Reference Architecture

Monitoring and Logging for Docker Enterprise Edition

Version 1.1

March 2017

Sematext Group Inc.

Table of contents

1 Introduction
- 1.1 Monitoring and Logging Designed for Docker
- 1.1 Sematext Docker Agent

2 What you will Learn
3 Understand Key Docker Metrics
- 3.1 Operating System Metrics for each Node
  - 3.1.1 Host CPU
  - 3.1.2 Host Memory
  - 3.1.3 Host Disk Space
- 3.2 Total Number of Running Containers
- 3.3 Container Metrics
  - 3.3.1 Container CPU - Throttled CPU Time
  - 3.3.2 Container Memory - Fail Counters
  - 3.3.3 Container Memory Usage
  - 3.3.4 Container Swap
  - 3.3.5 Container Disk I/O
  - 3.3.6 Container Network Metrics

4 Assumptions
5 Requirements
6 Prerequisites
7 Monitoring & Logging Deployment
- 7.1 Docker Remote API Integration Options
- 7.2 Summary

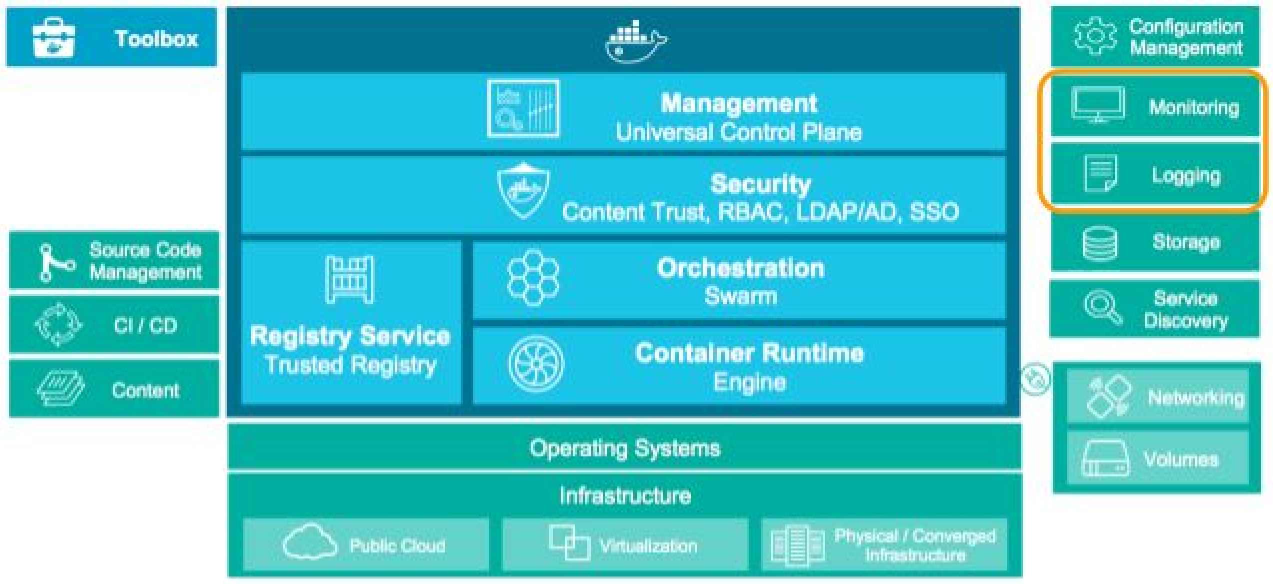
8 Solution Deployment
- 8.1 Connect Docker-Compose to Docker UCP Cluster
- 8.2 Configure Sematext Docker Agent for Docker UCP
  - 8.2.1 Create SPM and Logsene Apps
  - 8.2.2 Deploy the Agent to all Docker UCP Nodes
  - 8.2.3 Voila! Your Docker Monitoring & Logging Works
  - 8.2.4 Connect Metrics and Logs for Faster Troubleshooting

9 Configuring Sematext Docker Agent
- 9.1 Connection to SPM and Logsene
  - 9.1.1 SPM and Logsene in the Cloud (SaaS)
  - 9.1.2 SPM and Logsene On Premises
- 9.2 Log Handling Options
  - 9.2.1 Blacklisting and Whitelisting Logs
  - 9.2.2 Automatic Parser for Container Logs
  - 9.2.3 Log Routing with Docker Labels
  - 9.2.4 Masking Sensitive Data in Logs
  - 9.2.5 Automatic Geo-IP Enrichment for Container Logs
  - 9.2.6 Post-processing Parsed Logs with JavaScript Functions

10 Summary
11 References
Appendix A - Sematext Docker Agent Configuration Options

1 Introduction

Docker Enterprise Edition (Docker EE) simplifies container orchestration and increases the flexibility and scalability of application deployments. However, the high level of automation create new challenges for monitoring and log management. Why? Because each container typically runs a single process, has its own environment, utilizes virtual networks, or has various methods of managing storage. Traditional monitoring solutions take metrics from each server and applications they run. These servers and applications running on them are typically very static, with very long uptimes. Docker deployments are different: a set of containers may run many applications, all sharing the resources of one or more underlying hosts. It's not uncommon for Docker servers to run many short-term containers for batch jobs, while a set of permanent services runs in parallel. Traditional monitoring tools not used to such dynamic environments are not suited for such deployments. On the other hand, some modern monitoring solutions were built with such dynamic systems in mind and even have out of the box reporting for Docker monitoring. Moreover, container resource sharing calls for stricter enforcement of resource usage limits, an additional issue you must watch carefully. To make appropriate adjustments for resource quotas you need good visibility into any limits containers have reached or errors they have caused. We recommend using alerts according to defined limits; this way you can adjust limits or resource usage even before errors start happening.



1.1 Monitoring and Logging Designed for Docker

Docker UCP includes only real-time monitoring of the cluster state, real-time metrics and logs for each container. Operating larger infrastructures requires a longer retention time for logs and metrics and the capability to correlate metrics, logs and events on several levels (cluster, nodes, applications and containers). A comprehensive monitoring and logging solution ought to provide the following operational insights:

- **Auditing of Docker events.** Tracking of all Docker events provides a clear view of containers life cycle. For example, by collecting events you gain insight into what happens with your containers during (re)deployments or the re-scheduling of containers to different nodes. Some containers might be configured for automatic restarts and the events could indicate whether container processes crash frequently. In case of out-of-memory events, it might be wise to modify the memory limits or check with developers why this event happened. Docker Events also carry information critical for the security of applications, such as:
    - Version changes of application images
    - Application shutdowns
    - Changes of storage volumes or network settings
    - Deletion of storage volumes, which might cause data loss

- **Resource usage for capacity planning and tuning.** The resource management with Docker is one of the main advantages of running multi-tenant workflows on shared resources. To do so, definitions for resource limits like CPU, IO and Memory are required. Many organisations face the challenge that they don’t know the exact requirements of their Dockerized applications, typically because they might have been deployed in other ways in the past. At this point monitoring the resources required by containers helps one determine the right limits, as well as observe whether the assumed limits are truly appropriate.

- **Detailed metrics for cluster nodes and containers.** Having detailed metrics helps optimize application resource usage. Detailed metrics are the basis for defining application-specific alert criteria for any critical resources applications depend on. Metrics are aggregated for all hosts, images and containers and are filterable by hosts, images, and containers. This lets you drill down from a cluster view down to a single container while troubleshooting or simply trying to understand operations details. Long retention times for metrics make it possible to compare resources before and after different deployments and releases or troubleshoot problems that appear only when a service has been running over several days or weeks!

- **Centralized log management with full-text search, filtering, and analytics across all containers.** Logs should be collected, parsed and shipped to an indexing engine. The integrated charting functions in Logsene and integrations for Kibana and Grafana make it easy to analyze logs collected in Docker EE.

- **Anomaly detection and alerts for all logs and metrics.** Anomaly detection can help reduce the noise and alert fatigue often caused by classic threshold-based alerts. Even log-alerting is possible with Logsene e.g. to detect anomalies in the log frequency of a specific query. For example, a search for "error" in the system might normally return a dozen non-critical errors, which could be ignored. An increase of error logs indicates that something might be going wrong. Another type of alerts is the Heartbeat alert for all cluster nodes. Disk space alerts are very useful for Docker nodes, because Docker images might consume a lot of disk space. Docker EE runs some cleanup agents to remove unused containers and images; nevertheless the default disk-space alert created by SPM provides an early warning before the capacity limit is reached.

- **Long retention time for logs, metrics and events.** Comparing metrics and logs during deployments or watching the performance under different workloads requires one to store logs and metrics for a reasonable time. We have seen cases where memory leaks started to get serious after a few weeks of stable operations, although initially they were not detected. In such cases all context information like logs, events and metrics could be very valuable in identifying the root cause of such problems.

1.1 Sematext Docker Agent

The Docker Datacenter architecture is open for extensions, such as Monitoring and Logging. This document explains how Docker Datacenter can be extended with Sematext SPM for Docker Performance Monitoring [1] and Logsene [2] for Log Management. More specifically, we will use the open-source Sematext Docker Agent to get all data from hosts and containers to have the complete Docker monitoring and logging solution.

Sematext Docker Agent is a modern, Docker-aware metrics, events, and log collection agent. It runs as a tiny container on every Docker host collecting logs, metrics and events for all cluster hosts and all containers from the Docker Remote API. It auto-discovers all containers including the containers for Docker UCP services. Sematext Docker Agent streams metrics, events, and logs via TLS (HTTPS) to SPM and Logsene. After the deployment of the agent all logs and metrics are immediately available in SPM and Logsene.

2 What you will Learn

In this reference architecture document, you will find out about all key Docker metrics to watch. Following that, you will learn how to set up monitoring and logging for a Docker UCP cluster. Specifically, this document shows how to use Sematext Docker Agent to collect metrics, events and logs for all nodes and containers.

3 Understand Key Docker Metrics

3.1 Operating System Metrics for each Node

3.1.1 Host CPU

Understanding the CPU utilization of hosts and containers helps one optimize the resource usage of Docker UCP nodes. The container CPU usage can be throttled in order to avoid a single busy container slowing down other containers by taking away all available CPU resources. Throttling the CPU time is a good way to ensure a minimum of processing power for essential services - it's like the good old nice levels in Unix/Linux.

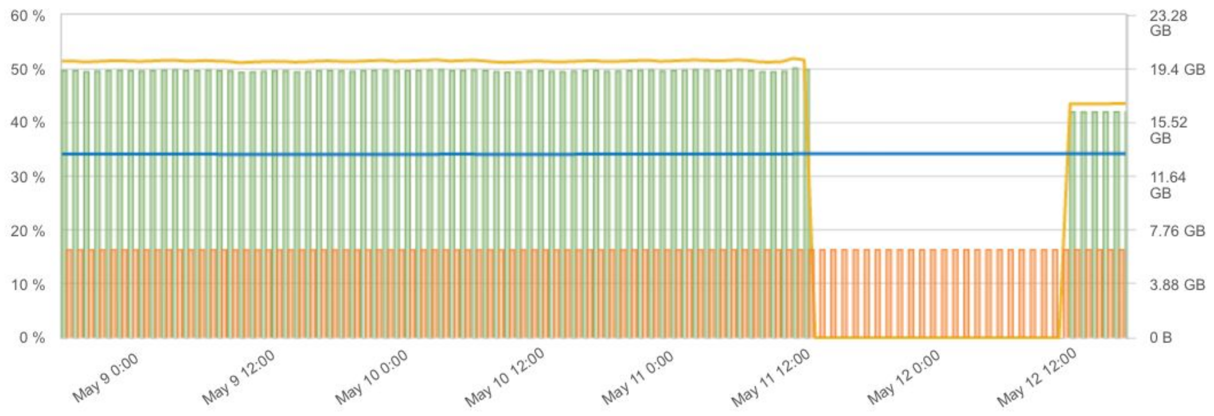
When the resource usage is optimized, a high CPU utilization might actually be expected and even desired, and alerts might make sense only for when CPU utilisation drops (service outages) or increases for a longer period over some max limit (e.g. 85%).

3.1.2 Host Memory

The total memory used in each Docker UCP node is important to know for the current operations and for capacity planning. Dynamic cluster managers like Docker Swarm use the total memory available on the node and the requested memory for containers to decide on which host a new container should ideally be launched. Deployments might fail if a cluster manager is unable to find a host with sufficient resources for the container. That's why it is important to know the host memory usage and the memory limits of containers. Adjusting the capacity of new cluster nodes according to the footprint of Docker applications could help optimize the resource usage.

