `Sum` pour toutes les métriques. Pour choisir une autre statistique et personnaliser le graphique, utilisez les options de l'onglet Graphique des métriques.

> **Note**
>
> L'horodatage d'une métrique indique le moment où la fonction a été invoquée. En fonction de la durée de l'invocation, il peut se rapporter à plusieurs minutes avant l'émission de la métrique. Par exemple, si le délai d'expiration de votre fonction est de 10 minutes, effectuez une recherche plus de 10 minutes dans le passé pour obtenir des métriques précises.

Pour plus d'informations CloudWatch, consultez le guide de CloudWatch l'utilisateur Amazon.

## Types de métriques

La section suivante décrit les types de métriques Lambda disponibles sur la CloudWatch console.

### Métriques d'invocation

Les métriques d'invocation sont des indicateurs binaires du résultat d'une invocation de fonction Lambda. Par exemple, si la fonction renvoie une erreur, Lambda envoie la métrique `Errors` avec une valeur de 1. Pour obtenir le nombre d'erreurs de fonction qui se sont produites chaque minute, examinez la somme `Sum` de la métrique `Errors` avec une période d'une minute.

> **Note**
>
> Affichez les métriques d'invocation suivantes à l'aide de la statistique `Sum`.

- `Invocations` – Nombre de fois où votre code de fonction est invoqué, y compris les invocations réussies et les invocations qui entraînent une erreur de fonction. Les invocations ne sont pas enregistrés si la demande d'invocation est limitée ou si elle entraîne une erreur d'invocation. La valeur de `Invocations` est égale au nombre de demandes facturées.
- `Errors` – Nombre d'invocations entraînant une erreur de fonction. Les erreurs de fonction incluent les exceptions levées par votre code et par l'exécution Lambda. L'environnement d'exécution renvoie des erreurs pour des problèmes tels que les expirations de délai et les erreurs de configuration. Pour calculer le taux d'erreur, divisez la valeur `Errors` par la valeur `Invocations`. Notez que l'horodatage d'une métrique d'erreur reflète l'heure d'invocation de la fonction, et non l'heure à laquelle l'erreur s'est produite.
- `DeadLetterErrors`— Pour un appel asynchrone, le nombre de fois que Lambda tente d'envoyer un événement vers une file d'attente contenant des lettres mortes () mais échoue. DLQ Des erreurs de type lettre morte peuvent se produire en raison d'une mauvaise définition des ressources ou des limites de taille.
- `DestinationDeliveryFailures` – Pour l'invocation asynchrone et les mappages des sources d'événements pris en charge, le nombre de fois où Lambda tente d'envoyer un événement à une destination, mais échoue. Pour les mappages des sources d'événements, Lambda prend en charge les destinations des sources de flux (DynamoDB et Kinesis). Des erreurs de livraison peuvent survenir en raison d'erreurs d'autorisation, de ressources mal configurées ou de limites de taille. Des erreurs peuvent également se produire si le type de destination que vous avez configuré n'est pas pris en charge, comme une SQS FIFO file d'attente Amazon ou une SNS FIFO rubrique Amazon.
- `Throttles` – Nombre de demandes d'invocation limitées. Lorsque toutes les instances de fonction traitent des requêtes et qu'aucune simultanéité n'est disponible pour effectuer une augmentation, Lambda rejette les requêtes supplémentaires avec une erreur `TooManyRequestsException`. Les demandes limitées et les autres erreurs d'invocation ne comptent pas comme `Invocations` ou `Errors`.
- `OversizedRecordCount` – Pour les sources d'événements Amazon DocumentDB, le nombre d'événements que votre fonction reçoit de votre flux de modifications et dont la taille est supérieure à 6 Mo. Lambda supprime le message et émet cette métrique.
- `ProvisionedConcurrencyInvocations` : le nombre d'invocation de votre code de fonction à l'aide de la simultanéité provisionnée.
- `ProvisionedConcurrencySpilloverInvocations` : le nombre d'invocations de votre code de fonction à l'aide d'une simultanéité standard lorsque toutes les simultanéités provisionnées sont utilisées.

- `RecursiveInvocationsDropped`— Le nombre de fois où Lambda a arrêté l'invocation de votre fonction parce qu'il a détecté que celle-ci faisait partie d'une boucle récursive infinie. La détection de boucle récursive surveille le nombre de fois qu'une fonction est invoquée dans le cadre d'une chaîne de requêtes en suivant les métadonnées ajoutées par support AWS SDKs. Par défaut, si votre fonction est invoquée dans le cadre d'une chaîne de requêtes environ 16 fois, Lambda abandonne l'appel suivant. Si vous désactivez la détection des boucles récursives, cette métrique n'est pas émise. Pour en savoir plus sur cette fonction, consultez Utilisez la détection des boucles récursives Lambda pour empêcher les boucles infinies.

## Métriques de performances

Les métriques de performance fournissent des détails sur les performances d'un invocation de fonction individuel. Par exemple, la métrique `Duration` indique la durée en millisecondes que votre fonction consacre au traitement d'un événement. Pour avoir une idée de la rapidité avec laquelle votre fonction traite les événements, affichez ces métriques avec la statistique `Average` ou `Max`.

- `Duration` – Durée de traitement d'un événement par votre code de fonction. La durée facturée pour une invocation est la valeur de la `Duration` arrondie à la milliseconde la plus proche. La `Duration` n'inclut pas le temps de démarrage à froid.
- `PostRuntimeExtensionsDuration` – Durée cumulée que l'environnement d'exécution a consacrée à l'exécution de code pour des extensions une fois l'exécution du code de la fonction terminée.
- `IteratorAge` – Pour les sources d'événements DynamoDB, Kinesis et Amazon DocumentDB, l'âge du dernier enregistrement dans l'événement. Ce métrique mesure le temps écoulé entre le moment où un flux reçoit l'enregistrement et le moment où le mappage des sources d'événements envoie l'événement à la fonction.
- `OffsetLag`— Pour les sources d'événements autogérées Apache Kafka et Amazon Managed Streaming for Apache Kafka (MSKAmazon), différence de décalage entre le dernier enregistrement écrit dans un sujet et le dernier enregistrement traité par le groupe de consommateurs de votre fonction. Bien qu'une rubrique Kafka puisse comporter plusieurs partitions, cette métrique mesure le décalage de décalage au niveau de la rubrique.

`Duration` prend également en charge les statistiques de centiles (p). Utilisez les centiles pour exclure les valeurs aberrantes qui faussent les statistiques `Average` et `Maximum`. Par exemple, la statistique p95 indique la durée maximale de 95 % des invocations, en excluant les 5 % les plus

lentes. Pour plus d'informations, consultez la section Percentiles dans le guide de CloudWatch l'utilisateur Amazon.

### Métriques de simultanéité

Lambda signale les métriques de simultanéité sous la forme d'une valeur agrégée du nombre d'instances traitant des événements au sein d'une fonction, d'une version, d'un alias ou d'une Région AWS. Pour voir la distance qui vous sépare des limites de simultanéité, affichez ces métriques à l'aide de la statistique `Max`.

- `ConcurrentExecutions` – Nombre d'instances de fonction qui traitent des événements. Si ce nombre atteint votre quota d'exécutions simultanées pour la région ou la limite de simultanéité réservée sur la fonction, Lambda limite les demandes d'invocation supplémentaires.
- `ProvisionedConcurrentExecutions` : le nombre d'instances de fonction qui traitent des événements à l'aide de la simultanéité provisionnée. Pour chaque invocation d'alias ou de version avec une simultanéité approvisionnée, Lambda émet le nombre actuel.
- `ProvisionedConcurrencyUtilization`— Pour une version ou un alias, la valeur de `ProvisionedConcurrentExecutions` divisée par le montant total de simultanéité provisionnée configurée. Par exemple, si vous configurez une simultanéité provisionnée de 10 pour votre fonction et que votre valeur `ProvisionedConcurrentExecutions` est 7, alors votre valeur `ProvisionedConcurrencyUtilization` est 0,7.
- `UnreservedConcurrentExecutions` – Pour une région, nombre d'événements traités par des fonctions qui n'ont pas de simultanéité réservée.
- `ClaimedAccountConcurrency` – Pour une région, le niveau de simultanéité qui n'est pas disponible pour les appels à la demande. `ClaimedAccountConcurrency` est égal à `UnreservedConcurrentExecutions` plus le montant de la simultanéité allouée (c'est-à-dire le total de la simultanéité réservée plus le total de la simultanéité provisionnée). Pour de plus amples informations, veuillez consulter Travailler avec la métrique `ClaimedAccountConcurrency`.

### Métriques d'invocations asynchrones

Les métriques d'invocations asynchrones fournissent des informations détaillées sur les invocations asynchrones à partir des sources d'événements et des invocations directes. Vous pouvez définir des seuils et des alarmes pour vous avertir de certaines modifications. Par exemple, en cas d'augmentation indésirable du nombre d'événements mis en file d'attente pour traitement (`AsyncEventsReceived`). Ou encore lorsqu'un événement attend depuis longtemps d'être traité (`AsyncEventAge`).

- `AsyncEventsReceived` : le nombre d'événements que Lambda met correctement en file d'attente pour traitement. Cette métrique fournit des informations sur le nombre d'événements qu'une fonction Lambda reçoit. Surveillez cette métrique et définissez des alarmes pour des seuils afin de détecter les problèmes. Par exemple, pour détecter un nombre indésirable d'événements envoyés à Lambda et pour diagnostiquer rapidement les problèmes résultant de configurations de déclencheurs ou de fonctions incorrectes. Les incompatibilités entre `AsyncEventsReceived` et `Invocations` peuvent indiquer une disparité de traitement, des événements abandonnés ou un éventuel backlog dans les files d'attente.
- `AsyncEventAge` : le temps entre le moment où Lambda met correctement l'événement en file d'attente et le moment où la fonction est invoquée. La valeur de cette métrique augmente lorsque des événements font l'objet d'une nouvelle tentative en raison d'échecs d'invocation ou de limitation. Surveillez cette métrique et définissez des alarmes pour les seuils sur différentes statistiques lorsqu'une accumulation de file d'attente se produit. Pour résoudre le problème d'une augmentation de cette métrique, examinez la métrique `Errors` pour identifier les erreurs de fonction et la métrique `Throttles` pour identifier les problèmes de simultanéité.
- `AsyncEventsDropped` : le nombre d'événements supprimés sans que la fonction ne soit exécutée correctement. Si vous configurez une file d'attente (DLQ) ou une `OnFailure` destination contenant des lettres mortes, les événements y sont envoyés avant d'être supprimés. Les événements sont supprimés pour diverses raisons. Par exemple, les événements peuvent dépasser l'âge maximal de l'événement ou atteindre le nombre maximal de nouvelles tentatives, ou la simultanéité réservée peut être définie sur 0. Pour déterminer pourquoi les événements sont supprimés, examinez la métrique `Errors` pour identifier les erreurs de fonction et la métrique `Throttles` pour identifier les problèmes de simultanéité.

# Utilisation des CloudWatch journaux et journaux avec Lambda

AWS Lambda surveille automatiquement les fonctions Lambda en votre nom pour vous aider à résoudre les défaillances de vos fonctions. Tant que le rôle d'exécution de votre fonction dispose des autorisations nécessaires, Lambda capture les journaux de toutes les demandes traitées par votre fonction et les envoie à Amazon CloudWatch Logs.

Vous pouvez insérer des instructions de journalisation dans votre code afin de faciliter la validation de son bon fonctionnement. Lambda s'intègre automatiquement à CloudWatch Logs et envoie tous les journaux de votre code à un groupe de CloudWatch journaux associé à une fonction Lambda.

Par défaut, Lambda envoie les journaux à un groupe de journaux nommé `/aws/lambda/<function name>`. Si vous souhaitez que votre fonction envoie des journaux à un autre groupe, vous pouvez le configurer à l'aide de la console Lambda, du AWS Command Line Interface (AWS CLI) ou du Lambda. API Pour en savoir plus, veuillez consulter the section called “Configuration des groupes de CloudWatch journaux”.

Vous pouvez consulter les journaux des fonctions Lambda à l'aide de la console Lambda, de la CloudWatch console, du AWS Command Line Interface (AWS CLI) ou du. CloudWatch API

> **Note**
>
> L’affichage des journaux après l’invocation d’une fonction peut prendre de 5 à 10 minutes .

## IAMAutorisations requises

Votre rôle d'exécution a besoin des autorisations suivantes pour télécharger des CloudWatch journaux dans Logs :

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Pour en savoir plus, consultez la section Utilisation de politiques basées sur l'identité (IAMpolitiques) pour les CloudWatch journaux dans le guide de  CloudWatch l'utilisateur Amazon.

Vous pouvez ajouter ces autorisations CloudWatch Logs à l'aide de la politique `AWSLambdaBasicExecutionRole` AWS gérée fournie par Lambda. Exécutez la commande suivante pour ajouter cette politique à votre rôle :

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Pour de plus amples informations, veuillez consulter the section called “AWS politiques gérées”.

## Tarification

L'utilisation des journaux Lambda est gratuite ; toutefois, les frais de CloudWatch journalisation standard s'appliquent. Pour plus d'informations, consultez CloudWatch les tarifs.

## Configuration de contrôles de journalisation avancés pour les fonctions Lambda

Pour vous permettre de mieux contrôler la manière dont les journaux de vos fonctions sont capturés, traités et consommés, Lambda propose les options de configuration de journalisation suivantes :

- Format de journal : choisissez entre le format texte brut et le JSON format structuré pour les journaux de votre fonction
- Niveau du journal : pour les journaux JSON structurés, choisissez le niveau de détail des journaux auxquels Lambda envoie CloudWatch, par exempleERROR, DEBUG ou INFO
- Groupe de journaux : choisissez le groupe de CloudWatch journaux auquel votre fonction envoie les journaux

Pour en savoir plus sur la configuration des contrôles de journalisation avancés, consultez les sections suivantes :

Rubriques

- Configuration JSON et formats de journal en texte brut
- Filtrage au niveau du journal
- Configuration des groupes de CloudWatch journaux

## Configuration JSON et formats de journal en texte brut

La capture des sorties de votre journal sous forme JSON de paires clé-valeur facilite la recherche et le filtrage lors du débogage de vos fonctions. Avec les journaux JSON formatés, vous pouvez également ajouter des balises et des informations contextuelles à vos journaux. Cela peut vous aider à effectuer une analyse automatique de gros volumes de données de journal. À moins que votre flux de développement ne repose sur des outils existants qui utilisent les journaux Lambda en texte brut, nous vous recommandons de JSON sélectionner le format de journal.

Pour tous les environnements d'exécution gérés par Lambda, vous pouvez choisir si les journaux système de votre fonction sont envoyés à CloudWatch Logs en texte brut ou au format non structuré. JSON Les journaux système sont les journaux générés par Lambda et sont parfois appelés journaux d'événements de plate-forme.

Pour les environnements d'exécution pris en charge, lorsque vous utilisez l'une des méthodes de journalisation intégrées prises en charge, Lambda peut également générer les journaux d'application de votre fonction (les journaux générés par votre code de fonction) au JSON format structuré. Lorsque vous configurez le format de journal de votre fonction pour ces environnements d'exécution, la configuration que vous choisissez s'applique à la fois aux journaux du système et aux journaux des applications.

Pour les environnements d'exécution pris en charge, si votre fonction utilise une bibliothèque ou une méthode de journalisation prise en charge, vous n'avez pas besoin de modifier votre code existant pour que Lambda capture les journaux de manière structurée. JSON

> **Note**
>
> L'utilisation du formatage du JSON journal ajoute des métadonnées supplémentaires et code les messages du journal sous forme d'JSONobjets contenant une série de paires clé-valeur. De ce fait, la taille des messages du journal de votre fonction peut augmenter.

### Temps d'exécution et méthodes de journalisation pris en charge

Lambda prend actuellement en charge la possibilité de générer des journaux d'application JSON structurés pour les environnements d'exécution suivants.

| Environnement d’exécution | Versions prises en charge |
| --- | --- |
| Java | Tous les environnements d'exécution Java sauf Java 8 sur Amazon Linux 1 |
| Node.js | Node.js 16 et versions ultérieures |
| Python | Python 3.8 et versions ultérieures |

Pour que Lambda envoie les journaux d'application de votre fonction CloudWatch dans un JSON format structuré, votre fonction doit utiliser les outils de journalisation intégrés suivants pour générer les journaux :

- Java - l’enregistreur `LambdaLogger` ou Log4j2.
- Node.js - les méthodes de console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` et `console.warn`
- Python - la bibliothèque Python `logging` standard

Pour plus d’informations sur l’utilisation des commandes de journalisation avancées avec les environnements d’exécution pris en charge, consultez the section called “Journalisation”, the section called “Journalisation” et the section called “Enregistrez et surveillez les fonctions Lambda en Python”.

Pour les autres environnements d'exécution Lambda gérés, Lambda ne prend actuellement en charge de manière native que la capture des journaux système au format structuré. JSON Cependant, vous pouvez toujours capturer les journaux d'applications JSON au format structuré dans n'importe quel environnement d'exécution en utilisant des outils de journalisation tels que Powertools pour générer AWS Lambda des sorties de journal JSON formatées.

Formats de journal par défaut

Actuellement, le format de journal par défaut pour tous les environnements d’exécution Lambda est le texte brut.

Si vous utilisez déjà des bibliothèques de journalisation telles que Powertools AWS Lambda pour générer vos journaux de fonctions au format JSON structuré, vous n'avez pas besoin de modifier votre code si vous sélectionnez le formatage des JSON journaux. Lambda n'encode pas deux fois

les journaux déjà JSON codés, de sorte que les journaux d'application de votre fonction continueront d'être capturés comme avant.

JSONformat pour les journaux système

Lorsque vous configurez le format de journal de votre fonction comme suitJSON, chaque élément du journal système (événement de plate-forme) est capturé sous la forme d'un JSON objet contenant des paires clé-valeur avec les clés suivantes :

- `"time"` - heure à laquelle le message de journal a été généré
- `"type"` - type d'événement enregistré
- `"record"` - contenu de la sortie du journal

Le format de la valeur `"record"` varie en fonction du type d'événement enregistré. Pour plus d'informations, voir the section called “Types d'objets de télémétrie API `Event`”. Pour plus d'informations sur les niveaux de journalisation attribués aux événements du journal système, consultez the section called “Mappage des événements au niveau du journal système”.

À titre de comparaison, les deux exemples suivants montrent le même résultat de journal à la fois en texte brut et en JSON format structuré. Notez que dans la plupart des cas, les événements du journal système contiennent plus d'informations lorsqu'ils sont produits au JSON format que lorsqu'ils sont produits en texte brut.

Example texte brut :

```
2024-03-13 18:56:24.046000 fbe8c1   INIT_START  Runtime Version:
 python:3.12.v18  Runtime Version ARN: arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0
```

Example structuré JSON :

```
{
  "time": "2024-03-13T18:56:24.046Z",
  "type": "platform.initStart",
  "record": {
    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "python:3.12.v18",
    "runtimeVersionArn": "arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0"
```

```
  }
}
```

> **Note**
>
> Il émet the section called “Télémétrie API” toujours des événements de plateforme tels que `START` et `REPORT` sous JSON format. La configuration du format des journaux système auxquels Lambda envoie des messages CloudWatch n'affecte pas le comportement de la télémétrie Lambda. API

JSONformat pour les journaux des applications

Lorsque vous configurez le format de journal de votre fonction comme suitJSON, les sorties du journal d'application écrites à l'aide des bibliothèques et méthodes de journalisation prises en charge sont capturées sous forme d'JSONobjet contenant des paires clé-valeur avec les clés suivantes.

- `"timestamp"` - heure à laquelle le message de journal a été généré
- `"level"` - niveau de journalisation attribué au message
- `"message"` - contenu du message de journal
- `"requestId"` (Python et Node.js) ou `"AWSrequestId"` (Java) : ID de demande unique pour l’invocation de la fonction

Selon la méthode d'exécution et de journalisation utilisée par votre fonction, cet JSON objet peut également contenir des paires de clés supplémentaires. Par exemple, dans Node.js, si votre fonction utilise des `console` méthodes pour enregistrer les objets d'erreur à l'aide de plusieurs arguments, l'JSONobjet contiendra des paires clé-valeur supplémentaires avec les clés `errorMessageerrorType`, et`stackTrace`. Pour en savoir plus sur les journaux JSON formatés dans les différents environnements d'exécution Lambda, the section called “Enregistrez et surveillez les fonctions Lambda en Python” consultezthe section called “Journalisation”, et. the section called “Journalisation”

> **Note**
>
> La clé utilisée par Lambda pour la valeur d’horodatage est différente pour les journaux système et les journaux des applications. Pour les journaux système, Lambda utilise la clé

"time" pour maintenir la cohérence avec la télémétrie. API Pour les journaux d'applications, Lambda suit les conventions des environnements d'exécution et utilise "timestamp".

À titre de comparaison, les deux exemples suivants montrent le même résultat de journal à la fois en texte brut et en JSON format structuré.

Example texte brut :

```
2024-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example structuré JSON :

```
{
    "timestamp":"2024-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"some log message",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Configuration du format du journal de votre fonction

Pour configurer le format du journal pour votre fonction, vous pouvez utiliser la console Lambda ou le AWS Command Line Interface ()AWS CLI. Vous pouvez également configurer le format du journal d'une fonction à l'aide des API commandes CreateFunctionet UpdateFunctionConfigurationLambda, de la ressource AWS Serverless Application Model (AWS SAM) AWS: :Serverless : :Function et de la ressource ::Lambda AWS CloudFormation AWS : :Function.

La modification du format de journal de votre fonction n'affecte pas les journaux existants stockés dans CloudWatch Logs. Seuls les nouveaux journaux utiliseront le format mis à jour.

Si vous modifiez le format de journal de votre fonction JSON et que vous ne définissez pas le niveau de journal, Lambda définit automatiquement le niveau de journal d'application et le niveau de journal système de votre fonction sur. INFO Cela signifie que Lambda envoie uniquement des sorties de journal de niveau ou inférieur à INFO CloudWatch Logs. Pour en savoir plus sur le filtrage au niveau des journaux d'applications et de systèmes, consultez the section called “Filtrage au niveau du journal”

> **Note**
>
> Pour les environnements d'exécution Python, lorsque le format de journal de votre fonction est défini sur du texte brut, le paramètre de niveau de journal par défaut est. WARN Cela signifie que Lambda envoie uniquement des sorties de journal de niveau ou inférieur à WARN CloudWatch Logs. Modifier le format du journal de votre fonction pour JSON modifier ce comportement par défaut. Pour en savoir plus sur la journalisation dans Python, consultez the section called “Enregistrez et surveillez les fonctions Lambda en Python”.

Pour les fonctions Node.js qui émettent des journaux au format métrique intégré (EMF), le fait de modifier le format de journal de votre fonction JSON peut CloudWatch empêcher la reconnaissance de vos métriques.

> **Important**
>
> Si votre fonction utilise Powertools for AWS Lambda (TypeScript) ou les bibliothèques EMF clientes open source pour émettre des EMF journaux, mettez à jour vos Powertools et vos EMFbibliothèques avec les dernières versions pour vous assurer qu'ils CloudWatch peuvent continuer à analyser correctement vos journaux. Si vous passez au format JSON journal, nous vous recommandons également d'effectuer des tests pour garantir la compatibilité avec les métriques intégrées de votre fonction. Pour plus d'informations sur les fonctions node.js qui émettent EMF des journaux, consultezthe section called “Utilisation de bibliothèques clientes au format métrique intégré (EMF) avec des JSON journaux structurés”.

Configurer le format de journal d’une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisir une fonction
3. Dans la page de configuration de la fonction, choisissez Outils de surveillance et d’exploitation.
4. Dans le volet de configuration de la journalisation, choisissez Modifier.
5. Sous Contenu du journal, pour Format du journal, sélectionnez Texte ou JSON.
6. Choisissez Save (Enregistrer).

Pour modifier le format du journal d'une fonction existante (AWS CLI)

- Pour modifier le format du journal d'une fonction existante, utilisez la update-function-configurationcommande. Définissez l'option `LogFormat` dans `LoggingConfig` sur `JSON` ou `Text`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON
```

Pour définir le format du journal lorsque vous créez une fonction (AWS CLI)

- Pour configurer le format du journal lorsque vous créez une nouvelle fonction, utilisez l'`--logging-config`option de la commande create-function. Définissez `LogFormat` sur `JSON` ou `Text`. L'exemple de commande suivant crée une fonction Node.js qui génère des journaux structurésJSON.

  Si vous ne spécifiez pas de format de journal lorsque vous créez une fonction, Lambda utilisera le format de journal par défaut pour la version d'exécution que vous sélectionnez. Pour plus d'informations sur les formats de journalisation par défaut, consultez the section called "Formats de journal par défaut".

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs20.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogFormat=JSON
```

## Filtrage au niveau du journal

Lambda peut filtrer les journaux de votre fonction afin que seuls les journaux d'un certain niveau de détail ou inférieur soient envoyés à CloudWatch Logs. Vous pouvez configurer le filtrage au niveau des journaux séparément pour les journaux système de votre fonction (les journaux générés par Lambda) et les journaux des applications (les journaux générés par le code de votre fonction).

Pour the section called “Temps d’exécution et méthodes de journalisation pris en charge”, vous n’avez pas besoin d’apporter de modifications au code de votre fonction pour que Lambda filtre les journaux d’application de votre fonction.

Pour tous les autres environnements d'exécution et méthodes de journalisation, le code de votre fonction doit générer les événements de journal vers `stdout` ou `stderr` sous JSON forme d'objets formatés contenant une paire clé-valeur avec la clé. `"level"` Par exemple, Lambda interprète la sortie suivante `stdout` comme un DEBUG journal de niveau.

```
print('{"level": "debug", "msg": "my debug log", "timestamp":
 "2024-11-02T16:51:31.587199Z"}')
```

Si le champ de `"level"` valeur n'est pas valide ou est manquant, Lambda attribuera le niveau à la sortie du journal. INFO Pour que Lambda utilise le champ d'horodatage, vous devez spécifier l'heure dans un format d'horodatage 3339 valideRFC. Si vous ne fournissez pas d'horodatage valide, Lambda attribuera le niveau au journal INFO et ajoutera un horodatage pour vous.

Lorsque vous nommez la clé d’horodatage, suivez les conventions du moteur d’exécution que vous utilisez. Lambda prend en charge les conventions de dénomination les plus courantes utilisées par les environnements d'exécution gérés. Par exemple, dans les fonctions qui utilisent le. NETruntime, Lambda reconnaît la clé. `"Timestamp"`

> **Note**
>
> Pour utiliser le filtrage au niveau du journal, votre fonction doit être configurée pour utiliser le format du JSON journal. Le format de journal par défaut pour tous les environnements d’exécution Lambda est actuellement le texte brut. Pour savoir comment configurer le format de journal de votre fonction surJSON, consultezthe section called “Configuration du format du journal de votre fonction”.

Pour les journaux d’applications (les journaux générés par votre code de fonction), vous pouvez choisir entre les niveaux de journalisation suivants.

| Niveau de journalisation | Utilisation standard |
| --- | --- |
| TRACE(le plus détaillé) | Les informations les plus précises utilisées pour tracer le chemin d’exécution de votre code |

| Niveau de journalisation | Utilisation standard |
| --- | --- |
| DEBUG | Informations détaillées pour le débogage du système |
| INFO | Messages qui enregistrent le fonctionnement normal de votre fonction |
| WARN | Messages relatifs à des erreurs potentielles susceptibles d’entraîner un comportement inattendu si elles ne sont pas traitées |
| ERROR | Messages concernant les problèmes qui empêchent le code de fonctionner comme prévu |
| FATAL(moindre détail) | Messages relatifs à des erreurs graves entraînant l’arrêt du fonctionnement de l’applica tion |

Lorsque vous sélectionnez un niveau de journal, Lambda envoie les journaux à ce niveau et aux niveaux inférieurs à CloudWatch Logs. Par exemple, si vous définissez le niveau de journal d'application d'une fonction surWARN, Lambda n'envoie pas de sorties de journal aux niveaux INFO etDEBUG. Le niveau de journal d'application par défaut pour le filtrage des journaux estINFO.

Lorsque Lambda filtre les journaux d'application de votre fonction, les messages de journal sans niveau se voient attribuer le niveau de journal. INFO

Pour les journaux système (les journaux générés par le service Lambda), vous pouvez choisir entre les niveaux de journalisation suivants.

| Niveau de journalisation | Utilisation |
| --- | --- |
| DEBUG(le plus détaillé) | Informations détaillées pour le débogage du système |
| INFO | Messages qui enregistrent le fonctionnement normal de votre fonction |

| Niveau de journalisation | Utilisation |
| --- | --- |
| WARN(moindre détail) | Messages relatifs à des erreurs potentielles susceptibles d'entraîner un comportement inattendu si elles ne sont pas traitées |

Lorsque vous sélectionnez un niveau de journal, Lambda envoie des journaux à ce niveau ou à un niveau inférieur. Par exemple, si vous définissez le niveau de journal système d'une fonction surINFO, Lambda n'envoie pas de sorties de journal à ce niveau. DEBUG

Par défaut, Lambda définit le niveau de journalisation du système sur. INFO Avec ce paramètre, Lambda envoie `"start"` et `"report"` enregistre automatiquement des messages à. CloudWatch Pour recevoir des journaux système plus ou moins détaillés, remplacez le niveau de journalisation par DEBUG ouWARN. Pour consulter la liste des niveaux de journalisation auxquels Lambda mappe les différents événements du journal système, consultez. the section called “Mappage des événements au niveau du journal système”

Configuration du filtrage au niveau du journal

Pour configurer le filtrage au niveau du journal des applications et du système pour votre fonction, vous pouvez utiliser la console Lambda ou le (). AWS Command Line Interface AWS CLI Vous pouvez également configurer le niveau de journalisation d'une fonction à l'aide des API commandes CreateFunctionet UpdateFunctionConfigurationLambda, de la ressource AWS Serverless Application Model (AWS SAM) AWS: :Serverless : :Function et de la ressource ::Lambda AWS CloudFormation AWS : :Function.

Notez que si vous définissez le niveau de journalisation de votre fonction dans votre code, ce paramètre a priorité sur tous les autres paramètres de journalisation que vous configurez. Par exemple, si vous utilisez la `logging` `setLevel()` méthode Python pour définir le niveau de journalisation de votre fonction surINFO, ce paramètre a priorité sur un paramètre WARN que vous configurez à l'aide de la console Lambda.

Pour configurer le niveau de journal de l'application ou du système d'une fonction existante (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Dans la page de configuration de la fonction, choisissez Outils de surveillance et d'exploitation.
4. Dans le volet de configuration de la journalisation, choisissez Modifier.

5. Sous Contenu du journal, assurez-vous que le format du journal JSONest sélectionné.
6. À l'aide des boutons d'option, sélectionnez le niveau de journal des applications et le niveau de journal du système souhaités pour votre fonction.
7. Choisissez Save (Enregistrer).

Pour configurer le niveau de journal de l'application ou du système d'une fonction existante (AWS CLI)

- Pour modifier le niveau de journalisation de l'application ou du système d'une fonction existante, utilisez la update-function-configurationcommande. `--logging-config`À utiliser pour `SystemLogLevel` définir l'une des `DEBUG` valeurs suivantes `INFO` : ou`WARN`. Défini `ApplicationLogLevel` sur l'une des valeurs `DEBUG`, `INFO`, `WARN`, `ERROR` ou `FATAL`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Pour configurer le filtrage au niveau du journal lorsque vous créez une fonction

- Pour configurer le filtrage au niveau du journal lorsque vous créez une nouvelle fonction, utilisez `--logging-config` pour définir les `ApplicationLogLevel` touches `SystemLogLevel` et dans la commande create-function. A défini `SystemLogLevel` sur l'une des valeurs `DEBUG`, `INFO` ou `WARN`. Défini `ApplicationLogLevel` sur l'une des valeurs `DEBUG`, `INFO`, `WARN`, `ERROR` ou `FATAL`.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs20.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Mappage des événements au niveau du journal système

Pour les événements de journal au niveau du système générés par Lambda, le tableau suivant définit le niveau de journal attribué à chaque événement. Pour plus d'informations sur les événements répertoriés dans le tableau, consultez the section called "Référence du schéma `Event`"

| Nom de l'événement | Condition | Niveau de journalisation attribué |
|---|---|---|
| initStart | runtimeVersion est réglé | INFO |
| initStart | runtimeVersion n'est pas défini | DEBUG |
| initRuntimeDone | status=success | DEBUG |
| initRuntimeDone | status!=success | WARN |
| initReport | initializationType= démarrage instantané | INFO |
| initReport | initializationType! = démarrage instantané | DEBUG |
| initReport | status!=success | WARN |
| restoreStart | runtimeVersion est réglé | INFO |
| restoreStart | runtimeVersion n'est pas défini | DEBUG |
| restoreRuntimeDone | status=success | DEBUG |
| restoreRuntimeDone | status!=success | WARN |
| restoreReport | status=success | INFO |
| restoreReport | status!=success | WARN |
| démarrer | - | INFO |
| runtimeDone | status=success | DEBUG |

| Nom de l'événement | Condition | Niveau de journalisation attribué |
|---|---|---|
| runtimeDone | status!=success | WARN |
| report | status=success | INFO |
| report | status!=success | WARN |
| Extension | status=success | INFO |
| Extension | status=success | WARN |
| logSubscription | - | INFO |
| telemetrySubscription | - | INFO |
| logsDropped | - | WARN |

> **Note**
>
> L'API the section called "Télémétrie API" toujours l'ensemble complet des événements de la plateforme. La configuration du niveau des journaux système auxquels Lambda envoie des messages CloudWatch n'affecte pas le comportement de la télémétrie Lambda. API

Filtrage au niveau du journal des applications avec des environnements d'exécution personnalisés

Lorsque vous configurez le filtrage au niveau du journal des applications pour votre fonction, Lambda définit en arrière-plan le niveau du journal des applications au moment de l'exécution à l'aide de la variable d'environnement `AWS_LAMBDA_LOG_LEVEL`. Lambda définit également le format du journal de votre fonction à l'aide de la variable d'environnement `AWS_LAMBDA_LOG_FORMAT`. Vous pouvez utiliser ces variables pour intégrer les commandes de journalisation avancées de Lambda dans un environnement d'exécution personnalisé.

Pour pouvoir configurer les paramètres de journalisation d'une fonction à l'aide d'un environnement d'exécution personnalisé avec la console Lambda et Lambda AWS CLI APIs, configurez votre environnement d'exécution personnalisé pour vérifier la valeur de ces variables d'environnement.

Vous pouvez ensuite configurer les enregistreurs de votre environnement d'exécution conformément au format de journal et aux niveaux de journal que vous avez sélectionnés.

## Configuration des groupes de CloudWatch journaux

Par défaut, crée CloudWatch automatiquement un groupe de journaux `/aws/lambda/<function name>` portant le nom de votre fonction lors de son appel initial. Pour configurer votre fonction afin d'envoyer des journaux à un groupe de journaux existant, ou pour créer un nouveau groupe de journaux pour votre fonction, vous pouvez utiliser la console Lambda ou AWS CLI. Vous pouvez également configurer des groupes de journaux personnalisés à l'aide des API commandes UpdateFunctionConfigurationLambda CreateFunctionet de la ressource AWS Serverless Application Model (AWS SAM) AWS: :Serverless : :Function.

Vous pouvez configurer plusieurs fonctions Lambda pour envoyer des journaux au même groupe de CloudWatch journaux. Par exemple, vous pouvez utiliser un seul groupe de journaux pour stocker les journaux de toutes les fonctions Lambda qui constituent une application particulière. Lorsque vous utilisez un groupe de journaux personnalisé pour une fonction Lambda, les flux de journaux créés par Lambda incluent le nom et la version de la fonction. Cela garantit que le mappage entre les messages du journal et les fonctions est préservé, même si vous utilisez le même groupe de journaux pour plusieurs fonctions.

Le format de dénomination des flux de journaux pour les groupes de journaux personnalisés suit cette convention :

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Notez que lors de la configuration d'un groupe de journaux personnalisé, le nom que vous sélectionnez pour votre groupe de journaux doit respecter les règles de dénomination CloudWatch des journaux. En outre, les noms de groupes de journaux personnalisés ne doivent pas commencer par la chaîne `aws/`. Si vous créez un groupe de journaux personnalisé en commençant par `aws/`, Lambda ne sera pas en mesure de créer le groupe de journaux. Par conséquent, les journaux de votre fonction ne seront pas envoyés à CloudWatch.

Pour modifier le groupe de journaux d'une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Dans la page de configuration de la fonction, choisissez Outils de surveillance et d'exploitation.

4. Dans le volet de configuration de la journalisation, choisissez Modifier.
5. Dans le volet Groupe de journalisation, pour le groupe de CloudWatch journaux, sélectionnez Personnalisé.
6. Sous Groupe de journaux personnalisé, entrez le nom du groupe de CloudWatch journaux auquel votre fonction doit envoyer des journaux. Si vous entrez le nom d'un groupe de journaux existant, votre fonction utilisera ce groupe. S'il n'existe aucun groupe de journaux portant le nom que vous entrez, Lambda créera un nouveau groupe de journaux portant ce nom pour votre fonction.

Pour modifier le groupe de journaux d'une fonction (AWS CLI)

- Pour modifier le groupe de journaux d'une fonction existante, utilisez la update-function-configurationcommande.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogGroup=myLogGroup
```

Pour indiquer un groupe de journaux personnalisé lorsque vous créez une fonction (AWS CLI)

- Pour spécifier un groupe de journaux personnalisé lorsque vous créez une nouvelle fonction Lambda à l'aide de AWS CLI, utilisez l'`--logging-config`option. L'exemple de commande suivant crée une fonction Lambda Node.js qui envoie des journaux à un groupe de journaux nommé `myLogGroup`.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs20.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogGroup=myLogGroup
```

Autorisations du rôle d'exécution

Pour que votre fonction puisse envoyer des CloudWatch journaux à Logs, elle doit disposer de l'PutLogEventsautorisation logs :. Lorsque vous configurez le groupe de journaux de votre fonction

à l'aide de la console Lambda, si votre fonction n'a pas cette autorisation, Lambda l'ajoute au rôle d'exécution de la fonction par défaut. Lorsque Lambda ajoute cette autorisation, elle autorise la fonction à envoyer des journaux à n'importe quel groupe de CloudWatch journaux Logs.

Pour empêcher Lambda de mettre automatiquement à jour le rôle d'exécution de la fonction et de le modifier manuellement, développez Autorisations et décochez Ajouter les autorisations requises.

Lorsque vous configurez le groupe de journaux de votre fonction à l'aide de AWS CLI, Lambda n'ajoute pas automatiquement l'`logs:PutLogEvents`autorisation. Ajoutez l'autorisation au rôle d'exécution de votre fonction si elle ne l'a pas déjà. Cette autorisation est incluse dans la politique AWSLambdaBasicExecutionRolegérée.

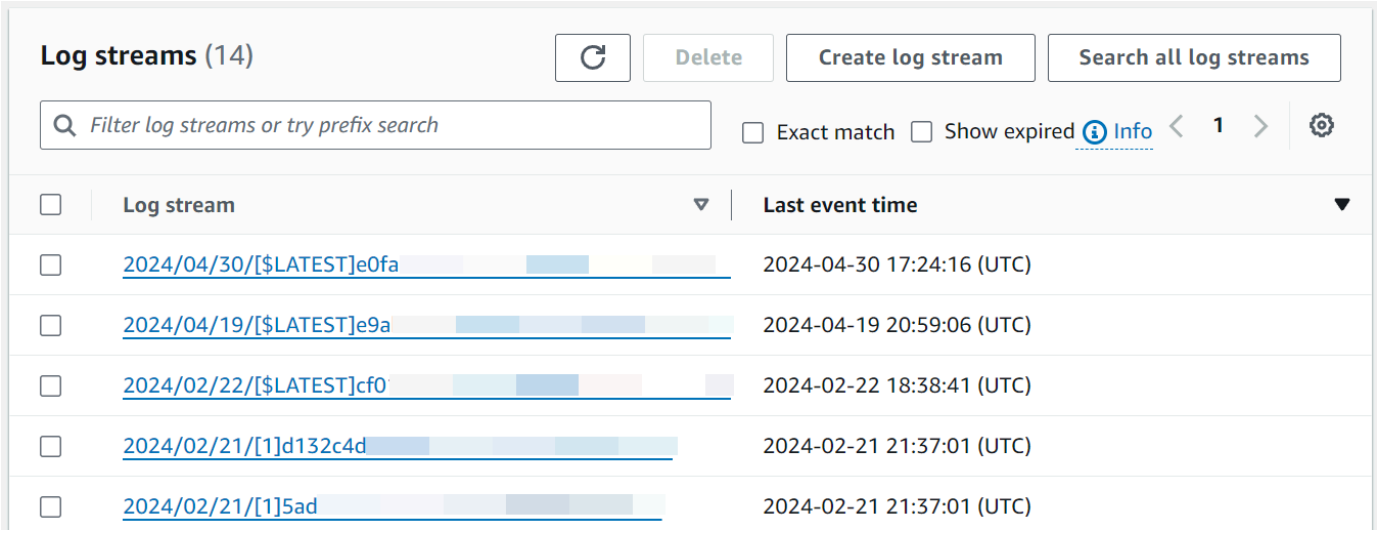
## Afficher les journaux des CloudWatch fonctions Amazon Lambda

Vous pouvez consulter CloudWatch les journaux de votre fonction Lambda à l'aide de la console Lambda, de la console CloudWatch Logs ou du (). AWS Command Line Interface AWS CLI Suivez les instructions des sections suivantes pour accéder aux journaux de votre fonction.

### Accès aux journaux à l'aide de la console

Pour afficher les journaux sur la console Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Sélectionnez une fonction.
3. Choisissez l'onglet Monitor (Surveiller).
4. Choisissez Afficher CloudWatch les journaux.
5. Faites défiler la page vers le bas et choisissez le flux Log pour les invocations de fonctions que vous souhaitez consulter.




Vous pouvez également accéder aux journaux de votre fonction directement depuis la CloudWatch console.

Pour afficher les journaux à l'aide de la CloudWatch console

1. Ouvrez la page Groupes de journaux de la CloudWatch console
2. Sélectionnez le groupe de journaux de votre fonction. Le format de nom par défaut pour les groupes de CloudWatch journaux est`/aws/lambda/<function_name>`.
3. Sélectionnez le flux journal pour les invocations de fonctions que vous souhaitez consulter.

## Accès aux journaux à l'aide du AWS CLI

AWS CLI Il s'agit d'un outil open source qui vous permet d'interagir avec les AWS services à l'aide de commandes dans votre interface de ligne de commande. Pour exécuter la procédure indiquée dans cette section, vous devez satisfaire aux exigences suivantes :

- AWS Command Line Interface (AWS CLI) version 2
- AWS CLI — Configuration rapide avec `aws configure`

Vous pouvez utiliser AWS CLI pour récupérer les journaux d'une invocation à l'aide de l'option de commande `--log-type`. La réponse inclut un champ `LogResult` qui contient jusqu'à 4 Ko de journaux codés en base64 provenant de l'invocation.

Example récupérer un ID de journal

L'exemple suivant montre comment récupérer un ID de journal à partir du champ `LogResult` d'une fonction nommée `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Vous devriez voir la sortie suivante:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example décoder les journaux

Dans la même invite de commandes, utilisez l'utilitaire `base64` pour décoder les journaux. L'exemple suivant montre comment récupérer les journaux encodés en base64 pour `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

Vous devriez voir la sortie suivante :

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

L'utilitaire `base64` est disponible sous Linux, macOS et Ubuntu sous Windows. Les utilisateurs de macOS auront peut-être besoin d'utiliser `base64 -D`.

Example Script get-logs.sh

Dans la même invite de commandes, utilisez le script suivant pour télécharger les cinq derniers événements de journalisation. Le script utilise `sed` pour supprimer les guillemets du fichier de sortie et attend 15 secondes pour permettre la mise à disposition des journaux. La sortie comprend la réponse de Lambda, ainsi que la sortie de la commande `get-log-events`.

Copiez le contenu de l'exemple de code suivant et enregistrez-le dans votre répertoire de projet Lambda sous `get-logs.sh`.

L'cli-binary-formatoption est obligatoire si vous utilisez AWS CLI la version 2. Pour faire de ce paramètre le paramètre par défaut, exécutez `aws configure set cli-binary-format raw-in-base64-out`. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur AWS Command Line Interface version 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS et Linux (uniquement)

Dans la même invite de commandes, les utilisateurs macOS et Linux peuvent avoir besoin d'exécuter la commande suivante pour s'assurer que le script est exécutable.

```
chmod -R 755 get-logs.sh
```

Example récupérer les cinq derniers événements de journal

Dans la même invite de commande, exécutez le script suivant pour obtenir les cinq derniers événements de journalisation.

```
./get-logs.sh
```

Vous devriez voir la sortie suivante :

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

# Enregistrement des appels AWS Lambda d'API à l'aide de AWS CloudTrail

AWS Lambda est intégré à AWS CloudTrailun service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. CloudTrail capture les appels d'API pour Lambda sous forme d'événements. Les appels capturés incluent des appels de la console Lambda et les appels de code vers les opérations d'API Lambda. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite à Lambda, l'adresse IP à partir de laquelle la demande a été faite, la date à laquelle elle a été faite et des informations supplémentaires.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou d'utilisateur root.
- Si la demande a été faite au nom d'un utilisateur de l'IAM Identity Center.
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la requête a été effectuée par un autre Service AWS.

CloudTrail est actif dans votre compte Compte AWS lorsque vous créez le compte et vous avez automatiquement accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section Utilisation de l'historique des CloudTrail événements dans le guide de AWS CloudTrail l'utilisateur. La consultation de CloudTrail l'historique des événements est gratuite.

Pour un enregistrement continu des événements de vos 90 Compte AWS derniers jours, créez un magasin de données sur les événements de Trail ou CloudTrailLake.

CloudTrail sentiers

Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Tous les sentiers créés à l'aide du AWS Management Console sont multirégionaux. Vous pouvez créer un parcours à région unique ou multirégionale à l'aide du. AWS CLI Il est recommandé de créer un parcours multirégional, car vous capturez l'activité dans l'ensemble Régions AWS de votre

compte. Si vous créez un parcours à région unique, vous ne pouvez voir que les événements enregistrés dans le parcours. Région AWS Pour plus d'informations sur les sentiers, consultez les sections Création d'un sentier pour votre organisation Compte AWS et Création d'un sentier pour une organisation dans le guide de AWS CloudTrail l'utilisateur.

Vous pouvez envoyer une copie de vos événements de gestion en cours dans votre compartiment Amazon S3 gratuitement CloudTrail en créant un journal. Toutefois, des frais de stockage Amazon S3 sont facturés. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification. Pour obtenir des informations sur la tarification Amazon S3, consultez Tarification Amazon S3.

CloudTrail Stockages de données sur les événements du lac

CloudTrail Lake vous permet d'exécuter des requêtes SQL sur vos événements. CloudTrail Lake convertit les événements existants au format JSON basé sur les lignes au format Apache ORC. ORC est un format de stockage en colonnes qui est optimisé pour une récupération rapide des données. Les événements sont agrégés dans des magasins de données d'événement. Ceux-ci constituent des collections immuables d'événements basées sur des critères que vous sélectionnez en appliquant des sélecteurs d'événements avancés. Les sélecteurs que vous appliquez à un magasin de données d'événement contrôlent les événements qui persistent et que vous pouvez interroger. Pour plus d'informations sur CloudTrail Lake, consultez la section Travailler avec AWS CloudTrail Lake dans le guide de AWS CloudTrail l'utilisateur.

CloudTrail Les stockages et requêtes de données sur les événements de Lake entraînent des coûts. Lorsque vous créez un magasin de données d'événement, vous choisissez l'option de tarification que vous voulez utiliser pour le magasin de données d'événement. L'option de tarification détermine le coût d'ingestion et de stockage des événements, ainsi que les périodes de conservation par défaut et maximale pour le magasin de données d'événement. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

## Événements relatifs aux données Lambda dans CloudTrail

Les événements de données fournissent des informations sur les opérations de ressources effectuées sur ou dans une ressource (par exemple, lecture ou écriture de données dans un objet Amazon S3). Ils sont également connus sous le nom opérations de plans de données. Les événements de données sont souvent des activités dont le volume est élevé. Par défaut, CloudTrail n'enregistre pas la plupart des événements liés aux données et l'historique des CloudTrail événements ne les enregistre pas.

Un événement de CloudTrail données enregistré par défaut pour les services pris en charge est`LambdaESMDisabled`. Pour en savoir plus sur l'utilisation de cet événement afin de résoudre les problèmes liés aux mappages de sources d'événements Lambda, consultez. the section called “Utilisation CloudTrail pour résoudre les problèmes liés aux sources d'événements Lambda désactivées”

Des frais supplémentaires s’appliquent pour les événements de données. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

Vous pouvez enregistrer les événements de données pour le type de `AWS::Lambda::Function` ressource à l'aide de la CloudTrail console ou AWS CLI des opérations de CloudTrail l'API. Pour plus d'informations sur la façon de consigner les événements liés aux données, consultez les sections Enregistrement des événements liés aux données avec le AWS Management Console et Enregistrement des événements liés aux données avec le AWS Command Line Interface dans le Guide de AWS CloudTrail l'utilisateur.

Le tableau suivant répertorie le type de ressource Lambda pour lequel vous pouvez enregistrer des événements de données. La colonne Type d'événement de données (console) indique la valeur à choisir dans la liste des types d'événements de données de la CloudTrail console. La colonne de valeur resources.type indique la **`resources.type`** valeur que vous devez spécifier lors de la configuration de sélecteurs d'événements avancés à l'aide des API or. AWS CLI CloudTrail La CloudTrail colonne Data APIs logged to indique les appels d'API enregistrés CloudTrail pour le type de ressource.

| Type d’événement de données (console) | valeur resources.type | API de données connectées à CloudTrail |
|---|---|---|
| Lambda | `AWS::Lambda::Function` | Invoke |

Vous pouvez configurer des sélecteurs d'événements avancés pour filtrer les `eventName`, `readOnly`, et `resources.ARN` des champs pour enregistrer uniquement les événements importants pour vous. L'exemple suivant est la vue JSON d'une configuration d'événements de données qui enregistre les événements pour une fonction spécifique uniquement. Pour plus d'informations sur ces champs, consultez AdvancedFieldSelectorla référence de l'AWS CloudTrail API.

```
[
```

```
  {
    "name": "function-invokes",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::Lambda::Function"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:lambda:us-east-1:111122223333:function:hello-world"
        ]
      }
    ]
  }
]
```

## Événements de gestion Lambda dans CloudTrail

Les événements de gestion fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre Compte AWS. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail enregistre les événements de gestion.

Lambda prend en charge la journalisation des actions suivantes sous forme d'événements de gestion dans des fichiers CloudTrail journaux.

> **Note**
>
> Dans le fichier CloudTrail journal, ils `eventName` peuvent inclure des informations de date et de version, mais cela fait toujours référence à la même action d'API publique. Par exemple, l'`GetFunction`action apparaît sous la forme`GetFunction20150331v2`. La liste suivante indique quand le nom de l'événement est différent du nom de l'action de l'API.

- AddLayerVersionPermission
- AddPermission(nom de l'événement :`AddPermission20150331v2`)
- CreateAlias(nom de l'événement :`CreateAlias20150331`)
- CreateEventSourceMapping(nom de l'événement :`CreateEventSourceMapping20150331`)
- CreateFunction(nom de l'événement :`CreateFunction20150331`)

  (Les `ZipFile` paramètres `Environment` et sont omis dans les CloudTrail journaux pour`CreateFunction`.)
- CreateFunctionUrlConfig
- DeleteAlias(nom de l'événement :`DeleteAlias20150331`)
- DeleteCodeSigningConfig
- DeleteEventSourceMapping(nom de l'événement :`DeleteEventSourceMapping20150331`)
- DeleteFunction(nom de l'événement :`DeleteFunction20150331`)
- DeleteFunctionConcurrence (nom de l'événement :`DeleteFunctionConcurrency20171031`)
- DeleteFunctionUrlConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias(nom de l'événement :`GetAlias20150331`)
- GetEventSourceMapping
- GetFunction
- GetFunctionUrlConfig
- GetFunctionConfiguration
- GetLayerVersionPolicy
- GetPolicy
- ListEventSourceMappings
- ListFunctions
- ListFunctionUrlConfigs
- PublishLayerVersion (nom de l'événement :`PublishLayerVersion20181031`)

  (Le `ZipFile` paramètre est omis dans les CloudTrail journaux de`PublishLayerVersion`.)
- PublishVersion(nom de l'événement :`PublishVersion20150331`)
- PutFunctionConcurrence (nom de l'événement :`PutFunctionConcurrency20171031`)
- PutFunctionCodeSigningConfig

- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- PutRuntimeManagementConfig
- RemovePermission(nom de l'événement :`RemovePermission20150331v2`)
- TagResource(nom de l'événement :`TagResource20170331v2`)
- UntagResource(nom de l'événement :`UntagResource20170331v2`)
- UpdateAlias(nom de l'événement :`UpdateAlias20150331`)
- UpdateCodeSigningConfig
- UpdateEventSourceMapping(nom de l'événement :`UpdateEventSourceMapping20150331`)
- UpdateFunctionCode (nom de l'événement :`UpdateFunctionCode20150331v2`)

  (Le `ZipFile` paramètre est omis dans les CloudTrail journaux de`UpdateFunctionCode`.)
- UpdateFunctionConfiguration (nom de l'événement :`UpdateFunctionConfiguration20150331v2`)

  (Le `Environment` paramètre est omis dans les CloudTrail journaux de`UpdateFunctionConfiguration`.)
- UpdateFunctionEventInvokeConfig
- UpdateFunctionUrlConfig

## Utilisation CloudTrail pour résoudre les problèmes liés aux sources d'événements Lambda désactivées

Lorsque vous modifiez l'état du mappage d'une source d'événement à l'aide de l'action d'UpdateEventSourceMappingAPI, l'appel d'API est enregistré en tant qu'événement de gestion CloudTrail. Les mappages de sources d'événements peuvent également passer directement à l'`Disabled`état en raison d'erreurs.

Pour les services suivants, Lambda publie l'événement de `LambdaESMDisabled` données CloudTrail lorsque votre source d'événements passe à l'état Désactivé :

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda ne prend en charge cet événement pour aucun autre type de mappage de source d'événements.

Pour recevoir des alertes lorsque les mappages de sources d'événements pour les services pris en charge passent à l'`Disabled`état, configurez une alarme dans Amazon à CloudWatch l'aide de l'`LambdaESMDisabled` CloudTrail événement. Pour en savoir plus sur la configuration d'une CloudWatch alarme, voir Création d' CloudWatch alarmes pour CloudTrail des événements : exemples.

L'`serviceEventDetails`entité figurant dans le message d'`LambdaESMDisabled`événement contient l'un des codes d'erreur suivants.

**`RESOURCE_NOT_FOUND`**

La ressource spécifiée dans la demande n’existe pas.

**`FUNCTION_NOT_FOUND`**

La fonction attachée à la source d'événement n'existe pas.

**`REGION_NAME_NOT_VALID`**

Un nom de région fourni à la source ou à la fonction d'événement n'est pas valide.

**`AUTHORIZATION_ERROR`**

Les autorisations n'ont pas été définies ou sont mal configurées.

**`FUNCTION_IN_FAILED_STATE`**

Le code de fonction ne compile pas, a rencontré une exception irrécupérable ou un mauvais déploiement s'est produit.

## Exemples d'événements Lambda

Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'opération d'API demandée, la date et l'heure de l'opération, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics. Les événements n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre les entrées du CloudTrail journal pour les `DeleteFunction` actions `GetFunction` et.

> **Note**
>
> `eventName` peut inclure des informations de date et de version, comme `"GetFunction20150331"`, mais il s'agit toujours de la même API publique.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:03:36Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "GetFunction",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httplib2/0.8 (gzip)",
      "errorCode": "AccessDenied",
      "errorMessage": "User: arn:aws:iam::111122223333:user/myUserName is not
 authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:111122223333:function:other-acct-function",
      "requestParameters": null,
      "responseElements": null,
      "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
      "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
```

