3 Key Elements for Your GitOps Strategy

EBOOK

GITOPS

Table of Contents

Introduction

GitOps has gained popularity in the cloud-native ecosystem, particularly in Kubernetes-based environments, where managing infrastructure as code is crucial. Knowing the philosophy of GitOps, what your current deployment strategy looks like, and essential tools are key to implementing a successful GitOps strategy.

In this ebook you will learn:

- a refresher on the the core principles of GitOps,
- how your deployment strategy affects your GitOps plan, and
- why observability is a critical part of GitOps and how it differs from monitoring

Getting Started with GitOps

GitOps is an operational framework that aims to streamline and automate the deployment and management of applications and infrastructure using Git as the single source of truth. It simplifies and standardizes the deployment and management of complex systems, improves collaboration between development and operations teams, and increases the reliability and reproducibility of deployments. GitOps is popular in the cloud-native ecosystem, particularly in Kubernetes-based environments, where managing infrastructure as code is crucial. Essential elements of GitOps include continuous integration / continuous delivery (CI/CD), choosing between pull- or push-based architecture, and observability.

Looking for an In-Depth Introduction to GitOps?

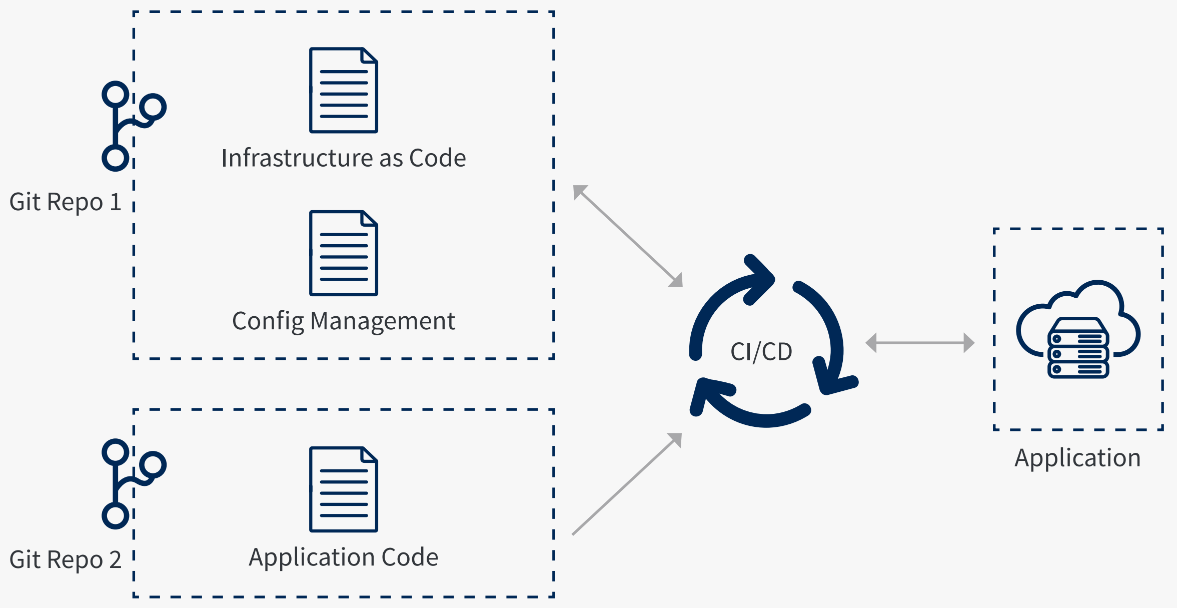
Learn the differences between GitOps and DevOps, and how to ensure Git is the single source of truth for your application in our free Understanding GitOps guide.

DOWNLOAD NOW

The Philosophy of GitOps

Before getting started, it's important to understand the philosophy of GitOps. GitOps borrows best practices from DevOps and applies them to infrastructure automation. Whereas DevOps is the practice of automating the software development lifecycle, GitOps contributes to the automation of infrastructure. This includes version control, collaboration, compliance, and CI/CD.

GitOps is a specific implementation of DevOps that uses Git as the single source of truth for declarative infrastructure and application code. In GitOps, the desired state of the system is versioned and stored in a Git repository, and a reconciliation loop continuously monitors and enforces that desired state. GitOps takes DevOps practices and puts them into action for managing cloud-native workloads.