3.1.3 Host Disk Space

Docker images and containers consume additional disk space. For example, an application image might include a Linux operating system and might have a size of 150-700 MB depending on the size of the base image and installed tools in the container. Persistent Docker volumes consume disk space on the host as well. In our experience watching the disk space and using cleanup tools is essential for continuous operations of Docker hosts.

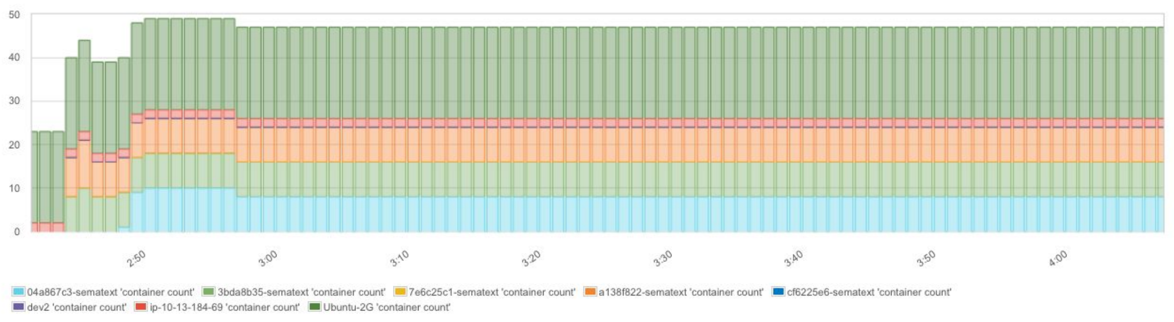


Disk Space used on two nodes

Because disk space is very critical it makes sense to define alerts for disk space utilization to serve as early warnings and provide enough time to clean up disks or add additional volumes. For example, SPM automatically sets alert rules for disk space usage for you, so you don't have to remember to do it.
A good practice is to run tasks to clean up the disk by removing unused containers and images frequently.

3.2 Total Number of Running Containers

The current and historical number of containers is an interesting metric for many reasons. For example, it is very handy during deployments and updates to check that everything is running like before.
When Docker UCP automatically schedules containers to run on different hosts using different scheduling policies, the number of containers running on each host can help one verify the activated scheduling policies. A stacked bar chart displaying the number of containers on each host and the total number of containers provides a quick visualization of how the cluster manager distributed the containers across the available hosts.



Container counts per Docker host over time

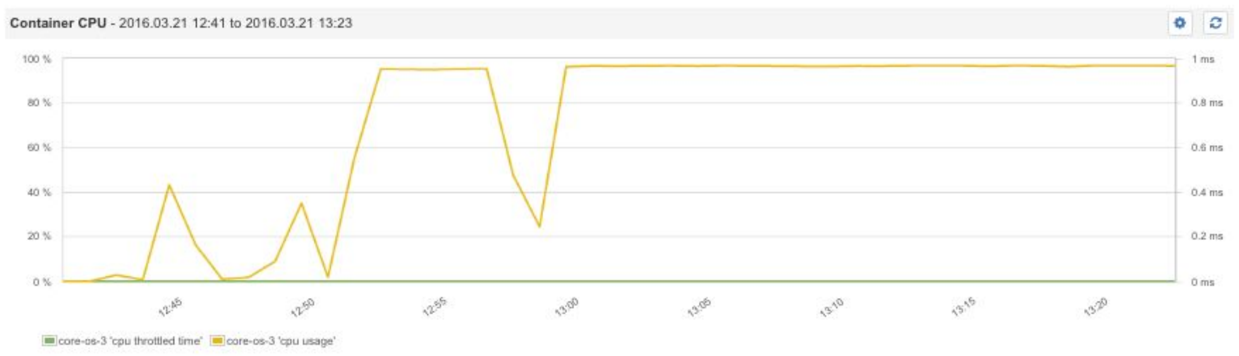
This metric can have different "patterns" depending on the use case. For example, batch jobs running in containers vs. long running services commonly result in different container count patterns. A batch job typically starts a container on demand, or starts it periodically, and the container with that job terminates after a relatively short time. In such a scenario one might see a big variation in the number of containers running resulting in a "spiky" container count metric. On the other hand, long running services such as web servers or databases typically run until they get re-deployed during software updates. Although scaling mechanisms might increase or decrease the number of containers depending on load, traffic, and other factors, the container count metric will typically be relatively steady because in such cases containers are often added and removed more gradually. Because of that, there is no general pattern we could use for a default Docker alert rule on the number of running containers.
Nevertheless, alerts based on anomaly detection, which detect sudden changes in the number of the containers in total (or for specific hosts) in a short time window, can be very handy for most of the use cases. The simple threshold-based alerts make sense only when the maximum or minimum number of running containers is known, and in dynamic environments that scale up and down based on external factors, this is often not the case.

3.3 Container Metrics

Container metrics are basically the same metrics available for every Linux process, but include limits set via cgroups by Docker, such as limits for CPU or memory usage. Please note that sophisticated monitoring solutions like SPM for Docker are able to aggregate Container Metrics on different levels like Docker Hosts/Cluster Nodes, Image Name or ID and Container Name or ID. Having the ability to do that makes it easy to track resources usage by hosts, application types (image names) or specific containers. In the following examples we might use aggregations on various levels.

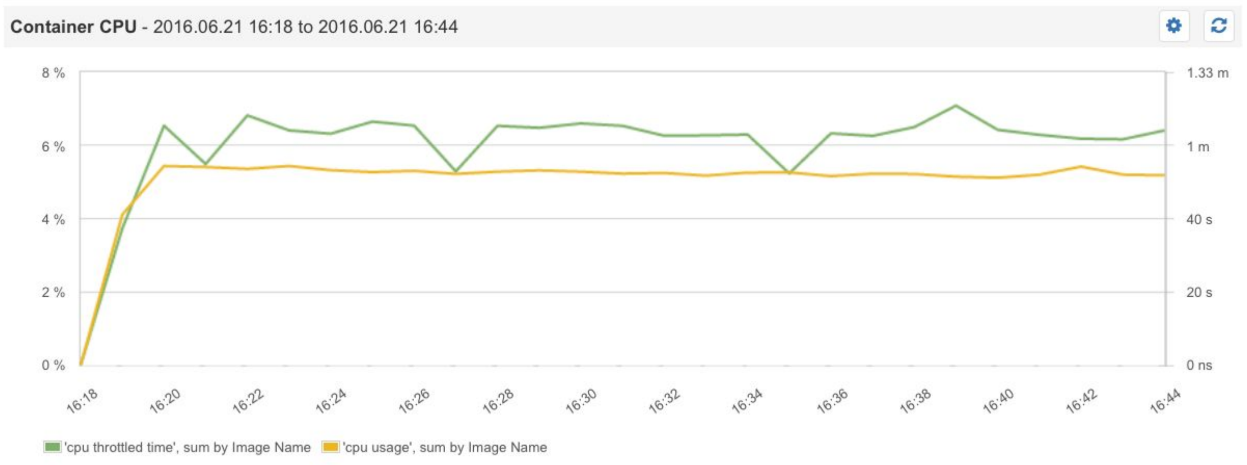
3.3.1 Container CPU - Throttled CPU Time

One of the most basic bits of information is information about how much CPU is being consumed by all containers, images, or by specific containers. A great advantage of using Docker is the capability to limit CPU utilisation by containers. Of course, you can't tune and optimize something if you don't measure it, so monitoring such limits is the prerequisite. Observing the total time that a container's CPU usage was throttled provides the information one needs to adjust the setting for CPU shares in Docker. Please note that CPU time is throttled only when the host CPU usage is maxed out. As long as the host has spare CPU cycles available for Docker it will not throttle containers' CPU usage. Therefore, the throttled CPU is typically zero and a spike of this metric is a typically a good indication of one or more containers needing more CPU power than the host can provide.



Container CPU usage and throttled CPU time

The following screenshot shows containers with 5% CPU quota using the command `docker run --cpu-quota=5000 nginx`, we see clearly how the throttled CPU grows until it reaches around 5%, enforced by the Docker engine.

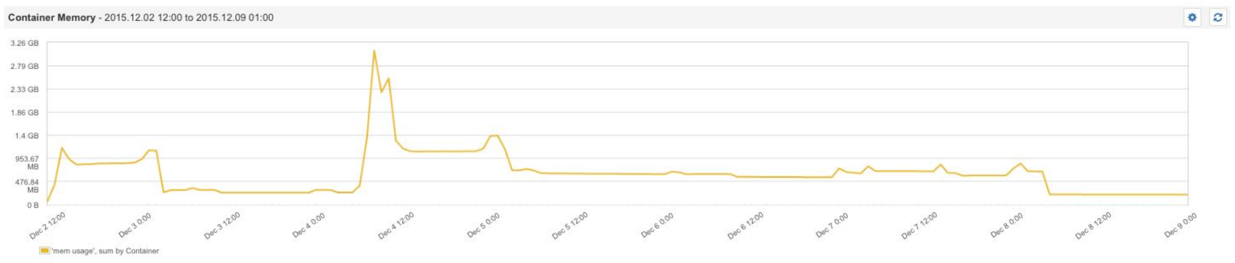


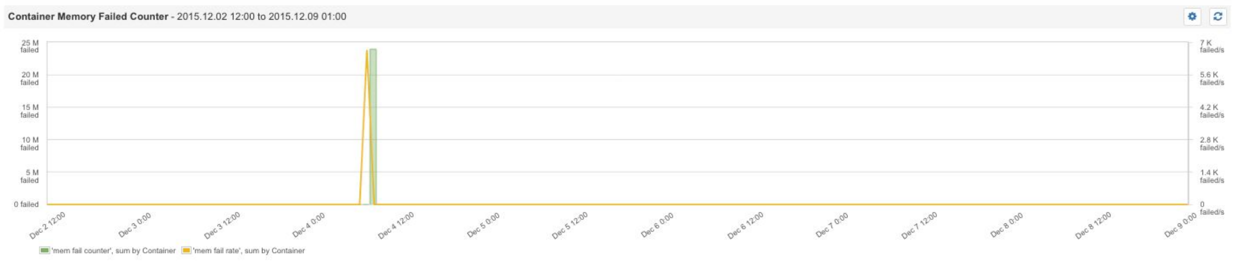
Container CPU usage and throttled CPU time with CPU quota of 5%

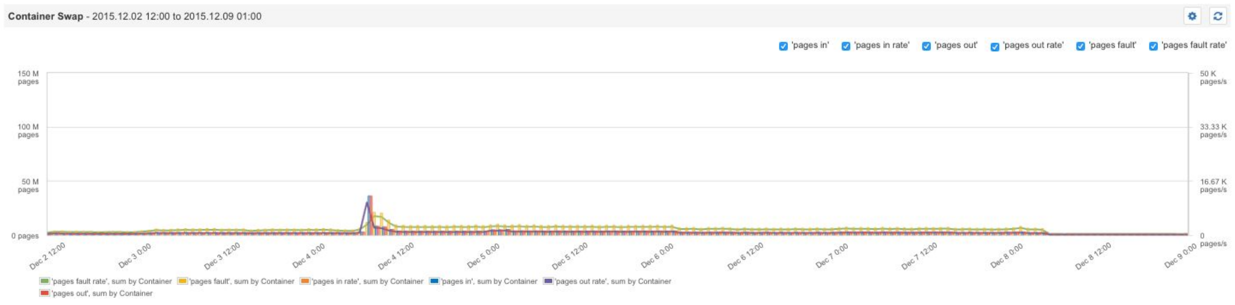
3.3.2 Container Memory - Fail Counters

It is a good practice to set memory limits for containers. Doing that helps avoid a memory-hungry container taking all available memory and starving all other containers on the same server. Runtime constraints on resources can be defined in the Docker run command. For example, "-m 300M" sets the memory limit for the container to 300 MB. Docker exposes a metric called container memory fail counters. This counter is increased each time memory allocation fails -- that is, each time the pre-set memory limit is hit. Thus, spikes in this metric indicate one or more containers needing more memory than was allocated. If the process in the container terminates because of this error, we might also see out of memory events from Docker.

A spike in memory fail counters is a critical event and putting alerts on the memory fail counter is very helpful to detect wrong settings for the memory limits or to discover containers that try to consume more memory than expected.