```
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:04:42Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "DeleteFunction20150331",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httplib2/0.8 (gzip)",
      "requestParameters": {
        "functionName": "basic-node-task"
      },
      "responseElements": null,
      "requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
      "eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    }
  ]
}
```

Pour plus d'informations sur le contenu des CloudTrail enregistrements, voir le contenu des CloudTrail enregistrements dans le Guide de AWS CloudTrail l'utilisateur.

# Visualisez les invocations de fonctions Lambda à l'aide de AWS X-Ray

Vous pouvez l'utiliser AWS X-Ray pour visualiser les composants de votre application, identifier les goulots d'étranglement liés aux performances et résoudre les demandes ayant entraîné une erreur. Vos fonctions Lambda envoient des données de suivi à X-Ray qui les traite pour générer une cartographie de service et des résumés de suivi pouvant faire l'objet d'une recherche.

Si vous avez activé le suivi X-Ray dans un service qui invoque votre fonction, Lambda envoie automatiquement les suivis à X-Ray. Le service en amont, tel qu'Amazon API Gateway, ou une application hébergée sur Amazon EC2 qui est instrumentée avec le X-RaySDK, échantillonne les demandes entrantes et ajoute un en-tête de suivi indiquant à Lambda d'envoyer des traces ou non. Les traces provenant des producteurs de messages en amont, tels qu'AmazonSQS, sont automatiquement liées aux traces des fonctions Lambda en aval, créant ainsi une end-to-end vue de l'ensemble de l'application. Pour plus d'informations, consultez Traçage des applications événementielles dans le Guide du développeur AWS X-Ray .

> **Note**
>
> Le suivi X-Ray n'est actuellement pas pris en charge pour les fonctions Lambda avec Amazon Managed Streaming for Apache Kafka (MSKAmazon), Apache Kafka autogéré, Amazon MQ avec ActiveMQ et RabbitMQ, ou les mappages de sources d'événements Amazon DocumentDB.

Pour activer/désactiver le traçage actif sur votre fonction Lambda avec la console, procédez comme suit :

Pour activer le traçage actif

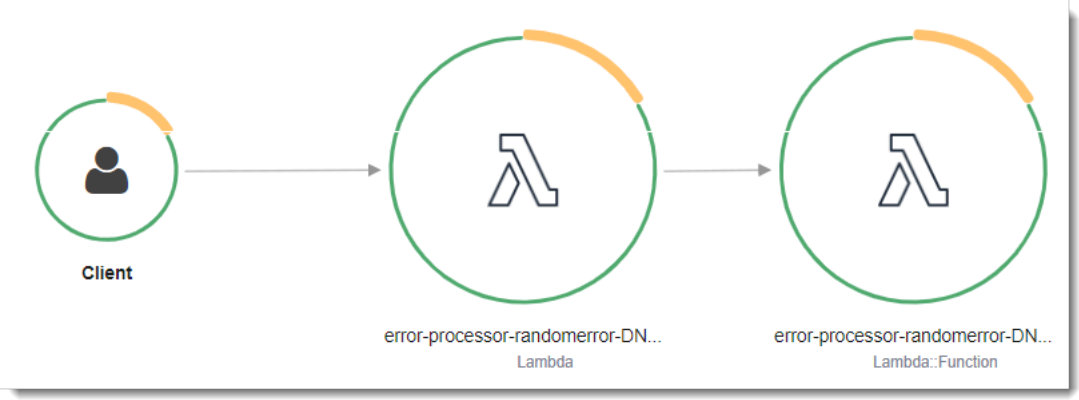
1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Choisissez Configuration, puis choisissez Outils de surveillance et d'opérations.
4. Choisissez Modifier.
5. Sous X-Ray, activer/désactiver Active tracing (Traçage actif).
6. Choisissez Save (Enregistrer).

Votre fonction a besoin d'une autorisation pour charger des données de suivi vers X-Ray. Lorsque vous activez le suivi actif dans la console Lambda, Lambda ajoute les autorisations requises au rôle d'exécution de votre fonction. Dans le cas contraire, ajoutez la AWSXRayDaemonWriteAccesspolitique au rôle d'exécution.

X-Ray ne trace pas toutes les requêtes vers votre application. X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif de toutes les demandes. Le taux d'échantillonnage est 1 demande par seconde et 5 % de demandes supplémentaires. Vous ne pouvez pas configurer la fréquence d'échantillonnage des rayons X pour vos fonctions.

## Comprendre les suivis X-Ray

Dans X-Ray, un suivi enregistre des informations sur une demande traitée par un ou plusieurs services. Lambda enregistre 2 segments par trace, ce qui crée deux nœuds sur le graphe de service. L'image suivante met en évidence ces deux nœuds :




Le premier nœud sur la gauche représente le service Lambda qui reçoit la demande d'invocation. Le deuxième nœud représente votre fonction Lambda spécifique.

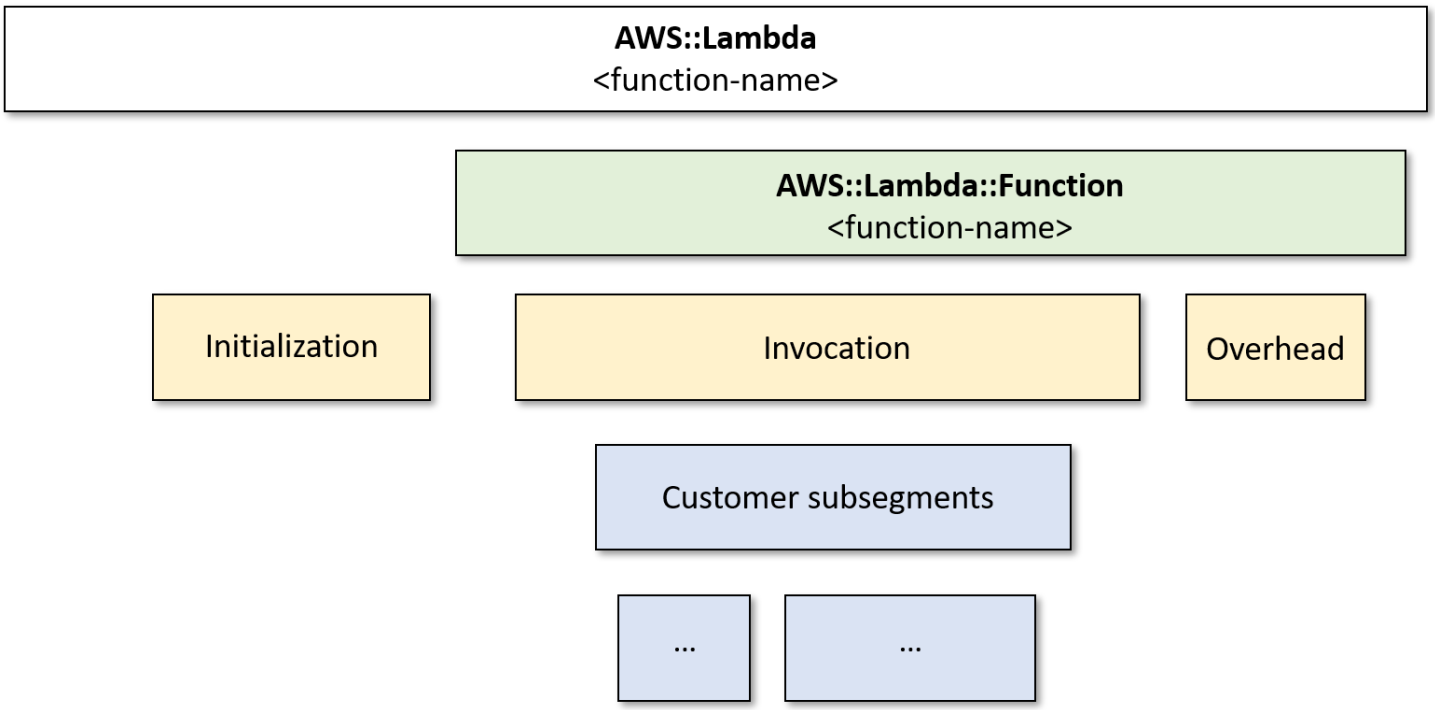
Le segment enregistré pour le service Lambda couvre toutes les étapes nécessaires à la préparation de l'environnement d'exécution Lambda. `AWS::Lambda` Cela inclut la planification de la microVM, la création ou le déblocage d'un environnement d'exécution avec les ressources que vous avez configurées, ainsi que le téléchargement de votre code de fonction et de toutes les couches.

Le `AWS::Lambda::Function` segment concerne le travail effectué par la fonction.

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> Cette modification affecte les sous-segments du segment de fonction. Les paragraphes suivants décrivent à la fois les anciens et les nouveaux formats de ces sous-segments.
> Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Structure de segment AWS X-Ray Lambda à l'ancienne

L'ancienne structure X-Ray du `AWS::Lambda` segment ressemble à ce qui suit :




Dans ce format, le segment de fonction comporte des sous-segments pour `Initialization`, `Invocation` et `Overhead`. Pour Lambda SnapStart uniquement, il existe également un `Restore` sous-segment (non représenté sur ce schéma).

Le sous-segment `Initialization` représente la phase d'initialisation du cycle de vie de l'environnement d'exécution Lambda. Au cours de cette phase, Lambda initialise les extensions, initialise le moteur d'exécution et exécute le code d'initialisation de la fonction.

Le sous-segment `Invocation` représente la phase d'invocation où Lambda invoque le gestionnaire de fonction. Cela commence par l'enregistrement de l'exécution et de l'extension et se termine lorsque l'exécution est prête à envoyer la réponse.

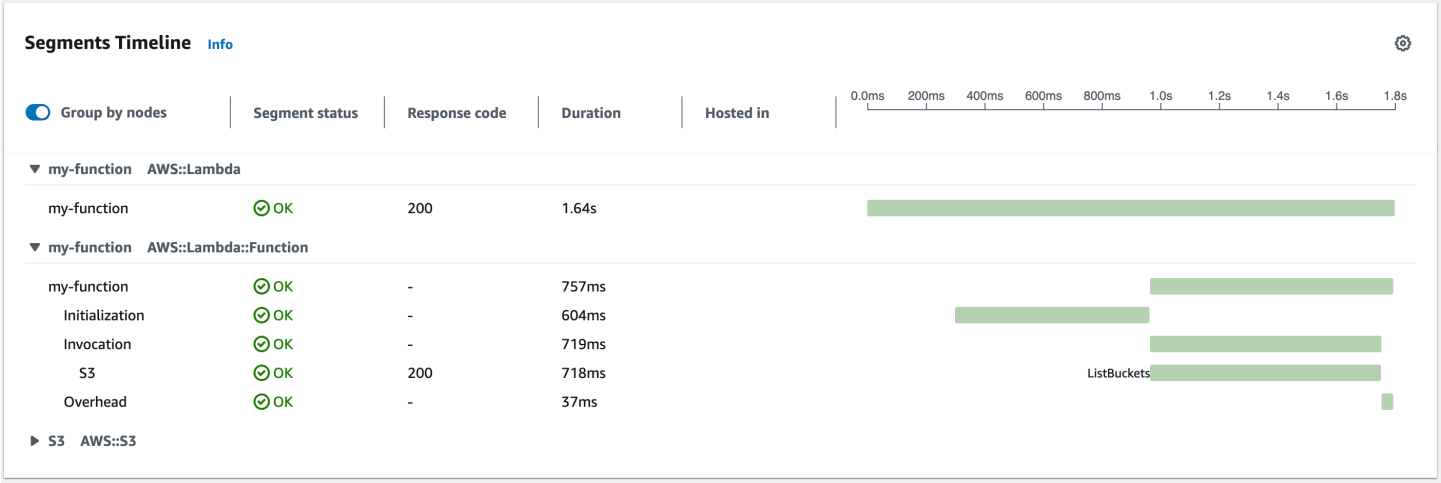
(Lambda SnapStart uniquement) Le `Restore` sous-segment indique le temps nécessaire à Lambda pour restaurer un instantané, charger le runtime (JVM) et exécuter les éventuels hooks d'exécution. `afterRestore` Le processus de restauration des instantanés peut inclure du temps consacré à des activités en dehors de la MicroVM. Cette heure est indiquée dans le sous-segment `Restore`. Le temps passé en dehors de la microVM pour restaurer un instantané ne vous est pas facturé.

Le sous-segment `Overhead` représente la phase qui se produit entre le moment où l'exécution envoie la réponse et le signal pour l'invocation suivante. Pendant ce temps, l'exécution termine toutes les tâches liées à une invocation et se prépare à geler l'environnement de test (sandbox).

> **Important**
>
> Vous pouvez utiliser le X-Ray SDK pour étendre le `Invocation` sous-segment avec des sous-segments supplémentaires pour les appels en aval, les annotations et les métadonnées. Vous ne pouvez pas accéder directement au segment de fonction ou enregistrer une tâche effectuée en dehors de la portée d'invocation du gestionnaire.

Pour plus d'informations sur les phases de l'environnement d'exécution Lambda, consultez. Environnement d’exécution

Un exemple de trace utilisant l'ancienne structure X-Ray est illustré dans le schéma suivant.

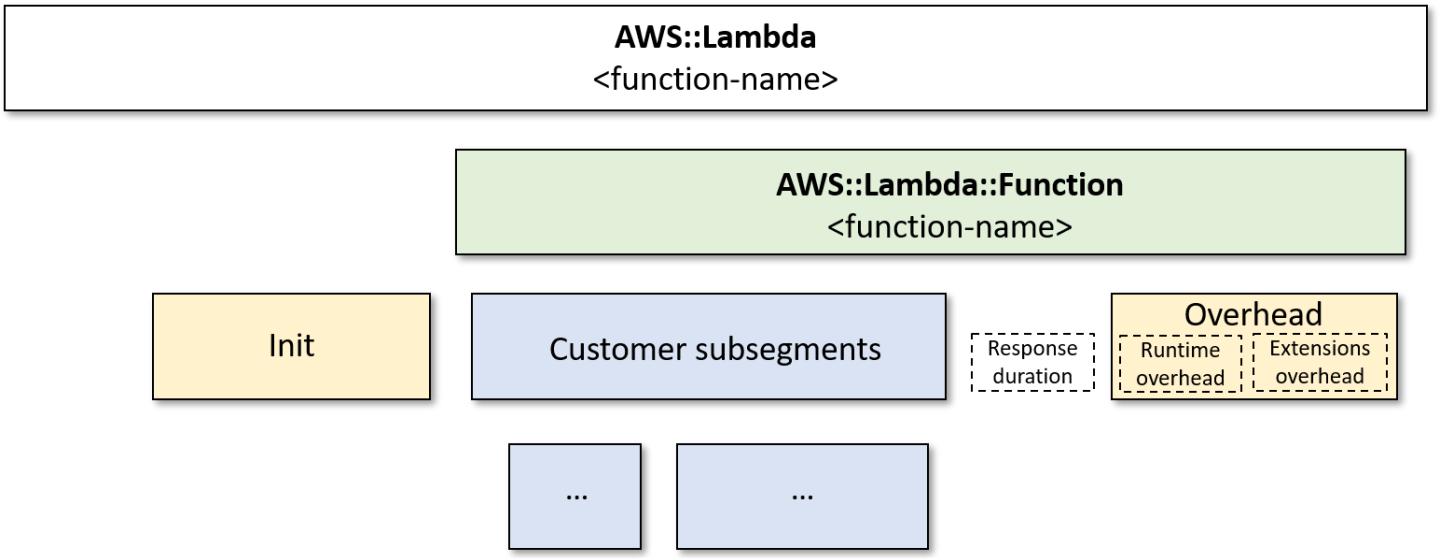



Notez les deux segments de l'exemple. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.

> **Note**
>
> Occasionnellement, vous pouvez remarquer un grand écart entre les phases d'initialisation et d'invocation de la fonction dans vos traces X-Ray. Pour les fonctions utilisant la simultanéité provisionnée, cela est dû au fait que Lambda initialise vos instances de fonction bien avant l'invocation. Pour les fonctions utilisant la simultanéité non réservée (à la demande), Lambda peut initialiser de manière proactive une instance de fonction, même s'il n'y a pas d'invocation. Visuellement, ces deux cas se manifestent par un écart de temps entre les phases d'initialisation et d'invocation.

Structure de segment AWS X-Ray Lambda de style nouveau

La nouvelle structure X-Ray du `AWS::Lambda` segment ressemble à ce qui suit :

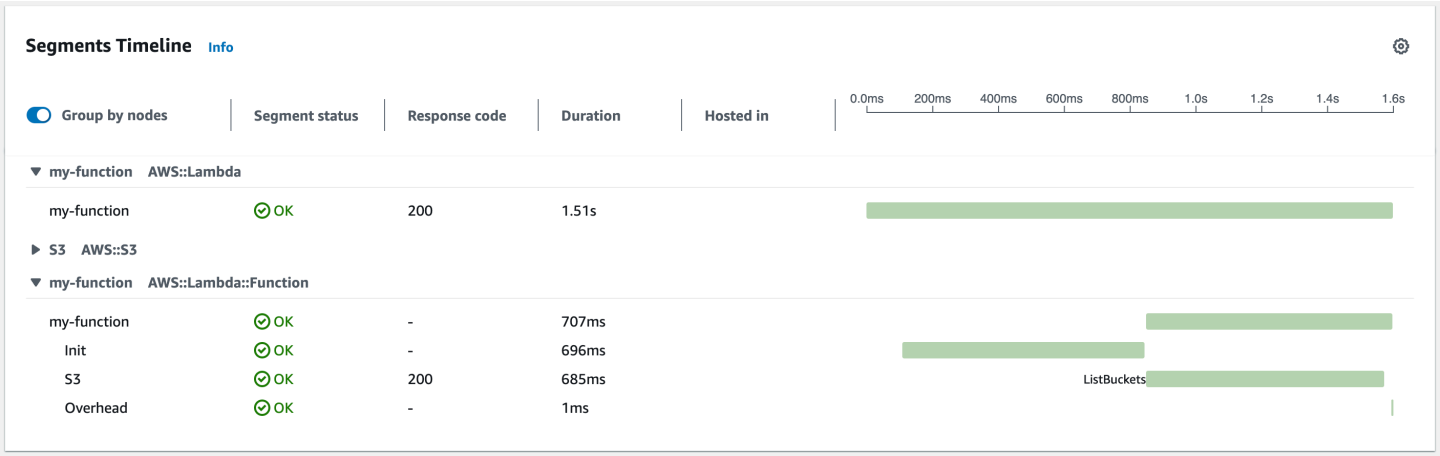



Dans ce nouveau format, le `Init` sous-segment représente la phase d'initialisation du cycle de vie de l'environnement d'exécution Lambda, comme auparavant.

Il n'existe aucun segment d'invocation dans le nouveau format. Au lieu de cela, les sous-segments de clients sont directement rattachés au `AWS::Lambda::Function` segment. Ce segment contient les métriques suivantes sous forme d'annotations :

- `aws.responseLatency`- le temps nécessaire à l'exécution de la fonction
- `aws.responseDuration`- le temps nécessaire pour transmettre la réponse au client
- `aws.runtimeOverhead`- le temps supplémentaire nécessaire à l'exécution pour terminer
- `aws.extensionOverhead`- le temps supplémentaire nécessaire pour terminer les prolongations

Un exemple de trace utilisant la nouvelle structure X-Ray est illustré dans le schéma suivant.



Notez les deux segments de l'exemple. Les deux sont nommés my-function, mais l'un a une origine de `AWS::Lambda` et l'autre a une origine de`AWS::Lambda::Function`. Si le `AWS::Lambda` segment affiche une erreur, cela signifie que le service Lambda a rencontré un problème. Si le `AWS::Lambda::Function` segment affiche une erreur, cela signifie que votre fonction a rencontré un problème.

Consultez les rubriques suivantes pour une présentation du suivi dans Lambda spécifique de chaque langage :

- Instrumentation du code Node.js dans AWS Lambda
- Instrumentation du code Python dans AWS Lambda
- Instrumentation du code Ruby dans AWS Lambda
- Instrumentation du code Java dans AWS Lambda
- Instrumentation du code Go AWS Lambda
- Instrumentation du code C# dans AWS Lambda

Pour obtenir la liste complète des services qui prennent en charge l'instrumentation active, consultez la section AWS Services pris en charge dans le Guide du AWS X-Ray développeur.

## Autorisations du rôle d'exécution

Lambda a besoin des autorisations suivantes pour envoyer des données de suivi à X-Ray. Ajoutez-les au rôle d'exécution de la fonction.

- radiographie : PutTraceSegments
- radiographie : PutTelemetryRecords

Ces autorisations sont incluses dans la politique AWSXRayDaemonWriteAccessgérée.

## Le AWS X-Ray daemon

Au lieu d'envoyer des données de suivi directement au X-RayAPI, celui-ci SDK utilise un processus daemon. Le AWS X-Ray daemon est une application qui s'exécute dans l'environnement Lambda et écoute le trafic contenant des segments et UDP des sous-segments. Il met en mémoire tampon les données entrantes et les écrit par lots sur X-Ray, ce qui a pour effet de réduire la surcharge de traitement et de mémoire requise pour suivre les invocations.

L'environnement d'exécution Lambda permet au démon d'utiliser jusqu'à 3 % de la mémoire configurée pour votre fonction ou 16 Mo, selon la valeur la plus élevée. Si votre fonction manque de mémoire pendant l'invocation, l'environnement d'exécution arrête d'abord le processus de démon pour libérer de la mémoire.

Le processus de démon étant entièrement géré par Lambda, l'utilisateur ne peut pas le configurer. Tous les segments générés par les invocations de fonction sont enregistrés dans le même compte que la fonction Lambda. Le démon ne peut pas être configuré pour les rediriger vers un autre compte.

Pour plus d'informations, consultez Démon X-Ray dans le Guide du développeur X-Ray.

## Activer le traçage actif avec le Lambda API

Pour gérer la configuration du suivi avec le AWS CLI ou AWS SDK, utilisez les API opérations suivantes :

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

L'exemple de AWS CLI commande suivant active le suivi actif sur une fonction nommée my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Le mode de suivi fait partie de la configuration spécifique de la version lorsque vous publiez une version de votre fonction. Vous ne pouvez pas modifier le mode de suivi sur une version publiée.

## Activation du suivi actif avec AWS CloudFormation

Pour activer le suivi d'une `AWS::Lambda::Function` ressource dans un AWS CloudFormation modèle, utilisez la `TracingConfig` propriété.

Example function-inline.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
```

```
      TracingConfig:
        Mode: Active
      ...
```

Pour une `AWS::Serverless::Function` ressource AWS Serverless Application Model (AWS SAM), utilisez la `Tracing` propriété.

Example template.yml – Configuration du suivi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

# Surveillez les performances des fonctions avec Amazon CloudWatch Lambda Insights

Amazon CloudWatch Lambda Insights collecte et agrège les indicateurs de performance d'exécution des fonctions Lambda et les journaux pour vos applications sans serveur. Cette page explique comment activer et utiliser Lambda Insights pour diagnostiquer des problèmes liés à vos fonctions Lambda.

Sections


- Comment Lambda Insights surveille les applications sans serveur
- Tarification
- Environnements d'exécution pris en charge
- Activation de Lambda Insights dans la console Lambda
- Activation de Lambda Insights par programme
- Utilisation du tableau de bord Lambda Insights
- Exemple de flux de travail permettant de détecter les anomalies de fonction
- Exemple de flux de travail utilisant des demandes pour dépanner une fonction
- Quelle est la prochaine étape ?


## Comment Lambda Insights surveille les applications sans serveur

CloudWatch Lambda Insights est une solution de surveillance et de dépannage pour les applications sans serveur exécutées sur. AWS Lambda La solution collecte, agrège et résume les indicateurs au niveau du système, notamment le CPU temps, la mémoire, l'utilisation du disque et du réseau. Elle collecte, agrège et résume également des informations de diagnostic telles que des démarrages à froid et des arrêts de rôle de travail Lambda pour vous aider à circonscrire des problèmes liés à vos fonctions Lambda, ainsi qu'à les résoudre rapidement.

Lambda Insights utilise une nouvelle extension CloudWatch  Lambda Insights, fournie sous forme de couche Lambda. Lorsque vous activez cette extension sur une fonction Lambda pour un environnement d'exécution pris en charge, elle collecte des métriques au niveau du système et émet un seul événement de journal des performances pour chaque appel de cette fonction Lambda. CloudWatch utilise le formatage des métriques intégré pour extraire les métriques des événements du journal. Pour plus d'informations, consultez la section Utilisation des AWS Lambda extensions.

La couche Lambda Insights étend `CreateLogStream` et `PutLogEvents` pour le groupe de journaux `/aws/lambda-insights/`.

## Tarification

Lorsque vous activez Lambda Insights pour votre fonction Lambda, Lambda Insights rapporte 8 mesures par fonction et chaque appel de fonction envoie environ 1 Ko de données de journal à. CloudWatch Vous payez uniquement les métriques et les journaux signalés pour votre fonction par Lambda Insights. Aucun frais minimum ni aucune politique d'utilisation obligatoire des services ne sont appliqués. Vous ne payez pas pour Lambda Insights si la fonction n'est pas appelée. Pour un exemple de tarification, consultez les CloudWatch tarifs Amazon.

## Environnements d'exécution pris en charge

Vous pouvez utiliser Lambda Insights avec n'importe quel runtime prenant en charge les extensions Lambda.

## Activation de Lambda Insights dans la console Lambda

Vous pouvez activer la surveillance améliorée de Lambda Insights sur des fonctions Lambda nouvelles et existantes. Lorsque vous activez Lambda Insights sur une fonction dans la console Lambda pour un runtime pris en charge, Lambda ajoute l'extension Lambda Insights en tant que couche à votre fonction, et vérifie ou tente d'attacher la stratégie `CloudWatchLambdaInsightsExecutionRolePolicy` au rôle d'exécution de votre fonction.

Pour activer Lambda Insights dans la console Lambda

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez votre fonction.
3. Choisissez l'onglet Configuration.
4. Dans le menu de gauche, choisissez Outils de surveillance et d'exploitation.
5. Dans le volet Outils de surveillance supplémentaires, choisissez Modifier.
6. Sous CloudWatch Lambda Insights, activez la surveillance améliorée.
7. Choisissez Save (Enregistrer).

## Activation de Lambda Insights par programme

Vous pouvez également activer Lambda Insights à l'aide des AWS CLI touches AWS Command Line Interface AWS Serverless Application Model (SAM) CLI AWS CloudFormation, () ou. AWS Cloud Development Kit (AWS CDK)Lorsque vous activez Lambda Insights par programmation sur une fonction pour un environnement d'exécution pris en charge, associez la `CloudWatchLambdaInsightsExecutionRolePolicy`politique CloudWatch au rôle d'exécution de votre fonction.

Pour plus d'informations, consultez Getting started with Lambda Insights dans le guide de CloudWatch l'utilisateur Amazon.

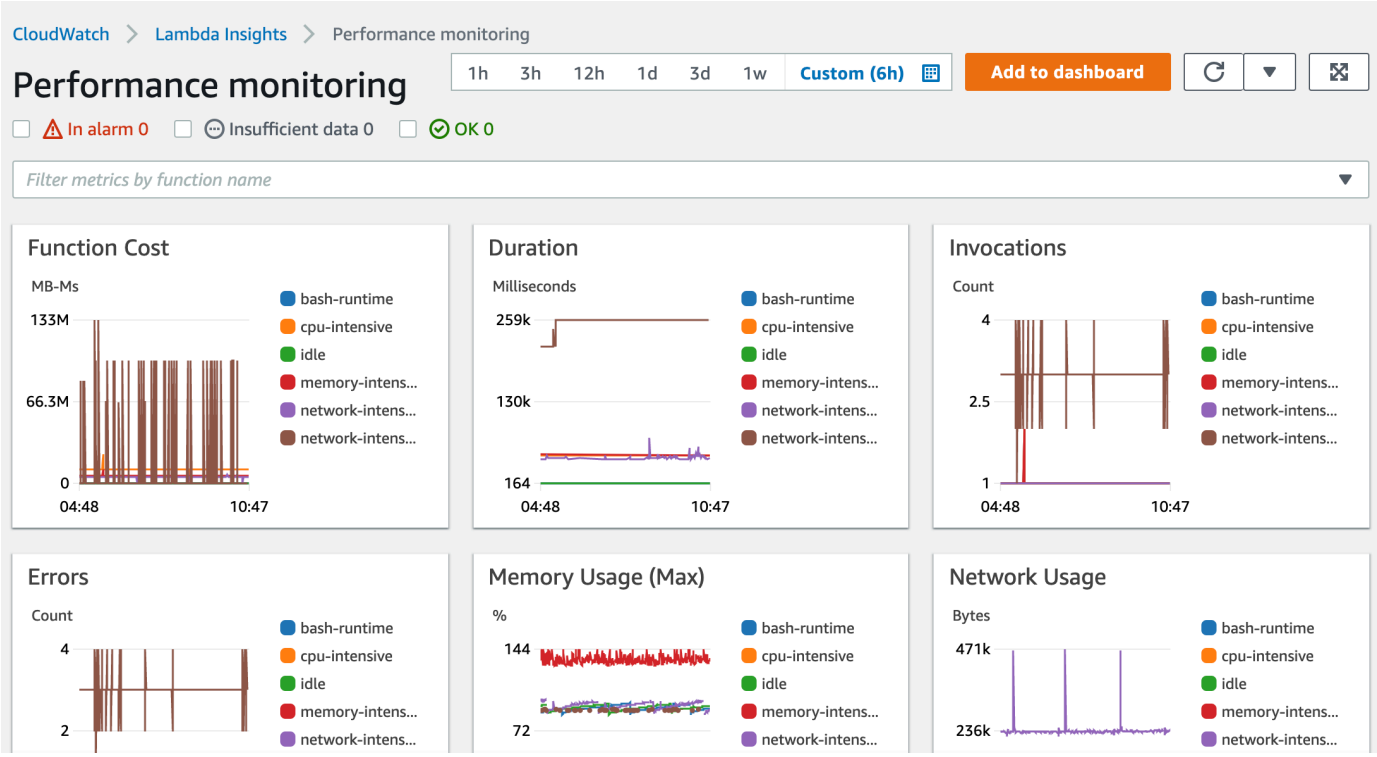
## Utilisation du tableau de bord Lambda Insights

Le tableau de bord Lambda Insights comporte deux vues dans la CloudWatch console : la vue d'ensemble multifonction et la vue monofonctionnelle. La vue d'ensemble multifonction regroupe les métriques d'exécution des fonctions Lambda dans le compte courant AWS et la région. La vue de fonction unique affiche les métriques de runtime disponibles pour une seule fonction Lambda.

Vous pouvez utiliser la vue d'ensemble multifonctions du tableau de bord Lambda Insights dans la CloudWatch console pour identifier les fonctions Lambda surutilisées ou sous-utilisées. Vous pouvez utiliser la vue à fonction unique du tableau de bord Lambda Insights dans la CloudWatch console pour résoudre les demandes individuelles.

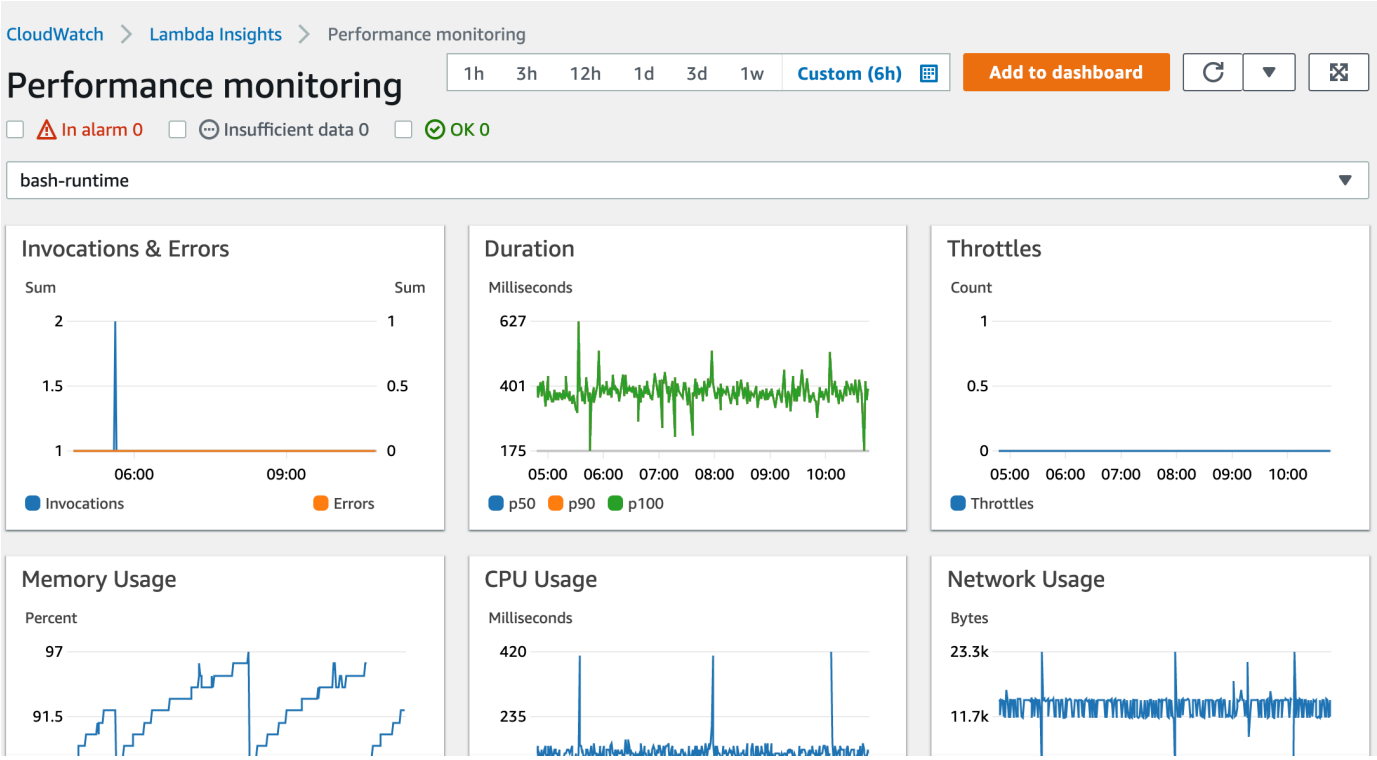
Pour afficher les métriques d'environnement d'exécution pour toutes les fonctions

1. Ouvrez la page Multifonction dans la CloudWatch console.
2. Choisissez parmi les plages de temps prédéfinies ou choisissez une plage de temps personnalisée.
3. (Facultatif) Choisissez Ajouter au tableau de bord pour ajouter les widgets à votre CloudWatch tableau de bord.




Pour afficher les métriques d'environnement d'exécution d'une fonction unique

1. Ouvrez la page Fonction unique dans la CloudWatch console.
2. Choisissez parmi les plages de temps prédéfinies ou choisissez une plage de temps personnalisée.
3. (Facultatif) Choisissez Ajouter au tableau de bord pour ajouter les widgets à votre CloudWatch tableau de bord.




Pour plus d'informations, consultez la section Création et utilisation de widgets sur les CloudWatch tableaux de bord.

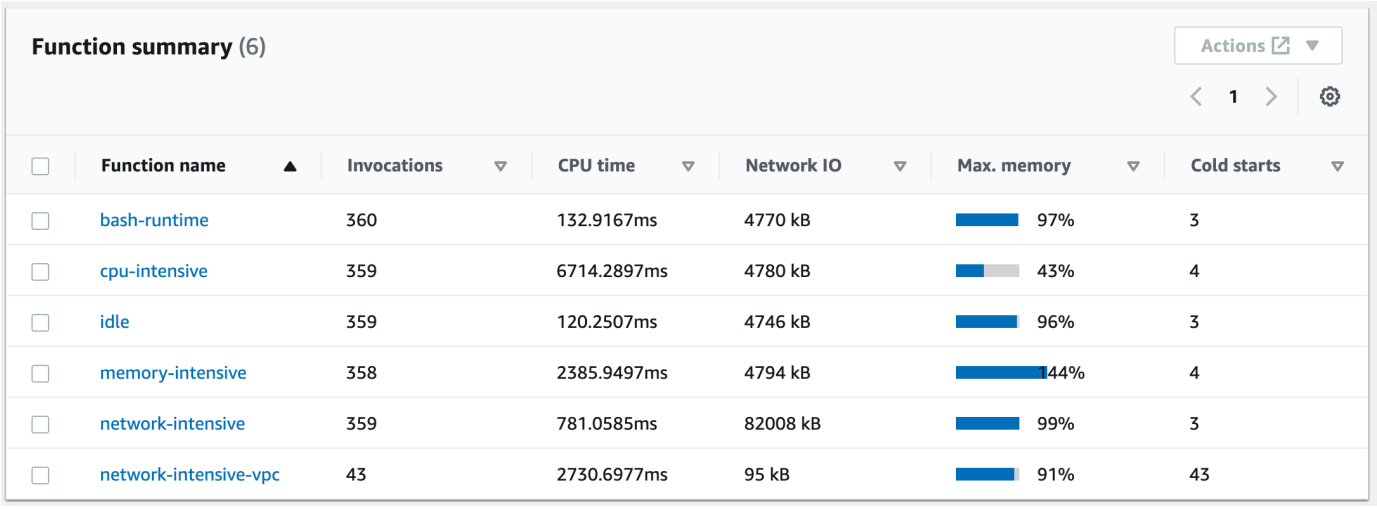
## Exemple de flux de travail permettant de détecter les anomalies de fonction

Vous pouvez utiliser la vue d'ensemble multifonction sur le tableau de bord Lambda Insights pour identifier et détecter des anomalies de mémoire de calcul liées à votre fonction. Par exemple, si la vue d'ensemble multifonction indique qu'une fonction utilise une grande quantité de mémoire, vous pouvez afficher les métriques détaillées d'utilisation de la mémoire dans le volet Memory Usage (Utilisation de la mémoire). Vous pouvez ensuite accéder au tableau de bord des métriques pour activer la détection d'anomalies ou créer une alarme.

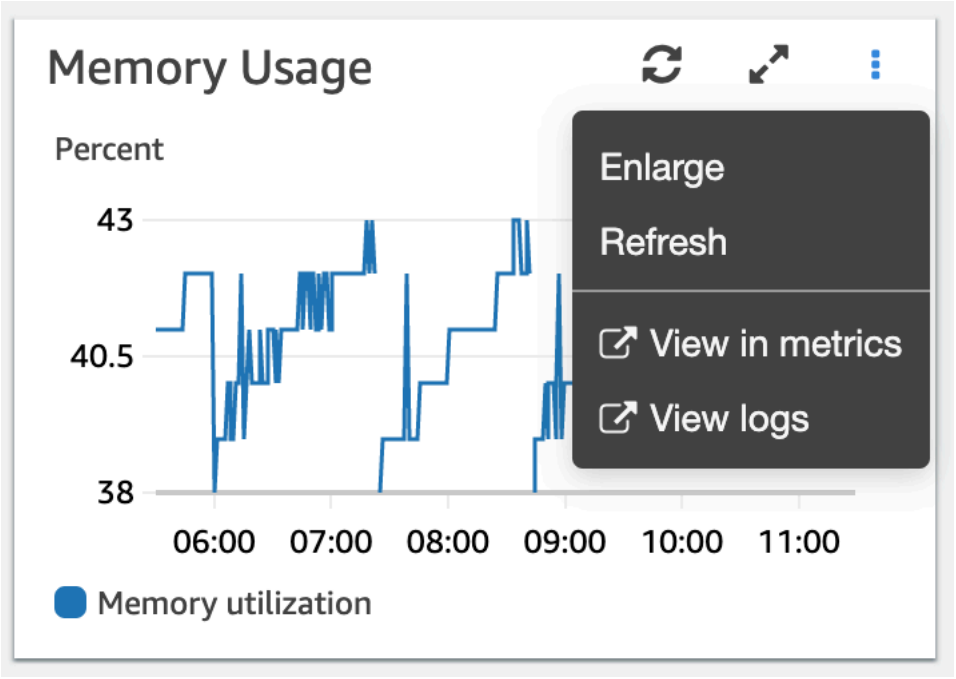
Pour activer la détection d'anomalies pour une fonction

1. Ouvrez la page Multifonction dans la CloudWatch console.
2. Sous Function summary (Récapitulatif de fonction), choisissez le nom de votre fonction.

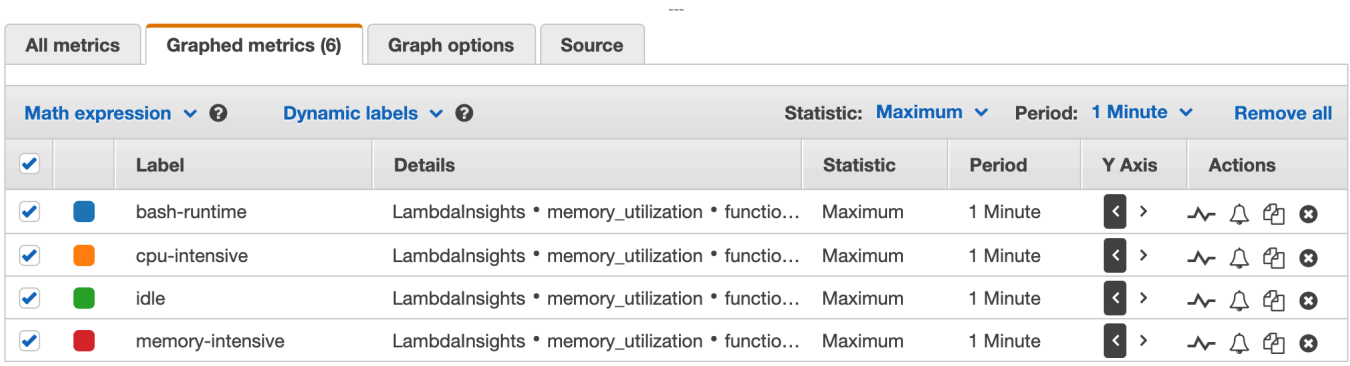
   La vue de fonction unique s'ouvre avec les métriques d'environnement d'exécution de la fonction.




3. Dans le volet Memory Usage (Utilisation de la mémoire) sélectionnez les trois points verticaux, puis choisissez View in metrics (Afficher dans les métriques) pour ouvrir le tableau de bord Metrics (Métriques).




4. Dans l'onglet Graphique des métriques, dans la colonne Actions, choisissez la première icône pour activer la détection d'anomalies pour la fonction.



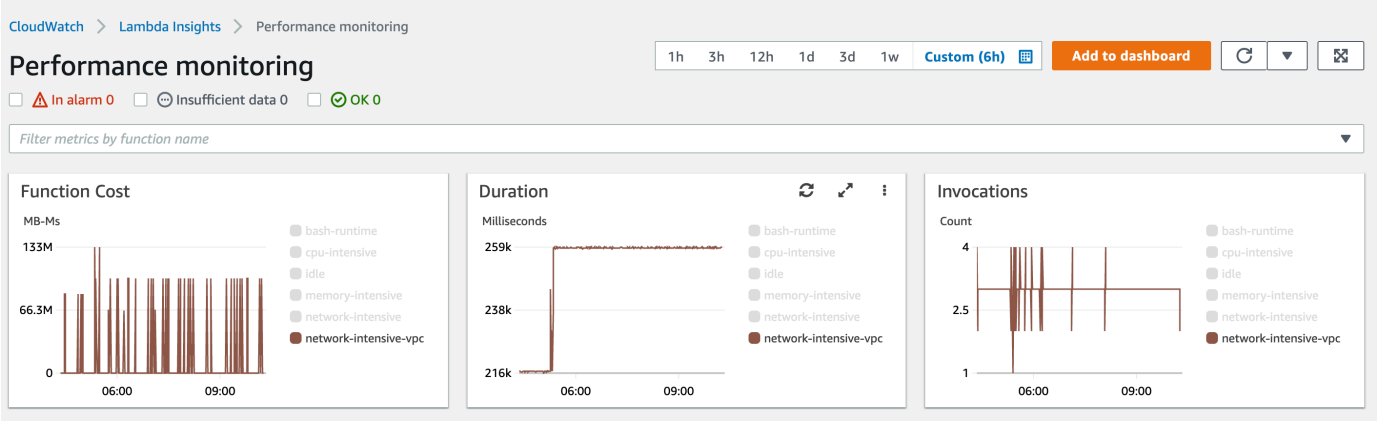

Pour plus d'informations, consultez la section Utilisation de la détection des CloudWatch anomalies.

## Exemple de flux de travail utilisant des demandes pour dépanner une fonction

Vous pouvez utiliser la vue de fonction unique sur le tableau de bord Lambda Insights pour identifier la cause première d'un pic de durée de fonction. Par exemple, si la vue d'ensemble multifonction indique une augmentation importante de la durée de la fonction, vous pouvez suspendre ou choisir chaque fonction dans le volet Durée afin de déterminer la fonction à l'origine de l'augmentation. Vous pouvez ensuite accéder à la vue de fonction unique et consulter les journaux d'application pour déterminer la cause principale.

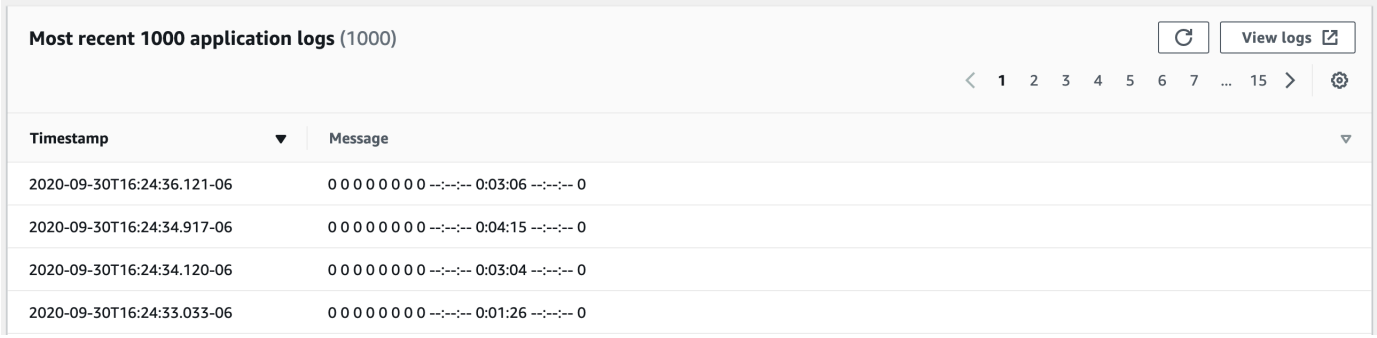
Pour exécuter des requêtes sur une fonction

1. Ouvrez la page Multifonction dans la CloudWatch console.
2. Dans le volet Durée choisissez votre fonction pour filtrer les métriques de durée.

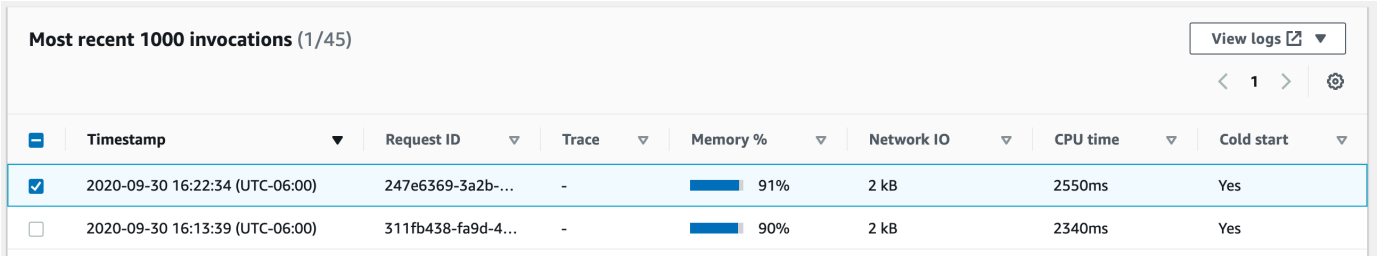



3. Ouvrez la page Fonction unique.

4. Choisissez la liste déroulante Filter metrics by function name (Filtrer les métriques par nom de fonction), puis choisissez votre fonction.
5. Pour afficher les 1000 journaux d'application les plus récents, choisissez l'onglet Application logs (Journaux d'application).
6. Examinez l'horodatage et le message pour identifier la demande d'invocation que vous souhaitez résoudre.

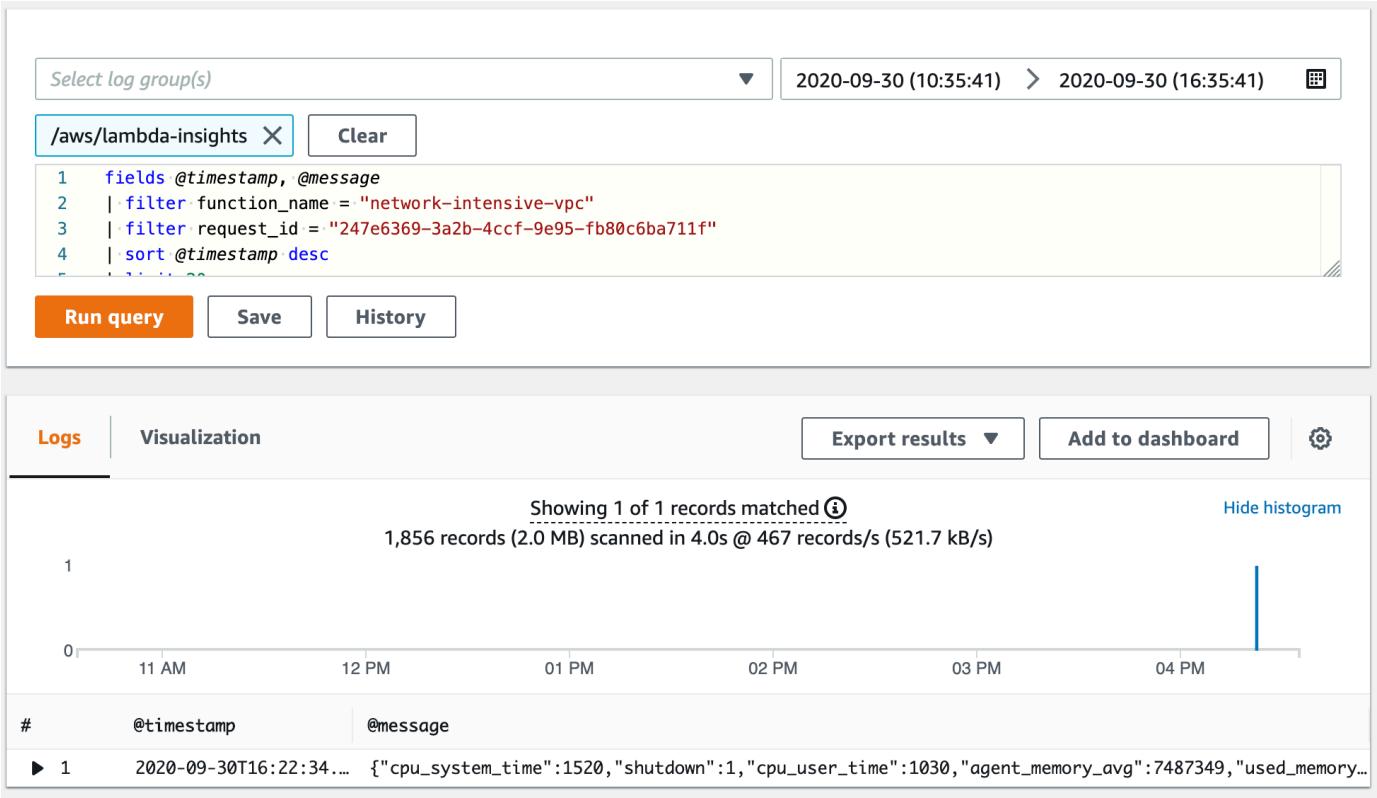



7. Pour afficher les Most recent 1000 invocations (1000 appels les plus récents), choisissez l'onglet Appels.
8. Sélectionnez l'horodatage ou le message pour la demande d'invocation que vous souhaitez résoudre.




9. Choisissez la liste déroulante Afficher les journaux, puis Afficher les journaux de performance.

   Une requête générée automatiquement pour votre fonction s'ouvre dans le tableau de bord Logs Insights.
10. Choisissez Exécuter la requête pour générer un message Journaux pour la demande d'invocation.




## Quelle est la prochaine étape ?

- Découvrez comment créer un tableau de bord CloudWatch Logs dans la section Créer un tableau de bord du guide de CloudWatch l'utilisateur Amazon.
- Découvrez comment ajouter des requêtes à un tableau de bord CloudWatch Logs dans Ajouter une requête au tableau de bord ou Exporter les résultats des requêtes dans le guide de CloudWatch l'utilisateur Amazon.

# Gestion des dépendances Lambda à l'aide de couches

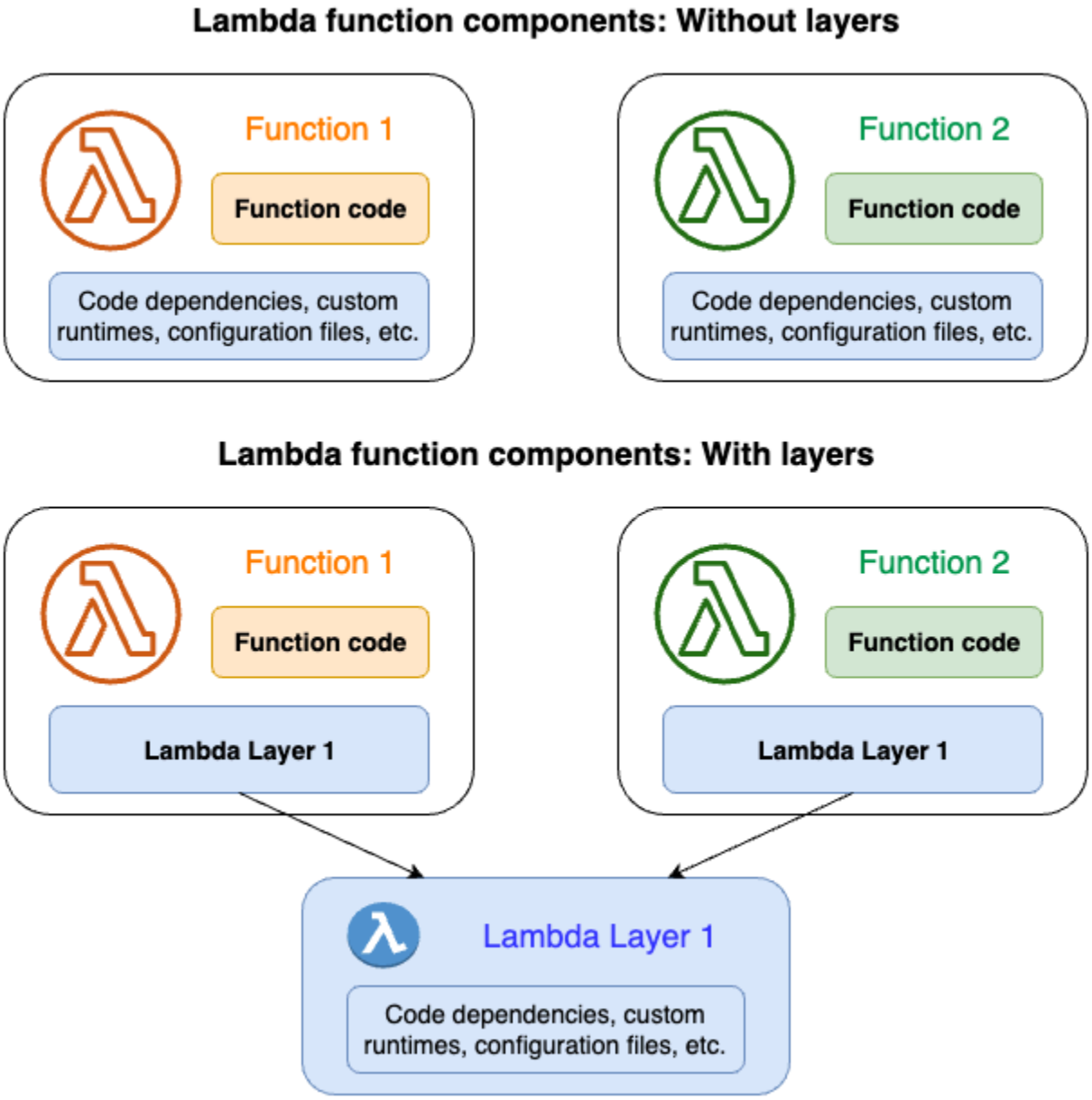
Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration.

Vous pouvez envisager d'utiliser des couches pour plusieurs raisons :

- Pour réduire la taille de vos packages de déploiement. Au lieu d'inclure toutes vos dépendances de fonctions avec votre code de fonction dans votre package de déploiement, placez-les dans une couche. Cela permet de maintenir la taille et l'organisation des packages de déploiement.
- Pour séparer la logique des fonctions de base des dépendances. Avec les couches, vous pouvez mettre à jour les dépendances de vos fonctions indépendamment du code de votre fonction, et vice versa. Cela favorise la séparation des préoccupations et vous aide à vous concentrer sur la logique de votre fonction.
- Pour partager les dépendances entre plusieurs fonctions. Après avoir créé une couche, vous pouvez l'appliquer à un certain nombre de fonctions de votre compte. Sans couches, vous devez inclure les mêmes dépendances dans chaque package de déploiement individuel.
- Pour utiliser l'éditeur de code de la console Lambda. L'éditeur de code est un outil utile pour tester rapidement les mises à jour mineures du code des fonctions. Toutefois, vous ne pouvez pas utiliser l'éditeur si la taille de votre package de déploiement est trop importante. L'utilisation de couches réduit la taille de votre package et peut débloquer l'utilisation de l'éditeur de code.

Si vous utilisez des fonctions Lambda dans Go ou Rust, nous vous déconseillons d'utiliser des couches. Pour les fonctions Go et Rust, vous fournissez votre code de fonction sous forme d'exécutable, qui inclut votre code de fonction compilé ainsi que toutes ses dépendances. Le fait de placer vos dépendances dans une couche oblige votre fonction à charger manuellement des assemblages supplémentaires pendant la phase d'initialisation, ce qui peut augmenter les temps de démarrage à froid. Pour des performances optimales pour les fonctions Go et Rust, incluez vos dépendances dans votre package de déploiement.

Le diagramme suivant illustre les différences architecturales de haut niveau entre deux fonctions qui partagent des dépendances. L'un utilise des couches Lambda, l'autre non.




Lorsque vous ajoutez une couche à une fonction, Lambda extrait le contenu de la couche dans le répertoire `/opt` de l'environnement d'exécution de votre fonction. Toutes les exécutions Lambda prises en charge en mode natif incluent des chemins vers des répertoires spécifiques dans le répertoire `/opt`. Cela permet à votre fonction d'accéder au contenu de votre couche. Pour plus d'informations sur ces chemins spécifiques et sur la manière d'empaqueter correctement vos couches, consultez the section called “Empaquetage des couches”.

Vous pouvez inclure jusqu’à cinq couches par fonction. En outre, vous ne pouvez utiliser des couches qu'avec des fonctions Lambda déployées en tant qu'archive de fichiers .zip. Pour des fonctions

définies en tant qu'image de conteneur, créez un package avec votre exécution préférée et toutes les dépendances de code lorsque vous créez l'image de conteneur. Pour plus d'informations, consultez la section Utilisation de couches et d'extensions Lambda dans des images de conteneur sur le blog AWS Compute.

Rubriques


- Utilisation des couches
- Couches et versions de couches
- Empaquetage du contenu de votre couche
- Création et suppression de couches dans Lambda
- Ajout de couches aux fonctions
- Utilisation AWS CloudFormation avec des couches
- Utilisation AWS SAM avec des couches


## Utilisation des couches

Pour créer une couche, empaquetez vos dépendances dans un fichier .zip, de la même manière que vous créer un package de déploiement normal. Plus précisément, le processus général de création et d'utilisation de couches comporte les trois étapes suivantes :

- Empaquetez d'abord le contenu de votre couche. Cela implique de créer une archive de fichiers .zip. Pour de plus amples informations, veuillez consulter the section called “Empaquetage des couches”.
- Créez ensuite la couche dans Lambda. Pour de plus amples informations, veuillez consulter the section called “Création et suppression de couches”.
- Ajoutez la couche à vos fonctions. Pour de plus amples informations, veuillez consulter the section called “Ajout de couches”.

## Couches et versions de couches

Une version de couche est un instantané immuable d'une version spécifique d'une couche. Lorsque vous créez une nouvelle couche, Lambda crée une nouvelle version de couche avec un numéro de version de 1. Chaque fois que vous publiez une mise à jour de la couche, Lambda incrémente le numéro de version et crée une nouvelle version de couche.

Chaque version de couche est identifiée par un nom de ressource Amazon unique (ARN). Lorsque vous ajoutez une couche à la fonction, vous devez spécifier la version de couche exacte que vous voulez utiliser.

# Empaquetage du contenu de votre couche

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration.

Cette section explique comment empaqueter correctement le contenu de votre couche. Pour plus d'informations conceptuelles sur les couches et les raisons pour lesquelles vous pourriez envisager de les utiliser, consultez Couches Lambda.

La première étape de la création d'une couche consiste à regrouper l'ensemble du contenu de la couche dans une archive de fichiers .zip. Parce que les fonctions Lambda s'exécutent sur Amazon Linux, le contenu de votre couche doit pouvoir être compilé et construit dans un environnement Linux.

Pour garantir que le contenu de votre couche fonctionne correctement dans un environnement Linux, nous vous recommandons de créer le contenu de votre couche à l'aide d'un outil tel que Docker ou AWS Cloud9. AWS Cloud9 est un environnement de développement intégré basé sur le cloud (IDE) qui fournit un accès intégré à un serveur Linux pour exécuter et tester du code. Pour plus d'informations, consultez Utilisation de couches Lambda pour simplifier votre processus de développement sur le blog AWS Compute.

Rubriques

- Chemins d'accès de couche pour chaque exécution Lambda

## Chemins d'accès de couche pour chaque exécution Lambda

Lorsque vous ajoutez une couche à une fonction, Lambda charge le contenu de la couche dans le répertoire `/opt` de cet environnement d'exécution. Pour chaque exécution Lambda, la variable `PATH` inclut déjà des chemins de dossiers spécifiques dans le répertoire `/opt`. Pour garantir que la `PATH` variable capte le contenu de votre couche, le fichier .zip de votre couche doit avoir ses dépendances dans les chemins de dossier suivants :

| Runtime | Chemin |
| --- | --- |
| Node.js | `nodejs/node_modules` |
| | `nodejs/node16/node_modules` (`NODE_PATH`) |

| Runtime | Chemin |
| --- | --- |
| | `nodejs/node18/node_modules` (`NODE_PATH`) |
| | `nodejs/node20/node_modules` (`NODE_PATH`) |
| Python | `python` |
| | `python/lib/` *`python3.x`* `/site-packages` (répertoires de site) |
| Java | `java/lib` (`CLASSPATH`) |
| Ruby | `ruby/gems/3.2.0` (`GEM_PATH`) |
| | `ruby/lib` (`RUBYLIB`) |
| Toutes les exécutions | `bin` (`PATH`) |
| | `lib` (`LD_LIBRARY_PATH`) |

Les exemples suivants montrent comment structurer les dossiers dans votre couche d'archive .zip.

Node.js

Example structure de fichier pour le fichier AWS X-Ray SDK pour Node.js

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Python

Example structure de fichier pour la bibliothèque de requêtes

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
```

```
                # ...
```

Ruby

Example structure de fichier pour la JSON gemme

```
json.zip
# ruby/gems/2.7.0/
                | build_info
                | cache
                | doc
                | extensions
                | gems
                | # json-2.1.0
                # specifications
                  # json-2.1.0.gemspec
```

Java

Example structure de fichier pour le JAR fichier Jackson

```
layer_content.zip
# java
    # lib
        # jackson-core-2.17.0.jar
        # <other potential dependencies>
        # ...
```

All

Example structure de fichiers pour la bibliothèque jq

```
jq.zip
# bin/jq
```

Pour obtenir des instructions spécifiques à la langue relatives à l'emballage, à la création et à l'ajout d'une couche, reportez-vous aux pages suivantes :

- Python : the section called “Couches”
- Java — the section called “Couches”

Nous vous déconseillons d'utiliser des couches pour les langues suivantes. Les pages liées contiennent plus d'informations.

- Go – the section called “Couches”
- Rust

# Création et suppression de couches dans Lambda

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration.

Cette section explique comment créer et supprimer des couches dans Lambda. Pour plus d'informations conceptuelles sur les couches et les raisons pour lesquelles vous pourriez envisager de les utiliser, consultez Couches Lambda.

Une fois que vous avez empaqueté le contenu de votre couche, l'étape suivante consiste à créer la couche dans Lambda. Cette section explique comment créer et supprimer des couches à l'aide de la console Lambda ou de l'API Lambda uniquement. Pour créer une couche à l'aide d'AWS CloudFormation, consultez the section called "Couches avec AWS CloudFormation". Pour créer une couche à l'aide d'AWS Serverless Application Model (AWS SAM), consultez the section called "Couches avec AWS SAM".

Rubriques


- Création d'une couche
- Suppression d'une version de couche


## Création d'une couche

Pour créer une couche, vous pouvez soit charger l'archive du fichier .zip depuis votre ordinateur local, soit depuis Amazon Simple Storage Service (Amazon S3). Lors de la configuration de l'environnement d'exécution pour la fonction, Lambda extrait le contenu de la couche dans le répertoire `/opt`.

Les couches peuvent avoir une ou plusieurs versions de couche. Lorsque vous créez une couche, Lambda définit la version de la couche sur version 1. Vous pouvez modifier les autorisations sur une version de couche existante à tout moment. Toutefois, pour mettre à jour le code ou apporter d'autres modifications de configuration, vous devez créer une nouvelle version de la couche.

Pour créer une couche (console)

1. Ouvrez la page Couches de la console Lambda.
2. Sélectionnez Créer un calque.
3. Sous Configuration de la couche, dans Nom, nommez votre couche.

4. (Facultatif) Dans le champ Description, saisissez une description pour le calque.
5. Pour télécharger le code de votre couche, effectuez l'une des opérations suivantes :
   - Pour charger un fichier .zip à partir de votre ordinateur, choisissez Charger un fichier .zip. Puis, sélectionnez Charger pour sélectionner votre fichier .zip local.
   - Pour charger un fichier à partir d'Amazon S3, choisissez Charger un fichier à partir d'Amazon S3. Ensuite, pour l'URL du lien Amazon S3, saisissez un lien vers le fichier.
6. (Facultatif) Pour Architectures compatibles, choisissez une valeur ou les deux valeurs. Pour plus d'informations, consultez the section called "Jeux d'instructions (ARM/x86)".
7. (Facultatif) Pour Exécutions compatibles, choisissez les exécutions avec lesquelles votre couche est compatible.
8. (Facultatif) Pour Licence, saisissez toutes les informations de licence nécessaires.
9. Choisissez Créer.

Vous pouvez également utiliser l'PublishLayerVersionAPI pour créer une couche. Par exemple, vous pouvez utiliser la commande AWS Command Line Interface (CLI) `publish-layer-version` avec un nom, une description et une archive .zip spécifiés. Les paramètres relatifs à la licence, aux exécutions compatibles et à l'architecture compatible sont facultatifs.

```
aws lambda publish-layer-version --layer-name my-layer \
    --description "My layer" \
    --license-info "MIT" \
    --zip-file fileb://layer.zip \
    --compatible-runtimes python3.10 python3.11 \
    --compatible-architectures "arm64" "x86_64"
```

Vous devez voir des résultats similaires à ce qui suit :

```
{
    "Content": {
        "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-layer-4aaa2fbb-ff77-4b0a-ad92-5b78a716a96a?
versionId=27iWyA73cCAYqyH...",
        "CodeSha256": "tv9jJO+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",
        "CodeSize": 169
    },
    "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
    "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:1",
    "Description": "My layer",
```

```
    "CreatedDate": "2023-11-14T23:03:52.894+0000",
    "Version": 1,
    "CompatibleArchitectures": [
        "arm64",
        "x86_64"
     ],
    "LicenseInfo": "MIT",
    "CompatibleRuntimes": [
        "python3.10",
        "python3.11"
    ]
}
```

Chaque fois que vous appelez `publish-layer-version`, vous créez une nouvelle version de la couche.

## Suppression d'une version de couche

Pour supprimer une version de couche, utilisez l'DeleteLayerVersionAPI. Par exemple, vous pouvez utiliser la commande CLI `delete-layer-version` avec le nom et la version de la couche spécifiés.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Lorsque vous supprimez une version de couche, vous ne pouvez plus configurer de fonction Lambda pour l'utiliser. En revanche, toute fonction qui utilise déjà la version continue d'y avoir accès. De plus, Lambda ne réutilise jamais les numéros de version pour le nom d'une couche.

# Ajout de couches aux fonctions

Une couche Lambda est une archive de fichier .zip qui contient du code ou des données supplémentaires. Les couches contiennent généralement des dépendances de bibliothèque, une exécution personnalisée, ou des fichiers de configuration.

Cette section explique comment ajouter une couche à une fonction Lambda. Pour plus d'informations conceptuelles sur les couches et les raisons pour lesquelles vous pourriez envisager de les utiliser, consultez Couches Lambda.

Avant de pouvoir configurer une fonction Lambda pour utiliser une couche, vous devez :

- Empaqueter le contenu de votre couche
- Créer une couche dans Lambda
- Assurez-vous que vous êtes autorisé à appeler la version GetLayerVersionAPIsur la couche. Pour les fonctions de votre Compte AWS, vous devez disposer de cette autorisation dans votre politique utilisateur. Pour utiliser une couche dans un autre compte, le propriétaire de ce compte doit accorder l'autorisation à votre compte dans une stratégie basée sur les ressources. Pour obtenir des exemples, consultez the section called "Accès aux couches pour les autres comptes".

Vous pouvez ajouter jusqu'à cinq couches à une fonction Lambda. La taille totale décompressée de la fonction avec toutes les couches ne peut pas dépasser le quota de taille de package de déploiement décompressé de 250 Mo. Pour de plus amples informations, veuillez consulter Quotas Lambda.

Vos fonctions peuvent continuer à utiliser n'importe quelle version de couche que vous avez déjà ajoutée, même après la suppression de cette version de couche ou après la révocation de votre autorisation d'accès à la couche. Toutefois, vous ne pouvez pas créer une nouvelle fonction qui utilise une version de couche supprimée.

> **Note**
>
> Assurez-vous que les couches que vous ajoutez à une fonction sont compatibles avec l'exécution et l'architecture de l'ensemble des instructions de la fonction.

Pour ajouter une couche à une fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez la fonction à configurer.
3. Sous Layers (Couches), choisissez Add a layer (Ajouter une couche).
4. Sous Choisir une couche, choisissez une source de couche :

    a. Pour les sources de couches couches AWS  ou Couches personnalisées, choisissez une couche dans le menu déroulant. Sous Version, choisissez une version de couche dans le menu déroulant.

    b. Pour Spécifier une source de ARN couche, entrez un ARN dans la zone de texte et choisissez Vérifier. Choisissez ensuite Ajouter.

L'ordre dans lequel vous ajoutez les couches correspond à l'ordre dans lequel Lambda fusionne ultérieurement le contenu de la couche dans l'environnement d'exécution. Vous pouvez modifier l'ordre de fusion des couches à l'aide de la console.

Pour mettre à jour l'ordre de fusion des couches pour votre fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez la fonction à configurer.
3. Sous Layers (Couches), choisissez Edit (Modifier).
4. Choisissez l'une des couches.
5. Choisissez Merge earlier (Fusionner plus tôt) ou Merge later (Fusionner plus tard) pour ajuster l'ordre des couches.
6. Choisissez Save (Enregistrer).

Les couches sont versionnées. Le contenu de chaque version de couche est immuable. Le propriétaire d'une couche peut publier une nouvelle version de la couche pour fournir du contenu mis à jour. Vous pouvez utiliser la console pour mettre à jour la version de couche associée à vos fonctions.

Pour mettre à jour les versions des couches de votre fonction (console)

1. Ouvrez la page Couches de la console Lambda.
2. Choisissez la couche pour laquelle vous voulez mettre à jour la version.

3. Choisissez l'onglet Fonctions utilisant cette version.
4. Choisissez les fonctions que vous voulez modifier, puis cliquez sur Modifier.
5. Pour Version de la couche, choisissez la version de la couche à modifier.
6. Choisissez Update functions (Mettre à jour les fonctions).

Vous ne pouvez pas mettre à jour les versions des couches fonctionnelles d'un AWS compte à l'autre.

Rubriques


- Accès au contenu de la couche à partir de la fonction
- Recherche d'informations sur la couche


## Accès au contenu de la couche à partir de la fonction

Si la fonction Lambda comprend des couches, Lambda extrait le contenu de la couche dans le répertoire `/opt` dans l'environnement d'exécution de la fonction. Lambda extrait les couches dans l'ordre (bas à élevé) répertorié par la fonction. Lambda fusionne les dossiers portant le même nom. Si le même fichier apparaît dans plusieurs couches, la fonction utilise la version de la dernière couche extraite.

Chaque exécution Lambda ajoute des dossiers de répertoire `/opt` spécifiques à la variable `PATH`. Votre code de fonction peut accéder au contenu de la couche sans avoir à spécifier le chemin d'accès. Pour plus d'informations sur les paramètres de chemin d'accès dans l'environnement d'exécution Lambda, consultez the section called "Variables d'environnement d'exécution définies".

Reportez-vous à the section called "Chemins d'accès de couche pour chaque exécution Lambda " pour savoir où inclure vos bibliothèques lors de la création d'une couche.

Si vous utilisez une exécution Node.js ou Python, vous pouvez utiliser l'éditeur de code intégré à la console Lambda. Vous devriez pouvoir importer n'importe quelle bibliothèque que vous avez ajoutée en tant que couche à la fonction en cours.

## Recherche d'informations sur la couche

Pour trouver dans votre compte des couches compatibles avec le temps d'exécution de votre fonction, utilisez le ListLayersAPI. Par exemple, vous pouvez utiliser la commande suivante `list-layers` AWS Command Line Interface (CLI) :

```
aws lambda list-layers --compatible-runtime python3.9
```

Vous devez voir des résultats similaires à ce qui suit :

```
{
    "Layers": [
        {
            "LayerName": "my-layer",
            "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
            "LatestMatchingVersion": {
                "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
                "Version": 2,
                "Description": "My layer",
                "CreatedDate": "2023-11-15T00:37:46.592+0000",
                "CompatibleRuntimes": [
                    "python3.9",
                    "python3.10",
                    "python3.11",
                ]
            }
        }
    ]
}
```

Pour répertorier toutes les couches de votre compte, ignorez l'option `--compatible-runtime`. Les détails de la réponse indiquent la dernière version de chaque couche.

Vous pouvez également obtenir la dernière version d'une couche à l'aide du ListLayerVersionsAPI. Par exemple, vous pouvez utiliser la `list-layer-versions` CLI commande suivante :

```
aws lambda list-layer-versions --layer-name my-layer
```

Vous devez voir des résultats similaires à ce qui suit :

```
{
    "LayerVersions": [
        {
            "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
            "Version": 2,
```

```
            "Description": "My layer",
            "CreatedDate": "2023-11-15T00:37:46.592+0000",
            "CompatibleRuntimes": [
                "java11"
            ]
        },
        {
            "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:1",
            "Version": 1,
            "Description": "My layer",
            "CreatedDate": "2023-11-15T00:27:46.592+0000",
            "CompatibleRuntimes": [
                "java11"
            ]
        }
    ]
}
```

# Utilisation AWS CloudFormation avec des couches

Vous pouvez l'utiliser AWS CloudFormation pour créer une couche et l'associer à votre fonction Lambda. L'exemple de modèle suivant crée une couche nommée `my-lambda-layer` et l'attache à la fonction Lambda à l'aide de la propriété Couches.

```
---
Description: CloudFormation Template for Lambda Function with Lambda Layer
Resources:
  MyLambdaLayer:
    Type: AWS::Lambda::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      Content:
        S3Bucket: amzn-s3-demo-bucket
        S3Key: my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-lambda-function
      Runtime: python3.9
      Handler: index.handler
      Timeout: 10
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Layers:
        - !Ref MyLambdaLayer
```

# Utilisation AWS SAM avec des couches

Vous pouvez utiliser le AWS Serverless Application Model (AWS SAM) pour automatiser la création de couches dans votre application. Le type de ressource `AWS::Serverless::LayerVersion` crée une version de couche que vous pouvez référencer à partir de la configuration de votre fonction Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: AWS SAM Template for Lambda Function with Lambda Layer

Resources:
  MyLambdaLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      ContentUri: s3://amzn-s3-demo-bucket/my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: MyLambdaFunction
      Runtime: python3.9
      Handler: app.handler
      CodeUri: s3://amzn-s3-demo-bucket/my-function
      Layers:
        - !Ref MyLambdaLayer
```

# Augmentez les fonctions Lambda à l'aide des extensions Lambda

Vous pouvez utiliser des extensions Lambda pour augmenter vos fonctions Lambda. Par exemple, utilisez des extensions Lambda pour intégrer des fonctions à vos outils de surveillance, d'observabilité, de sécurité et de gouvernance préférés. Vous pouvez choisir parmi un vaste éventail d'outils fournis par des partenaires AWS Lambda, ou créer vos propres extensions Lambda.

Lambda prend en charge les extensions externes et internes. Une extension externe s'exécute en tant que processus indépendant dans l'environnement d'exécution et continue à s'exécuter une fois l'invocation de fonction entièrement traité. Étant donné que les extensions s'exécutent sous forme de processus distincts, vous pouvez les écrire dans un langage différent de la fonction. Tous les Environnements d'exécution (runtimes) Lambda prennent en charge des extensions.

Une extension interne s'exécute dans le cadre du processus de runtime. Votre fonction accède aux extensions internes via des scripts encapsuleurs ou des mécanismes internes tels que `JAVA_TOOL_OPTIONS`. Pour de plus amples informations, veuillez consulter Modification de l'environnement d'exécution.

