title: Docker Monitoring Integration description: Container performance monitoring - metrics, event, log collection and parsing for Docker

The Sematext Integration for Docker uses Sematext Agent and the open-source Logagent for the collection of container logs.

We at Sematext aim to save you time and effort by giving you a strong starting point for monitoring Docker. You will **not** have to:

- figure out which metrics to collect and which ones to ignore
- give metrics meaningful labels
- hunt for metric descriptions in the docs so that you know what each of them actually shows
- build charts to group metrics that you really want on the same charts, not N separate charts
- figure out, for each metric, which aggregation to use (min? max? avg? something else?)
- build dashboards to combine charts with metrics you typically want to see together
- set up basic alert rules

We set it up for you, out-of-the-box!

Docker Monitoring with Sematext Agent

Sematext Agent collects various metrics about hosts and ships that to Sematext Cloud.

First create a Docker Monitoring App in Sematext Cloud.

Docker

You install the Agent simply by running one Docker command. This will start the Agent as a Docker container on your host.

```
docker run -d  --restart always --privileged -P --name st-agent \
  -v /:/hostfs:ro \
  -v /sys/kernel/debug:/sys/kernel/debug \
  -v /var/run/:/var/run/ \
  -v /proc:/host/proc:ro \
  -v /etc:/host/etc:ro \
  -v /sys:/host/sys:ro \
  -v /usr/lib:/host/usr/lib:ro \
  -e CONTAINER_TOKEN=84fbc37e-a0fb-418c-9bcb-ea7c763dd9ac \
  -e INFRA_TOKEN=6377be8e-8441-46de-85dc-11ee3646c3de \
  -e REGION=US \
  -e JOURNAL_DIR=/var/run/st-agent \
  -e LOGGING_WRITE_EVENTS=false \
  -e LOGGING_REQUEST_TRACKING=false \
```

```
  -e LOGGING_LEVEL=info \
  -e NODE_NAME=`hostname` \
  -e CONTAINER_SKIP_BY_IMAGE=sematext \
  sematext/agent:latest
```

Docker Compose

If you prefer adding the Agent in a `docker-compose` configuration, here's how you do it.

```
# docker-compose.yml
version: '3'
services:
  sematext-agent:
    image: 'sematext/agent:latest'
    environment:
      - affinity:container!=*sematext-agent*
      - CONTAINER_TOKEN=84fbc37e-a0fb-418c-9bcb-ea7c763dd9ac
      - INFRA_TOKEN=6377be8e-8441-46de-85dc-11ee3646c3de
      - REGION=US
      - JOURNAL_DIR=/var/run/st-agent
      - LOGGING_WRITE_EVENTS=false
      - LOGGING_REQUEST_TRACKING=false
      - LOGGING_LEVEL=info
      - NODE_NAME=$HOSTNAME
      - CONTAINER_SKIP_BY_IMAGE=sematext
    cap_add:
      - SYS_ADMIN
    restart: always
    volumes:
      - '/:/hostfs:ro'
      - '/var/run/:/var/run/'
      - '/sys/kernel/debug:/sys/kernel/debug'
      - '/proc:/host/proc:ro'
      - '/etc:/host/etc:ro'
      - '/sys:/host/sys:ro'
      - '/usr/lib:/host/usr/lib:ro'
```

Then you can run one command to start the Agent.

```
docker-compose up -d
```

Docker Swarm

If you're running a Docker Swarm cluster, it's just as easy to run a Docker Swarm service.

```
docker service create --mode global --name st-agent \
  --restart-condition any \
  --mount type=bind,src=/,dst=/hostfs,readonly \
  --mount type=bind,src=/var/run,dst=/var/run/ \
  --mount type=bind,src=/usr/lib,dst=/host/usr/lib \
  --mount type=bind,src=/sys/kernel/debug,dst=/sys/kernel/debug \
  --mount type=bind,src=/proc,dst=/host/proc,readonly \
  --mount type=bind,src=/etc,dst=/host/etc,readonly \
  --mount type=bind,src=/sys,dst=/host/sys,readonly \
  -e NODE_NAME={{.Node.Hostname}} \
  -e CONTAINER_TOKEN=84fbc37e-a0fb-418c-9bcb-ea7c763dd9ac \
  -e INFRA_TOKEN=6377be8e-8441-46de-85dc-11ee3646c3de \
  -e REGION=US \
  -e JOURNAL_DIR=/var/run/st-agent \
  -e LOGGING_REQUEST_TRACKING=false \
  -e LOGGING_WRITE_EVENTS=false \
  -e LOGGING_LEVEL=info \
  -e PKG_ENABLED=false \
  sematext/agent:latest
```