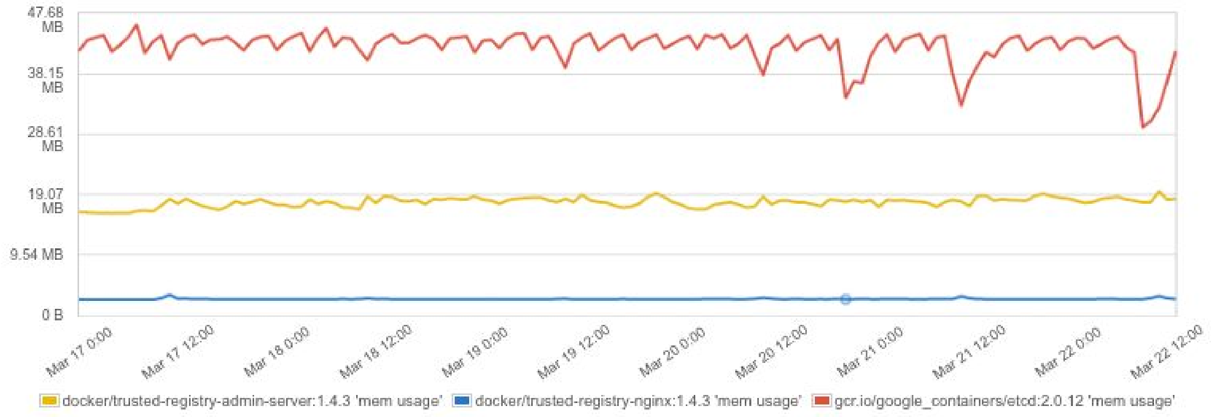


Container Memory, Memory Fail counters and Swap activity

3.3.3 Container Memory Usage

Different applications have different memory footprints. Knowing the memory footprint of the application containers is important for having a stable environment. Container memory limits ensure that applications perform well, without using too much memory, which could affect other containers on the same host. The best practice is to tune memory setting in a few iterations:

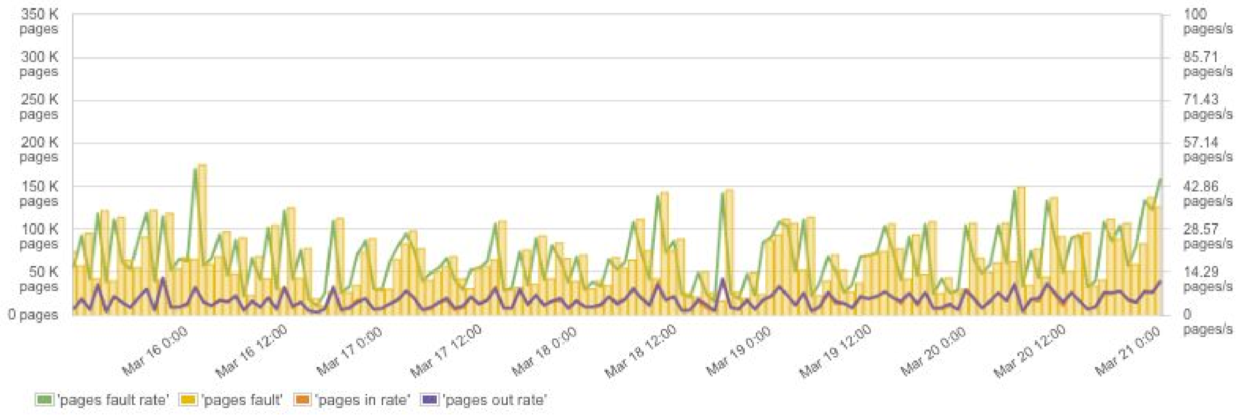
1. Monitor memory usage of the application container
2. Set memory limits according to the observations
3. Continue monitoring of memory, memory fail counters, and Out-Of-Memory events. If OOM events happen, the container memory limits may need to be increased, or debugging is required to find the reason for the high memory consumptions.



Container memory usage

3.3.4 Container Swap

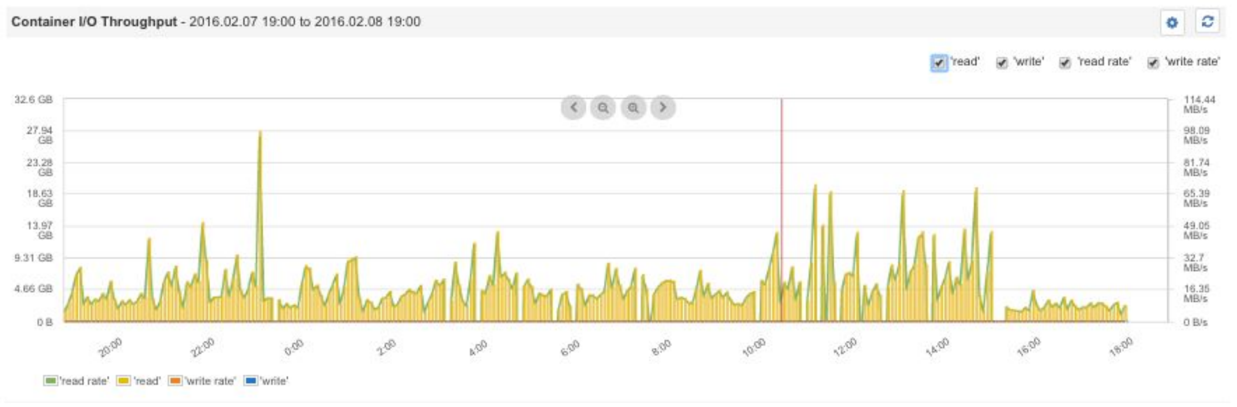
Like the memory of any other process, a container's memory could be swapped to disk. For applications like Elasticsearch or Solr one often finds instructions to deactivate swap on the Linux host - but if you run such applications on Docker it might be sufficient just to set "--memory-swap=-1" in the Docker run command!



Container swap, memory pages, and swap rate

3.3.5 Container Disk I/O

In Docker multiple applications use the same resources concurrently. Thus, watching the disk I/O helps one define limits for specific applications and give higher throughput to critical applications like data stores or web servers, while throttling disk I/O for batch operations. For example, the command `docker run -it --device-write-bps /dev/sda:1mb mybatchjob` would limit the container disk writes to a maximum of 1 MB/s.

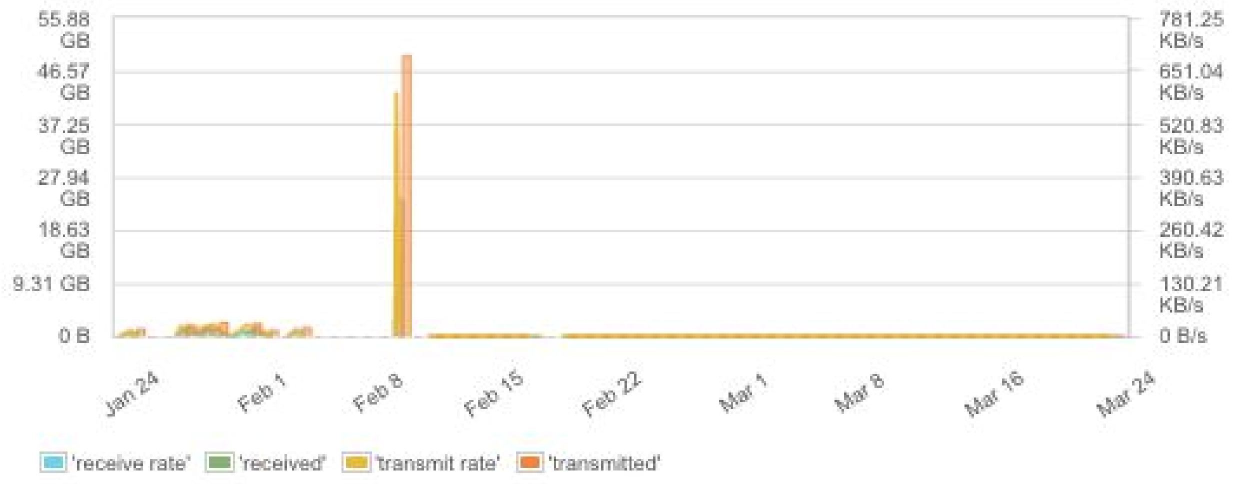


Container I/O throughput

3.3.6 Container Network Metrics

Networking for containers can be very challenging. By default all containers share a network, or containers might be linked together to share a separated network on the same host. However, when it comes to networking between containers running on different hosts an overlay network is required, or containers could share the host network. Having many options for network configurations means there are many possible causes of network errors.

Moreover, not only errors or dropped packets are important to watch out for. Today, most of the applications are deeply dependent on network communication. Throughput of virtual networks could be a bottleneck especially for containers like load balancers. In addition, the network traffic might be a good indicator how much applications are used by clients and sometimes you might see high spikes, which could indicate denial of service attacks, load tests, or a failure in client apps. So watch the network traffic - it is a useful metric in many cases.



Network traffic and transmission rates

4 Assumptions

This reference architecture assumes the reader already has a working understanding of Docker EE, in particular the following components: Docker Universal Control Plane, Swarm, and Compose. If you are not familiar with them, please refer to the following resources to gain the basic understanding:

- Docker Universal Control Plane at docs.docker.com/ucp
- Docker Swarm at docs.docker.com/swarm
- Docker Compose at docs.docker.com/compose
- Docker Daemon is configured for UNIX domain sockets e.g. "docker daemon -H `unix:///var/run/docker.sock`"

5 Requirements

There are software version requirements for this reference architecture. Other variations have not been tested or validated. For more details on software compatibility and interoperability please go to Compatibility Matrix page.

- Docker UCP v2.x
- Docker Engine 1.12
- Free account for Sematext SaaS or On Premises SPM/Logsene installation (email sales@sematext.com or call +1 (347) 480 1610 for a free SPM/Logsene evaluation)

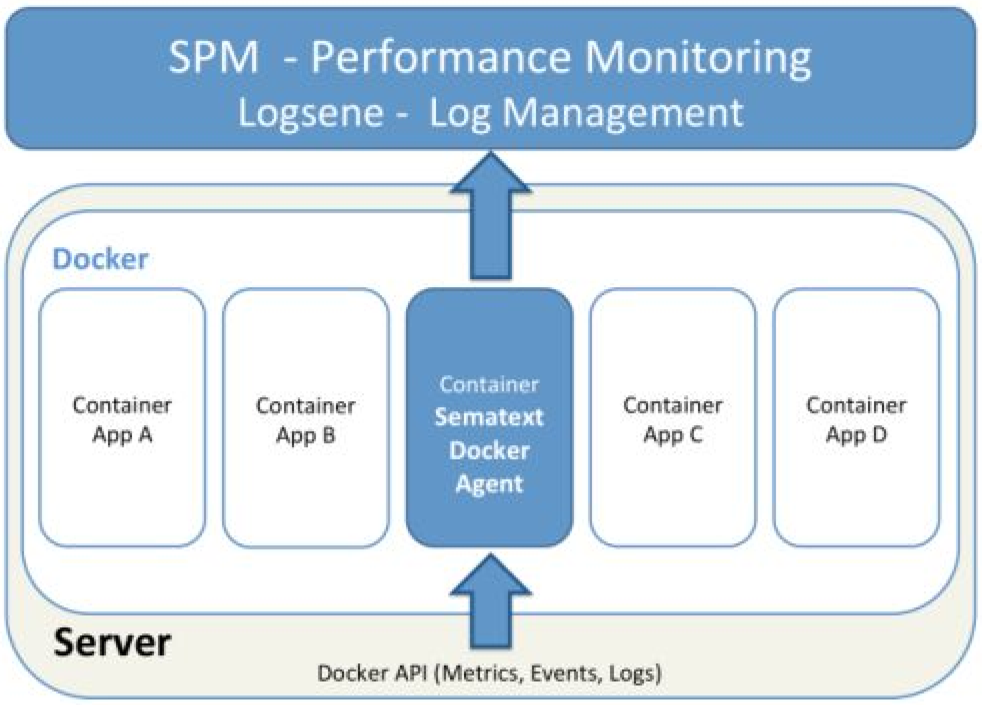
6 Prerequisites

For this reference architecture, you will need the following environment set up.

- Installed Docker UCP; see installation requirements.
- Unrestricted or proxy network connection between the Docker UCP nodes and SPM On Premises server or Sematext SaaS servers: spm-receiver.sematext.com:443, logsene-receiver.sematext.com:443
- Network reachability to the UCP controller nodes on TCP port 80/443.
- Docker and Docker Compose client
    - installation instructions: https://docs.docker.com/compose/install/
    - Linux or Mac OS X terminal with bash and docker client.

7 Monitoring & Logging Deployment

Sematext Docker Agent runs as a tiny Docker container on each Docker node. From there it calls Docker API to get logs, metrics, and events for all containers running on the same node, as well as the node itself. It then streams those data to Sematext SPM and Logsene over an encrypted connection.



