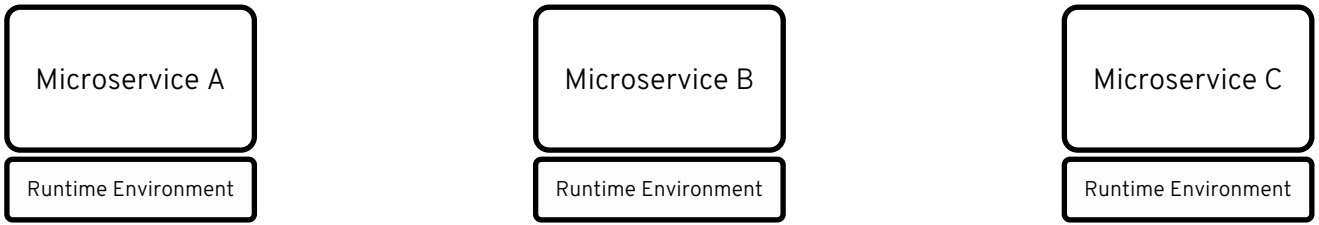
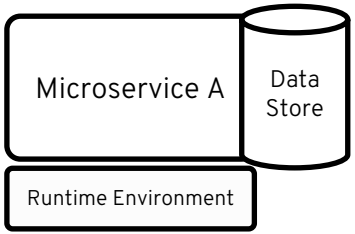
CHALLENGES IN A MICROSERVICES AGE: MONITORING, LOGGING AND TRACING ON OPENSHIFT

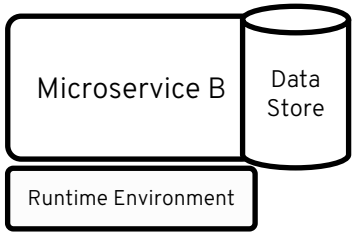
Martin Etmajer
Technology Lead @Dynatrace
May 4, 2017

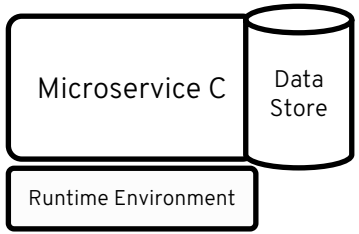
WHY A CHALLENGE?

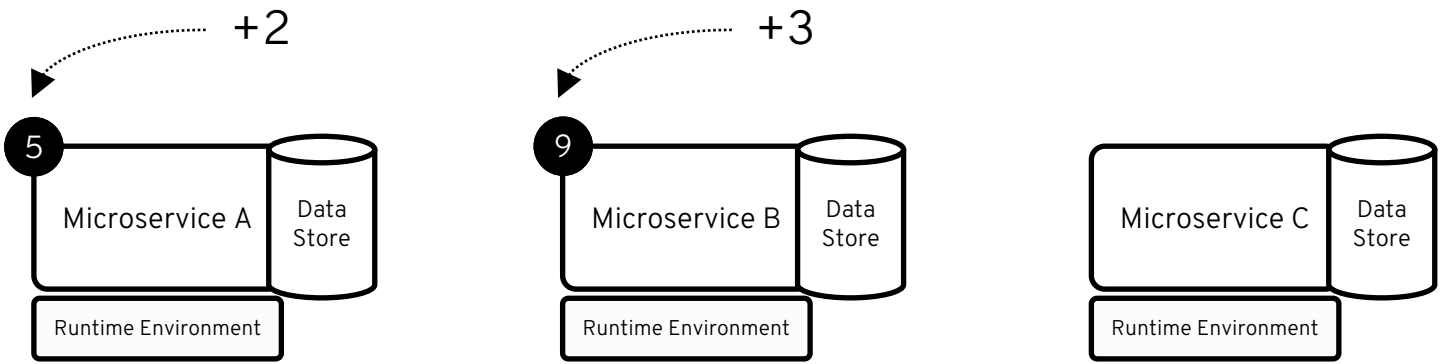

Microservice A
Microservice B
Microservice C

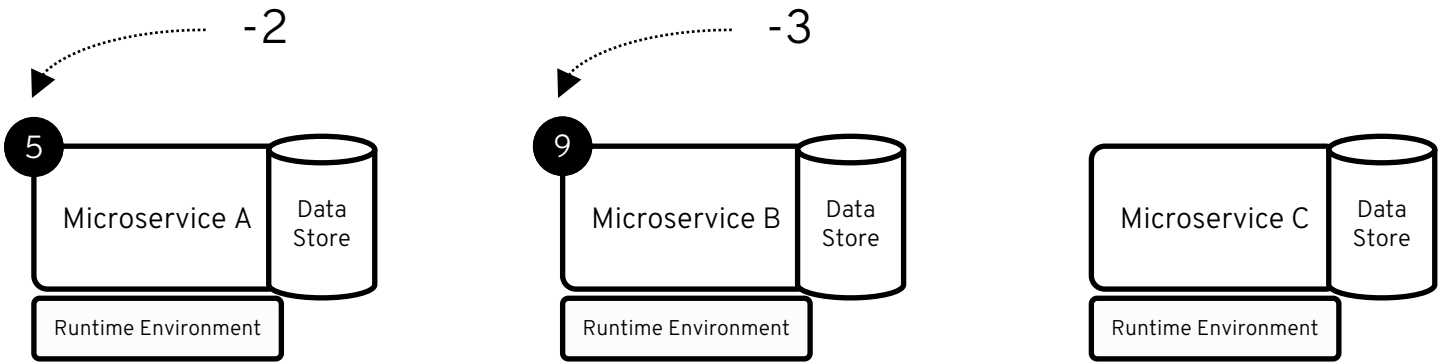

Microservice A
Runtime Environment
Microservice B
Runtime Environment
Microservice C
Runtime Environment