GitOps automates application code deployments and infrastructure management.
Adhere to best practices by creating separate repos for your application code, and configs.
Updates to the application repo trigger processes with your CI/CD tooling to update the config repo.
The config changes are then pushed/pulled to the deployment environment.

As a base for exploring and planning your GitOps strategy, review the OpenGitOps Principles, published by the GitOps Working Group:

The desired state of a GitOps managed system must be:

1. ***Declarative***
   A system managed by GitOps must have its desired state expressed declaratively.

2. ***Versioned and immutable***
   Desired state is stored in a way that enforces immutability and versioning and retains a complete version history.

3. ***Pulled automatically***
   Software agents automatically pull the desired state declarations from the source.

4. ***Continuously reconciled***
   Software agents continuously observe the actual system state and attempt to apply the desired state.

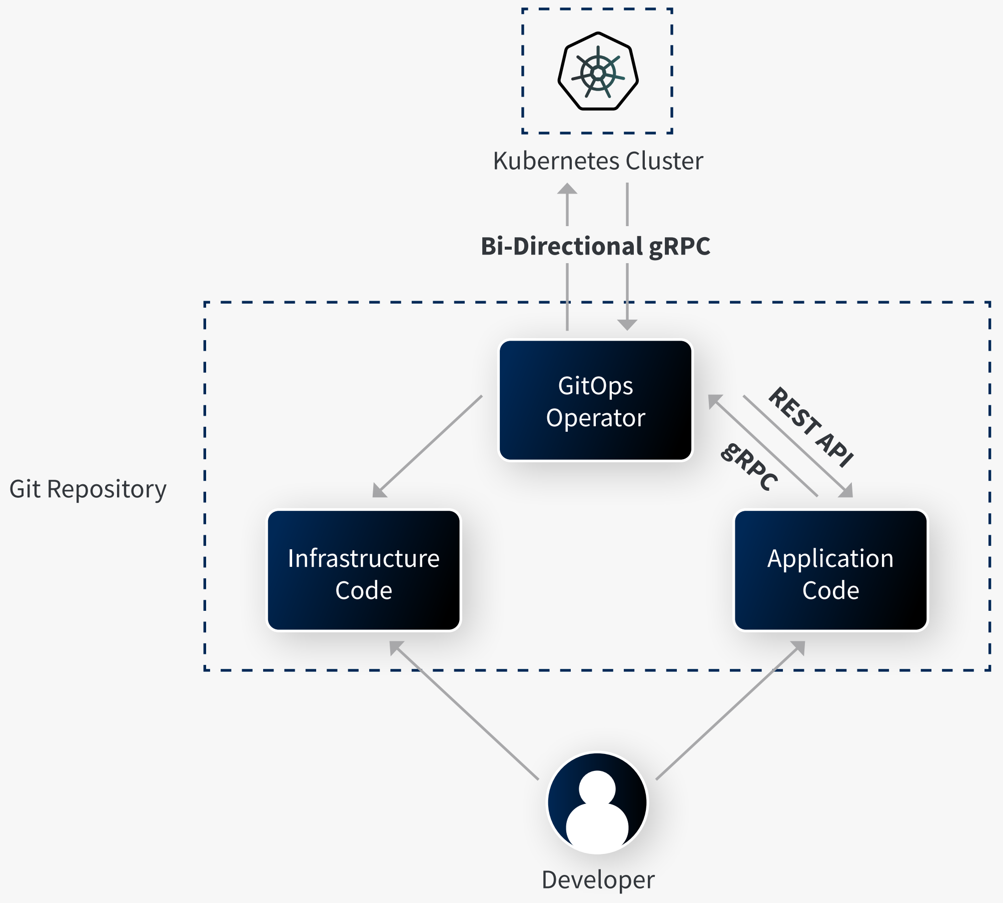
Push vs. Pull-Based Architecture

Adopting a cloud-native architecture is not without its challenges. While the benefits are substantial, familiarizing yourself with the initial challenges is crucial for a successful transition.