7.1 Docker Remote API Integration Options

There are several options to connect Sematext Docker Agent to the Docker Remote API, depending on the configuration of the Docker daemon:

- Connect to Docker host via UNIX domain socket
- Connect to Docker host via TLS socket

- Connect to Docker host via TCP socket
- Connect to the Docker UCP Remote API endpoint (proxy to Docker Swarm) via TLS

Sematext Docker Agent supports all of these methods, please refer to Appendix A - Configuration options for Sematext Docker Agent. For TLS connections, the agent requires access to the TLS certificates (e.g. mounted to a volume). The UNIX socket connection requires access permission to Docker's UNIX socket.

Keeping in mind that Docker UCP cluster looks like a single Docker host from the Docker Remote API point of view, it should be very easy to monitor Docker UCP / Swarm with existing Docker monitoring tools! Connecting a monitoring agent to the Swarm Master API endpoint is one potential option. The Sematext Docker Agent would collect all container metrics, events and all logs from the Swarm Master. The following considerations lead to the requirement to have the monitoring and logging agent running on each Docker UCP node:

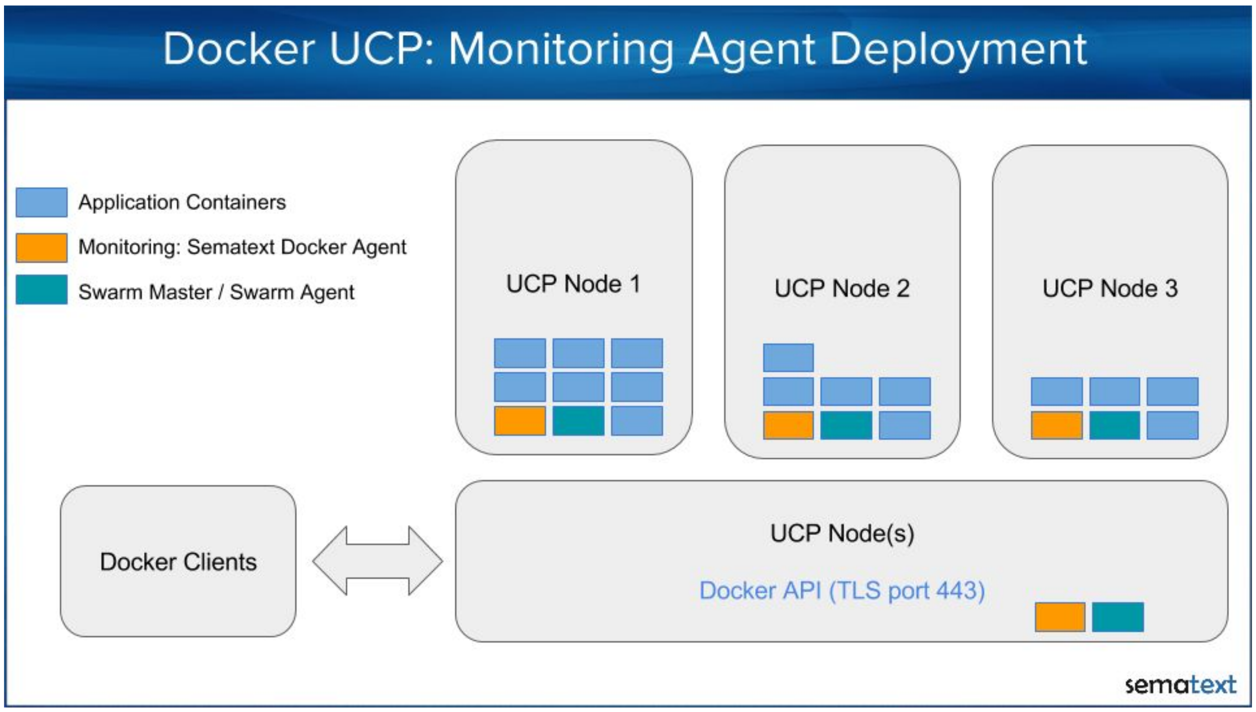
- If a single monitoring agent were to connect only to the master node, it would miss host metrics for all other nodes because the Docker API doesn't provide any host metrics. We could also not see how much memory, disk space, or CPU the Docker UCP / Swarm node itself consumes. Solution: deploy the monitoring agents to each node for collecting the host metrics locally.
- In a larger cluster with a high volume of logs, events and metrics to collect, a single monitoring agent connected to the master node would need to handle all operational data of the cluster. This would work for a small cluster but such an architecture would obviously be destined for failure on larger clusters. It's much better to have an agent running on each node and spread the monitoring and logging work over all nodes. Another positive side-effect of this is that there is no need to change the deployment strategy later, when the cluster scales out.
- If the monitoring agent were to lose the connection to the master node, the monitoring of all containers would fail. To avoid a single point of failure it makes sense to monitor each node individually.

7.2 Summary

Monitoring and logging agent should run on every node for the following reasons:

- Collection of performance metrics on each Docker UCP node provides complete information about the node and the containers running on each node
- Load sharing of monitoring and logging workloads

- No single point of failure
- The connection from the agent to the Docker Remote API should be established to the local Docker daemon via UNIX socket and not to the Docker UCP endpoint with TLS


Docker UCP: Monitoring Agent Deployment
Application Containers
Monitoring: Sematext Docker Agent
Swarm Master / Swarm Agent
UCP Node 1
UCP Node 2
UCP Node 3
Docker Clients
UCP Node(s)
Docker API (TLS port 443)
sematext

Target setup

To deploy Sematext Docker Agent to all Docker UCP nodes we will use docker-compose scale command, connected to the Docker API endpoint for the Docker UCP cluster. See chapter 8 for details.

8 Solution Deployment

The deployment of Sematext Docker Agent requires three steps:

1	**Connect docker-compose client to Docker UCP**	Download client bundle from Docker UCP user profile. Set the environment for the docker CLI with the certificates and scripts included in the client bundle.
2	**Create SPM and Logsene tokens for Sematext Docker Agent**	1. Obtain SPM and Logsene App Tokens from Sematext 2. Set container constraints with a negative affinity to the container name to ensure that Sematext Docker Agent runs only once on each node
3	**Deploy Sematext Docker Agent**	Deploy the monitoring agent to all cluster nodes as global service

As soon the agent is deployed it ships metrics, events and logs collected via the local Docker Remote API endpoint (e.g. /var/run/docker.sock) to the configured backend. In this example we don't configure specific SPM and Logsene servers, and rely on the default values for SaaS.

8.1 Connect Docker-Compose to Docker UCP Cluster

After installing UCP, you can run the docker command against UCP cluster nodes. Since all traffic between users and UCP nodes is secure and authenticated, when using the Docker CLI client, you'll need to provide client certificates.
Download the bundle that contains the client certificates for a user from the UCP web app:

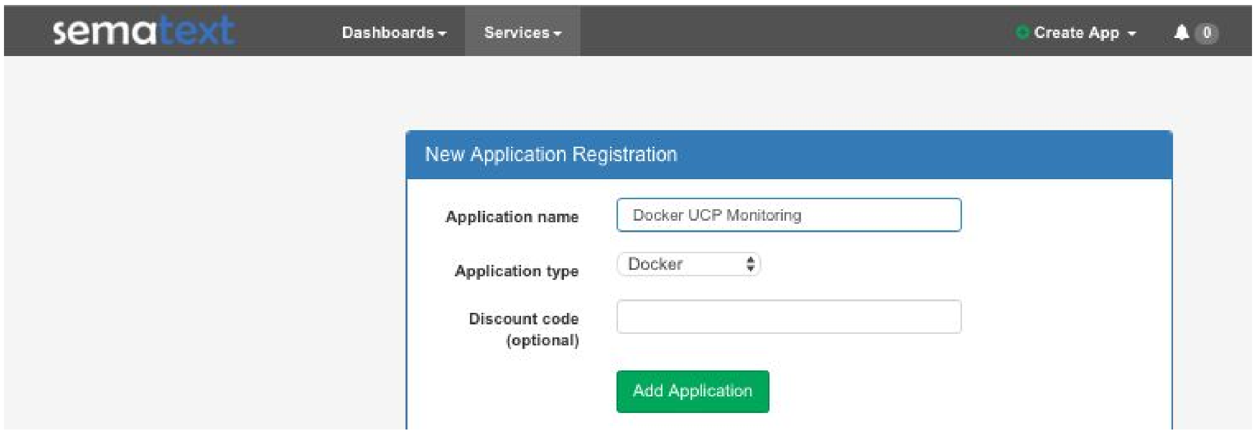
1. If you haven't already done so, log into UCP
2. Navigate to your profile
3. As an example, if you're logged in as the Admin user, on the right-hand menu, navigate to Admin > Profile
4. Click the Create Client Bundle button
5. The browser downloads the ucp-bundle-admin.zip file
6. Copy the client bundle in your working directory and extract the files
   ```
   $ cp ~/Downloads/ucp-bundle-admin.zip .
   $ unzip ucp-bundle-admin.zip
   ```
7. Run the included shell script to set the environment variables to access UCP with docker CLI client
   ```
   $ source env.sh
   ```
8. Check the settings by running "docker info" to see if docker CLI is connected to UCP
   ```
   $ docker info
   ```

8.2 Configure Sematext Docker Agent for Docker UCP

Sematext Docker Agent is configured via environment variables. Appendix A lists all configuration options (e.g. filter for specific images and containers), but we'll keep it simple here.

8.2.1 Create SPM and Logsene Apps

1. Sign up for free at apps.sematext.com, if you haven't done that already
2. Create an SPM App of type "Docker" to obtain the SPM App Token. SPM App will hold your Docker UCP performance metrics.



New Application Registration

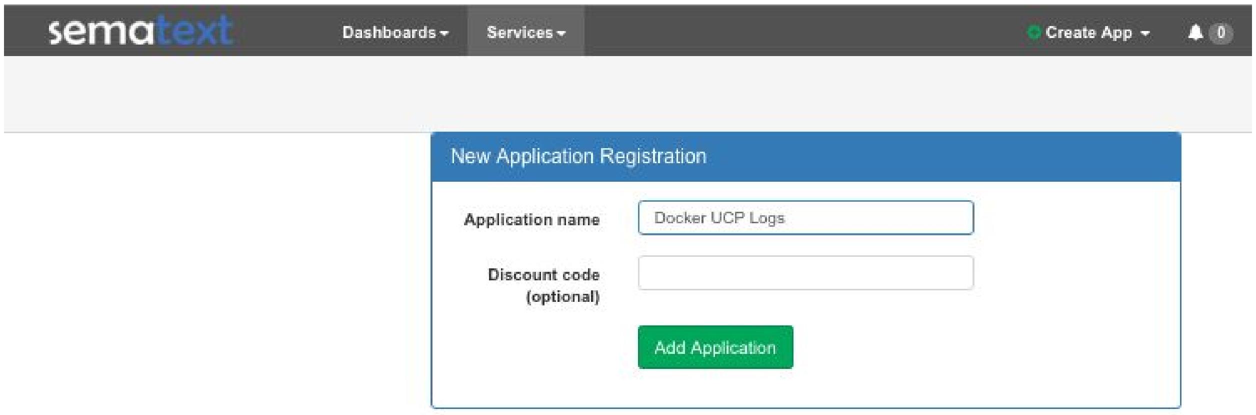
Application name: Docker UCP Monitoring

Application type: Docker

Discount code (optional):

Add Application

3. Create a Logsene App to obtain the Logsene Token. Logsene App will hold your Docker UCP logs.



New Application Registration

Application name: Docker UCP Logs

Discount code (optional):

Add Application

8.2.2 Deploy the Agent to all Docker UCP Nodes

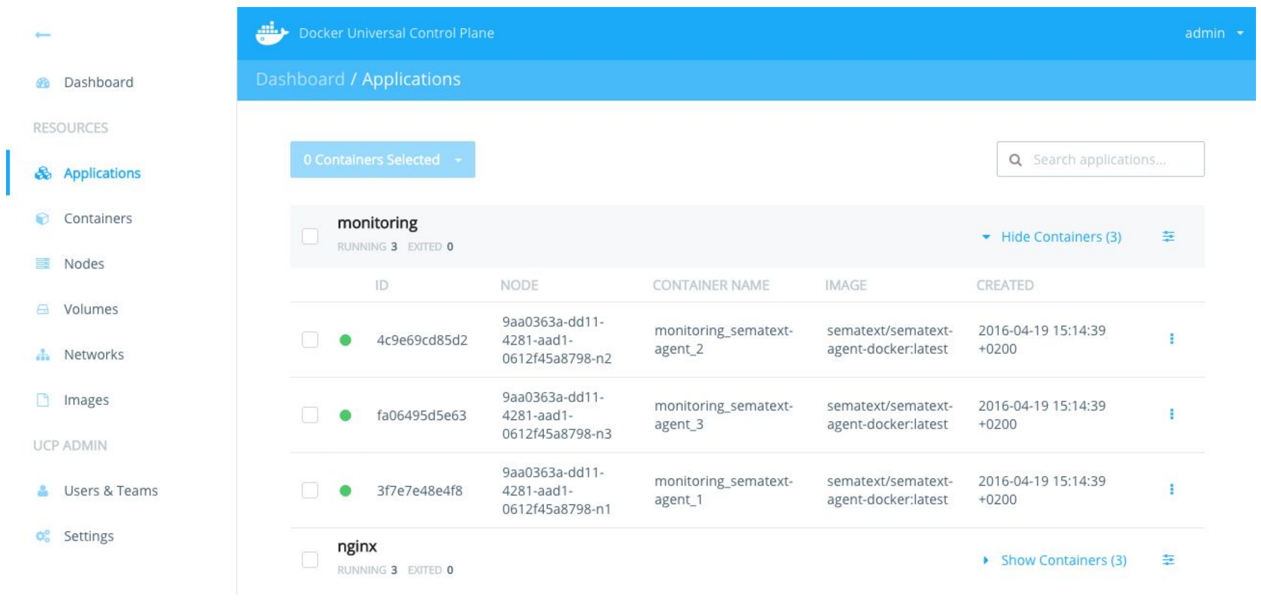
To deploy Sematext Docker Agent to all nodes, we have to run a global Swarm service, including the Logsene and SPM application tokens, and access to the local docker socket on each node:

```
docker service create --mode global --name sematext-agent-docker \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
-e SPM_TOKEN="REPLACE THIS WITH YOUR SPM TOKEN" \
-e LOGSENE_TOKEN="REPLACE THIS WITH YOUR LOGSENE TOKEN" \
sematext/sematext-agent-docker
```

If you have to use a proxy add `-e HTTPS_PROXY=https://your-proxy-server:port` to the service command above.