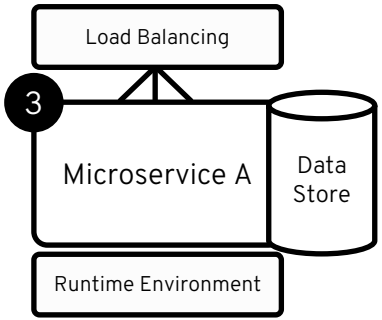

Microservice A
Data Store
Runtime Environment

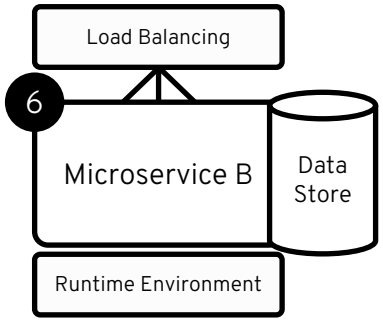

Microservice B
Data Store
Runtime Environment

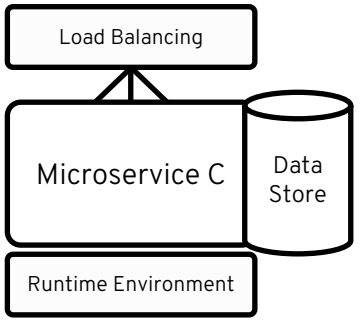

Microservice C
Data Store
Runtime Environment

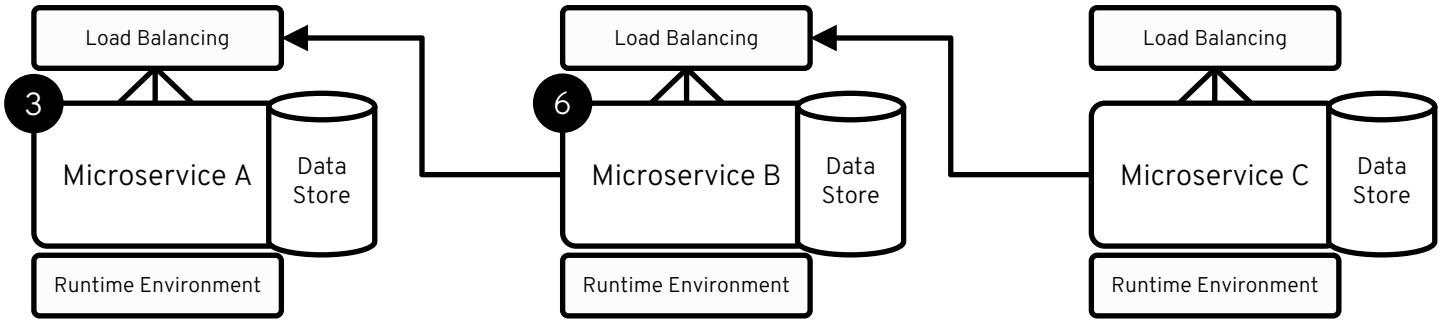

+2
5
Microservice A
Data Store
Runtime Environment
+3
9
Microservice B
Data Store
Runtime Environment
Microservice C
Data Store
Runtime Environment

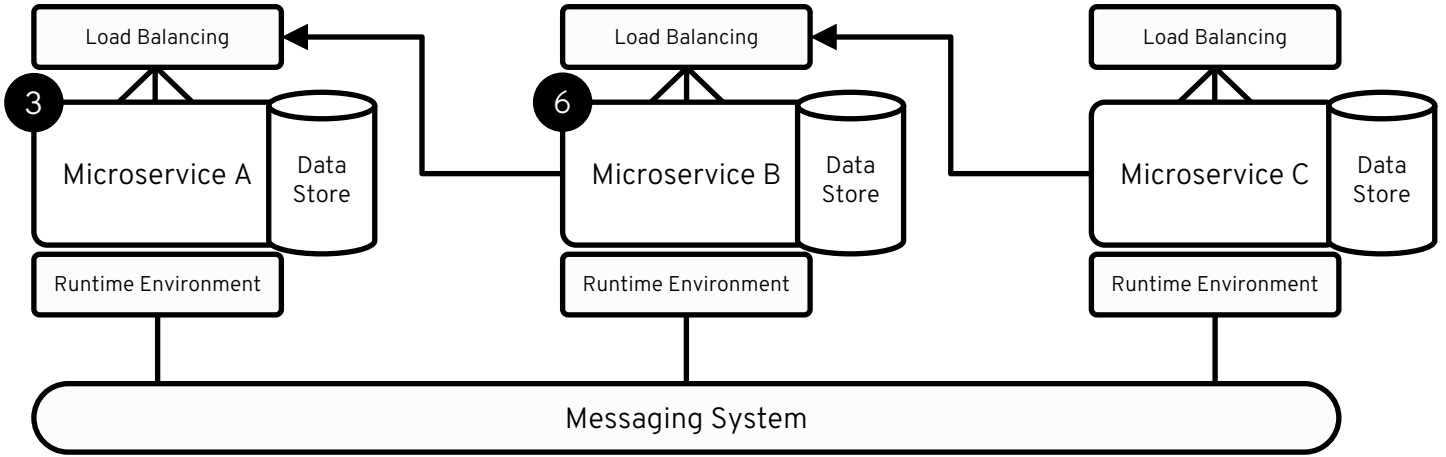

-2
5
Microservice A
Data Store
Runtime Environment
-3
9
Microservice B
Data Store
Runtime Environment
Microservice C
Data Store
Runtime Environment