If you like using `docker stack`, editing the `docker-compose.yml` from above slightly you'll have a working configuration.

```yaml
version: "3"
services:
  agent:
    image: sematext/agent:latest
    deploy:
      mode: global
      labels: [APP=AGENT]
      restart_policy:
        condition: any
        delay: 1s
    cap_add:
      - SYS_ADMIN
    restart: always
    environment:
      - affinity:container!=*sematext-agent*
      - CONTAINER_TOKEN=84fbc37e-a0fb-418c-9bcb-ea7c763dd9ac
      - INFRA_TOKEN=6377be8e-8441-46de-85dc-11ee3646c3de
      - JOURNAL_DIR=/var/run/st-agent
      - LOGGING_WRITE_EVENTS=false
      - LOGGING_REQUEST_TRACKING=false
      - LOGGING_LEVEL=info
      - NODE_NAME=$HOSTNAME
      - CONTAINER_SKIP_BY_IMAGE=sematext
      - REGION=US
```

```
    - PKG_ENABLED=false
  volumes:
    - "/:/hostfs:ro"
    - "/var/run:/var/run/"
    - "/usr/lib:/host/usr/lib"
    - "/sys/kernel/debug:/sys/kernel/debug"
    - "/proc:/host/proc:ro"
    - "/etc:/host/etc:ro"
    - "/sys:/host/sys:ro"
```

Then you run:

```
docker stack deploy -c docker-compose.yml <name>
```

The Sematext Agent will start collecting dozens of key metrics right away, and start showing you the performance and health of your Docker containers immediately.

Collected Docker Metrics

The Sematext Agent will collect the following container and host metrics.

Host Metrics

- CPU
- memory
- disk
- network
- processes
- containers
- orchestrator platforms

eBPF Support

To gain deep **insight into the Linux kernel**, Sematext Agent relies on **eBPF** to implant **instrumentation points**, which means to **attach eBPF programs to kprobes** on kernel functions. This ensures a very efficient and powerful system exploration approach with better network tracing and negligible overhead.

Service Auto-Discovery

Sematext Agent can **auto-discover services** deployed on physical/virtual hosts and containers. It also collects data about your infrastructure to provide you with infrastructure inventory reports. It collects events from different sources such as OOM notifications, container or Kubernetes events.

Container Metrics

- Container runtime agnostic discovery and monitoring
  - Containers are discovered from cgroupfs hierarchies
  - Supports Docker and Rkt container engines
- Container metrics fetched directly from cgroupfs
  - CPU usage
  - Disk space usage and IO stats
  - Memory usage, memory limits, and memory fail counters
  - Network IO stats
- Collection of host inventory information
  - Host kernel version/system information
  - Information about installed software packages
- Collection of container metadata
  - Container name
  - Image name
  - Container networks
  - Container volumes
  - Container environment
  - Container labels including relevant information about orchestration
  - Kubernetes metadata such as Pod name, UUID, Namespace
  - Docker Swarm metadata such as Service name, Swarm Task etc.
- Collection of container events
- Docker events such as start/stop/die/volume mount, etc.
- Kubernetes events such as Pod status changes deployed, destroyed etc.
- Tracking deployment status and Pod restarts over time

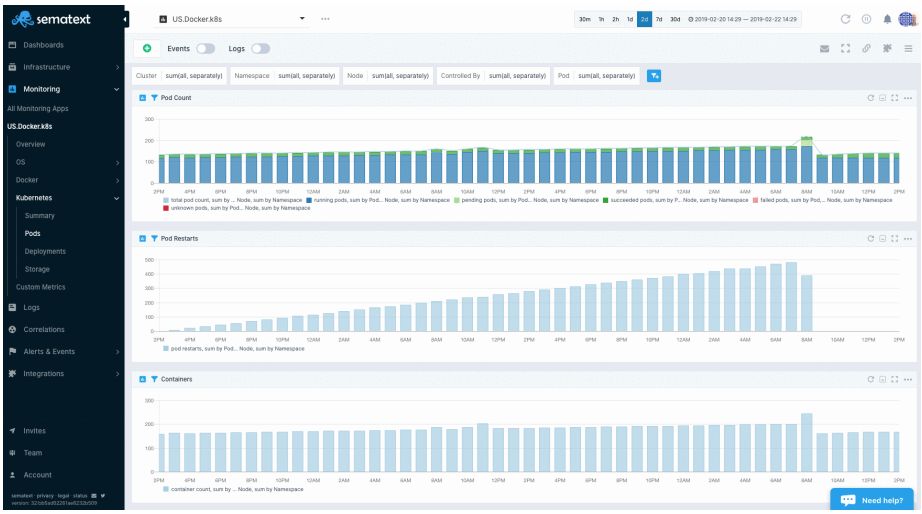
That is a lot of information and **Sematext organizes this information in reports** for **infrastructure monitoring**, **container monitoring**, and **Kubernetes cluster monitoring**.