The command outputs the Docker Swarm service ID, which could be used with the docker service command. If you check the status of the service with `docker service ps SERVICE_ID` you’ll see sematext-docker-agent getting scheduled for deployment on each node shortly after running the above command.

Note: The global services ensures that Sematext Docker Agents gets deployed to each node and every new node added to Docker UCP.



monitoring
RUNNING 3 EXITED 0

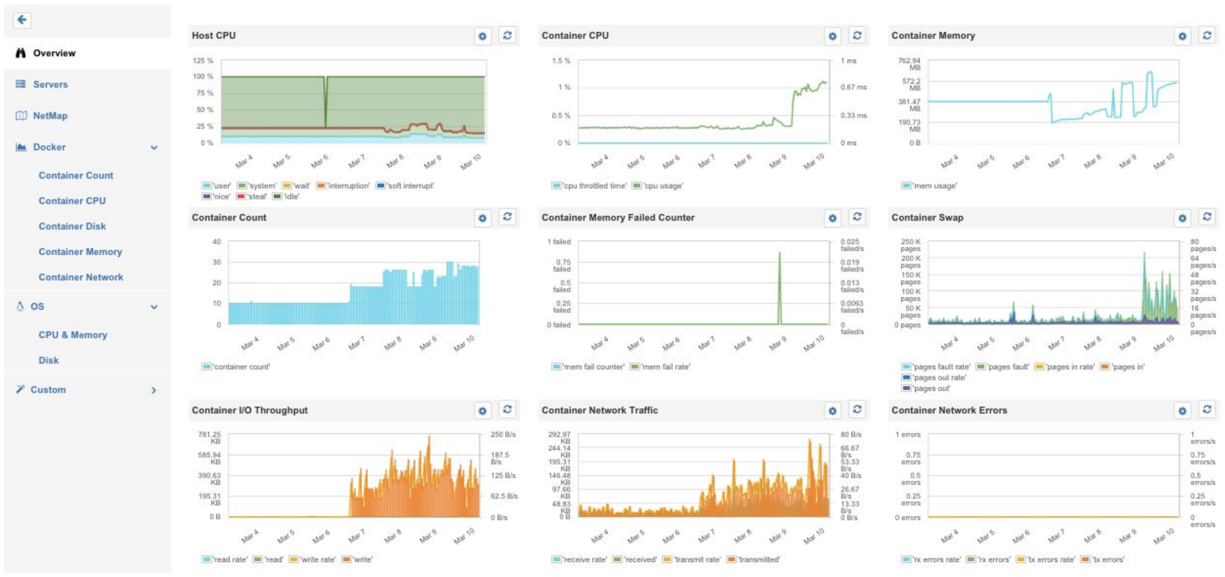
ID	NODE	CONTAINER NAME	IMAGE	CREATED
4c9e69cd85d2	9aa0363a-dd11-4281-aad1-0612f45a8798-n2	monitoring_sematext-agent_2	sematext/sematext-agent-docker:latest	2016-04-19 15:14:39 +0200
fa06495d5e63	9aa0363a-dd11-4281-aad1-0612f45a8798-n3	monitoring_sematext-agent_3	sematext/sematext-agent-docker:latest	2016-04-19 15:14:39 +0200
3f7e7e48e4f8	9aa0363a-dd11-4281-aad1-0612f45a8798-n1	monitoring_sematext-agent_1	sematext/sematext-agent-docker:latest	2016-04-19 15:14:39 +0200

nginx
RUNNING 3 EXITED 0

Sematext Docker Agent running on all Docker UCP nodes

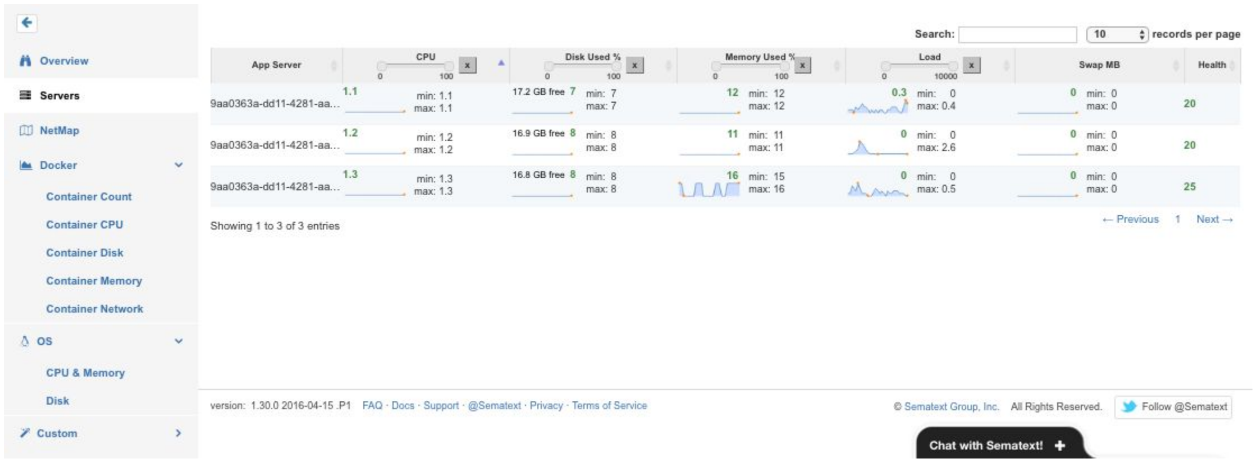
8.2.3 Voila! Your Docker Monitoring & Logging Works

After about a minute you should see the performance metrics of all Docker UCP nodes and deployed containers in your SPM Apps...



Key Docker metrics in SPM

The Server view should show all Docker UCP nodes:



Search: 10 records per page

App Server	CPU	Disk Used %	Memory Used %	Load	Swap MB	Health
9aa0363a-dd11-4281-aa…	1.1 min: 1.1 max: 1.1	17.2 GB free 7 min: 7 max: 7	12 min: 12 max: 12	0.3 min: 0 max: 0.4	0 min: 0 max: 0	20
9aa0363a-dd11-4281-aa…	1.2 min: 1.2 max: 1.2	16.9 GB free 8 min: 8 max: 8	11 min: 11 max: 11	0 min: 0 max: 2.6	0 min: 0 max: 0	20
9aa0363a-dd11-4281-aa…	1.3 min: 1.3 max: 1.3	16.8 GB free 8 min: 8 max: 8	16 min: 15 max: 16	0 min: 0 max: 0.5	0 min: 0 max: 0	25

Showing 1 to 3 of 3 entries

← Previous 1 Next →

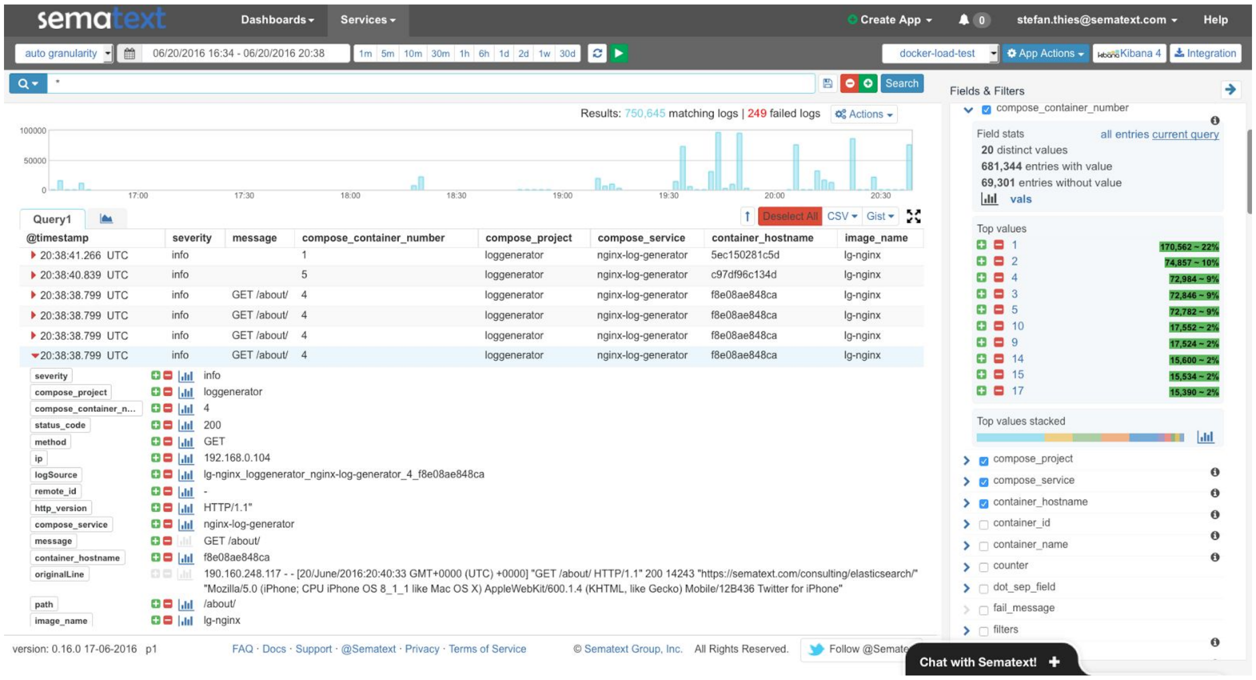
version: 1.30.0 2016-04-15 .P1 FAQ · Docs · Support · @Sematext · Privacy · Terms of Service

© Sematext Group, Inc. All Rights Reserved. Follow @Sematext

Chat with Sematext!