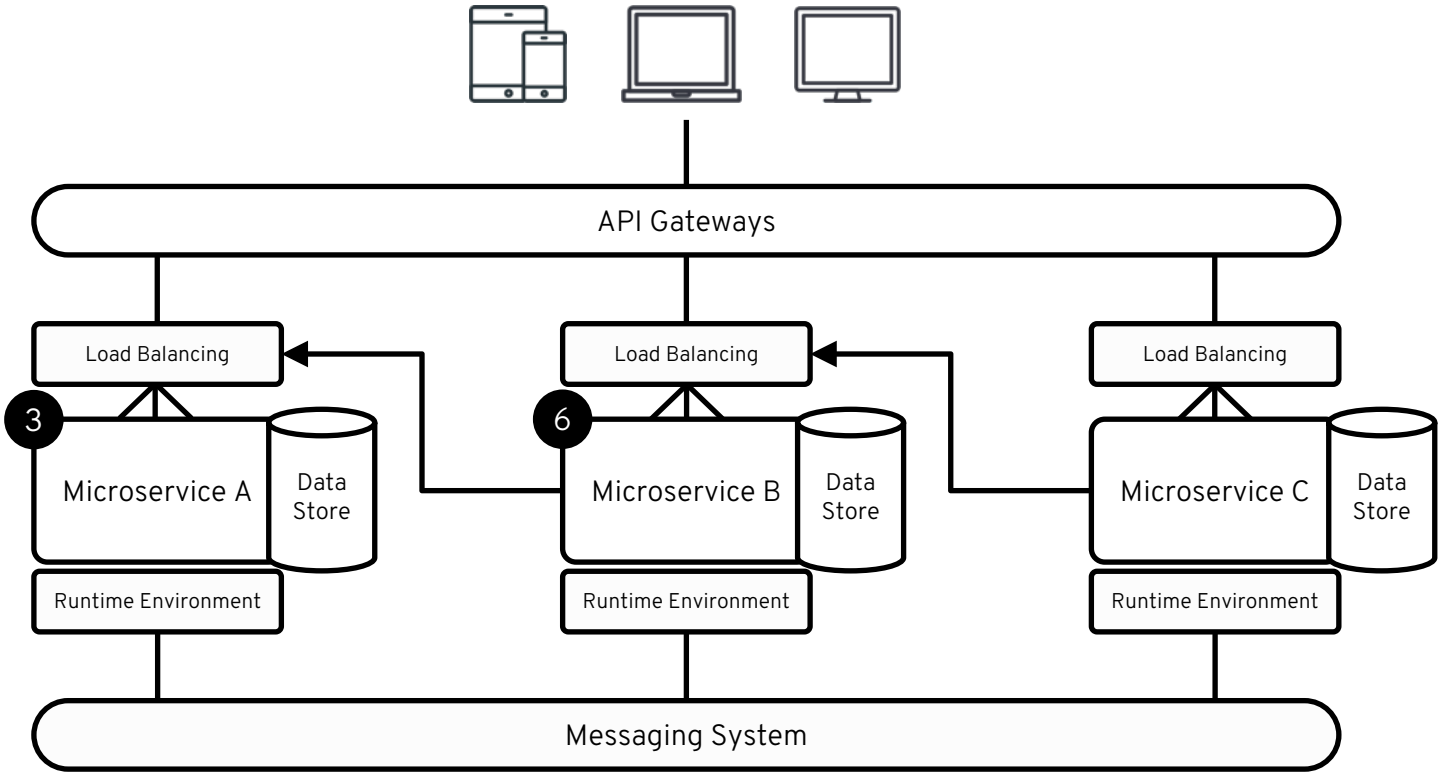

Load Balancing
3
Microservice A
Data Store
Runtime Environment

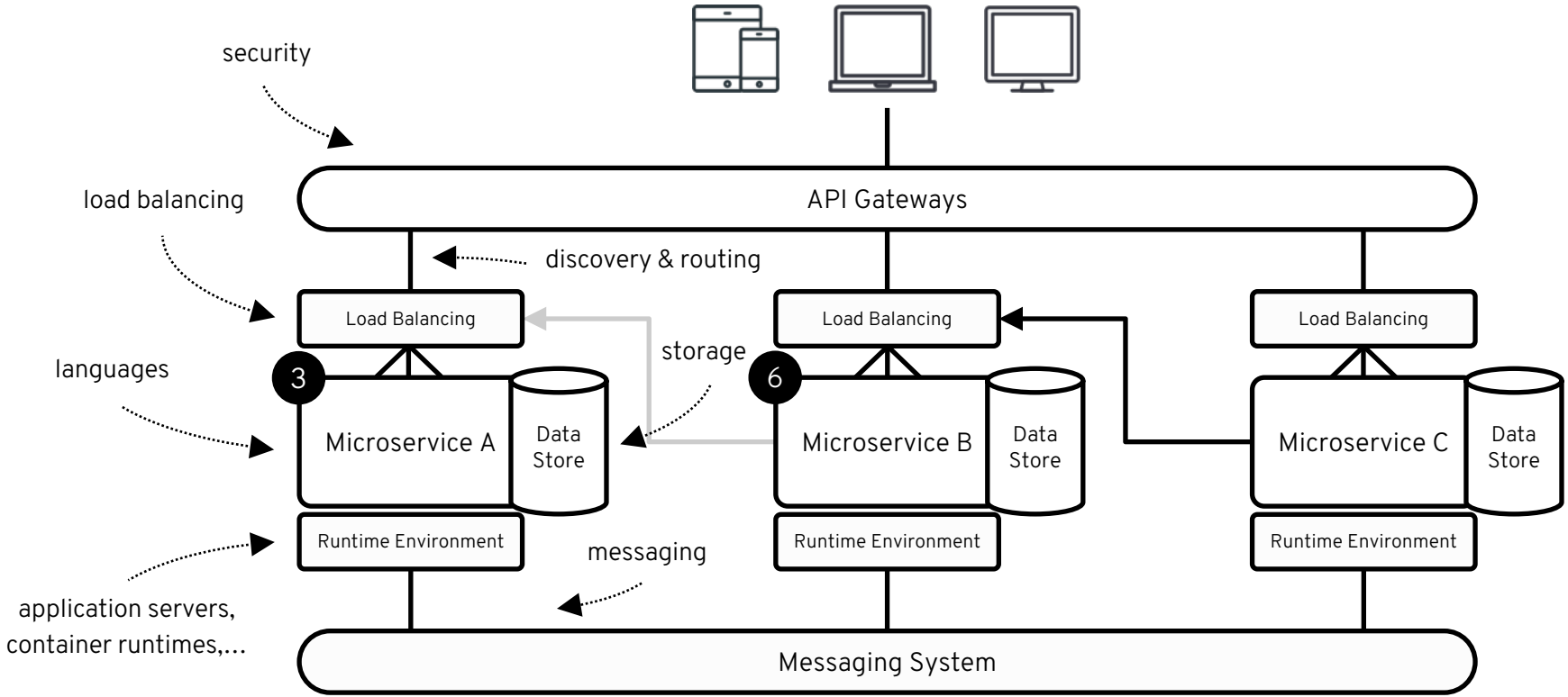

Load Balancing
6
Microservice B
Data Store
Runtime Environment


Load Balancing
Microservice C
Data Store
Runtime Environment


Load Balancing
3
Microservice A
Data Store
Runtime Environment
Load Balancing
6
Microservice B
Data Store
Runtime Environment
Load Balancing
Microservice C
Data Store
Runtime Environment


Load Balancing
Load Balancing
Load Balancing
3
6
Microservice A
Data Store
Microservice B
Data Store
Microservice C
Data Store
Runtime Environment
Runtime Environment
Runtime Environment
Messaging System


API Gateways
Load Balancing
Load Balancing
Load Balancing
3
6
Microservice A
Data Store
Microservice B
Data Store
Microservice C
Data Store
Runtime Environment
Runtime Environment
Runtime Environment
Messaging System


security
load balancing
languages
application servers,
container runtimes,...
API Gateways
discovery & routing
Load Balancing
Load Balancing
Load Balancing
storage
3
6
Microservice A