Vous pouvez ajouter des extensions à une fonction à l'aide de la console Lambda, du AWS Command Line Interface (AWS CLI) ou des services et outils d'infrastructure en tant que code (IaC) tels que AWS CloudFormation, AWS Serverless Application Model (AWS SAM) et Terraform.

Vous êtes facturé pour le temps d'exécution que l'extension consomme (par incréments de 1 ms). Aucun coût n'est facturé pour l'installation de vos propres extensions. Pour obtenir des informations sur la tarification des extensions, consultez Tarification AWS Lambda. Pour obtenir des informations sur la tarification des extensions de partenaires, consultez les sites web de ces partenaires. Consultez the section called "Partenaires des extensions" pour obtenir la liste des extensions partenaires officielles.

Pour obtenir un didacticiel sur les extensions et la manière de les utiliser avec vos fonctions Lambda, consultez l'Atelier sur les extensions AWS Lambda.

Rubriques

- Environnement d'exécution
- Impact sur les performances et les ressources

- Autorisations
- Configuration des extensions Lambda
- AWS Lambda partenaires d'extensions
- Utilisation des extensions Lambda API pour créer des extensions
- Accès aux données de télémétrie en temps réel pour les extensions à l'aide de la télémétrie API

# Environnement d’exécution

Lambda invoque votre fonction dans un environnement d’exécution qui fournit un environnement d’exécution sécurisé et isolé. L’environnement d’exécution gère les ressources requises pour exécuter votre fonction et prend en charge le cycle de vie pour l’exécution de la fonction et les extensions associées à votre fonction.

Le cycle de vie de l’environnement d’exécution comprend les phases suivantes :

- **`Init`** : au cours de cette phase, Lambda crée ou libère un environnement d’exécution avec les ressources configurées, télécharge le code pour la fonction et toutes les couches, initialise les extensions, initialise l’exécution et exécute le code d’initialisation de la fonction (code en dehors du gestionnaire principal). La phase `Init` se produit soit lors de la première invocation, soit avant invocations de fonctions si vous avez activé la simultanéité approvisionnée.

  La phase `Init` est fractionnée en trois sous-phases : `Extension init`, `Runtime init` et `Function init`. Ces sous-phases garantissent que toutes les extensions et l’exécution accomplissent leurs tâches de configuration avant l’exécution du code de la fonction.

  Lorsque Lambda SnapStart est activé, la phase `Init` se produit lorsque vous publiez une version de la fonction. Lambda enregistre un instantané de l’état de la mémoire et du disque de l’environnement d’exécution initialisé, fait persister l’instantané chiffré et le met en cache pour un accès à faible latence. Si vous avez un hook d’exécution `beforeCheckpoint`, le code s’exécute à la fin de la phase `Init`.
- **`Restore`**(SnapStart uniquement) : Lorsque vous appelez une SnapStartfonction pour la première fois et que celle-ci prend de l'ampleur, Lambda reprend les nouveaux environnements d'exécution à partir de l'instantané persistant au lieu d'initialiser la fonction à partir de zéro. Si vous avez un hook d’exécution `afterRestore()`, le code s’exécute à la fin de la phase `Restore`. Vous êtes facturé pour la durée des hooks d’exécution `afterRestore()`. Le runtime (JVM) doit se charger et les hooks `afterRestore()` d'exécution doivent se terminer dans le délai imparti (10

secondes). Sinon, vous obtiendrez un SnapStartTimeoutException. Lorsque la phase `Restore` se termine, Lambda invoque le gestionnaire de fonction (Phase d'invocation).

- **`Invoke`** : au cours de cette phase, Lambda invoque le gestionnaire de la fonction. Une fois l'exécution de la fonction terminée, Lambda se prépare à gérer une autre invocation de fonction.
- **`Shutdown`**: cette phase se déclenche si la fonction Lambda ne reçoit aucune invocation pendant un certain temps. Au cours de la phase `Shutdown`, Lambda arrête l'exécution, alerte les extensions pour les laisser s'arrêter proprement, puis supprime l'environnement. Lambda envoie à chaque extension un événement `Shutdown` indiquant que l'environnement est sur le point d'être arrêté.

Au cours de la phase `Init`, Lambda extrait des couches contenant des extensions dans le répertoire `/opt` dans l'environnement d'exécution. Lambda recherche des extensions dans le répertoire `/opt/extensions/`, interprète chaque fichier comme un fichier d'amorçage exécutable pour le lancement de l'extension, puis démarre toutes les extensions en parallèle.

## Impact sur les performances et les ressources

La taille des extensions de votre fonction compte dans la limite de taille du package de déploiement. Pour une archive de fichiers .zip, la taille totale après décompression de la fonction et de toutes les extensions ne peut pas dépasser la limite de taille du package de déploiement fixée à 250 Mo.

Les extensions peuvent avoir un impact sur les performances de votre fonction car elles partagent des ressources fonctionnelles telles que CPU la mémoire et le stockage. Par exemple, si une extension effectue des opérations intensives de calcul, il se peut que vous observiez une augmentation de la durée d'exécution de la fonction.

Chaque extension doit terminer son initialisation avant que Lambda invoque la fonction. Par conséquent, une extension qui consomme un temps d'initialisation important peut augmenter la latence de l'invocation de la fonction.

Pour mesurer le temps supplémentaire que prend l'extension après l'exécution de la fonction, vous pouvez utiliser la métrique de fonction `PostRuntimeExtensionsDuration`. Pour mesurer l'augmentation de la mémoire utilisée, vous pouvez utiliser la métrique `MaxMemoryUsed`. Pour comprendre l'impact d'une extension spécifique, vous pouvez exécuter différentes versions de vos fonctions côte à côte.

# Autorisations

Les extensions ont accès aux mêmes ressources que les fonctions. Étant donné que les extensions sont exécutées dans le même environnement que la fonction, les autorisations sont partagées entre la fonction et l’extension.

Pour une archive de fichier .zip, vous pouvez créer un AWS CloudFormation modèle afin de simplifier la tâche consistant à associer la même configuration d'extension, y compris les autorisations AWS Identity and Access Management (IAM), à plusieurs fonctions.

# Configuration des extensions Lambda

## Configuration des extensions (archive de fichiers .zip)

Vous pouvez ajouter une extension à votre fonction en tant que couche Lambda. L'utilisation de couches vous permet de partager des extensions au sein de votre organisation ou avec l'ensemble de la communauté des développeurs Lambda. Vous pouvez ajouter une ou plusieurs extensions à une couche. Vous pouvez enregistrer jusqu'à 10 extensions pour une fonction.

Ajoutez l'extension à votre fonction en utilisant la même méthode que pour n'importe quelle couche. Pour plus d'informations, consultez Couches Lambda.

Ajout d'une extension à votre fonction (console)

1. Ouvrez la page Functions (Fonctions) de la console Lambda.
2. Choisissez une fonction.
3. Sélectionnez l'onglet Code s'il n'est pas déjà sélectionné.
4. Sous Layers (Couches), sélectionnez Edit (Modifier).
5. Pour Choose a layer (Choisir une couche), sélectionnez Specify an ARN (Spécifier un nom ARN).
6. Pour Specify an ARN (Spécifier un ARN), entrez l'Amazon Resource Name (ARN) d'une couche d'extension.
7. Choisissez Ajouter.

## Utilisation des extensions dans les images de conteneur

Vous pouvez ajouter des extensions à votre image de conteneur. Le paramètre d'image de conteneur ENTRYPOINT détermine le processus principal de la fonction. Configurez le paramètre ENTRYPOINT dans le Dockerfile ou en tant que remplacement dans la configuration de la fonction.

Vous pouvez exécuter plusieurs traitements au sein d'un conteneur. Lambda gère le cycle de vie du traitement principal et tous les traitements supplémentaires. Lambda utilise l'API des extensions pour gérer le cycle de vie des extensions.

### Exemple : ajout d'une extension externe

Une extension externe s'exécute dans un processus distinct de la fonction Lambda. Lambda démarre un traitement pour chaque extension dans le répertoire `/opt/extensions/`. Lambda utilise l'API

d'extensions pour gérer le cycle de vie des extensions. Une fois l'exécution de la fonction terminée, Lambda envoie un événement `Shutdown` à chaque extension externe.

Example d'ajouter une extension externe à une image de base Python

```
FROM public.ecr.aws/lambda/python:3.11

# Copy and install the app
COPY /app /app
WORKDIR /app
RUN pip install -r requirements.txt

# Add an extension from the local directory into /opt
ADD my-extension.zip /opt
CMD python ./my-function.py
```

## Étapes suivantes

Pour de plus amples informations sur les extensions, nous vous recommandons les ressources suivantes :

- Pour consulter un exemple pratique de base, consultez Building Extensions for AWS Lambda sur le blog AWS Compute.
- Pour plus d'informations sur les extensions fournies par les partenaires AWS Lambda, consultez Introducing AWS Lambda Extensions sur le blog AWS Compute.
- Pour consulter les exemples d'extensions et de scripts wrapper disponibles, consultez la section AWS LambdaExtensions du GitHub référentiel AWS Samples.

# AWS Lambda partenaires d'extensions

AWS Lambda s'est associé à plusieurs entités tierces pour fournir des extensions à intégrer à vos fonctions Lambda. La liste suivante détaille les extensions tierces qui sont prêtes à être utilisées à tout moment.

- AppDynamics : fournit une instrumentation automatique des fonctions Node.js ou Python Lambda, offrant une visibilité et des alertes sur les performances des fonctions.
- Axiom— Fournit des tableaux de bord pour surveiller les performances de la fonction Lambda et les mesures agrégées au niveau du système.
- Check Point CloudGuard : solution d'environnement d'exécution basé sur les extensions, qui offre une sécurité complète du cycle de vie pour les applications sans serveur.
- Datadog : offre une visibilité complète en temps réel pour vos applications sans serveur grâce à l'utilisation de métriques, de traces et de journaux.
- Dynatrace : offre une visibilité sur les traces et les métriques, et exploite l'IA pour la détection automatisée des erreurs et l'analyse des causes profondes sur l'ensemble de la pile d'applications.
- Elastic— Assure la surveillance des performances des applications (APM) pour identifier et résoudre les problèmes à l'origine à l'aide de traces, de métriques et de journaux corrélés.
- Epsagon : écoute les événements d'appel, stocke les traces et les envoie en parallèle aux exécutions de fonctions Lambda.
- Fastly— Protège vos fonctions Lambda contre les activités suspectes, telles que les attaques par injection, le piratage de compte via le bourrage d'informations d'identification, les robots malveillants et les abus. API
- HashiCorp Vault : gère les secrets et les met à la disposition des développeurs pour qu'ils les utilisent dans le code de fonction, sans rendre compte aux fonctions Vault.
- Honeycomb : outil d'observabilité pour déboguer votre pile d'applications.
- Lumigo : profile les appels de fonctions Lambda et collecte des métriques pour résoudre les problèmes dans les environnements sans serveur et de microservice.
- New Relic : exécutée avec les fonctions Lambda, collecte, améliore et transporte automatiquement la télémétrie vers la plateforme d'observabilité unifiée de New Relic.
- Sedai : une plateforme autonome de gestion du cloud, alimentée par l'IA/ML, qui offre une optimisation continue aux équipes d'exploitation du cloud afin de maximiser les économies, les performances et la disponibilité du cloud à grande échelle.
- Sentry : diagnostique, corrige et optimise les performances des fonctions Lambda.

- Site24x7 : permet une observabilité en temps réel dans vos environnements Lambda.
- Splunk : collecte des métriques haute résolution à faible latence pour une surveillance efficiente et efficace des fonctions Lambda.
- Sumo Logic : offre une visibilité sur l'état et les performances des applications sans serveur.
- Thundra : fournit des rapports de télémétrie asynchrones, tels que des traces, des métriques et des journaux.
- Salt Security — Simplifie la gouvernance des API postures et API la sécurité des fonctions Lambda grâce à une configuration automatisée et à une prise en charge de divers environnements d'exécution.

## AWS extensions gérées

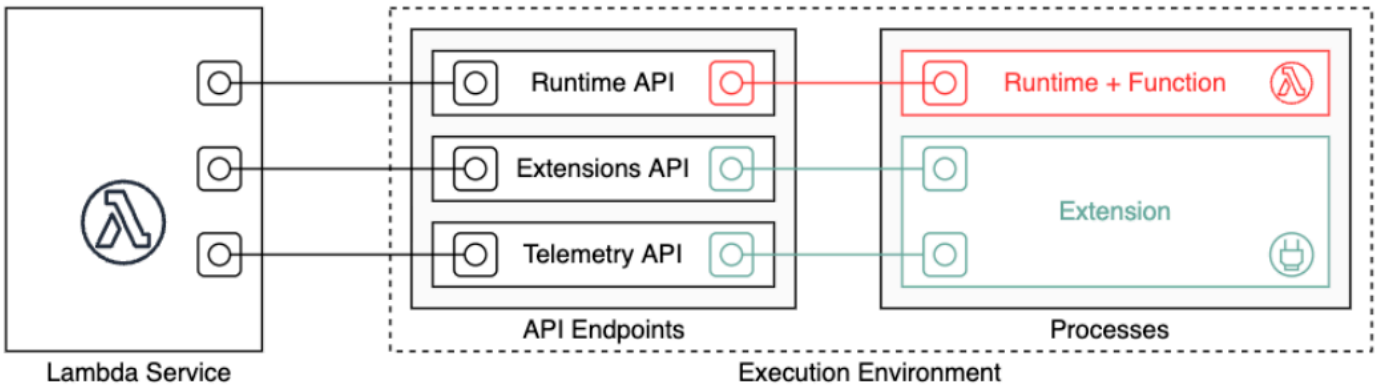
AWS fournit ses propres extensions gérées, notamment :

- AWS AppConfig— Utilisez des indicateurs de fonctionnalité et des données dynamiques pour mettre à jour vos fonctions Lambda. Vous pouvez également utiliser cette extension pour mettre à jour d'autres configurations dynamiques, telles que la limitation et le réglage des opérations.
- Amazon CodeGuru Profiler — Améliore les performances des applications et réduit les coûts en identifiant la ligne de code la plus coûteuse d'une application et en fournissant des recommandations pour améliorer le code.
- CloudWatch Lambda Insights — Surveillez, dépannez et optimisez les performances de vos fonctions Lambda grâce à des tableaux de bord automatisés.
- AWS Distro for OpenTelemetry (ADOT) — Permet aux fonctions d'envoyer des données de AWS suivi à des services de surveillance tels que AWS X-Ray et à des destinations compatibles OpenTelemetry telles que Honeycomb et Lightstep.
- AWS Paramètres et secrets : permet aux clients de récupérer en toute sécurité les paramètres de AWS Systems Manager Parameter Store et les secrets de AWS Secrets Manager.

Pour plus d'exemples d'extensions supplémentaires et de projets de démonstration, voir Extensions AWS Lambda.

# Utilisation des extensions Lambda API pour créer des extensions

Les auteurs de fonctions Lambda utilisent des extensions pour intégrer Lambda avec leurs outils de prédilection en matière de surveillance, d'observabilité, de sécurité et de gouvernance. Les auteurs de fonctions peuvent utiliser des extensions issues de AWS projets open source, de AWS partenaires et de projets open source. Pour plus d'informations sur l'utilisation des extensions, consultez la section Présentation des AWS Lambda extensions sur le blog AWS Compute. Cette section décrit comment utiliser les extensions LambdaAPI, le cycle de vie de l'environnement d'exécution Lambda et la référence des extensions Lambda. API




En tant qu'auteur d'extensions, vous pouvez utiliser les extensions Lambda API pour vous intégrer en profondeur dans l'environnement d'exécution Lambda. Votre extension peut enregistrer les événements du cycle de vie de la fonction et de l'environnement d'exécution. En réponse à ces événements, vous pouvez démarrer de nouveaux processus, exécuter une logique, ainsi que contrôler et orienter toutes les phases du cycle de vie Lambda : initialisation, appel et arrêt. En outre, vous pouvez utiliser les journaux d'exécution API pour recevoir un flux de journaux.

Une extension s'exécute en tant que processus indépendant dans l'environnement d'exécution et peut continuer de s'exécuter une fois l'appel de fonction entièrement traité. Étant donné que les extensions s'exécutent en tant que processus, vous pouvez les écrire dans un langage différent de celui de la fonction. Nous vous recommandons d'implémenter des extensions à l'aide d'un langage compilé. Dans ce cas, l'extension est un fichier binaire autonome compatible avec les environnements d'exécution pris en charge. Tous les Environnements d'exécution (runtimes) Lambda prennent en charge des extensions. Si vous utilisez un langage non compilé, assurez-vous d'inclure un environnement d'exécution compatible dans l'extension.

Lambda prend également en charge les extensions internes. Une extension interne s'exécute comme un thread séparé dans le processus d'exécution. L'exécution démarre et arrête l'extension interne. Un

autre mode d'intégration à l'environnement Lambda consiste à utiliser des variables d'environnement spécifiques au langage et des scripts encapsuleurs. Vous pouvez utiliser ces paramètres pour configurer l'environnement d'exécution et modifier le comportement de démarrage du processus d'exécution.

Vous pouvez ajouter des extensions à une fonction de deux manières. Pour une fonction déployée en tant qu'archive de fichier .zip, vous déployez votre extension en tant que couche. Pour une fonction définie comme une image de conteneur, vous ajoutez les extensions à cette dernière.

> **Note**
>
> Pour des exemples d'extensions et de scripts wrapper, voir AWS Lambda Extensions dans le GitHub référentiel AWS Samples.

Rubriques

- Cycle de vie d'un environnement d'exécution Lambda
- APIRéférence des extensions

## Cycle de vie d'un environnement d'exécution Lambda

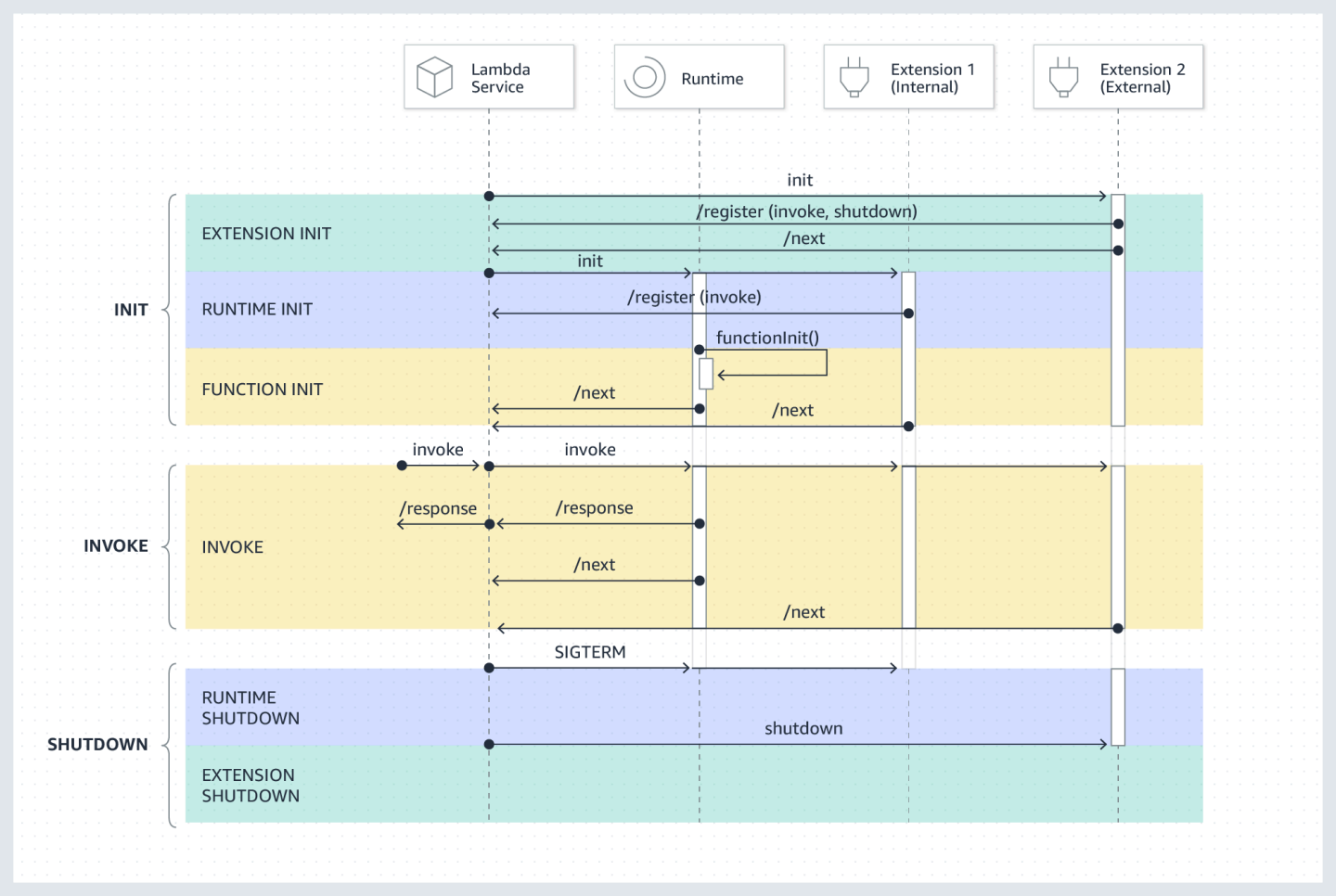
Le cycle de vie de l'environnement d'exécution comprend les phases suivantes :

- **`Init`** : au cours de cette phase, Lambda crée ou libère un environnement d'exécution avec les ressources configurées, télécharge le code pour la fonction et toutes les couches, initialise les extensions, initialise l'exécution et exécute le code d'initialisation de la fonction (code en dehors du gestionnaire principal). La phase `Init` se produit soit lors de la première invocation, soit avant invocations de fonctions si vous avez activé la simultanéité approvisionnée.

  La phase `Init` est fractionnée en trois sous-phases : `Extension init`, `Runtime init` et `Function init`. Ces sous-phases garantissent que toutes les extensions et l'exécution accomplissent leurs tâches de configuration avant l'exécution du code de la fonction.
- **`Invoke`** : au cours de cette phase, Lambda appelle le gestionnaire de la fonction. Une fois l'exécution de la fonction terminée, Lambda se prépare à gérer une autre invocation de fonction.
- **`Shutdown`**: cette phase se déclenche si la fonction Lambda ne reçoit aucune invocation pendant un certain temps. Au cours de la phase `Shutdown`, Lambda arrête l'exécution, alerte les extensions pour les laisser s'arrêter proprement, puis supprime l'environnement. Lambda envoie à

chaque extension un événement `Shutdown` indiquant que l'environnement est sur le point d'être arrêté.

Chaque phase commence par un événement que Lambda envoie au runtime et à toutes les extensions enregistrées. Le runtime et chaque extension signalent l'achèvement en envoyant une `Next` API demande. Lambda bloque l'environnement d'exécution lorsque l'exécution de chaque processus est terminée, et qu'il n'y a pas d'événement en attente.




Rubriques

- Phase d'initialisation
- Phase d'invocation
- Phase d'arrêt
- Autorisations et configuration
- Gestion des défaillances

- Dépannage des extensions

## Phase d'initialisation

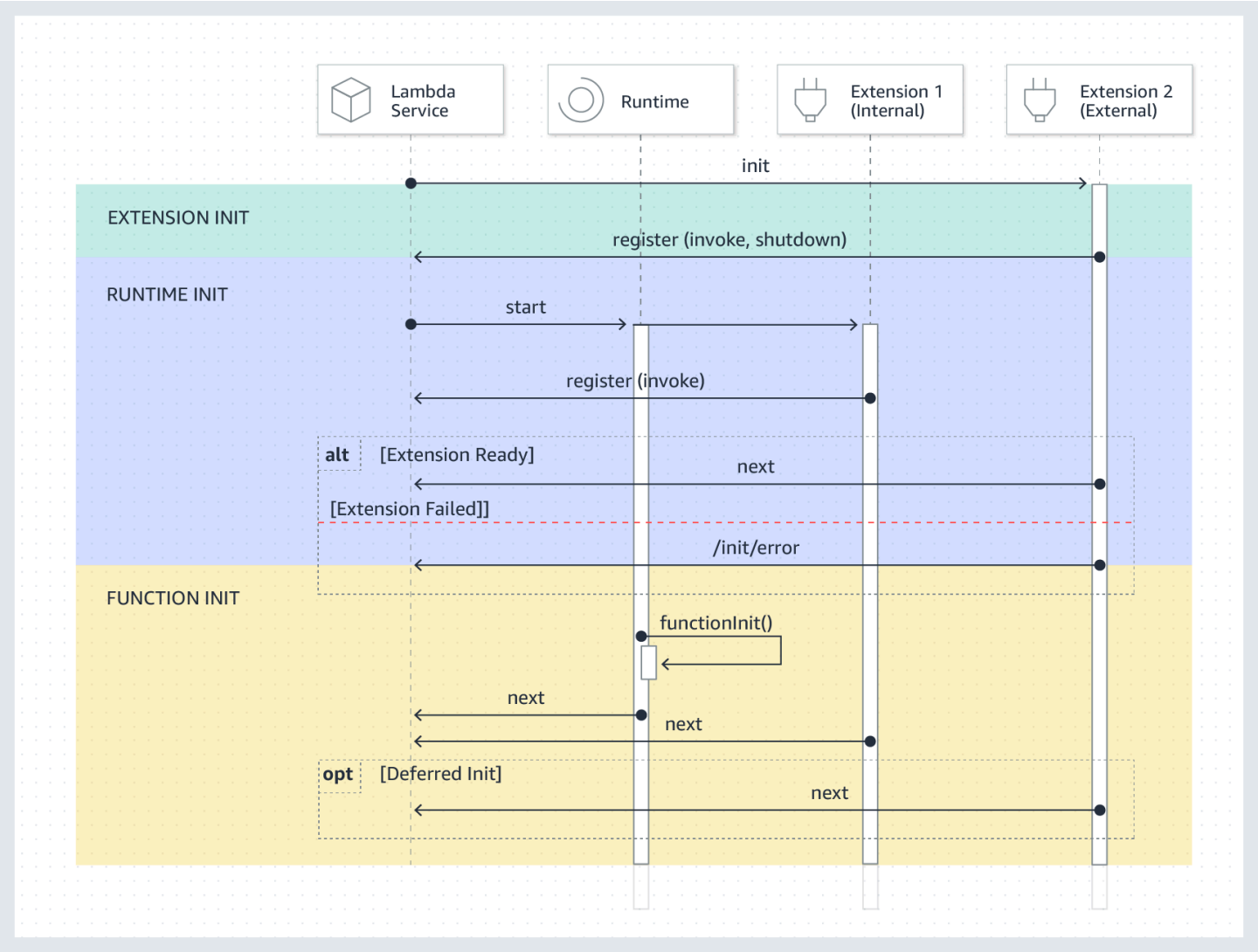
Au cours de la phase `Extension init`, chaque extension doit s'enregistrer auprès de Lambda pour recevoir des événements. Lambda utilise le nom de fichier complet de l'extension pour vérifier que la séquence d'amorçage de celle-ci est terminée. Par conséquent, chaque `Register` API appel doit inclure l'`Lambda-Extension-Name`en-tête avec le nom de fichier complet de l'extension.

Vous pouvez enregistrer jusqu'à 10 extensions pour une fonction. Cette limite est appliquée par le biais de l'`Register`APIappel.

Après l'enregistrement de chaque extension, Lambda démarre la phase `Runtime init`. Le processus d'exécution appelle `functionInit` pour démarrer la phase `Function init`.

La `Init` phase se termine après l'exécution et chaque extension enregistrée indique qu'elle est terminée en envoyant une `Next` API demande.

> **Note**
>
> Les extensions peuvent terminer leur initialisation à n'importe quel moment de la phase `Init`.



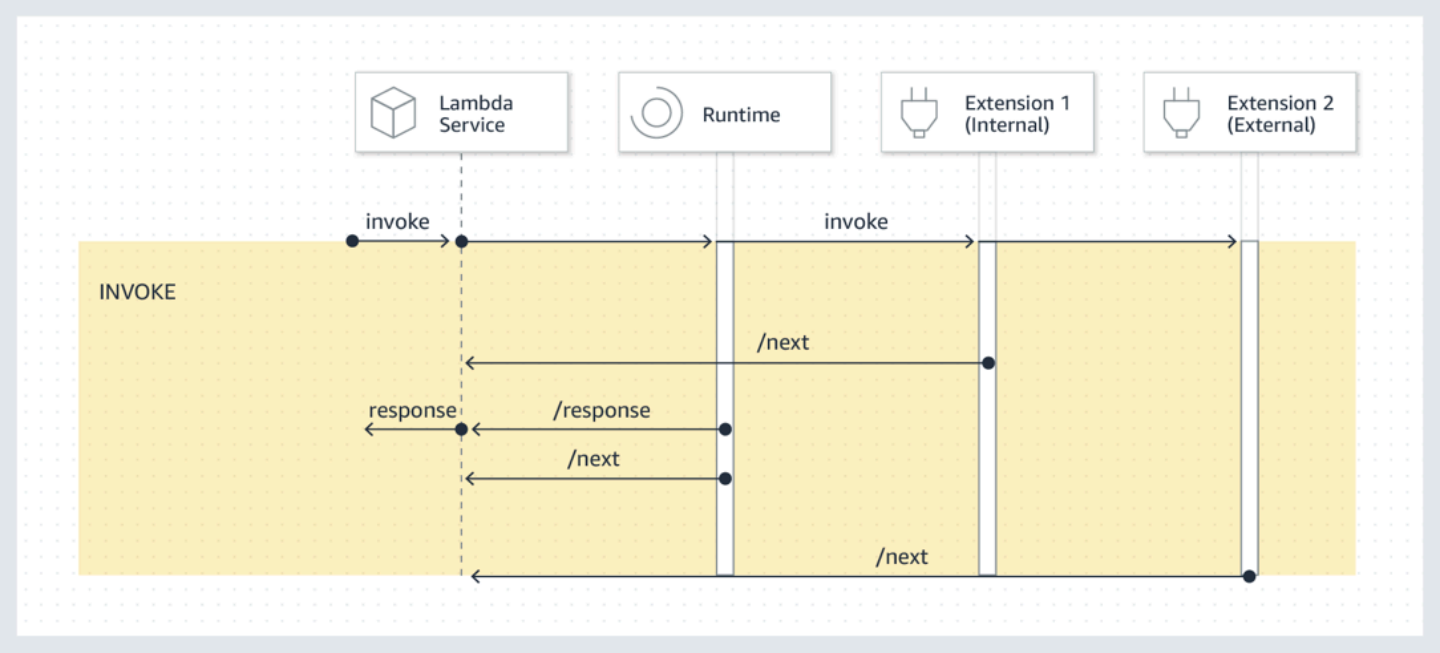

## Phase d’invocation

Lorsqu'une fonction Lambda est invoquée en réponse à une `Next` API demande, Lambda envoie un `Invoke` événement au moteur d'exécution et à chaque extension enregistrée pour l'événement. `Invoke`

Pendant l’appel, les extensions externes s’exécutent en parallèle avec la fonction. Elles continuent également à s’exécuter après la fin de la fonction. Cela vous permet de capturer des informations de diagnostic ou d’envoyer des journaux, des métriques et des suivis à l’emplacement de votre choix.

Après réception de la réponse de fonction de l’environnement d’exécution, Lambda renvoie celle-ci au client, même si les extensions sont toujours en cours d’exécution.

La `Invoke` phase se termine après l'exécution et toutes les extensions signalent qu'elles sont terminées en envoyant une `Next` API demande.




Charge utile de l'événement : l'événement envoyé à l'environnement d'exécution (et à la fonction Lambda) transporte la totalité de la requête, les en-têtes (tels que `RequestId`) et la charge utile. L'événement envoyé à chaque extension contient des métadonnées décrivant le contenu de l'événement. Cet événement du cycle de vie inclut le type d'événement, l'heure d'expiration de la fonction (`deadlineMs`)`requestId`, le Amazon Resource Name (ARN) de la fonction invoquée et les en-têtes de suivi.

Les extensions qui souhaitent accéder au corps de l'événement de la fonction peuvent utiliser un environnement d'exécution SDK qui communique avec l'extension. Les développeurs de fonctions utilisent la fonction en cours d'exécution SDK pour envoyer la charge utile à l'extension lorsque la fonction est invoquée.

Voici un exemple de charge utile :

```
{
    "eventType": "INVOKE",
    "deadlineMs": 676051,
    "requestId": "3da1f2dc-3222-475e-9205-e2e6c6318895",
    "invokedFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:ExtensionTest",
    "tracing": {
        "type": "X-Amzn-Trace-Id",
```

```
        "value":
"Root=1-5f35ae12-0c0fec141ab77a00bc047aa2;Parent=2be948a625588e32;Sampled=1"
    }
}
```

Limite de durée : le paramètre d'expiration de la fonction limite la durée de la phase `Invoke` entière. Par exemple, si vous définissez le délai d'expiration de la fonction sur 360 secondes, la fonction et toutes les extensions doivent être terminées dans un délai de 360 secondes. Notez qu'il n'y a pas de phase post-invocation indépendante. La durée est le temps total nécessaire à votre exécution et à toutes les invocations de vos extensions. Elle n'est calculée que lorsque la fonction et toutes les extensions sont terminées.

Impact sur les performances et surcharge d'extension : les extensions peuvent avoir un impact sur les performances des fonctions. En tant qu'auteur d'extension, vous contrôlez l'impact de votre extension sur les performances. Par exemple, si votre extension effectue des opérations de calcul intensives, la durée de la fonction augmente car l'extension et le code de fonction partagent les mêmes CPU ressources. En outre, si votre extension effectue des opérations importantes après la fin de l'appel de fonction, la durée de la fonction augmente, car la phase `Invoke` continue jusqu'à ce que toutes les extensions signalent qu'elles sont terminées.

> **Note**
>
> Lambda alloue la CPU puissance proportionnellement au réglage de la mémoire de la fonction. La durée d'exécution et d'initialisation peut augmenter lorsque les paramètres de mémoire sont faibles, car les processus de fonction et d'extension sont en concurrence pour les mêmes CPU ressources. Pour réduire la durée d'exécution et d'initialisation, essayez d'augmenter le paramètre de mémoire.

Pour vous aider à déterminer l'impact sur les performances des extensions dans la phase `Invoke`, Lambda génère la métrique `PostRuntimeExtensionsDuration`. Cette métrique mesure le temps cumulé passé entre la `Next` API demande d'exécution et la dernière `Next` API demande d'extension. La métrique `MaxMemoryUsed` permet de mesurer l'augmentation de la mémoire utilisée. Pour de plus amples informations sur les métriques de fonction, veuillez consulter Afficher les métriques des fonctions Lambda.

Les développeurs de fonctions peuvent exécuter différentes versions de leurs fonctions côte à côte pour comprendre l'impact d'une extension spécifique. Nous recommandons aux auteurs d'extension

de publier la consommation de ressources prévue de manière à aider les développeurs de fonction dans leur choix de l'extension appropriée.

## Phase d'arrêt

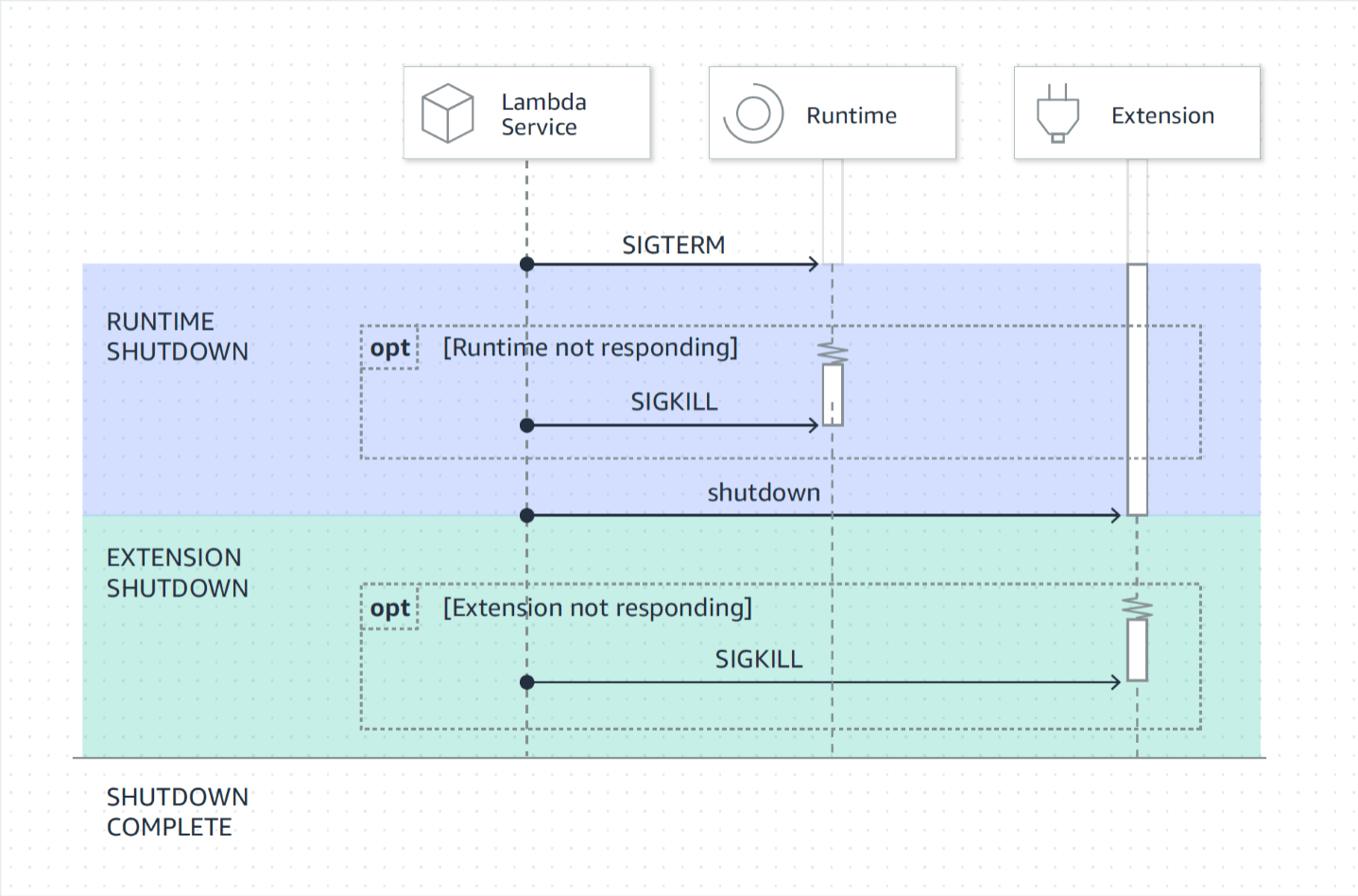
Quand Lambda est sur le point d'arrêter l'exécution, il envoie un `Shutdown` à chaque extension externe enregistrée. Les extensions peuvent utiliser ce temps pour les tâches de nettoyage final. L'`Shutdown`événement est envoyé en réponse à une `Next` API demande.

Limite de durée : la durée maximale de la phase `Shutdown` dépend de la configuration des extensions enregistrées :

- 0 ms : fonction sans extension enregistrée
- 500 ms : fonction avec une extension interne enregistrée.
- 2 000 ms : fonction avec une ou plusieurs extensions externes enregistrées.

Pour une fonction avec des extensions externes, Lambda réserve jusqu'à 300 ms (500 ms pour un runtime avec une extension interne) afin de permettre au processus de l'environnement d'exécution de s'arrêter proprement. Lambda alloue le reste de la limite de 2 000 ms à l'arrêt des extensions externes.

Si l'environnement d'exécution ou une extension ne répondent pas à l'événement `Shutdown` dans cette limite de temps, Lambda met fin au processus à l'aide d'un signal `SIGKILL`.




Charge utile d'événement : l'événement `Shutdown` contient la raison de l'arrêt et le temps restant en millisecondes.

`shutdownReason` contient les éléments suivants :

- SPINDOWN— Arrêt normal
- TIMEOUT— La limite de durée a expiré
- FAILURE— Condition d'erreur, telle qu'un `out-of-memory` événement

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time
 milliseconds"
}
```

## Autorisations et configuration

Les extensions s'exécutent dans le même environnement d'exécution que la fonction Lambda. Les extensions partagent également des ressources avec la fonctionCPU, telles que la mémoire et le stockage `/tmp` sur disque. De plus, les extensions utilisent le même rôle AWS Identity and Access Management (IAM) et le même contexte de sécurité que la fonction.

Autorisations d'accès au système de fichiers et au réseau : les extensions s'exécutent dans le même espace de noms de système de fichiers et de nom de réseau que l'environnement d'exécution de la fonction. Cela signifie que les extensions doivent être compatibles avec le système d'exploitation associé. Si une extension nécessite des règles supplémentaires de trafic réseau sortant, vous devez appliquer celles-ci à la configuration de la fonction.

> **Note**
>
> Étant donné que le répertoire du code de la fonction est en lecture seule, les extensions ne peuvent pas modifier le code de la fonction.

Variables d'environnement : les extensions peuvent accéder aux variables d'environnement de la fonction, à l'exception des variables suivantes spécifiques au processus d'environnement d'exécution :

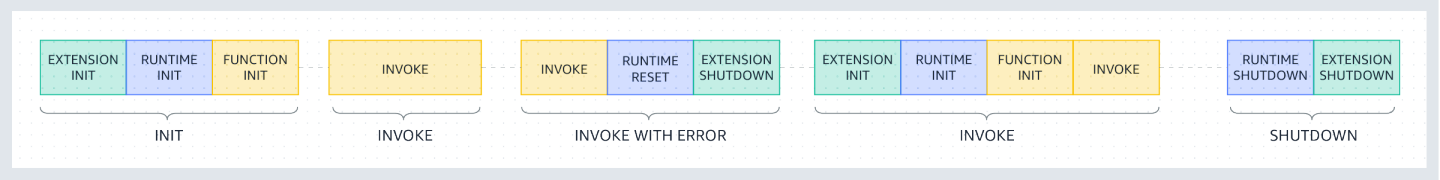
- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

## Gestion des défaillances

Échecs d'initialisation : si une extension échoue, Lambda redémarre l'environnement d'exécution pour assurer un comportement cohérent et encourager un échec rapide des extensions. En outre, pour certains clients, les extensions doivent répondre à des besoins stratégiques tels que la journalisation, la sécurité, la gouvernance et la collecte de télémétrie.

Échecs d'appel (par exemple, manque de mémoire, expiration de fonction) : les extensions partageant des ressources avec l'environnement d'exécution, elles sont affectées par l'épuisement de la mémoire. Lorsque l'environnement d'exécution échoue, toutes les extensions et l'environnement d'exécution lui-même participent à la phase `Shutdown`. En outre, l'environnement d'exécution est redémarré soit automatiquement dans le cadre de l'appel actuel, soit via un mécanisme de réinitialisation différée.

En cas d'échec (par exemple, dépassement de délai d'attente de fonction ou erreur d'exécution) pendant la phase `Invoke`, le service Lambda effectue une réinitialisation. La réinitialisation se comporte comme un événement `Shutdown`. Lambda commence par arrêter l'environnement d'exécution, puis envoie un événement `Shutdown` à chaque extension externe enregistrée. L'événement comprend le motif de l'arrêt. Si cet environnement est utilisé pour un nouvel appel, l'extension et l'environnement d'exécution sont réinitialisés dans le cadre de l'appel suivant.




Pour une explication plus détaillée du diagramme précédent, consultez Échecs pendant la phase d'invocation.

Journaux des extensions : Lambda envoie le résultat du journal des extensions à CloudWatch Logs. Lambda génère également un événement de journal supplémentaire pour chaque extension pendant la phase `Init`. L'événement de journal enregistre le nom et la préférence d'enregistrement (événement, configuration) en cas de succès, ou la raison de l'échec le cas échéant.

## Dépannage des extensions

- Si une `Register` demande échoue, assurez-vous que l'`Lambda-Extension-Name`en-tête de l'`Register`APIappel contient le nom de fichier complet de l'extension.
- Si la demande `Register` échoue pour une extension interne, assurez-vous que la demande n'est pas enregistrée pour l'événement `Shutdown`.

# APIRéférence des extensions

La API spécification Open pour la API version 2020-01-01 des extensions est disponible ici : extensions-api.zip

Vous pouvez récupérer la valeur du API point de terminaison à partir de la variable d'`AWS_LAMBDA_RUNTIME_API`environnement. Pour envoyer une `Register` demande, utilisez le préfixe `2020-01-01/` avant chaque API chemin. Par exemple :

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

APIméthodes

- Enregistrement
- Suivant
- Erreur d'initiation
- Erreur de sortie

## Enregistrement

Au cours de la phase `Extension init`, chaque extension doit s'enregistrer auprès de Lambda pour recevoir les événements. Lambda utilise le nom de fichier complet de l'extension pour vérifier que la séquence d'amorçage de celle-ci est terminée. Par conséquent, chaque `Register` API appel doit inclure l'`Lambda-Extension-Name`en-tête avec le nom de fichier complet de l'extension.

Les extensions internes sont lancées et arrêtées par le processus d'environnement d'exécution, de sorte qu'elles ne peuvent pas être enregistrées pour l'événement `Shutdown`.

Chemin – `/extension/register`

Méthode — POST

En-têtes de demandes

- `Lambda-Extension-Name` : nom de fichier complet de l'extension. Obligatoire : oui. Type : chaîne.
- `Lambda-Extension-Accept-Feature` – Utilisez ceci pour spécifier les fonctionnalités optionnelles d'Extensions pendant l'enregistrement. Requis : non. Type : chaîne séparée par des virgules. Fonctions disponibles pour spécifier en utilisant ce paramètre :

- `accountId` – Si elle est spécifiée, la réponse d'enregistrement de l'extension contiendra l'ID du compte associé à la fonction Lambda pour laquelle vous enregistrez l'extension.

Paramètres du corps de la demande

- `events` : tableau des événements pour lesquels s'enregistrer. Requis : non. Type : tableau de chaînes. Chaînes valides : `INVOKE`, `SHUTDOWN`.

En-têtes de réponse

- `Lambda-Extension-Identifier`— Identifiant d'agent unique (UUIDchaîne) généré qui est requis pour toutes les demandes suivantes.

Codes de réponse

- 200 – Corps de la réponse contenant le nom de la fonction, la version de la fonction et le nom du gestionnaire.
- 400 – Demande erronée.
- 403 – Interdit
- 500 – Erreur de conteneur. État non récupérable. L'extension doit se fermer rapidement.

Example Exemple de corps de la demande

```
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Example Exemple de corps de la réponse

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler"
}
```

Example Exemple de corps de réponse avec accountId fonctionnalité optionnelle

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler",
    "accountId": "123456789012"
}
```

## Suivant

Les extensions envoient une `Next` API demande pour recevoir le prochain événement, qui peut être un `Invoke` événement ou un `Shutdown` événement. Le corps de la réponse contient la charge utile, qui est un JSON document contenant des données d'événements.

L'extension envoie une `Next` API demande pour signaler qu'elle est prête à recevoir de nouveaux événements. Il s'agit d'un appel bloquant.

Ne fixez pas de délai d'attente pour l'GETappel, car la prolongation peut être suspendue pendant un certain temps jusqu'à ce qu'un événement revienne.

Chemin – `/extension/event/next`

Méthode — GET

En-têtes de demandes

- `Lambda-Extension-Identifier`— Identifiant unique pour l'extension (UUIDchaîne). Obligatoire : oui. Type : UUID chaîne

En-têtes de réponse

- `Lambda-Extension-Event-Identifier`— Identifiant unique de l'événement (UUIDchaîne).

Codes de réponse

- 200 : réponse contenant des informations sur l'événement suivant (`EventInvoke` ou `EventShutdown`).
- 403 – Interdit.

- 500 – Erreur de conteneur. État non récupérable. L'extension doit se fermer rapidement.

## Erreur d'initiation

L'extension utilise cette méthode pour signaler une erreur d'initialisation à Lambda. Appelez-la lorsque l'extension ne parvient pas à s'initialiser après son enregistrement. Une fois que Lambda a reçu l'erreur, aucun autre API appel ne réussit. L'extension doit quitter après avoir reçu la réponse de Lambda.

Chemin – `/extension/init/error`

Méthode — POST

En-têtes de demandes

- `Lambda-Extension-Identifier` : identifiant unique pour l'extension. Obligatoire : oui. Type : UUID chaîne
- `Lambda-Extension-Function-Error-Type` – Type d'erreur rencontré par l'extension. Obligatoire : oui. Cet en-tête se compose d'une valeur de chaîne. Lambda accepte n'importe quelle chaîne, mais nous recommandons le format <category.reason>. Par exemple :
  - Prolongation. NoSuchHandler
  - Prolongation. APIKeyNotFound
  - Prolongation. ConfigInvalid
  - Prolongation. UnknownReason

Paramètres du corps de la demande

- `ErrorRequest` : informations sur l'erreur. Requis : non.

Ce champ est un JSON objet dont la structure est la suivante :

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Notez que Lambda accepte n'importe quelle valeur pour `errorType`.

L'exemple suivant montre un message d'erreur de fonction Lambda indiquant que la fonction n'a pas pu analyser les données d'événement fournies dans l'invocation.

Example Erreur de fonction

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Codes de réponse

- 202 – Accepté
- 400 – Demande erronée.
- 403 – Interdit
- 500 – Erreur de conteneur. État non récupérable. L'extension doit se fermer rapidement.

## Erreur de sortie

L'extension utilise cette méthode pour signaler une erreur à Lambda avant de quitter. Appelez-la lorsque vous rencontrez une défaillance inattendue. Une fois que Lambda a reçu l'erreur, aucun autre API appel ne réussit. L'extension doit quitter après avoir reçu la réponse de Lambda.

Chemin – `/extension/exit/error`

Méthode — POST

En-têtes de demandes

- `Lambda-Extension-Identifier` : identifiant unique pour l'extension. Obligatoire : oui. Type : UUID chaîne
- `Lambda-Extension-Function-Error-Type` – Type d'erreur rencontré par l'extension. Obligatoire : oui. Cet en-tête se compose d'une valeur de chaîne. Lambda accepte n'importe quelle chaîne, mais nous recommandons le format <category.reason>. Par exemple :
  - Prolongation. NoSuchHandler
  - Prolongation. APIKeyNotFound
  - Prolongation. ConfigInvalid

- Prolongation. UnknownReason

Paramètres du corps de la demande

- `ErrorRequest` : informations sur l'erreur. Requis : non.

Ce champ est un JSON objet dont la structure est la suivante :

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Notez que Lambda accepte n'importe quelle valeur pour `errorType`.

L'exemple suivant montre un message d'erreur de fonction Lambda indiquant que la fonction n'a pas pu analyser les données d'événement fournies dans l'invocation.

Example Erreur de fonction

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Codes de réponse

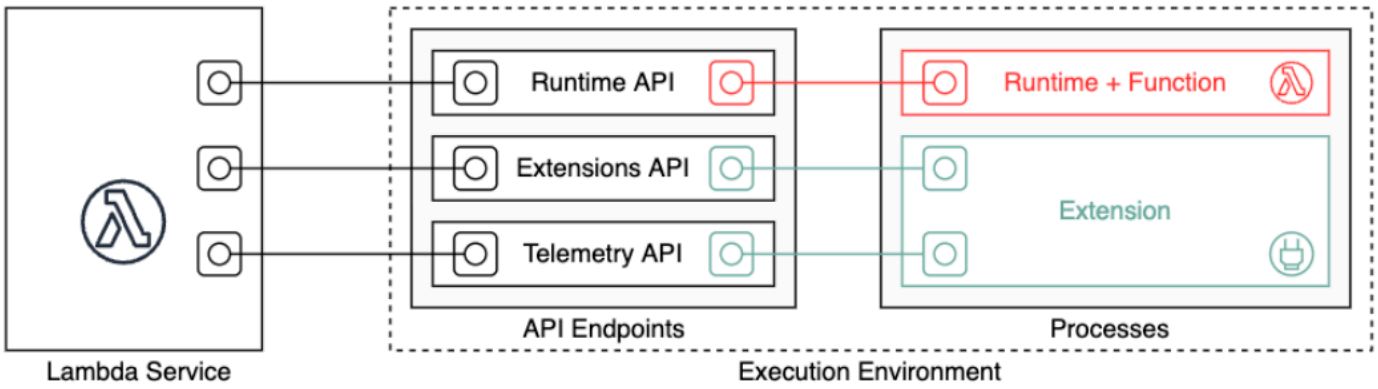
- 202 – Accepté
- 400 – Demande erronée.
- 403 – Interdit
- 500 – Erreur de conteneur. État non récupérable. L'extension doit se fermer rapidement.

# Accès aux données de télémétrie en temps réel pour les extensions à l'aide de la télémétrie API

La télémétrie API permet à vos extensions de recevoir des données de télémétrie directement depuis Lambda. Pendant l'initialisation et l'invocation des fonctions, Lambda capture automatiquement la télémétrie, notamment les journaux, les métriques de la plateforme et les traces de la plateforme. La télémétrie API permet aux extensions d'accéder à ces données de télémétrie directement depuis Lambda en temps quasi réel.

Dans l'environnement d'exécution Lambda, vous pouvez abonner vos extensions Lambda à des flux de télémétrie. Après la souscription, Lambda envoie automatiquement toutes les données de télémétrie à vos extensions. Vous pouvez ensuite traiter, filtrer et distribuer les données à votre destination préférée, telle qu'un compartiment Amazon Simple Storage Service (Amazon S3) ou un fournisseur d'outils d'observabilité tiers.

Le schéma suivant montre comment les extensions API et la télémétrie API relient les extensions à Lambda depuis l'environnement d'exécution. De plus, le Runtime API connecte votre environnement d'exécution et votre fonction à Lambda.




> **Important**
>
> La télémétrie Lambda remplace API les journaux Lambda. API Bien que les journaux API restent entièrement fonctionnels, nous vous recommandons de n'utiliser que la télémétrie à l'APIavenir. Vous pouvez abonner votre extension à un flux de télémétrie à l'aide de la télémétrie API ou des journaux. API Après vous être abonné à l'aide de l'un d'entre euxAPIs, toute tentative de souscription à l'aide de l'autre API renvoie une erreur.

Les extensions peuvent utiliser la télémétrie API pour s'abonner à trois flux de télémétrie différents :

- Télémétrie de plateforme – Journaux, métriques et traces, qui décrivent les événements et les erreurs liés au cycle de vie de l'environnement d'exécution, au cycle de vie des extensions et aux appels de fonctions
- Journaux de fonction – Journaux personnalisés que le code de la fonction Lambda génère.
- Journaux d'extension – Journaux personnalisés générés par le code d'extension Lambda.

> **Note**
>
> Lambda envoie des journaux CloudWatch, des métriques et des traces à X-Ray (si vous avez activé le suivi), même si une extension s'abonne à des flux de télémétrie.

Sections

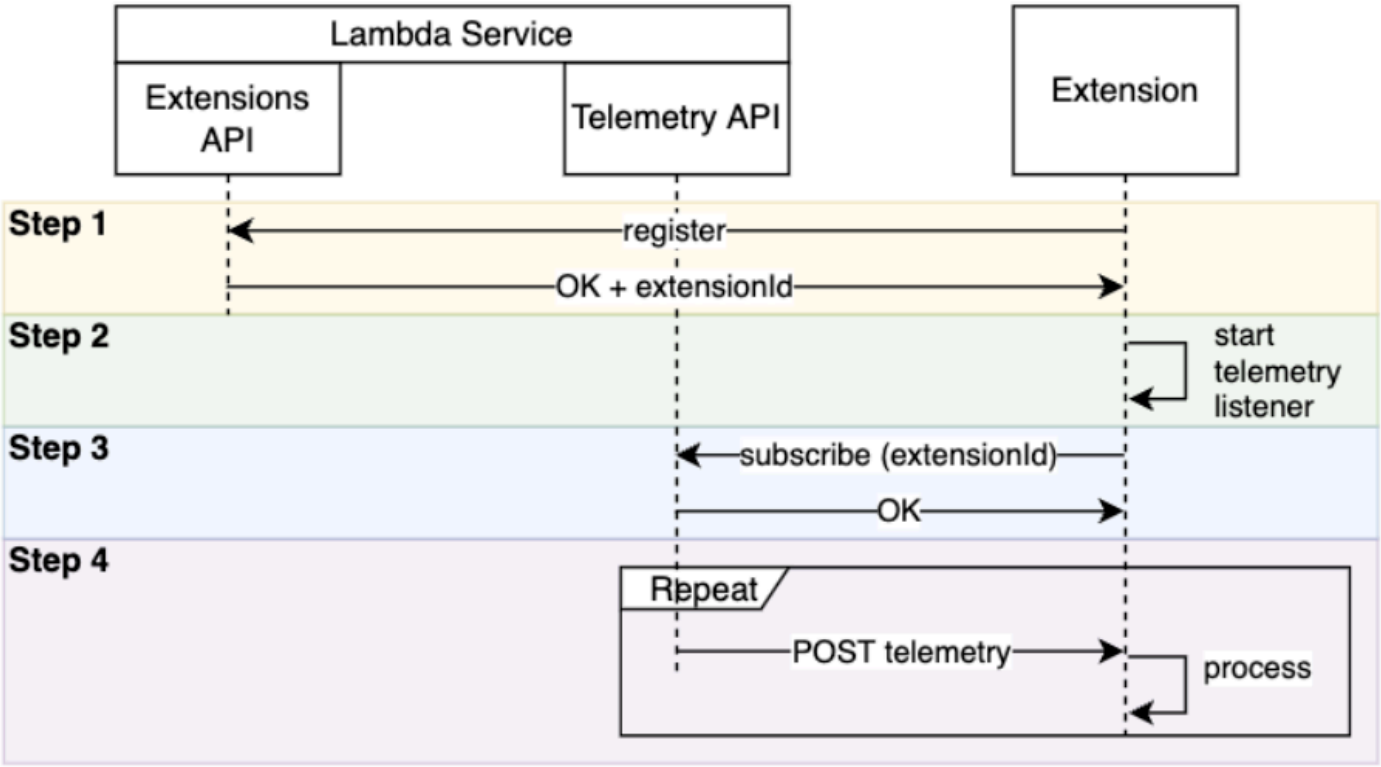

- Création d'extensions à l'aide de la télémétrie API
- Enregistrement de votre extension
- Création d'un écouteur de télémétrie
- Spécification d'un protocole de destination
- Configuration de l'utilisation de la mémoire et de la mise en mémoire tampon
- Envoi d'une demande d'abonnement à la télémétrie API
- Messages de télémétrie API entrants
- Référence de l'API de télémétrie Lambda
- Référence du schéma de API Event télémétrie Lambda
- Conversion d'objets de télémétrie API Event Lambda en Spans OpenTelemetry
- Utilisation des journaux Lambda API


## Création d'extensions à l'aide de la télémétrie API

Les extensions Lambda s'exécutent comme des processus indépendants dans l'environnement d'exécution. Les extensions peuvent continuer à s'exécuter une fois l'invocation des fonctions terminée. Comme les extensions sont des processus séparés, vous pouvez les écrire dans un langage différent du code de la fonction. Nous recommandons d'écrire les extensions en utilisant un

langage compilé tel que Golang ou Rust. De cette façon, l'extension est un binaire autonome qui peut être compatible avec tout environnement d'exécution pris en charge.

Le schéma suivant illustre un processus en quatre étapes pour créer une extension qui reçoit et traite les données de télémétrie à l'aide de la télémétrie. API




Voici chaque étape plus en détail :

1. Enregistrez votre extension à l'aide du the section called "Prolongations API". Cela vous fournit un `Lambda-Extension-Identifier`, dont vous aurez besoin dans les étapes suivantes. Pour plus d'informations sur la façon d'enregistrer votre extension, consultez the section called "Enregistrement de votre extension".
2. Créez un écouteur de télémétrie. Il peut s'agir d'un serveur de base HTTP ou d'un TCP serveur. Lambda utilise le récepteur URI de télémétrie pour envoyer des données de télémétrie à votre extension. Pour de plus amples informations, veuillez consulter the section called "Création d'un écouteur de télémétrie".
3. À l'aide du bouton S'abonner API à la télémétrieAPI, abonnez votre extension aux flux de télémétrie souhaités. Vous aurez besoin URI de votre écouteur de télémétrie pour cette étape. Pour de plus amples informations, veuillez consulter the section called "Envoi d'une demande d'abonnement à la télémétrie API".

4. Obtenez les données de télémétrie de Lambda via l'écouteur de télémétrie. Vous pouvez effectuer tout traitement personnalisé de ces données, par exemple en les envoyant vers Amazon S3 ou vers un service d'observabilité externe.

> **Note**
>
> L'environnement d'exécution d'une fonction Lambda peut démarrer et s'arrêter plusieurs fois dans le cadre de son cycle de vie. En général, votre code d'extension s'exécute pendant les appels de la fonction, et aussi jusqu'à 2 secondes pendant la phase d'arrêt. Nous vous recommandons de regrouper la télémétrie au fur et à mesure qu'elle parvient à votre écouteur. Utilisez ensuite les événements du cycle de vie `Invoke` et `Shutdown` pour envoyer chaque lot vers les destinations souhaitées.

## Enregistrement de votre extension

Avant de pouvoir vous abonner aux données de télémétrie, vous devez enregistrer votre extension Lambda. L'enregistrement a lieu pendant la phase d'initialisation de l'extension. L'exemple suivant montre une HTTP demande d'enregistrement d'une extension.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
 Lambda-Extension-Name: lambda_extension_name
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Si la demande aboutit, l'abonné reçoit une réponse positive de HTTP 200. L'en-tête de réponse contient le `Lambda-Extension-Identifier`. Le corps de la réponse contient d'autres propriétés de la fonction.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
{
    "functionName": "lambda_function",
    "functionVersion": "$LATEST",
    "handler": "lambda_handler",
    "accountId": "123456789012"
}
```

Pour plus d'informations, consultez le the section called "APIRéférence des extensions".

## Création d'un écouteur de télémétrie

Votre extension Lambda doit disposer d'un écouteur qui gère les demandes entrantes provenant de la télémétrie. API Le code suivant présente un exemple de mise en œuvre d'un écouteur de télémétrie en Golang :

```
// Starts the server in a goroutine where the log events will be sent
func (s *TelemetryApiListener) Start() (string, error) {
 address := listenOnAddress()
 l.Info("[listener:Start] Starting on address", address)
 s.httpServer = &http.Server{Addr: address}
 http.HandleFunc("/", s.http_handler)
 go func() {
  err := s.httpServer.ListenAndServe()
  if err != http.ErrServerClosed {
   l.Error("[listener:goroutine] Unexpected stop on Http Server:", err)
   s.Shutdown()
  } else {
   l.Info("[listener:goroutine] Http Server closed:", err)
  }
 }()
 return fmt.Sprintf("http://%s/", address), nil
}

// http_handler handles the requests coming from the Telemetry API.
// Everytime Telemetry API sends log events, this function will read them from the
 response body
// and put into a synchronous queue to be dispatched later.
// Logging or printing besides the error cases below is not recommended if you have
 subscribed to
// receive extension logs. Otherwise, logging here will cause Telemetry API to send new
 logs for
// the printed lines which may create an infinite loop.
func (s *TelemetryApiListener) http_handler(w http.ResponseWriter, r *http.Request) {
 body, err := ioutil.ReadAll(r.Body)
 if err != nil {
  l.Error("[listener:http_handler] Error reading body:", err)
  return
 }

 // Parse and put the log messages into the queue
```

```
 var slice []interface{}
 _ = json.Unmarshal(body, &slice)

 for _, el := range slice {
  s.LogEventsQueue.Put(el)
 }

 l.Info("[listener:http_handler] logEvents received:", len(slice), " LogEventsQueue
 length:", s.LogEventsQueue.Len())
 slice = nil
}
```

## Spécification d’un protocole de destination

Lorsque vous vous abonnez pour recevoir de la télémétrie à l'aide de la télémétrieAPI, vous pouvez spécifier un protocole de destination en plus de la destination : URI

```
{
    "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
    }
}
```

Lambda accepte deux protocoles pour recevoir des télémétries :

- HTTP(recommandé) — Lambda fournit la télémétrie à un point de HTTP terminaison local (`http://sandbox.localdomain:${PORT}/${PATH}`) sous forme de tableau d'enregistrements au format. JSON Le paramètre `$PATH` est facultatif. Lambda prend uniquement en chargeHTTP, pas. HTTPS Lambda fournit des services de télémétrie par le biais de demandes. POST
- TCP— Lambda fournit la télémétrie à un TCP port au format Newline Delimited (). JSON NDJSON

> **Note**
>
> Nous vous recommandons vivement d'utiliser HTTP plutôt queTCP. AinsiTCP, la plateforme Lambda ne peut pas accuser réception lorsqu'elle fournit de la télémétrie à la couche application. Par conséquent, si votre extension se bloque, vous risquez de perdre la télémétrie. HTTPn'a pas cette limitation.

Avant de vous abonner pour recevoir des données de télémétrie, définissez le port ou l'HTTPécouteur local. TCP Au cours de l’installation, notez ce qui suit :

- Lambda n’envoie la télémétrie qu’à des destinations au sein de l’environnement d’exécution.
- Lambda essaie à nouveau d'envoyer des données télémétriques (avec interruption) en l'absence d'écouteur ou si la demande rencontre une erreur. POST Si l'écouteur de télémétrie se bloque, il reprend la réception de la télémétrie après que Lambda a redémarré l'environnement d'exécution.
- Lambda réserve le port 9001. Il n’y a pas d’autres restrictions ou recommandations relatives au numéro de port.

## Configuration de l’utilisation de la mémoire et de la mise en mémoire tampon

L'utilisation de la mémoire dans un environnement d'exécution augmente de façon linéaire avec le nombre d'abonnés. Les abonnements consomment des ressources de mémoire, car chacun d’eux ouvre un nouveau tampon mémoire pour stocker les données de télémétrie. L'utilisation de la mémoire tampon contribue à la consommation globale de mémoire dans l'environnement d'exécution.

Lorsque vous vous abonnez pour recevoir de la télémétrie par le biais de la télémétrieAPI, vous avez la possibilité de mettre en mémoire tampon les données de télémétrie et de les transmettre aux abonnés par lots. Pour optimiser l'utilisation de la mémoire, vous pouvez spécifier une configuration de mise en mémoire tampon :

```
{
    "buffering": {
        "maxBytes": 256*1024,
        "maxItems": 1000,
        "timeoutMs": 100
    }
}
```

| Paramètre | Description | Valeurs par défaut et limites |
| --- | --- | --- |
| `maxBytes` | Le volume maximal de télémétrie (en octets) à mettre en mémoire tampon. | Par défaut : 262 144<br>Minimum : 262 144<br>Maximum : 1 048 576 |

| Paramètre | Description | Valeurs par défaut et limites |
| --- | --- | --- |
| `maxItems` | Le nombre maximal d'événements à mettre en mémoire tampon. | Par défaut : 10 000<br>Minimum : 1 000<br>Maximum : 10 000. |
| `timeoutMs` | La durée maximale (en millisecondes) pour la mise en mémoire tampon d'un lot. | Par défaut : 1 000<br>Minimum : 25<br>Maximum : 30 000 |

Lorsque vous configurez la mise en mémoire tampon, tenez compte des points suivants :

- Si l'un des flux d'entrée est fermé, Lambda vide les journaux. Cela peut se produire si, par exemple, l'environnement d'exécution se bloque.
- Chaque abonné peut spécifier une configuration de mise en mémoire tampon différente dans sa demande d'abonnement.
- Lorsque vous déterminez la taille de la mémoire tampon pour la lecture des données, attendez-vous à recevoir des charges utiles aussi importantes que `2 * maxBytes + metadataBytes`, `maxBytes` étant un composant de votre configuration de mise en mémoire tampon. Pour évaluer la quantité de `metadataBytes` à prendre en compte, consultez les métadonnées suivantes. Lambda adjoint des métadonnées similaires à celles-ci à chaque enregistrement :

  ```
  {
     "time": "2022-08-20T12:31:32.123Z",
     "type": "function",
     "record": "Hello World"
  }
  ```

- Si l'abonné ne peut pas traiter la télémétrie entrante assez rapidement, ou si votre code de fonction génère un volume de journal très élevé, Lambda peut abandonner des enregistrements pour limiter l'utilisation de la mémoire. Lorsque cela se produit, Lambda envoie un événement `platform.logsDropped`.

# Envoi d'une demande d'abonnement à la télémétrie API

Les extensions Lambda peuvent s'abonner pour recevoir des données de télémétrie en envoyant une demande d'abonnement à la télémétrie. API La demande d'abonnement doit contenir des informations sur les types d'événements auxquels vous voulez que l'extension s'abonne. En outre, la demande peut contenir des informations sur la destination de la livraison et une configuration de mise en mémoire tampon.

Avant d'envoyer une demande d'abonnement, vous devez disposer d'un ID d'extension (`Lambda-Extension-Identifier`). Lorsque vous enregistrez votre extension auprès des extensions API, vous obtenez un identifiant d'extension à partir de la API réponse.

L'abonnement a lieu pendant la phase d'initialisation de l'extension. L'exemple suivant montre une HTTP demande d'abonnement aux trois flux de télémétrie : télémétrie de plate-forme, journaux de fonctions et journaux d'extension.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
   "schemaVersion": "2022-12-13",
   "types": [
        "platform",
        "function",
        "extension"
   ],
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   },
   "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
   }
}
```

Si la demande aboutit, l'abonné reçoit une réponse positive de HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

# Messages de télémétrie API entrants

Après s'être abonnée à l'aide de la télémétrieAPI, une extension commence automatiquement à recevoir la télémétrie de Lambda via des requêtes. POST Chaque corps de POST requête contient un tableau d'`Event`objets. Chaque `Event` est structuré selon le schéma suivant :

```
{
   time: String,
   type: String,
   record: Object
}
```

- La propriété `time` définit le moment où la plateforme Lambda a généré l'événement. Cela est différent du moment où l'événement s'est réellement produit. La valeur de chaîne de `time` est un horodatage au format ISO 8601.
- La propriété `type` définit le type d'événement. Le tableau suivant décrit toutes les valeurs possibles.
- La `record` propriété définit un JSON objet qui contient les données de télémétrie. Le schéma de cet JSON objet dépend du`type`.