Docker UCP nodes

...and you will see all your Docker logs in Logsene.
This screenshot shows parsed web server logs tagged with Docker compose metadata, such as compose project, server, container number:



Results: 750,645 matching logs | 249 failed logs

@timestamp	severity	message	compose_container_number	compose_project	compose_service	container_hostname	image_name
20:38:41.266 UTC	info		1	loggenerator	nginx-log-generator	5ec150281c5d	lg-nginx
20:38:40.839 UTC	info		5	loggenerator	nginx-log-generator	c97df96c134d	lg-nginx
20:38:38.799 UTC	info	GET /about/	4	loggenerator	nginx-log-generator	f8e08ae848ca	lg-nginx
20:38:38.799 UTC	info	GET /about/	4	loggenerator	nginx-log-generator	f8e08ae848ca	lg-nginx
20:38:38.799 UTC	info	GET /about/	4	loggenerator	nginx-log-generator	f8e08ae848ca	lg-nginx
20:38:38.799 UTC	info	GET /about/	4	loggenerator	nginx-log-generator	f8e08ae848ca	lg-nginx

severity	info
compose_project	loggenerator
compose_container_n...	4
status_code	200
method	GET
ip	192.168.0.104
logSource	lg-nginx_loggenerator_nginx-log-generator_4_f8e08ae848ca
remote_id	-
http_version	HTTP/1.1"
compose_service	nginx-log-generator
message	GET /about/
container_hostname	f8e08ae848ca
originalLine	190.160.248.117 - - [20/June/2016:20:40:33 GMT+0000 (UTC) +0000] "GET /about/ HTTP/1.1" 200 14243 "https://sematext.com/consulting/elasticsearch/" "Mozilla/5.0 (iPhone; CPU iPhone OS 8_1_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Mobile/12B436 Twitter for iPhone"
path	/about/
image_name	lg-nginx

Fields & Filters

compose_container_number

Field stats — all entries current query
20 distinct values
681,344 entries with value
69,301 entries without value
vals

Top values

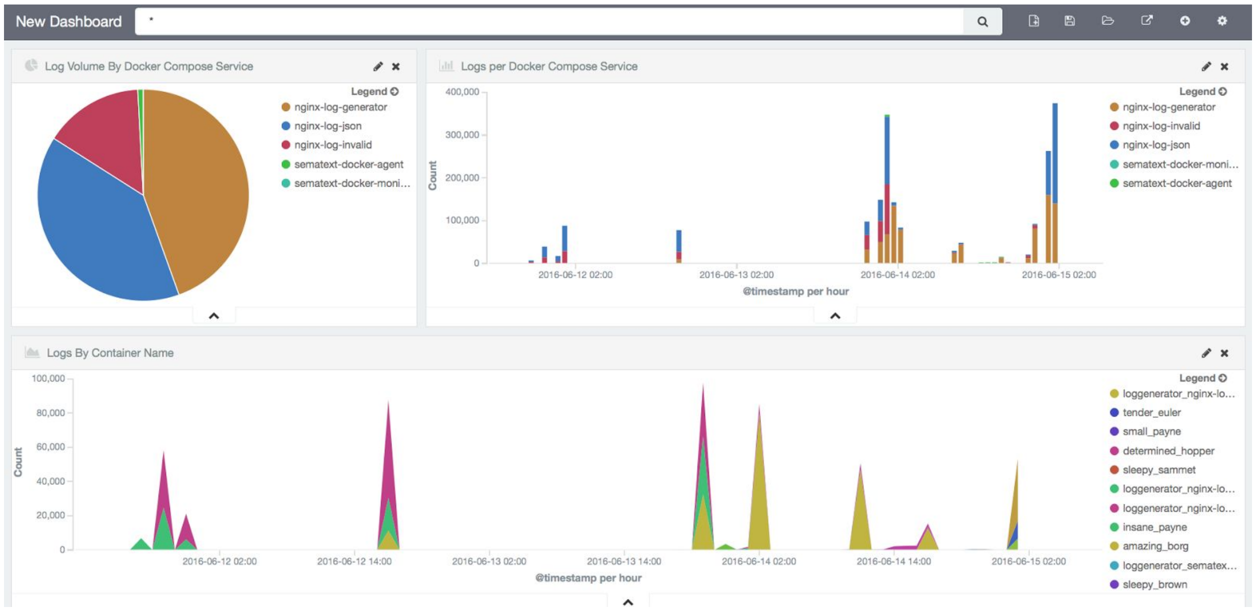
1	170,562 ~ 22%
2	74,857 ~ 10%
4	72,984 ~ 9%
3	72,846 ~ 9%
5	72,782 ~ 9%
10	17,552 ~ 2%
9	17,524 ~ 2%
14	15,600 ~ 2%
15	15,534 ~ 2%
17	15,390 ~ 2%

Top values stacked

- ✓ compose_project
- ✓ compose_service
- ✓ container_hostname
- container_id
- container_name
- counter
- dot_sep_field
- fail_message
- filters

version: 0.16.0 17-06-2016 p1 FAQ · Docs · Support · @Sematext · Privacy · Terms of Service © Sematext Group, Inc. All Rights Reserved. Follow @Semate

Logsene native UI shows automatically parsed log structure



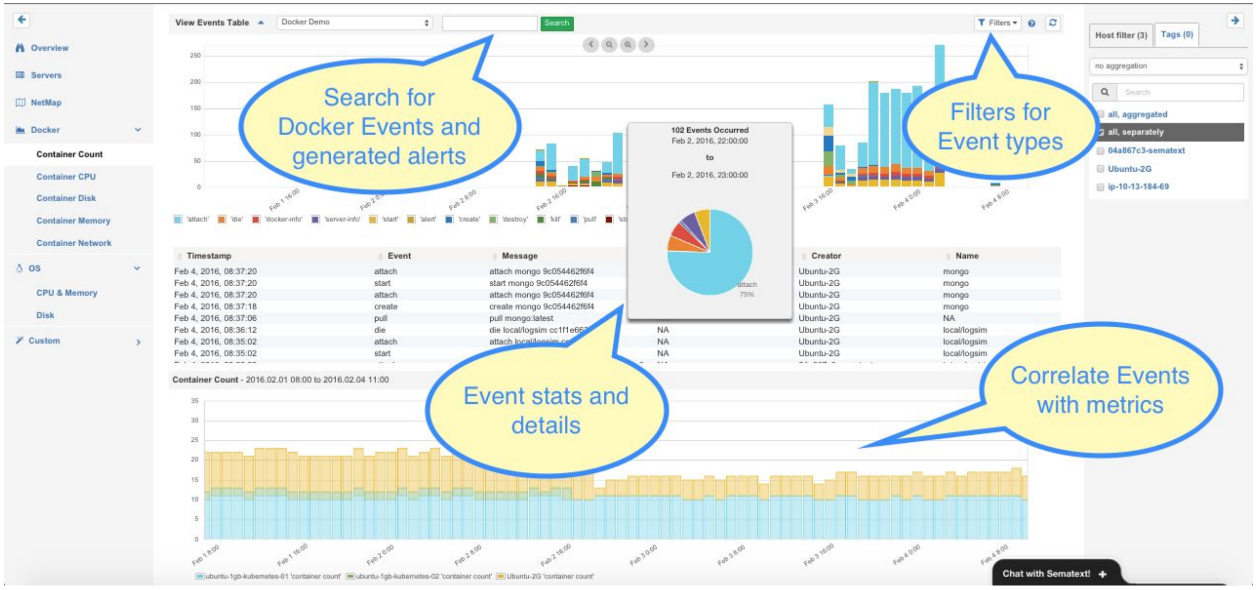
Logsene integrated Kibana dashboards to slice and dice Docker log data

8.2.4 Connect Metrics and Logs for Faster Troubleshooting

For faster troubleshooting you can connect apps (SPM app with SPM app, Logsene with Logsene, or SPM with Logsene). Connecting apps lets you more easily correlate metrics and logs and thus more quickly find the root cause of performance and other issues.

1. Open "Logs" section in SPM, click to the link icon in the top right corner and choose your Logsene Application.
2. Press the connect button to save the setting

Once SPM app with your Docker metrics and events and your Logsene app with your Docker logs are connected, you can see all these data in a single pane of glass:



Timestamp	Event	Message		Creator	Name
Feb 4, 2016, 08:37:20	attach	attach mongo 9c054462f6f4		Ubuntu-2G	mongo
Feb 4, 2016, 08:37:20	start	start mongo 9c054462f6f4		Ubuntu-2G	mongo
Feb 4, 2016, 08:37:20	attach	attach mongo 9c054462f6f4		Ubuntu-2G	mongo
Feb 4, 2016, 08:37:18	create	create mongo 9c054462f6f4		Ubuntu-2G	mongo
Feb 4, 2016, 08:37:06	pull	pull mongo:latest		Ubuntu-2G	NA
Feb 4, 2016, 08:36:12	die	die local/logsim cc1f1e66	NA	Ubuntu-2G	local/logsim
Feb 4, 2016, 08:35:02	attach	attach local/logsim	NA	Ubuntu-2G	local/logsim
Feb 4, 2016, 08:35:02	start		NA	Ubuntu-2G	local/logsim

Container Count - 2016.02.01 08:00 to 2016.02.04 11:00

Docker performance metrics and logs in a single pane of glass

9 Configuring Sematext Docker Agent

9.1 Connection to SPM and Logsene

SPM and Logsene are available in the Cloud (SaaS) or On Premises. Depending on this setup Sematext Docker Agent needs to be configured to ship data to the appropriate SPM and Logsene data receiver endpoints.

9.1.1 SPM and Logsene in the Cloud (SaaS)

The default configuration of Sematext Docker Agent is to connect via TLS (HTTPS) to the SaaS provided by Sematext using the following API endpoints:

- SPM Receiver for metrics collection: https://spm-receiver.sematext.com:443
- Event Receiver for (Docker) events: https://event-receiver.sematext.com:443
- Logsene Receiver for logs: https://logsene-receiver.sematext.com:443

If you are using SPM or Logsene SaaS there is no configuration required to use the above default settings.

To reach the above mentioned Receiver services through firewalls, it is possible to configure proxy server settings as URL using the environment variable `HTTPS_PROXY`.

9.1.2 SPM and Logsene On Premises

If SPM and Logsene (not just the agents, but the whole SPM and Logsene solution) are deployed in the local network the servers will have local IP addresses or DNS names. Sematext Docker Agent lets you change the Receiver addresses for SPM and Logsene using environment variables:

- `SPM_RECEIVER_URL` - URL to your SPM Receiver
- `EVENTS_RECEIVER_URL` - URL to your Events Receiver
- `LOGSENE_RECEIVER_URL` - URL to your Logsene Receiver

Detailed installation instructions are included in the SPM and Logsene On Premises package - email sales@sematext.com or call +1 (347) 480 1610 for a free evaluation copy.

9.2 Log Handling Options

9.2.1 Blacklisting and Whitelisting Logs

Not all logs might be of interest, so sooner or later you will have the need to blacklist some log types. This is one of the reasons why Sematext Docker Agent automatically adds the following tags to all logs:

- Container ID
- Container Name
- Image Name
- Docker Compose Project Name
- Docker Compose Service Name
- Docker Compose Container Number

Using this "log metadata" you can whitelist or blacklist log outputs by image or container names. The relevant environment variables are:

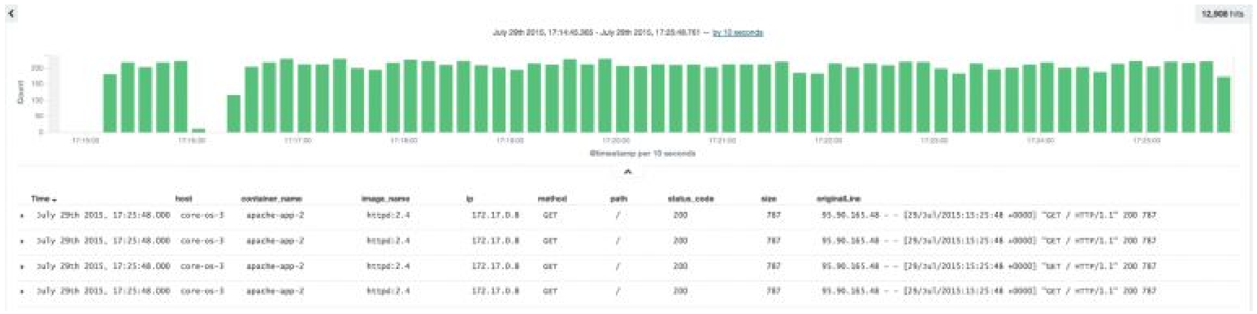
- MATCH_BY_NAME — a regular expression to whitelist container names
- MATCH_BY_IMAGE — a regular expression to whitelist image names
- SKIP_BY_NAME — a regular expression to blacklist container names
- SKIP_BY_IMAGE — a regular expression to blacklist image names

9.2.2 Automatic Parser for Container Logs

In Docker logs are console output streams from containers. They might be a mix of plain text messages from start scripts and structured logs from applications. The problem is obvious – you can't just take a stream of log events all mixed up and treat them like a blob. You need to be able to tell which log event belongs to what container, what app, parse it correctly in order to structure it so you can later derive more insight and operational intelligence from logs, etc.

Sematext Docker Agent analyzes the event format, parses out data, and turns logs into structured JSON. This is important, because the value of logs increases when you structure them — you can then slice and dice them and gain a lot more insight about how your containers, servers, and applications operate.

Traditionally it was necessary to use log shippers like Logstash, Fluentd or rsyslog to parse log messages. The problem is that such setups are typically deployed in a very static fashion and configured for each input source. That does not work well in the hyper-dynamic world of containers! We have seen people struggling with the syslog drivers, parsers configurations, log routing, and more! With its integrated automatic format detection Sematext Docker Agent eliminates this struggle — and the waste of resources — both computing and human time that goes into dealing with such things! This integration has a low footprint, doesn't need retransmissions of logs to external services, and it detects log types for the most popular applications and generic JSON and line-oriented log formats out of the box!