Data Store
Microservice B
Data Store
Microservice C
Data Store
Runtime Environment
Runtime Environment
Runtime Environment
messaging
Messaging System

Demo

RED HAT „Hello World“ MSA

Red Hat - Hello world MSA (Microservices Architecture)

Browser as a client | API Gateway | Service chaining | Hystrix Dashboard | Zipkin Dashboard

Using Browser as a Client

Refresh Results

Hola Service
(JAX-RS / WildFly Swarm)

Hola de hola-1-bcnbq

Olá Service
(Spring Boot)

Olá de ola-1-hq56h

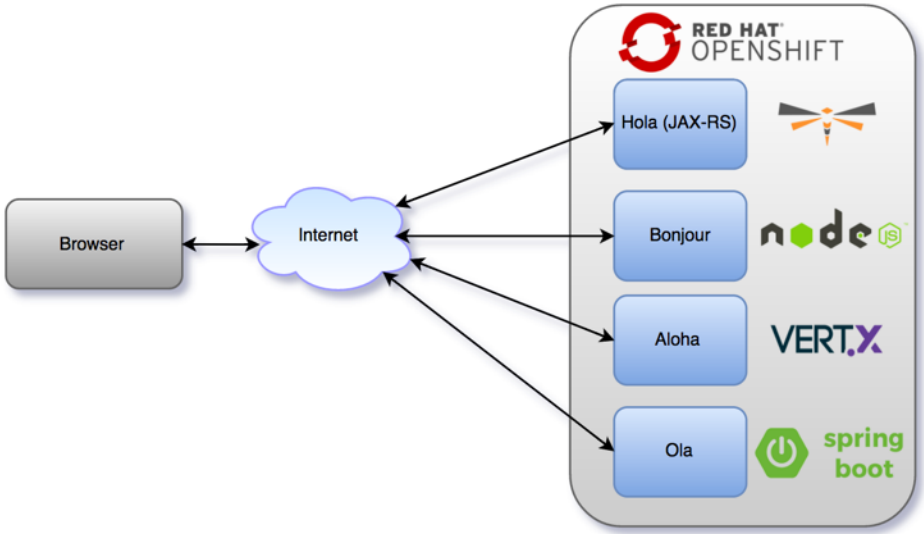
Bonjour Service
(NodeJS / Express)

Bonjour de bonjour-1-n3qnb

Aloha Service
(Vert.x)

Aloha mai aloha-7-px0qf

Refresh Results



Red Hat - Hello world MSA (Microservices Architecture)

Browser as a client | API Gateway | Service chaining | Hystrix Dashboard | Zipkin Dashboard

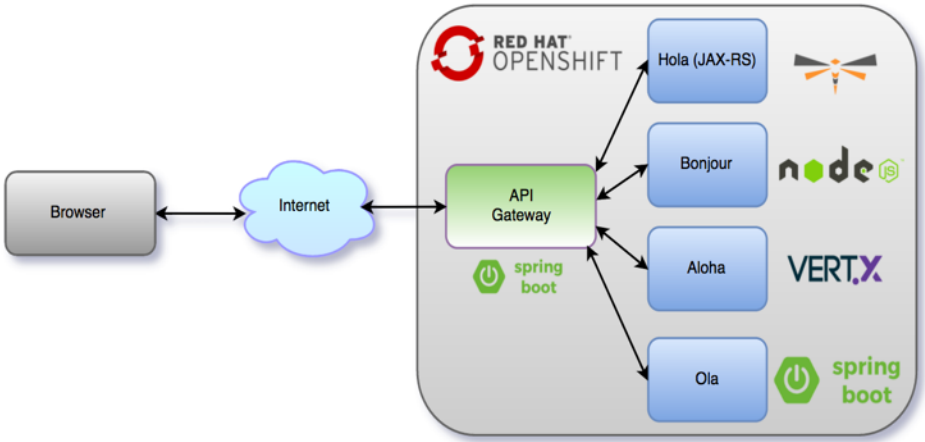
Using an API Gateway

Refresh Results

API Gateway

- Aloha mai aloha-7-px0qf
- Hola de hola-1-bcnbq
- Olá de ola-1-hq56h
- Bonjour response (fallback)

Refresh Results



Red Hat - Hello world MSA (Microservices Architecture)