The first hurdle is modifying your application to fit the cloud-native model. This process involves re-architecting applications fit into a standardized and vendor-agnostic workflow. It often requires breaking down monolithic applications into microservices, which can be a complex and resource-intensive task - but one that will pay off by providing flexibility in the future.

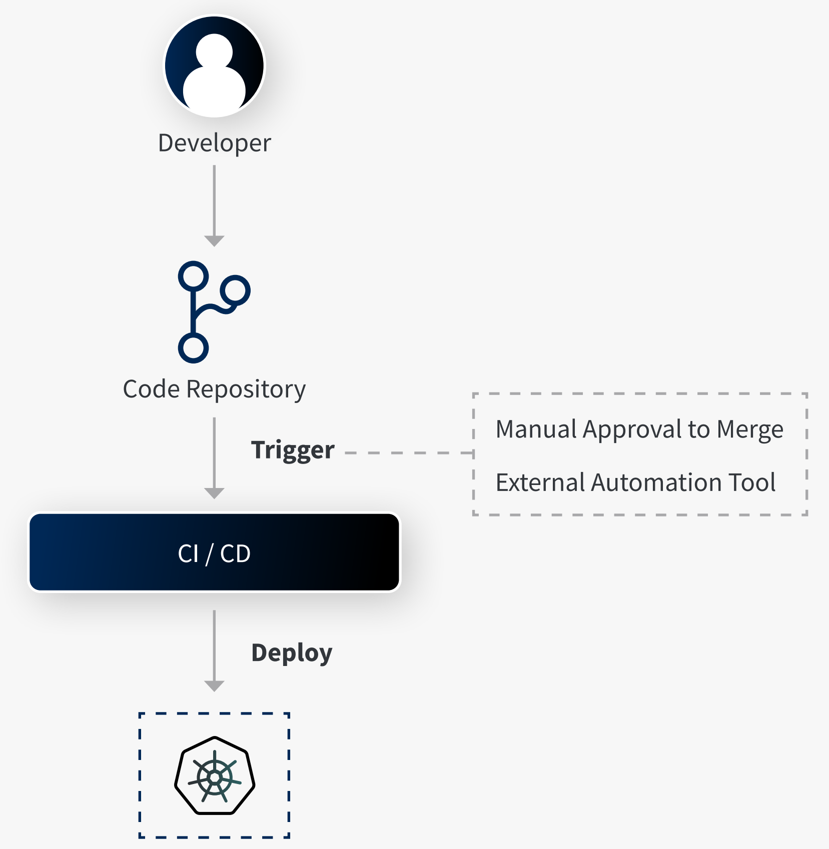
Another significant challenge is dealing with the complexities of multicloud environments. Each cloud service has different APIs, syntax, and other features, and each comes with its own unique set of “gotchas.” This complexity is compounded by the need to ensure consistency in deployment, security, and operations across various platforms.

Transitioning to a cloud-native architecture also brings about an increase in maintenance tasks. The dynamic and distributed nature of cloud-native applications requires continuous monitoring, updates, and security checks to ensure optimal performance and security.



Pull-based architecture requires a GitOps operator, or agent, to evaluate both the infrastructure and application configuration in the repository to detect when the Kubernetes cluster resources should automatically update based on config changes.

A push-based, or agentless, approach can be seen as a deviation from the original GitOps principles, but sometimes is preferable for managing varying infrastructure components outside the Kubernetes environment.



Push-based architecture also relies on CI/CD tools like Jenkins to deploy code changes from the Git repository but lacks the automation benefits of pull-based deployments.

Complex architectures with a mixture of Kubernetes and non-Kubernetes workloads can leverage a combination of both. Robust observability and reconciliation strategies must be thoroughly planned in this scenario, and built-in by design. A more preferable approach would be to implement a pull-based approach that combines a cloud-native continuous delivery (CD) with a cloud-native control plane. Combining ArgoCD and Crossplane is a popular approach for Kubernetes applications.

Designing Your GitOps Strategy

Observability & Monitoring

Observability and monitoring are related concepts but have distinct differences in their approach and scope. Monitoring provides a predefined set of metrics and indicators for tracking system health, while observability offers a more flexible and exploratory approach to gain insights into system behavior and diagnose issues effectively.

