# Kubernetes - Monitoring

Monitoring is one of the key component for managing large clusters. For this, we have a number of tools.

## Monitoring with Prometheus

It is a monitoring and alerting system. It was built at SoundCloud and was open sourced in 2012. It handles the multi-dimensional data very well.

Prometheus has multiple components to participate in monitoring −

**Prometheus** − It is the core component that scraps and stores data.

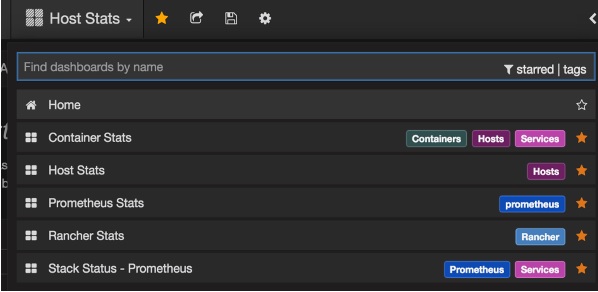
**Prometheus node explore** − Gets the host level matrices and exposes them to Prometheus.

**Ranch-eye** − is an **haproxy** and exposes **cAdvisor** stats to Prometheus.

**Grafana** − Visualization of data.

**InfuxDB** − Time series database specifically used to store data from rancher.

**Prom-ranch-exporter** − It is a simple node.js application, which helps in querying Rancher server for the status of stack of service.



## Sematext Docker Agent

It is a modern Docker-aware metrics, events, and log collection agent. It runs as a tiny container on every Docker host and collects logs, metrics, and events for all cluster node and containers. It discovers all containers (one pod might contain multiple containers) including containers for Kubernetes core services, if the core services are deployed in Docker containers. After its deployment, all logs and metrics are immediately available out of the box.

## Deploying Agents to Nodes

Kubernetes provides DeamonSets which ensures pods are added to the cluster.

## Configuring SemaText Docker Agent

It is configured via environment variables.

Get a free account at [apps.sematext.com](https://apps.sematext.com/ui/registration" \t "https://www.tutorialspoint.com/kubernetes/_blank), if you don’t have one already.

Create an SPM App of type “Docker” to obtain the SPM App Token. SPM App will hold your Kubernetes performance metrics and event.

Create a Logsene App to obtain the Logsene App Token. Logsene App will hold your Kubernetes logs.

Edit values of LOGSENE\_TOKEN and SPM\_TOKEN in the DaemonSet definition as shown below.

Grab the latest sematext-agent-daemonset.yml (raw plain-text) template (also shown below).

Store it somewhere on the disk.

Replace the SPM\_TOKEN and LOGSENE\_TOKEN placeholders with your SPM and Logsene App tokens.

## Create DaemonSet Object

apiVersion: extensions/v1beta1

kind: DaemonSet

metadata:

name: sematext-agent

spec:

template:

metadata:

labels:

app: sematext-agent

spec:

selector: {}

dnsPolicy: "ClusterFirst"

restartPolicy: "Always"

containers:

- name: sematext-agent

image: sematext/sematext-agent-docker:latest

imagePullPolicy: "Always"

env:

- name: SPM\_TOKEN

value: "REPLACE THIS WITH YOUR SPM TOKEN"

- name: LOGSENE\_TOKEN

value: "REPLACE THIS WITH YOUR LOGSENE TOKEN"

- name: KUBERNETES

value: "1"

volumeMounts:

- mountPath: /var/run/docker.sock

name: docker-sock

- mountPath: /etc/localtime

name: localtime

volumes:

- name: docker-sock

hostPath:

path: /var/run/docker.sock

- name: localtime

hostPath:

path: /etc/localtime

## Running the Sematext Agent Docker with kubectl

$ kubectl create -f sematext-agent-daemonset.yml

daemonset "sematext-agent-daemonset" created

## Kubernetes Log

Kubernetes containers’ logs are not much different from Docker container logs. However, Kubernetes users need to view logs for the deployed pods. Hence, it is very useful to have Kubernetes-specific information available for log search, such as −

* Kubernetes namespace
* Kubernetes pod name
* Kubernetes container name
* Docker image name
* Kubernetes UID

## Using ELK Stack and LogSpout

ELK stack includes Elasticsearch, Logstash, and Kibana. To collect and forward the logs to the logging platform, we will use LogSpout (though there are other options such as FluentD).

The following code shows how to set up ELK cluster on Kubernetes and create service for ElasticSearch −

apiVersion: v1

kind: Service

metadata:

name: elasticsearch

namespace: elk

labels:

component: elasticsearch

spec:

type: LoadBalancer

selector:

component: elasticsearch

ports:

- name: http

port: 9200

protocol: TCP

- name: transport

port: 9300

protocol: TCP

## Creating Replication Controller

apiVersion: v1

kind: ReplicationController

metadata:

name: es

namespace: elk

labels:

component: elasticsearch

spec:

replicas: 1

template:

metadata:

labels:

component: elasticsearch

spec:

serviceAccount: elasticsearch

containers:

- name: es

securityContext:

capabilities:

add:

- IPC\_LOCK

image: quay.io/pires/docker-elasticsearch-kubernetes:1.7.1-4

env:

- name: KUBERNETES\_CA\_CERTIFICATE\_FILE

value: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

- name: NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

- name: "CLUSTER\_NAME"

value: "myesdb"

- name: "DISCOVERY\_SERVICE"

value: "elasticsearch"

- name: NODE\_MASTER

value: "true"

- name: NODE\_DATA

value: "true"

- name: HTTP\_ENABLE

value: "true"

ports:

- containerPort: 9200

name: http

protocol: TCP

- containerPort: 9300

volumeMounts:

- mountPath: /data

name: storage

volumes:

- name: storage

emptyDir: {}

## Kibana URL

For Kibana, we provide the Elasticsearch URL as an environment variable.

- name: KIBANA\_ES\_URL

value: "http://elasticsearch.elk.svc.cluster.local:9200"

- name: KUBERNETES\_TRUST\_CERT

value: "true"

Kibana UI will be reachable at container port 5601 and corresponding host/Node Port combination. When you begin, there won’t be any data in Kibana (which is expected as you have not pushed any data).