Le tableau suivant récapitule tous les types d'`Event`objets et contient des liens vers la référence du API `Event` schéma de télémétrie pour chaque type d'événement.

| Catégorie | Type d'événement | Description | Schéma d'enregistrement des événements |
|---|---|---|---|
| Événement de plateforme | `platform.initStart` | L'initialisation de la fonction a commencé. | Schéma the section called "`platform.initStart`" |
| Événement de plateforme | `platform.initRuntimeDone` | L'initialisation de la fonction est terminée. | Schéma the section called "`platform.initRuntimeDone`" |

| Catégorie | Type d'événement | Description | Schéma d'enregistrement des événements |
|---|---|---|---|
| Événement de plateforme | `platform.initReport` | Un rapport d'initialisation de la fonction. | Schéma the section called "`platform.initReport`" |
| Événement de plateforme | `platform.start` | L'invocation de la fonction a commencé. | Schéma the section called "`platform.start`" |
| Événement de plateforme | `platform.runtimeDone` | L'environnement d'exécution a fini de traiter un événement avec succès ou échec. | Schéma the section called "`platform.runtimeDone`" |
| Événement de plateforme | `platform.report` | Un rapport sur l'invocation de la fonction. | Schéma the section called "`platform.report`" |
| Événement de plateforme | `platform.restoreStart` | La restauration de l'exécution a commencé. | Schéma the section called "`platform.restoreStart`" |
| Événement de plateforme | `platform.restoreRuntimeDone` | La restauration de l'exécution est terminée. | Schéma the section called "`platform.restoreRuntimeDone`" |
| Événement de plateforme | `platform.restoreReport` | Le rapport de restauration de d'exécution. | Schéma the section called "`platform.restoreReport`" |

| Catégorie | Type d'événement | Description | Schéma d'enregistrement des événements |
|---|---|---|---|
| Événement de plateforme | `platform.telemetrySubscription` | L'extension s'est abonnée à la télémétrieAPI. | Schéma the section called "`platform.telemetrySubscription`" |
| Événement de plateforme | `platform.logsDropped` | Les entrées de journal abandonnées par Lambda. | Schéma the section called "`platform.logsDropped`" |
| Journaux de fonctions | `function` | Une ligne de journal du code de la fonction. | Schéma the section called "`function`" |
| Journaux d'extension | `extension` | Une ligne de journal du code de l'extension. | Schéma the section called "`extension`" |

# Référence de l'API de télémétrie Lambda

Utilisez le point de terminaison de l'API de télémétrie Lambda pour abonner des extensions aux flux de télémétrie. Vous pouvez extraire le point de terminaison d'API de télémétrie à partir de la variable d'environnement `AWS_LAMBDA_RUNTIME_API`. Pour envoyer une demande d'API, ajoutez la version de l'API (`2022-07-01/`) et `telemetry/`. Par exemple :

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Pour la définition de la spécification OpenAPI (OAS) de la version des réponses d'abonnement `2022-12-13`, voir ce qui suit :

- HTTP — telemetry-api-http-schema.zip
- TCP — .zip telemetry-api-tcp-schema

Opérations d'API

- S'abonner

## S'abonner

Pour s'abonner à un flux de télémétrie, une extension Lambda peut envoyer une demande d'API d'abonnement.

- Chemin – `/telemetry`
- Méthode – PUT
- En-têtes
  - `Content-Type`: `application/json`
- Paramètres du corps de la demande
  - schemaVersion
    - Obligatoire : oui
    - Type : String
    - Valeurs valides : `"2022-12-13"` ou `"2022-07-01"`
  - destination – Les paramètres de configuration qui définissent la destination de l'événement de télémétrie et le protocole de livraison de l'événement.
    - Obligatoire : oui

- Type : objet

```
{
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
}
```

  - protocol – Le protocole que Lambda utilise pour envoyer les données de télémétrie.
    - Obligatoire : oui
    - Type : String
    - Valeurs valides : `"HTTP"|"TCP"`
  - URI – L'URI auquel envoyer les données de télémétrie.
    - Obligatoire : oui
    - Type : String
  - Pour plus d'informations, consultez the section called "Spécification d'un protocole de destination".
- types – Les types de télémétrie auxquels vous voulez que l'extension s'abonne.
  - Obligatoire : oui
  - Type : tableau de chaînes
  - Valeurs valides : `"platform"|"function"|"extension"`
- buffering – Les paramètres de configuration de la mise en mémoire tampon des événements.
  - Obligatoire : non
  - Type : objet

```
{
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   }
}
```

  - maxItems – Nombre maximum d'événements à mettre en mémoire tampon.
    - Obligatoire : non
    - Type : entier

    - Par défaut : 1 000
    - Minimum : 1 000
    - Maximum : 10 000.
- maxBytes – Le volume maximal de télémétrie (en octets) à mettre en mémoire tampon.
    - Obligatoire : non
    - Type : entier
    - Par défaut : 262 144
    - Minimum : 262 144
    - Maximum : 1 048 576
- timeoutMs – Durée maximum (en millisecondes) de mise en mémoire tampon d'un lot.
    - Obligatoire : non
    - Type : entier
    - Par défaut : 1 000
    - Minimum : 25
    - Maximum : 30 000
- Pour plus d'informations, consultez the section called "Configuration de l'utilisation de la mémoire et de la mise en mémoire tampon".

Exemple de demande d'API d'abonnement

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
   "schemaVersion": "2022-12-13",
   "types": [
        "platform",
        "function",
        "extension"
   ],
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   },
   "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
```

```
    }
}
```

Si la demande d'abonnement aboutit, l'extension reçoit une réponse HTTP 200 :

```
HTTP/1.1 200 OK
"OK"
```

Si la demande d'abonnement échoue, l'extension reçoit une réponse d'erreur. Par exemple :

```
HTTP/1.1 400 OK
{
    "errorType": "ValidationError",
    "errorMessage": "URI port is not provided; types should not be empty"
}
```

Voici quelques codes de réponse supplémentaires que l'extension peut recevoir :

- 200 – Demande effectuée avec succès.
- 202 – Demande acceptée. Réponse à une demande d'abonnement dans un environnement de test local
- 400 – Demande erronée
- 500 – Erreur de service.

# Référence du schéma de API **`Event`** télémétrie Lambda

Utilisez le point de API terminaison de télémétrie Lambda pour abonner des extensions aux flux de télémétrie. Vous pouvez récupérer le point de API terminaison de télémétrie à partir de la variable d'`AWS_LAMBDA_RUNTIME_API`environnement. Pour envoyer une API demande, ajoutez les API versions (`2022-07-01/`) et`telemetry/`. Par exemple :

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Pour la définition Open API Specification (OAS) de la version des réponses aux abonnements`2022-12-13`, consultez ce qui suit :

- HTTP— telemetry-api-http-schema.zip
- TCP— telemetry-api-tcp-schema.zip

Le tableau suivant est un résumé de tous les types d'`Event`objets pris en charge par la télémétrieAPI.

| Catégorie | Type d'événement | Description | Schéma d'enregis trement des événements |
|---|---|---|---|
| Événement de plateforme | `platform. initStart` | L'initialisation de la fonction a commencé. | Schéma the section called "`platform. initStart`" |
| Événement de plateforme | `platform. initRunti meDone` | L'initialisation de la fonction est terminée. | Schéma the section called "`platform. initRunti meDone`" |
| Événement de plateforme | `platform. initReport` | Un rapport d'initial isation de la fonction. | Schéma the section called "`platform. initReport`" |

| Catégorie | Type d'événement | Description | Schéma d'enregistrement des événements |
|---|---|---|---|
| Événement de plateforme | `platform.start` | L'invocation de la fonction a commencé. | Schéma the section called "`platform.start`" |
| Événement de plateforme | `platform.runtimeDone` | L'environnement d'exécution a fini de traiter un événement avec succès ou échec. | Schéma the section called "`platform.runtimeDone`" |
| Événement de plateforme | `platform.report` | Un rapport sur l'invocation de la fonction. | Schéma the section called "`platform.report`" |
| Événement de plateforme | `platform.restoreStart` | La restauration de l'exécution a commencé. | Schéma the section called "`platform.restoreStart`" |
| Événement de plateforme | `platform.restoreRuntimeDone` | La restauration de l'exécution est terminée. | Schéma the section called "`platform.restoreRuntimeDone`" |
| Événement de plateforme | `platform.restoreReport` | Le rapport de restauration de d'exécution. | Schéma the section called "`platform.restoreReport`" |
| Événement de plateforme | `platform.telemetrySubscription` | L'extension s'est abonnée à la télémétrieAPI. | Schéma the section called "`platform.telemetrySubscription`" |

| Catégorie | Type d'événement | Description | Schéma d'enregistrement des événements |
|---|---|---|---|
| Événement de plateforme | `platform.logsDropped` | Les entrées de journal abandonnées par Lambda. | Schéma the section called "`platform.logsDropped`" |
| Journaux de fonctions | `function` | Une ligne de journal du code de la fonction. | Schéma the section called "`function`" |
| Journaux d'extension | `extension` | Une ligne de journal du code de l'extension. | Schéma the section called "`extension`" |

Table des matières


- Types d'objets de télémétrie API Event
  - platform.initStart
  - platform.initRuntimeDone
  - platform.initReport
  - platform.start
  - platform.runtimeDone
  - platform.report
  - platform.restoreStart
  - platform.restoreRuntimeDone
  - platform.restoreReport
  - platform.extension
  - platform.telemetrySubscription
  - platform.logsDropped
  - function
  - extension
- Types d'objets partagés

- InitPhase
- InitReportMetrics
- InitType
- ReportMetrics
- RestoreReportMetrics
- RuntimeDoneMetrics
- Span
- Status
- TraceContext
- TracingType

## Types d'objets de télémétrie API **`Event`**

Cette section détaille les types d'`Event`objets pris en charge par la API télémétrie Lambda. Dans les descriptions d'événements, un point d'interrogation (?) indique que l'attribut peut ne pas être présent dans l'objet.

### `platform.initStart`

Un événement `platform.initStart` indique que la phase d'initialisation de la fonction a commencé. Un objet `Event` `platform.initStart` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

L'objet `PlatformInitStart` possède les attributs suivants :

- functionName – `String`
- functionVersion – `String`
- initializationType— `the section called "InitType"` objet
- instanceId? – `String`
- instanceMaxMemory? – `Integer`
- phase – Objet `the section called "InitPhase"`

- runtimeVersion? – `String`
- runtimeVersionArn? – `String`

Voici un exemple de `Event` de type `platform.initStart` :

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.initStart",
    "record": {
        "initializationType": "on-demand",
        "phase": "init",
        "runtimeVersion": "nodejs-14.v3",
        "runtimeVersionArn": "arn",
        "functionName": "myFunction",
        "functionVersion": "$LATEST",
        "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
        "instanceMaxMemory": 256
    }
}
```

### `platform.initRuntimeDone`

Un événement `platform.initRuntimeDone` indique que la phase d'initialisation de la fonction est terminée. Un objet `Event` `platform.initRuntimeDone` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone
```

L'objet `PlatformInitRuntimeDone` possède les attributs suivants :

- initializationType— the section called "InitType" objet
- phase – Objet the section called "InitPhase"
- status – Objet the section called "Status"
- spans? – Liste d'objets the section called "Span"

Voici un exemple de `Event` de type `platform.initRuntimeDone` :

```
{
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.initRuntimeDone",
    "record": {
        "initializationType": "on-demand"
        "status": "success",
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-06-02T12:02:33.913Z",
                "durationMs": 70.5
            }
        ]
    }
}
```

### `platform.initReport`

Un événement `platform.initReport` contient un rapport global de la phase d'initialisation de la fonction. Un objet `Event` `platform.initReport` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport
```

L'objet `PlatformInitReport` possède les attributs suivants :

- errorType? — chaîne
- initializationType— the section called "InitType" objet
- phase – Objet the section called "InitPhase"
- metrics – Objet the section called "InitReportMetrics"
- spans? – Liste d'objets the section called "Span"
- status – Objet the section called "Status"

Voici un exemple de `Event` de type `platform.initReport` :

```
{
    "time": "2022-10-12T00:01:15.000Z",
```

```
    "type": "platform.initReport",
    "record": {
        "initializationType": "on-demand",
        "status": "success",
        "phase": "init",
        "metrics": {
            "durationMs": 125.33
        },
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-06-02T12:02:33.913Z",
                "durationMs": 90.1
            }
        ]
    }
}
```

### `platform.start`

Un événement `platform.start` indique que la phase d'invocation de la fonction a commencé. Un objet `Event` `platform.start` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart
```

L'objet `PlatformStart` possède les attributs suivants :

- requestId – `String`
- version? – `String`
- tracing? – the section called "TraceContext"

Voici un exemple de `Event` de type `platform.start` :

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.start",
    "record": {
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
```

```
        "version": "$LATEST",
        "tracing": {
            "spanId": "54565fb41ac79632",
            "type": "X-Amzn-Trace-Id",
            "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
        }
    }
}
```

### `platform.runtimeDone`

Un événement `platform.runtimeDone` indique que la phase d'invocation de la fonction est terminée. Un objet `Event` `platform.runtimeDone` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

L'objet `PlatformRuntimeDone` possède les attributs suivants :

- errorType? – `String`
- metrics? – Objet the section called "RuntimeDoneMetrics"
- requestId – `String`
- status – Objet the section called "Status"
- spans? – Liste d'objets the section called "Span"
- tracing? – Objet the section called "TraceContext"

Voici un exemple de `Event` de type `platform.runtimeDone` :

```
{
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.runtimeDone",
    "record": {
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
        "status": "success",
        "tracing": {
            "spanId": "54565fb41ac79632",
```

```
            "type": "X-Amzn-Trace-Id",
            "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
        },
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 80.0
            }
        ],
        "metrics": {
            "durationMs": 140.0,
            "producedBytes": 16
        }
    }
}
```

### `platform.report`

Un `platform.report` événement contient un rapport global de la phase d'appel de la fonction. Un objet `Event` `platform.report` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

L'objet `PlatformReport` possède les attributs suivants :

- metrics – Objet the section called "ReportMetrics"
- requestId – `String`
- spans? – Liste d'objets the section called "Span"
- status – Objet the section called "Status"
- tracing? – Objet the section called "TraceContext"

Voici un exemple de `Event` de type `platform.report` :

```
{
    "time": "2022-10-12T00:01:15.000Z",
```

```
    "type": "platform.report",
    "record": {
        "metrics": {
            "billedDurationMs": 694,
            "durationMs": 693.92,
            "initDurationMs": 397.68,
            "maxMemoryUsedMB": 84,
            "memorySizeMB": 128
        },
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
    }
}
```

### `platform.restoreStart`

Un événement `platform.restoreStart` indique qu'un événement de restauration d'environnement de fonction a commencé. Dans un événement de restauration d'environnement, Lambda crée l'environnement à partir d'un instantané mis en cache plutôt que de l'initialiser à partir de zéro. Pour de plus amples informations, veuillez consulter Lambda SnapStart. Un objet `Event` `platform.restoreStart` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

L'objet `PlatformRestoreStart` possède les attributs suivants :

- functionName – `String`
- functionVersion – `String`
- instanceId? – `String`
- instanceMaxMemory? – `String`
- runtimeVersion? – `String`
- runtimeVersionArn? – `String`

Voici un exemple de `Event` de type `platform.restoreStart` :

```
{
    "time": "2022-10-12T00:00:15.064Z",
```

```
    "type": "platform.restoreStart",
    "record": {
        "runtimeVersion": "nodejs-14.v3",
        "runtimeVersionArn": "arn",
        "functionName": "myFunction",
        "functionVersion": "$LATEST",
        "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
        "instanceMaxMemory": 256
    }
}
```

### `platform.restoreRuntimeDone`

Un événement `platform.restoreRuntimeDone` indique qu'un événement de restauration d'environnement de fonction s'est terminé. Dans un événement de restauration d'environnement, Lambda crée l'environnement à partir d'un instantané mis en cache plutôt que de l'initialiser à partir de zéro. Pour de plus amples informations, veuillez consulter Lambda SnapStart. Un objet `Event` `platform.restoreRuntimeDone` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

L'objet `PlatformRestoreRuntimeDone` possède les attributs suivants :

- errorType? – `String`
- spans? – Liste d'objets the section called "Span"
- status – Objet the section called "Status"

Voici un exemple de `Event` de type `platform.restoreRuntimeDone` :

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.restoreRuntimeDone",
    "record": {
        "status": "success",
        "spans": [
            {
                "name": "someTimeSpan",
```

```
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 80.0
            }
        ]
    }
}
```

### `platform.restoreReport`

Un événement `platform.restoreReport` contient un rapport global d'un événement de restauration de fonction. Un objet `Event` `platform.restoreReport` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

L'objet `PlatformRestoreReport` possède les attributs suivants :

- errorType? — chaîne
- metrics? – Objet the section called "RestoreReportMetrics"
- spans? – Liste d'objets the section called "Span"
- status – Objet the section called "Status"

Voici un exemple de `Event` de type `platform.restoreReport` :

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.restoreReport",
    "record": {
        "status": "success",
        "metrics": {
            "durationMs": 15.19
        },
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 30.0
            }
```

```
        ]
    }
}
```

### platform.extension

Un événement `extension` contient des journaux provenant du code d'extension. Un objet `Event` `extension` a la forme suivante :

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

L'objet `PlatformExtension` possède les attributs suivants :

- events – Liste de `String`
- nom – `String`
- state – `String`

Voici un exemple de `Event` de type `platform.extension` :

```
{
    "time": "2022-10-12T00:02:15.000Z",
    "type": "platform.extension",
    "record": {
        "events": [ "INVOKE", "SHUTDOWN" ],
        "name": "my-telemetry-extension",
        "state": "Ready"
    }
}
```

### platform.telemetrySubscription

Un événement `platform.telemetrySubscription` contient des informations sur un abonnement d'extension. Un objet `Event` `platform.telemetrySubscription` a la forme suivante :

```
Event: Object
- time: String
```

```
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

L'objet `PlatformTelemetrySubscription` possède les attributs suivants :

- nom – `String`
- state – `String`
- types – Liste de `String`

Voici un exemple de `Event` de type `platform.telemetrySubscription` :

```
{
    "time": "2022-10-12T00:02:35.000Z",
    "type": "platform.telemetrySubscription",
    "record": {
        "name": "my-telemetry-extension",
        "state": "Subscribed",
        "types": [ "platform", "function" ]
    }
}
```

### `platform.logsDropped`

Un événement `platform.logsDropped` contient des informations sur les événements abandonnés. Lambda émet un `platform.logsDropped` événement lorsqu'une fonction produit des logs à un taux trop élevé pour que Lambda puisse les traiter. Lorsque Lambda ne parvient pas à envoyer des journaux vers CloudWatch ou vers l'extension abonnée à Telemetry API au rythme où la fonction les produit, il supprime les journaux pour empêcher l'exécution de la fonction de ralentir. Un objet `Event` `platform.logsDropped` a la forme suivante :

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

L'objet `PlatformLogsDropped` possède les attributs suivants :

- droppedBytes – `Integer`
- droppedRecords – `Integer`

- reason – `String`

Voici un exemple de `Event` de type `platform.logsDropped` :

```
{
    "time": "2022-10-12T00:02:35.000Z",
    "type": "platform.logsDropped",
    "record": {
        "droppedBytes": 12345,
        "droppedRecords": 123,
        "reason": "Some logs were dropped because the downstream consumer is slower
 than the logs production rate"
    }
}
```

### `function`

Un événement `function` contient des journaux provenant du code de fonction. Un objet `Event` `function` a la forme suivante :

```
Event: Object
- time: String
- type: String = function
- record: {}
```

Le format du `record` champ varie selon que les journaux de votre fonction sont formatés en texte brut ou au formatJSON. Pour en savoir plus sur les options de configuration du format des journaux, voir the section called “Configuration JSON et formats de journal en texte brut”

Voici un exemple `Event` de type `function` où le format du journal est en texte brut :

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": "[INFO] Hello world, I am a function!"
}
```

Voici un exemple `Event` de type `function` où le format du journal est le suivant JSON :

```
{
```

```
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": {
        "timestamp": "2022-10-12T00:03:50.000Z",
        "level": "INFO",
        "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
        "message": "Hello world, I am a function!"
    }
}
```

> **Note**
>
> Si la version du schéma que vous utilisez est plus ancienne que la `2022-12-13` version, elle `"record"` est toujours affichée sous forme de chaîne, même lorsque le format de journalisation de votre fonction est configuré comme suitJSON.

### `extension`

Un événement `extension` contient des journaux provenant du code d'extension. Un objet `Event` `extension` a la forme suivante :

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

Le format du `record` champ varie selon que les journaux de votre fonction sont formatés en texte brut ou au formatJSON. Pour en savoir plus sur les options de configuration du format des journaux, voir the section called "Configuration JSON et formats de journal en texte brut"

Voici un exemple `Event` de type `extension` où le format du journal est en texte brut :

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": "[INFO] Hello world, I am an extension!"
}
```

Voici un exemple `Event` de type `extension` où le format du journal est le suivant JSON :

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": {
       "timestamp": "2022-10-12T00:03:50.000Z",
       "level": "INFO",
       "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
       "message": "Hello world, I am an extension!"
    }
}
```

> **Note**
>
> Si la version du schéma que vous utilisez est plus ancienne que la `2022-12-13` version, elle `"record"` est toujours affichée sous forme de chaîne, même lorsque le format de journalisation de votre fonction est configuré comme suitJSON.

## Types d’objets partagés

Cette section détaille les types d'objets partagés pris en charge par la API télémétrie Lambda.

### `InitPhase`

Une enum de chaînes de caractères qui décrit la phase où l’étape d’initialisation se produit. Dans la plupart des cas, Lambda exécute le code d’initialisation de la fonction pendant la phase `init`. Toutefois, dans certains cas d’erreur, Lambda peut réexécuter le code d’initialisation de la fonction pendant la phase `invoke`. (Cela s’appelle une init supprimée.)

- Type – `String`
- Valeurs valides : `init|invoke|snap-start`

### `InitReportMetrics`

Un objet qui contient des métriques sur une phase d’initialisation.

- Type – `Object`

Un objet `InitReportMetrics` a la forme suivante :

```
InitReportMetrics: Object
- durationMs: Double
```

Voici un exemple d'objet `InitReportMetrics` :

```
{
    "durationMs": 247.88
}
```

### `InitType`

Une enum de chaînes de caractères qui décrit comment Lambda a initialisé l'environnement.

- Type – `String`
- Valeurs valides – `on-demand`|`provisioned-concurrency`

### `ReportMetrics`

Un objet qui contient des métriques sur une phase terminée.

- Type – `Object`

Un objet `ReportMetrics` a la forme suivante :

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

Voici un exemple d'objet `ReportMetrics` :

```
{
    "billedDurationMs": 694,
    "durationMs": 693.92,
    "initDurationMs": 397.68,
    "maxMemoryUsedMB": 84,
```

```
    "memorySizeMB": 128
}
```

### RestoreReportMetrics

Un objet qui contient des métriques sur une phase de restauration terminée.

- Type – `Object`

Un objet `RestoreReportMetrics` a la forme suivante :

```
RestoreReportMetrics: Object
- durationMs: Double
```

Voici un exemple d'objet `RestoreReportMetrics` :

```
{
    "durationMs": 15.19
}
```

### RuntimeDoneMetrics

Un objet qui contient des métriques sur une phase d'invocation.

- Type – `Object`

Un objet `RuntimeDoneMetrics` a la forme suivante :

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

Voici un exemple d'objet `RuntimeDoneMetrics` :

```
{
    "durationMs": 200.0,
    "producedBytes": 15
}
```

### Span

Un objet qui contient des détails sur un span. Un span représente une unité de travail ou une opération dans une trace. Pour plus d'informations sur les intervalles, consultez Span sur la API page de suivi du site Web de OpenTelemetry Docs.

Lambda prend en charge les span suivants pour l'événement `platform.RuntimeDone` :

- Le span `responseLatency` décrit le temps qu'il a fallu à votre fonction Lambda pour commencer à envoyer la réponse.
- Le span `responseDuration` décrit le temps qu'il a fallu à votre fonction Lambda pour finir d'envoyer la réponse entière.
- Le span `runtimeOverhead` décrit le temps qu'il a fallu à l'environnement d'exécution Lambda pour signaler qu'il était prêt à traiter la prochaine invocation de fonction. C'est le temps qu'il a fallu au moteur d'exécution pour appeler le prochain appel API afin d'obtenir le prochain événement après avoir renvoyé la réponse de votre fonction.

Voici un exemple d'objet span `responseLatency` :

```
{
        "name": "responseLatency",
        "start": "2022-08-02T12:01:23.521Z",
        "durationMs": 23.02
      }
```

### Status

Un objet qui décrit le statut d'une phase d'initialisation ou d'invocation. Si le statut est `failure` ou `error`, l'objet `Status` contient également un champ `errorType` décrivant l'erreur.

- Type – `Object`
- Valeurs de status valides : `success|failure|error|timeout`

### TraceContext

Un objet qui décrit les propriétés d'une trace.

- Type – `Object`

Un objet `TraceContext` a la forme suivante :

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

Voici un exemple d'objet `TraceContext` :

```
{
    "spanId": "073a49012f3c312e",
    "type": "X-Amzn-Trace-Id",
    "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

### `TracingType`

Un enum de chaînes de caractères qui décrit le type de trace dans un objet the section called "TraceContext".

- Type – `String`
- Valeurs valides – `X-Amzn-Trace-Id`

# Conversion d'objets de télémétrie API **`Event`** Lambda en Spans OpenTelemetry

Le API schéma de AWS Lambda télémétrie est sémantiquement compatible avec (). OpenTelemetry OTel Cela signifie que vous pouvez convertir vos API `Event` objets de AWS Lambda télémétrie en OpenTelemetry (OTel) Spans. Lors de la conversion, vous ne devez pas mapper un seul `Event` objet à un seul OTel Span. Vous devez plutôt présenter les trois événements liés à une phase du cycle de vie dans un seul OTel intervalle. Par exemple, les événements `start`, `runtimeDone` et `runtimeReport` représentent un seul appel de fonction. Présentez ces trois événements sous la forme d'un seul OTel Span.

Vous pouvez convertir vos événements en utilisant des événements Span ou des Spans enfants (imbriqués). Les tableaux de cette page décrivent les mappages entre les propriétés du API schéma de télémétrie et les propriétés OTel Span pour les deux approches. Pour plus d'informations sur OTel les Spans, consultez Span sur la API page de suivi du site Web de OpenTelemetry Docs.

Sections


- Carte de OTel Spans avec Span Events
- Carte des OTel travées avec des travées pour les enfants


## Carte de OTel Spans avec Span Events

Dans les tableaux suivants, e représente l'événement provenant de la source de télémétrie.

Cartographie des événements *Start

| OpenTelemetry | Schéma de télémétrie Lambda API |
|---|---|
| `Span.Name` | Votre extension génère cette valeur sur la base du champ `type`. |
| `Span.StartTime` | Utilisez `e.time`. |
| `Span.EndTime` | S/O, car l'événement n'est pas encore terminé. |
| `Span.Kind` | Définie sur `Server`. |
| `Span.Status` | Définie sur `Unset`. |

| OpenTelemetry | Schéma de télémétrie Lambda API |
| --- | --- |
| `Span.TraceId` | Analysez l' AWS X-Ray en-tête trouvé dans`e.tracing.value` , puis utilisez la `TraceId` valeur. |
| `Span.ParentId` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Parent`. |
| `Span.SpanId` | Utilisez `e.tracing.spanId` si disponible. Sinon, générer un nouveau `SpanId`. |
| `Span.SpanContext.TraceState` | S/O pour un contexte de trace X-Ray. |
| `Span.SpanContext.TraceFlags` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Sampled`. |
| `Span.Attributes` | Votre extension peut ajouter toute valeur personnalisée ici. |

Cartographie des RuntimeDone événements*

| OpenTelemetry | Schéma de télémétrie Lambda API |
| --- | --- |
| `Span.Name` | Votre extension génère la valeur basée sur le champ `type`. |
| `Span.StartTime` | Utilisez `e.time` à partir de l'événement `*Start` correspondant.<br><br>Sinon, utilisez `e.time - e.metrics.durationMs` . |
| `Span.EndTime` | S/O, car l'événement n'est pas encore terminé. |
| `Span.Kind` | Définie sur `Server`. |

| OpenTelemetry | Schéma de télémétrie Lambda API |
| --- | --- |
| `Span.Status` | Si `e.status` n'est pas égal à `success`, mettez `Error`.<br><br>Sinon, définissez sur `Ok`. |
| `Span.Events[]` | Utilisez `e.spans[]` . |
| `Span.Events[i].Name` | Utilisez `e.spans[i].name` . |
| `Span.Events[i].Time` | Utilisez `e.spans[i].start` . |
| `Span.TraceId` | Analysez l' AWS X-Ray en-tête trouvé dans`e.tracing.value` , puis utilisez la `TraceId` valeur. |
| `Span.ParentId` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Parent`. |
| `Span.SpanId` | Utilisez le même `SpanId` de l'événement `*Start`. S'il n'est pas disponible, utilisez alors `e.tracing.spanId` , ou générez un nouveau `SpanId`. |
| `Span.SpanContext.TraceState` | S/O pour un contexte de trace X-Ray. |
| `Span.SpanContext.TraceFlags` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Sampled`. |
| `Span.Attributes` | Votre extension peut ajouter toute valeur personnalisée ici. |

Cartographie des événements *Report

| OpenTelemetry | Schéma de télémétrie Lambda API |
| --- | --- |
| `Span.Name` | Votre extension génère la valeur basée sur le champ `type`. |
| `Span.StartTime` | Utilisez `e.time` à partir de l'événement `*Start` correspondant.<br><br>Sinon, utilisez `e.time - e.metrics.durationMs` . |
| `Span.EndTime` | Utilisez `e.time`. |
| `Span.Kind` | Définie sur `Server`. |
| `Span.Status` | Utilisez la même valeur que l'événement `*RuntimeDone` . |
| `Span.TraceId` | Analysez l' AWS X-Ray en-tête trouvé dans`e.tracing.value` , puis utilisez la `TraceId` valeur. |
| `Span.ParentId` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Parent`. |
| `Span.SpanId` | Utilisez le même `SpanId` de l'événement `*Start`. S'il n'est pas disponible, utilisez alors `e.tracing.spanId` , ou générez un nouveau `SpanId`. |
| `Span.SpanContext.TraceState` | S/O pour un contexte de trace X-Ray. |
| `Span.SpanContext.TraceFlags` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Sampled`. |
| `Span.Attributes` | Votre extension peut ajouter toute valeur personnalisée ici. |

## Carte des OTel travées avec des travées pour les enfants

Le tableau suivant décrit comment convertir les API événements de télémétrie Lambda en OTel intervalles avec intervalles secondaires (imbriqués) pour intervalles. `*RuntimeDone` Pour les mappages `*Start` et `*Report`, reportez-vous aux tableaux de the section called “Carte de OTel Spans avec Span Events”, car ils sont les mêmes pour les Spans enfant. Dans cette table, e représente l'événement provenant de la source de télémétrie.

Cartographie des RuntimeDone événements*

| OpenTelemetry | Schéma de télémétrie Lambda API |
|---|---|
| `Span.Name` | Votre extension génère la valeur basée sur le champ `type`. |
| `Span.StartTime` | Utilisez `e.time` à partir de l'événement `*Start` correspondant.<br><br>Sinon, utilisez `e.time - e.metrics.durationMs` . |
| `Span.EndTime` | S/O, car l'événement n'est pas encore terminé. |
| `Span.Kind` | Définie sur `Server`. |
| `Span.Status` | Si `e.status` n'est pas égal à `success`, mettez `Error`.<br><br>Sinon, définissez sur `Ok`. |
| `Span.TraceId` | Analysez l' AWS X-Ray en-tête trouvé dans`e.tracing.value` , puis utilisez la `TraceId` valeur. |
| `Span.ParentId` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Parent`. |
| `Span.SpanId` | Utilisez le même `SpanId` de l'événement `*Start`. S'il n'est pas disponible, utilisez |

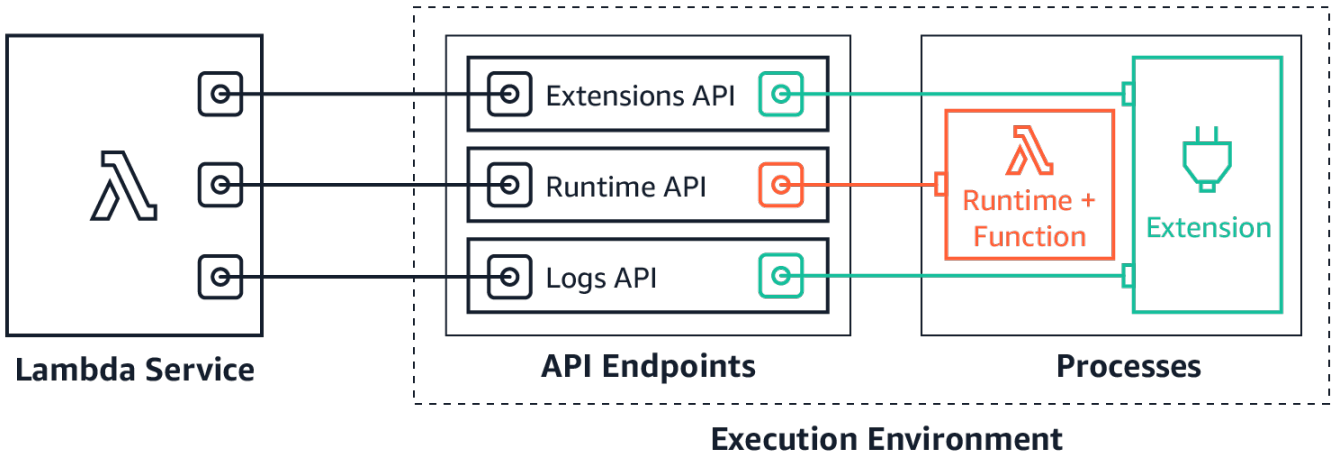
| OpenTelemetry | Schéma de télémétrie Lambda API |
| --- | --- |
| | alors `e.tracing.spanId` , ou générez un nouveau `SpanId`. |
| `Span.SpanContext.TraceState` | S/O pour un contexte de trace X-Ray. |
| `Span.SpanContext.TraceFlags` | Analysez l'en-tête X-Ray trouvé dans `e.tracing.value` , puis utilisez la valeur `Sampled`. |
| `Span.Attributes` | Votre extension peut ajouter toute valeur personnalisée ici. |
| `ChildSpan[i].Name` | Utilisez `e.spans[i].name` . |
| `ChildSpan[i].StartTime` | Utilisez `e.spans[i].start` . |
| `ChildSpan[i].EndTime` | Utilisez `e.spans[i].start + e.spans[i].durations` . |
| `ChildSpan[i].Kind` | Identique au parent `Span.Kind` . |
| `ChildSpan[i].Status` | Identique au parent `Span.Status` . |
| `ChildSpan[i].TraceId` | Identique au parent `Span.TraceId` . |
| `ChildSpan[i].ParentId` | Utilisez le parent `Span.SpanId` . |
| `ChildSpan[i].SpanId` | Générer un nouveau `SpanId`. |
| `ChildSpan[i].SpanContext.TraceState` | S/O pour un contexte de trace X-Ray. |
| `ChildSpan[i].SpanContext.TraceFlags` | Identique au parent `Span.SpanContext.TraceFlags` . |

# Utilisation des journaux Lambda API

> **Important**
>
> La télémétrie Lambda remplace API les journaux Lambda. API Bien que les journaux API restent entièrement fonctionnels, nous vous recommandons de n'utiliser que la télémétrie à l'APIavenir. Vous pouvez abonner votre extension à un flux de télémétrie à l'aide de la télémétrie API ou des journaux. API Une fois que vous vous êtes abonné à l'aide de l'un d'APIsentre eux, toute tentative de souscription à l'aide de l'autre API renvoie une erreur.

Lambda capture automatiquement les journaux d'exécution et les diffuse vers Amazon. CloudWatch Ce flux de journaux contient les journaux que génèrent votre code de fonction et vos extensions, ainsi que les journaux que Lambda génère dans le cadre de l'appel de fonction.

Les extensions Lambda peuvent utiliser les journaux d'exécution Lambda API pour s'abonner à des flux de journaux directement depuis l'environnement d'exécution Lambda. Lambda diffuse les journaux vers l'extension qui peut ensuite les traiter, les filtrer et les envoyer à n'importe quelle destination de prédilection.




The Logs API permet aux extensions de s'abonner à trois flux de logs différents :

- Journaux de fonction que la fonction Lambda génère et écrit dans `stdout` ou `stderr`.
- Journaux d'extension que le code d'extension génère.
- Journaux de plateforme Lambda qui enregistrent les événements et les erreurs liés aux appels et aux extensions.

> **Note**
>
> Lambda envoie tous les journaux à CloudWatch, même lorsqu'une extension s'abonne à un ou plusieurs flux de journaux.

Rubriques


- S'abonner pour recevoir des journaux
- Utilisation de la mémoire
- Protocoles de destination
- Configuration de mise en mémoire tampon
- Exemple d'abonnement
- Exemple de code pour Logs API
- APIRéférence des journaux
- Messages de journaux


## S'abonner pour recevoir des journaux

Une extension Lambda peut s'abonner pour recevoir des journaux en envoyant une demande d'abonnement aux journaux. API

Pour vous abonner afin de recevoir des journaux, vous avez besoin de l'identifiant d'extension (`Lambda-Extension-Identifier`). Enregistrez d'abord l'extension pour recevoir l'identifiant de l'extension. Abonnez-vous ensuite aux journaux API lors de l'initialisation. Une fois la phase d'initialisation terminée, Lambda ne traite pas les demandes d'abonnement.

> **Note**
>
> L'APIabonnement aux journaux est idempotent. Les demandes d'abonnement en double n'entraînent pas de doublons d'abonnements.

## Utilisation de la mémoire

L'utilisation de la mémoire augmente de façon linéaire à mesure que le nombre d'abonnés augmente. Les abonnements consomment des ressources de mémoire car chaque abonnement ouvre un

nouveau tampon mémoire pour stocker les journaux. Pour optimiser l'utilisation de la mémoire tampon, vous pouvez ajuster la configuration de la mise en mémoire tampon. L'utilisation de la mémoire tampon compte pour la consommation globale de mémoire dans l'environnement d'exécution.

### Protocoles de destination

Vous pouvez choisir l'un des protocoles suivants pour recevoir les journaux :

1. HTTP(recommandé) — Lambda fournit des journaux à un point de HTTP terminaison local (`http://sandbox.localdomain:${PORT}/${PATH}`) sous forme de tableau d'enregistrements au JSON format. Le paramètre `$PATH` est facultatif. Notez que seul HTTP est pris en charge, pasHTTPS. Vous pouvez choisir de recevoir les journaux via PUT ouPOST.
2. TCP— Lambda envoie les journaux à un TCP port au format Newline délimité JSON (). NDJSON

Nous vous recommandons d'utiliser HTTP plutôt queTCP. AinsiTCP, la plateforme Lambda ne peut pas accuser réception lorsqu'elle fournit des journaux à la couche application. Vous risquez par conséquent de perdre des journaux si votre extension se bloque. HTTPne partage pas cette limitation.

Nous vous recommandons également de configurer l'HTTPécouteur local ou le TCP port avant de vous abonner pour recevoir des journaux. Au cours de l'installation, notez ce qui suit :

- Lambda n'envoie de journaux qu'à des destinations au sein de l'environnement d'exécution.
- Lambda tente à nouveau d'envoyer les journaux (avec interruption) s'il n'y a pas d'écouteur ou si la PUT demande POST ou entraîne une erreur. Si l'abonné aux journaux se bloque, il continue de recevoir des journaux après que Lambda a redémarré l'environnement d'exécution.
- Lambda réserve le port 9001. Il n'y a pas d'autres restrictions ou recommandations relatives au numéro de port.

### Configuration de mise en mémoire tampon

Lambda peut mettre les journaux en mémoire tampon avant de les livrer à l'abonné. Vous pouvez configurer ce comportement dans la demande d'abonnement en spécifiant les champs facultatifs suivants. Notez que Lambda utilise la valeur par défaut pour tout champ que vous ne spécifiez pas.

- timeoutMs— Durée maximale (en millisecondes) de mise en mémoire tampon d'un lot. Par défaut : 1 000. Minimum : 25. Maximum : 30 000.

- maxBytes— Taille maximale (en octets) des journaux à mettre en mémoire tampon. Par défaut : 262 144. Minimum : 262 144. Maximum : 1 048 576.
- maxItems— Le nombre maximum d'événements à mettre en mémoire tampon. Par défaut : 10 000. Minimum : 1 000. Maximum : 10 000.

Lors de la configuration de la mise en mémoire tampon, notez les points suivants :

- Lambda vide les journaux en cas de fermeture de l'un des flux d'entrée, par exemple, si le runtime se bloque.
- Chaque abonné peut spécifier une configuration de mise en mémoire tampon différente dans sa demande d'abonnement.
- Tenez compte de la taille de tampon dont vous avez besoin pour lire les données. Attendez-vous à recevoir des charges utiles aussi volumineuses que `2*maxBytes+metadata`, où `maxBytes` est configuré dans la demande d'abonnement. Par exemple, Lambda ajoute les octets de métadonnées suivants à chaque enregistrement :

  ```
  {
  "time": "2020-08-20T12:31:32.123Z",
  "type": "function",
  "record": "Hello World"
  }
  ```

- Si l'abonné ne peut pas traiter les journaux entrants assez rapidement, Lambda peut supprimer les journaux pour limiter l'utilisation de la mémoire. Pour indiquer le nombre d'enregistrements supprimés, Lambda envoie un journal `platform.logsDropped`. Pour de plus amples informations, veuillez consulter the section called “Lambda : certains journaux de ma fonction n'apparaissent pas”.

## Exemple d'abonnement

L'exemple suivant montre une demande d'abonnement aux journaux de la plateforme et des fonctions.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs HTTP/1.1
{ "schemaVersion": "2020-08-15",
  "types": [
      "platform",
      "function"
```

```
    ],
  "buffering": {
      "maxItems": 1000,
      "maxBytes": 262144,
      "timeoutMs": 100
    },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080/lambda_logs"
  }
}
```

Si la demande aboutit, l'abonné reçoit une réponse positive de HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

## Exemple de code pour Logs API

Pour un exemple de code montrant comment envoyer des journaux vers une destination personnalisée, consultez la section Utilisation d' AWS Lambda extensions pour envoyer des journaux vers des destinations personnalisées sur le blog AWS Compute.

Pour des exemples de code Python et Go montrant comment développer une extension Lambda de base et s'abonner aux journauxAPI, voir AWS Lambda Extensions sur le référentiel AWS Samples GitHub . Pour plus d'informations sur la génération d'une extension Lambda, consultez the section called “Prolongations API”.

## APIRéférence des journaux

Vous pouvez récupérer le point de API terminaison Logs à partir de la variable d'`AWS_LAMBDA_RUNTIME_API`environnement. Pour envoyer une API demande, utilisez le préfixe `2020-08-15/` avant le API chemin. Par exemple :

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

La API spécification Open pour la API version Logs 2020-08-15 est disponible ici : .zip logs-api-request

S’abonner

Pour s'abonner à un ou plusieurs flux de journaux disponibles dans l'environnement d'exécution Lambda, les extensions envoient une demande API Subscribe.

Chemin – `/logs`

Méthode — PUT

Paramètres de corps

`destination` – Voir the section called “Protocoles de destination”. Obligatoire : oui. Type : chaînes.

`buffering` – Voir the section called “Configuration de mise en mémoire tampon”. Requis : non. Type : chaînes.

`types` – Tableau des types de journaux à recevoir. Obligatoire : oui. Type : tableau de chaînes. Valeurs valides : « plateforme », « fonction», « extension ».

`schemaVersion` – Obligatoire : non. Valeur par défaut : « 2020-08-15 ». Attribuez la valeur « 2021-03-18 » pour que l'extension reçoive les messages `platform.runtimeDone`.

Paramètres de réponse

Les API spécifications Open pour les réponses aux abonnements, version 2020-08-15, sont disponibles pour les HTTP protocoles et : TCP

- HTTP: logs-api-http-response.zip
- TCP: logs-api-tcp-response.zip

Codes de réponse

- 200 – Demande effectuée avec succès.
- 202 – Demande acceptée. Réponse à une demande d'abonnement pendant les tests locaux.
- 4XX – Demande erronée.
- 500 – Erreur de service.

Si la demande aboutit, l'abonné reçoit une réponse positive de HTTP 200.

```
HTTP/1.1 200 OK
```

```
"OK"
```

Si la demande échoue, l'abonné reçoit une réponse d'erreur. Exemples :

```
HTTP/1.1 400 OK
{
    "errorType": "Logs.ValidationError",
    "errorMessage": URI port is not provided; types should not be empty"
}
```

## Messages de journaux

The Logs API permet aux extensions de s'abonner à trois flux de logs différents :

- Fonction – Journaux que la fonction Lambda génère et écrit dans `stdout` ou `stderr`.
- Extension – Journaux que le code d'extension génère.
- Plateforme – Journaux que la plateforme de runtime génère, qui consignent des événements et des erreurs liés aux appels et aux extensions.

Rubriques


- Journaux de fonctions
- Journaux d'extension
- Journaux de plateforme


### Journaux de fonctions

La fonction Lambda et les extensions internes génèrent des journaux de fonction et les écrivent dans `stdout` ou `stderr`.

L'exemple suivant montre le format d'un message du journal des fonctions. {« time » : « 2020-08-20T 12:31:32.123 Z », « type » : « function », « record » : « found ». ERROR Stack trace:\n\my-function (line 10)\n" }

### Journaux d'extension

Les extensions peuvent générer des journaux d'extensions. Le format du journal est le même que pour un journal de fonction.

### Journaux de plateforme

Lambda génère des messages de journal pour des événements de plateforme tels que `platform.start`, `platform.end` et `platform.fault`.

Vous pouvez éventuellement vous abonner à la version 18/03/2021 du API schéma Logs, qui inclut le message du `platform.runtimeDone` journal.

#### Exemple de messages du journal de la plateforme

L'exemple suivant montre les journaux de début et de fin de la plateforme. Ces journaux indiquent l'heure de début de l'invocation et l'heure de fin de l'invocation spécifiée. requestId

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.start",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.end",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
```

La plateforme. initRuntimeDoneun message de journal indique l'état de la `Runtime init` sous-phase, qui fait partie de la phase du cycle de vie d'initialisation. En cas `Runtime init` de succès, le moteur d'exécution envoie une API demande `/next` d'exécution (pour les types d'`provisioned-concurrency`initialisation `on-demand` et) ou `restore/next` (pour le type d'`snap-start`initialisation). L'exemple suivant montre une plateforme performante. initRuntimeDonemessage de journal pour le type `snap-start` d'initialisation.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initRuntimeDone",
  "record":{
      "initializationType":"snap-start",
      "status":"success"
  }
}
```

La plateforme. initReportun message de journal indique la durée de la `Init` phase et le nombre de millisecondes qui vous ont été facturés pendant cette phase. Lorsque le type d'initialisation est `provisioned-concurrency`, Lambda envoie ce message pendant l'appel. Lorsque le type d'initialisation est `snap-start`, Lambda envoie ce message après avoir restauré l'instantané. L'exemple suivant montre une plate-forme. initReportmessage de journal pour le type `snap-start` d'initialisation.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initReport",
  "record":{
      "initializationType":"snap-start",
      "metrics":{
          "durationMs":731.79,
          "billedDurationMs":732
          }
  }
}
```

Le journal des rapports de la plateforme inclut des mesures relatives à l'invocation spécifiée requestId . Le champ `initDurationMs` n'est inclus dans le journal que si l'appel comprend un démarrage à froid. Si le suivi AWS X-Ray est actif, le journal inclut les métadonnées X-Ray. L'exemple suivant montre un journal de rapport de la plateforme pour un appel qui comprend un démarrage à froid.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.report",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56",
        "metrics": {"durationMs": 101.51,
            "billedDurationMs": 300,
            "memorySizeMB": 512,
            "maxMemoryUsedMB": 33,
            "initDurationMs": 116.67
        }
    }
}
```

Le journal des pannes de la plateforme capture les erreurs d'exécution ou d'environnement d'exécution. L'exemple suivant montre un message de journal des erreurs de plateforme.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.fault",
    "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
 completing request"
}
```

> **Note**
>
> AWS met actuellement en œuvre des modifications du service Lambda. En raison de ces modifications, vous pouvez constater des différences mineures entre la structure et le contenu des messages du journal système et des segments de trace émis par les différentes fonctions Lambda de votre ordinateur. Compte AWS
> L'une des sorties du journal concernées par cette modification est le `"record"` champ du journal des défaillances de la plate-forme. Les exemples suivants montrent des `"record"` champs illustratifs dans les anciens et les nouveaux formats. Le nouveau style de journal des défaillances contient un message plus concis
> Ces modifications seront mises en œuvre au cours des prochaines semaines, et toutes les fonctions, Régions AWS sauf en Chine et dans les GovCloud régions, passeront à l'utilisation du nouveau format des messages de journal et des segments de trace.

Example enregistrement du journal des défaillances de la plate-forme (ancien style)

```
"record":"RequestId: ...\tError: Runtime exited with error: exit status
 255\nRuntime.ExitError"
```

Example enregistrement du journal des défaillances de la plate-forme (nouveau style)

```
"record":"RequestId: ... Status: error\tErrorType: Runtime.ExitError"
```

Lambda génère un journal des extensions de plate-forme lorsqu'une extension s'enregistre auprès des extensions. API L'exemple suivant montre un message d'extension de la plateforme.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.extension",
```

```
    "record": {"name": "Foo.bar",
        "state": "Ready",
        "events": ["INVOKE", "SHUTDOWN"]
     }
}
```

Lambda génère un journal d'abonnement aux journaux de plate-forme lorsqu'une extension s'abonne aux journaux. API L'exemple suivant montre un message d'abonnement aux journaux.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsSubscription",
    "record": {"name": "Foo.bar",
        "state": "Subscribed",
        "types": ["function", "platform"],
    }
}
```

Lambda génère un journal des journaux de plateforme supprimés quand une extension n'est pas en mesure de traiter le nombre de journaux qu'elle reçoit. L'exemple suivant montre un message de journal `platform.logsDropped`.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsDropped",
    "record": {"reason": "Consumer seems to have fallen behind as it has not
 acknowledged receipt of logs.",
        "droppedRecords": 123,
        "droppedBytes" 12345
    }
}
```

La plateforme. restoreStartle message de journal indique l'heure à laquelle la `Restore` phase a commencé (type `snap-start` d'initialisation uniquement). Exemple :

```
{
  "time":"2022-07-17T18:43:44.782Z",
  "type":"platform.restoreStart",
  "record":{}
}
```

La plateforme. restoreReportle message de journal indique la durée de la `Restore` phase et le nombre de millisecondes qui vous ont été facturées pendant cette phase (type d'`snap-start`initialisation uniquement). Exemple :

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreReport",
  "record":{
      "metrics":{
          "durationMs":70.87,
          "billedDurationMs":13
      }
  }
}
```

Messages **`runtimeDone`** de plateforme

Si vous définissez la version du schéma sur « 2021-03-18 » dans la demande d'abonnement, Lambda envoie un message `platform.runtimeDone` une fois l'appel de fonction terminé avec succès ou avec une erreur. L'extension peut utiliser ce message pour arrêter toute la collecte de données télémétriques pour cet appel de fonction.

La API spécification Open pour le type d'événement Log dans la version du schéma 2021-03-18 est disponible ici : schema-2021-03-18.zip

Lambda génère le message de `platform.runtimeDone` journal lorsque le moteur d'exécution envoie une demande d'exécution `Next` ou `Error` d'exécutionAPI. Le `platform.runtimeDone` journal informe les consommateurs des journaux API que l'invocation de la fonction est terminée. Les extensions peuvent utiliser ces informations pour décider quand envoyer toutes les données télémétriques collectées au cours de cet appel.

Exemples

Lambda envoie le `platform.runtimeDone` message une fois que le moteur d'exécution a envoyé la NEXT demande lorsque l'invocation de la fonction est terminée. Les exemples suivants présentent des messages pour chacune des valeurs de statut : succès, échec et expiration du délai.

Example Exemple de message de réussite

```
{
    "time": "2021-02-04T20:00:05.123Z",
```

```
    "type": "platform.runtimeDone",
    "record": {
       "requestId":"6f7f0961f83442118a7af6fe80b88",
       "status": "success"
    }
}
```

Example Exemple de message d'échec

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "failure"
   }
}
```

Example Exemple de message d'expiration du délai

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "timeout"
  }
}
```

Example Exemple de plateforme. restoreRuntimeDone message (type **`snap-start`** d'initialisation uniquement)

La plateforme. restoreRuntimeDoneun message de journal indique si la `Restore` phase a réussi ou non. Lambda envoie ce message lorsque le moteur d'exécution envoie une demande `restore/next` d'exécutionAPI. Il existe trois statuts possibles : succès, échec et dépassement de délai. L'exemple suivant montre une plateforme performante. restoreRuntimeDonemessage de journal.

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreRuntimeDone",
  "record":{
```

```
      "status":"success"
  }
}
```

# Résolution des problèmes dans Lambda

Les rubriques suivantes fournissent des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation de LambdaAPI, de la console ou des outils. Si vous rencontrez un problème qui n’est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Pour plus de conseils de dépannage et de réponses aux questions courantes de support, visitez le Centre de connaissances AWS.

Pour plus d’informations sur le débogage et le dépannage des applications Lambda, consultez Débogage dans Serverless Land.

Rubriques


- Résoudre les problèmes de déploiement dans Lambda
- Résoudre les problèmes d'invocation dans Lambda
- Résoudre les problèmes d'exécution dans Lambda
- Résolution des problèmes de réseaux dans Lambda


## Résoudre les problèmes de déploiement dans Lambda

Lorsque vous mettez à jour votre fonction, Lambda déploie la modification en lançant de nouvelles instances de cette fonction avec le code ou les paramètres mis à jour. Les erreurs de déploiement empêchent l’utilisation de la nouvelle version et peuvent résulter de problèmes liés au package de déploiement, au code, à vos autorisations ou à vos outils.

Lorsque vous déployez des mises à jour de votre fonction directement avec le Lambda API ou avec un client tel que le AWS CLI, vous pouvez voir les erreurs provenant de Lambda directement dans la sortie. Si vous utilisez des services tels que AWS CloudFormation AWS CodeDeploy, ou AWS CodePipeline, recherchez la réponse de Lambda dans les journaux ou le flux d'événements de ce service.

Les rubriques suivantes fournissent des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation de LambdaAPI, de la console ou des outils. Si vous rencontrez un problème qui n’est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Pour plus de conseils de dépannage et de réponses aux questions courantes de support, visitez le Centre de connaissances AWS.

Pour plus d'informations sur le débogage et le dépannage des applications Lambda, consultez Débogage dans Serverless Land.

Rubriques


- Général : autorisation refusée/impossible de charger ce fichier
- Général : Une erreur se produit lors de l'appel du UpdateFunctionCode
- Amazon S3 : code d'erreur PermanentRedirect.
- Général : impossible de trouver, impossible de charger, impossible d'importer, classe introuvable, aucun fichier ou répertoire de ce type
- Général : gestionnaire de méthode non défini
- Lambda : échec de la conversion de couche
- Lambda : ou InvalidParameterValueException RequestEntityTooLargeException
- Lambda : InvalidParameterValueException
- Lambda : quotas de simultanéité et de mémoire


## Général : autorisation refusée/impossible de charger ce fichier

Erreur EACCES : autorisation refusée, ouvrez « /var/task/index.js »

Erreur : impossible de charger ce fichier - cette fonction

Erreur : [Errno 13] Autorisation refusée : '/var/task/function.py'

Le runtime Lambda a besoin d'une autorisation pour lire les fichiers de votre package de déploiement. Dans la notation octale des autorisations Linux, Lambda a besoin de 644 autorisations pour les fichiers non exécutables (rw-r--r--) et de 755 autorisations () pour les répertoires et les fichiers exécutables. rwxr-xr-x

Sous Linux et macOS, utilisez la commande `chmod` pour modifier les autorisations de fichiers sur les fichiers et les répertoires de votre package de déploiement. Par exemple, pour accorder les autorisations appropriées à un fichier exécutable, exécutez la commande suivante.

```
chmod 755 <filepath>
```

Pour modifier les autorisations relatives aux fichiers dans Windows, voir Définir, afficher, modifier ou supprimer des autorisations sur un objet dans la documentation Microsoft Windows.

## Général : Une erreur se produit lors de l'appel du UpdateFunctionCode

Erreur : une erreur s'est produite (RequestEntityTooLargeException) lors de l'appel de l' UpdateFunctionCodeopération

Lorsque vous téléchargez un package de déploiement ou une archive de couches directement vers Lambda, la taille du ZIP fichier est limitée à 50 Mo. Pour charger un fichier plus volumineux, stockez-le dans Amazon S3, puis utilisez les paramètres S3Bucket et S3Key.

> **Note**
>
> Lorsque vous téléchargez un fichier directement avec AWS CLI, ou autrement AWS SDK, le ZIP fichier binaire est converti en base64, ce qui augmente sa taille d'environ 30 %. Pour permettre cela et aussi prendre en compte la taille d'autres paramètres dans la demande, la limite de taille réelle de demande que Lambda applique est supérieure. Pour cette raison, la limite de 50 Mo est approximative.

## Amazon S3 : code d'erreur PermanentRedirect.

Erreur : une erreur s'est produite pendant GetObject. Code d'erreur S3 : PermanentRedirect. Message d'erreur S3 : Le compartiment se trouve dans cette région : us-east-2. Veuillez utiliser cette région pour renouveler la demande

Lorsque vous chargez le package de déploiement d'une fonction à partir d'un compartiment Amazon S3, le compartiment doit se trouver dans la même région que la fonction. Ce problème peut se produire lorsque vous spécifiez un objet Amazon S3 dans un appel à UpdateFunctionCode, ou lorsque vous utilisez les commandes de package et de déploiement dans le AWS CLI ou AWS SAM CLI. Créez un compartiment d'artefact de déploiement pour chaque région dans laquelle vous développez des applications.

## Général : impossible de trouver, impossible de charger, impossible d'importer, classe introuvable, aucun fichier ou répertoire de ce type

Erreur : module 'function' introuvable

Erreur : impossible de charger ce fichier - cette fonction

Erreur : impossible d'importer le module 'function'

Erreur : Classe introuvable : Function.handler

Erreur : fork/exec /var/task/function : aucun fichier ou répertoire de ce type

Erreur : impossible de charger le type 'Function.Handler' à partir de l'assemblage 'Function'.

Le nom du fichier ou de la classe dans la configuration du gestionnaire de votre fonction ne correspond pas à votre code. Consultez la section suivante pour plus d'informations.

## Général : gestionnaire de méthode non défini

Erreur : index.handler n'est pas défini ou n'est pas exporté

Erreur : gestionnaire 'handler' manquant dans le module 'function'

Erreur : « gestionnaire » de méthode non défini pour #<:0x000055b76ccebf98> LambdaHandler

Erreur : aucune méthode publique nommée handleRequest avec la signature de méthode appropriée n'a été trouvée dans la classe Function.Handler

Erreur : Impossible de trouver la méthode « handleRequest » dans le type « Function.Handler » de l'assemblage « Function »

Le nom de la méthode du gestionnaire dans la configuration du gestionnaire de votre fonction ne correspond pas à votre code. Chaque environnement d'exécution définit une convention de dénomination pour les gestionnaires, telle que *`filename.methodname`*. Le gestionnaire est la méthode du code de votre fonction que le moteur d'exécution exécute lorsque votre fonction est invoquée.

Pour certains langages, Lambda fournit une bibliothèque avec une interface pour laquelle une méthode de gestionnaire doit avoir un nom spécifique. Pour plus d'informations sur l'attribution de noms de gestionnaire pour chaque langue, consultez les rubriques suivantes.

- Création de fonctions Lambda avec Node.js
- Création de fonctions Lambda avec Python
- Création de fonctions Lambda avec Ruby

- Création de fonctions Lambda avec Java
- Création de fonctions Lambda avec Go
- Création de fonctions Lambda avec C#
- Création de fonctions Lambda avec PowerShell

## Lambda : échec de la conversion de couche

Erreur : la conversion de la couche Lambda a échoué. Pour obtenir des conseils sur la résolution de ce problème, consultez la page Résolution des problèmes de déploiement dans Lambda du Guide de l'utilisateur Lambda.

Lorsque vous configurez une fonction Lambda avec une couche, Lambda fusionne la couche avec le code de votre fonction. Si ce processus échoue, Lambda renvoie cette erreur. Si vous rencontrez cette erreur, réalisez les actions suivantes :

- Supprimer tous les fichiers inutiles de votre couche
- Supprimer tous les liens symboliques de votre couche
- Renommer tous les fichiers qui ont le même nom qu'un répertoire dans l'une des couches de votre fonction

## Lambda : ou InvalidParameterValueException RequestEntityTooLargeException

Erreur InvalidParameterValueException : Lambda n'a pas pu configurer vos variables d'environnement car les variables d'environnement que vous avez fournies ont dépassé la limite de 4 Ko. Corde mesurée : {"A1" : » uSFe Y5 cyPiPn 7atNx5bSM...

Erreur RequestEntityTooLargeException : la demande doit être inférieure à 5120 octets pour l'opération UpdateFunctionConfiguration

La taille maximale de l'objet variable stocké dans la configuration de la fonction ne doit pas dépasser 4096 octets. Cela inclut les noms de clés, les valeurs, les guillemets, les virgules et les crochets. La taille totale du corps de la HTTP demande est également limitée.