Browser as a client | API Gateway | Service chaining | Hystrix Dashboard | Zipkin Dashboard

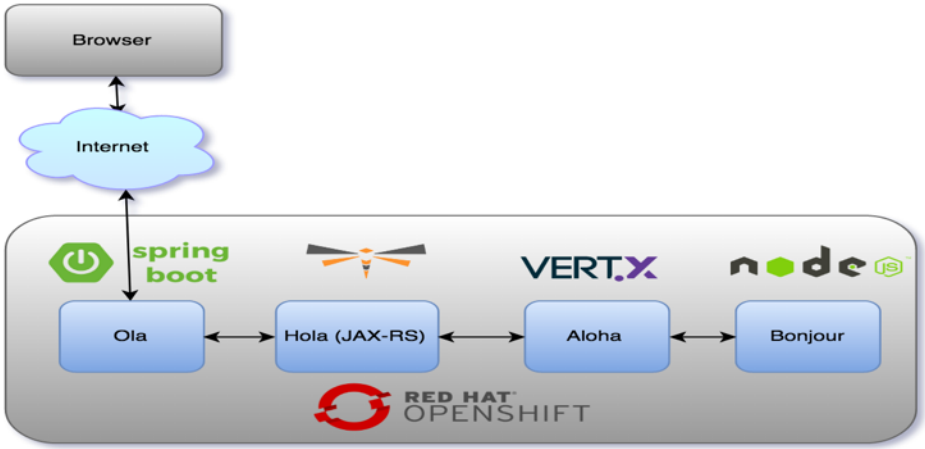
Using Service chaining

Refresh Results

Service Chaining

- 1 - Olá de ola-1-hq56h
- 2 - Hola de hola-1-bcnbq
- 3 - Aloha mai aloha-7-px0qf
- 4 - Bonjour de bonjour-1-n3qnb

Refresh Results



MONITORING

Monitoring

container health checks with *OpenShift Liveness Probes*

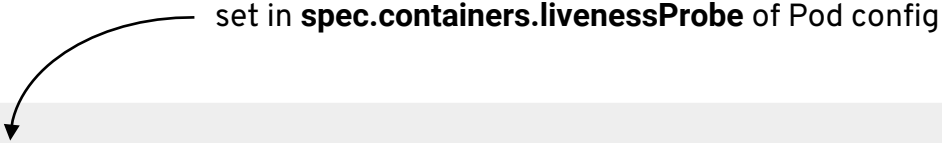
What?

- a liveness probe periodically checks if a container is able to handle requests
- when a condition has not been met within a given timeout, the probe fails
- if a probe fails, the container is killed and subjected to its restart policy

Monitoring

container health checks with *OpenShift Liveness Probes*

How?




```
…
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 15
  timeoutSeconds: 1
…
```

Monitoring

container health checks with *OpenShift Liveness Probes*

How?

supports HTTP Get, Container Command and TCP Socket types

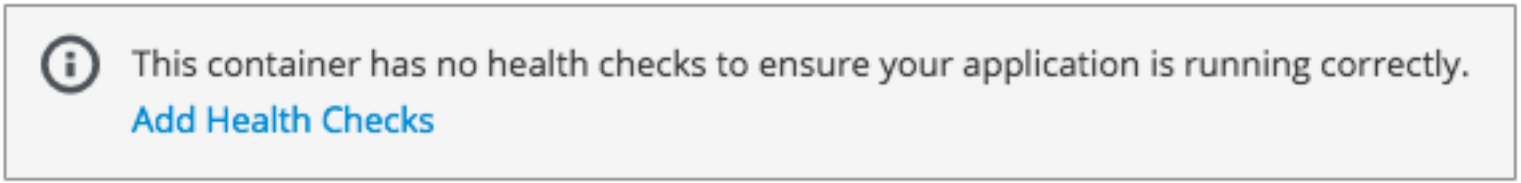
```
…
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 15
  timeoutSeconds: 1
…
```

Demo

Monitoring

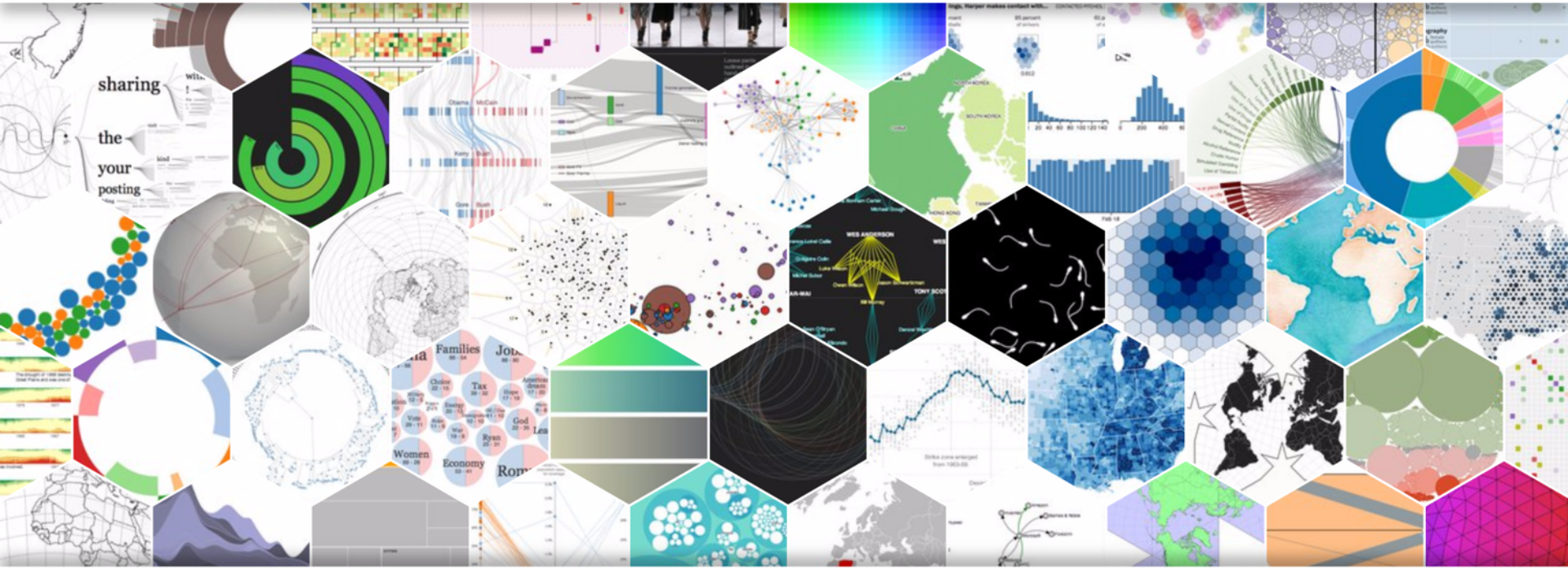
container health checks with *OpenShift Liveness Probes*