Observability refers to the ability to understand and infer the internal state and behavior of a system based on its external outputs, events, and data. It involves gathering and analyzing various types of data including logs, metrics, and traces in order to gain insights into a system's performance, health, and other issues. Paired with open source monitoring tools like OpenTelemetry, developers can gain real-time visibility of the system's state and changes.

Observability complements monitoring by providing a broader and more detailed view of the system, allowing for better understanding and troubleshooting of complex and dynamic environments. This allows teams to monitor and troubleshoot issues more effectively. Additionally, since all changes are recorded in Git, auditing and compliance requirements can be easily met.

Infrastructure as Code (IaC)

One very common practice teams adopt for their GitOps strategy is implementing IaC tooling and best practices. The process involves managing and provisioning infrastructure resources using code rather than manual processes or interactive configuration tools. Ultimately this means codifying your workload by defining every application and infrastructure resource as declarative configuration files, allowing for rapid, repeatable, and error-free provisioning of infrastructure.

Popular tooling to support IaC includes Terraform, Pulumi, Ansible, Salt, and Puppet. These tools provide the necessary capabilities to define, provision, and manage infrastructure resources using code.

Maintain Separate Repositories for Application Code and Infrastructure Code

Having separate repositories for application code and infrastructure code is a best practice. This promotes modularity, independence, collaboration, versioning, and security. It allows for efficient development, testing, and deployment of both application and infrastructure components while ensuring clear separation and control over changes in each layer of the system.

User Permissions and Access Controls

To operate and maintain a GitOps system effectively, you will need a certain level of visibility. Your team will need to detect, diagnose, and resolve issues promptly, ensuring the system's stability, reliability, and scalability.

It's important to implement appropriate permissions models and access controls to ensure the security and integrity of your code and infrastructure. Common practices include role-based access control (RBAC), Git repository permissions, branch protection, secure practices for managing and storing sensitive information like API keys and strong passwords, infrastructure access control, multi-factor authentication (MFA), and continuous monitoring and observability.

It is crucial to regularly review and update permissions and access controls based on changes in team compositions, responsibilities, or security requirements. Also, you can further enhance your security posture by educating team members about security practices and enforcing strong password policies.

Selecting Tools and Common Practices

With these three foundational elements in mind, you are ready to select your tools and preferred practices. GitOps is fundamentally about managing your workload the same way you manage your codebase.

Selecting the right tools will depend on your choice of deployment strategy. In a push-based approach, only the automation tooling needs write access to the environment. In a pull-based approach, the environment just needs read access to the git repository. This greatly reduces the attack surface as it eliminates the need for most individuals or teams to have direct access to the environment. A pool of contributors needs nothing more than existing repository-level git permissions.

Using GitOps with Akamai Cloud Computing

Akamai cloud computing makes it easy to get started with the IaC tools that are integral to GitOps. Get a jump start with our Try IaC ebook and on-demand course taught by Justin Mitchel from Coding for Entrepreneurs.

As you explore what tools and practices you want to see in your GitOps strategy, be sure to remember those foundational elements: stay true to the core GitOps principles, understand your deployment strategy, and be sure to empower your team with observability to keep your systems resilient and reliable.

A GitOps approach complements the rapidity and flexibility of cloud-native development, and you don't need to completely re-architect your application to start implementing a GitOps strategy. Repeatable processes and unified deployment methodologies allow teams to work asynchronously without sacrificing collaboration–this has become the key to bringing stable and scalable products to market faster.

Here are other resources to help you get started with Git to create your own GitOps pipeline.

Resources

- Set up Git (GitHub documentation)
- Install GitLab with Docker
- Resolve Merge Conflicts in Git
- View all Git guides in Akamai's documentation library
- Learn how to manage Kubernetes resources with Terraform

About Akamai

Akamai accelerates innovation with scalable, simple, affordable, and accessible Linux cloud solutions and services. Our products, services, and people give developers and enterprises the flexibility, support, and trust they need to build, deploy, secure, and scale applications more easily and cost-effectively from cloud to edge on the world's most distributed network.

www.akamai.com
www.linode.com

Cloud Computing Services Developers Trust