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
```

```
    "Runtime": "nodejs20.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "amzn-s3-demo-bucket",
            "KEY": "file.txt"
        }
    },
    ...
}
```

Dans cet exemple, l'objet comporte 39 caractères et utilise 39 octets lorsqu'il est stocké (sans espace) comme chaîne `{"BUCKET":"amzn-s3-demo-bucket","KEY":"file.txt"}`. ASCIILes caractères standard des valeurs des variables d'environnement utilisent un octet chacun. Les caractères étendus ASCII et Unicode peuvent utiliser entre 2 et 4 octets par caractère.

## Lambda : InvalidParameterValueException

Erreur InvalidParameterValueException : Lambda n'a pas pu configurer vos variables d'environnement car les variables d'environnement que vous avez fournies contiennent des clés réservées dont la modification n'est actuellement pas prise en charge.

Lambda réserve certaines clés de variables d'environnement pour une utilisation interne. Par exemple, `AWS_REGION` est utilisé par l'environnement d'exécution pour déterminer la région actuelle et ne peut pas être remplacé. D'autres variables, comme `PATH`, sont utilisées par l'environnement d'exécution, mais peuvent être étendues dans la configuration de votre fonction. Pour obtenir une liste complète, veuillez consulter Variables d'environnement d'exécution définies.

## Lambda : quotas de simultanéité et de mémoire

Erreur : la fonction spécifiée ConcurrentExecutions réduit le compte en UnreservedConcurrentExecution dessous de sa valeur minimale

Erreur : la valeur MemorySize « » n'a pas satisfait la contrainte : le membre doit avoir une valeur inférieure ou égale à 3008

Ces erreurs se produisent lorsque vous dépassez les quotas de simultanéité ou de mémoire pour votre compte. AWS Les nouveaux comptes ont réduit la simultanéité et les quotas de mémoire. Pour résoudre les erreurs liées à la simultanéité, vous pouvez demander une augmentation des quotas. Vous ne pouvez pas demander d'augmentation des quotas de mémoire.

- Concurrence : un message d'erreur peut s'afficher si vous essayez de créer une fonction à l'aide de la simultanéité réservée ou provisionnée, ou si votre demande de simultanéité par fonction (PutFunctionConcurrency) dépasse le quota de simultanéité de votre compte.
- Mémoire : des erreurs surviennent lorsque la capacité de mémoire allouée à la fonction dépasse le quota de mémoire de votre compte.

# Résoudre les problèmes d'invocation dans Lambda

Lorsque vous invoquez une fonction Lambda, Lambda valide la demande et vérifie la capacité de dimensionnement avant d'envoyer l'événement à votre fonction ou, pour une invocation asynchrone, à la file d'attente d'événements. Les erreurs d'invocation peuvent être causées par des problèmes de paramètres de demande, de structure d'événement, de paramètres de fonction, d'autorisations utilisateur, d'autorisations de ressources ou de limites.

Si vous invoquez la fonction directement, les erreurs d'invocation sont visibles dans la réponse de Lambda. Si vous invoquez la fonction de manière asynchrone avec un mappage de source d'événement ou via un autre service, vous pouvez trouver des erreurs dans les journaux, une file d'attente de lettres mortes ou une destination en cas d'échec. Les options de gestion des erreurs et le comportement des nouvelles tentatives varient en fonction de la façon dont vous invoquez la fonction et du type d'erreur.

Pour obtenir la liste des types d'erreur que l'opération `Invoke` peut renvoyer, consultez Invoquer.

## Lambda : expiration de la fonction pendant la phase d'initialisation (Sandbox.Timedout)

Erreur : le délai de la tâche a expiré au bout de 3,00 secondes

Lorsque la phase d'initialisation expire, Lambda initialise à nouveau l'environnement d'exécution en relançant la phase lorsque `Init` la prochaine demande d'appel arrive. Cela s’appelle une init supprimée. Toutefois, si votre fonction est configurée avec un délai d'expiration court (généralement environ 3 secondes), l'initialisation supprimée risque de ne pas se terminer pendant le délai imparti, ce qui provoquera un nouveau délai d'expiration de la `Init` phase. Sinon, l'initialisation supprimée se termine mais ne laisse pas suffisamment de temps à la phase Invoke pour se terminer, ce qui entraîne son expiration. `Invoke`

Pour réduire les erreurs liées au délai d'expiration, appliquez une ou plusieurs des stratégies suivantes :

- Augmenter la durée du délai d'expiration de la fonction : Prolongez le délai d'expiration pour donner aux `Invoke` phases `Init` et le temps de se terminer avec succès.
- Augmenter l'allocation de mémoire des fonctions : Plus de mémoire signifie également une CPU allocation plus proportionnelle, ce qui peut accélérer à la fois les `Invoke` phases `Init` et.
- Optimisez le code d'initialisation de la fonction : réduisez le temps nécessaire à l'initialisation afin de garantir que la `Invoke` phase `Init` et puisse se terminer dans le délai défini.
- Ajouter une gestion des erreurs : une gestion appropriée des erreurs dans le code de fonction peut empêcher l'échec de l'environnement d'exécution Lambda et le déclenchement de tentatives d'initialisation répétées.

## IAM: lambda : InvokeFunction non autorisé

Erreur : L'utilisateur : arn:aws:iam : :123456789012 : l'utilisateur/développeur n'est pas autorisé à exécuter : lambda : on resource : my-function InvokeFunction

Votre utilisateur, ou le rôle que vous assumez, doit avoir une autorisation pour invoquer une fonction. Cette exigence s'applique également aux fonctions Lambda et à d'autres ressources de calcul qui invoquent des fonctions. Ajoutez la politique AWS gérée AWSLambdaRoleà votre utilisateur ou ajoutez une politique personnalisée qui autorise l'`lambda:InvokeFunction`action sur la fonction cible.

> **Note**
>
> Le nom de l'IAMaction (`lambda:InvokeFunction`) fait référence à l'opération `Invoke` LambdaAPI.

Pour plus d'informations, consultez Gestion des autorisations dans AWS Lambda.

## Lambda : Impossible de trouver un bootstrap valide (Runtime). InvalidEntrypoint)

Erreur : impossible de trouver un ou plusieurs bootstrap valides : [/var/task/bootstrap /opt/bootstrap]

Cette erreur se produit généralement lorsque la racine de votre package de déploiement ne contient pas de fichier exécutable nommé `bootstrap`. Par exemple, si vous déployez une fonction

`provided.al2023` avec un fichier .zip, le fichier `bootstrap` doit se trouver à la racine du fichier .zip, et non dans un répertoire.

## Lambda : L'opération ne peut pas être effectuée ResourceConflictException

Erreur ResourceConflictException : Impossible d'effectuer l'opération pour le moment. La fonction est actuellement dans l'état suivant : En attente

Lorsque vous connectez une fonction à un cloud privé virtuel (VPC) au moment de sa création, la fonction entre dans un `Pending` état tandis que Lambda crée des interfaces réseau élastiques. Pendant ce temps, vous ne pouvez pas invoquer ni modifier votre fonction. Si vous connectez votre fonction à une fonction VPC après sa création, vous pouvez l'invoquer pendant que la mise à jour est en attente, mais vous ne pouvez pas modifier son code ou sa configuration.

Pour plus d'informations, consultez États de la fonction Lambda.

## Lambda : La fonction est bloquée à l'état Pending (En attente)

Erreur : Une fonction est bloquée à l'état *`Pending`* depuis plusieurs minutes.

Si une fonction reste bloquée pendant `Pending` plus de six minutes, effectuez l'une des API opérations suivantes pour la débloquer :

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

Lambda annule l'opération en attente et met la fonction à l'état `Failed`. Vous pouvez ensuite essayer une autre mise à jour.

## Lambda : Une fonction utilise toute la simultanéité

Problème : une fonction utilise toute la simultanéité disponible, limitant ainsi d'autres fonctions.

Pour diviser la simultanéité disponible de votre AWS compte dans une AWS région en groupes, utilisez la simultanéité réservée. La simultanéité réservée garantit qu'une fonction s'adapte toujours à la simultanéité qui lui est assignée et qu'elle n'utilise pas plus de simultanéité que ce qui lui est attribué.

## Général : Impossible d'invoquer la fonction avec d'autres comptes ou services

Problème : vous pouvez invoquer la fonction directement, mais elle ne s'exécute pas lorsqu'elle est invoquée par un autre service ou compte.

Vous accordez à d'autres services et comptes l'autorisation d'invoquer une fonction dans la stratégie basée sur les ressourcesde la fonction. Si l'appelant se trouve dans un autre compte, cet utilisateur doit également avoir l'autorisation d'invoquer des fonctions.

## Général : L'invocation de la fonction boucle

Problème : La fonction est invoquée en continu dans une boucle.

Cela se produit généralement lorsque votre fonction gère les ressources du même AWS service qui la déclenche. Par exemple, il est possible de créer une fonction qui stocke un objet dans un compartiment Amazon Simple Storage Service (Amazon S3) configuré avec une notification qui invoque à nouveau la fonction. Pour arrêter l'exécution de la fonction, réduisez la simultanéité disponible à zéro, ce qui limite toutes les futures invocations. Identifiez ensuite le chemin de code ou l'erreur de configuration qui a provoqué l'invocation récursive. Lambda détecte et arrête automatiquement les boucles récursives pour certains AWS services et. SDKs Pour de plus amples informations, veuillez consulter the section called “Détection de boucle récursive”.

## Lambda : Routage d'alias avec une simultanéité approvisionnée

Problème: invocations de débordement de simultanéité approvisionnée pendant le routage d'alias.

Lambda utilise un modèle probabiliste simple pour distribuer le trafic entre les deux versions de la fonction. Quand le niveau de trafic est faible, il se peut que vous observiez une variance élevée entre les pourcentages de trafic configuré et réel sur chaque version. Si votre fonction utilise une simultanéité approvisionnée, vous pouvez éviter des invocations de débordement en configurant un plus grand nombre d'instances de simultanéité approvisionnées pendant que le routage d'alias est actif.

## Lambda : Démarrages à froid avec la simultanéité allouée

Problème :  Vous remarquez des démarrages à froid après avoir activé la simultanéité allouée.

Lorsque le nombre d'exécutions simultanées sur une fonction est inférieur ou égal au niveau de simultanéité configuré, il ne devrait pas y avoir de démarrage à froid. Pour vous aider à vérifier le bon fonctionnement de la simultanéité, procédez comme suit :

- Vérifiez que la simultanéité allouée est activée sur la version de fonction ou l'alias.

  > **Note**
  >
  > La simultanéité provisionnée n'est pas configurable sur la version non publiée de la fonction ($). LATEST

- Assurez-vous que vos déclencheurs invoquent la version ou l'alias de fonction correct. Par exemple, si vous utilisez Amazon API Gateway, vérifiez que API Gateway invoque la version ou l'alias de la fonction avec une simultanéité provisionnée, et non $. LATEST Pour confirmer que la simultanéité provisionnée est utilisée, vous pouvez consulter la métrique ProvisionedConcurrencyInvocations Amazon CloudWatch . Une valeur non nulle indique que la fonction traite des invocations sur les environnements d'exécution initialisés.
- Déterminez si la simultanéité de vos fonctions dépasse le niveau configuré de simultanéité provisionnée en vérifiant la métrique. ProvisionedConcurrencySpilloverInvocations CloudWatch Une valeur différente de zéro indique que toute la simultanéité allouée est en cours d'utilisation et qu'une invocation s'est produite avec un démarrage à froid.
- Vérifiez la fréquence des invocations (demandes par seconde). Les fonctions avec simultanéité allouée ont un taux maximal de 10 demandes par seconde par simultanéité allouée. Par exemple, une fonction configurée avec une simultané allouée de 100 peut traiter 1 000 demandes par seconde. Si le taux d'invocations dépasse 1 000 requêtes par seconde, certains démarrages à froid peuvent se produire.

## Lambda : Démarrages à froid avec de nouvelles versions

Problème : Vous notez des démarrages à froid lors du déploiement de nouvelles versions de votre fonction.

Lorsque vous mettez à jour un alias de fonction, Lambda déplace automatiquement la simultanéité approvisionnée vers la nouvelle version en fonction des pondérations configurées sur l'alias.

Erreur KMSDisabledException : Lambda n'a pas pu déchiffrer les variables d'environnement car la KMS clé utilisée est désactivée. Vérifiez les KMS principaux paramètres de la fonction.

Cette erreur peut se produire si votre clé AWS Key Management Service (AWS KMS) est désactivée ou si l'autorisation permettant à Lambda d'utiliser la clé est révoquée. Si l'octroi est manquant, configurez la fonction pour utiliser une autre clé. Ensuite, réaffectez la clé personnalisée pour recréer l'octroi.

## EFS: la fonction n'a pas pu monter le système de EFS fichiers

Erreur EFSMountFailureException : La fonction n'a pas pu monter le système de EFS fichiers avec le point d'accès arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd.

La demande de montage sur le système de fichiers de la fonction a été rejetée. Vérifiez les autorisations de la fonction et confirmez que son système de fichiers et son point d'accès existent et sont prêts à l'emploi.

## EFS: la fonction n'a pas pu se connecter au système de EFS fichiers

Erreur EFSMountConnectivityException : La fonction n'a pas pu se connecter au système de EFS fichiers Amazon avec le point d'accès arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd. Vérifiez la configuration de votre réseau et réessayez.

La fonction n'a pas pu établir de connexion avec le système de fichiers de la fonction avec le NFS protocole (TCPport 2049). Vérifiez le groupe de sécurité et la configuration du routage pour VPC les sous-réseaux.

Si ces erreurs se produisent après avoir mis à jour les paramètres de VPC configuration de votre fonction, essayez de démonter puis de remonter le système de fichiers.

## EFS: La fonction n'a pas pu monter le système de EFS fichiers en raison d'un délai d'attente

Erreur EFSMountTimeoutException : La fonction n'a pas pu monter le système de EFS fichiers avec le point d'accès {arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd} en raison d'un délai de montage dépassé.

La fonction a pu se connecter au système de fichiers de la fonction, mais l'opération de montage a expiré. Réessayez après un court laps de temps et envisagez de limiter la concurrence de la fonction pour réduire la charge sur le système de fichiers.

## Lambda : Lambda a détecté un processus d'E/S qui prenait trop de temps

EFSIOException: Cette instance de fonction a été arrêtée car Lambda a détecté un processus d'E/S trop long.

Une invocation précédente a expiré et Lambda n'a pas pu mettre fin à l'exécution du gestionnaire de fonction. Ce problème peut se produire lorsqu'un système de fichiers joint manque de crédits de rafale et que le débit de base est insuffisant. Pour augmenter le débit, vous pouvez augmenter la taille du système de fichiers ou utiliser le débit provisionné.

# Résoudre les problèmes d'exécution dans Lambda

Lorsque le runtime Lambda exécute le code de fonction, l'événement peut être traité sur une instance de la fonction qui traite déjà les événements depuis un certain temps, ou nécessiter l'initialisation d'une nouvelle instance. Des erreurs peuvent se produire lors de l'initialisation de la fonction, lorsque le code de gestionnaire traite l'événement ou lorsque la fonction renvoie (ou ne parvient pas à renvoyer) une réponse.

Les erreurs d'exécution de fonction peuvent être causées par des problèmes de code, de configuration de fonction, de ressources en aval ou d'autorisations. Si vous appelez la fonction directement, les erreurs de fonction sont visibles dans la réponse de Lambda. Si vous appelez la fonction de manière asynchrone, avec un mappage de source d'événement ou via un autre service, vous pouvez trouver les erreurs dans les journaux, une file d'attente de lettres mortes ou une destination réservées aux échecs. Les options de gestion des erreurs et le comportement des nouvelles tentatives varient en fonction de la façon dont vous appelez la fonction et du type d'erreur.

Lorsque le code de fonction ou le runtime Lambda renvoient une erreur, le code d'état indiqué dans la réponse de Lambda est 200 OK. La présence d'une erreur dans la réponse est indiquée par un en-tête nommé `X-Amz-Function-Error`. Les codes d'état des séries 400 et 500 sont réservés aux erreurs d'invocation.

## Lambda : l'exécution prend trop de temps

Problème : l'exécution de la fonction prend trop de temps.

Si l'exécution de votre code est beaucoup plus longue dans Lambda que sur votre ordinateur local, cela peut être dû à une limite au niveau de la mémoire ou de la puissance de traitement disponible pour la fonction. Configurez la fonction avec de la mémoire supplémentaire pour augmenter à la fois la mémoire etCPU.

## Lambda : les journaux ou les traces n'apparaissent pas

Problème : les journaux n'apparaissent pas dans CloudWatch les journaux.

Problème : les traces n'apparaissent pas dans AWS X-Ray.

Votre fonction doit être autorisée à appeler CloudWatch Logs and X-Ray. Mettez à jour son rôle d'exécution pour lui accorder l'autorisation. Ajoutez les stratégies gérées suivantes pour activer les journaux et le suivi.

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

Lorsque vous ajoutez des autorisations à votre fonction, effectuez également une mise à jour triviale de son code ou de sa configuration. Cela force l'arrêt et le remplacement des instances en cours d'exécution de votre fonction, qui ont des informations d'identification obsolètes.

> **Note**
>
> L’affichage des journaux après l’invocation d’une fonction peut prendre de 5 à 10 minutes .

## Lambda : certains journaux de ma fonction n'apparaissent pas

Problème : les journaux des fonctions sont absents dans CloudWatch les journaux, même si mes autorisations sont correctes

Si vous atteignez Compte AWS les limites de quota de CloudWatch journalisation, CloudWatch les régulateurs fonctionnent à la journalisation. Dans ce cas, certains journaux générés par vos fonctions peuvent ne pas apparaître dans CloudWatch les journaux.

Si votre fonction produit des journaux à un taux trop élevé pour que Lambda puisse les traiter, cela peut également empêcher les sorties des journaux d'apparaître dans CloudWatch les journaux. Lorsque Lambda ne parvient pas à envoyer des journaux CloudWatch au rythme auquel votre fonction les produit, il supprime les journaux pour empêcher l'exécution de votre fonction de ralentir. Attendez-vous à observer régulièrement la perte de journaux lorsque le débit de vos journaux dépasse 2 Mo/s pour un seul flux de journaux.

Si votre fonction est configurée pour utiliser des journaux JSON formatés, Lambda essaie d'envoyer `logsDropped`un événement CloudWatch à Logs lorsqu'elle supprime des journaux. Toutefois, lorsque vous CloudWatch limitez la journalisation de votre fonction, cet événement risque de ne pas atteindre les CloudWatch journaux. Vous ne verrez donc pas toujours d'enregistrement lorsque Lambda supprime les journaux.

Pour vérifier si votre quota de CloudWatch logs Compte AWS est atteint, procédez comme suit :

1. Ouvrez la console Service Quotas.
2. Dans le panneau de navigation, choisissez Services AWS .
3. Dans la liste des AWS services, recherchez Amazon CloudWatch Logs.
4. Dans la liste des quotas de service, choisissez les quotas `CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions per second` et `PutLogEvents throttle limit in transactions per second` pour afficher votre utilisation.

Vous pouvez également configurer des CloudWatch alarmes pour vous avertir lorsque l'utilisation de votre compte dépasse la limite que vous avez spécifiée pour ces quotas. Voir Créer une CloudWatch alarme basée sur un seuil statique pour en savoir plus.

Si les limites de quota par défaut pour CloudWatch les journaux ne sont pas suffisantes pour votre cas d'utilisation, vous pouvez demander une augmentation du quota.

## Lambda : la fonction renvoie avant la fin de l'exécution

Problème : (Node.js) la fonction renvoie un objet avant la fin de l'exécution du code

De nombreuses bibliothèques, y compris la AWS SDK, fonctionnent de manière asynchrone. Lorsque vous effectuez un appel réseau ou toute autre opération nécessitant l'attente d'une réponse, les bibliothèques renvoient un objet appelé promesse, qui suit la progression de l'opération en arrière-plan.

Pour attendre que la promesse soit résolue en réponse, utilisez le mot-clé `await`. Celui-ci empêche le code de gestionnaire de s'exécuter jusqu'à ce que la promesse soit résolue en objet contenant la réponse. Si vous n'avez pas besoin d'utiliser les données de la réponse dans votre code, vous pouvez retourner la promesse directement à l'environnement d'exécution.

Certaines bibliothèques ne retournent pas de promesses, mais peuvent être enveloppées dans du code qui le fait. Pour de plus amples informations, veuillez consulter Définir le gestionnaire de fonctions Lambda dans Node.js.

## AWS SDK: Versions et mises à jour

Problème : la version AWS SDK incluse dans le runtime n'est pas la dernière version

Problème : le contenu AWS  SDKinclus dans le moteur d'exécution est automatiquement mis à jour

Les environnements d'exécution des langages de script incluent AWS SDK et sont régulièrement mis à jour vers la dernière version. La version actuelle de chaque moteur d'exécution est répertoriée sur la page d'exécution. Pour utiliser une version plus récente du AWS SDK ou pour verrouiller vos fonctions sur une version spécifique, vous pouvez regrouper la bibliothèque avec votre code de fonction ou créer une couche Lambda. Pour plus d'informations sur la création d'un package de déploiement avec des dépendances, consultez les rubriques suivantes :

Node.js

- Déployer des fonctions Lambda en Node.js avec des archives de fichiers .zip

Python

- Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python

Ruby

- Déployer des fonctions Lambda en Ruby avec des archives de fichiers .zip

Java

- Déployez des fonctions Java Lambda avec des archives .zip ou de fichiers JAR

Go

- Déployer des fonctions Lambda Go avec des archives de fichiers .zip

C#

- Créez et déployez des fonctions Lambda C# à l’aide des archives de fichiers .zip

PowerShell

- Déployer des fonctions PowerShell Lambda avec des archives de fichiers .zip

## Python : les bibliothèques se chargent de manière incorrecte

Problème : (Python) certaines bibliothèques ne se chargent pas correctement à partir du package de déploiement

Les bibliothèques avec des modules d'extension écrits en C ou C++ doivent être compilées dans un environnement avec la même architecture de processeur que Lambda (Amazon Linux). Pour de plus amples informations, veuillez consulter Travailler avec des archives de fichiers .zip pour les fonctions Lambda Python.

## Java : votre fonction met plus de temps à traiter les événements après la mise à jour de Java 11 vers Java 17

Problème : (Java) Votre fonction met plus de temps à traiter les événements après la mise à jour de Java 11 vers Java 17

Réglez votre compilateur à l'aide du `JAVA_TOOL_OPTIONS` paramètre. Les environnements d'exécution Lambda pour Java 17 et versions ultérieures de Java modifient les options du compilateur par défaut. Cette modification améliore les temps de démarrage à froid pour les fonctions de courte durée, mais le comportement précédent est mieux adapté aux fonctions de longue durée nécessitant des calculs intensifs. Définissez `JAVA_TOOL_OPTIONS` cette valeur `-XX:-TieredCompilation` sur pour revenir au comportement de Java 11. Pour plus d'informations sur le paramètre `JAVA_TOOL_OPTIONS`, consultez the section called “Comprendre la variable d'`JAVA_TOOL_OPTIONS`environnement”.

# Résolution des problèmes de réseaux dans Lambda

Par défaut, Lambda exécute vos fonctions dans un cloud privé virtuel (Virtual Private Cloud, VPC) interne disposant d'connectivité aux services AWS et à Internet. Pour accéder aux ressources réseau en local, vous pouvez configurer votre fonction afin qu'elle se connecte à un VPC de votre compte. Lorsque vous utilisez cette fonctionnalité, vous gérez l'accès Internet et la connectivité réseau de la fonction avec les ressources Amazon Virtual Private Cloud (Amazon VPC).

Les erreurs de connectivité réseau peuvent être dues à des problèmes liés à la configuration du routage de votre VPC, aux règles du groupe de sécurité, aux autorisations de rôle AWS Identity and Access Management (IAM) ou à la traduction d'adresses réseau (NAT), ou à la disponibilité de ressources telles que les adresses IP ou les interfaces réseau. En fonction du problème, vous pouvez voir une erreur spécifique ou un délai d'attente si une requête ne peut pas atteindre sa destination.

## VPC : la fonction perd l'accès à Internet ou expire

Problème : votre fonction Lambda perd l'accès Internet après la connexion à un VPC.

Erreur : Error: connect ETIMEDOUT 176.32.98.189:443

Erreur : Error: Task timed out after 10.00 seconds

Erreur ReadTimeoutError : Le délai de lecture a expiré. (délai de lecture = 15)

Lorsque vous connectez une fonction à un VPC, toutes les demandes sortantes passent par le VPC. Pour vous connecter à Internet, configurez votre VPC pour envoyer le trafic sortant depuis le sous-réseau de la fonction vers une passerelle NAT dans un sous-réseau public. Pour obtenir plus d'informations et des exemples de configurations VPC, consultez the section called “Accès à Internet pour les VPC fonctions”.

Si certaines de vos connexions TCP sont interrompues, cela peut être dû à la fragmentation des paquets. Les fonctions Lambda ne peuvent pas traiter les requêtes TCP fragmentées entrantes, car Lambda ne prend pas en charge la fragmentation IP pour TCP ou ICMP.

## VPC : la fonction doit accéder aux AWS services sans utiliser Internet

Problème : Votre fonction Lambda doit accéder aux AWS services sans utiliser Internet.

Pour connecter une fonction à AWS des services depuis un sous-réseau privé sans accès à Internet, utilisez des points de terminaison VPC.

## VPC : limite d'interface réseau Elastic atteinte

Erreur : ENI LimitReachedException : la limite de l'interface Elastic network a été atteinte pour le VPC de la fonction.

Lorsque vous connectez une fonction Lambda à un VPC, Lambda crée une interface réseau Elastic pour chaque combinaison de sous-réseau et de groupe de sécurité attaché à cette fonction. Le quota de service par défaut est de 250 interfaces réseau par VPC. Pour demander une augmentation d'un quota, vous pouvez utiliser la console Service Quotas.

## EC2 : interface réseau élastique de type « lambda »

Code d'erreur : client. OperationNotPermitted

Message d'erreur : le groupe de sécurité ne peut pas être modifié pour ce type d'interface

Vous recevrez cette erreur si vous tentez de modifier une Interface réseau Elastic (ENI) gérée par Lambda. Le `ModifyNetworkInterfaceAttribute` n'est pas inclus dans l'API Lambda pour les opérations de mise à jour sur les interfaces réseau élastiques créées par Lambda.

## DNS : échec de la connexion aux hôtes avec UNKNOWNHOSTEXCEPTION

Message d'erreur : UNKNOWNHOSTEXCEPTION

Les fonctions Lambda prennent en charge un maximum de 20 connexions TCP simultanées pour la résolution DNS. Votre fonction est peut-être en train d'épuiser cette limite. Les requêtes DNS les plus courantes sont effectuées via UDP. Si votre fonction établit uniquement des connexions DNS UDP, il est peu probable que ce soit votre problème. Cette erreur est souvent due à une mauvaise configuration ou à une infrastructure dégradée. Par conséquent, avant d'examiner votre trafic DNS en profondeur, vérifiez que votre infrastructure DNS est correctement configurée et saine et que votre fonction Lambda fait référence à un hôte spécifié dans le DNS.

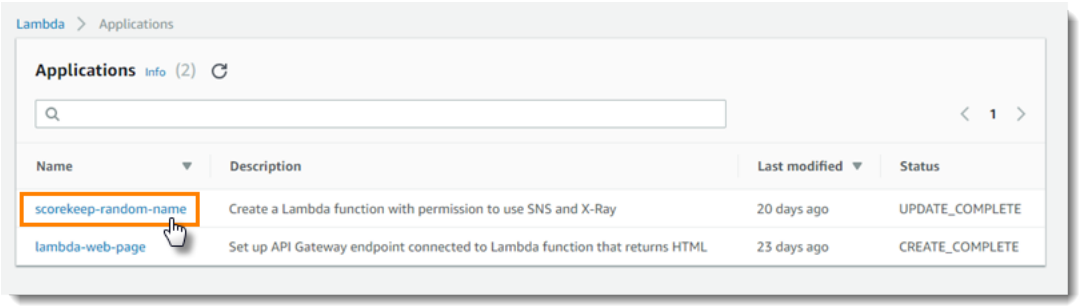
Si vous diagnostiquez que votre problème est lié à la connexion TCP maximale, notez que vous ne pouvez pas demander d'augmentation de cette limite. Si votre fonction Lambda revient au DNS TCP en raison de charges utiles DNS importantes, vérifiez que votre solution utilise des bibliothèques compatibles EDNS. Pour plus d'informations sur EDNS, consultez la norme RFC 6891. Si vos charges utiles DNS dépassent régulièrement les tailles maximales EDNS, votre solution risque d'épuiser la limite DNS TCP.

# Affichage des applications dans la AWS Lambda console

La AWS Lambda console vous permet de surveiller et de gérer vos applications Lambda. La section Applications répertorie les AWS CloudFormation piles dotées de fonctions Lambda. Le menu inclut des piles dans lesquelles vous pouvez lancer en AWS CloudFormation utilisant la AWS CloudFormation console, le AWS Serverless Application Repository AWS CLI, ou le AWS SAM CLI.

Pour afficher une application Lambda

1. Ouvrez la page Applications de la console Lambda.
2. Choisissez une application.




La présentation affiche les informations suivantes sur votre application.

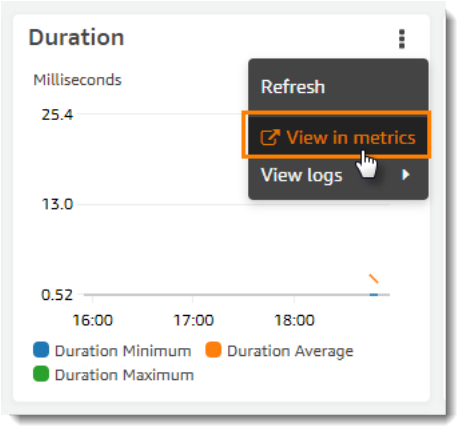
- AWS CloudFormation modèle ou SAMmodèle : modèle qui définit votre application.
- Ressources : AWS ressources définies dans le modèle de votre application. Pour gérer les fonctions Lambda de votre application, choisissez un nom de fonction dans la liste.

# Surveillance des applications Lambda

La section Applications de la console Lambda inclut un onglet Monitoring dans lequel vous pouvez consulter un tableau de CloudWatch bord Amazon contenant des métriques agrégées pour les ressources de votre application.

Pour surveiller une application Lambda

1. Ouvrez la page Applications de la console Lambda.
2. Choisissez Surveillance.
3. Pour obtenir plus de détails sur les mesures dans n'importe quel graphique, choisissez Afficher dans les mesures dans le menu déroulant.




   Le graphique s'affiche dans un nouvel onglet, avec les métriques correspondantes répertoriées sous le graphique. Vous pouvez personnaliser votre affichage de ce graphique, en modifiant les métriques et les ressources affichées, les statistiques, la période, et d'autres facteurs, afin de mieux comprendre la situation actuelle.

Par défaut, la console Lambda affiche un tableau de bord de base. Vous pouvez personnaliser cette page en ajoutant un ou plusieurs CloudWatch tableaux de bord Amazon à votre modèle d'application avec le type de ressource AWS: : CloudWatch : :Dashboard. Lorsque votre modèle inclut un ou plusieurs tableaux de bord, la page affiche vos tableaux de bord à la place du tableau de bord par défaut. Vous pouvez basculer entre les tableaux de bord avec le menu déroulant en haut à droite de la page. L'exemple suivant permet de créer un tableau de bord avec un seul widget qui représente graphiquement le nombre d'appels d'une fonction nommée `my-function`.

## Example modèle de tableau de bord de fonction

```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
            "widgets": [
                {
                    "type": "metric",
                    "width": 12,
                    "height": 6,
                    "properties": {
                        "metrics": [
                            [
                                "AWS/Lambda",
                                "Invocations",
                                "FunctionName",
                                "my-function",
                                {
                                    "stat": "Sum",
                                    "label": "MyFunction"
                                }
                            ],
                            [
                                {
                                    "expression": "SUM(METRICS())",
                                    "label": "Total Invocations"
                                }
                            ]
                        ],
                        "region": "us-east-1",
                        "title": "Invocations",
                        "view": "timeSeries",
                        "stacked": false
                    }
                }
            ]
        }
```

Pour plus d'informations sur la création de CloudWatch tableaux de bord et de widgets, consultez la section Structure et syntaxe du corps du tableau de bord dans le Amazon CloudWatch API Reference.

# Créez des déploiements progressifs pour les fonctions Lambda

Utilisez des déploiements propagés pour contrôler les risques associés à l'introduction de nouvelles versions de votre fonction Lambda. Dans un déploiement propagé, le système déploie automatiquement la nouvelle version de la fonction et envoie progressivement un volume croissant de trafic vers la nouvelle version. La quantité de trafic et le taux d'augmentation sont des paramètres que vous pouvez configurer.

Vous configurez un déploiement progressif à l'aide de AWS CodeDeploy et AWS SAM. CodeDeploy est un service qui automatise les déploiements d'applications sur les plateformes informatiques Amazon telles qu'Amazon EC2 et. AWS Lambda Pour plus d'informations, voir Qu'est-ce que c'est CodeDeploy ? . En utilisant CodeDeploy pour déployer votre fonction Lambda, vous pouvez facilement surveiller l'état du déploiement et lancer une annulation si vous détectez des problèmes.

AWS SAM est un framework open source permettant de créer des applications sans serveur. Vous créez un AWS SAM modèle (au YAML format) pour spécifier la configuration des composants requis pour le déploiement progressif. AWS SAM utilise le modèle pour créer et configurer les composants. Pour plus d'informations, consultez Qu'est-ce qu' AWS SAM ?

Dans le cadre d'un déploiement continu, AWS SAM exécute les tâches suivantes :

- Il configure votre fonction Lambda et crée un alias.

  La configuration de routage d'alias est la capacité sous-jacente qui implémente le déploiement propagé.
- Il crée une CodeDeploy application et un groupe de déploiement.

  Le groupe de déploiement gère le déploiement propagé et la restauration (si nécessaire).
- Il détecte le moment où vous créez une nouvelle version de votre fonction Lambda.
- Il déclenche CodeDeploy le lancement du déploiement de la nouvelle version.

## Exemple de AWS SAM modèle Lambda

L'exemple suivant présente un modèle AWS SAM pour un déploiement propagé simple.

```
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
```

```
Description: A sample SAM template for deploying Lambda functions.

Resources:
# Details about the myDateTimeFunction Lambda function
  myDateTimeFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: myDateTimeFunction.handler
      Runtime: nodejs18.x
# Creates an alias named "live" for the function, and automatically publishes when you
 update the function.
      AutoPublishAlias: live
      DeploymentPreference:
# Specifies the deployment configuration
          Type: Linear10PercentEvery2Minutes
```

Ce modèle définit une fonction Lambda nommée `myDateTimeFunction` avec les propriétés suivantes.

AutoPublishAlias

La propriété `AutoPublishAlias` crée un alias nommé `live`. De plus, l'infrastructure AWS SAM détecte automatiquement le moment où vous enregistrez du code nouveau pour la fonction. L'infrastructure publie ensuite une nouvelle version de fonction et met à jour l'alias `live` de sorte qu'il pointe vers la nouvelle version.

DeploymentPreference

La `DeploymentPreference` propriété détermine la vitesse à laquelle l' CodeDeploy application déplace le trafic de la version d'origine de la fonction Lambda vers la nouvelle version. La valeur `Linear10PercentEvery2Minutes` déplace 10 % du trafic toutes les deux minutes vers la nouvelle version.

Pour obtenir la liste des configurations de déploiement prédéfinies, consultez Configurations de déploiement.

Pour un didacticiel détaillé sur l'utilisation des fonctions Lambda, voir Déployer une fonction Lambda mise à jour CodeDeploy avec. CodeDeploy

# Utilisation de Lambda avec Kubernetes

Vous pouvez déployer et gérer des fonctions Lambda avec l'API Kubernetes à l'aide d'AWS Controllers for Kubernetes (ACK) ou Crossplane.

## AWS Controllers for Kubernetes (ACK)

Vous pouvez utiliser ACK pour déployer et gérer des ressources AWS de l'API Kubernetes. Grâce à ACK, AWS fournit des contrôleurs personnalisés open source pour AWS des services tels que Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) et Amazon. SageMaker Chaque service AWS pris en charge possède son propre contrôleur personnalisé. Dans votre cluster Kubernetes, installez un contrôleur pour chaque service AWS que vous souhaitez utiliser. Créez ensuite une définition de ressource personnalisée (Custom Resource Definition, CRD) pour définir les ressources AWS.

Nous vous recommandons d'utiliser Helm 3.8 ou version ultérieure pour installer les contrôleurs ACK. Chaque contrôleur ACK est doté de ses propres Charts de Helm, qui installent le contrôleur, les CRD et les règles RBAC de Kubernetes. Pour plus d'informations, consultez Installer un contrôleur ACK dans la documentation ACK.

Après avoir créé la ressource personnalisée ACK, vous pouvez l'utiliser comme n'importe quel autre objet Kubernetes intégré. Par exemple, vous pouvez déployer et gérer des fonctions Lambda avec vos chaînes d'outils Kubernetes préférées, notamment kubectl.

Voici quelques exemples de cas d'utilisation pour le provisionnement de fonctions Lambda via ACK :

- Votre organisation utilise le contrôle d'accès basé sur les rôles (RBAC) et des rôles IAM pour les comptes de service afin de créer les limites des autorisations. Avec ACK, vous pouvez réutiliser ce modèle de sécurité pour Lambda sans avoir à créer d'autres utilisateurs et stratégies.
- Votre organisation dispose d'un DevOps processus pour déployer des ressources dans un cluster Amazon Elastic Kubernetes Service (Amazon EKS) à l'aide de manifestes Kubernetes. Avec ACK, vous pouvez utiliser un manifeste pour provisionner des fonctions Lambda sans créer d'infrastructure distincte sous forme de modèles de code.

Pour plus d'informations sur l'utilisation d'ACK, consultez le didacticiel Lambda dans la documentation ACK.

# Crossplane

Crossplane est un projet open source de la Cloud Native Computing Foundation (CNCF) qui utilise Kubernetes pour gérer les ressources de l'infrastructure cloud. Avec Crossplane, les développeurs peuvent demander une infrastructure sans avoir à en comprendre la complexité. Les équipes chargées de la plateforme gardent le contrôle de la manière dont l'infrastructure est provisionnée et gérée.

À l'aide de Crossplane, vous pouvez déployer et gérer des fonctions Lambda avec vos chaînes d'outils Kubernetes préférées, telles que kubectl, et tout pipeline CI/CD capable de déployer des manifestes sur Kubernetes. Voici quelques exemples de cas d'utilisation pour le provisionnement de fonctions Lambda via Crossplane :

- Votre entreprise souhaite renforcer la conformité en s'assurant que les fonctions Lambda disposent des balises appropriées. Les équipes chargées de la plateforme peuvent utiliser Compositions Crossplane pour définir cette stratégie par le biais d'abstractions d'API. Les développeurs peuvent ensuite utiliser ces abstractions pour déployer des fonctions Lambda avec des balises.
- Votre projet utilise GitOps Kubernetes. Dans ce modèle, Kubernetes réconcilie en permanence le référentiel Git (état souhaité) avec les ressources qui s'exécutent à l'intérieur du cluster (état actuel). En cas de différences, le GitOps processus apporte automatiquement des modifications au cluster. Vous pouvez utiliser GitOps Kubernetes pour déployer et gérer des fonctions Lambda via Crossplane, en utilisant des outils et des concepts Kubernetes familiers tels que les CRD et les contrôleurs.

Pour en savoir plus sur l'utilisation de Crossplane avec Lambda, consultez les rubriques suivantes :

- Plans AWS pour Crossplane : ce référentiel inclut des exemples d'utilisation de Crossplane pour déployer des ressources AWS, y compris les fonctions Lambda.

  > **Note**
  >
  > Les plans AWS pour Crossplane sont en cours de développement et ne devraient pas être utilisés en production.

- Déploiement de Lambda avec Amazon EKS et Crossplane : cette vidéo montre un exemple avancé de déploiement d'une architecture sans serveur AWS avec Crossplane, explorant la conception du point de vue du développeur et de la plateforme.

# Exemples d'applications Lambda

Le GitHub référentiel de ce guide inclut des exemples d'applications illustrant l'utilisation de différents langages et AWS services. Chaque exemple d'application inclut des scripts pour faciliter le déploiement, le nettoyage et les ressources de support.

Node.js

Exemples d’applications Lambda en Node.js

- blank-nodejs — Une fonction Node.js qui montre l'utilisation de la journalisation, des variables d'environnement, du AWS X-Ray traçage, des couches, des tests unitaires et du SDK. AWS
- nodejs-apig – Fonction avec un point de terminaison d’API public qui traite un événement d’API Gateway et renvoie une réponse HTTP.
- efs-nodejs – Fonction qui utilise un système de fichiers Amazon EFS dans un VPC Amazon. Cet exemple inclut un VPC, un système de fichiers, des cibles de montage et un point d’accès configurés pour une utilisation avec Lambda.

Python

Exemples d’applications Lambda en Python

- blank-python — Fonction Python qui montre l'utilisation de la journalisation, des variables d'environnement, du AWS X-Ray traçage, des couches, des tests unitaires et du AWS SDK.

Ruby

Exemples d’applications Lambda en Ruby

- blank-ruby — Une fonction Ruby qui montre l'utilisation de la journalisation, des variables d'environnement, du AWS X-Ray traçage, des couches, des tests unitaires et du AWS SDK.
- Exemples de code Ruby pour AWS Lambda — Exemples de code écrits en Ruby qui montrent comment interagir avec Lambda AWS .

## Java

### Exemples d'applications Lambda en Java

- java17-examples : fonction Java qui montre comment utiliser un enregistrement Java pour représenter un objet de données d'événement en entrée.
- java-basic – Ensemble de fonctions Java minimales avec des tests unitaires et une configuration de journalisation variable.
- java events – Ensemble de fonctions Java contenant du code squelette permettant de gérer les événements de divers services tels qu'Amazon API Gateway, Amazon SQS et Amazon Kinesis. Ces fonctions utilisent la dernière version de la aws-lambda-java-eventsbibliothèque (3.0.0 et versions ultérieures). Ces exemples ne nécessitent pas le AWS SDK comme dépendance.
- s3-java – Fonction Java qui traite les événements de notification d'Amazon S3 et utilise la bibliothèque de classes Java (JCL) pour créer des miniatures à partir de fichiers d'image chargés.
- Utiliser API Gateway pour invoquer une fonction Lambda : Fonction Java qui analyse une table Amazon DynamoDB contenant les informations relatives aux employés. Il utilise ensuite Amazon Simple Notification Service pour envoyer un message texte aux employés qui fêtent leur anniversaire professionnel. Cet exemple utilise API Gateway pour invoquer la fonction.

### Exécution de cadres Java populaires sur Lambda

- spring-cloud-function-samples— Un exemple tiré de Spring qui montre comment utiliser le framework Spring Cloud Function pour créer des fonctions AWS Lambda.
- Démo de l'application Spring Boot sans serveur : exemple qui montre comment configurer une application Spring Boot typique dans un environnement d'exécution Java géré avec ou sans SnapStart, ou en tant qu'image native de GraalVM avec un environnement d'exécution personnalisé.
- Démonstration de l'application Micronaut sans serveur — Un exemple qui montre comment utiliser Micronaut dans un environnement d'exécution Java géré avec et sans SnapStart, ou en tant qu'image native de GraalVM avec un environnement d'exécution personnalisé. Pour en savoir plus, consultez les guides Micronaut/Lambda.
- Démo de l'application Quarkus sans serveur — Un exemple qui montre comment utiliser Quarkus dans un environnement d'exécution Java géré avec et sans SnapStart, ou en tant qu'image native de GraalVM avec un environnement d'exécution personnalisé. Pour en savoir plus, consultez le guide Quarkus/Lambda et le guide Quarkus/Lambda. SnapStart

### Go

Lambda fournit les exemples d'applications suivants pour l'environnement d'exécution Go :

Exemples d’applications Lambda en Go

- go-al2 – Une fonction Hello World qui renvoie l’adresse IP publique. Cette application utilise l’exécution personnalisée `provided.al2`.
- blank-go — Une fonction Go qui montre l'utilisation des bibliothèques Go de Lambda, de la journalisation, des variables d'environnement et du SDK. AWS Cette application utilise l'exécution `go1.x`.

### C#

Exemples d'applications Lambda en C#

- blank-csharp – Fonction C# montrant l'utilisation des bibliothèques .NET de Lambda, la journalisation, les variables d'environnement, le suivi AWS X-Ray , les tests unitaires et le kit AWS SDK.
- blank-csharp-with-layer— Fonction C# qui utilise la CLI .NET pour créer une couche qui regroupe les dépendances de la fonction.
- ec2-spot – Fonction qui gère les demandes d'instance Spot dans Amazon EC2.

### PowerShell

Lambda fournit les exemples d'applications suivants pour : PowerShell

- blank-powershell — PowerShell Fonction qui montre l'utilisation de la journalisation, des variables d'environnement et du SDK. AWS

Pour déployer un exemple d'application, suivez les instructions de son fichier README.

# Utilisation de Lambda avec un AWS SDK

AWS des kits de développement logiciel (SDKs) sont disponibles pour de nombreux langages de programmation courants. Chacun SDK fournit unAPI, des exemples de code et une documentation qui permettent aux développeurs de créer plus facilement des applications dans leur langage préféré.

| SDKdocumentation | Exemples de code |
| --- | --- |
| AWS SDK for C++ | AWS SDK for C++ exemples de code |
| AWS CLI | AWS CLI exemples de code |
| AWS SDK for Go | AWS SDK for Go exemples de code |
| AWS SDK for Java | AWS SDK for Java exemples de code |
| AWS SDK for JavaScript | AWS SDK for JavaScript exemples de code |
| Kit AWS SDK pour Kotlin | Kit AWS SDK pour Kotlin exemples de code |
| AWS SDK for .NET | AWS SDK for .NET exemples de code |
| AWS SDK for PHP | AWS SDK for PHP exemples de code |
| AWS Tools for PowerShell | Outils pour des exemples PowerShell de code |
| AWS SDK for Python (Boto3) | AWS SDK for Python (Boto3) exemples de code |
| AWS SDK for Ruby | AWS SDK for Ruby exemples de code |
| Kit AWS SDK pour Rust | Kit AWS SDK pour Rust exemples de code |
| AWS SDK pour SAP ABAP | AWS SDK pour SAP ABAP exemples de code |
| Kit AWS SDK pour Swift | Kit AWS SDK pour Swift exemples de code |

Pour plus d'exemples spécifiques à Lambda, consultez Exemples de code pour Lambda utilisant AWS SDKs.

Exemple de disponibilité

Vous n’avez pas trouvé ce dont vous avez besoin ? Demandez un exemple de code en utilisant le lien Provide feedback (Fournir un commentaire) en bas de cette page.

# Exemples de code pour Lambda utilisant AWS SDKs

Les exemples de code suivants montrent comment utiliser Lambda avec un kit de développement AWS logiciel ()SDK.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous montrent comment appeler des fonctions de service individuelles, vous pouvez les visualiser dans leur contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un service ou en les combinant à d'autres Services AWS.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

Mise en route

## Hello Lambda

Les exemples de code suivants montrent comment démarrer avec Lambda.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace LambdaActions;

using Amazon.Lambda;
```

```
public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
 existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {

 Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code du fichier CMakeLists CMake .txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)
```

```
# Set this project's name.
project("hello_lambda")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                 # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code pour le fichier source hello_lambda.cpp.

```
#include <aws/core/Aws.h>
```

```
#include <aws/lambda/LambdaClient.h>
#include <aws/lambda/model/ListFunctionsRequest.h>
#include <iostream>

/*
 *  A "Hello Lambda" starter application which initializes an AWS Lambda (Lambda)
 client and lists the Lambda functions.
 *
 *  main function
 *
 *  Usage: 'hello_lambda'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Lambda::LambdaClient lambdaClient(clientConfig);
        std::vector<Aws::String> functions;
        Aws::String marker; // Used for pagination.

        do {
            Aws::Lambda::Model::ListFunctionsRequest request;
            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::Lambda::Model::ListFunctionsOutcome outcome =
 lambdaClient.ListFunctions(
                    request);

            if (outcome.IsSuccess()) {
                const Aws::Lambda::Model::ListFunctionsResult
 &listFunctionsResult = outcome.GetResult();
                std::cout << listFunctionsResult.GetFunctions().size()
                          << " lambda functions were retrieved." << std::endl;
```

```
                for (const Aws::Lambda::Model::FunctionConfiguration
 &functionConfiguration: listFunctionsResult.GetFunctions()) {
                    functions.push_back(functionConfiguration.GetFunctionName());
                    std::cout << functions.size() << "  "
                              << functionConfiguration.GetDescription() <<
 std::endl;
                    std::cout << "   "
                              <<
 Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                      functionConfiguration.GetRuntime()) << ": "
                              << functionConfiguration.GetHandler()
                              << std::endl;
                }
                marker = listFunctionsResult.GetNextMarker();
            } else {
                std::cerr << "Error with Lambda::ListFunctions. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!marker.empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for C++ APIRéférence.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up
 to 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 lambdaClient := lambda.NewFromConfig(sdkConfig)

 maxItems := 10
 fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
 result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 if err != nil {
  fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Functions) == 0 {
  fmt.Println("You don't have any functions!")
 } else {
  for _, function := range result.Functions {
   fmt.Printf("\t%v\n", *function.FunctionName)
  }
```

```
 }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Go APIRéférence.

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
package com.example.lambda;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.ListFunctionsResponse;
import software.amazon.awssdk.services.lambda.model.FunctionConfiguration;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListLambdaFunctions {
    public static void main(String[] args) {
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();
```

```
        listFunctions(awsLambda);
        awsLambda.close();
    }

    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
                System.out.println("The function name is " +
 config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Java 2.x APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
```

```
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for JavaScript APIRéférence.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import boto3


def main():
    """
    List the Lambda functions in your AWS account.
    """
    # Create the Lambda client
    lambda_client = boto3.client("lambda")

    # Use the paginator to list the functions
    paginator = lambda_client.get_paginator("list_functions")
    response_iterator = paginator.paginate()

    print("Here are the Lambda functions in your account:")
    for page in response_iterator:
        for function in page["Functions"]:
```

```
            print(f"  {function['FunctionName']}")


if __name__ == "__main__":
    main()
```

- Pour API plus de détails, reportez-vous ListFunctionsà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
require 'aws-sdk-lambda'

# Creates an AWS Lambda client using the default credentials and configuration
def lambda_client
  Aws::Lambda::Client.new
end

# Lists the Lambda functions in your AWS account, paginating the results if
 necessary
def list_lambda_functions
  lambda = lambda_client

  # Use a pagination iterator to list all functions
  functions = []
  lambda.list_functions.each_page do |page|
    functions.concat(page.functions)
  end

  # Print the name and ARN of each function
```

```
  functions.each do |function|
    puts "Function name: #{function.function_name}"
    puts "Function ARN: #{function.function_arn}"
    puts
  end

  puts "Total functions: #{functions.count}"
end

list_lambda_functions if __FILE__ == $PROGRAM_NAME
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Ruby APIRéférence.

Exemples de code

- Exemples de base pour l'utilisation de Lambda AWS SDKs
  - Hello Lambda
  - Actions pour Lambda utilisant AWS SDKs
    - À utiliser CreateAlias avec un AWS SDK ou CLI
    - À utiliser CreateFunction avec un AWS SDK ou CLI
    - À utiliser DeleteAlias avec un AWS SDK ou CLI
    - À utiliser DeleteFunction avec un AWS SDK ou CLI
    - À utiliser DeleteFunctionConcurrency avec un AWS SDK ou CLI
    - À utiliser DeleteProvisionedConcurrencyConfig avec un AWS SDK ou CLI
    - À utiliser GetAccountSettings avec un AWS SDK ou CLI
    - À utiliser GetAlias avec un AWS SDK ou CLI
    - À utiliser GetFunction avec un AWS SDK ou CLI
    - À utiliser GetFunctionConcurrency avec un AWS SDK ou CLI
    - À utiliser GetFunctionConfiguration avec un AWS SDK ou CLI
    - À utiliser GetPolicy avec un AWS SDK ou CLI
    - À utiliser GetProvisionedConcurrencyConfig avec un AWS SDK ou CLI
    - À utiliser Invoke avec un AWS SDK ou CLI
    - À utiliser ListFunctions avec un AWS SDK ou CLI

    - À utiliser ListProvisionedConcurrencyConfigs avec un AWS SDK ou CLI
    - À utiliser ListTags avec un AWS SDK ou CLI
    - À utiliser ListVersionsByFunction avec un AWS SDK ou CLI
    - À utiliser PublishVersion avec un AWS SDK ou CLI
    - À utiliser PutFunctionConcurrency avec un AWS SDK ou CLI
    - À utiliser PutProvisionedConcurrencyConfig avec un AWS SDK ou CLI
    - À utiliser RemovePermission avec un AWS SDK ou CLI
    - À utiliser TagResource avec un AWS SDK ou CLI
    - À utiliser UntagResource avec un AWS SDK ou CLI
    - À utiliser UpdateAlias avec un AWS SDK ou CLI
    - À utiliser UpdateFunctionCode avec un AWS SDK ou CLI
    - À utiliser UpdateFunctionConfiguration avec un AWS SDK ou CLI
- Scénarios d'utilisation de Lambda AWS SDKs
    - Confirmez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK
    - Migrez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK
    - Créez une API passerelle REST API pour suivre les données COVID -19
    - Création d'une bibliothèque de prêt REST API
    - Créer une application de messagerie avec Step Functions
    - Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l’aide d’étiquettes
    - Créez une application de chat Websocket avec Gateway API
    - Créez une application qui analyse les commentaires des clients et synthétise le son
    - Commencez à créer et à invoquer des fonctions Lambda à l'aide d'un AWS SDK
    - Invoquer une fonction Lambda à partir d’un navigateur
    - Transformez les données de votre application avec S3 Object Lambda
    - Utiliser API Gateway pour appeler une fonction Lambda
    - Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda
    - Utilisent des événements planifiés pour appeler une fonction Lambda

- Rédigez des données d'activité personnalisées à l'aide d'une fonction Lambda après l'authentification de l'utilisateur Amazon Cognito à l'aide d'un AWS SDK
- Exemples d'utilisation de Lambda sans serveur AWS SDKs
  - Connexion à une RDS base de données Amazon dans une fonction Lambda
  - Invoquer une fonction Lambda à partir d'un déclencheur Kinesis
  - Invoquer une fonction Lambda à partir d'un déclencheur DynamoDB
  - Invoquer une fonction Lambda depuis un déclencheur Amazon DocumentDB
  - Invoquer une fonction Lambda depuis un déclencheur Amazon MSK
  - Invoquer une fonction lambda à partir d'un déclencheur Amazon S3
  - Invoquer une fonction Lambda depuis un déclencheur Amazon SNS
  - Invoquer une fonction Lambda depuis un déclencheur Amazon SQS
  - Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis
  - Signalement des défaillances d'éléments de lot pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB
  - Signalement des défaillances d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

# Exemples de base pour l'utilisation de Lambda AWS SDKs

Les exemples de code suivants montrent comment utiliser les principes de base de AWS Lambda with AWS SDKs.

Exemples

- Hello Lambda
- Actions pour Lambda utilisant AWS SDKs
  - À utiliser CreateAlias avec un AWS SDK ou CLI
  - À utiliser CreateFunction avec un AWS SDK ou CLI
  - À utiliser DeleteAlias avec un AWS SDK ou CLI
  - À utiliser DeleteFunction avec un AWS SDK ou CLI
  - À utiliser DeleteFunctionConcurrency avec un AWS SDK ou CLI
  - À utiliser DeleteProvisionedConcurrencyConfig avec un AWS SDK ou CLI

- À utiliser GetAccountSettings avec un AWS SDK ou CLI
- À utiliser GetAlias avec un AWS SDK ou CLI
- À utiliser GetFunction avec un AWS SDK ou CLI
- À utiliser GetFunctionConcurrency avec un AWS SDK ou CLI
- À utiliser GetFunctionConfiguration avec un AWS SDK ou CLI
- À utiliser GetPolicy avec un AWS SDK ou CLI
- À utiliser GetProvisionedConcurrencyConfig avec un AWS SDK ou CLI
- À utiliser Invoke avec un AWS SDK ou CLI
- À utiliser ListFunctions avec un AWS SDK ou CLI
- À utiliser ListProvisionedConcurrencyConfigs avec un AWS SDK ou CLI
- À utiliser ListTags avec un AWS SDK ou CLI
- À utiliser ListVersionsByFunction avec un AWS SDK ou CLI
- À utiliser PublishVersion avec un AWS SDK ou CLI
- À utiliser PutFunctionConcurrency avec un AWS SDK ou CLI
- À utiliser PutProvisionedConcurrencyConfig avec un AWS SDK ou CLI
- À utiliser RemovePermission avec un AWS SDK ou CLI
- À utiliser TagResource avec un AWS SDK ou CLI
- À utiliser UntagResource avec un AWS SDK ou CLI
- À utiliser UpdateAlias avec un AWS SDK ou CLI
- À utiliser UpdateFunctionCode avec un AWS SDK ou CLI
- À utiliser UpdateFunctionConfiguration avec un AWS SDK ou CLI

## Hello Lambda

Les exemples de code suivants montrent comment démarrer avec Lambda.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
 existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {

 Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code du fichier CMakeLists CMake .txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)

# Set this project's name.
project("hello_lambda")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code pour le fichier source hello_lambda.cpp.

```
#include <aws/core/Aws.h>
#include <aws/lambda/LambdaClient.h>
#include <aws/lambda/model/ListFunctionsRequest.h>
#include <iostream>

/*
 *  A "Hello Lambda" starter application which initializes an AWS Lambda (Lambda)
 client and lists the Lambda functions.
 *
 *  main function
 *
 *  Usage: 'hello_lambda'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
        Aws::Lambda::LambdaClient lambdaClient(clientConfig);
        std::vector<Aws::String> functions;
        Aws::String marker; // Used for pagination.

        do {
            Aws::Lambda::Model::ListFunctionsRequest request;
            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::Lambda::Model::ListFunctionsOutcome outcome =
lambdaClient.ListFunctions(
                    request);

            if (outcome.IsSuccess()) {
                const Aws::Lambda::Model::ListFunctionsResult
&listFunctionsResult = outcome.GetResult();
                std::cout << listFunctionsResult.GetFunctions().size()
                          << " lambda functions were retrieved." << std::endl;

                for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: listFunctionsResult.GetFunctions()) {
                    functions.push_back(functionConfiguration.GetFunctionName());
                    std::cout << functions.size() << "  "
                              << functionConfiguration.GetDescription() <<
std::endl;
                    std::cout << "   "
                              <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                      functionConfiguration.GetRuntime()) << ": "
                              << functionConfiguration.GetHandler()
                              << std::endl;
                }
                marker = listFunctionsResult.GetNextMarker();
            } else {
                std::cerr << "Error with Lambda::ListFunctions. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!marker.empty());
    }
```

```
    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for C++ APIRéférence.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up
 to 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
```

```
  fmt.Println(err)
  return
 }
 lambdaClient := lambda.NewFromConfig(sdkConfig)

 maxItems := 10
 fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
 result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 if err != nil {
  fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Functions) == 0 {
  fmt.Println("You don't have any functions!")
 } else {
  for _, function := range result.Functions {
   fmt.Printf("\t%v\n", *function.FunctionName)
  }
 }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Go APIRéférence.

## Java

### SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
package com.example.lambda;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.LambdaException;
```

```
import software.amazon.awssdk.services.lambda.model.ListFunctionsResponse;
import software.amazon.awssdk.services.lambda.model.FunctionConfiguration;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListLambdaFunctions {
    public static void main(String[] args) {
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        listFunctions(awsLambda);
        awsLambda.close();
    }

    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
                System.out.println("The function name is " +
 config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Java 2.x APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for JavaScript APIRéférence.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import boto3


def main():
    """
    List the Lambda functions in your AWS account.
    """
    # Create the Lambda client
    lambda_client = boto3.client("lambda")

    # Use the paginator to list the functions
    paginator = lambda_client.get_paginator("list_functions")
    response_iterator = paginator.paginate()

    print("Here are the Lambda functions in your account:")
    for page in response_iterator:
        for function in page["Functions"]:
            print(f"  {function['FunctionName']}")


if __name__ == "__main__":
    main()
```

- Pour API plus de détails, reportez-vous ListFunctionsà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
require 'aws-sdk-lambda'
```

```
# Creates an AWS Lambda client using the default credentials and configuration
def lambda_client
  Aws::Lambda::Client.new
end

# Lists the Lambda functions in your AWS account, paginating the results if
 necessary
def list_lambda_functions
  lambda = lambda_client

  # Use a pagination iterator to list all functions
  functions = []
  lambda.list_functions.each_page do |page|
    functions.concat(page.functions)
  end

  # Print the name and ARN of each function
  functions.each do |function|
    puts "Function name: #{function.function_name}"
    puts "Function ARN: #{function.function_arn}"
    puts
  end

  puts "Total functions: #{functions.count}"
end

list_lambda_functions if __FILE__ == $PROGRAM_NAME
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Ruby APIRéférence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Actions pour Lambda utilisant AWS SDKs

Les exemples de code suivants montrent comment effectuer des actions Lambda individuelles avec. AWS SDKs Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions pour configurer et exécuter le code.

Ces extraits, appelés API Lambda, sont des extraits de code de programmes plus volumineux qui doivent être exécutés en contexte. Vous pouvez voir les actions dans leur contexte dansScénarios d'utilisation de Lambda AWS SDKs .

Les exemples suivants incluent uniquement les actions les plus couramment utilisées. Pour une liste complète, voir la AWS Lambda APIréférence.

Exemples

- À utiliser CreateAlias avec un AWS SDK ou CLI
- À utiliser CreateFunction avec un AWS SDK ou CLI
- À utiliser DeleteAlias avec un AWS SDK ou CLI
- À utiliser DeleteFunction avec un AWS SDK ou CLI
- À utiliser DeleteFunctionConcurrency avec un AWS SDK ou CLI
- À utiliser DeleteProvisionedConcurrencyConfig avec un AWS SDK ou CLI
- À utiliser GetAccountSettings avec un AWS SDK ou CLI
- À utiliser GetAlias avec un AWS SDK ou CLI
- À utiliser GetFunction avec un AWS SDK ou CLI
- À utiliser GetFunctionConcurrency avec un AWS SDK ou CLI
- À utiliser GetFunctionConfiguration avec un AWS SDK ou CLI
- À utiliser GetPolicy avec un AWS SDK ou CLI
- À utiliser GetProvisionedConcurrencyConfig avec un AWS SDK ou CLI
- À utiliser Invoke avec un AWS SDK ou CLI
- À utiliser ListFunctions avec un AWS SDK ou CLI
- À utiliser ListProvisionedConcurrencyConfigs avec un AWS SDK ou CLI
- À utiliser ListTags avec un AWS SDK ou CLI
- À utiliser ListVersionsByFunction avec un AWS SDK ou CLI
- À utiliser PublishVersion avec un AWS SDK ou CLI
- À utiliser PutFunctionConcurrency avec un AWS SDK ou CLI
- À utiliser PutProvisionedConcurrencyConfig avec un AWS SDK ou CLI
- À utiliser RemovePermission avec un AWS SDK ou CLI
- À utiliser TagResource avec un AWS SDK ou CLI

- À utiliser UntagResource avec un AWS SDK ou CLI
- À utiliser UpdateAlias avec un AWS SDK ou CLI
- À utiliser UpdateFunctionCode avec un AWS SDK ou CLI
- À utiliser UpdateFunctionConfiguration avec un AWS SDK ou CLI

## À utiliser **`CreateAlias`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`CreateAlias`.

CLI

AWS CLI

Pour créer un alias pour une fonction Lambda

L'`create-alias`exemple suivant crée un alias nommé `LIVE` qui pointe vers la version 1 de la fonction `my-function` Lambda.

```
aws lambda create-alias \
    --function-name my-function \
    --description "alias for live version of function" \
    --function-version 1 \
    --name LIVE
```

Sortie :

```
{
    "FunctionVersion": "1",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",
    "Description": "alias for live version of function"
}
```

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir CreateAliasla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple crée un nouvel alias Lambda pour une version et une configuration de routage spécifiées afin de spécifier le pourcentage de demandes d'appel qu'il reçoit.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Pour API plus de détails, consultez la section CreateAliasRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`CreateFunction`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`CreateFunction`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Creates a new Lambda function.
```

```
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
 (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::CreateFunctionRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
        request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
        request.SetTimeout(15);
        request.SetMemorySize(128);

        // Assume the AWS Lambda function was built in Docker with same
 architecture
        // as this code.
#if  defined(__x86_64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_8);
#endif
        request.SetRole(roleArn);
        request.SetHandler(LAMBDA_HANDLER_NAME);
```

```
        request.SetPublish(true);
        Aws::Lambda::Model::FunctionCode code;
        std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();

        code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
 buffer.str().c_str(),
                                               buffer.str().length()));
        request.SetCode(code);

        Aws::Lambda::Model::CreateFunctionOutcome outcome =
 client.CreateFunction(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda function was successfully created. " <<
 seconds
                      << " seconds elapsed." << std::endl;
            break;
        }

        else {
            std::cerr << "Error with CreateFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for C++ APIRéférence.

CLI

AWS CLI

Pour créer une fonction Lambda

L'exemple `create-function` suivant crée une fonction Lambda nommée `my-function`.

```
aws lambda create-function \
    --function-name my-function \
    --runtime nodejs18.x \
    --zip-file fileb://my-function.zip \
    --handler my-function.handler \
    --role arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
tges6bf4
```

Contenu de `my-function.zip` :

```
This file is a deployment package that contains your function code and any
 dependencies.
```

Sortie :

```
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "PFn4S+er27qk+UuZSTKEQfNKG/XNn7QJs90mJgq6oH8=",
    "FunctionName": "my-function",
    "CodeSize": 308,
    "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
zgur6bf4",
    "Timeout": 3,
```

```
    "LastModified": "2023-10-14T22:26:11.234+0000",
    "Handler": "my-function.handler",
    "Runtime": "nodejs18.x",
    "Description": ""
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir CreateFunctionla section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
```

```
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython38,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for Go APIRéférence.

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.CreateFunctionRequest;
import software.amazon.awssdk.services.lambda.model.FunctionCode;
import software.amazon.awssdk.services.lambda.model.CreateFunctionResponse;
import software.amazon.awssdk.services.lambda.model.GetFunctionRequest;
import software.amazon.awssdk.services.lambda.model.GetFunctionResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.Runtime;
import software.amazon.awssdk.services.lambda.waiters.LambdaWaiter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

/**
 * This code example requires a ZIP or JAR that represents the code of the
 * Lambda function.
 * If you do not have a ZIP or JAR, please refer to the following document:
 *
 * https://github.com/aws-doc-sdk-examples/tree/master/javav2/usecases/
creating_workflows_stepfunctions
 *
 * Also, set up your development environment, including your credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class CreateFunction {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <functionName> <filePath> <role> <handler>\s

                Where:
                    functionName - The name of the Lambda function.\s
                    filePath - The path to the ZIP or JAR where the code is
 located.\s
                    role - The role ARN that has Lambda permissions.\s
                    handler - The fully qualified method name (for example,
 example.Handler::handleRequest). \s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        String filePath = args[1];
        String role = args[2];
        String handler = args[3];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        createLambdaFunction(awsLambda, functionName, filePath, role, handler);
        awsLambda.close();
    }

    public static void createLambdaFunction(LambdaClient awsLambda,
            String functionName,
            String filePath,
            String role,
            String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            InputStream is = new FileInputStream(filePath);
```

```
            SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

            FunctionCode code = FunctionCode.builder()
                    .zipFile(fileToUpload)
                    .build();

            CreateFunctionRequest functionRequest =
 CreateFunctionRequest.builder()
                    .functionName(functionName)
                    .description("Created by the Lambda Java API")
                    .code(code)
                    .handler(handler)
                    .runtime(Runtime.JAVA8)
                    .role(role)
                    .build();

            // Create a Lambda function using a waiter.
            CreateFunctionResponse functionResponse =
 awsLambda.createFunction(functionRequest);
            GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
                    .functionName(functionName)
                    .build();
            WaiterResponse<GetFunctionResponse> waiterResponse =
 waiter.waitUntilFunctionExists(getFunctionRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("The function ARN is " +
 functionResponse.functionArn());

        } catch (LambdaException | FileNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for Java 2.x APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for JavaScript APIRéférence.

Kotlin

SDKpour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun createNewFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String? {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }

    val request =
        CreateFunctionRequest {
            functionName = myFunctionName
            code = functionCode
            description = "Created by the Lambda Kotlin API"
            handler = myHandler
            role = myRole
            runtime = Runtime.Java8
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn
    }
}
```

- Pour API plus de détails, voir CreateFunctionla APIréférence AWS SDK à Kotlin.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function createFunction($functionName, $role, $bucketName, $handler)
    {
        //This assumes the Lambda function is in an S3 bucket.
        return $this->customWaiter(function () use ($functionName, $role,
$bucketName, $handler) {
            return $this->lambdaClient->createFunction([
                'Code' => [
                    'S3Bucket' => $bucketName,
                    'S3Key' => $functionName,
                ],
                'FunctionName' => $functionName,
                'Role' => $role['Arn'],
                'Runtime' => 'python3.9',
                'Handler' => "$handler.lambda_handler",
            ]);
        });
    }
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for PHP APIRéférence.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple crée une nouvelle fonction C# (dotnetcore1.0 runtime) nommée dans MyFunction AWS Lambda, fournissant les fichiers binaires compilés pour la fonction à partir d'un fichier zip sur le système de fichiers local (des chemins relatifs ou absolus peuvent être utilisés). Les fonctions Lambda C# spécifient le gestionnaire de la fonction en utilisant la désignation : :Namespace. AssemblyName ClassName: :MethodName. Vous devez remplacer les parties du nom de l'assembly (sans le suffixe .dll), de l'espace de noms, du nom de classe

et du nom de méthode de la spécification du gestionnaire de manière appropriée. La nouvelle fonction aura les variables d'environnement 'envvar1' et 'envvar2' configurées à partir des valeurs fournies.