Docker Alerting

To save you time Sematext automatically creates a set of default alert rules such as alerts for low disk space. You can create additional alerts on any metric.

There are 3 types of alerts in Sematext:

- **Heartbeat alerts**, which notify you when a server is down
- **Threshold-based alerts** that notify you when a metric value crosses a predefined threshold
- **Alerts** based on statistical **anomaly detection** that notify you when metric values suddenly change and deviate from the baseline



Docker Events

Events reflect changes in your infrastructure, from node restarts to container deployments, or changes in running containers. Events can track every Docker command. **Sematext Agent collects Events from the Docker Engine and Kubernetes API.** Whenever something goes wrong in your container stack, you can **correlate Logs or Metrics with the time of Docker events!**

Here's the list of Docker container events Sematext collects:

Container lifecycle events

- Create – when a container is created
- Start – when a container starts
- Restart – when a container gets restarted
- Stop – when a container stops
- Oom – when a container runs out of memory
- Pause – when a container gets paused
- Unpause – when a container continues to run after a pause
- Die – when the main process in a container dies
- Kill – when the container gets killed
- Destroy – when a container gets destroyed

Container runtime events

- Commit – when changes to the container filesystem are committed. Modifying deployed containers in production is not a common practice, therefore the commit could - indicate a "hack" and should be watched carefully.
- Copy – when files are copied from/to a container. Could indicate a potential data leak.

- Attach – when a process connects to container console – somebody is reading your container logs
- Detach – when a process disconnects from container console streams
- Exec – when a command is executed in container console, very helpful to investigate in potential hacker attacks
- Export – when a container gets exported
- Health_status – when health_status is checked
- Rename – when a container gets renamed
- Resize – when a container gets resized
- Top – when somebody list top processes in a container
- Update – when a container is updated e.g. with new labels

Container image events

- Delete – when an image gets deleted
- Import – when an image gets imported
- Load – when an image is loaded
- Pull – when an image is pulled from a registry
- Push – when an image is pushed to a registry
- Save – when an image is saved
- Tag – when an image is tagged with labels
- Untag – when an image tag is removed

Container plugin events

- Enable – when a plugin gets enabled
- Disable – when a plugin gets disabled
- Install – when a plugin gets installed
- Remove – when a plugin gets removed

Container volume events

- Create – when a volume is created
- Destroy – when a volume gets destroyed
- Mount – when a volume is mounted to a container
- Unmount – when a volume is removed from a container

Container network events

- Create – when a network is created
- Connect – when a container connects to a network
- Remove – when the network is removed
- Destroy – when a network is destroyed
- Disconnect – when a container disconnects from a network

Docker daemon events

- Reload

Docker services, nodes, secrets, and config events

- Create – on the creation of a resource
- Remove – on the removal of a resource
- Update – on the creation of a resource

Metrics Overview

The following information is collected and transmitted to Sematext Cloud or Sematext Enterprise.

Type

Description

Operating System Metrics

Host machine metrics

CPU Usage

Memory Usage

Network Stats

Disk I/O Stats

Container Metrics/Stats

CPU Usage / limits

Memory Usage / Limits / Fail Counters

Network Stats

Disk I/O Stats

Events

Agent Startup Event

server-info – created by spm-agent framework with node.js and OS version info on startup. Please note the agent is implemented in node.js.