How?



Monitoring

Where are the fancy dashboards?



Source: d3js.org - Data-Driven Documents

Monitoring

container, pod and node metrics with *Heapster and Hawkular Metrics*

Heapster

Hawkular Metrics

Compute Resource Usage Analysis and Monitoring of Container Clusters

README.md

Heapster

Heapster enables Container Cluster Monitoring and Performance Analysis.

Heapster currently supports Kubernetes and CoreOS natively. *Heapster is compatible with kubernetes versions starting from v1.0.6 only*

It can be extended to support other cluster management solutions easily.

Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics via REST endpoints. **Note: Some of the endpoints are only valid in Kubernetes clusters**

Heapster supports multiple sources of data. More information here.

Heapster supports a pluggable storage backend. It supports InfluxDB with Grafana, Google Cloud Monitoring, ElasticSearch, Google Cloud Logging, Hawkular, Riemann and Kafka. We welcome patches that add additional storage backends. Documentation on storage sinks here The current version of Storage Schema is documented here.

kubernetes / heapster

Watch 116 | Star 1,169 | Fork 539

Code | Issues 137 | Pull requests 23 | Projects 0 | Wiki | Pulse | Graphs

Compute Resource Usage Analysis and Monitoring of Container Clusters

README.md

Heapster

godoc reference | build passing

Heapster enables Container Cluster Monitoring and Performance Analysis.

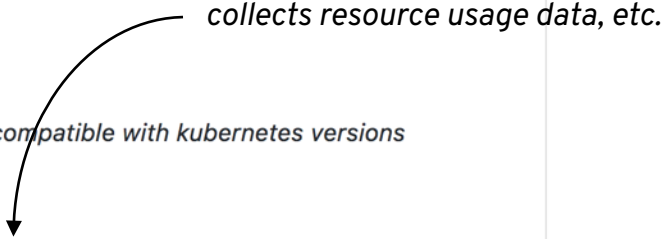
Heapster currently supports Kubernetes and CoreOS natively. *Heapster is compatible with kubernetes versions starting from v1.0.6 only*

It can be extended to support other cluster management solutions easily.

Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics via REST endpoints. **Note: Some of the endpoints are only valid in Kubernetes clusters**

Heapster supports multiple sources of data. More information here.

Heapster supports a pluggable storage backend. It supports InfluxDB with Grafana, Google Cloud Monitoring, ElasticSearch, Google Cloud Logging, Hawkular, Riemann and Kafka. We welcome patches that add additional storage backends. Documentation on storage sinks here The current version of Storage Schema is documented here.



kubernetes / heapster

Watch 116 | Star 1,169 | Fork 539

Code | Issues 137 | Pull requests 23 | Projects 0 | Wiki | Pulse | Graphs

Compute Resource Usage Analysis and Monitoring of Container Clusters

README.md

Heapster

godoc reference | build passing

exposes metrics via REST

Heapster enables Container Cluster Monitoring and Performance Analysis.

Heapster currently supports Kubernetes and CoreOS natively. *Heapster is compatible with kubernetes versions starting from v1.0.6 only*

It can be extended to support other cluster management solutions easily.

Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics via REST endpoints. **Note: Some of the endpoints are only valid in Kubernetes clusters**

Heapster supports multiple sources of data. More information here.

Heapster supports a pluggable storage backend. It supports InfluxDB with Grafana, Google Cloud Monitoring, ElasticSearch, Google Cloud Logging, Hawkular, Riemann and Kafka. We welcome patches that add additional storage backends. Documentation on storage sinks here The current version of Storage Schema is documented here.

kubernetes / heapster

Watch 116 | Star 1,169 | Fork 539

Code | Issues 137 | Pull requests 23 | Projects 0 | Wiki | Pulse | Graphs

Compute Resource Usage Analysis and Monitoring of Container Clusters

README.md

Heapster

godoc reference | build passing

Heapster enables Container Cluster Monitoring and Performance Analysis.

Heapster currently supports Kubernetes and CoreOS natively. *Heapster is compatible with kubernetes versions starting from v1.0.6 only*

It can be extended to support other cluster management solutions easily.