```
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
        -ZipFilename .\MyFunctionBinaries.zip `
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

Sortie :

```
CodeSha256       : /NgBMd...gq71I=
CodeSize         : 214784
DeadLetterConfig :
Description      : My C# Lambda Function
Environment      : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn      : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName     : MyFunction
Handler          : AssemblyName::Namespace.ClassName::MethodName
KMSKeyArn        :
LastModified     : 2016-12-29T23:50:14.207+0000
MemorySize       : 128
Role             : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime          : dotnetcore1.0
Timeout          : 3
Version          : $LATEST
VpcConfig        :
```

Exemple 2 : Cet exemple est similaire au précédent, sauf que les fichiers binaires des fonctions sont d'abord chargés dans un compartiment Amazon S3 (qui doit se trouver dans la même région que la fonction Lambda prévue) et que l'objet S3 obtenu est ensuite référencé lors de la création de la fonction.

```
Write-S3Object -BucketName amzn-s3-demo-bucket -Key MyFunctionBinaries.zip -
File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
        -BucketName amzn-s3-demo-bucket `
        -Key MyFunctionBinaries.zip `
```

```
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- Pour API plus de détails, consultez la section CreateFunctionRéférence des AWS Tools for PowerShell applets de commande.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
 This
                             must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
 function
                                   code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
        """
        try:
```

```
            response = self.lambda_client.create_function(
                FunctionName=function_name,
                Description="AWS Lambda doc example",
                Runtime="python3.8",
                Role=iam_role.arn,
                Handler=handler_name,
                Code={"ZipFile": deployment_package},
                Publish=True,
            )
            function_arn = response["FunctionArn"]
            waiter = self.lambda_client.get_waiter("function_active_v2")
            waiter.wait(FunctionName=function_name)
            logger.info(
                "Created function '%s' with ARN: '%s'.",
                function_name,
                response["FunctionArn"],
            )
        except ClientError:
            logger.error("Couldn't create function %s.", function_name)
            raise
        else:
            return function_arn
```

- Pour API plus de détails, reportez-vous CreateFunctionà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client
```

```
 def initialize
   @lambda_client = Aws::Lambda::Client.new
   @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
   @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
   @logger = Logger.new($stdout)
   @logger.level = Logger::WARN
 end

 # Deploys a Lambda function.
 #
 # @param function_name: The name of the Lambda function.
 # @param handler_name: The fully qualified name of the handler function.
 # @param role_arn: The IAM role to use for the function.
 # @param deployment_package: The deployment package that contains the function
code in .zip format.
 # @return: The Amazon Resource Name (ARN) of the newly created function.
 def create_function(function_name, handler_name, role_arn, deployment_package)
   response = @lambda_client.create_function({
                                               role: role_arn.to_s,
                                               function_name: function_name,
                                               handler: handler_name,
                                               runtime: 'ruby2.7',
                                               code: {
                                                 zip_file: deployment_package
                                               },
                                               environment: {
                                                 variables: {
                                                   'LOG_LEVEL' => 'info'
                                                 }
                                               }
                                             })
   @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
   response
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error creating #{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

- Pour API plus de détails, voir CreateFunctionla section AWS SDK for Ruby APIRéférence.

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;
```

```
        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }
```

- Pour API plus de détails, reportez-vous CreateFunctionà la section AWS SDKpour la API référence à Rust.

## SAP ABAP

### SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        lo_lmd->createfunction(
            iv_functionname = iv_function_name
            iv_runtime = `python3.9`
            iv_role = iv_role_arn
            iv_handler = iv_handler
            io_code = io_zip_file
            iv_description = 'AWS Lambda code example'
        ).
        MESSAGE 'Lambda function created.' TYPE 'I'.
      CATCH /aws1/cx_lmdcodesigningcfgno00.
        MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodestorageexcdex.
        MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodeverification00.
        MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvalidcodesigex.
        MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
```

```
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Pour API plus de détails, voir CreateFunction AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`DeleteAlias`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`DeleteAlias`.

CLI

AWS CLI

Pour supprimer l'alias d'une fonction Lambda

L'`delete-alias`exemple suivant supprime l'alias nommé dans la `LIVE` fonction `my-function` Lambda.

```
aws lambda delete-alias \
    --function-name my-function \
    --name LIVE
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir DeleteAliasla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple supprime la fonction Lambda Alias mentionnée dans la commande.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Pour API plus de détails, consultez la section DeleteAliasRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`DeleteFunction`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`DeleteFunction`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
```

```
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

    Aws::Lambda::Model::DeleteFunctionRequest request;
    request.SetFunctionName(LAMBDA_NAME);

    Aws::Lambda::Model::DeleteFunctionOutcome outcome = client.DeleteFunction(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda function was successfully deleted." <<
std::endl;
```

```
    }
    else {
        std::cerr << "Error with Lambda::DeleteFunction. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for C++ APIRéférence.

CLI

AWS CLI

Exemple 1 : pour supprimer une fonction Lambda par nom de fonction

L'exemple `delete-function` suivant supprime la fonction Lambda `my-function` nommée en spécifiant le nom de la fonction.

```
aws lambda delete-function \
    --function-name my-function
```

Cette commande ne produit aucun résultat.

Exemple 2 : Pour supprimer une fonction Lambda par fonction ARN

L'`delete-function`exemple suivant supprime la fonction Lambda `my-function` nommée en spécifiant celle de la fonction. ARN

```
aws lambda delete-function \
    --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Cette commande ne produit aucun résultat.

Exemple 3 : Pour supprimer une fonction Lambda par fonction partielle ARN

L'`delete-function`exemple suivant supprime la fonction Lambda `my-function` nommée en spécifiant le partiel de la fonction. ARN

```
aws lambda delete-function \
    --function-name 123456789012:function:my-function
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir DeleteFunctionla section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for Go APIRéférence.

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.DeleteFunctionRequest;
import software.amazon.awssdk.services.lambda.model.LambdaException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFunction {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <functionName>\s

                Where:
                    functionName - The name of the Lambda function.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        Region region = Region.US_EAST_1;
```

```
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        deleteLambdaFunction(awsLambda, functionName);
        awsLambda.close();
    }

    public static void deleteLambdaFunction(LambdaClient awsLambda, String
 functionName) {
        try {
            DeleteFunctionRequest request = DeleteFunctionRequest.builder()
                    .functionName(functionName)
                    .build();

            awsLambda.deleteFunction(request);
            System.out.println("The " + functionName + " function was deleted");

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for Java 2.x APIRéférence.

## JavaScript

### SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/**
 * @param {string} funcName
```

```
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for JavaScript APIRéférence.

Kotlin

SDKpour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun delLambdaFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Pour API plus de détails, voir DeleteFunctionla APIréférence AWS SDK à Kotlin.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for PHP APIRéférence.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple supprime une version spécifique d'une fonction Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Pour API plus de détails, consultez la section DeleteFunctionRéférence des AWS Tools for PowerShell applets de commande.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise
```

- Pour API plus de détails, reportez-vous DeleteFunctionà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
```

```
    @logger.level = Logger::WARN
  end

  # Deletes a Lambda function.
  # @param function_name: The name of the function to delete.
  def delete_function(function_name)
    print "Deleting function: #{function_name}..."
    @lambda_client.delete_function(
      function_name: function_name
    )
    print 'Done!'.green
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
  end
```

- Pour API plus de détails, voir DeleteFunctionla section AWS SDK for Ruby APIRéférence.

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
```

```
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                info!(?location, "Skipping delete object");
                None
            };

        (delete_function, delete_role, delete_object)
    }
```

- Pour API plus de détails, reportez-vous DeleteFunctionà la section AWS SDKpour la API référence à Rust.

SAP ABAP

SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       lo_lmd->deletefunction( iv_functionname = iv_function_name ).
       MESSAGE 'Lambda function deleted.' TYPE 'I'.
     CATCH /aws1/cx_lmdinvparamvalueex.
       MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
     CATCH /aws1/cx_lmdresourceconflictex.
       MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
     CATCH /aws1/cx_lmdresourcenotfoundex.
       MESSAGE 'The requested resource does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdserviceexception.
       MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
     CATCH /aws1/cx_lmdtoomanyrequestsex.
       MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
   ENDTRY.
```

- Pour API plus de détails, voir DeleteFunction AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`DeleteFunctionConcurrency`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`DeleteFunctionConcurrency`.

CLI

AWS CLI

Pour supprimer la limite d'exécution simultanée réservée à une fonction

L'`delete-function-concurrency`exemple suivant supprime la limite d'exécution simultanée réservée de la `my-function` fonction.

```
aws lambda delete-function-concurrency \
    --function-name  my-function
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Réserver la simultanéité pour une fonction Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir DeleteFunctionConcurrencyla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple supprime la simultanéité des fonctions de la fonction Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Pour API plus de détails, consultez la section DeleteFunctionConcurrencyRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`DeleteProvisionedConcurrencyConfig`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`DeleteProvisionedConcurrencyConfig`.

CLI

AWS CLI

Pour supprimer une configuration de simultanéité provisionnée

L'`delete-provisioned-concurrency-config`exemple suivant supprime la configuration de simultanéité provisionnée pour l'`GREEN`alias de la fonction spécifiée.

```
aws lambda delete-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier GREEN
```

- Pour API plus de détails, voir DeleteProvisionedConcurrencyConfigla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple supprime la configuration de simultanéité provisionnée pour un alias spécifique.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Pour API plus de détails, consultez la section DeleteProvisionedConcurrencyConfigRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetAccountSettings`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetAccountSettings`.

CLI

AWS CLI

Pour récupérer les informations relatives à votre compte dans une AWS région

L'`get-account-settings`exemple suivant affiche les limites Lambda et les informations d'utilisation de votre compte.

```
aws lambda get-account-settings
```

Sortie :

```
{
    "AccountLimit": {
       "CodeSizeUnzipped": 262144000,
       "UnreservedConcurrentExecutions": 1000,
       "ConcurrentExecutions": 1000,
       "CodeSizeZipped": 52428800,
       "TotalCodeSize": 80530636800
    },
    "AccountUsage": {
       "FunctionCount": 4,
       "TotalCodeSize": 9426
    }
}
```

Pour plus d'informations, consultez la section Limites AWS Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir GetAccountSettingsla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple s'affiche pour comparer la limite du compte et l'utilisation du compte

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Sortie :

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Pour API plus de détails, consultez la section GetAccountSettingsRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetAlias`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetAlias`.

CLI

AWS CLI

Pour récupérer les informations relatives à un alias de fonction

L'`get-alias`exemple suivant affiche les détails de l'alias nommé dans la `LIVE` fonction `my-function` Lambda.

```
aws lambda get-alias \
    --function-name my-function \
    --name LIVE
```

Sortie :

```
{
    "FunctionVersion": "3",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",
    "Description": "alias for live version of function"
}
```

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir GetAliasla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple récupère les poids de la configuration de routage pour un alias de fonction Lambda spécifique.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Sortie :

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Pour API plus de détails, consultez la section GetAliasRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetFunction`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetFunction`.

Les exemples d’actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l’exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::GetFunctionRequest request;
        request.SetFunctionName(functionName);

        Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

        if (outcome.IsSuccess()) {
            std::cout << "Function retrieve.\n" <<

outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
                      << std::endl;
        }
        else {
            std::cerr << "Error with Lambda::GetFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for C++ APIRéférence.

## CLI

### AWS CLI

#### Pour récupérer des informations sur une fonction

L'exemple `get-function` suivant affiche des informations sur la fonction `my-function`.

```
aws lambda get-function \
    --function-name  my-function
```

Sortie :

```
{
```

```
    "Concurrency": {
        "ReservedConcurrentExecutions": 100
    },
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://awslambda-us-west-2-tasks.s3.us-
west-2.amazonaws.com/snapshots/123456789012/my-function..."
    },
    "Configuration": {
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "$LATEST",
        "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJmlKidWaaCgk=",
        "FunctionName": "my-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 128,
        "RevisionId": "28f0fb31-5c5c-43d3-8955-03e76c5c1075",
        "CodeSize": 304,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/helloWorldPython-
role-uy3l9qyq",
        "Timeout": 3,
        "LastModified": "2019-09-24T18:20:35.054+0000",
        "Runtime": "nodejs10.x",
        "Description": ""
    }
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir GetFunctionla section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
 string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
 &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for Go APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for JavaScript APIRéférence.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function getFunction($functionName)
    {
        return $this->lambdaClient->getFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for PHP APIRéférence.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def get_function(self, function_name):
        """
        Gets data about a Lambda function.

        :param function_name: The name of the function.
        :return: The function data.
        """
        response = None
        try:
            response =
 self.lambda_client.get_function(FunctionName=function_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Function %s does not exist.", function_name)
            else:
                logger.error(
                    "Couldn't get function %s. Here's why: %s: %s",
                    function_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response
```

- Pour API plus de détails, reportez-vous GetFunctionà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Gets data about a Lambda function.
  #
  # @param function_name: The name of the function.
  # @return response: The function data, or nil if no such function exists.
  def get_function(function_name)
    @lambda_client.get_function(
      {
        function_name: function_name
      }
    )
  rescue Aws::Lambda::Errors::ResourceNotFoundException => e
    @logger.debug("Could not find function: #{function_name}:\n #{e.message}")
    nil
  end
```

- Pour API plus de détails, voir GetFunctionla section AWS SDK for Ruby APIRéférence.

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /** Get the Lambda function with this Manager's name. */
   pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
       info!("Getting lambda function");
       self.lambda_client
           .get_function()
           .function_name(self.lambda_name.clone())
           .send()
           .await
           .map_err(anyhow::Error::from)
   }
```

- Pour API plus de détails, reportez-vous GetFunctionà la section AWS SDKpour la API référence à Rust.

SAP ABAP

SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       oo_result =  lo_lmd->getfunction( iv_functionname = iv_function_name ).
    " oo_result is returned for testing purposes. "
       MESSAGE 'Lambda function information retrieved.' TYPE 'I'.
```

```
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Pour API plus de détails, voir GetFunction AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetFunctionConcurrency`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetFunctionConcurrency`.

CLI

AWS CLI

Pour afficher le paramètre de simultanéité réservé pour une fonction

L'`get-function-concurrency`exemple suivant récupère le paramètre de simultanéité réservé pour la fonction spécifiée.

```
aws lambda get-function-concurrency \
    --function-name my-function
```

Sortie :

```
{
    "ReservedConcurrentExecutions": 250
}
```

- Pour API plus de détails, voir GetFunctionConcurrencyla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple permet d'obtenir la simultanéité réservée pour la fonction Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Sortie :

```
ReservedConcurrentExecutions
----------------------------
100
```

- Pour API plus de détails, consultez la section GetFunctionConcurrencyRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetFunctionConfiguration`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetFunctionConfiguration`.

CLI

AWS CLI

Pour récupérer les paramètres spécifiques à la version d'une fonction Lambda

L'`get-function-configuration`exemple suivant affiche les paramètres de la version 2 de la `my-function` fonction.

```
aws lambda get-function-configuration \
    --function-name  my-function:2
```

Sortie :

```
{
```

```
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "MemorySize": 256,
    "Version": "2",
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
    "Handler": "index.handler"
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir GetFunctionConfigurationla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple renvoie la configuration spécifique à la version d'une fonction Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
 "PowershellAlias"
```

Sortie :

```
CodeSha256                  : uWOW0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize                    : 1426
DeadLetterConfig            : Amazon.Lambda.Model.DeadLetterConfig
Description                 : Verson 3 to test Aliases
Environment                 : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn                 : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                              :PowershellAlias
FunctionName                : MylambdaFunction123
Handler                     : lambda_function.launch_instance
KMSKeyArn                   :
LastModified                : 2019-12-25T09:52:59.872+0000
LastUpdateStatus            : Successful
LastUpdateStatusReason      :
LastUpdateStatusReasonCode  :
Layers                      : {}
MasterArn                   :
MemorySize                  : 128
RevisionId                  : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                        : arn:aws:iam::123456789012:role/service-role/lambda
Runtime                     : python3.8
State                       : Active
StateReason                 :
StateReasonCode             :
Timeout                     : 600
TracingConfig               : Amazon.Lambda.Model.TracingConfigResponse
Version                     : 4
VpcConfig                   : Amazon.Lambda.Model.VpcConfigDetail
```

- Pour API plus de détails, consultez la section GetFunctionConfigurationRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetPolicy`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetPolicy`.

CLI

AWS CLI

Pour récupérer la IAM politique basée sur les ressources pour une fonction, une version ou un alias

L'`get-policy`exemple suivant affiche les informations de politique relatives à la fonction `my-function` Lambda.

```
aws lambda get-policy \
    --function-name my-function
```

Sortie :

```
{
    "Policy": {
        "Version":"2012-10-17",
        "Id":"default",
        "Statement":
        [
            {
                "Sid":"iot-events",
                "Effect":"Allow",
                "Principal": {"Service":"iotevents.amazonaws.com"},
                "Action":"lambda:InvokeFunction",
                "Resource":"arn:aws:lambda:us-west-2:123456789012:function:my-
function"
            }
        ]
    },
    "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668"
}
```

Pour plus d'informations, consultez la section Utilisation de politiques basées sur les ressources pour Lambda AWS dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir GetPolicyla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple affiche la politique de fonction de la fonction Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Sortie :

```
{"Version":"2012-10-17","Id":"default","Statement":
[{"Sid":"xxxx","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lamb
east-1:123456789102:function:test"}]}
```

- Pour API plus de détails, consultez la section GetPolicyRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`GetProvisionedConcurrencyConfig`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`GetProvisionedConcurrencyConfig`.

CLI

AWS CLI

Pour consulter une configuration de simultanéité provisionnée

L'`get-provisioned-concurrency-config`exemple suivant affiche les détails de la configuration de simultanéité provisionnée pour l'`BLUE`alias de la fonction spécifiée.

```
aws lambda get-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE
```

Sortie :

```
{
    "RequestedProvisionedConcurrentExecutions": 100,
    "AvailableProvisionedConcurrentExecutions": 100,
    "AllocatedProvisionedConcurrentExecutions": 100,
    "Status": "READY",
    "LastModified": "2019-12-31T20:28:49+0000"
}
```

- Pour API plus de détails, voir GetProvisionedConcurrencyConfigla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple obtient la configuration de simultanéité provisionnée pour l'alias spécifié de la fonction Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

Sortie :

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                   : IN_PROGRESS
StatusReason                             :
```

- Pour API plus de détails, consultez la section GetProvisionedConcurrencyConfigRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`Invoke`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`Invoke`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

- Pour API plus de détails, voir Invoke in AWS SDK for .NET APIReference.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::InvokeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetLogType(logType);
        std::shared_ptr<Aws::IOStream> payload =
Aws::MakeShared<Aws::StringStream>(
                "FunctionTest");
        *payload << jsonPayload.View().WriteReadable();
        request.SetBody(payload);
        request.SetContentType("application/json");
        Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

        if (outcome.IsSuccess()) {
            invokeResult = std::move(outcome.GetResult());
            result = true;
            break;
        }

        else {
            std::cerr << "Error with Lambda::InvokeRequest. "
                      << outcome.GetError().GetMessage()
```

```
                        << std::endl;
            break;
        }
```

- Pour API plus de détails, voir Invoke in AWS SDK for C++ APIReference.

## CLI

### AWS CLI

Exemple 1 : pour invoquer une fonction Lambda de manière synchrone

L'exemple `invoke` suivant invoque la fonction `my-function` de manière synchrone. L'`cli-binary-format`option est obligatoire si vous utilisez AWS CLI la version 2. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur de l'interface de ligne de AWS commande.

```
aws lambda invoke \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }' \
    response.json
```

Sortie :

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Pour plus d'informations, consultez Invocation synchrone dans le Guide du développeur AWS .

Exemple 2 : pour invoquer une fonction Lambda de manière asynchrone

L'exemple `invoke` suivant invoque la fonction `my-function` de manière asynchrone. L'`cli-binary-format`option est obligatoire si vous utilisez AWS CLI la version 2. Pour plus d'informations, consultez les options de ligne de commande globales AWS CLI prises en charge dans le Guide de l'utilisateur de l'interface de ligne de AWS commande.

```
aws lambda invoke \
```

```
    --function-name my-function \
    --invocation-type Event \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }' \
    response.json
```

Sortie :

```
{
    "StatusCode": 202
}
```

Pour plus d'informations, consultez Invocation asynchrone dans le Guide du développeur AWS .

- Pour API plus de détails, voir Invoke dans AWS CLI la référence des commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
```

```
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}
```

- Pour API plus de détails, voir Invoke in AWS SDK for Go APIReference.

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
import org.json.JSONObject;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
```

```
import software.amazon.awssdk.services.lambda.model.LambdaException;

public class LambdaInvoke {

    /*
     * Function names appear as
     * arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
     * you can retrieve the value by looking at the function in the AWS Console
     *
     * Also, set up your development environment, including your credentials.
     *
     * For information, see this documentation topic:
     *
     * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.
     * html
     */

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <functionName>\s

                Where:
                    functionName - The name of the Lambda function\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        invokeFunction(awsLambda, functionName);
        awsLambda.close();
    }
```

```
    public static void invokeFunction(LambdaClient awsLambda, String
 functionName) {

        InvokeResponse res = null;
        try {
            // Need a SdkBytes instance for the payload.
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("inputValue", "2000");
            String json = jsonObj.toString();
            SdkBytes payload = SdkBytes.fromUtf8String(json);

            // Setup an InvokeRequest.
            InvokeRequest request = InvokeRequest.builder()
                    .functionName(functionName)
                    .payload(payload)
                    .build();

            res = awsLambda.invoke(request);
            String value = res.payload().asUtf8String();
            System.out.println(value);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Pour API plus de détails, voir Invoke in AWS SDK for Java 2.x APIReference.

## JavaScript

### SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const invoke = async (funcName, payload) => {
```

```
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Pour API plus de détails, voir Invoke in AWS SDK for JavaScript APIReference.

Kotlin

SDKpour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun invokeFunction(functionNameVal: String) {
    val json = """{"inputValue":"1000"}"""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            logType = LogType.Tail
            payload = byteArray
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("${res.payload?.toString(Charsets.UTF_8)}")
        println("The log result is ${res.logResult}")
    }
}
```

- Pour API plus de détails, voir Invoke in AWS SDKpour une API référence à Kotlin.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function invoke($functionName, $params, $logType = 'None')
    {
        return $this->lambdaClient->invoke([
            'FunctionName' => $functionName,
            'Payload' => json_encode($params),
            'LogType' => $logType,
        ]);
    }
```

- Pour API plus de détails, voir Invoke in AWS SDK for PHP APIReference.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
```

```
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
dict
                                is serialized to JSON before it is sent to
Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
included in
                        the response.
        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
        return response
```

- Pour API plus de détails, voir Invoke in AWS SDKfor Python (Boto3) Reference. API

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Invokes a Lambda function.
  # @param function_name [String] The name of the function to invoke.
  # @param payload [nil] Payload containing runtime parameters.
  # @return [Object] The response from the function invocation.
  def invoke_function(function_name, payload = nil)
    params = { function_name: function_name }
    params[:payload] = payload unless payload.nil?
    @lambda_client.invoke(params)
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error executing #{function_name}:\n
 #{e.message}")
  end
```

- Pour API plus de détails, voir Invoke in AWS SDK for Ruby APIReference.

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /** Invoke the lambda function using calculator InvokeArgs. */
   pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
 anyhow::Error> {
       info!(?args, "Invoking {}", self.lambda_name);
```

```
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
```

- Pour API plus de détails, voir Invoke in AWS SDKpour la API référence à Rust.

## SAP ABAP

### SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
          `{`  &&
            `"action": "increment",`  &&
```

```
            `"number": 10` &&
          `}`
        ).
        oo_result =  lo_lmd->invoke(                  " oo_result is returned for
testing purposes. "
                iv_functionname = iv_function_name
                iv_payload = lv_json
            ).
        MESSAGE 'Lambda function invoked.' TYPE 'I'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvrequestcontex.
        MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvalidzipfileex.
        MESSAGE 'The deployment package could not be unzipped.' TYPE 'E'.
      CATCH /aws1/cx_lmdrequesttoolargeex.
        MESSAGE 'Invoke request body JSON input limit was exceeded by the request
payload.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
      CATCH /aws1/cx_lmdunsuppedmediatyp00.
        MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
    ENDTRY.
```

- Pour API plus de détails, voir Invoke in AWS SDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`ListFunctions`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`ListFunctions`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Lambda::LambdaClient client(clientConfig);

   std::vector<Aws::String> functions;
   Aws::String marker;

   do {
       Aws::Lambda::Model::ListFunctionsRequest request;
       if (!marker.empty()) {
           request.SetMarker(marker);
       }

       Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
               request);

       if (outcome.IsSuccess()) {
           const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
           std::cout << result.GetFunctions().size()
                     << " lambda functions were retrieved." << std::endl;

           for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
               functions.push_back(functionConfiguration.GetFunctionName());
               std::cout << functions.size() << "  "
                         << functionConfiguration.GetDescription() << std::endl;
               std::cout << "   "
```

```
                            <<
 Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                    functionConfiguration.GetRuntime()) << ": "
                            << functionConfiguration.GetHandler()
                            << std::endl;
            }
            marker = result.GetNextMarker();
        }
        else {
            std::cerr << "Error with Lambda::ListFunctions. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for C++ APIRéférence.

CLI

AWS CLI

Pour récupérer la liste des fonctions Lambda

L'exemple `list-functions` suivant affiche une liste de toutes les fonctions pour l'utilisateur actuel.

```
aws lambda list-functions
```

Sortie :

```
{
    "Functions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
            "FunctionName": "helloworld",
            "MemorySize": 128,
            "RevisionId": "1718e831-badf-4253-9518-d0644210af7b",
```

```
            "CodeSize": 294,
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:helloworld",
            "Handler": "helloworld.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-
role-zgur6bf4",
            "Timeout": 3,
            "LastModified": "2023-09-23T18:32:33.857+0000",
            "Runtime": "nodejs18.x",
            "Description": ""
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2023-10-01T16:47:28.490+0000",
            "Runtime": "nodejs18.x",
            "Description": ""
        },
        {
            "Layers": [
                {
                    "CodeSize": 41784542,
                    "Arn": "arn:aws:lambda:us-
west-2:420165488524:layer:AWSLambda-Python37-SciPy1x:2"
                },
                {
```

```
                    "CodeSize": 4121,
                    "Arn": "arn:aws:lambda:us-
west-2:123456789012:layer:pythonLayer:1"
                }
            ],
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "ZQukCqxtkqFgyF2cU41Avj99TKQ/hNihPtDtRcc08mI=",
            "FunctionName": "my-python-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 128,
            "RevisionId": "80b4eabc-acf7-4ea8-919a-e874c213707d",
            "CodeSize": 299,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
python-function",
            "Handler": "lambda_function.lambda_handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/my-python-
function-role-z5g7dr6n",
            "Timeout": 3,
            "LastModified": "2023-10-01T19:40:41.643+0000",
            "Runtime": "python3.11",
            "Description": ""
        }
    ]
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir ListFunctionsla section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// ListFunctions lists up to maxItems functions for the account. This function
 uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Go APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for JavaScript APIRéférence.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function listFunctions($maxItems = 50, $marker = null)
    {
        if (is_null($marker)) {
            return $this->lambdaClient->listFunctions([
                'MaxItems' => $maxItems,
            ]);
        }
```

```
        return $this->lambdaClient->listFunctions([
            'Marker' => $marker,
            'MaxItems' => $maxItems,
        ]);
    }
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for PHP APIRéférence.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple affiche toutes les fonctions Lambda avec une taille de code triée

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
 RunTime, Timeout, CodeSize
```

Sortie :

```
FunctionName                                                    Runtime   Timeout
 CodeSize
------------                                                    -------   -------
 --------
test                                                            python2.7       3
   243
MylambdaFunction123                                             python3.8     600
   659
myfuncpython1                                                   python3.8     303
   675
```

- Pour API plus de détails, consultez la section ListFunctionsRéférence des AWS Tools for PowerShell applets de commande.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
                        print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
        except ClientError as err:
            logger.error(
                "Couldn't list functions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Pour API plus de détails, reportez-vous ListFunctionsà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response['functions'].each do |function|
        functions.append(function['function_name'])
      end
    end
    functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
  end
```

- Pour API plus de détails, voir ListFunctionsla section AWS SDK for Ruby APIRéférence.

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /** List all Lambda functions in the current Region. */
   pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
       info!("Listing lambda functions");
       self.lambda_client
           .list_functions()
           .send()
           .await
           .map_err(anyhow::Error::from)
   }
```

- Pour API plus de détails, reportez-vous ListFunctionsà la section AWS SDKpour la API référence à Rust.

SAP ABAP

SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       oo_result =  lo_lmd->listfunctions( ).       " oo_result is returned for
testing purposes. "
       DATA(lt_functions) = oo_result->get_functions( ).
```

```
        MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Pour API plus de détails, voir ListFunctions AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`ListProvisionedConcurrencyConfigs`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`ListProvisionedConcurrencyConfigs`.

CLI

AWS CLI

Pour obtenir une liste des configurations de simultanéité provisionnées

L'`list-provisioned-concurrency-configs`exemple suivant répertorie les configurations de simultanéité provisionnées pour la fonction spécifiée.

```
aws lambda list-provisioned-concurrency-configs \
    --function-name my-function
```

Sortie :

```
{
    "ProvisionedConcurrencyConfigs": [
        {
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:GREEN",
```

```
            "RequestedProvisionedConcurrentExecutions": 100,
            "AvailableProvisionedConcurrentExecutions": 100,
            "AllocatedProvisionedConcurrentExecutions": 100,
            "Status": "READY",
            "LastModified": "2019-12-31T20:29:00+0000"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:BLUE",
            "RequestedProvisionedConcurrentExecutions": 100,
            "AvailableProvisionedConcurrentExecutions": 100,
            "AllocatedProvisionedConcurrentExecutions": 100,
            "Status": "READY",
            "LastModified": "2019-12-31T20:28:49+0000"
        }
    ]
}
```

- Pour API plus de détails, voir ListProvisionedConcurrencyConfigsla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple extrait la liste des configurations de simultanéité provisionnées pour une fonction Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Pour API plus de détails, consultez la section ListProvisionedConcurrencyConfigsRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`ListTags`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`ListTags`.

CLI

AWS CLI

Pour récupérer la liste des balises d'une fonction Lambda

L'`list-tags`exemple suivant affiche les balises associées à la fonction `my-function` Lambda.

```
aws lambda list-tags \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Sortie :

```
{
    "Tags": {
        "Category": "Web Tools",
        "Department": "Sales"
    }
}
```

Pour plus d'informations, consultez la section Marquage des fonctions Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir ListTagsla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : récupère les balises et leurs valeurs actuellement définies sur la fonction spécifiée.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Sortie :

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
```

```
Washington Olympia
```

- Pour API plus de détails, consultez la section ListTagsRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`ListVersionsByFunction`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`ListVersionsByFunction`.

CLI

AWS CLI

Pour récupérer la liste des versions d'une fonction

L'`list-versions-by-function`exemple suivant affiche la liste des versions de la fonction `my-function` Lambda.

```
aws lambda list-versions-by-function \
    --function-name my-function
```

Sortie :

```
{
    "Versions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
```

```
            "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:$LATEST",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-10-01T16:47:28.490+0000",
            "Runtime": "nodejs10.x",
            "Description": ""
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "1",
            "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJmlKidWaaCgk=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "949c8914-012e-4795-998c-e467121951b1",
            "CodeSize": 304,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:1",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-09-26T20:28:40.438+0000",
            "Runtime": "nodejs10.x",
            "Description": "new version"
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "2",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
```

```
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "cd669f21-0f3d-4e1c-9566-948837f2e2ea",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-10-01T16:47:28.490+0000",
            "Runtime": "nodejs10.x",
            "Description": "newer version"
        }
    ]
}
```

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir ListVersionsByFunctionla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple renvoie la liste des configurations spécifiques à chaque version de la fonction Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Sortie :

```
FunctionName        Runtime   MemorySize Timeout CodeSize LastModified
      RoleName
------------        -------   ---------- ------- -------- ------------
      --------
```

```
MylambdaFunction123 python3.8        128     600      659
 2020-01-10T03:20:56.390+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:19:02.238+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:39:36.779+0000 lambda
MylambdaFunction123 python3.8        128     600     1426
 2019-12-25T09:52:59.872+0000 lambda
```

- Pour API plus de détails, consultez la section ListVersionsByFunctionRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`PublishVersion`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`PublishVersion`.

CLI

AWS CLI

Pour publier une nouvelle version d'une fonction

L'`publish-version`exemple suivant publie une nouvelle version de la fonction `my-function` Lambda.

```
aws lambda publish-version \
    --function-name my-function
```

Sortie :

```
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
    "FunctionName": "my-function",
    "CodeSize": 294,
```

```
    "RevisionId": "f31d3d39-cc63-4520-97d4-43cd44c94c20",
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:3",
    "Version": "2",
    "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
zgur6bf4",
    "Timeout": 3,
    "LastModified": "2019-09-23T18:32:33.857+0000",
    "Handler": "my-function.handler",
    "Runtime": "nodejs10.x",
    "Description": ""
}
```

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir PublishVersionla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple crée une version pour l'instantané existant du code de fonction Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Pour API plus de détails, consultez la section PublishVersionRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`PutFunctionConcurrency`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`PutFunctionConcurrency`.

CLI

AWS CLI

Pour configurer une limite de simultanéité réservée pour une fonction

L'`put-function-concurrency`exemple suivant configure 100 exécutions simultanées réservées pour la `my-function` fonction.

```
aws lambda put-function-concurrency \
    --function-name  my-function  \
    --reserved-concurrent-executions 100
```

Sortie :

```
{
    "ReservedConcurrentExecutions": 100
}
```

Pour plus d'informations, consultez la section Réserver la simultanéité pour une fonction Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir PutFunctionConcurrencyla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple applique les paramètres de simultanéité pour la fonction dans son ensemble.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Pour API plus de détails, consultez la section PutFunctionConcurrencyRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`PutProvisionedConcurrencyConfig`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`PutProvisionedConcurrencyConfig`.

CLI

AWS CLI

Pour allouer la simultanéité provisionnée

L'`put-provisioned-concurrency-config`exemple suivant alloue 100 accès simultanés à l'`BLUE`alias de la fonction spécifiée.

```
aws lambda put-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE \
    --provisioned-concurrent-executions 100
```

Sortie :

```
{
    "Requested ProvisionedConcurrentExecutions": 100,
    "Allocated ProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

- Pour API plus de détails, voir PutProvisionedConcurrencyConfigla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : cet exemple ajoute une configuration de simultanéité provisionnée à l'alias d'une fonction

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Pour API plus de détails, consultez la section PutProvisionedConcurrencyConfigRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`RemovePermission`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`RemovePermission`.

CLI

AWS CLI

Pour supprimer les autorisations d'une fonction Lambda existante

L'`remove-permission`exemple suivant supprime l'autorisation d'invoquer une fonction nomméemy-function.

```
aws lambda remove-permission \
    --function-name my-function \
    --statement-id sns
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Utilisation de politiques basées sur les ressources pour Lambda AWS dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir RemovePermissionla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple supprime la politique de fonction pour la fonction Lambda spécifiée StatementId .

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Pour API plus de détails, consultez la section RemovePermissionRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`TagResource`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`TagResource`.

CLI

AWS CLI

Pour ajouter des balises à une fonction Lambda existante

L'`tag-resource`exemple suivant ajoute une balise avec le nom de la clé `DEPARTMENT` et une valeur de `Department A` à la fonction Lambda spécifiée.

```
aws lambda tag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tags "DEPARTMENT=Department A"
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des fonctions Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir TagResourcela section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : ajoute les trois balises (Washington, Oregon et Californie) et leurs valeurs associées à la fonction spécifiée identifiée par sonARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Pour API plus de détails, consultez la section TagResourceRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`UntagResource`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`UntagResource`.

CLI

AWS CLI

Pour supprimer des balises d'une fonction Lambda existante

L'`untag-resource`exemple suivant supprime la balise avec la `DEPARTMENT` balise key name de la fonction `my-function` Lambda.

```
aws lambda untag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tag-keys DEPARTMENT
```

Cette commande ne produit aucun résultat.

Pour plus d'informations, consultez la section Marquage des fonctions Lambda dans le guide du développeur AWS Lambda.

- Pour API plus de détails, voir UntagResourcela section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Supprime les balises fournies d'une fonction. L'applet de commande demandera une confirmation avant de continuer, sauf si le commutateur -Force est spécifié. Un seul appel est envoyé au service pour retirer les tags.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Exemple 2 : Supprime les balises fournies d'une fonction. L'applet de commande demandera une confirmation avant de continuer, sauf si le commutateur -Force est spécifié. Une fois que l'appel au service est effectué, selon le tag fourni.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Pour API plus de détails, consultez la section UntagResourceRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`UpdateAlias`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`UpdateAlias`.

CLI

AWS CLI

Pour mettre à jour un alias de fonction

L'`update-alias`exemple suivant met à jour l'alias nommé `LIVE` pour qu'il pointe vers la version 3 de la fonction `my-function` Lambda.

```
aws lambda update-alias \
```

```
    --function-name my-function \
    --function-version 3 \
    --name LIVE
```

Sortie :

```
{
    "FunctionVersion": "3",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",
    "Description": "alias for live version of function"
}
```

Pour plus d'informations, consultez la section Configuration des alias de fonction AWS Lambda dans le guide du développeur Lambda AWS .

- Pour API plus de détails, voir UpdateAliasla section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple met à jour la configuration d'un alias de fonction Lambda existant. Il met à jour la RoutingConfiguration valeur pour transférer 60 % (0,6) du trafic vers la version 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Pour API plus de détails, consultez la section UpdateAliasRéférence des AWS Tools for PowerShell applets de commande.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`UpdateFunctionCode`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`UpdateFunctionCode`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
```

```
        Console.WriteLine($"The Function was last modified at
 {response.LastModified}.");
    }
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
 (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::UpdateFunctionCodeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteLambdaFunction(client);
```

```
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
                Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                                       buffer.str().length()));
        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
std::endl;
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for C++ APIRéférence.

## CLI

### AWS CLI

Pour mettre à jour le code d'une fonction Lambda

L'`update-function-code`exemple suivant remplace le code de la version non publiée ($LATEST) de la `my-function` fonction par le contenu du fichier zip spécifié.

```
aws lambda update-function-code \
    --function-name  my-function \
    --zip-file fileb://my-function.zip
```

Sortie :

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "MemorySize": 256,
    "Version": "$LATEST",
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Handler": "index.handler"
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir UpdateFunctionCodela section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in
 the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
 functionName string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
 functionName, err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for Go APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for JavaScript APIRéférence.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
    {
        return $this->lambdaClient->updateFunctionCode([
            'FunctionName' => $functionName,
            'S3Bucket' => $s3Bucket,
            'S3Key' => $s3Key,
        ]);
    }
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for PHP APIRéférence.

PowerShell

Outils pour PowerShell

Exemple 1 : met à jour la fonction nommée MyFunction « » avec le nouveau contenu contenu dans le fichier zip spécifié. Pour un C#. NETFonction Lambda principale : le fichier zip doit contenir l'assemblage compilé.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Exemple 2 : Cet exemple est similaire au précédent mais utilise un objet Amazon S3 contenant le code mis à jour pour mettre à jour la fonction.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Pour API plus de détails, consultez la section UpdateFunctionCodeRéférence des AWS Tools for PowerShell applets de commande.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
 contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
 in
                                   .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return response
```

- Pour API plus de détails, reportez-vous UpdateFunctionCodeà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the code for a Lambda function by submitting a .zip archive that
 contains
  # the code for the function.
  #
  # @param function_name: The name of the function to update.
  # @param deployment_package: The function code to update, packaged as bytes in
  #                            .zip format.
  # @return: Data about the update, including the status.
  def update_function_code(function_name, deployment_package)
    @lambda_client.update_function_code(
```

```
      function_name: function_name,
      zip_file: deployment_package
    )
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
```

- Pour API plus de détails, voir UpdateFunctionCodela section AWS SDK for Ruby APIRéférence.

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
```

```
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
```

```
    }
```

- Pour API plus de détails, reportez-vous UpdateFunctionCodeà la section AWS SDKpour la API référence à Rust.

## SAP ABAP

### SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       oo_result = lo_lmd->updatefunctioncode(     " oo_result is returned for
testing purposes. "
            iv_functionname = iv_function_name
            iv_zipfile = io_zip_file
         ).

       MESSAGE 'Lambda function code updated.' TYPE 'I'.
     CATCH /aws1/cx_lmdcodesigningcfgno00.
       MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdcodestorageexcdex.
       MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
     CATCH /aws1/cx_lmdcodeverification00.
       MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvalidcodesigex.
       MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvparamvalueex.
       MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
     CATCH /aws1/cx_lmdresourceconflictex.
       MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
     CATCH /aws1/cx_lmdresourcenotfoundex.
       MESSAGE 'The requested resource does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdserviceexception.
```

```
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
     CATCH /aws1/cx_lmdtoomanyrequestsex.
       MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
   ENDTRY.
```

- Pour API plus de détails, voir UpdateFunctionCode AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## À utiliser **`UpdateFunctionConfiguration`** avec un AWS SDK ou CLI

Les exemples de code suivants montrent comment utiliser`UpdateFunctionConfiguration`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Mise en route avec des fonctions

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   /// <summary>
   /// Update the code of a Lambda function.
   /// </summary>
   /// <param name="functionName">The name of the function to update.</param>
   /// <param name="functionHandler">The code that performs the function's
actions.</param>
```

```
    /// <param name="environmentVariables">A dictionary of environment
variables.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for .NET APIRéférence.

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
```

```
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        Aws::Lambda::Model::Environment environment;
        environment.AddVariables("LOG_LEVEL", "DEBUG");
        request.SetEnvironment(environment);

        Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda configuration was successfully updated."
                      << std::endl;
            break;
        }

        else {
            std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for C++ APIRéférence.

CLI

AWS CLI

Pour modifier la configuration d'une fonction

L'`update-function-configuration`exemple suivant modifie la taille de la mémoire à 256 Mo pour la version non publiée ($LATEST) de la `my-function` fonction.

```
aws lambda update-function-configuration \
```

```
    --function-name  my-function \
    --memory-size 256
```

Sortie :

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "MemorySize": 256,
    "Version": "$LATEST",
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Handler": "index.handler"
}
```

Pour plus d'informations, consultez Configuration des options de fonction Lambda AWS dans le Guide du développeur AWS .

- Pour API plus de détails, voir UpdateFunctionConfigurationla section Référence des AWS CLI commandes.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionConfiguration updates a map of environment variables configured
 for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v",
 functionName, err)
 }
}
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for Go APIRéférence.

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for JavaScript APIRéférence.

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
    {
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
```

```
            'Environment' => $environment,
        ]);
    }
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for PHP APIRéférence.

PowerShell

Outils pour PowerShell

Exemple 1 : Cet exemple met à jour la configuration de la fonction Lambda existante

```
Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler
 "lambda_function.launch_instance" -Timeout 600 -Environment_Variable
 @{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
 123456789101:MyfirstTopic
```

- Pour API plus de détails, consultez la section UpdateFunctionConfigurationRéférence des AWS Tools for PowerShell applets de commande.

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def update_function_configuration(self, function_name, env_vars):
```

```
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function configuration %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Pour API plus de détails, reportez-vous UpdateFunctionConfigurationà la section AWS SDKrelative à la référence Python (Boto3). API

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
```

```
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the environment variables for a Lambda function.
  # @param function_name: The name of the function to update.
  # @param log_level: The log level of the function.
  # @return: Data about the update, including the status.
  def update_function_configuration(function_name, log_level)
    @lambda_client.update_function_configuration({
                                                   function_name: function_name,
                                                   environment: {
                                                     variables: {
                                                       'LOG_LEVEL' => log_level
                                                     }
                                                   }
                                                 })
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
  end
```

- Pour API plus de détails, voir UpdateFunctionConfigurationla section AWS SDK for Ruby APIRéférence.

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(updated)
}
```

- Pour API plus de détails, reportez-vous UpdateFunctionConfigurationà la section AWS SDKpour la API référence à Rust.

## SAP ABAP

### SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       oo_result = lo_lmd->updatefunctionconfiguration(     " oo_result is
returned for testing purposes. "
             iv_functionname = iv_function_name
             iv_runtime = iv_runtime
             iv_description  = 'Updated Lambda function'
             iv_memorysize  = iv_memory_size
         ).

       MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
     CATCH /aws1/cx_lmdcodesigningcfgno00.
       MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdcodeverification00.
       MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvalidcodesigex.
       MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvparamvalueex.
       MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
     CATCH /aws1/cx_lmdresourceconflictex.
       MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
     CATCH /aws1/cx_lmdresourcenotfoundex.
       MESSAGE 'The requested resource does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdserviceexception.
       MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
     CATCH /aws1/cx_lmdtoomanyrequestsex.
       MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
   ENDTRY.
```

- Pour API plus de détails, voir UpdateFunctionConfiguration AWSSDKpour SAP ABAP API référence.

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Scénarios d'utilisation de Lambda AWS SDKs

Les exemples de code suivants vous montrent comment implémenter des scénarios courants dans Lambda avec. AWS SDKs Ces scénarios vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions dans Lambda ou en les combinant avec d'autres. Services AWS Chaque scénario inclut un lien vers le code source complet, où vous trouverez des instructions sur la façon de configurer et d'exécuter le code.

Les scénarios ciblent un niveau d'expérience intermédiaire pour vous aider à comprendre les actions de service dans leur contexte.

Exemples

- Confirmez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK
- Migrez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK
- Créez une API passerelle REST API pour suivre les données COVID -19
- Création d'une bibliothèque de prêt REST API
- Créer une application de messagerie avec Step Functions
- Création d’une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l’aide d’étiquettes
- Créez une application de chat Websocket avec Gateway API
- Créez une application qui analyse les commentaires des clients et synthétise le son
- Commencez à créer et à invoquer des fonctions Lambda à l'aide d'un AWS SDK
- Invoquer une fonction Lambda à partir d’un navigateur
- Transformez les données de votre application avec S3 Object Lambda
- Utiliser API Gateway pour appeler une fonction Lambda
- Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda

- Utilisent des événements planifiés pour appeler une fonction Lambda
- Rédigez des données d'activité personnalisées à l'aide d'une fonction Lambda après l'authentification de l'utilisateur Amazon Cognito à l'aide d'un AWS SDK

## Confirmez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK

L'exemple de code suivant montre comment confirmer automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda.

- Configurez un groupe d'utilisateurs pour appeler une fonction Lambda pour le `PreSignUp` déclencheur.
- Inscrivez un utilisateur avec Amazon Cognito.
- La fonction Lambda analyse une table DynamoDB et confirme automatiquement les utilisateurs connus.
- Connectez-vous en tant que nouvel utilisateur, puis nettoyez les ressources.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
// AutoConfirm separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type AutoConfirm struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
```

```
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) AutoConfirm {
 scenario := AutoConfirm{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
 PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
 Cognito.\n" +
  "This trigger happens when a user signs up, and lets your function take action
 before the main Cognito\n" +
  "sign up processing occurs.\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
 trigger.\n",
  functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you
 specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
 usersTable string) (string, string) {
 log.Println("Let's sign up a user to your Cognito user pool. When the user's
 email matches an email in the\n" +
```

```
  "DynamoDB known users table, it is automatically verified and the user is
 confirmed.")

 knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
 if err != nil {
  panic(err)
 }
 userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
 knownUsers.UserNameList())
 user := knownUsers.Users[userChoice]

 var signedUp bool
 var userConfirmed bool
 password := runner.questioner.AskPassword("Enter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !signedUp {
  log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
 user.UserEmail)
  userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
 password, user.UserEmail)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("Enter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   signedUp = true
  }
 }
 log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string,
 userName string, password string) string {
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 log.Printf("Let's sign in as %v...\n", userName)
```

```
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 log.Println(strings.Repeat("-", 88))
 return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

 runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
```

```
 log.Println(strings.Repeat("-", 88))
}
```

Manipulez le `PreSignUp` déclencheur avec une fonction Lambda.

```
const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
 error) {
 log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "PreSignUp_SignUp" {
  // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
 response from this handler.
  return event, nil
 }
```

```
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserEmail: event.Request.UserAttributes["email"],
 }
 log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
 output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key:       user.GetKey(),
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Error looking up email %v.\n", user.UserEmail)
  return event, err
 }
 if output.Item == nil {
  log.Printf("Email %v not found. Email verification is required.\n",
 user.UserEmail)
  return event, err
 }

 err = attributevalue.UnmarshalMap(output.Item, &user)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
  return event, err
 }

 if user.UserName != event.UserName {
  log.Printf("UserEmail %v found, but stored UserName '%v' does not match
 supplied UserName '%v'. Verification is required.\n",
   user.UserEmail, user.UserName, event.UserName)
 } else {
  log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
  event.Response.AutoConfirmUser = true
  event.Response.AutoVerifyEmail = true
 }

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
```

```
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Créez une structure qui exécute des tâches courantes.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
```

```
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
```

```
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Créez une structure qui englobe les actions Amazon Cognito.

```
type CognitoActions struct {
```

```
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
```

```
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
```

```
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}
```

```
// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
```

```
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
```

```
}
```

Créez une structure qui enveloppe les actions DynamoDB.

```
// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}
```

```
// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
```

```
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Créez une structure qui englobe les actions CloudWatch Logs.

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
```

```
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Créez une structure qui englobe les actions. AWS CloudFormation

```
// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
```

```
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Nettoyez les ressources.

```
// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
```

```
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
     "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Go APIRéférence.

- DeleteUser
- InitiateAuth
- SignUp
- UpdateUserPool

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Migrez automatiquement les utilisateurs connus d'Amazon Cognito à l'aide d'une fonction Lambda à l'aide d'un AWS SDK

L'exemple de code suivant montre comment migrer automatiquement des utilisateurs Amazon Cognito connus à l'aide d'une fonction Lambda.

- Configurez un groupe d'utilisateurs pour appeler une fonction Lambda pour le `MigrateUser` déclencheur.
- Connectez-vous à Amazon Cognito avec un nom d'utilisateur et une adresse e-mail qui ne figurent pas dans le groupe d'utilisateurs.
- La fonction Lambda analyse une table DynamoDB et migre automatiquement les utilisateurs connus vers le groupe d'utilisateurs.
- Exécutez le flux de mots de passe oubliés pour réinitialiser le mot de passe de l'utilisateur migré.
- Connectez-vous en tant que nouvel utilisateur, puis nettoyez les ressources.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type MigrateUser struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) MigrateUser {
 scenario := MigrateUser{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
 MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
```

```
 log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
 Cognito.\n" +
  "This trigger happens when an unknown user signs in, and lets your function
 take action before Cognito\n" +
  "rejects the user.\n\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
 trigger.\n",
  functionArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to
 Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
 clientId string) (bool, actions.User) {
 log.Println("Let's sign in a user to your Cognito user pool. When the username
 and email matches an entry in the\n" +
  "DynamoDB known users table, the email is automatically verified and the user
 is migrated to the Cognito user pool.")

 user := actions.User{}
 user.UserName = runner.questioner.Ask("\nEnter a username:")
 user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This
 email will be used to confirm user migration\n" +
  "during this example:")

 runner.helper.AddKnownUser(ctx, usersTable, user)

 var err error
 var resetRequired *types.PasswordResetRequiredException
 var authResult *types.AuthenticationResultType
 signedIn := false
 for !signedIn && resetRequired == nil {
  log.Printf("Signing in to Cognito as user '%v'. The expected result is a
 PasswordResetRequiredException.\n\n", user.UserName)
  authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
```

```
  if err != nil {
   if errors.As(err, &resetRequired) {
    log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
 DynamoDB known users table.\n"+
     "User migration is started and a password reset is required.",
 user.UserName)
   } else {
    panic(err)
   }
  } else {
   log.Printf("User '%v' successfully signed in. This is unexpected and probably
 means you have not\n"+
    "cleaned up a previous run of this scenario, so the user exist in the Cognito
 user pool.\n"+
    "You can continue this example and select to clean up resources, or manually
 remove\n"+
    "the user from your user pool and try again.", user.UserName)
   runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
   signedIn = true
  }
 }

 log.Println(strings.Repeat("-", 88))
 return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,
 user actions.User) {
 wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
 to Cognito, you must be able to receive a confirmation\n"+
  "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
 if !wantCode {
  log.Println("To complete this example and successfully migrate a user to
 Cognito, you must enter an email\n" +
   "you own that can receive a confirmation code.")
  return
 }
 codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
 user.UserName)
 if err != nil {
  panic(err)
 }
```

```
 log.Printf("\nA confirmation code has been sent to %v.",
 *codeDelivery.Destination)
 code := runner.questioner.Ask("Check your email and enter it here:")

 confirmed := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !confirmed {
  log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
 user.UserName, password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   confirmed = true
  }
 }
 log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
 log.Println("Signing in with your username and password...")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
 password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)

 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
```

```
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
  runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
 }

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Manipulez le `MigrateUser` déclencheur avec une fonction Lambda.

```
const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
```

```
 UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsMigrateUser)
 (events.CognitoEventUserPoolsMigrateUser, error) {
 log.Printf("Received migrate trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 if event.TriggerSource != "UserMigration_Authentication" {
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName: event.UserName,
 }
 log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
 filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
 expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
 if err != nil {
  log.Printf("Error building expression to query for user '%v'.\n",
 user.UserName)
  return event, err
 }
 output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName:                 aws.String(tableName),
  FilterExpression:          expr.Filter(),
  ExpressionAttributeNames:  expr.Names(),
  ExpressionAttributeValues: expr.Values(),
 })
 if err != nil {
  log.Printf("Error looking up user '%v'.\n", user.UserName)
  return event, err
 }
 if len(output.Items) == 0 {
  log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
  return event, err
 }
```

```
 var users []UserInfo
 err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
  return event, err
 }

 user = users[0]
 log.Printf("UserName '%v' found with email %v. User is migrated and must reset
 password.\n", user.UserName, user.UserEmail)
 event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
 map[string]string{
  "email":          user.UserEmail,
  "email_verified": "true", // email_verified is required for the forgot password
 flow.
 }
 event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
 "RESET_REQUIRED"
 event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Créez une structure qui exécute des tâches courantes.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
```

```
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
```

```
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
```

```
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Créez une structure qui englobe les actions Amazon Cognito.

```
type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)
```

```
type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
```

```
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
```

```
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
```

```
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
```

```
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Créez une structure qui enveloppe les actions DynamoDB.

```


// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
```

```
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
```

```
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
```

```
 return err
}
```

Créez une structure qui englobe les actions CloudWatch Logs.

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
```

```
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Créez une structure qui englobe les actions. AWS CloudFormation

```
// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Nettoyez les ressources.

```
// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
```

```
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Go APIRéférence.
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Créez une API passerelle REST API pour suivre les données COVID -19

L'exemple de code suivant montre comment créer un système REST API qui simule un système permettant de suivre les cas quotidiens de COVID -19 aux États-Unis, à l'aide de données fictives.

Python

SDKpour Python (Boto3)

Montre comment utiliser AWS Chalice avec le AWS SDK for Python (Boto3) pour créer un serveur sans serveur REST API utilisant Amazon API Gateway et Amazon DynamoDB. AWS Lambda Il REST API simule un système qui suit les cas quotidiens de COVID -19 aux États-Unis, à l'aide de données fictives. Découvrez comment :

- Utilisez AWS Chalice pour définir des routes dans les fonctions Lambda appelées pour gérer les REST demandes qui passent par Gateway. API
- Utilisez les fonctions Lambda pour récupérer et stocker des données dans une table DynamoDB afin de répondre aux demandes. REST
- Définissez la structure des tables et les ressources des rôles de sécurité dans un AWS CloudFormation modèle.
- Utilisez AWS Chalice CloudFormation pour empaqueter et déployer toutes les ressources nécessaires.
- CloudFormation À utiliser pour nettoyer toutes les ressources créées.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- APIPasserelle
- AWS CloudFormation
- DynamoDB
- Lambda

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Création d'une bibliothèque de prêt REST API

L'exemple de code suivant montre comment créer une bibliothèque de prêt dans laquelle les clients peuvent emprunter et retourner des livres à l'aide d'une base de données REST API soutenue par Amazon Aurora.

Python

SDKpour Python (Boto3)

Montre comment utiliser le AWS SDK for Python (Boto3) avec Amazon Relational Database Service API (RDSAmazon) et AWS Chalice pour créer REST API une base de données basée sur une base de données Amazon Aurora. Le service Web est entièrement sans serveur et représente une bibliothèque de prêt simple où les clients peuvent emprunter et retourner des livres. Découvrez comment :

- Créer et gérer un cluster de bases de données Aurora sans serveur.
- AWS Secrets Manager À utiliser pour gérer les informations d'identification de base de données.
- Implémentez une couche de stockage de données qui utilise Amazon RDS pour déplacer les données dans et hors de la base de données.
- Utilisez AWS Chalice pour déployer un serveur sans serveur REST API sur Amazon API Gateway et. AWS Lambda
- Utiliser le package Requests (Requêtes) pour envoyer des requêtes au service web.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- APIPasserelle
- Aurora
- Lambda
- Secrets Manager

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Créer une application de messagerie avec Step Functions

L'exemple de code suivant montre comment créer une application de AWS Step Functions messagerie qui extrait les enregistrements de messages d'une table de base de données.

Python

SDKpour Python (Boto3)

Montre comment utiliser le AWS SDK for Python (Boto3) with AWS Step Functions pour créer une application de messagerie qui récupère les enregistrements de messages d'une table Amazon DynamoDB et les envoie via Amazon Simple Queue Service (Amazon). SQS La machine d'état intègre une AWS Lambda fonction permettant de scanner la base de données à la recherche de messages non envoyés.

- Créez une machine d’état qui extrait et met à jour des enregistrements de message d’une table Amazon DynamoDB.
- Mettez à jour la définition de la machine à états pour envoyer également des messages à Amazon Simple Queue Service (AmazonSQS).
- Démarrez et arrêtez les exécutions de la machine.
- Connectez-vous à Lambda, DynamoDB et SQS Amazon depuis une machine à états en utilisant des intégrations de services.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Création d'une application de gestion des ressources photographiques permettant aux utilisateurs de gérer les photos à l'aide d'étiquettes

Les exemples de code suivants montrent comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

.NET

AWS SDK for .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDKpour C++

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDKpour Java 2.x

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDKpour JavaScript (v3)

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDKpour Kotlin

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK pour PHP

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDKpour Rust

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur AWS Community.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Créez une application de chat Websocket avec Gateway API

L'exemple de code suivant montre comment créer une application de chat desservie par un websocket API basé sur Amazon API Gateway.

Python

SDKpour Python (Boto3)

Montre comment utiliser Amazon API Gateway V2 pour créer un websocket API qui s'intègre à Amazon AWS Lambda DynamoDB. AWS SDK for Python (Boto3)

- Créez un websocket API servi par API Gateway.
- Définissez un gestionnaire Lambda qui stocke les connexions dans DynamoDB et publie des messages pour d’autres participants au chat.
- Connectez-vous à l’application de chat Websocket et envoyez des messages avec le package Websockets.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Créez une application qui analyse les commentaires des clients et synthétise le son

Les exemples de code suivants montrent comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

.NET

AWS SDK for .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDKpour Java 2.x

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les

commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDKpour JavaScript (v3)

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub. Les extraits suivants montrent comment le AWS SDK for JavaScript est utilisé dans les fonctions Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";
```

```
/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
```

```
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});
```

```
  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDKpour Ruby

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend

- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Commencez à créer et à invoquer des fonctions Lambda à l'aide d'un AWS SDK

Les exemples de code suivants montrent comment :

- Créez un IAM rôle et une fonction Lambda, puis téléchargez le code du gestionnaire.
- Invoquer la fonction avec un seul paramètre et obtenir des résultats.
- Mettre à jour le code de la fonction et configurer avec une variable d'environnement.
- Invoquer la fonction avec de nouveaux paramètres et obtenir des résultats. Afficher le journal d'exécution renvoyé.
- Répertorier les fonctions pour votre compte, puis nettoyer les ressources.

Pour plus d'informations, consultez Créer une fonction Lambda à l'aide de la console.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez des méthodes qui exécutent des actions Lambda.

```
namespace LambdaActions;
```

```
using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
```

```
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
 _lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }


    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }


    /// <summary>
```

```
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }



    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

```
    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }


    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
```

```
        Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
    }


    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
actions.</param>
    /// <param name="environmentVariables">A dictionary of environment
variables.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


}
```

Créer une fonction qui exécute le scénario.