Docker-info – Docker Version, API Version, Kernel Version on startup

Docker Events

Container Lifecycle Events| create, exec_create, destroy, export, …

Container Runtime Events

die, exec_start, kill, pause, restart, start, stop, unpause, …

Docker Logs

Default Fields

hostname / IP address

container id

container name

image name

message

Log formats

(detection and log parsers)

NGINX

APACHE httpd, Kafka, Solr, HBase, Zookeeper, Cassandra

MySQL

MongoDB

Redis

Elasticsearch

NSQ / Nsq.io

patterns are maintained here:

https://github.com/sematext/logagent-js

JSON, Plain Text

Supported Platforms

- Docker Engine >= 17.0.0
- Platforms using Docker:
  - Docker Cloud
  - Docker Data Center
  - Kubernetes
  - Mesos
  - CoreOS
  - Rancher
  - Amazon ECS
  - Red Hat OpenShift

Metrics Fields

Name	Type	Unit	Numeric Type	Label	Description
container.memory.usage	gauge	bytes	long	memory	container memory usage in bytes
container.memory.fail.count	counter		long	memory	the number of times that memory cgroup limit was exceeded
container.memory.limit	gauge	bytes	long	memory	the max allowed memory limit for the container cgroup
container.memory.limit.soft	gauge	bytes	long	soft memory limit	soft memory limit represents the initial memory reservation for the container
container.memory.rss	gauge	bytes	long	RSS memory	number of bytes of anonymous (file unmapped memory) and swap cache memory
container.cache.usage	gauge	bytes	long	cache memory	number of bytes of page cache memory
container.memory.pages.in	counter		long	memory pages in	memory pages in,description=the number of events each time the page is accounted to the cgroup
container.memory.pages.out	counter		long	memory pages out	memory pages out,description=the number of events each time a page is unaccounted from the cgroup
container.memory.pages.faults	counter		long	memory page faults	the number of page faults accounted to the cgroup

Name	Type	Unit	Numeric Type	Label	Description
container.memory.pages.faults.major	counter		long	major memory page faults	the number of major page faults accounted to the cgroup
container.swap.size	gauge	bytes	long	swap	the number of bytes of swap usage
container.swap.limit	gauge	bytes	long	swap limit	the swap memory usage limit
container.io.read	gauge		long	disk read	the number of bytes read from the disk
container.io.read.time	gauge	ns	long	disk read time	the total amount of time (in nanoseconds) between request dispatch and request completion
container.io.read.wait.time	counter	ns	long	disk read wait time	total amount of time the IO operations for this cgroup spent waiting in the scheduler queues
container.io.write	counter	bytes	long	disk write	the number of bytes written to the disk
container.io.write.time	counter	ns	long	disk write time	the total amount of time (in nanoseconds) between request dispatch and request completion
container.io.write.wait.time	counter	ns	long	disk write wait time	total amount of time the IO operations for this cgroup spent waiting in the scheduler queues

Name	Type	Unit	Numeric Type	Label	Description
container.cpu.weight	gauge	ns	long	disk io weight	specifies the relative proportion of block I/O access ranging from 100 to 1000
container.cpu.usage	gauge	percent	double	CPU usage	container CPU usage
container.cpu.throttled.time	counter	seconds	long	CPU throttled time	the total amount of time that processes have been throttled in the container cgroup
container.cpu.shares	gauge	ns	long	CPU shares	represents the weight of the cgroup that translates into the amount of CPU it is expected to get. Upon cgroup creation each group gets assigned a default of 1024
container.cpu.quota	gauge	microseconds	long	CPU quota	enforces a hard limit to the CPU time allocated to processes
container.cpu.period	gauge	microseconds	long	CPU period	is the time window expressed in microseconds that represents the period for which processes are allowed to run under specific quota
container.network.io.bytes	counter	bytes	long	network received bytes	received amount of bytes on the network interface
container.network.rx.packets	counter		long	network received packets	received amount of packets on the network interface

Name	Type	Unit	Numeric Type	Label	Description
container.network.rx.errors	counter		long	network rx errors	received amount of errors on the network interface
container.network.rx.dropped	counter		long	network packets rx dropped	amount of dropped inbound packets on the network interface
container.network.tx	counter		long	network transmitted	transmitted amount of bytes on the network interface
container.network.bytes	counter	bytes	long	network received	transmitted amount of bytes on the network interface
container.network.tx.packets	counter		long	network packets transmitted	transmitted amount of packets on the network interface
container.network.tx.errors	counter		long	network tx errors	transmitted amount of errors on the network interface
container.network.tx.dropped	counter		long	network packets tx dropped	amount of dropped outbound packets on the network interface

More about Docker Monitoring

- Docker Container Monitoring and Management Challenges
- Docker Container Performance Metrics
- Docker Container Monitoring Open Source Tools
- Docker Container Monitoring with Sematext