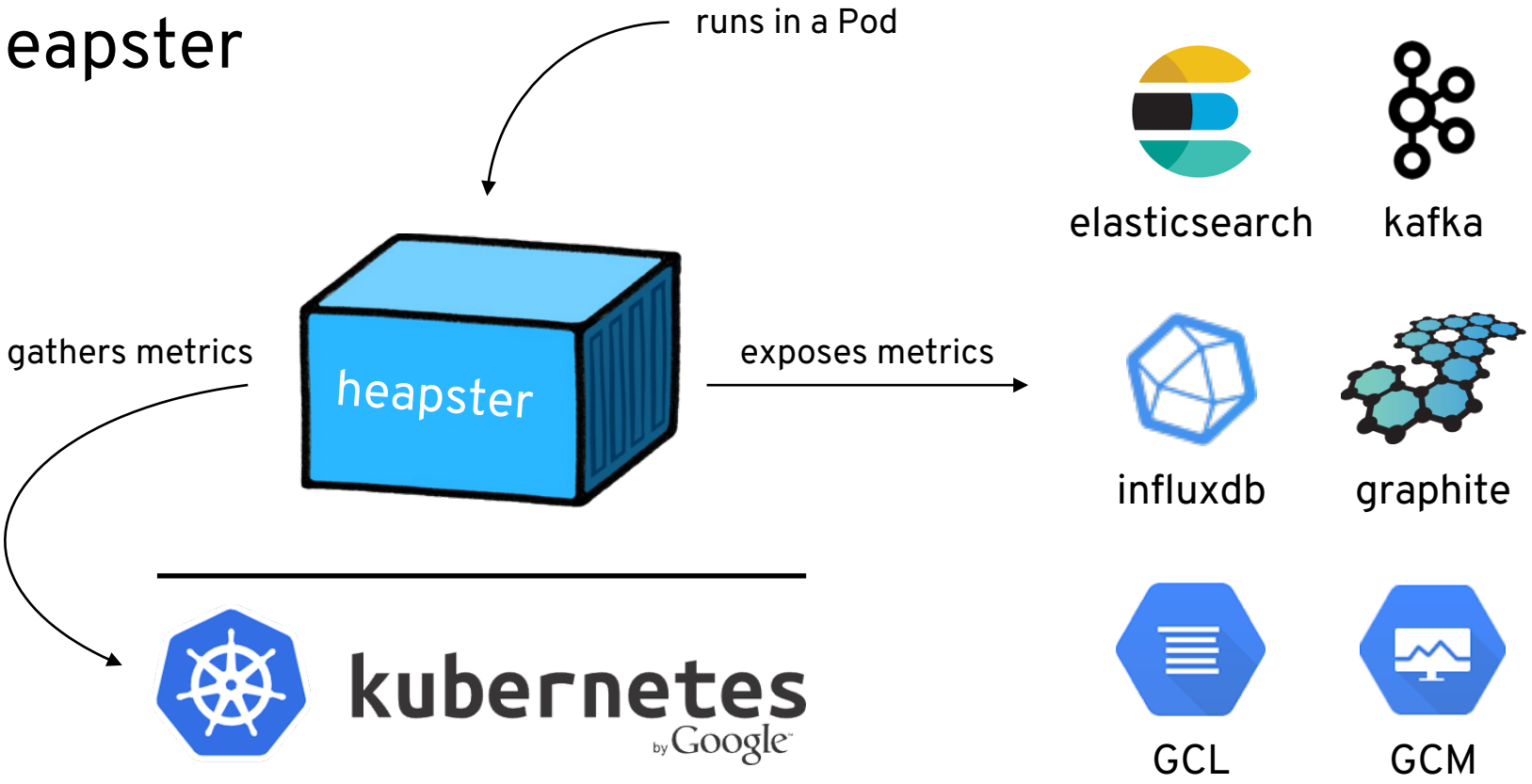
Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics via REST endpoints. **Note: Some of the endpoints are only valid in Kubernetes clusters**

Heapster supports multiple sources of data. More information here.

supports various storage backends

Heapster supports a pluggable storage backend. It supports InfluxDB with Grafana, Google Cloud Monitoring, ElasticSearch, Google Cloud Logging, Hawkular, Riemann and Kafka. We welcome patches that add additional storage backends. Documentation on storage sinks here The current version of Storage Schema is documented here.

Heapster


runs in a Pod
elasticsearch
kafka
gathers metrics
heapster
exposes metrics
influxdb
graphite
kubernetes
by Google
GCL
GCM

kubernetes / heapster

Watch 116 | Star 1,175 | Fork 541

Code | Issues 139 | Pull requests 25 | Projects 0 | Wiki | Pulse | Graphs

Compute Resource Usage Analysis and Monitoring of Container Clusters

README.md

Heapster

Heapster enables Container Cluster Monitoring and Performance Analysis.

Heapster currently supports Kubernetes and CoreOS natively. *Heapster is compatible with kubernetes versions starting from v1.0.6 only*

It can be extended to support other cluster management solutions easily.

Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics via REST endpoints. **Note: Some of the endpoints are only valid in Kubernetes clusters**

Heapster supports multiple sources of data. More information here.

Heapster supports a pluggable storage backend. It supports InfluxDB with Grafana, Google Cloud Monitoring, ElasticSearch, Google Cloud Logging, Hawkular, Riemann and Kafka. We welcome patches that add additional storage backends. Documentation on storage sinks here The current version of Storage Schema is documented here.

e.g. Hawkular

Hawkular

A collection of open source monitoring components by Red Hat

Hawkular

Monitoring services: Metrics, Alerting, Inventory, Application Performance Management