12,908 hits
July 29th 2015, 17:14:45.365 - July 29th 2015, 17:25:48.781 — by 10 seconds
Count
@timestamp per 10 seconds

Time	host	container_name	image_name	ip	method	path	status_code	size	originalLine
July 29th 2015, 17:25:48.000	core-os-3	apache-app-2	httpd:2.4	172.17.0.8	GET	/	200	787	95.90.165.48 - - [29/Jul/2015:15:25:48 +0000] "GET / HTTP/1.1" 200 787
July 29th 2015, 17:25:48.000	core-os-3	apache-app-2	httpd:2.4	172.17.0.8	GET	/	200	787	95.90.165.48 - - [29/Jul/2015:15:25:48 +0000] "GET / HTTP/1.1" 200 787
July 29th 2015, 17:25:48.000	core-os-3	apache-app-2	httpd:2.4	172.17.0.8	GET	/	200	787	95.90.165.48 - - [29/Jul/2015:15:25:48 +0000] "GET / HTTP/1.1" 200 787
July 29th 2015, 17:25:48.000	core-os-3	apache-app-2	httpd:2.4	172.17.0.8	GET	/	200	787	95.90.165.48 - - [29/Jul/2015:15:25:48 +0000] "GET / HTTP/1.1" 200 787

Example: Apache Access Log fields generated by Sematext Docker Agent

For example, Sematext Docker Agent can parse logs from official images like:

- Nginx, Apache, Redis, MongoDB, MySQL
- Elasticsearch, Solr, Kafka, Zookeeper
- Hadoop, HBase, Cassandra
- Any JSON output with special support for Logstash or Bunyan format
- Plain text messages with or without timestamps in various formats
- Various Linux and Mac OSX system logs

In addition, you can define your own patterns for any log format you need to be able to parse and structure. There are three options to pass individual log parser patterns [3]:

- Configuration file in a mounted volume:
  `-v PATH_TO_YOUR_FILE:/etc/logagent/patterns.yml`
- Content of the configuration file in an environment variable
  `-e LOGAGENT_PATTERNS="$(cat patterns.yml)"`
- Download pattern definitions via HTTP
  `-e PATTERNS_URL=http://yourserver/patterns.yml`

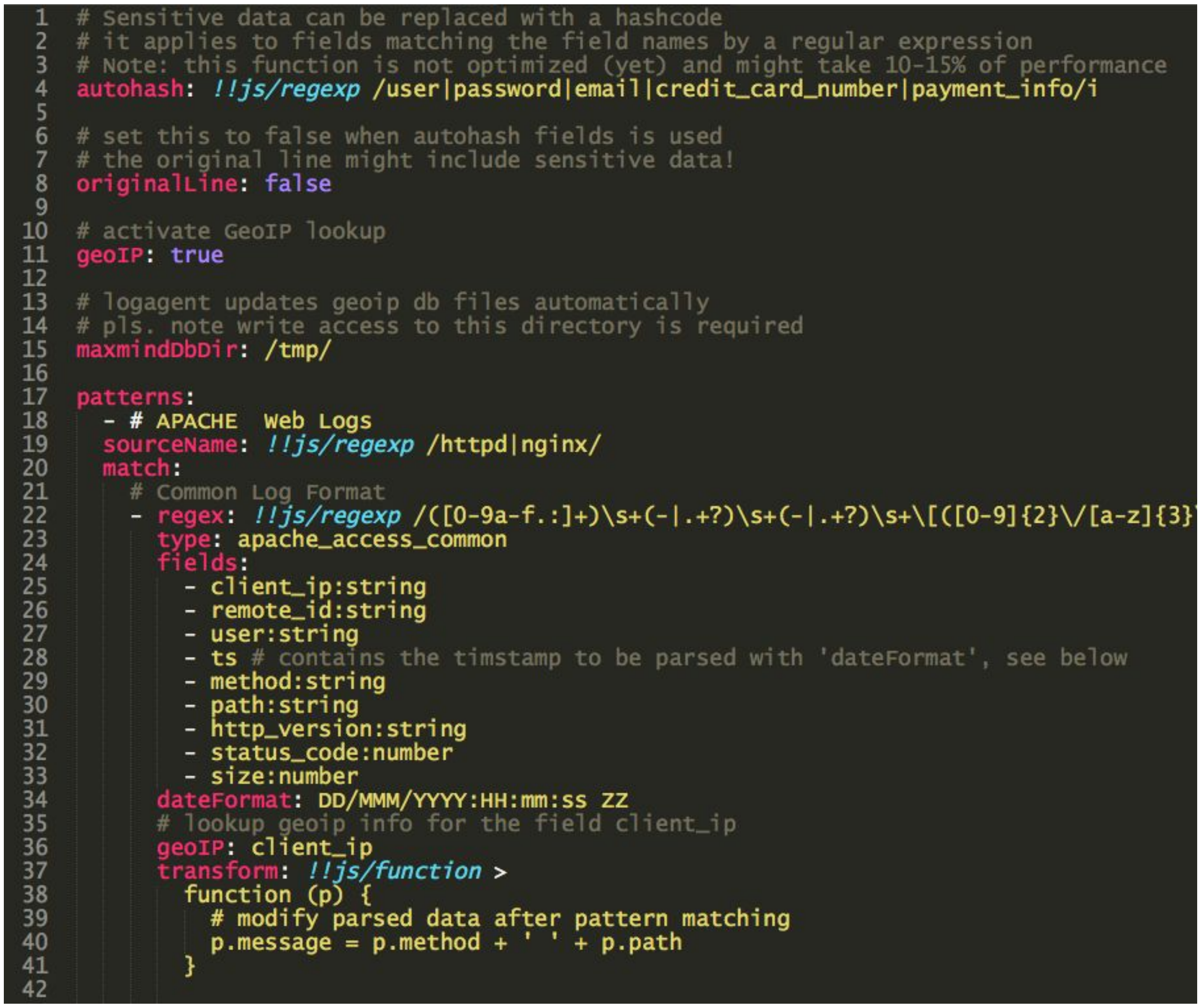
The file format for the patterns.yml file is based on JS-YAML, in short:
- indicates an array element
`!js/regexp` - indicates a JavaScript regular expression
`!!js/function >` - indicates a JavaScript function

The file has the following properties:

- `patterns`: list of patterns, each pattern starts with "-"
- `match`: group of patterns for a specific log source (image / container)
- `regex`: JS regular expression
- `fields`: field list of extracted match groups from the regex
- `type`: type used in Logsene (Elasticsearch Mapping)

- `dateFormat`: format of the special fields 'ts', if the date format matches, a new field @timestamp is generated
- `transform`: JS function to manipulate the result of regex and date parsing

The following example shows pattern definitions for web server logs, which is one of the patterns available by default:



```
# Sensitive data can be replaced with a hashcode
# it applies to fields matching the field names by a regular expression
# Note: this function is not optimized (yet) and might take 10-15% of performance
autohash: !!js/regexp /user|password|email|credit_card_number|payment_info/i

# set this to false when autohash fields is used
# the original line might include sensitive data!
originalLine: false

# activate GeoIP lookup
geoIP: true

# logagent updates geoip db files automatically
# pls. note write access to this directory is required
maxmindDbDir: /tmp/

patterns:
  - # APACHE  Web Logs
  sourceName: !!js/regexp /httpd|nginx/
  match:
    # Common Log Format
    - regex: !!js/regexp /([0-9a-f.:]+)\s+(-|.+?)\s+(-|.+?)\s+\[([0-9]{2}\/[a-z]{3}
      type: apache_access_common
      fields:
        - client_ip:string
        - remote_id:string
        - user:string
        - ts # contains the timstamp to be parsed with 'dateFormat', see below
        - method:string
        - path:string
        - http_version:string
        - status_code:number
        - size:number
      dateFormat: DD/MMM/YYYY:HH:mm:ss ZZ
      # lookup geoip info for the field client_ip
      geoIP: client_ip
      transform: !!js/function >
        function (p) {
          # modify parsed data after pattern matching
          p.message = p.method + ' ' + p.path
        }

```

Example from https://sematext.github.io/logagent-js/parser/

This example shows a few very interesting features:

- **Masking sensitive data** with "autohash" property, listing fields to be replaced with a hash code. See section 9.2.4.
- **Automatic Geo-IP lookups** including automatic updates for Maxmind Geo-IP lite database. See section 9.2.5.
- **Post-processing of parsed logs** with JavaScript functions. See section 9.2.6.

The component for detecting and parsing log messages — logagent-js — is open source and contributions for even more log formats are welcome.

9.2.3 Log Routing with Docker Labels

Storing all logs in a single searchable index represented by a Logsene App Token might be very convenient for quick troubleshooting. However, there are common scenarios when you would want to have different logs indexed in separate Logsene Apps, such as:

- **Limit access to different logs to different teams or team members.** In Logsene the access permissions can be granted on a per Logsene App basis. This means you can have very fine control over who has the rights to see which logs.
- **Analytics for logs.** All data used for structured analytics like web server logs, sensor data or KPI's are much easier to process when stored in their own Logsene Apps without the "noise" from other applications with different log structures. For example, you probably wouldn't want to mix logs from your custom app running in a container with Nginx and MySQL logs, so you might create separate Logsene Apps for each of them.

Sematext Docker Agent can route logs from different containers to specific Logsene Apps. It builds the log routing table by reading the Logsene App Token from containers' Docker Labels. This approach is much more dynamic than maintaining a large configuration file that maps container IDs to Logsene App Tokens.

Example:
To route logs from Nginx to a dedicated Logsene App and attach a Docker Label to the Nginx containers:

```
docker run --label LOGSENE_TOKEN=YOUR-LOGSENE-TOKEN-HERE nginx
```

Sematext Docker Agent will recognize the Label during the auto-discovery of any new containers and will use the corresponding Logsene App Token to ship logs to that Logsene App. The end result is that all logs from all containers labeled "nginx" will get aggregated in the same Logsene App.

9.2.4 Masking Sensitive Data in Logs

Logs can contain sensitive data — credit card numbers, social security numbers, birthdays, and so on. Sematext Docker Agent lets you mask such sensitive data before shipping it, thus hiding it from overly curious 3rd parties (network proxies, storage providers, etc.).

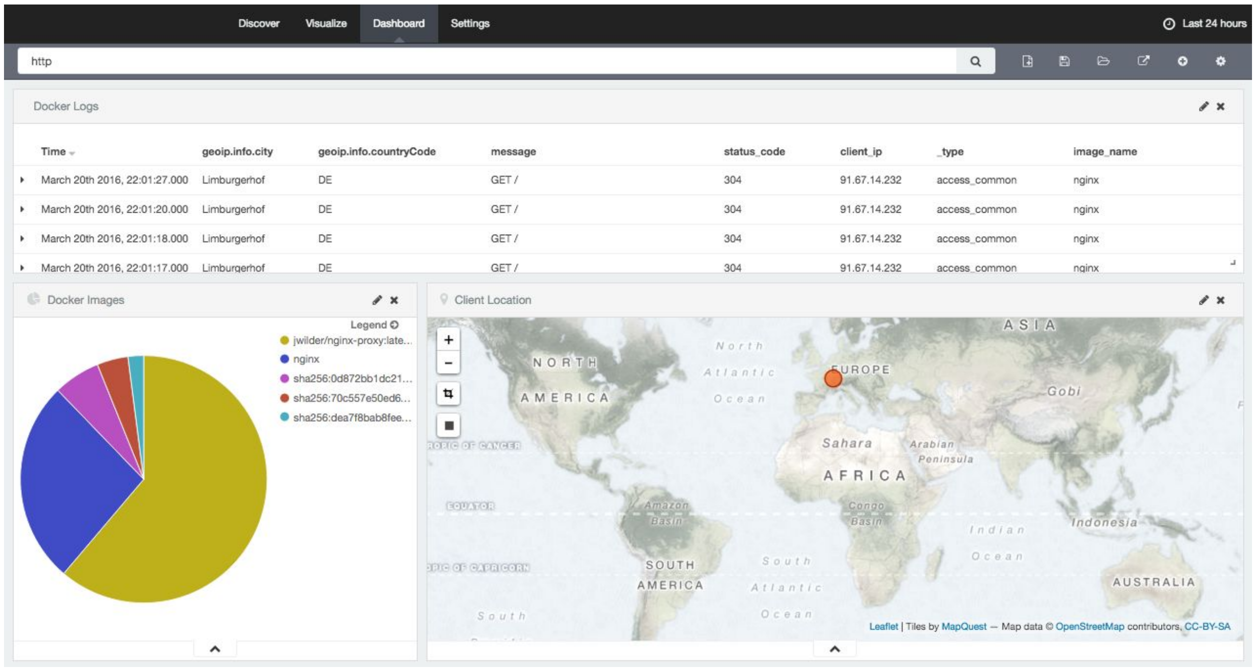
Replacing content with hash codes has the advantage that the content is not “readable” for 3rd parties, but knowing the original value lets you calculate the hash code for later search. This makes it possible to search for logs with a specific hashed field content.
Consider a scenario with a client phone number that was stored as SHA hash code in a masked field in Logsene. During an investigation of a problem related to that phone number you would be able to calculate the SHA hash code of the client phone number and then search for that hash code in your logs in Logsene to find all related logs — *without exposing the actual phone number to 3rd parties (including Sematext)*.