```
global using System.Threading.Tasks;
```

```
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonLambda>()
            .AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<LambdaWrapper>()
            .AddTransient<LambdaRoleWrapper>()
            .AddTransient<UIWrapper>()
        )
        .Build();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
```

```
        .Build();


        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LambdaBasics>();

        var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
        var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        string functionName = configuration["FunctionName"]!;
        string roleName = configuration["RoleName"]!;
        string policyDocument = "{" +
            " \"Version\": \"2012-10-17\"," +
            " \"Statement\": [ " +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Principal\": {" +
            "            \"Service\": \"lambda.amazonaws.com\" " +
            "    }," +
            "        \"Action\": \"sts:AssumeRole\" " +
            "    }" +
            "]" +
        "}";

        var incrementHandler = configuration["IncrementHandler"];
        var calculatorHandler = configuration["CalculatorHandler"];
        var bucketName = configuration["BucketName"];
        var incrementKey = configuration["IncrementKey"];
        var calculatorKey = configuration["CalculatorKey"];
        var policyArn = configuration["PolicyArn"];

        uiWrapper.DisplayLambdaBasicsOverview();

        // Create the policy to use with the AWS Lambda functions and then attach
the
        // policy to a new role.
        var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

        Console.WriteLine("Waiting for role to become active.");
        uiWrapper.WaitABit(15, "Wait until the role is active before trying to
use it.");
```

```
        // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
        var success = await
lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn, roleName);
        uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to
the role.");

        // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
        // (Amazon S3) bucket.
        uiWrapper.DisplayTitle("Create Lambda Function");
        Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
        var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
            functionName,
            bucketName,
            incrementKey,
            roleArn,
            incrementHandler);

        Console.WriteLine("Waiting for the new function to be available.");
        Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

        // Get the Lambda function.
        Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
        FunctionConfiguration config;
        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.State != State.Active);

        Console.WriteLine($"\nThe function, {functionName} has been created.");
        Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

        uiWrapper.PressEnter();

        // List the Lambda functions.
        uiWrapper.DisplayTitle("Listing all Lambda functions.");
        var functions = await lambdaWrapper.ListFunctionsAsync();
        DisplayFunctionList(functions);
```

```
        uiWrapper.DisplayTitle("Invoke increment function");
        Console.WriteLine("Now that it has been created, invoke the Lambda
increment function.");
        string? value;
        do
        {
            Console.Write("Enter a value to increment: ");
            value = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(value));

        string functionParameters = "{" +
            "\"action\": \"increment\", " +
            "\"x\": \"" + value + "\"" +
        "}";
        var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"{value} + 1 = {answer}.");

        uiWrapper.DisplayTitle("Update function");
        Console.WriteLine("Now update the Lambda function code.");
        await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,
            new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        uiWrapper.DisplayTitle("Call updated function");
```

```
        Console.WriteLine("Now call the updated function...");

        bool done = false;

        do
        {
            string? opSelected;

            Console.WriteLine("Select the operation to perform:");
            Console.WriteLine("\t1. add");
            Console.WriteLine("\t2. subtract");
            Console.WriteLine("\t3. multiply");
            Console.WriteLine("\t4. divide");
            Console.WriteLine("\tOr enter \"q\" to quit.");
            Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the
operation you want to perform: ");
            do
            {
                Console.Write("Your choice? ");
                opSelected = Console.ReadLine();
            }
            while (opSelected == string.Empty);

            var operation = (opSelected) switch
            {
                "1" => "add",
                "2" => "subtract",
                "3" => "multiply",
                "4" => "divide",
                "q" => "quit",
                _ => "add",
            };

            if (operation == "quit")
            {
                done = true;
            }
            else
            {
                // Get two numbers and an action from the user.
                value = string.Empty;
                do
                {
                    Console.Write("Enter the first value: ");
```

```
                value = Console.ReadLine();
            }
            while (value == string.Empty);

            string? value2;
            do
            {
                Console.Write("Enter a second value: ");
                value2 = Console.ReadLine();
            }
            while (value2 == string.Empty);

            functionParameters = "{" +
                "\"action\": \"" + operation + "\", " +
                "\"x\": \"" + value + "\"," +
                "\"y\": \"" + value2 + "\"" +
            "}";

            answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
            Console.WriteLine($"The answer when we {operation} the two
numbers is: {answer}.");
        }

        uiWrapper.PressEnter();
    } while (!done);

    // Delete the function created earlier.

    uiWrapper.DisplayTitle("Clean up resources");
    // Detach the IAM policy from the IAM role.
    Console.WriteLine("First detach the IAM policy from the role.");
    success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
    uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached
from the role.");

    Console.WriteLine("Delete the AWS Lambda function.");
    success = await lambdaWrapper.DeleteFunctionAsync(functionName);
    if (success)
    {
        Console.WriteLine($"The {functionName} function was deleted.");
    }
    else
```

```
        {
            Console.WriteLine($"Could not remove the function {functionName}");
        }

        // Now delete the IAM role created for use with the functions
        // created by the application.
        Console.WriteLine("Now we can delete the role that we created.");
        success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
        if (success)
        {
            Console.WriteLine("The role has been successfully removed.");
        }
        else
        {
            Console.WriteLine("Couldn't delete the role.");
        }

        Console.WriteLine("The Lambda Scenario is now complete.");
        uiWrapper.PressEnter();

        // Displays a formatted list of existing functions returned by the
        // LambdaMethods.ListFunctions.
        void DisplayFunctionList(List<FunctionConfiguration> functions)
        {
            functions.ForEach(functionConfig =>
            {

 Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
            });
        }
    }
}


namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
```

```
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the
 scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
 policy.</param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
 to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
 policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
```

```
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}


namespace LambdaScenarioCommon;
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management
 (IAM) role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
 permissions.");
```

```
        Console.WriteLine("\t3. Creates a Lambda function that increments the
value passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a
value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
```

```
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Définir un gestionnaire Lambda qui incrémente un nombre.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace LambdaIncrement;

public class Function
{

    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
```

```
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}
```

Définir un deuxième gestionnaire Lambda qui effectue des opérations arithmétiques.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace LambdaCalculator;

public class Function
{

    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
```

```
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
                else
                    result = x / y;
                break;
            default:
                Console.Error.WriteLine($"{action} is not a valid operation.");
                result = 0;
                break;
        }
        return result;
    }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for .NET APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

C++

SDKpour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
//! Get started with functions scenario.
/*!
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Lambda::getStartedWithFunctionsScenario(
        const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::Lambda::LambdaClient client(clientConfig);

    // 1. Create an AWS Identity and Access Management (IAM) role for Lambda
 function.
    Aws::String roleArn;
    if (!getIamRoleArn(roleArn, clientConfig)) {
        return false;
    }

    // 2. Create a Lambda function.
```

```
    int seconds = 0;
    do {
        Aws::Lambda::Model::CreateFunctionRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
        request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
        request.SetTimeout(15);
        request.SetMemorySize(128);

        // Assume the AWS Lambda function was built in Docker with same
 architecture
        // as this code.
#if  defined(__x86_64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_8);
#endif
        request.SetRole(roleArn);
        request.SetHandler(LAMBDA_HANDLER_NAME);
        request.SetPublish(true);
        Aws::Lambda::Model::FunctionCode code;
        std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
```

```
        buffer << ifstream.rdbuf();

        code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                               buffer.str().length()));
        request.SetCode(code);

        Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda function was successfully created. " <<
seconds
                      << " seconds elapsed." << std::endl;
            break;
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::Lambda::LambdaErrors::INVALID_PARAMETER_VALUE &&
                 outcome.GetError().GetMessage().find("role") >= 0) {
            if ((seconds % 5) == 0) { // Log status every 10 seconds.
                std::cout
                        << "Waiting for the IAM role to become available as a
CreateFunction parameter. "
                        << seconds
                        << " seconds elapsed." << std::endl;

                std::cout << outcome.GetError().GetMessage() << std::endl;
            }
        }
        else {
            std::cerr << "Error with CreateFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (60 > seconds);

    std::cout << "The current Lambda function increments 1 by an input." <<
std::endl;
```

```
    // 3.  Invoke the Lambda function.
    {
        int increment = askQuestionForInt("Enter an increment integer: ");

        Aws::Lambda::Model::InvokeResult invokeResult;
        Aws::Utils::Json::JsonValue jsonPayload;
        jsonPayload.WithString("action", "increment");
        jsonPayload.WithInteger("number", increment);
        if (invokeLambdaFunction(jsonPayload, Aws::Lambda::Model::LogType::Tail,
                                 invokeResult, client)) {
            Aws::Utils::Json::JsonValue jsonValue(invokeResult.GetPayload());
            Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
                    jsonValue.View().GetAllObjects();
            auto iter = values.find("result");
            if (iter != values.end() && iter->second.IsIntegerType()) {
                {
                    std::cout << INCREMENT_RESUlT_PREFIX
                              << iter->second.AsInteger() << std::endl;
                }
            }
            else {
                std::cout << "There was an error in execution. Here is the log."
                          << std::endl;
                Aws::Utils::ByteBuffer buffer =
Aws::Utils::HashingUtils::Base64Decode(
                        invokeResult.GetLogResult());
                std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
            }
        }
    }

    std::cout
            << "The Lambda function will now be updated with new code. Press
return to continue, ";
    Aws::String answer;
    std::getline(std::cin, answer);

    // 4.  Update the Lambda function code.
    {
        Aws::Lambda::Model::UpdateFunctionCodeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
```

```
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteLambdaFunction(client);
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
                Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                                       buffer.str().length()));
        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
 client.UpdateFunctionCode(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
 std::endl;
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "This function uses an environment variable to control the logging
 level."
            << std::endl;
    std::cout
            << "UpdateFunctionConfiguration will be used to set the LOG_LEVEL to
 DEBUG."
```

```
            << std::endl;
    seconds = 0;

    // 5.  Update the Lambda function configuration.
    do {
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        Aws::Lambda::Model::Environment environment;
        environment.AddVariables("LOG_LEVEL", "DEBUG");
        request.SetEnvironment(environment);

        Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda configuration was successfully updated."
                      << std::endl;
            break;
        }

            // RESOURCE_IN_USE: function code update not completed.
        else if (outcome.GetError().GetErrorType() !=
                 Aws::Lambda::LambdaErrors::RESOURCE_IN_USE) {
            if ((seconds % 10) == 0) { // Log status every 10 seconds.
                std::cout << "Lambda function update in progress . After " <<
seconds
                          << " seconds elapsed." << std::endl;
            }
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }

    } while (0 < seconds);

    if (0 > seconds) {
        std::cerr << "Function failed to become active." << std::endl;
    }
    else {
```

```
        std::cout << "Updated function active after " << seconds << " seconds."
                  << std::endl;
    }

    std::cout
            << "\nThe new code applies an arithmetic operator to two variables, x
 an y."
            << std::endl;
    std::vector<Aws::String> operators = {"plus", "minus", "times", "divided-
by"};
    for (size_t i = 0; i < operators.size(); ++i) {
        std::cout << "   " << i + 1 << " " << operators[i] << std::endl;
    }

    // 6.  Invoke the updated Lambda function.
    do {
        int operatorIndex = askQuestionForIntRange("Select an operator index 1 -
 4 ", 1,
                                                   4);
        int x = askQuestionForInt("Enter an integer for the x value ");
        int y = askQuestionForInt("Enter an integer for the y value ");

        Aws::Utils::Json::JsonValue calculateJsonPayload;
        calculateJsonPayload.WithString("action", operators[operatorIndex - 1]);
        calculateJsonPayload.WithInteger("x", x);
        calculateJsonPayload.WithInteger("y", y);
        Aws::Lambda::Model::InvokeResult calculatedResult;
        if (invokeLambdaFunction(calculateJsonPayload,
                                 Aws::Lambda::Model::LogType::Tail,
                                 calculatedResult, client)) {
            Aws::Utils::Json::JsonValue jsonValue(calculatedResult.GetPayload());
            Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
                    jsonValue.View().GetAllObjects();
            auto iter = values.find("result");
            if (iter != values.end() && iter->second.IsIntegerType()) {
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                          << operators[operatorIndex - 1] << " "
                          << y << " is " << iter->second.AsInteger() <<
 std::endl;
            }
            else if (iter != values.end() && iter->second.IsFloatingPointType())
 {
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                          << operators[operatorIndex - 1] << " "
```

```
                        << y << " is " << iter->second.AsDouble() << std::endl;
            }
            else {
                std::cout << "There was an error in execution. Here is the log."
                          << std::endl;
                Aws::Utils::ByteBuffer buffer =
Aws::Utils::HashingUtils::Base64Decode(
                        calculatedResult.GetLogResult());
                std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
            }
        }

        answer = askQuestion("Would you like to try another operation? (y/n) ");
    } while (answer == "y");

    std::cout
            << "A list of the lambda functions will be retrieved. Press return to
continue, ";
    std::getline(std::cin, answer);

    // 7.  List the Lambda functions.

    std::vector<Aws::String> functions;
    Aws::String marker;

    do {
        Aws::Lambda::Model::ListFunctionsRequest request;
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
                request);

        if (outcome.IsSuccess()) {
            const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
            std::cout << result.GetFunctions().size()
                      << " lambda functions were retrieved." << std::endl;

            for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
                functions.push_back(functionConfiguration.GetFunctionName());
```

```
                std::cout << functions.size() << "  "
                          << functionConfiguration.GetDescription() << std::endl;
                std::cout << "   "
                          <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                  functionConfiguration.GetRuntime()) << ": "
                          << functionConfiguration.GetHandler()
                          << std::endl;
            }
            marker = result.GetNextMarker();
        }
        else {
            std::cerr << "Error with Lambda::ListFunctions. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    // 8.  Get a Lambda function.
    if (!functions.empty()) {
        std::stringstream question;
        question << "Choose a function to retrieve between 1 and " <<
functions.size()
                 << " ";
        int functionIndex = askQuestionForIntRange(question.str(), 1,

static_cast<int>(functions.size()));

        Aws::String functionName = functions[functionIndex - 1];

        Aws::Lambda::Model::GetFunctionRequest request;
        request.SetFunctionName(functionName);

        Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

        if (outcome.IsSuccess()) {
            std::cout << "Function retrieve.\n" <<

outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
                      << std::endl;
        }
        else {
            std::cerr << "Error with Lambda::GetFunction. "
```

```
                        << outcome.GetError().GetMessage()
                        << std::endl;
        }
    }

    std::cout << "The resources will be deleted. Press return to continue, ";
    std::getline(std::cin, answer);

    // 9.  Delete the Lambda function.
    bool result = deleteLambdaFunction(client);

    // 10. Delete the IAM role.
    return result && deleteIamRole(clientConfig);
}

//! Routine which invokes a Lambda function and returns the result.
/*!
 \param jsonPayload: Payload for invoke function.
 \param logType: Log type setting for invoke function.
 \param invokeResult: InvokeResult object to receive the result.
 \param client: Lambda client.
 \return bool: Successful completion.
 */
bool
AwsDoc::Lambda::invokeLambdaFunction(const Aws::Utils::Json::JsonValue
 &jsonPayload,
                                     Aws::Lambda::Model::LogType logType,
                                     Aws::Lambda::Model::InvokeResult
 &invokeResult,
                                     const Aws::Lambda::LambdaClient &client) {
    int seconds = 0;
    bool result = false;
    /*
     * In this example, the Invoke function can be called before recently created
 resources are
     * available.  The Invoke function is called repeatedly until the resources
 are
     * available.
     */
    do {
        Aws::Lambda::Model::InvokeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetLogType(logType);
```

```
        std::shared_ptr<Aws::IOStream> payload =
 Aws::MakeShared<Aws::StringStream>(
                "FunctionTest");
        *payload << jsonPayload.View().WriteReadable();
        request.SetBody(payload);
        request.SetContentType("application/json");
        Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

        if (outcome.IsSuccess()) {
            invokeResult = std::move(outcome.GetResult());
            result = true;
            break;
        }

            // ACCESS_DENIED: because the role is not available yet.
            // RESOURCE_CONFLICT: because the Lambda function is being created or
 updated.
        else if ((outcome.GetError().GetErrorType() ==
                  Aws::Lambda::LambdaErrors::ACCESS_DENIED) ||
                 (outcome.GetError().GetErrorType() ==
                  Aws::Lambda::LambdaErrors::RESOURCE_CONFLICT)) {
            if ((seconds % 5) == 0) { // Log status every 10 seconds.
                std::cout << "Waiting for the invoke api to be available, status
 " <<
                          ((outcome.GetError().GetErrorType() ==
                            Aws::Lambda::LambdaErrors::ACCESS_DENIED ?
                            "ACCESS_DENIED" : "RESOURCE_CONFLICT")) << ". " <<
 seconds
                          << " seconds elapsed." << std::endl;
            }
        }
        else {
            std::cerr << "Error with Lambda::InvokeRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            break;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (seconds < 60);

    return result;
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for C++ APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un scénario interactif qui vous montre comment démarrer avec les fonctions Lambda.

```
// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
 following
// actions:
//
//  1. Create an AWS Identity and Access Management (IAM) role and Lambda
 function, then upload handler code.
//  2. Invoke the function with a single parameter and get results.
//  3. Update the function code and configure with an environment variable.
//  4. Invoke the function with new parameters and get results. Display the
 returned execution log.
//  5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
 sdkConfig       aws.Config
 functionWrapper actions.FunctionWrapper
```

```
 questioner      demotools.IQuestioner
 helper          IScenarioHelper
 isTestRun       bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario
 instance from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for
 the actions
// used in the scenario.
func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedFunctionsScenario {
 lambdaClient := lambda.NewFromConfig(sdkConfig)
 return GetStartedFunctionsScenario{
  sdkConfig:       sdkConfig,
  functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
  questioner:      questioner,
  helper:          helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Lambda get started with functions demo.")
 log.Println(strings.Repeat("-", 88))

 role := scenario.GetOrCreateRole(ctx)
 funcName := scenario.CreateFunction(ctx, role)
 scenario.InvokeIncrement(ctx, funcName)
 scenario.UpdateFunction(ctx, funcName)
 scenario.InvokeCalculator(ctx, funcName)
 scenario.ListFunctions(ctx)
 scenario.Cleanup(ctx, role, funcName)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
```

```
 log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
 does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
 role
// is returned.
func (scenario GetStartedFunctionsScenario) GetOrCreateRole(ctx context.Context)
 *iamtypes.Role {
 var role *iamtypes.Role
 iamClient := iam.NewFromConfig(scenario.sdkConfig)
 log.Println("First, we need an IAM role that Lambda can assume.")
 roleName := scenario.questioner.Ask("Enter a name for the role:",
 demotools.NotEmpty{})
 getOutput, err := iamClient.GetRole(ctx, &iam.GetRoleInput{
  RoleName: aws.String(roleName)})
 if err != nil {
  var noSuch *iamtypes.NoSuchEntityException
  if errors.As(err, &noSuch) {
   log.Printf("Role %v doesn't exist. Creating it....\n", roleName)
  } else {
   log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
    roleName, err)
  }
 } else {
  role = getOutput.Role
  log.Printf("Found role %v.\n", *role.RoleName)
 }
 if role == nil {
  trustPolicy := PolicyDocument{
   Version: "2012-10-17",
   Statement: []PolicyStatement{{
    Effect:    "Allow",
    Principal: map[string]string{"Service": "lambda.amazonaws.com"},
    Action:    []string{"sts:AssumeRole"},
   }},
  }
  policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
  createOutput, err := iamClient.CreateRole(ctx, &iam.CreateRoleInput{
   AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
   RoleName:                 aws.String(roleName),
  })
```

```
  if err != nil {
   log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
  }
  role = createOutput.Role
  _, err = iamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
   PolicyArn: aws.String(policyArn),
   RoleName:  aws.String(roleName),
  })
  if err != nil {
   log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
 err)
  }
  log.Printf("Created role %v.\n", *role.RoleName)
  log.Println("Let's give AWS a few seconds to propagate resources...")
  scenario.helper.Pause(10)
 }
 log.Println(strings.Repeat("-", 88))
 return role
}

// CreateFunction creates a Lambda function and uploads a handler written in
 Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(ctx context.Context,
 role *iamtypes.Role) string {
 log.Println("Let's create a function that increments a number.\n" +
  "The function uses the 'lambda_handler_basic.py' script found in the \n" +
  "'handlers' directory of this project.")
 funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
 demotools.NotEmpty{})
 zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
 fmt.Sprintf("%v.py", funcName))
 log.Printf("Creating function %v and waiting for it to be ready.", funcName)
 funcState := scenario.functionWrapper.CreateFunction(ctx, funcName,
 fmt.Sprintf("%v.lambda_handler", funcName),
  role.Arn, zipPackage)
 log.Printf("Your function is %v.", funcState)
 log.Println(strings.Repeat("-", 88))
 return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The
 function
```

```
// parameters are contained in a Go struct that is used to serialize the
 parameters to
// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int
 value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(ctx context.Context,
 funcName string) {
 parameters := actions.IncrementParameters{Action: "increment"}
 log.Println("Let's invoke our function. This function increments a number.")
 parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
 demotools.NotEmpty{})
 log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
 invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, parameters,
 false)
 var payload actions.LambdaResultInt
 err := json.Unmarshal(invokeOutput.Payload, &payload)
 if err != nil {
  log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
   funcName, err)
 }
 log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
 payload)
 log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
 arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(ctx context.Context,
 funcName string) {
 log.Println("Let's update the function to an arithmetic calculator.\n" +
  "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
  "'handlers' directory of this project.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 log.Println("Creating deployment package...")
 zipPackage :=
 scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
  fmt.Sprintf("%v.py", funcName))
 log.Println("...and updating the Lambda function and waiting for it to be
 ready.")
```

```
 funcState := scenario.functionWrapper.UpdateFunctionCode(ctx, funcName,
 zipPackage)
 log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
 log.Println("This function uses an environment variable to control logging
 level.")
 log.Println("Let's set it to DEBUG to get the most logging.")
 scenario.functionWrapper.UpdateFunctionConfiguration(ctx, funcName,
  map[string]string{"LOG_LEVEL": "DEBUG"})
 log.Println(strings.Repeat("-", 88))
}

// InvokeCalculator invokes the Lambda calculator function. The parameters are
 stored in a
// Go struct that is used to serialize the parameters to a JSON payload. That
 payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
 either an
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(ctx context.Context,
 funcName string) {
 wantInvoke := true
 choices := []string{"plus", "minus", "times", "divided-by"}
 for wantInvoke {
  choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
 choices)
  x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
  y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
  log.Printf("Invoking %v %v %v...", x, choices[choice], y)
  calcParameters := actions.CalculatorParameters{
   Action: choices[choice],
   X:      x,
   Y:      y,
  }
  invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, calcParameters,
 true)
  var payload any
  if choice == 3 { // divide-by results in a float.
   payload = actions.LambdaResultFloat{}
  } else {
   payload = actions.LambdaResultInt{}
  }
  err := json.Unmarshal(invokeOutput.Payload, &payload)
  if err != nil {
```

```
   log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
    funcName, err)
  }
  log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
   calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
  scenario.questioner.Ask("Press Enter to see the logs from the call.")
  logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
  if err != nil {
   log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
  }
  log.Println(string(logRes))
  wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/
n)", "y")
 }
 log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
func (scenario GetStartedFunctionsScenario) ListFunctions(ctx context.Context) {
 count := scenario.questioner.AskInt(
  "Let's list functions for your account. How many do you want to see?",
 demotools.NotEmpty{})
 functions := scenario.functionWrapper.ListFunctions(ctx, count)
 log.Printf("Found %v functions:", len(functions))
 for _, function := range functions {
  log.Printf("\t%v", *function.FunctionName)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(ctx context.Context, role
 *iamtypes.Role, funcName string) {
 if scenario.questioner.AskBool("Do you want to clean up resources created for
 this example? (y/n)",
  "y") {
  iamClient := iam.NewFromConfig(scenario.sdkConfig)
  policiesOutput, err := iamClient.ListAttachedRolePolicies(ctx,
   &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
    *role.RoleName, err)
  }
  for _, policy := range policiesOutput.AttachedPolicies {
```

```
   _, err = iamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
    PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
   })
   if err != nil {
    log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
     *policy.PolicyArn, *role.RoleName, err)
   }
  }
  _, err = iamClient.DeleteRole(ctx, &iam.DeleteRoleInput{RoleName:
 role.RoleName})
  if err != nil {
   log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
  }
  log.Printf("Deleted role %v.\n", *role.RoleName)

  scenario.functionWrapper.DeleteFunction(ctx, funcName)
  log.Printf("Deleted function %v.\n", funcName)
 } else {
  log.Println("Okay. Don't forget to delete the resources when you're done with
 them.")
 }
}
```

Créez une structure qui englobe les actions Lambda individuelles.

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}


// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
 string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
 &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
```

```
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython38,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
```

```
    log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
   } else {
    state = funcOutput.Configuration.State
   }
 }
 return state
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in
 the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
 functionName string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
 functionName, err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
   } else {
    state = funcOutput.Configuration.State
   }
 }
 return state
}
```

```
// UpdateFunctionConfiguration updates a map of environment variables configured
 for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v",
 functionName, err)
 }
}



// ListFunctions lists up to maxItems functions for the account. This function
 uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
```

```
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}



// IncrementParameters is used to serialize parameters to the increment Lambda
 handler.
type IncrementParameters struct {
 Action string `json:"action"`
 Number int    `json:"number"`
}
```

```
// CalculatorParameters is used to serialize parameters to the calculator Lambda
 handler.
type CalculatorParameters struct {
 Action string `json:"action"`
 X      int    `json:"x"`
 Y      int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
 Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda
 handler.
type LambdaResultFloat struct {
 Result float32 `json:"result"`
}
```

Créez une structure qui met en œuvre des fonctions pour aider à exécuter le scénario.

```
// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
 HandlerPath string
}

// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
 file. The
```

```
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can
 be
// used to pass a []byte to Lambda when creating the function.
// The specified destinationFile is the name to give the file when it's deployed
 to Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
 destinationFile string) *bytes.Buffer {
 var err error
 buffer := &bytes.Buffer{}
 writer := zip.NewWriter(buffer)
 zFile, err := writer.Create(destinationFile)
 if err != nil {
  log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
 destinationFile, err)
 }
 sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
 sourceFile))
 if err != nil {
  log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
   sourceFile, err)
 } else {
  _, err = zFile.Write(sourceBody)
  if err != nil {
   log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
    sourceFile, err)
  }
 }
 err = writer.Close()
 if err != nil {
  log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
 }
 return buffer
}
```

Définir un gestionnaire Lambda qui incrémente un nombre.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

```
def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Définir un deuxième gestionnaire Lambda qui effectue des opérations arithmétiques.

```
import logging
import os


logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}
```

```
def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
 numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Go APIRéférence.
  - CreateFunction
  - DeleteFunction

- GetFunction
- Invoke
- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
/*
 *  Lambda function names appear as:
 *
 *  arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
 *
 *  To find this value, look at the function in the AWS Management Console.
 *
 *  Before running this Java code example, set up your development environment,
 including your credentials.
 *
 *  For more information, see this documentation topic:
 *
 *  https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 *  This example performs the following tasks:
 *
 * 1. Creates an AWS Lambda function.
 * 2. Gets a specific AWS Lambda function.
 * 3. Lists all Lambda functions.
 * 4. Invokes a Lambda function.
 * 5. Updates the Lambda function code and invokes it again.
 * 6. Updates a Lambda function's configuration value.
 * 7. Deletes a Lambda function.
```

```
 */

public class LambdaScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = """

                Usage:
                    <functionName> <filePath> <role> <handler> <bucketName> <key>
\s

                Where:
                    functionName - The name of the Lambda function.\s
                    filePath - The path to the .zip or .jar where the code is
 located.\s
                    role - The AWS Identity and Access Management (IAM) service
 role that has Lambda permissions.\s
                    handler - The fully qualified method name (for example,
 example.Handler::handleRequest).\s
                    bucketName - The Amazon Simple Storage Service (Amazon S3)
 bucket name that contains the .zip or .jar used to update the Lambda function's
 code.\s
                    key - The Amazon S3 key name that represents the .zip or .jar
 (for example, LambdaHello-1.0-SNAPSHOT.jar).
                    """;

        if (args.length != 6) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        String filePath = args[1];
        String role = args[2];
        String handler = args[3];
        String bucketName = args[4];
        String key = args[5];

        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();
```

```
        System.out.println(DASHES);
        System.out.println("Welcome to the AWS Lambda example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Create an AWS Lambda function.");
        String funArn = createLambdaFunction(awsLambda, functionName, filePath,
role, handler);
        System.out.println("The AWS Lambda ARN is " + funArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Get the " + functionName + " AWS Lambda
function.");
        getFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. List all AWS Lambda functions.");
        listFunctions(awsLambda);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Invoke the Lambda function.");
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Update the Lambda function code and invoke it
again.");
        updateFunctionCode(awsLambda, functionName, bucketName, key);
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Update a Lambda function's configuration value.");
        updateFunctionConfiguration(awsLambda, functionName, handler);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("7. Delete the AWS Lambda function.");
        LambdaScenario.deleteLambdaFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS Lambda scenario completed successfully");
        System.out.println(DASHES);
        awsLambda.close();
    }

    public static String createLambdaFunction(LambdaClient awsLambda,
            String functionName,
            String filePath,
            String role,
            String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            InputStream is = new FileInputStream(filePath);
            SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

            FunctionCode code = FunctionCode.builder()
                    .zipFile(fileToUpload)
                    .build();

            CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
                    .functionName(functionName)
                    .description("Created by the Lambda Java API")
                    .code(code)
                    .handler(handler)
                    .runtime(Runtime.JAVA8)
                    .role(role)
                    .build();

            // Create a Lambda function using a waiter
            CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
            GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
                    .functionName(functionName)
                    .build();
            WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
```

```
            waiterResponse.matched().response().ifPresent(System.out::println);
            return functionResponse.functionArn();

        } catch (LambdaException | FileNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static void getFunction(LambdaClient awsLambda, String functionName) {
        try {
            GetFunctionRequest functionRequest = GetFunctionRequest.builder()
                    .functionName(functionName)
                    .build();

            GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
            System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
                System.out.println("The function name is " +
config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void invokeFunction(LambdaClient awsLambda, String
functionName) {
```

```
        InvokeResponse res;
        try {
            // Need a SdkBytes instance for the payload.
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("inputValue", "2000");
            String json = jsonObj.toString();
            SdkBytes payload = SdkBytes.fromUtf8String(json);

            InvokeRequest request = InvokeRequest.builder()
                    .functionName(functionName)
                    .payload(payload)
                    .build();

            res = awsLambda.invoke(request);
            String value = res.payload().asUtf8String();
            System.out.println(value);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
        try {
            LambdaWaiter waiter = awsLambda.waiter();
            UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
                    .functionName(functionName)
                    .publish(true)
                    .s3Bucket(bucketName)
                    .s3Key(key)
                    .build();

            UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
            GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
                    .functionName(functionName)
                    .build();
```

```
            WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
waiter
                    .waitUntilFunctionUpdated(getFunctionConfigRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("The last modified value is " +
response.lastModified());

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
        try {
            UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
                    .functionName(functionName)
                    .handler(handler)
                    .runtime(Runtime.JAVA11)
                    .build();

            awsLambda.updateFunctionConfiguration(configurationRequest);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
        try {
            DeleteFunctionRequest request = DeleteFunctionRequest.builder()
                    .functionName(functionName)
                    .build();

            awsLambda.deleteFunction(request);
            System.out.println("The " + functionName + " function was deleted");

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }
    }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Java 2.x APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez un rôle AWS Identity and Access Management (IAM) qui accorde à Lambda l'autorisation d'écrire dans les journaux.

```
    log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
```

```
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

Créer une fonction Lambda et télécharger le code de gestionnaire.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

Invoquer la fonction avec un seul paramètre et obtenir des résultats.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
```

```
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Mettre à jour le code de fonction et configurer son environnement Lambda avec une variable d'environnement.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

Répertorier les fonctions pour votre compte.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Supprimez le IAM rôle et la fonction Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for JavaScript APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

## Kotlin

### SDKpour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <functionName> <role> <handler> <bucketName> <updatedBucketName>
 <key>

        Where:
            functionName - The name of the AWS Lambda function.
            role - The AWS Identity and Access Management (IAM) service role that
 has AWS Lambda permissions.
            handler - The fully qualified method name (for example,
 example.Handler::handleRequest).
            bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
 name that contains the ZIP or JAR used for the Lambda function's code.
            updatedBucketName - The Amazon S3 bucket name that contains the .zip
 or .jar used to update the Lambda function's code.
            key - The Amazon S3 key name that represents the .zip or .jar file
 (for example, LambdaHello-1.0-SNAPSHOT.jar).
            """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val functionName = args[0]
    val role = args[1]
    val handler = args[2]
    val bucketName = args[3]
    val updatedBucketName = args[4]
    val key = args[5]
```

```
    println("Creating a Lambda function named $functionName.")
    val funArn = createScFunction(functionName, bucketName, key, handler, role)
    println("The AWS Lambda ARN is $funArn")

    // Get a specific Lambda function.
    println("Getting the $functionName AWS Lambda function.")
    getFunction(functionName)

    // List the Lambda functions.
    println("Listing all AWS Lambda functions.")
    listFunctionsSc()

    // Invoke the Lambda function.
    println("*** Invoke the Lambda function.")
    invokeFunctionSc(functionName)

    // Update the AWS Lambda function code.
    println("*** Update the Lambda function code.")
    updateFunctionCode(functionName, updatedBucketName, key)

    // println("*** Invoke the function again after updating the code.")
    invokeFunctionSc(functionName)

    // Update the AWS Lambda function configuration.
    println("Update the run time of the function.")
    updateFunctionConfiguration(functionName, handler)

    // Delete the AWS Lambda function.
    println("Delete the AWS Lambda function.")
    delFunction(functionName)
}

suspend fun createScFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }
```

```
    val request =
        CreateFunctionRequest {
            functionName = myFunctionName
            code = functionCode
            description = "Created by the Lambda Kotlin API"
            handler = myHandler
            role = myRole
            runtime = Runtime.Java8
        }

    // Create a Lambda function using a waiter
    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn.toString()
    }
}

suspend fun getFunction(functionNameVal: String) {
    val functionRequest =
        GetFunctionRequest {
            functionName = functionNameVal
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.getFunction(functionRequest)
        println("The runtime of this Lambda function is
 ${response.configuration?.runtime}")
    }
}

suspend fun listFunctionsSc() {
    val request =
        ListFunctionsRequest {
            maxItems = 10
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.listFunctions(request)
        response.functions?.forEach { function ->
            println("The function name is ${function.functionName}")
```

```
        }
    }
}

suspend fun invokeFunctionSc(functionNameVal: String) {
    val json = """{"inputValue":"1000"}"""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            payload = byteArray
            logType = LogType.Tail
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("The function payload is
 ${res.payload?.toString(Charsets.UTF_8)}")
    }
}

suspend fun updateFunctionCode(
    functionNameVal: String?,
    bucketName: String?,
    key: String?,
) {
    val functionCodeRequest =
        UpdateFunctionCodeRequest {
            functionName = functionNameVal
            publish = true
            s3Bucket = bucketName
            s3Key = key
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.updateFunctionCode(functionCodeRequest)
        awsLambda.waitUntilFunctionUpdated {
            functionName = functionNameVal
        }
        println("The last modified value is " + response.lastModified)
    }
}

suspend fun updateFunctionConfiguration(
```

```
    functionNameVal: String?,
    handlerVal: String?,
) {
    val configurationRequest =
        UpdateFunctionConfigurationRequest {
            functionName = functionNameVal
            handler = handlerVal
            runtime = Runtime.Java11
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.updateFunctionConfiguration(configurationRequest)
    }
}

suspend fun delFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Pour API plus de détails, consultez les rubriques suivantes AWS SDKpour API référence à Kotlin.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
namespace Lambda;

use Aws\S3\S3Client;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithLambda
{
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the AWS Lambda getting started demo using PHP!\n");
        echo("---------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $iamService = new IAMService();
        $s3client = new S3Client($clientArgs);
        $lambdaService = new LambdaService();

        echo "First, let's create a role to run our Lambda code.\n";
        $roleName = "test-lambda-role-$uniqid";
        $rolePolicyDocument = "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
                {
```

```
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"Service\": \"lambda.amazonaws.com\"
                    },
                    \"Action\": \"sts:AssumeRole\"
                }
            ]
        }";
        $role = $iamService->createRole($roleName, $rolePolicyDocument);
        echo "Created role {$role['RoleName']}.\n";

        $iamService->attachRolePolicy(
            $role['RoleName'],
            "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        );
        echo "Attached the AWSLambdaBasicExecutionRole to {$role['RoleName']}.
\n";

        echo "\nNow let's create an S3 bucket and upload our Lambda code there.
\n";
        $bucketName = "test-example-bucket-$uniqid";
        $s3client->createBucket([
            'Bucket' => $bucketName,
        ]);
        echo "Created bucket $bucketName.\n";

        $functionName = "doc_example_lambda_$uniqid";
        $codeBasic = __DIR__ . "/lambda_handler_basic.zip";
        $handler = "lambda_handler_basic";
        $file = file_get_contents($codeBasic);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "Uploaded the Lambda code.\n";

        $createLambdaFunction = $lambdaService->createFunction($functionName,
 $role, $bucketName, $handler);
        // Wait until the function has finished being created.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['State'] == "Pending");
```

```
        echo "Created Lambda function {$getLambdaFunction['Configuration']
['FunctionName']}.\n";

        sleep(1);

        echo "\nOk, let's invoke that Lambda code.\n";
        $basicParams = [
            'action' => 'increment',
            'number' => 3,
        ];
        /** @var Stream $invokeFunction */
        $invokeFunction = $lambdaService->invoke($functionName, $basicParams)
['Payload'];
        $result = json_decode($invokeFunction->getContents())->result;
        echo "After invoking the Lambda code with the input of
 {$basicParams['number']} we received $result.\n";

        echo "\nSince that's working, let's update the Lambda code.\n";
        $codeCalculator = "lambda_handler_calculator.zip";
        $handlerCalculator = "lambda_handler_calculator";
        echo "First, put the new code into the S3 bucket.\n";
        $file = file_get_contents($codeCalculator);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "New code uploaded.\n";

        $lambdaService->updateFunctionCode($functionName, $bucketName,
 $functionName);
        // Wait for the Lambda code to finish updating.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "New Lambda code uploaded.\n";

        $environment = [
            'Variable' => ['Variables' => ['LOG_LEVEL' => 'DEBUG']],
        ];
        $lambdaService->updateFunctionConfiguration($functionName,
 $handlerCalculator, $environment);
```

```
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "Lambda code updated with new handler and a LOG_LEVEL of DEBUG for
 more information.\n";

        echo "Invoke the new code with some new data.\n";
        $calculatorParams = [
            'action' => 'plus',
            'x' => 5,
            'y' => 4,
        ];
        $invokeFunction = $lambdaService->invoke($functionName,
 $calculatorParams, "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "Indeed, {$calculatorParams['x']} + {$calculatorParams['y']} does
 equal $result.\n";
        echo "Here's the extra debug info: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nBut what happens if you try to divide by zero?\n";
        $divZeroParams = [
            'action' => 'divide',
            'x' => 5,
            'y' => 0,
        ];
        $invokeFunction = $lambdaService->invoke($functionName, $divZeroParams,
 "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "You get a |$result| result.\n";
        echo "And an error message: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nHere's all the Lambda functions you have in this Region:\n";
        $listLambdaFunctions = $lambdaService->listFunctions(5);
        $allLambdaFunctions = $listLambdaFunctions['Functions'];
        $next = $listLambdaFunctions->get('NextMarker');
        while ($next != false) {
            $listLambdaFunctions = $lambdaService->listFunctions(5, $next);
            $next = $listLambdaFunctions->get('NextMarker');
            $allLambdaFunctions = array_merge($allLambdaFunctions,
 $listLambdaFunctions['Functions']);
```

```
        }
        foreach ($allLambdaFunctions as $function) {
            echo "{$function['FunctionName']}\n";
        }

        echo "\n\nAnd don't forget to clean up your data!\n";

        $lambdaService->deleteFunction($functionName);
        echo "Deleted Lambda function.\n";
        $iamService->deleteRole($role['RoleName']);
        echo "Deleted Role.\n";
        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Deleted all objects from the S3 bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);
        echo "Deleted the bucket.\n";
    }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for PHP APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définir un gestionnaire Lambda qui incrémente un nombre.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Définir un deuxième gestionnaire Lambda qui effectue des opérations arithmétiques.

```
import logging
import os


logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}


def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
 numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
```

```
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

Créer des fonctions qui enveloppent les actions Lambda.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    @staticmethod
    def create_deployment_package(source_file, destination_file):
        """
        Creates a Lambda deployment package in .zip format in an in-memory
 buffer. This
        buffer can be passed directly to Lambda when creating the function.

        :param source_file: The name of the file that contains the Lambda handler
                            function.
        :param destination_file: The name to give the file when it's deployed to
 Lambda.
        :return: The deployment package.
        """
        buffer = io.BytesIO()
        with zipfile.ZipFile(buffer, "w") as zipped:
            zipped.write(source_file, destination_file)
        buffer.seek(0)
        return buffer.read()

    def get_iam_role(self, iam_role_name):
        """
        Get an AWS Identity and Access Management (IAM) role.
```

```
        :param iam_role_name: The name of the role to retrieve.
        :return: The IAM role.
        """
        role = None
        try:
            temp_role = self.iam_resource.Role(iam_role_name)
            temp_role.load()
            role = temp_role
            logger.info("Got IAM role %s", role.name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NoSuchEntity":
                logger.info("IAM role %s does not exist.", iam_role_name)
            else:
                logger.error(
                    "Couldn't get IAM role %s. Here's why: %s: %s",
                    iam_role_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return role

    def create_iam_role_for_lambda(self, iam_role_name):
        """
        Creates an IAM role that grants the Lambda function basic permissions. If
a
        role with the specified name already exists, it is used for the demo.

        :param iam_role_name: The name of the role to create.
        :return: The role and a value that indicates whether the role is newly
created.
        """
        role = self.get_iam_role(iam_role_name)
        if role is not None:
            return role, False

        lambda_assume_role_policy = {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {"Service": "lambda.amazonaws.com"},
                    "Action": "sts:AssumeRole",
```

```
                }
            ],
        }
        policy_arn = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

        try:
            role = self.iam_resource.create_role(
                RoleName=iam_role_name,
                AssumeRolePolicyDocument=json.dumps(lambda_assume_role_policy),
            )
            logger.info("Created role %s.", role.name)
            role.attach_policy(PolicyArn=policy_arn)
            logger.info("Attached basic execution policy to role %s.", role.name)
        except ClientError as error:
            if error.response["Error"]["Code"] == "EntityAlreadyExists":
                role = self.iam_resource.Role(iam_role_name)
                logger.warning("The role %s already exists. Using it.",
 iam_role_name)
            else:
                logger.exception(
                    "Couldn't create role %s or attach policy %s.",
                    iam_role_name,
                    policy_arn,
                )
                raise

        return role, True

    def get_function(self, function_name):
        """
        Gets data about a Lambda function.

        :param function_name: The name of the function.
        :return: The function data.
        """
        response = None
        try:
            response =
 self.lambda_client.get_function(FunctionName=function_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Function %s does not exist.", function_name)
            else:
```

```
                logger.error(
                    "Couldn't get function %s. Here's why: %s: %s",
                    function_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response


    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
This
                             must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
function
                                   code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
        """
        try:
            response = self.lambda_client.create_function(
                FunctionName=function_name,
                Description="AWS Lambda doc example",
                Runtime="python3.8",
                Role=iam_role.arn,
                Handler=handler_name,
                Code={"ZipFile": deployment_package},
                Publish=True,
            )
            function_arn = response["FunctionArn"]
            waiter = self.lambda_client.get_waiter("function_active_v2")
            waiter.wait(FunctionName=function_name)
            logger.info(
                "Created function '%s' with ARN: '%s'.",
                function_name,
                response["FunctionArn"],
            )
```

```
        except ClientError:
            logger.error("Couldn't create function %s.", function_name)
            raise
        else:
            return function_arn


    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise


    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
dict
                                is serialized to JSON before it is sent to
Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
included in
                        the response.
        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
```

```
        return response


    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
in
                                   .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
```

```
            logger.error(
                "Couldn't update function configuration %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
                        print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
        except ClientError as err:
            logger.error(
                "Couldn't list functions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Créer une fonction qui exécute le scénario.

```
class UpdateFunctionWaiter(CustomWaiter):
    """A custom waiter that waits until a function is successfully updated."""

    def __init__(self, client):
        super().__init__(
```

```
            "UpdateSuccess",
            "GetFunction",
            "Configuration.LastUpdateStatus",
            {"Successful": WaitState.SUCCESS, "Failed": WaitState.FAILURE},
            client,
        )

    def wait(self, function_name):
        self._wait(FunctionName=function_name)


def run_scenario(lambda_client, iam_resource, basic_file, calculator_file,
 lambda_name):
    """
    Runs the scenario.

    :param lambda_client: A Boto3 Lambda client.
    :param iam_resource: A Boto3 IAM resource.
    :param basic_file: The name of the file that contains the basic Lambda
 handler.
    :param calculator_file: The name of the file that contains the calculator
 Lambda handler.
    :param lambda_name: The name to give resources created for the scenario, such
 as the
                        IAM role and the Lambda function.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the AWS Lambda getting started with functions demo.")
    print("-" * 88)

    wrapper = LambdaWrapper(lambda_client, iam_resource)

    print("Checking for IAM role for Lambda...")
    iam_role, should_wait = wrapper.create_iam_role_for_lambda(lambda_name)
    if should_wait:
        logger.info("Giving AWS time to create resources...")
        wait(10)

    print(f"Looking for function {lambda_name}...")
    function = wrapper.get_function(lambda_name)
    if function is None:
        print("Zipping the Python script into a deployment package...")
```

```
        deployment_package = wrapper.create_deployment_package(
            basic_file, f"{lambda_name}.py"
        )
        print(f"...and creating the {lambda_name} Lambda function.")
        wrapper.create_function(
            lambda_name, f"{lambda_name}.lambda_handler", iam_role,
deployment_package
        )
    else:
        print(f"Function {lambda_name} already exists.")
    print("-" * 88)

    print(f"Let's invoke {lambda_name}. This function increments a number.")
    action_params = {
        "action": "increment",
        "number": q.ask("Give me a number to increment: ", q.is_int),
    }
    print(f"Invoking {lambda_name}...")
    response = wrapper.invoke_function(lambda_name, action_params)
    print(
        f"Incrementing {action_params['number']} resulted in "
        f"{json.load(response['Payload'])}"
    )
    print("-" * 88)

    print(f"Let's update the function to an arithmetic calculator.")
    q.ask("Press Enter when you're ready.")
    print("Creating a new deployment package...")
    deployment_package = wrapper.create_deployment_package(
        calculator_file, f"{lambda_name}.py"
    )
    print(f"...and updating the {lambda_name} Lambda function.")
    update_waiter = UpdateFunctionWaiter(lambda_client)
    wrapper.update_function_code(lambda_name, deployment_package)
    update_waiter.wait(lambda_name)
    print(f"This function uses an environment variable to control logging
level.")
    print(f"Let's set it to DEBUG to get the most logging.")
    wrapper.update_function_configuration(
        lambda_name, {"LOG_LEVEL": logging.getLevelName(logging.DEBUG)}
    )

    actions = ["plus", "minus", "times", "divided-by"]
    want_invoke = True
```

```
    while want_invoke:
        print(f"Let's invoke {lambda_name}. You can invoke these actions:")
        for index, action in enumerate(actions):
            print(f"{index + 1}: {action}")
        action_params = {}
        action_index = q.ask(
            "Enter the number of the action you want to take: ",
            q.is_int,
            q.in_range(1, len(actions)),
        )
        action_params["action"] = actions[action_index - 1]
        print(f"You've chosen to invoke 'x {action_params['action']} y'.")
        action_params["x"] = q.ask("Enter a value for x: ", q.is_int)
        action_params["y"] = q.ask("Enter a value for y: ", q.is_int)
        print(f"Invoking {lambda_name}...")
        response = wrapper.invoke_function(lambda_name, action_params, True)
        print(
            f"Calculating {action_params['x']} {action_params['action']}
{action_params['y']} "
            f"resulted in {json.load(response['Payload'])}"
        )
        q.ask("Press Enter to see the logs from the call.")
        print(base64.b64decode(response["LogResult"]).decode())
        want_invoke = q.ask("That was fun. Shall we do it again? (y/n) ",
q.is_yesno)
    print("-" * 88)

    if q.ask(
        "Do you want to list all of the functions in your account? (y/n) ",
q.is_yesno
    ):
        wrapper.list_functions()
    print("-" * 88)

    if q.ask("Ready to delete the function and role? (y/n) ", q.is_yesno):
        for policy in iam_role.attached_policies.all():
            policy.detach_role(RoleName=iam_role.name)
        iam_role.delete()
        print(f"Deleted role {lambda_name}.")
        wrapper.delete_function(lambda_name)
        print(f"Deleted function {lambda_name}.")

    print("\nThanks for watching!")
    print("-" * 88)
```

```
if __name__ == "__main__":
    try:
        run_scenario(
            boto3.client("lambda"),
            boto3.resource("iam"),
            "lambda_handler_basic.py",
            "lambda_handler_calculator.py",
            "doc_example_lambda_calculator",
        )
    except Exception:
        logging.exception("Something went wrong with the demo!")
```

- Pour API plus de détails, consultez les rubriques suivantes dans le AWS SDKdocument de référence Python (Boto3). API
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Configurez les IAM autorisations requises pour une fonction Lambda capable d'écrire des journaux.

```
  # Get an AWS Identity and Access Management (IAM) role.
  #
  # @param iam_role_name: The name of the role to retrieve.
  # @param action: Whether to create or destroy the IAM apparatus.
  # @return: The IAM role.
  def manage_iam(iam_role_name, action)
    case action
    when 'create'
      create_iam_role(iam_role_name)
    when 'destroy'
      destroy_iam_role(iam_role_name)
    else
      raise "Incorrect action provided. Must provide 'create' or 'destroy'"
    end
  end

  private

  def create_iam_role(iam_role_name)
    role_policy = {
      'Version': '2012-10-17',
      'Statement': [
        {
          'Effect': 'Allow',
          'Principal': { 'Service': 'lambda.amazonaws.com' },
          'Action': 'sts:AssumeRole'
        }
      ]
    }
    role = @iam_client.create_role(
      role_name: iam_role_name,
      assume_role_policy_document: role_policy.to_json
    )
    @iam_client.attach_role_policy(
      {
        policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
        role_name: iam_role_name
      }
    )
    wait_for_role_to_exist(iam_role_name)
    @logger.debug("Successfully created IAM role: #{role['role']['arn']}")
    sleep(10)
```

```
    [role, role_policy.to_json]
  end

  def destroy_iam_role(iam_role_name)
    @iam_client.detach_role_policy(
      {
        policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
        role_name: iam_role_name
      }
    )
    @iam_client.delete_role(role_name: iam_role_name)
    @logger.debug("Detached policy & deleted IAM role: #{iam_role_name}")
  end

  def wait_for_role_to_exist(iam_role_name)
    @iam_client.wait_until(:role_exists, { role_name: iam_role_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  end
```

Définissez un gestionnaire Lambda qui incrémente un nombre fourni en tant que paramètre d'invocation.

```
require 'logger'

# A function that increments a whole number by one (1) and logs the result.
# Requires a manually-provided runtime parameter, 'number', which must be Int
#
# @param event [Hash] Parameters sent when the function is invoked
# @param context [Hash] Methods and properties that provide information
# about the invocation, function, and execution environment.
# @return incremented_number [String] The incremented number.
def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  log_level = ENV['LOG_LEVEL']
  logger.level = case log_level
                 when 'debug'
                   Logger::DEBUG
                 when 'info'
                   Logger::INFO
```

```
                else
                  Logger::ERROR
                end
  logger.debug('This is a debug log message.')
  logger.info('This is an info log message. Code executed successfully!')
  number = event['number'].to_i
  incremented_number = number + 1
  logger.info("You provided #{number.round} and it was incremented to
 #{incremented_number.round}")
  incremented_number.round.to_s
end
```

Zippez votre fonction Lambda dans un package de déploiement.

```
  # Creates a Lambda deployment package in .zip format.
  #
  # @param source_file: The name of the object, without suffix, for the Lambda
 file and zip.
  # @return: The deployment package.
  def create_deployment_package(source_file)
    Dir.chdir(File.dirname(__FILE__))
    if File.exist?('lambda_function.zip')
      File.delete('lambda_function.zip')
      @logger.debug('Deleting old zip: lambda_function.zip')
    end
    Zip::File.open('lambda_function.zip', create: true) do |zipfile|
      zipfile.add('lambda_function.rb', "#{source_file}.rb")
    end
    @logger.debug("Zipping #{source_file}.rb into: lambda_function.zip.")
    File.read('lambda_function.zip').to_s
  rescue StandardError => e
    @logger.error("There was an error creating deployment package:\n
 #{e.message}")
  end
```

Créez une fonction Lambda.

```
  # Deploys a Lambda function.
  #
  # @param function_name: The name of the Lambda function.
  # @param handler_name: The fully qualified name of the handler function.
```

```
 # @param role_arn: The IAM role to use for the function.
 # @param deployment_package: The deployment package that contains the function
code in .zip format.
 # @return: The Amazon Resource Name (ARN) of the newly created function.
 def create_function(function_name, handler_name, role_arn, deployment_package)
   response = @lambda_client.create_function({
                                               role: role_arn.to_s,
                                               function_name: function_name,
                                               handler: handler_name,
                                               runtime: 'ruby2.7',
                                               code: {
                                                 zip_file: deployment_package
                                               },
                                               environment: {
                                                 variables: {
                                                   'LOG_LEVEL' => 'info'
                                                 }
                                               }
                                             })
   @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
   response
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error creating #{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

Invoquez votre fonction Lambda avec des paramètres d'exécution facultatifs.

```
 # Invokes a Lambda function.
 # @param function_name [String] The name of the function to invoke.
 # @param payload [nil] Payload containing runtime parameters.
 # @return [Object] The response from the function invocation.
 def invoke_function(function_name, payload = nil)
   params = { function_name: function_name }
   params[:payload] = payload unless payload.nil?
   @lambda_client.invoke(params)
```

```
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
 end
```

Mettez la configuration de votre fonction Lambda à jour pour injecter une nouvelle variable d'environnement.

```
 # Updates the environment variables for a Lambda function.
 # @param function_name: The name of the function to update.
 # @param log_level: The log level of the function.
 # @return: Data about the update, including the status.
 def update_function_configuration(function_name, log_level)
   @lambda_client.update_function_configuration({
                                                  function_name: function_name,
                                                  environment: {
                                                    variables: {
                                                      'LOG_LEVEL' => log_level
                                                    }
                                                  }
                                                })
   @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

Mettez le code de votre fonction Lambda à jour avec un package de déploiement différent contenant un code différent.

```
 # Updates the code for a Lambda function by submitting a .zip archive that
contains
 # the code for the function.
```

```
  #
  # @param function_name: The name of the function to update.
  # @param deployment_package: The function code to update, packaged as bytes in
  #                            .zip format.
  # @return: Data about the update, including the status.
  def update_function_code(function_name, deployment_package)
    @lambda_client.update_function_code(
      function_name: function_name,
      zip_file: deployment_package
    )
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
```

Répertoriez toutes les fonctions Lambda existantes à l'aide du programme de pagination intégré.

```
  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response['functions'].each do |function|
        functions.append(function['function_name'])
      end
    end
    functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
  end
```

Supprimez une fonction Lambda spécifique.

```
  # Deletes a Lambda function.
  # @param function_name: The name of the function to delete.
  def delete_function(function_name)
    print "Deleting function: #{function_name}..."
    @lambda_client.delete_function(
      function_name: function_name
    )
    print 'Done!'.green
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
  end
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Ruby APIRéférence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Le fichier Cargo.toml avec les dépendances utilisées dans ce scénario.

```
[package]
name = "lambda-code-examples"
version = "0.1.0"
```

```
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-ec2 = { version = "1.3.0" }
aws-sdk-iam = { version = "1.3.0" }
aws-sdk-lambda = { version = "1.3.0" }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-types = { version = "1.0.1" }
clap = { version = "~4.4", features = ["derive"] }
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
tracing = "0.1.37"
serde_json = "1.0.94"
anyhow = "1.0.71"
uuid = { version = "1.3.3", features = ["v4"] }
lambda_runtime = "0.8.0"
serde = "1.0.164"
```

Un ensemble d'utilitaires qui simplifient l'appel à Lambda pour ce scénario. Ce fichier est src/ations.rs dans la caisse.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
 delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
```

```
    operation::{delete_bucket::DeleteBucketOutput,
 delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes  */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
```

```
    }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {
            InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
            InvokeArgs::Arithmetic(o, i, j) => {
                let mut map: S::SerializeMap =
 serializer.serialize_map(Some(3))?;
                map.serialize_key(&"op".to_string())?;
                map.serialize_value(&o.to_string())?;
                map.serialize_key(&"i".to_string())?;
                map.serialize_value(&i)?;
                map.serialize_key(&"j".to_string())?;
                map.serialize_value(&j)?;
                map.end()
            }
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
 behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#"{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
```

```
    ]
}"#;

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

// These unit type structs provide nominal typing on top of String parameters for
 LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
```

```
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random
name if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the
bucket.
     * If the bucket name is generated, it will be created.
     */
    pub async fn load_from_env(lambda_name: Option<String>, bucket:
Option<String>) -> Self {
        let sdk_config = aws_config::load_from_env().await;
        let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
            std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
        }));
        let role_name = RoleName(format!("{}_role", lambda_name.0));
        let (bucket, own_bucket) =
            match bucket {
                Some(bucket) => (Bucket(bucket), false),
                None => (
                    Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                        format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                    })),
                    true,
                ),
            };

        let s3_client = aws_sdk_s3::Client::new(&sdk_config);

        if own_bucket {
            info!("Creating bucket for demo: {}", bucket.0);
            s3_client
                .create_bucket()
                .bucket(bucket.0.clone())
                .create_bucket_configuration(
                    CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                            sdk_config.region().unwrap().as_ref(),
                        ))
                        .build(),
                )
```

```
                .send()
                .await
                .unwrap();
        }

        Self::new(
            aws_sdk_iam::Client::new(&sdk_config),
            aws_sdk_lambda::Client::new(&sdk_config),
            s3_client,
            lambda_name,
            role_name,
            bucket,
            OwnBucket(own_bucket),
        )
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
```

```
            .build())
    }

    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Create an IAM execution role for the managed Lambda function.
     * If the role already exists, use that instead.
```

```
    */
   async fn create_role(&self) -> Result<aws_sdk_iam::types::Role,
CreateRoleError> {
       info!("Creating execution role for function");
       let get_role = self
           .iam_client
           .get_role()
           .role_name(self.role_name.clone())
           .send()
           .await;
       if let Ok(get_role) = get_role {
           if let Some(role) = get_role.role {
               return Ok(role);
           }
       }

       let create_role = self
           .iam_client
           .create_role()
           .role_name(self.role_name.clone())
           .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
           .send()
           .await;

       match create_role {
           Ok(create_role) => match create_role.role {
               Some(role) => Ok(role),
               None => Err(CreateRoleError::generic(
                   ErrorMetadata::builder()
                       .message("CreateRole returned empty success")
                       .build(),
               )),
           },
           Err(err) => Err(err.into_service_error()),
       }
   }

   /**
    * Poll `is_function_ready` with a 1-second delay. It returns when the
function is ready or when there's an error checking the function's state.
    */
   pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
       info!("Waiting for function");
       while !self.is_function_ready(None).await? {
```

```
            info!("Function is not ready, sleeping 1s");
            tokio::time::sleep(Duration::from_secs(1)).await;
        }
        Ok(())
    }

    /**
     * Check if a Lambda function is ready to be invoked.
     * A Lambda function is ready for this scenario when its state is active and
its LastUpdateStatus is Successful.
     * Additionally, if a sha256 is provided, the function must have that as its
current code hash.
     * Any missing properties or failed requests will be reported as an Err.
     */
    async fn is_function_ready(
        &self,
        expected_code_sha256: Option<&str>,
    ) -> Result<bool, anyhow::Error> {
        match self.get_function().await {
            Ok(func) => {
                if let Some(config) = func.configuration() {
                    if let Some(state) = config.state() {
                        info!(?state, "Checking if function is active");
                        if !matches!(state, State::Active) {
                            return Ok(false);
                        }
                    }
                    match config.last_update_status() {
                        Some(last_update_status) => {
                            info!(?last_update_status, "Checking if function is
ready");
                            match last_update_status {
                                LastUpdateStatus::Successful => {
                                    // continue
                                }
                                LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                                    return Ok(false);
                                }
                                unknown => {
                                    warn!(
                                        status_variant = unknown.as_str(),
                                        "LastUpdateStatus unknown"
                                    );
```

```
                                    return Err(anyhow!(
                                        "Unknown LastUpdateStatus, fn config is
{config:?}"
                                    ));
                                }
                            }
                        }
                        None => {
                            warn!("Missing last update status");
                            return Ok(false);
                        }
                    };
                    if expected_code_sha256.is_none() {
                        return Ok(true);
                    }
                    if let Some(code_sha256) = config.code_sha256() {
                        return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
                    }
                }
            }
            Err(e) => {
                warn!(?e, "Could not get function while waiting");
            }
        }
        Ok(false)
    }

    /** Get the Lambda function with this Manager's name. */
    pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
        info!("Getting lambda function");
        self.lambda_client
            .get_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** List all Lambda functions in the current Region. */
    pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
        info!("Listing lambda functions");
```

```
        self.lambda_client
            .list_functions()
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Invoke the lambda function using calculator InvokeArgs. */
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;
```

```
        Ok(update)
    }

    /** Update the environment for a function. */
    pub async fn update_function_configuration(
        &self,
        environment: Environment,
    ) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
        info!(
            ?environment,
            "Updating environment for {}", self.lambda_name
        );
        let updated = self
            .lambda_client
            .update_function_configuration()
            .function_name(self.lambda_name.clone())
            .environment(environment)
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(updated)
    }

    /** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);
```

```
        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                info!(?location, "Skipping delete object");
                None
            };

        (delete_function, delete_role, delete_object)
    }

    pub async fn cleanup(
        &self,
        location: Option<String>,
    ) -> (
        (
            Result<DeleteFunctionOutput, anyhow::Error>,
            Result<DeleteRoleOutput, anyhow::Error>,
            Option<Result<DeleteObjectOutput, anyhow::Error>>,
        ),
        Option<Result<DeleteBucketOutput, anyhow::Error>>,
    ) {
        let delete_function = self.delete_function(location).await;

        let delete_bucket = if self.own_bucket {
```

```
            info!("Deleting bucket {}", self.bucket);
            if delete_function.2.is_none() ||
 delete_function.2.as_ref().unwrap().is_ok() {
                Some(
                    self.s3_client
                        .delete_bucket()
                        .bucket(self.bucket.clone())
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                None
            }
        } else {
            info!("No bucket to clean up");
            None
        };

        (delete_function, delete_bucket)
    }
}

/**
 * Testing occurs primarily as an integration test running the `scenario` bin
 successfully.
 * Each action relies deeply on the internal workings and state of Amazon Simple
 Storage Service (Amazon S3), Lambda, and IAM working together.
 * It is therefore infeasible to mock the clients to test the individual actions.
 */
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's
 expected by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
```

```
    }
}
```

Un binaire pour exécuter le scénario de bout en bout, en utilisant des indicateurs de ligne de commande pour contrôler certains comportements. Ce fichier est src/bin/scenario.rs dans la caisse.

```
/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
 necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction

## Scenario
A scenario runs at a command prompt and prints output to the user on the result
 of each service action. A scenario can run in one of two ways: straight through,
 printing out progress as it goes, or as an interactive question/answer script.

## Getting started with functions

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update
 its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:
_Note: Handlers don't use AWS SDK API calls._

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:
```

```
1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two
 numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO,
 WARNING, ERROR).
It logs a few things at different levels, such as:
    * DEBUG: Full event data.
    * INFO: The calculation result.
    * WARN~ING~: When a divide by zero error occurs.
    * This will be the typical `RUST_LOG` variable.


The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the
 following requirements:
    * Has an assume_role policy that grants 'lambda.amazonaws.com' the
 'sts:AssumeRole' action.
    * Attaches the 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' managed role.
    * _You must wait for ~10 seconds after the role is created before you can use
 it!_
2. Create a function (CreateFunction) for the increment handler by packaging it
 as a zip and doing one of the following:
    * Adding it with CreateFunction Code.ZipFile.
    * --or--
    * Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it
 with CreateFunction Code.S3Bucket/S3Key.
    * _Note: Zipping the file does not have to be done in code._
    * If you have a waiter, use it to wait until the function is active.
 Otherwise, call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
4. Update the function (UpdateFunctionCode) to the arithmetic handler by
 packaging it as a zip and doing one of the following:
    * Adding it with UpdateFunctionCode ZipFile.
    * --or--
    * Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/
S3Key.
5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or
 'Failed').
6. Update the environment variable by calling UpdateFunctionConfiguration and
 pass it a log level, such as:
    * Environment={'Variables': {'RUST_LOG': 'TRACE'}}
```

```
7. Invoke the function with an action from the list and a couple of values.
 Include LogType='Tail' to get logs in the result. Print the result of the
 calculation and the log.
8. [Optional] Invoke the function to provoke a divide-by-zero error and show the
 log result.
9. List all functions for the account, using pagination (ListFunctions).
10. Delete the function (DeleteFunction).
11. Delete the role.

Each step should use the function created in Service Actions to abstract calling
 the SDK.
 */

use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {
    /// The AWS Region.
    #[structopt(short, long)]
    pub region: Option<String>,

    // The bucket to use for the FunctionCode.
    #[structopt(short, long)]
    pub bucket: Option<String>,

    // The name of the Lambda function.
    #[structopt(short, long)]
    pub lambda_name: Option<String>,

    // The number to increment.
    #[structopt(short, long, default_value = "12")]
    pub inc: i32,

    // The left operand.
    #[structopt(long, default_value = "19")]
```

```
    pub num_a: i32,

    // The right operand.
    #[structopt(long, default_value = "23")]
    pub num_b: i32,

    // The arithmetic operation.
    #[structopt(short, long, default_value = "plus")]
    pub operation: Operation,

    #[structopt(long)]
    pub cleanup: Option<bool>,

    #[structopt(long)]
    pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
```

```
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([(
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                )])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");

    let invoke = manager
        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with increased logging",
    );

    let invoke = manager
        .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with divide by zero",
    );

    Ok::<(), anyhow::Error>(())
}

#[tokio::main]
async fn main() {
```

```
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
opt.bucket.clone()).await;

    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };

    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    } else {
        let delete = manager.cleanup(key).await;
        info!(?delete, "Deleted function & cleaned up resources");
    }
}
```

- Pour API plus de détails, consultez les rubriques suivantes AWS SDKpour API référence à Rust.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode

- UpdateFunctionConfiguration

## SAP ABAP

### SDKpour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
   TRY.
       "Create an AWS Identity and Access Management (IAM) role that grants AWS
Lambda permission to write to logs."
       DATA(lv_policy_document) = `{` &&
           `"Version":"2012-10-17",` &&
                 `"Statement": [` &&
                   `{` &&
                     `"Effect": "Allow",` &&
                     `"Action": [` &&
                       `"sts:AssumeRole"` &&
                     `],` &&
                     `"Principal": {` &&
                       `"Service": [` &&
                         `"lambda.amazonaws.com"` &&
                       `]` &&
                     `}` &&
                   `}` &&
                 `]` &&
               `}`.
       TRY.
           DATA(lo_create_role_output) =  lo_iam->createrole(
                   iv_rolename = iv_role_name
                   iv_assumerolepolicydocument = lv_policy_document
                   iv_description = 'Grant lambda permission to write to logs'
               ).
           MESSAGE 'IAM role created.' TYPE 'I'.
           WAIT UP TO 10 SECONDS.           " Make sure that the IAM role is
ready for use. "
```

```
          CATCH /aws1/cx_iamentityalrdyexex.
            MESSAGE 'IAM role already exists.' TYPE 'E'.
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iammalformedplydocex.
            MESSAGE 'Policy document in the request is malformed.' TYPE 'E'.
        ENDTRY.

        TRY.
            lo_iam->attachrolepolicy(
                iv_rolename  = iv_role_name
                iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole'
            ).
            MESSAGE 'Attached policy to the IAM role.' TYPE 'I'.
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
          CATCH /aws1/cx_iamplynotattachableex.
            MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

        " Create a Lambda function and upload handler code. "
        " Lambda function performs 'increment' action on a number. "
        TRY.
            lo_lmd->createfunction(
                 iv_functionname = iv_function_name
                 iv_runtime = `python3.9`
                 iv_role = lo_create_role_output->get_role( )->get_arn( )
                 iv_handler = iv_handler
                 io_code = io_initial_zip_file
                 iv_description = 'AWS Lambda code example'
             ).
            MESSAGE 'Lambda function created.' TYPE 'I'.
          CATCH /aws1/cx_lmdcodestorageexcdex.
            MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
```

```
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.

        " Verify the function is in Active state "
        WHILE lo_lmd->getfunction( iv_functionname = iv_function_name )-
>get_configuration( )->ask_state( ) <> 'Active'.
          IF sy-index = 10.
            EXIT.               " Maximum 10 seconds. "
          ENDIF.
          WAIT UP TO 1 SECONDS.
        ENDWHILE.

        "Invoke the function with a single parameter and get results."
        TRY.
            DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
              `{`  &&
                `"action": "increment",`  &&
                `"number": 10` &&
              `}`
            ).
            DATA(lo_initial_invoke_output) =  lo_lmd->invoke(
                       iv_functionname = iv_function_name
                       iv_payload = lv_json
                  ).
            ov_initial_invoke_payload = lo_initial_invoke_output->get_payload( ).
           " ov_initial_invoke_payload is returned for testing purposes. "
            DATA(lo_writer_json) = cl_sxml_string_writer=>create( type =
 if_sxml=>co_xt_json ).
            CALL TRANSFORMATION id SOURCE XML ov_initial_invoke_payload RESULT
 XML lo_writer_json.
            DATA(lv_result) = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
            MESSAGE 'Lambda function invoked.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvrequestcontex.
            MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
          CATCH /aws1/cx_lmdunsuppedmediatyp00.
            MESSAGE 'Invoke request body does not have JSON as its content type.'
 TYPE 'E'.
        ENDTRY.
```

```
        " Update the function code and configure its Lambda environment with an
 environment variable. "
        " Lambda function is updated to perform 'decrement' action also. "
        TRY.
            lo_lmd->updatefunctioncode(
                  iv_functionname = iv_function_name
                  iv_zipfile = io_updated_zip_file
              ).
            WAIT UP TO 10 SECONDS.            " Make sure that the update is
 completed. "
            MESSAGE 'Lambda function code updated.' TYPE 'I'.
          CATCH /aws1/cx_lmdcodestorageexcdex.
            MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.

        TRY.
            DATA lt_variables TYPE /aws1/
cl_lmdenvironmentvaria00=>tt_environmentvariables.
            DATA ls_variable LIKE LINE OF lt_variables.
            ls_variable-key = 'LOG_LEVEL'.
            ls_variable-value = NEW /aws1/cl_lmdenvironmentvaria00( iv_value =
 'info' ).
            INSERT ls_variable INTO TABLE lt_variables.

            lo_lmd->updatefunctionconfiguration(
                  iv_functionname = iv_function_name
                  io_environment = NEW /aws1/cl_lmdenvironment( it_variables =
 lt_variables )
              ).
            WAIT UP TO 10 SECONDS.            " Make sure that the update is
 completed. "
            MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourceconflictex.
            MESSAGE 'Resource already exists or another operation is in
 progress.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.
```

```
        "Invoke the function with new parameters and get results. Display the
 execution log that's returned from the invocation."
        TRY.
            lv_json = /aws1/cl_rt_util=>string_to_xstring(
              `{`  &&
                `"action": "decrement",`  &&
                `"number": 10` &&
              `}`
            ).
            DATA(lo_updated_invoke_output) =  lo_lmd->invoke(
                       iv_functionname = iv_function_name
                       iv_payload = lv_json
                  ).
            ov_updated_invoke_payload = lo_updated_invoke_output->get_payload( ).
           " ov_updated_invoke_payload is returned for testing purposes. "
            lo_writer_json = cl_sxml_string_writer=>create( type =
 if_sxml=>co_xt_json ).
            CALL TRANSFORMATION id SOURCE XML ov_updated_invoke_payload RESULT
 XML lo_writer_json.
            lv_result = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
            MESSAGE 'Lambda function invoked.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvrequestcontex.
            MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
          CATCH /aws1/cx_lmdunsuppedmediatyp00.
            MESSAGE 'Invoke request body does not have JSON as its content type.'
 TYPE 'E'.
        ENDTRY.

        " List the functions for your account. "
        TRY.
            DATA(lo_list_output) = lo_lmd->listfunctions( ).
            DATA(lt_functions) = lo_list_output->get_functions( ).
            MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        ENDTRY.

        " Delete the Lambda function. "
```

```
        TRY.
            lo_lmd->deletefunction( iv_functionname = iv_function_name ).
            MESSAGE 'Lambda function deleted.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.