To use this, in the custom pattern definition list all log fields that need to be masked. The Sematext Docker agent will then automatically mask all such fields. For example, we could use this settings in patterns.yml:

```
# Sensitive data can be replaced with a hashcode.
# It applies to fields matching the field names by a regular expression
autohash: !!js/regexp /user|password|email|credit_card_number|payment_info/i
#
# set the option to include original log to ‘false’
# when autohash is used.
# The original log line might include sensitive data!
originalLine: false
```

9.2.5 Automatic Geo-IP Enrichment for Container Logs

Getting logs from Docker Containers collected, shipped and parsed out of the box is already a big time saver, but some application logs need additional enrichment with information from other data sources. A common use case is to enrich web server logs (or really any logs with IP addresses) with geographical information derived from those IP addresses.
Sematext Docker Agent supports Geo-IP enrichment, simply activated by the the environment variable `GEOIP_ENABLED=true`.
It uses Maxmind Geo-IP lite database, which is updated automatically in the running container! There is no need to stop the container, mount new volumes with the Geo-IP database, etc.



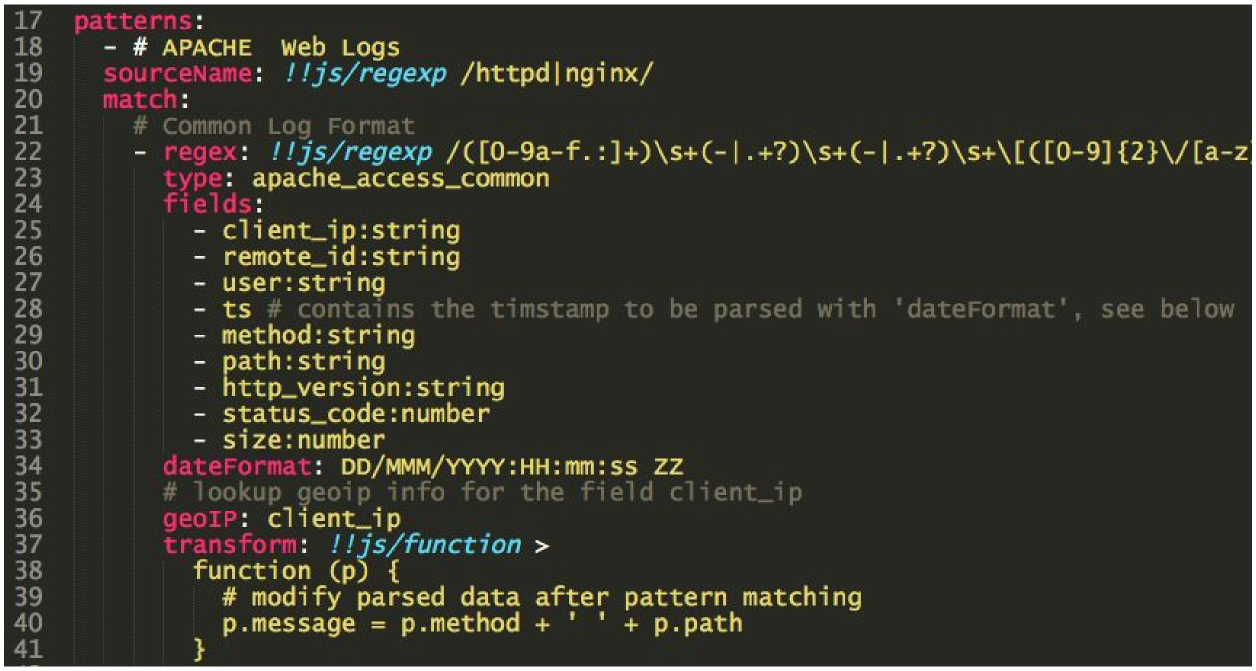
Time	geoip.info.city	geoip.info.countryCode	message	status_code	client_ip	_type	image_name
March 20th 2016, 22:01:27.000	Limburgerhof	DE	GET /	304	91.67.14.232	access_common	nginx
March 20th 2016, 22:01:20.000	Limburgerhof	DE	GET /	304	91.67.14.232	access_common	nginx
March 20th 2016, 22:01:18.000	Limburgerhof	DE	GET /	304	91.67.14.232	access_common	nginx
March 20th 2016, 22:01:17.000	Limburgerhof	DE	GET /	304	91.67.14.232	access_common	nginx

Visualization of Geo-IP data in Logsene / Kibana

9.2.6 Post-processing Parsed Logs with JavaScript Functions

Log data can be complex! Simple extraction of text might not be sufficient for advanced analytics — you might need to perform simple calculations based on extracted fields. Or, in another case, you like to transform the most relevant part of a large log entry into a human readable message. Sematext Docker Agent lets you apply post-processing to the output of the log parser to further restructure logs before they are shipped and indexed.

In custom pattern definitions (`patterns.yml` for the logagent-js parser), post processing hooks can be defined in JavaScript (Node.js runtime). Each pattern definition has an optional "transform" property for such JavaScript functions. In the following example we simply overwrite the "message field" in a web server log with the HTTP method and the path to generate a short but readable content in the "message" field:

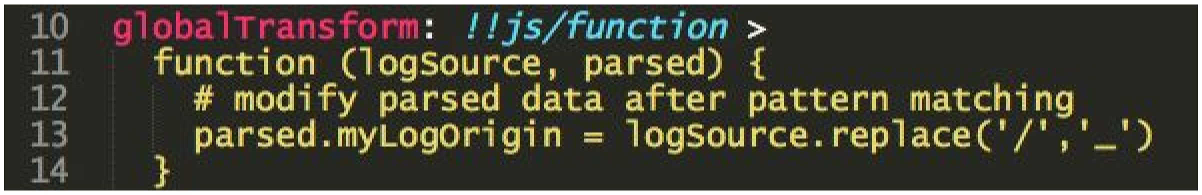


```
patterns:
  - # APACHE  Web Logs
  sourceName: !!js/regexp /httpd|nginx/
  match:
    # Common Log Format
    - regex: !!js/regexp /([0-9a-f.:]+)\s+(-|.+?)\s+(-|.+?)\s+\[([0-9]{2}\/[a-z
      type: apache_access_common
      fields:
        - client_ip:string
        - remote_id:string
        - user:string
        - ts # contains the timstamp to be parsed with 'dateFormat', see below
        - method:string
        - path:string
        - http_version:string
        - status_code:number
        - size:number
      dateFormat: DD/MMM/YYYY:HH:mm:ss ZZ
      # lookup geoip info for the field client_ip
      geoIP: client_ip
      transform: !!js/function >
        function (p) {
          # modify parsed data after pattern matching
          p.message = p.method + ' ' + p.path
        }
```

Example from https://sematext.github.io/logagent-js/parser/

If you want to apply a function to all logs, and not just to specific patterns, Sematext Docker Agent supports this as well. To do that use the JavaScript function called "`globalTransform`" with two parameters: the name of the log source (image_name/container_name/id) and the parsed object to be modified.

The "*globalTransform*" function is a top level property in the `patterns.yml` file and not bound to any specific subsection for patterns.



```
globalTransform: !!js/function >
  function (logSource, parsed) {
    # modify parsed data after pattern matching
    parsed.myLogOrigin = logSource.replace('/','_')
  }
```

10 Summary

In this Reference Architecture, we set up monitoring and log collection for a Docker EE cluster, which you can use with Docker Datacenter deployment of any size. We learned about Docker Key Metrics and how to interpret them in various scenarios. Sematext Docker Agent was deployed via Docker UCP Remote API endpoint and configured via docker-compose to collect metrics, logs, and events locally on each node. In addition, we described several useful configuration options for log processing to improve log analytics for better operational insights.

11 References

[1] SPM Performance Monitoring - http://sematext.com/spm

[2] Logsene - http://sematext.com/logsene

[3] Logagent-js - https://sematext.github.io/logagent-js

[4] Sematext Docker Agent - https://github.com/sematext/sematext-agent-docker

[5] Sematext Group, Inc.
540 President St. 3rd Floor
Brooklyn, NY 11215
USA
E-Mail: sales@sematext.com
Phone: +1 (347) 480 1610

Appendix A - Sematext Docker Agent Configuration Options

Parameter / Environment variable	**Description**
Required Parameters	
SPM_TOKEN	SPM Application Token, enables metric and event collection
LOGSENE_TOKEN	Logsene Application Token enables logging to Logsene, see logging specific parameters for filter options and Log Routing section to route logs from different containers to separate Logsene applications
-v /var/run/docker.sock:/var/run/docker.sock	Path to the docker socket (optional, if dockerd provides TCP on 2375, see also DOCKER_PORT and DOCKER_HOST parameter)
TCP and TLS connection If the UNIX socket is not available Sematext Agent assumes the Container Gateway Address (autodetect) and port 2375 as default (no TLS) - this needs no configuration. If Docker Daemon TCP settings are different you have to configure the TCP settings. The TCP settings can be modified with the following parameters. Please note the following parameters are compatible with the variables set in the “docker-machine env” command.	
DOCKER_HOST	e.g. tcp://ip-reachable-from-container:2375/ - default value 'unix:///var/run/docker.sock'. When the UNIX socket is not available the agent tries to connect to tcp://gateway:2375. In case a TCP socket is used there is no need to mount the Docker UNIX socket as volume
DOCKER_PORT	Sematext Docker Agent will use its gateway address (autodetect) with the given DOCKER_PORT

DOCKER_TLS_VERIFY	0 or 1
DOCKER_CERT_PATH	Path to your certificate files. Mount the path to the certificates with `-v $DOCKER_CERT_PATH:$DOCKER_CERT_PATH`
Optional Parameters	
--privileged	The parameter might be helpful when Sematext Agent cannot start because of limited permission to connect and write to the Docker socket /var/run/docker.sock. The privileged mode is a potential security risk, so we recommend to enable the appropriate security. Please read about Docker security: https://docs.docker.com/engine/security/security/
HOSTNAME_LOOKUP_URL	On Amazon ECS, a metadata query must be used to get the instance hostname (e.g. "169.254.169.254/latest/meta-data/local-hostname")
HTTPS_PROXY	URL for a proxy server (behind firewalls)
LOGSENE_RECEIVER_URL	URL for bulk inserts into Logsene. Required for Logsene On Premises only.
SPM_RECEIVER_URL	URL for bulk inserts into SPM. Required for SPM On Premises only.
EVENTS_RECEIVER_URL	URL for SPM events receiver. Required for SPM On Premises only.
Docker Logs Parameters	
Whitelist containers	
MATCH_BY_NAME	Regular expression to white list container names
MATCH_BY_IMAGE	Regular expression to white list image names
Blacklist containers	

SKIP_BY_NAME	Regular expression to black list container names
SKIP_BY_IMAGE	Regular expression to black list image names for logging
-v /yourpatterns/patterns.yml:/etc/logagent/patterns.yml	Pass custom patterns.yml for log parsing, see logagent-js
-v /tmp:/logsene-log-buffer	Directory to store logs, in case of a network or service outage. Docker Agent deletes these files after a successful transmission.
GEOIP_ENABLED	`true` enables Geo-IP lookups in the log parser; default value: `false`
MAXMIND_DB_DIR	Directory for the Geo-IP lite database, must end with /. Storing the DB in a volume could save downloads for updates after restarts. Using /tmp/ (ramdisk) could speed up Geo-IP lookups (consumes add. ~30 MB main memory).

For the most up to date list of Sematext Docker Agent options see https://github.com/sematext/sematext-agent-docker.

Sematext Group Inc.
sematext.com
sematext.com/blog
twitter.com/sematext
+1-347-480-1610

540 President Street, 3rd Floor
Brooklyn, NY 11215 USA

Sematext Group Inc.

Single Pane of Glass for Monitoring, Logging & Analytics
Search and Big Data Consulting
Production Support and Training for Solr and Elasticsearch