        " Detach role policy. "
        TRY.
            lo_iam->detachrolepolicy(
                iv_rolename  = iv_role_name
                iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole'
            ).
            MESSAGE 'Detached policy from the IAM role.' TYPE 'I'.
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
          CATCH /aws1/cx_iamplynotattachableex.
            MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

        " Delete the IAM role. "
        TRY.
            lo_iam->deleterole( iv_rolename = iv_role_name ).
            MESSAGE 'IAM role deleted.' TYPE 'I'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

      CATCH /aws1/cx_rt_service_generic INTO lo_exception.
        DATA(lv_error) = lo_exception->get_longtext( ).
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Pour API plus de détails, consultez les rubriques suivantes à AWS SDKtitre de SAP ABAP API référence.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction Lambda à partir d'un navigateur

L'exemple de code suivant montre comment invoquer une AWS Lambda fonction depuis un navigateur.

JavaScript

SDKpour JavaScript (v2)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda

SDKpour JavaScript (v3)

Vous pouvez créer une application basée sur un navigateur qui utilise une AWS Lambda fonction pour mettre à jour une table Amazon DynamoDB avec les sélections des utilisateurs. Cette application utilise la AWS SDK for JavaScript version 3.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Transformez les données de votre application avec S3 Object Lambda

L'exemple de code suivant montre comment transformer des données pour votre application avec S3 Object Lambda.

.NET

AWS SDK for .NET

Montre comment ajouter du code personnalisé aux GET demandes S3 standard afin de modifier l'objet demandé extrait de S3 afin que l'objet réponde aux besoins du client ou de l'application demandeur.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Lambda
- Amazon S3

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Utiliser API Gateway pour appeler une fonction Lambda

Les exemples de code suivants montrent comment créer une AWS Lambda fonction invoquée par Amazon API Gateway.

Java

SDKpour Java 2.x

Montre comment créer une AWS Lambda fonction à l'aide du moteur d'exécution Lambda Java. API Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui scanne une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (SNSAmazon) pour envoyer un SMS à vos employés afin de les féliciter à l'occasion de leur premier anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDKpour JavaScript (v3)

Montre comment créer une AWS Lambda fonction à l'aide du moteur d'exécution Lambda JavaScript . API Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une fonction Lambda invoquée par Amazon API Gateway qui scanne une table Amazon DynamoDB à la recherche d'anniversaires professionnels et utilise Amazon Simple Notification Service (SNSAmazon)

pour envoyer un SMS à vos employés afin de les féliciter à l'occasion de leur premier anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- APIPasserelle
- DynamoDB
- Lambda
- Amazon SNS

Python

SDKpour Python (Boto3)

Cet exemple montre comment créer et utiliser un Amazon API Gateway REST API qui cible une AWS Lambda fonction. Le gestionnaire Lambda montre comment effectuer un routage en fonction de HTTP méthodes, comment obtenir des données à partir de la chaîne de requête, de l'en-tête et du corps de la requête, et comment renvoyer une réponse. JSON

- Déploier une fonction Lambda.
- Créez une API passerelle RESTAPI.
- Créez une REST ressource qui cible la fonction Lambda.
- Accordez l'autorisation de laisser API Gateway invoquer la fonction Lambda.
- Utilisez le package Requests pour envoyer des demandes au RESTAPI.
- Nettoyer toutes les ressources créées lors de la démonstration.

Il est préférable de visionner cet exemple sur GitHub. Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- APIPasserelle
- Lambda

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Utiliser les fonctions Step Functions pour invoquer des fonctions Lambda

L'exemple de code suivant montre comment créer une machine à AWS Step Functions états qui invoque des AWS Lambda fonctions en séquence.

Java

SDKpour Java 2.x

Montre comment créer un flux de travail AWS sans serveur en utilisant AWS Step Functions et le AWS SDK for Java 2.x. Chaque étape du flux de travail est implémentée à l'aide d'une AWS Lambda fonction.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Utilisent des événements planifiés pour appeler une fonction Lambda

Les exemples de code suivants montrent comment créer une AWS Lambda fonction invoquée par un événement EventBridge planifié par Amazon.

Java

SDKpour Java 2.x

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide du moteur d'exécution Lambda Java. API Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDKpour JavaScript (v3)

Montre comment créer un événement EventBridge planifié Amazon qui invoque une AWS Lambda fonction. Configurez EventBridge pour utiliser une expression cron afin de planifier le moment où la fonction Lambda est invoquée. Dans cet exemple, vous créez une fonction Lambda à l'aide du moteur d'exécution Lambda. JavaScript API Cet exemple fait appel à différents AWS services pour réaliser un cas d'utilisation spécifique. Cet exemple montre comment créer une application qui envoie un message texte mobile à vos employés pour les féliciter à leur date d'anniversaire.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Cet exemple est également disponible dans le AWS SDK for JavaScript guide du développeur v3.

Les services utilisés dans cet exemple

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDKpour Python (Boto3)

Cet exemple montre comment enregistrer une AWS Lambda fonction en tant que cible d'un EventBridge événement Amazon planifié. Le gestionnaire Lambda écrit un message convivial et les données complètes de l'événement dans Amazon CloudWatch Logs pour une récupération ultérieure.

- Déploie une fonction Lambda.
- Crée un événement EventBridge planifié et fait de la fonction Lambda la cible.
- Accorde l'autorisation de laisser EventBridge invoquer la fonction Lambda.
- Imprime les dernières données des CloudWatch journaux pour afficher le résultat des appels planifiés.
- Nettoie toutes les ressources créées lors de la démonstration.

Il est préférable de visionner cet exemple sur GitHub. Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- CloudWatch Journaux
- EventBridge
- Lambda

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Rédigez des données d'activité personnalisées à l'aide d'une fonction Lambda après l'authentification de l'utilisateur Amazon Cognito à l'aide d'un AWS SDK

L'exemple de code suivant montre comment écrire des données d'activité personnalisées avec une fonction Lambda après l'authentification de l'utilisateur Amazon Cognito.

- Utilisez les fonctions d'administrateur pour ajouter un utilisateur à un groupe d'utilisateurs.
- Configurez un groupe d'utilisateurs pour appeler une fonction Lambda pour le `PostAuthentication` déclencheur.
- Connectez le nouvel utilisateur à Amazon Cognito.
- La fonction Lambda écrit des informations personnalisées dans des CloudWatch journaux et dans une table DynamoDB.
- Obtenez et affichez des données personnalisées à partir de la table DynamoDB, puis nettoyez les ressources.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
// ActivityLog separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ActivityLog struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}
```

```
// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) ActivityLog {
 scenario := ActivityLog{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
 credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
 tableName string) (string, string) {
 log.Println("To facilitate this example, let's add a user to the user pool using
 administrator privileges.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 user := users.Users[0]
 log.Printf("Adding known user %v to the user pool.\n", user.UserName)
 err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
 user.UserEmail)
 if err != nil {
  panic(err)
 }
 pwSet := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !pwSet {
  log.Printf("\nSetting password for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
 password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
```

```
   } else {
    panic(err)
   }
  } else {
   pwSet = true
  }
 }

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
 PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
 string, activityLogArn string) {
 log.Println("Let's add a Lambda function to handle the PostAuthentication
 trigger from Cognito.\n" +
  "This trigger happens after a user is authenticated, and lets your function
 take action, such as logging\n" +
  "the outcome.")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
 aws.String(activityLogArn)})
 if err != nil {
  panic(err)
 }
 runner.resources.triggers = append(runner.resources.triggers,
 actions.PostAuthentication)
 log.Printf("Lambda function %v added to user pool %v to handle
 PostAuthentication Cognito trigger.\n",
  activityLogArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string,
 userName string, password string) {
 log.Printf("Now we'll sign in user %v and check the results in the logs and the
 DynamoDB table.", userName)
 runner.questioner.Ask("Press Enter when you're ready.")
```

```
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Println("Sign in successful.",
  "The PostAuthentication Lambda handler writes custom information to CloudWatch
 Logs.")

 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
 table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
 string, userName string) {
 log.Println("The PostAuthentication handler also writes login data to the
 DynamoDB table.")
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 for _, user := range users.Users {
  if user.UserName == userName {
   log.Println("The last login info for the user in the known users table is:")
   log.Printf("\t%+v", *user.LastLogin)
  }
 }
 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")
```

```
 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Manipulez le `PostAuthentication` déclencheur avec une fonction Lambda.

```
const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
 format.
type LoginInfo struct {
 UserPoolId string `dynamodbav:"UserPoolId"`
 ClientId   string `dynamodbav:"ClientId"`
 Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string    `dynamodbav:"UserName"`
```

```
 UserEmail string    `dynamodbav:"UserEmail"`
 LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PostAuthentication event by writing custom data to
 the logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
 event events.CognitoEventUserPoolsPostAuthentication)
 (events.CognitoEventUserPoolsPostAuthentication, error) {
 log.Printf("Received post authentication trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName:  event.UserName,
  UserEmail: event.Request.UserAttributes["email"],
  LastLogin: LoginInfo{
   UserPoolId: event.UserPoolID,
   ClientId:   event.CallerContext.ClientID,
   Time:       time.Now().Format(time.UnixDate),
  },
 }
 // Write to CloudWatch Logs.
 fmt.Printf("%#v", user)

 // Also write to an external system. This examples uses DynamoDB to demonstrate.
 userMap, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
 } else if len(userMap) == 0 {
  log.Printf("User info marshaled to an empty map.")
```

```
 } else {
  _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
   Item:      userMap,
   TableName: aws.String(tableName),
  })
  if err != nil {
   log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
  } else {
   log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
  }
 }

 return event, nil
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Créez une structure qui exécute des tâches courantes.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}
```

```
// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
```

```
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
```

```
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Créez une structure qui englobe les actions Amazon Cognito.

```
type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
```

```
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
```

```
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}
```

```
// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}
```

```
// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
```

```
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Créez une structure qui enveloppe les actions DynamoDB.

```
// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
```

```
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
```

```
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Créez une structure qui englobe les actions CloudWatch Logs.

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
```

```
  events = output.Events
 }
 return events, err
}
```

Créez une structure qui englobe les actions. AWS CloudFormation

```
// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Nettoyez les ressources.

```
// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
```

```
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
```

```
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- Pour API plus de détails, consultez les rubriques suivantes dans AWS SDK for Go APIRéférence.
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Exemples d'utilisation de Lambda sans serveur AWS SDKs

Les exemples de code suivants montrent comment utiliser Lambda avec. AWS SDKs

Exemples

- Connexion à une RDS base de données Amazon dans une fonction Lambda
- Invoquer une fonction Lambda à partir d’un déclencheur Kinesis
- Invoquer une fonction Lambda à partir d'un déclencheur DynamoDB
- Invoquer une fonction Lambda depuis un déclencheur Amazon DocumentDB

- Invoquer une fonction Lambda depuis un déclencheur Amazon MSK
- Invoquer une fonction lambda à partir d'un déclencheur Amazon S3
- Invoquer une fonction Lambda depuis un déclencheur Amazon SNS
- Invoquer une fonction Lambda depuis un déclencheur Amazon SQS
- Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis
- Signalement des défaillances d'éléments de lot pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB
- Signalement des défaillances d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

## Connexion à une RDS base de données Amazon dans une fonction Lambda

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui se connecte à une RDS base de données. La fonction effectue une simple demande de base de données et renvoie le résultat.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Go.

```
/*
Golang v2 code here.
*/

package main

import (
```

```
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }
```

```
 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
```

```
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
 RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
 event, Context context) {
        APIGatewayProxyResponseEvent response = new
 APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                    System.getenv("ProxyHostName"),
                    System.getenv("Port"),
                    System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
 DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
 token);
                 PreparedStatement statement =
 connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);
```

```
                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
                .region(Region.of(System.getenv("AWS_REGION")))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .resourceArn(System.getenv("ProxyHostName"))
                .secretArn(System.getenv("DBUserName"))
                .database(System.getenv("DBName"))
                .sql("SELECT 'RDS IAM Authentication'")
                .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
 rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {
```

```
  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
 the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
```

```
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
 body: string }> => {
    // Execute database flow
    const result = await dbOps();
```

```
    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une base de RDS données Amazon dans une fonction Lambda à l'aide de. PHP

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
```

```
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']


# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
```

```
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'}  # Ensure you have the CA bundle for SSL
 connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}")  # Return an error message if an exception
 occurs
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
  port = ENV['Port']           # 3306
  user = ENV['DBUser']
  region = ENV['DBRegion']     # 'us-east-1'
```

```
  db_name = ENV['DBName']

  credentials = Aws::Credentials.new(
    ENV['AWS_ACCESS_KEY_ID'],
    ENV['AWS_SECRET_ACCESS_KEY'],
    ENV['AWS_SESSION_TOKEN']
  )
  rds_client = Aws::RDS::AuthTokenGenerator.new(
    region: region,
    credentials: credentials
  )

  token = rds_client.auth_token(
    endpoint: endpoint+ ':' + port,
    user_name: user,
    region: region
  )

  begin
    conn = Mysql2::Client.new(
      host: endpoint,
      username: user,
      password: token,
      port: port,
      database: db_name,
      sslca: '/var/task/global-bundle.pem',
      sslverify: true,
      enable_cleartext_plugin: true
    )
    a = 3
    b = 2
    result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
    puts result
    conn.close
    {
      statusCode: 200,
      body: result.to_json
    }
  rescue => e
    puts "Database connection failed due to #{e}"
  end
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Connexion à une RDS base de données Amazon dans une fonction Lambda à l'aide de Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
```

```
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
 aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
 SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
```

```
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    }))
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Invoquer une fonction Lambda à partir d'un déclencheur Kinesis

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux Kinesis. La fonction récupère la charge utile Kinesis, décode à partir de Base64 et enregistre le contenu de l'enregistrement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
```

```
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
```

```
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
```

```
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
```

```
  return data;
}
```

Utilisation d'un événement Kinesis avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
```

```
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
```

```
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Kinesis avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
```

```
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Kinesis avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
```

```
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un événement Kinesis avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
```

```
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction Lambda à partir d'un déclencheur DynamoDB

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements d'un flux DynamoDB. La fonction récupère la charge utile DynamoDB et enregistre le contenu de l'enregistrement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
```

```
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
```

```
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement DynamoDB avec Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consommation d'un événement DynamoDB avec Lambda en utilisant. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
```

```
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda en utilisant. PHP

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
```

```
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement DynamoDB avec Lambda à l'aide de Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
```

```
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

### Rust

#### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement DynamoDB avec Lambda à l'aide de Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Invoquer une fonction Lambda depuis un déclencheur Amazon DocumentDB

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements provenant d'un flux de modifications DocumentDB. La fonction récupère la charge utile DocumentDB et enregistre le contenu de l'enregistrement.

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
```

```
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
```

```
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de. PHP

```
<?php

require __DIR__.'/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
```

```
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
 "\n";
    }
}
return new DocumentDBEventHandler();
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement Amazon DocumentDB avec Lambda à l'aide de Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement Amazon DocumentDB avec Lambda à l'aide de Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
 Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
```

```
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())

}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction Lambda depuis un déclencheur Amazon MSK

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements d'un cluster AmazonMSK. La fonction récupère la MSK charge utile et enregistre le contenu de l'enregistrement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda en utilisant. NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to
 process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
 logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);
```

```
                Console.WriteLine("Message:" + valueText);
            }
        }
    }


}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
```

```
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
 event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);
```

```
                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda en utilisant. JavaScript

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda en utilisant. PHP

```
<?php
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

// using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kafka\KafkaEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();
```

```
        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }

            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Python.

```
import base64

def lambda_handler(event, context):
    # Iterate through keys
```

```
    for key in event['records']:
        print('Key:', key)
        # Iterate through records
        for record in event['records'][key]:
            print('Record:', record)
            # Decode base64
            msg = base64.b64decode(record['value']).decode('utf-8')
            print('Message:', msg)
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un MSK événement Amazon avec Lambda à l'aide de Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
  # Iterate through keys
  event['records'].each do |key, records|
    puts "Key: #{key}"

    # Iterate through records
    records.each do |record|
      puts "Record: #{record}"

      # Decode base64
      msg = Base64.decode64(record['value'])
      puts "Message: #{msg}"
    end
  end
end
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction lambda à partir d'un déclencheur Amazon S3

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par le chargement d'un objet vers un compartiment S3. La fonction extrait le nom du compartiment S3 et la clé de l'objet à partir du paramètre d'événement et appelle Amazon S3 API pour récupérer et enregistrer le type de contenu de l'objet.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
```

```
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request -
 {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
```

```
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
 com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNotifi

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
          S3EventNotificationRecord record = s3event.getRecords().get(0);
          String srcBucket = record.getS3().getBucket().getName();
          String srcKey = record.getS3().getObject().getUrlDecodedKey();

          S3Client s3Client = S3Client.builder().build();
          HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
 srcKey);

          logger.info("Successfully retrieved " + srcBucket + "/" + srcKey + " of
 type " + headObject.contentType());

          return "Ok";
        } catch (Exception e) {
          throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String bucket,
 String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
                .bucket(bucket)
                .key(key)
                .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```

## JavaScript

### SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. JavaScript

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g,
 ' '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Consommation d'un événement S3 avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
```

```
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda en utilisant. PHP

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';
```

```
class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
 $bucket . '. Make sure they exist and your bucket is in the same region as this
 function.' . "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],
 encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and
 your bucket is in the same region as this function.'.format(key, bucket))
        raise e
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement S3 avec Lambda à l'aide de Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']['key'],
 Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they exist
 and your bucket is in the same region as this function."
    raise e
  end
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement S3 avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};


/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
```

```
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
 SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket
 name to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
 exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction Lambda depuis un déclencheur Amazon SNS

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages provenant d'une SNS rubrique. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
 {record.Sns.Message}");
```

```
            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

## Go

### SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
```

```
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un SNS événement avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;


import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;
```

```
    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SNS événement avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consommation d'un SNS événement avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
```

```
    throw err;
  }
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Consommation d'un SNS événement avec Lambda en utilisant. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
```

```
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
 marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here
```

```
    except Exception as e:
        print("An error occurred")
        raise e
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
  raise
end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SNS événement avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features
 = ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Invoquer une fonction Lambda depuis un déclencheur Amazon SQS

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages provenant d'une SQS file d'attente. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda en utilisant. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un SQS événement avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
```

```
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda en utilisant. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consommation d'un SQS événement avec Lambda en utilisant. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un SQS événement avec Lambda en utilisant. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
```

```
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommer un SQS événement avec Lambda en utilisant Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
```

```
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis

Les exemples de code suivants montrent comment implémenter une réponse par lots partielle pour les fonctions Lambda qui reçoivent des événements d'un flux Kinesis. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
```

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
```

```
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
```

```
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
```

```
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
```

```
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot Kinesis avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end
```

```
def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances des éléments d'un lot Kinesis avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
```

```
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
```

```
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Signalement des défaillances d'éléments de lot pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

Les exemples de code suivants montrent comment implémenter une réponse par lots partielle pour les fonctions Lambda qui reçoivent des événements d'un flux DynamoDB. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
```

```
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
```

```
}
```

Go

SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }
```

```
 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
```

```
public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
```

```
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de. PHP

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler des défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler des défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot DynamoDB avec Lambda à l'aide de Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
```

```
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

## Signalement des défaillances d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

Les exemples de code suivants montrent comment implémenter une réponse par lots partielle pour les fonctions Lambda qui reçoivent des événements d'une SQS file d'attente. La fonction signale

les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

.NET

AWS SDK for .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
```

```
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

## Go

### SDKpour Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDKpour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {

        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(messageId));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDKpour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. JavaScript

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
```

```
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
```

```
    }
    console.log(`Processed message ${record.body}`);
}
```

## PHP

### SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

#### Signaler les défaillances d'articles SQS par lots avec Lambda à l'aide de. PHP

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
```

```
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDKpour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
```

```
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                # process message
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

## Ruby

### SDKpour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments d'un SQS lot avec Lambda à l'aide de Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
```

```
    return sqs_batch_response
  end
end
```

## Rust

### SDKpour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments SQS par lots avec Lambda à l'aide de Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
```

```
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

Pour obtenir la liste complète des guides AWS SDK de développement et des exemples de code, consultezUtilisation de Lambda avec un AWS SDK. Cette rubrique inclut également des informations sur la mise en route et des détails sur SDK les versions précédentes.

# Quotas Lambda

> **Important**
>
> Comptes AWS Les nouveautés ont réduit la simultanéité et les quotas de mémoire. AWS augmente automatiquement ces quotas en fonction de votre utilisation.

## calcul et stockage

Lambda définit des quotas pour les ressources de calcul et de stockage que vous pouvez utiliser afin d'exécuter et de stocker des fonctions. Les quotas d'exécutions simultanées et de stockage s'appliquent par Région AWS. Les quotas d'interface réseau élastique (ENI) s'appliquent par cloud privé virtuel (VPC), quelle que soit la région. Les quotas suivants peuvent être augmentés par rapport à leurs valeurs par défaut. Pour de plus amples informations, veuillez consulter Demande d'augmentation de quota dans le Guide de l'utilisateur Service Quotas.

| Ressource | Quota par défaut | Peut être augmentée jusqu'à |
|---|---|---|
| Exécutions simultanées | 1 000 | Dizaines de milliers |
| Stockage pour les fonctions téléchargées (archives de fichiers .zip) et les couches. Chaque version de fonction et de couche consomme de l'espace de stockage.<br><br>Pour respecter les bonnes pratiques en matière d'administration du stockage du code, veuillez consulter Surveillance du stockage de code Lambda dans Serverless Land. | 75 Go | Téra-octets |
| Stockage des fonctions définies en tant qu'images de conteneur. Ces images sont stockées sur AmazonECR. | Consultez la section Quotas ECR de service Amazon. | |

| Ressource | Quota par défaut | Peut être augmentée jusqu'à |
| --- | --- | --- |
| Interfaces réseau élastiques par cloud privé virtuel (VPC)<br><br>**Note**<br>Ce quota est partagé avec d'autres services, tels qu'Amazon Elastic File System (AmazonEFS). Consultez la section VPCQuotas Amazon. | 500 | Milliers |

Pour en savoir plus sur la simultanéité et la manière dont Lambda met à l'échelle la simultanéité de votre fonction en réponse au trafic, consultez Comprendre la mise à l'échelle des fonctions Lambda.

## Configuration, déploiement et exécution de fonction

Les quotas suivants s'appliquent à la configuration, au déploiement et à l'exécution des fonctions. Sauf indication contraire, ils ne peuvent pas être modifiés.

**Note**

La documentation Lambda, les messages de journal et la console utilisent l'abréviation Mo (plutôt que Mio) pour faire référence à 1024 Ko.

| Ressource | Quota |
| --- | --- |
| Allocation de mémoire des fonctions | 128 Mo à 10 240 Mo, par incréments de 1 Mo<br><br>Remarque : Lambda alloue l'CPUénergie proportionnellement à la quantité de mémoire configurée. |

| Ressource | Quota |
| --- | --- |
| | Vous pouvez augmenter ou diminuer la mémoire et la CPU puissance allouées à votre fonction à l'aide du paramètre Mémoire (Mo). À 1 769 Mo, une fonction a l'équivalent d'un v. CPU |
| Délai d’expiration des fonctions | 900 secondes (15 minutes) |
| Variables d’environnement des fonctions | 4 Ko, pour toutes les variables d’environnement associées à la fonction, au total |
| stratégie de fonction basée sur les ressources | 20 Ko |
| Couches de fonctions | cinq couches |
| Limite d’échelle de simultanéité des fonctions | Pour chaque fonction, 1 000 environnements d’exécution toutes les 10 secondes |
| Charge utile d’invocation (demande et réponse) | 6 Mo chacun pour la demande et la réponse (synchrone)<br><br>20 Mo pour chaque réponse diffusée (synchrone). La taille de la charge utile pour les réponses diffusées en continu peut être augmentée par rapport aux valeurs par défaut. Contactez-nous AWS Support pour en savoir plus.)<br><br>256 Ko (asynchrone)<br><br>1 Mo pour la taille totale combinée de la ligne de demande et des valeurs d'en-tête |

| Ressource | Quota |
| --- | --- |
| Bande passante pour les réponses diffusées | Non plafonné pour les 6 premiers Mo de la réponse de votre fonction<br><br>Pour les réponses supérieures à 6 Mo, 2 MBps pour le reste de la réponse |
| Taille du package de déploiement (archive de fichiers .zip) | 50 Mo (compressés, lors du téléchargement via le API Lambda SDKs ou). Importez des fichiers plus volumineux avec Amazon S3.<br><br>3 Mo (en cas de téléchargement via la console Lambda)<br><br>250 Mo Taille maximale du contenu d'un package de déploiement, y compris les couches et les environnements d'exécution personnalisés. (dézippé) |
| Taille des paramètres de l'image de conteneur | 16 Ko |
| Taille du package du code de l'image de conteneur | 10 Go (taille maximale de l'image non compressée, comprenant toutes les couches) |
| Événements de test (éditeur de console) | 10 |
| Stockage dans le répertoire `/tmp` | Entre 512 Mo et 10 240 Mo par incréments de 1 Mo |
| Descripteurs de fichier | 1,024 |
| Processus/threads d'exécution | 1,024 |

# Requêtes Lambda API

Les quotas suivants sont associés aux requêtes LambdaAPI.

| Ressource | Quota |
| --- | --- |
| Demandes d'invocation par fonction par région (synchrone) | Chaque instance de votre environnement d'exécution peut servir jusqu'à 10 demandes par seconde. En d'autres termes, la limite d'invocation totale correspond à 10 fois votre limite de simultanéité. Consultez Comprendre la mise à l'échelle des fonctions Lambda. |
| Demandes d'invocation par fonction par région (asynchrone) | Chaque instance de votre environnement d'exécution peut servir un nombre illimité de demandes. En d'autres termes, la limite d'invocation totale est basée uniquement sur la simultanéité disponible pour votre fonction. Consultez Comprendre la mise à l'échelle des fonctions Lambda. |
| Demandes d'invocation par version ou alias de fonction (demandes par seconde) | 10 x simultanéité provisionnée<br><br>**Note**<br>Ce quota s'applique uniquement aux fonctions qui utilisent la simultanéité provisionnée. |
| GetFunctionAPIdemandes | 100 requêtes par seconde. Ne peut pas être augmenté. |

| Ressource | Quota |
| --- | --- |
| GetPolicyAPIdemandes | 15 requêtes par seconde. Ne peut pas être augmenté. |
| Reste des API demandes du plan de contrôle (à l'exclusion de l'invocation et GetFunction des GetPolicy demandes) | 15 demandes par seconde en tout APIs (et non 15 demandes par seconde par secondeAPI). Ne peut pas être augmenté. |

## Autres services

Les quotas pour d'autres services, tels que AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda @Edge) et Amazon Virtual Private Cloud (AmazonVPC), peuvent avoir un impact sur vos fonctions Lambda. Pour plus d'informations, consultez Quotas Service AWS dans le Référence générale d'Amazon Web Services et Invoquer Lambda avec des événements provenant d'autres services AWS.

# Historique du document

Le tableau suivant décrit les modifications importantes apportées au Guide du développeur d'AWS Lambda après mai 2018. Pour recevoir des notifications concernant les mises à jour de cette documentation, abonnez-vous au RSSflux.

| Modification | Description | Date |
| --- | --- | --- |
| Support pour SnapStart les nouvelles régions | Lambda SnapStartest désormais disponible dans les régions suivantes : Europe (Espagne), Europe (Zurich), Asie-Pacifique (Melbourne), Asie-Pacifique (Hyderabad) et Moyen-Orient (). UAE | 12 janvier 2024 |
| AWS mises à jour des politiques gérées | Service Quotas a mis à jour une politique AWS gérée existante (`AWSLambda VPCAccessExecution Role` ). | 5 janvier 2024 |
| Exécution Python 3.12 | Lambda prend désormais en charge Python 3.12 en tant qu'exécution managée et image de base de conteneur . Pour plus d'informations, consultez le runtime Python 3.12 désormais disponible AWS Lambda sur le AWS Compute Blog. | 14 décembre 2023 |
| Environnement d'exécution Java 21 | Lambda prend désormais en charge Java 21 en tant qu'exécution managée et image de base de conteneur (`java21`). | 16 novembre 2023 |

| | | |
|---|---|---|
| Environnement d’exécution Node.js 20 | Lambda prend désormais en charge Node.js 20 en tant qu’exécution managée et image de base de conteneur (`nodejs20.x`). Pour plus d'informations, consultez le runtime Node.js 20.x désormais disponible AWS Lambda sur le AWS Compute Blog. | 14 novembre 2023 |
| Exécution provided.al2023 | Lambda prend désormais en charge Amazon Linux 2023 en tant qu’environnement d’exécution géré et image de base de conteneur. Pour plus d'informations, consultez Présentation de l'environnement d'exécution Amazon Linux 2023 AWS Lambda sur le blog AWS Compute. | 9 novembre 2023 |
| IPv6prise en charge des sous-réseaux à double pile | Lambda prend désormais en charge le IPv6 trafic sortant vers des sous-réseaux à double pile. Pour plus d'informations, consultez le IPv6support. | 12 octobre 2023 |

| | | |
|---|---|---|
| Test de fonctions et d'applications sans serveur | Découvrez les techniques de débogage et d'automatisation des tests des fonctions sans serveur dans le cloud. Un chapitre sur les tests et des ressources sont désormais inclus dans les sections consacrées aux langages Python et Typescript. Pour en savoir plus, veuillez consulter la rubrique Tester des fonctions et des applications sans serveur. | 16 juin 2023 |
| Exécution Ruby 3.2 | Lambda prend désormais en charge une nouvelle exécution pour Ruby 3.2. Pour plus d'informations, veuillez consulter la rubrique Création de fonctions Lambda avec Ruby. | 7 juin 2023 |
| Streaming des réponses | Lambda prend désormais en charge le streaming des réponses des fonctions. Pour plus d'informations, consultez Configuration d'une fonction Lambda pour le streaming des réponses (français non garanti). | 6 avril 2023 |
| Métriques d'invocations asynchrones | Lambda publie des métriques d'invocations asynchrones. Pour plus d'informations, consultez Métriques d'invocations asynchrones. | 9 février 2023 |

| | | |
|---|---|---|
| Contrôles de version de l'environnement d'exécution | Lambda publie de nouvelles versions de l'environnement d'exécution qui incluent des mises à jour de sécurité, des corrections de bogues et de nouvelles fonctions . Vous pouvez désormais contrôler quand vos fonctions sont mises à jour vers les nouvelles versions de l'environnement d'exécution. Pour plus d'informations, consultez Mises à jour de l'environnement d'exécution Lambda (français non garanti). | 23 janvier 2023 |
| Lambda SnapStart | Utilisez Lambda SnapStart pour réduire le temps de démarrage des fonctions Java sans fournir de ressources supplémentaires ni implémenter des optimisations de performances complexes . Pour plus d'informations, consultez la section Amélioration des performances de démarrage avec Lambda SnapStart. | 28 novembre 2022 |

| | | |
|---|---|---|
| Environnement d'exécution Node.js 18 | Lambda prend désormais en charge un nouvel environnement d'exécution (runtime) pour Node.js 18. Node.js 18 utilise Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js. | 18 novembre 2022 |
| lambda : clé de SourceFunctionArn condition | Pour une AWS ressource, la clé de `lambda:SourceFunctionArn` condition filtre l'accès à la ressource par le biais ARN d'une fonction Lambda. Pour plus d'informations, consultez utilisation des identifiants d'un environnement d'exécution Lambda. | 1et juillet 2022 |
| Environnement d'exécution Node.js 16 | Lambda prend désormais en charge un nouvel environnement d'exécution (runtime) pour Node.js 16. Node.js 16 utilise Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js. | 11 mai 2022 |
| Fonction Lambda URLs | Lambda prend désormais en charge les fonctionsURLs, qui sont des points de terminaison HTTP (S) dédiés aux fonctions Lambda. Pour plus de détails, consultez la section Fonction Lambda. URLs | 6 avril 2022 |

| | | |
|---|---|---|
| Événements de test partagés dans la AWS Lambda console | Lambda prend désormais en charge le partage d'événements de test avec d'autres utilisateurs dans le même Compte AWS. Pour plus de détails, consultez Test des fonctions Lambda dans la console. | 16 mars 2022 |
| PrincipalOrgId dans les politiques basées sur les ressources | Lambda prend désormais en charge l'octroi d'autorisations à une organisation dans AWS Organizations. Pour plus d'informations, consultez Utilisation de stratégies basées sur les ressources pour AWS Lambda. | 11 mars 2022 |
| . NET6 temps d'exécution | Lambda prend désormais en charge un nouveau runtime pour. NET6. Pour en savoir plus, consultez Environnements d'exécution Lambda. | 23 février 2022 |
| Filtrage des événements pour les sources d'événements Kinesis, DynamoDB et Amazon SQS | Lambda prend désormais en charge le filtrage des événements pour les sources d'événements Kinesis, DynamoDB et Amazon. SQS Pour plus de détails, consultez Filtrage des événements Lambda. | 24 novembre 2021 |

| | | |
|---|---|---|
| TLSauthentification m pour Amazon MSK et sources d'événements Apache Kafka autogérées | Lambda prend désormais en charge TLS l'authentification m pour Amazon MSK et les sources d'événements Apache Kafka autogérées. Pour plus de détails, consultez la section Utilisation de Lambda avec Amazon. MSK | 19 novembre 2021 |
| Lambda sur Graviton2 | Lambda prend désormais en charge Graviton2 pour les fonctions utilisant l'architecture arm64. Pour plus de détails, consultez Architectures de l'ensemble des instructions Lambda. | 29 septembre 2021 |
| Exécution Python 3.9 | Lambda prend désormais en charge un nouveau runtime pour Python 3.9. Pour en savoir plus, consultez Environnements d'exécution Lambda. | 16 août 2021 |
| Nouvelles versions d'environnements d'exécution pour Node.js, Python et Java | De nouvelles versions d'environnements d'exécution sont disponibles pour Node.js, Python et Java. Pour en savoir plus, consultez Environnements d'exécution Lambda. | 21 juillet 2021 |

| | | |
|---|---|---|
| Prise en charge de RabbitMQ en tant que source d'événements sur Lambda | Lambda prend désormais en charge Amazon MQ pour RabbitMQ en tant que source d'événements. Amazon MQ est un service d'agent de messages géré pour Apache ActiveMQ, et RabbitMQ, qui facilite la configuration et l'utilisation d'agents de messages dans le cloud. Pour plus de détails, consultez Utilisation de Lambda avec Amazon MQ. | 7 juillet 2021 |
| SASL/PLAINauthentification pour Kafka autogéré sur Lambda | SASL/PLAINest désormais un mécanisme d'authentification pris en charge pour les sources d'événements Kafka autogérées sur Lambda Les clients utilisant déjàSASL/PLAINsur leur cluster Kafka autogéré peuvent désormais facilement utiliser Lambda pour créer des applications grand public sans avoir à modifier leur mode d'authentification. Pour plus d'informations, consultez Utilisation de Lambda avec Apache Kafka autogéré. | 29 juin 2021 |

| | | |
|---|---|---|
| Extensions Lambda API | Disponibilité générale pour les extensions Lambda. Utilisez des extensions pour augmenter vos fonctions Lambda. Vous pouvez utiliser des extensions fournies par des partenaires Lambda ou créer vos propres extensions Lambda. Pour plus de détails, consultez la section Extension s Lambda. API | 24 mai 2021 |
| Nouvelle expérience de la console Lambda | La console Lambda a été remaniée pour améliorer les performances et la cohérence. | 2 mars 2021 |
| Environnement d'exécution Node.js 14 | Lambda prend désormais en charge un nouvel environne ment d'exécution (runtime) pour Node.js 14. Node.js 14 utilise Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js. | 27 janvier 2021 |
| Images de conteneurs Lambda | Lambda prend désormais en charge les fonctions définies en tant qu'images conteneur s. Vous pouvez combiner la flexibilité des outils de conteneur avec l'agilité et la simplicité opérationnelle de Lambda pour générer des applications. Pour plus de détails, consultez Utilisation d'images de conteneurs avec Lambda. | 1er décembre 2020 |

| | | |
|---|---|---|
| Signature de code pour fonctions Lambda | Lambda prend désormais en charge la signature de code. Les administrateurs peuvent configurer des fonctions Lambda de façon à n'accepter que du code signé lors du déploiement. Lambda vérifie les signature s afin de s'assurer que le code n'est pas modifié ou altéré. En outre, Lambda s'assure que le code est signé par des développeurs de confiance avant d'accepter le déploiement. Pour plus de détails, consultez Configura tion de la signature de code pour Lambda. | 23 novembre 2020 |
| Aperçu : Lambda Runtime Logs API | Lambda prend désormais en charge les journaux d'exécuti on. API Les extensions Lambda peuvent utiliser les journaux API pour s'abonner à des flux de journaux dans l'environnement d'exécution. Pour plus de détails, consultez Lambda Runtime Logs. API | 12 novembre 2020 |

| | | |
|---|---|---|
| Nouvelle source d'événements pour Amazon MQ | Lambda prend désormais en charge Amazon MQ en tant que source d'événements. Utilisez une fonction Lambda pour traiter les enregistr ements de votre agent de messages Amazon MQ. Pour plus de détails, consultez Utilisation de Lambda avec Amazon MQ. | 5 novembre 2020 |
| Aperçu : Lambda Extensions API | Utilisez des extensions Lambda pour augmenter vos fonctions Lambda. Vous pouvez utiliser des extension s fournies par des partenaires Lambda ou créer vos propres extensions Lambda. Pour plus de détails, consultez la section Extensions Lambda. API | 8 octobre 2020 |
| Support pour Java 8 et environnements d'exécution personnalisés sur AL2 | Lambda prend désormais en charge Java 8 et les environne ments d'exécution (runtimes ) personnalisés sur Amazon Linux 2. Pour en savoir plus, consultez Environnements d'exécution Lambda. | 12 août 2020 |

| | | |
|---|---|---|
| Nouvelle source d'événements pour Amazon Managed Streaming for Apache Kafka | Lambda prend désormais en charge Amazon MSK en tant que source d'événements. Utilisez une fonction Lambda avec Amazon MSK pour traiter les enregistrements d'une rubrique Kafka. Pour plus de détails, consultez la section Utilisation de Lambda avec Amazon. MSK | 11 août 2020 |
| IAMclés de condition pour les VPC paramètres Amazon | Vous pouvez désormais utiliser des clés de condition spécifiques à Lambda pour les paramètres. VPC Par exemple, vous pouvez exiger que toutes les fonctions de votre organisation soient connectées à unVPC. Vous pouvez également spécifier les sous-réseaux et les groupes de sécurité que les utilisateurs de la fonction peuvent et ne peuvent pas utiliser. Pour plus de détails, consultez la section Configuration VPC des IAM fonctions. | 10 août 2020 |

| | | |
|---|---|---|
| Paramètres de simultanéité pour les consommateurs de flux HTTP Kinesis /2 | Vous pouvez désormais utiliser les paramètres de simultanéité suivants pour les utilisateurs de Kinesis avec un ventilateur amélioré HTTP (/2 flux) ParallelizationFactor : MaximumRetryAttempts,,, et MaximumRecordAgeInSeconds. DestinationConfig BisectBatchOnFunctionError Pour plus de détails, consultez la section Utilisation AWS Lambda avec Amazon Kinesis. | 7 juillet 2020 |
| Fenêtre de traitement par lots pour les utilisateurs de Kinesis HTTP /2 Stream | Vous pouvez désormais configurer une fenêtre de traitement par lots (MaximumBatchingWindowInSeconds) pour HTTP /2 flux. Lambda lit les enregistrements du flux jusqu'à avoir rassemblé un lot complet, ou jusqu'à l'expiration de la fenêtre de traitement par lots. Pour plus de détails, consultez la section Utilisation AWS Lambda avec Amazon Kinesis. | 18 juin 2020 |

| | | |
|---|---|---|
| Support pour les systèmes de EFS fichiers Amazon | Vous pouvez désormais connecter un système de EFS fichiers Amazon à vos fonctions Lambda pour accéder aux fichiers réseau partagés. Pour plus de détails, consultez Configuration de l'accès au système de fichiers pour les fonctions Lambda. | 16 juin 2020 |
| AWS CDK exemples d'applications dans la console Lambda | La console Lambda inclut désormais des exemples d'applications utilisant le AWS Cloud Development Kit (AWS CDK) for. TypeScript AWS CDK Il s'agit d'un framework qui vous permet de définir les ressources de votre application en Python TypeScript, Java ou. NET. | 1er juin 2020 |
| Support pour. NETExécution de Core 3.1.0 dans AWS Lambda | AWS Lambda prend désormais en charge le. NETRuntime Core 3.1.0. Pour plus de détails, voir. NETNoyau CLI. | 31 mars 2020 |

| | | |
|---|---|---|
| Support pour API Gateway HTTP APIs | Documentation mise à jour et étendue sur l'utilisation de Lambda avec API Gateway, y compris la prise en charge de. HTTP APIs Ajout d'un exemple d'application qui crée une fonction API et avec AWS CloudFormation. Pour plus de détails, consultez la section Utilisation de Lambda avec Amazon API Gateway. | 23 mars 2020 |
| Ruby 2.7 | Un nouvel environnement d'exécution est disponible pour Ruby 2.7, ruby2.7, qui est le premier environnement d'exécution Ruby à utiliser Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Ruby. | 19 février 2020 |

| | | |
|---|---|---|
| Métriques de simultanéité | Lambda rapporte désormais les métriques `ConcurrentExecutions` pour l'ensemble des fonctions, alias et versions. Vous pouvez afficher un graphique pour cette métrique dans la page de surveillance de votre fonction. Auparavant, `ConcurrentExecutions` était seulement signalé au niveau du compte et pour les fonctions utilisant la simultanéité réservée. Pour plus d'informations, consultez AWS Lambda Métriques des fonctions. | 18 février 2020 |

| | | |
|---|---|---|
| Mise à jour des états des fonctions | Les états de fonction sont désormais appliqués pour toutes les fonctions par défaut. Lorsque vous connectez une fonction à unVPC, Lambda crée des interfaces réseau élastiques partagées. Cela permet à votre fonction d'évoluer sans créer d'interfa ces réseau supplémentaires. Pendant ce temps, vous ne pouvez pas effectuer d'opérati ons supplémentaires sur la fonction, y compris la mise à jour de sa configuration et de ses versions de publication. Dans certains cas, l'invocation est également affectée. Les détails sur l'état actuel d'une fonction sont disponibles sur le LambdaAPI.<br><br>Cette mise à jour est publiée en plusieurs phases. Pour plus de détails, consultez la section Mise à jour du cycle de vie des états Lambda pour le VPC réseau sur le blog AWS Compute. Pour plus d'informations sur les états, consultez États de fonctions AWS Lambda. | 24 janvier 2020 |

| | | |
|---|---|---|
| Mises à jour de la API sortie de configuration des fonctions | Ajout de codes de motif à StateReasonCode(InvalidSubnet, InvalidSecurityGroup) et LastUpdateStatusReasonCode (SubnetOutOfIPAddresses, InvalidSubnet, InvalidSecurityGroup) pour les fonctions qui se connectent à unVPC. Pour plus d'informations sur les états, consultez États de fonctions AWS Lambda. | 20 janvier 2020 |
| Simultanéité allouée | Vous pouvez désormais allouer la simultanéité provisionnée pour une version de fonction ou un alias. La simultanéité provisionnée permet à une fonction d'évoluer sans fluctuations de latence. Pour plus de détails, consultez Gestion de la simultanéité pour une fonction Lambda. | 3 décembre 2019 |

| | | |
|---|---|---|
| Créer un proxy de base de données | Vous pouvez désormais utiliser la console Lambda pour créer un proxy de base de données pour une fonction Lambda. Un proxy de base de données permet à une fonction d'atteindre des niveaux de simultanéité élevés sans épuiser les connexion s de base de données. Pour plus de détails, consultez Configuration de l'accès à une base de données pour une fonction Lambda. | 3 décembre 2019 |
| Prise en charge des percentil es pour la métrique de durée | Vous pouvez désormais filtrer la métrique de durée en fonction des percentiles. Pour plus d'informations, consultez Métriques AWS Lambda. | 26 novembre 2019 |
| Augmentation de la simultané ité pour des sources d'événement de flux | Une nouvelle option pour les mappages de source d'événements de flux DynamoDB et de flux Kinesis vous permet de traiter plusieurs lots à la fois à partir de chaque partition. Lorsque vous augmentez le nombre de lots simultanés par partition , la simultanéité de votre fonction peut atteindre dix fois le nombre de partitions dans votre flux. Pour en savoir plus, consultez Mappage de source d'événement Lambda. | 25 novembre 2019 |

| | | |
|---|---|---|
| États des fonctions | Lorsque vous créez ou mettez à jour une fonction, celle-ci passe en état d'attente pendant que Lambda approvisionne les ressources pour sa prise en charge. Si vous connectez votre fonction à unVPC, Lambda peut créer une interface réseau Elastic partagée immédiatement, au lieu de créer des interfaces réseau lorsque votre fonction est invoquée. Cela améliore les performances des fonctions VPC connectées, mais peut nécessiter une mise à jour de votre automatisation. Pour plus d'informations, consultez États de fonction AWS Lambda. | 25 novembre 2019 |
| Options de gestion des erreurs pour les invocations asynchrones | De nouvelles options de configuration sont disponibles pour l'invocation asynchrone. Vous pouvez configurer Lambda de manière à limiter les nouvelles tentatives et à définir un âge d'événement maximal. Pour plus de détails, consultez Configuration de la gestion des erreurs pour les invocations asynchrones. | 25 novembre 2019 |

| | | |
|---|---|---|
| Gestion des erreurs pour les sources d'événement de flux | De nouvelles options de configuration sont disponibles pour les mappages de source d'événement qui lisent depuis les flux. Vous pouvez configurer des mappages de source d'événements DynamoDB Streams et Kinesis Streams pour limiter les nouvelles tentatives et définir une ancienneté de registre maximale. Lorsque des erreurs se produisent, vous pouvez configurer le mappage de source d'événement pour fractionner les lots avant d'effectuer des nouvelles tentatives et d'envoyer des enregistrements d'invocations pour les lots ayant échoué vers une file d'attente ou une rubrique. Pour en savoir plus, consultez Mappage de source d'événement Lambda. | 25 novembre 2019 |

| | | |
|---|---|---|
| Destinations pour les invocations asynchrones | Vous pouvez désormais configurer Lambda pour envoyer des enregistrements d'invocations asynchrones à un autre service. Les enregistrements d'invocation contiennent des détails sur l'événement, le contexte et la réponse de la fonction. Vous pouvez envoyer des enregistrements d'invocation vers une SQS file d'attente, un SNS sujet, une fonction Lambda EventBridge ou un bus d'événements. Pour plus de détails, consultez Configuration des destinations pour les invocations asynchrones. | 25 novembre 2019 |
| Nouveaux environnements d'exécution pour Node.js, Python et Java | De nouveaux environnements d'exécution sont disponibles pour Node.js 12, Python 3.8 et Java 11. Pour en savoir plus, consultez Environnements d'exécution Lambda. | 18 novembre 2019 |
| FIFOprise en charge des files d'attente pour les sources SQS d'événements Amazon | Vous pouvez désormais créer un mappage de source d'événements qui lit à partir d'une file d'attente premier entré, premier sorti (FIFO). Auparavant, seules les files d'attente standard étaient prises en charge. Pour plus de détails, consultez la section Utilisation de Lambda avec Amazon. SQS | 18 novembre 2019 |

| | | |
|---|---|---|
| Créer des applications dans la console Lambda | La création d'applications dans la console Lambda est désormais généralement disponible. Pour obtenir des instructions, consultez la section Gestion des applications dans la console Lambda. | 31 octobre 2019 |
| Créer des applications dans la console Lambda (bêta) | Vous pouvez désormais créer une application Lambda avec un pipeline de livraison continue intégré dans la console Lambda. La console fournit des exemples d'applications que vous pouvez utiliser comme point de départ pour votre propre projet. Choisissez entre AWS CodeCommit et GitHub pour le contrôle de source. Chaque fois que vous transmettez des modifications à votre référentiel, le pipeline inclus les construit et les déploie automatiquement. Pour obtenir des instructions, consultez la section Gestion des applications dans la console Lambda. | 3 octobre 2019 |

| | | |
|---|---|---|
| Améliorations des performances pour les VPC fonctions connectées | Lambda utilise désormais un nouveau type d'interface réseau élastique partagée par toutes les fonctions d'un sous-réseau de cloud privé virtuel (VPC). Lorsque vous connectez une fonction à unVPC, Lambda crée une interface réseau pour chaque combinaison de groupe de sécurité et de sous-rése au que vous choisissez. Lorsque les interfaces réseau partagées sont disponibles, la fonction n'a plus besoin de créer d'interfaces réseau supplémentaires au fur et à mesure de sa mise à l'échelle ascendante. Ainsi, les temps de démarrage sont considéra blement améliorés. Pour plus de détails, voir Configuration d'une fonction Lambda pour accéder aux ressources d'un. VPC | 3 septembre 2019 |

| | | |
|---|---|---|
| Paramètres dpour les lots de flux | Vous pouvez désormais configurer une fenêtre de traitement par lots pour les mappages de source d'événements Amazon DynamoDB et Amazon Kinesis. Configurez une fenêtre de traitement par lots de cinq minutes maximum pour mettre en mémoire tampon les enregistrements entrants jusqu'à ce qu'un lot complet soit disponible. Cela réduit le nombre de fois où votre fonction est invoquée lorsque le flux est moins actif. | 29 août 2019 |
| CloudWatch Intégration de Logs Insights | La page de surveillance de la console Lambda inclut désormais les rapports d'Amazon CloudWatch Logs Insights. | 18 juin 2019 |
| Amazon Linux 2018.03 | L'environnement d'exécution Lambda est en cours de mise à jour pour utiliser Amazon Linux 2018.03. Pour plus de détails, consultez Environnement d'exécution. | 21 mai 2019 |

| | | |
|---|---|---|
| Node.js 10 | Un nouvel environnement d'exécution est disponible pour Node.js 10, nodejs10.x. Cet environnement d'exécution utilise Node.js 10.15 et sera mis à jour avec la dernière version de Node.js 10 régulièrement. Node.js 10 est également le premier environnement d'exécution pour utiliser Amazon Linux 2. Pour plus d'informations, consultez Création de fonctions Lambda avec Node.js. | 13 mai 2019 |
| GetLayerVersionByArn API | Utilisez le GetLayerVersionByArnAPIpour télécharger les informations de version de la couche avec la version ARN en entrée. Par rapport à GetLayerVersion, vous GetLayerVersionByArn permet d'utiliser le ARN directement au lieu de l'analyser pour obtenir le nom de la couche et le numéro de version. | 25 avril 2019 |
| Ruby | AWS Lambda supporte désormais Ruby 2.5 avec un nouveau runtime. Pour plus d'informations, consultez Création de fonctions Lambda avec Ruby. | 29 novembre 2018 |

| | | |
|---|---|---|
| Couches | Avec les couches Lambda, vous pouvez regrouper et déployer des bibliothèques, des environnements d'exécution personnalisés, ainsi que d'autres dépendances séparément depuis le code de votre fonction. Partagez vos couches avec vos autres comptes ou avec le monde entier. Pour plus de détails, consultez Couches Lambda. | 29 novembre 2018 |
| Exécutions personnalisées | Créez un environnement d'exécution (runtime) personnalisé pour exécuter des fonctions Lambda dans votre langage de programmation favori. Pour plus de détails, consultez Runtimes Lambda personnalisés. | 29 novembre 2018 |
| Déclencheurs d'Application Load Balancer | Elastic Load Balancing prend désormais en charge les fonctions Lambda en tant que cibles pour les équilibreurs de charge d'applications. Pour plus de détails, consultez Utilisation de Lambda avec des équilibreurs de charge des applications. | 29 novembre 2018 |

| | | |
|---|---|---|
| Utilisez les consommateurs de flux Kinesis HTTP /2 comme déclencheur | Vous pouvez utiliser les consommateurs de flux de données Kinesis HTTP /2 pour envoyer des événements. AWS Lambda Les consommateurs de flux disposent d'un débit de lecture dédié pour chaque partition de votre flux de données et utilisent HTTP /2 pour minimiser la latence. Pour plus de détails, consultez Utilisation de Lambda avec Kinesis. | 19 novembre 2018 |
| Python 3.7 | AWS Lambda supporte désormais Python 3.7 avec un nouveau runtime. Pour plus d'informations, consultez Création de fonctions Lambda avec Python. | 19 novembre 2018 |
| Augmentation de la limite de la charge utile d'invocation de fonction asynchrone | La taille de charge utile maximale pour les appels asynchrones est passée de 128 Ko à 256 Ko, ce qui correspond à la taille maximale des messages provenant d'un déclencheur Amazon. SNS Pour plus de détails, consultez Quotas Lambda. | 16 novembre 2018 |
| AWS GovCloud Région (USA Est) | AWS Lambda est désormais disponible dans la région AWS GovCloud (USA Est). | 12 novembre 2018 |

| | | |
|---|---|---|
| AWS SAM Rubriques déplacées vers un guide du développeur distinct | Un certain nombre de sujets étaient axés sur la création d'applications sans serveur à l'aide de AWS Serverles s Application Model (AWS SAM). Ces rubriques ont été déplacées vers le Guide du développeur AWS Serverless Application Model. | 25 octobre 2018 |
| Afficher des applications Lambda dans la console | Vous pouvez afficher l'état de vos applications Lambda sur la page Applications dans la console Lambda. Cette page indique l'état de la AWS CloudFormation pile. Elle inclut des liens vers les pages où vous pouvez consulter plus d'informations sur les ressources figurant dans la pile. Vous pouvez également consulter les métriques agrégées pour l'application et créer des tableaux de bord de surveillance personnalisés. | 11 octobre 2018 |
| Délai d'attente d'exécution de la fonction | Pour permettre d'utiliser des fonctions de longue durée, le délai d'exécution maximal configurable est passé de 5 minutes à 15 minutes. Pour plus de détails, consultez Limites Lambda. | 10 octobre 2018 |

| | | |
|---|---|---|
| Support du langage PowerShell de base dans AWS Lambda | AWS Lambda supporte désormais le langage PowerShell de base. Pour plus d'informations, voir Modèle de programmation pour la création de fonctions Lambda dans. PowerShell | 11 septembre 2018 |
| Support pour. NETExécution de Core 2.1.0 dans AWS Lambda | AWS Lambda prend désormais en charge le. NETRuntime Core 2.1.0. Pour plus d'informations, consultez. NETNoyau CLI. | 9 juillet 2018 |
| Les mises à jour sont désormais disponibles sur RSS | Vous pouvez désormais vous abonner à un RSS fil pour suivre les publications de ce guide. | 5 juillet 2018 |
| Support pour Amazon SQS en tant que source d'événements | AWS Lambda prend désormais en charge Amazon Simple Queue Service (AmazonSQS) en tant que source d'événements. Pour plus d'informations, consultez Invocation des fonctions Lambda. | 28 juin 2018 |
| Région Chine (Ningxia) | AWS Lambda est désormais disponible dans la région de Chine (Ningxia). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 28 juin 2018 |

# Mises à jour antérieures

Le tableau ci-après décrit les modifications importantes apportées dans chaque version du Manuel du développeur AWS Lambda  avant juin 2018.

| Modification | Description | Date |
| --- | --- | --- |
| Prise en charge de l'environnement d'exécution Node.js 8.10 | AWS Lambda prend désormais en charge la version 8.10 du runtime Node.js. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js. | 2 avril 2018 |
| Révision des fonctions et des alias IDs | AWS Lambda prend désormais en charge la révision IDs des versions et des alias de vos fonctions. Vous pouvez les utiliser IDs pour suivre et appliquer des mises à jour conditionnelles lorsque vous mettez à jour la version de votre fonction ou vos ressources d'alias. | 25 janvier 2018 |
| Support d'exécution pour Go et. NET2,0 | AWS Lambda a ajouté le support d'exécution pour Go et. NET2,0. Pour plus d'informations, consultez Création de fonctions Lambda avec Go et Création de fonctions Lambda avec C#. | 15 janvier 2018 |
| Nouvelle conception de la console | AWS Lambda a introduit une nouvelle console Lambda pour simplifier votre expérience et a ajouté un éditeur de code Cloud9 pour améliorer votre capacité à déboguer et à réviser votre code de fonction. | 30 novembre 2017 |
| Définition de limites de simultanéité pour des fonctions individuelles | AWS Lambda permet désormais de définir des limites de simultanéité pour des fonctions individuelles. Pour de plus amples informations, veuillez consulter Configuration de la simultanéité réservée pour une fonction. | 30 novembre 2017 |
| Déplacement du trafic avec des alias | AWS Lambda prend désormais en charge le transfert de trafic avec des alias. Pour de plus amples informations, veuillez consulter Créez des déploiements progressifs pour les fonctions Lambda. | 28 novembre 2017 |

| Modification | Description | Date |
| --- | --- | --- |
| Déploiement de code graduel | AWS Lambda permet désormais de déployer en toute sécurité de nouvelles versions de votre fonction Lambda en tirant parti de Code Deploy. Pour plus d'informations, consultez Déploiement de code graduel. | 28 novembre 2017 |
| Région Chine (Pékin) | AWS Lambda est désormais disponible dans la région Chine (Pékin). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 9 novembre 2017 |
| Présentation de SAM Local | AWS Lambda présente SAM Local (désormais connu sous le nom de SAMCLI), un AWS CLI outil qui fournit un environnement dans lequel vous pouvez développer, tester et analyser vos applications sans serveur localemen t avant de les télécharger sur le runtime Lambda. Pour plus d'informations, consultez Test et débogage d'applications sans serveur. | 11 août 2017 |
| Région Canada (Centre) | AWS Lambda est désormais disponible dans la région du Canada (Centre). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 22 juin 2017 |
| South America (São Paulo) Region | AWS Lambda est désormais disponible dans la région Amérique du Sud (São Paulo). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 6 juin 2017 |
| AWS Lambda support pour AWS X-Ray. | Lambda prend désormais en charge X-Ray, ce qui vous permet de détecter, d'analyser et d'optimiser les problèmes de performance avec les applications Lambda. Pour plus d'informations, consultez Visualisez les invocations de fonctions Lambda à l'aide de AWS X-Ray. | 19 avril 2017 |

| Modification | Description | Date |
| --- | --- | --- |
| Région Asie-Pacifique (Mumbai) | AWS Lambda est désormais disponible dans la région Asie-Pacifique (Mumbai). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 28 mars 2017 |
| AWS Lambda prend désormais en charge le runtime Node.js v6.10 | AWS Lambda ajout du support pour le runtime Node.js v6.10. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js. | 22 mars 2017 |
| Région Europe (Londres) | AWS Lambda est désormais disponible dans la région Europe (Londres). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 1 février 2017 |
| AWS Lambda soutien au. NETruntime, Lambda @Edge (version préliminaire), Dead Letter Queues et déploiement automatisé d'applica tions sans serveur. | AWS Lambda ajout du support pour C#. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec C#.<br><br>Lambda @Edge vous permet d'exécuter des fonctions Lambda sur les sites AWS Edge en réponse à des événements. CloudFront Pour plus d'informations, voir Personnaliser en périphérie avec Lambda @Edge. | 3 décembre 2016 |
| AWS Lambda ajoute Amazon Lex en tant que source d'événeme nts prise en charge. | Avec Lambda et Amazon Lex, vous pouvez créer rapidement des chatbots pour divers services, comme Slack et Facebook. | 30 novembre 2016 |
| US West (N. Californi a) Region | AWS Lambda est désormais disponible dans la région USA Ouest (Californie du Nord). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 21 novembre 2016 |

| Modification | Description | Date |
| --- | --- | --- |
| Introduction de la fonctionnalité AWS SAM permettant de créer et de déployer des applications basées sur Lambda et d'utiliser des variables d'environnement pour les paramètres de configuration des fonctions Lambda. | AWS SAM: vous pouvez désormais utiliser le AWS SAM pour définir la syntaxe d'expression des ressources dans une application sans serveur. Pour déployer votre application, il vous suffit de spécifier les ressources dont vous avez besoin dans le cadre de votre application, ainsi que les politiques d'autorisation associées dans un fichier AWS CloudFormation modèle (écrit dans l'un JSON ou l'autreYAML), de regrouper vos artefacts de déploiement et de déployer le modèle.<br><br>Variables d'environnement : vous pouvez utiliser des variables d'environnement afin de spécifier des paramètres de configuration pour votre fonction Lambda en dehors du code de votre fonction. Pour plus d'informations, consultez Utiliser les variables d'environnement Lambda pour configurer les valeurs dans le code. | 18 novembre 2016 |
| Région Asia Pacific (Seoul) | AWS Lambda est désormais disponible dans la région Asie-Pacifique (Séoul). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 29 août 2016 |
| Asia Pacific (Sydney) Region | Lambda est désormais disponible dans la région Asie-Pacifique (Sydney). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 23 juin 2016 |
| Mises à jour de la console Lambda | La console Lambda a été mise à jour pour simplifier le processus de création de rôles. | 23 juin 2016 |
| AWS Lambda prend désormais en charge le runtime Node.js v4.3 | AWS Lambda ajout du support pour le runtime Node.js v4.3. Pour de plus amples informations, veuillez consulter Création de fonctions Lambda avec Node.js. | 07 avril 2016 |

| Modification | Description | Date |
| --- | --- | --- |
| Région Europe (Frankfurt) | Lambda est désormais disponible dans la région Europe (Frankfurt). Pour plus d'informations sur les régions et les points de terminaison Lambda, consultez Régions et points de terminaison (français non garanti) dans la Références générales AWS. | 14 mars 2016 |
| VPCsoutien | Vous pouvez désormais configurer une fonction Lambda pour accéder aux ressources de votre. VPC Pour de plus amples informations, veuillez consulter Permettre aux fonctions Lambda d'accéder aux ressources d'un Amazon VPC. | 11 février 2016 |
| Le runtime Lambda a été mis à jour. | L'environnement d'exécution a été mis à jour. | 4 novembre 2015 |

| Modification | Description | Date |
| --- | --- | --- |
| Prise en charge de la gestion des versions, Python pour le développement de code pour des fonctions Lambda, événements planifiés et allongement du temps d'exécution | Vous pouvez désormais développer le code de la fonction Lambda à l'aide du langage Python. Pour plus d'informations, consultez Création de fonctions Lambda avec Python.<br><br>Gestion des versions : vous pouvez conserver une ou plusieurs versions de la fonction Lambda. La gestion des versions vous permet de contrôler quelle version de fonction Lambda est exécutée dans des environnements différents (par exemple, développement, test ou production). Pour plus d'informations, consultez Gérer les versions des fonctions Lambda.<br><br>Événements planifiés : vous pouvez également configurer Lambda pour invoquer régulièrement le code à l'aide de la console Lambda. Vous pouvez spécifier un taux fixe (nombre d'heures, de jours ou de semaines) ou vous pouvez spécifier une expression cron. Pour de plus amples informations, veuillez consulter Invoquer une fonction Lambda dans un planning.<br><br>Rallongement du temps d'exécution : vous pouvez maintenant définir un temps d'exécution pouvant allant jusqu'à cinq minutes pour les fonctions Lambda afin de rendre possibles les fonctions de plus longue durée, telles que l'ingestion de données volumineuses et le traitement des tâches. | 08 octobre 2015 |

| Modification | Description | Date |
| --- | --- | --- |
| Prise en charge de DynamoDB Streams | DynamoDB Streams est désormais disponible dans toutes les régions qui acceptent DynamoDB. Vous pouvez activer DynamoDB Streams pour votre table et utiliser une fonction Lambda comme déclencheur de cette table. Les déclencheurs sont des actions personnalisées que vous effectuez en réponse aux mises à jour apportées à la table DynamoDB. Pour afficher un exemple de procédure, veuillez consulter Tutoriel : Utilisation AWS Lambda avec les flux Amazon DynamoDB . | 14 juillet 2015 |
| Lambda prend désormais en charge l'appel de fonctions Lambda avec des clients compatibles. REST | Jusqu'à présent, pour appeler votre fonction Lambda depuis votre application Web, mobile ou IoT, vous en aviez besoin AWS SDKs (par exemple, AWS SDK pour Java, AWS SDK pour Android ou pour AWS SDK iOS). Lambda prend désormais en charge l'appel d'une fonction Lambda avec des clients REST compatibles via une fonction personnalisée que API vous pouvez créer à l'aide d'Amazon Gateway. API Vous pouvez envoyer des demandes au point de terminaison de votre fonction Lambda. URL Vous pouvez configurer la sécurité du terminal pour autoriser le libre accès, tirer parti de AWS Identity and Access Management (IAM) pour autoriser l'accès ou utiliser des API clés pour mesurer l'accès d'autres personnes à vos fonctions Lambda.<br><br>Pour voir un exemple d’exercice de mise en route, veuillez consulter Invocation d'une fonction Lambda à l'aide d'un point de terminaison Amazon Gateway API.<br><br>Pour plus d'informations sur Amazon API Gateway, consultez https://aws.amazon.com/api-gateway/. | 09 juillet 2015 |

| Modification | Description | Date |
| --- | --- | --- |
| La console Lambda fournit maintenant des plans pour créer facilement des fonctions Lambda et les tester. | La console Lambda fournit un ensemble de plans. Chaque plan fournit un exemple de configuration de source d'événement et un exemple de code que vous pouvez utiliser pour créer facilement des applications basées sur Lambda. Tous les exercices de mise en route Lambda utilisent désormais les plans. Pour plus d'informations, consultez ???. | 09 juillet 2015 |
| Lambda prend désormais en charge Java pour créer vos fonctions Lambda. | Vous pouvez désormais créer le code Lambda en Java. Pour plus d'informations, consultez Création de fonctions Lambda avec Java. | 15 juin 2015 |
| Lambda prend désormais en charge la spécification d'un objet Amazon S3 en tant que fonction .zip lorsque vous créez ou mettez à jour une fonction Lambda. | Vous pouvez importer un package de déploiement de fonctions Lambda (fichier .zip) dans un compartiment Amazon S3 de la région dans laquelle vous souhaitez créer une fonction Lambda. Vous pouvez ensuite spécifier le nom du compartiment et le nom de la clé objet lorsque vous créez ou mettez à jour une fonction Lambda. | 28 mai 2015 |

| Modification | Description | Date |
| --- | --- | --- |
| Lambda est désormais disponible de manière globale et prend en charge les backends mobiles | Lambda est désormais disponible de manière globale en production. Cette version présente également de nouvelles fonctionnalités qui facilitent la création de backends pour les mobiles, les tablettes et l'Internet des objets (IoT) via Lambda, qui adaptent automatiquement leur échelle, sans que vous ayez à mettre en service ou à gérer une infrastructure. Lambda prend désormais en charge les événements en temps réel (synchrones) et asynchrones. Les fonctionnalités supplémentaires comprennent la rationalisation de la configuration et de la gestion des sources d'événement. Le modèle d'autorisation et le modèle de programmation ont été simplifiés par la mise en place de stratégies de ressources pour les fonctions Lambda.<br><br>La documentation a été mise à jour en conséquence. Pour plus d'informations, consultez les rubriques suivantes :<br><br>???<br><br>AWS Lambda | 9 avril 2015 |
| Version préliminaire | Version préliminaire du Guide du développeur AWS Lambda . | 13 novembre 2014 |

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.