http://www.hawkular.org

Repositories People 10

Pinned repositories

hawkular-metrics

Time Series Metrics Engine based on Cassandra

Java ★ 145 62

hawkular-apm

Distributed Tracing and Application Performance Management

Java ★ 105 36

hawkular-alerts

Alerting subsystem for Hawkular

Java ★ 22 22

hawkular-client-ruby

Ruby client for Hawkular

hawkular-agent

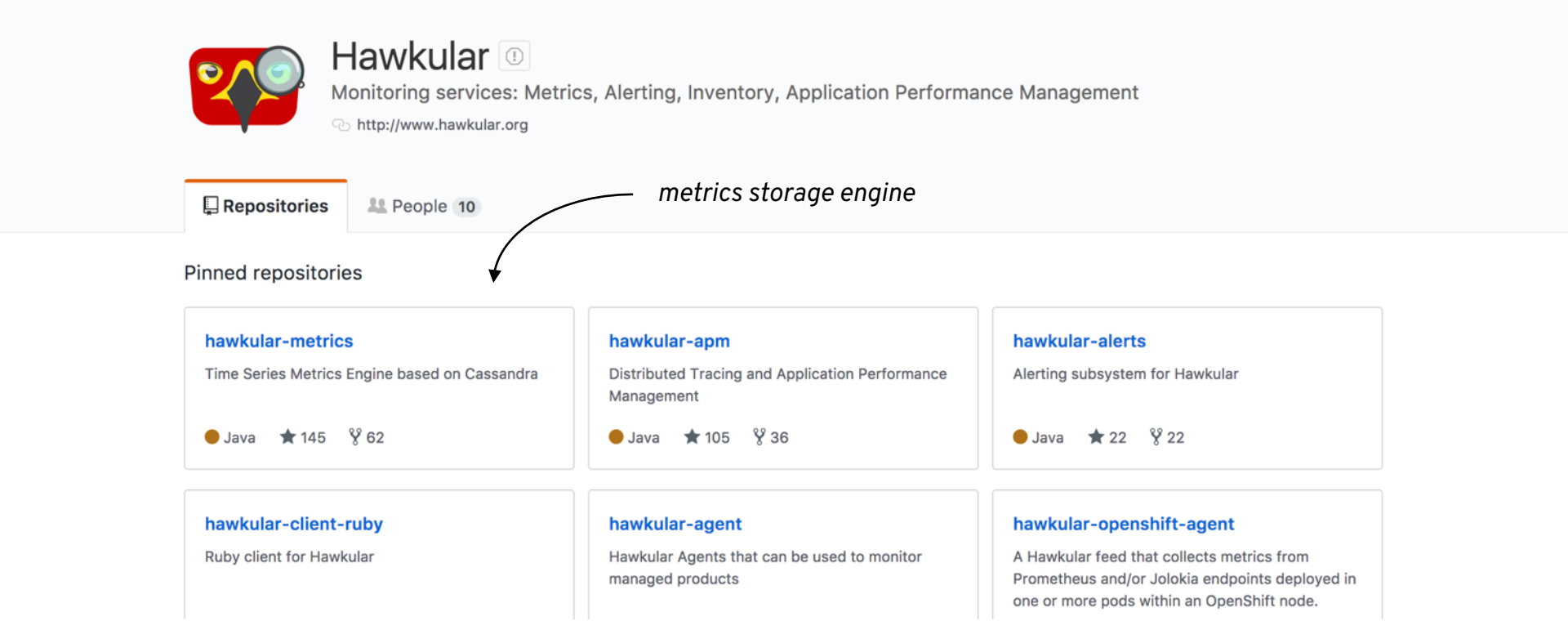
Hawkular Agents that can be used to monitor managed products

hawkular-openshift-agent

A Hawkular feed that collects metrics from Prometheus and/or Jolokia endpoints deployed in one or more pods within an OpenShift node.

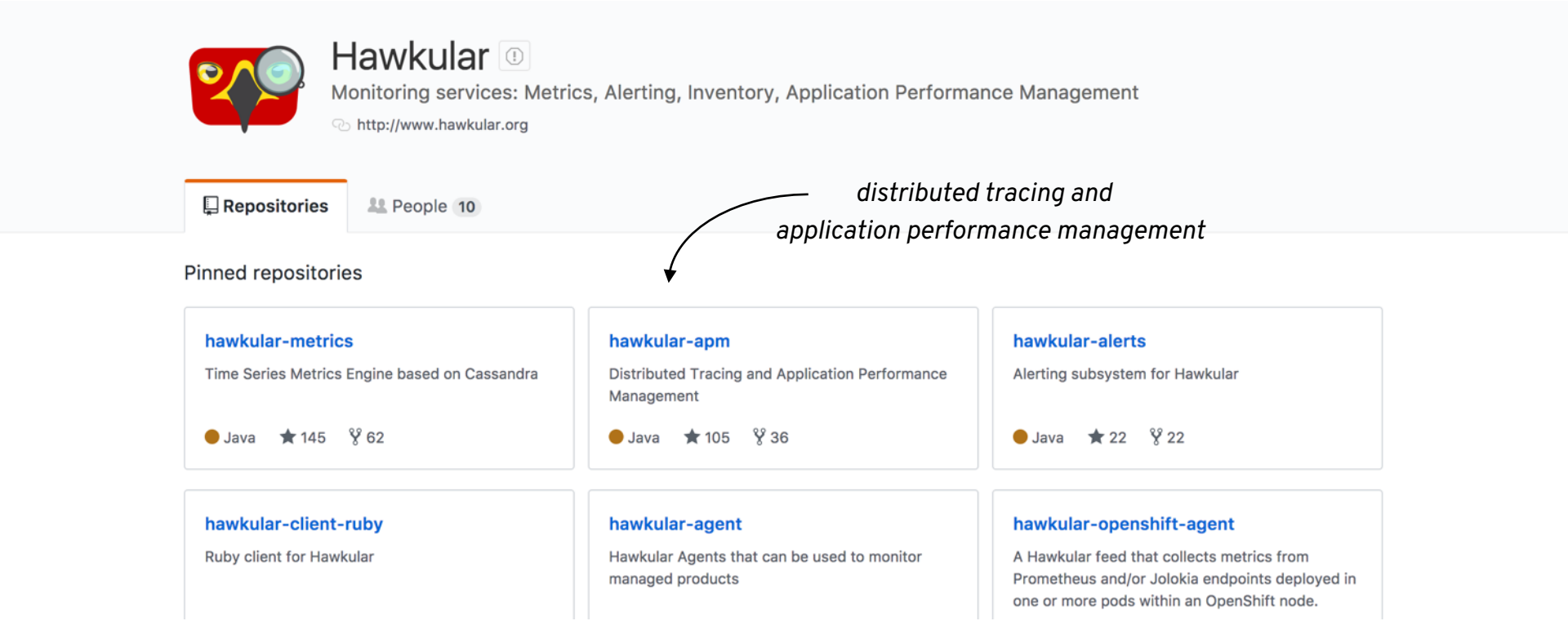
Hawkular

A collection of open source monitoring components by Red Hat



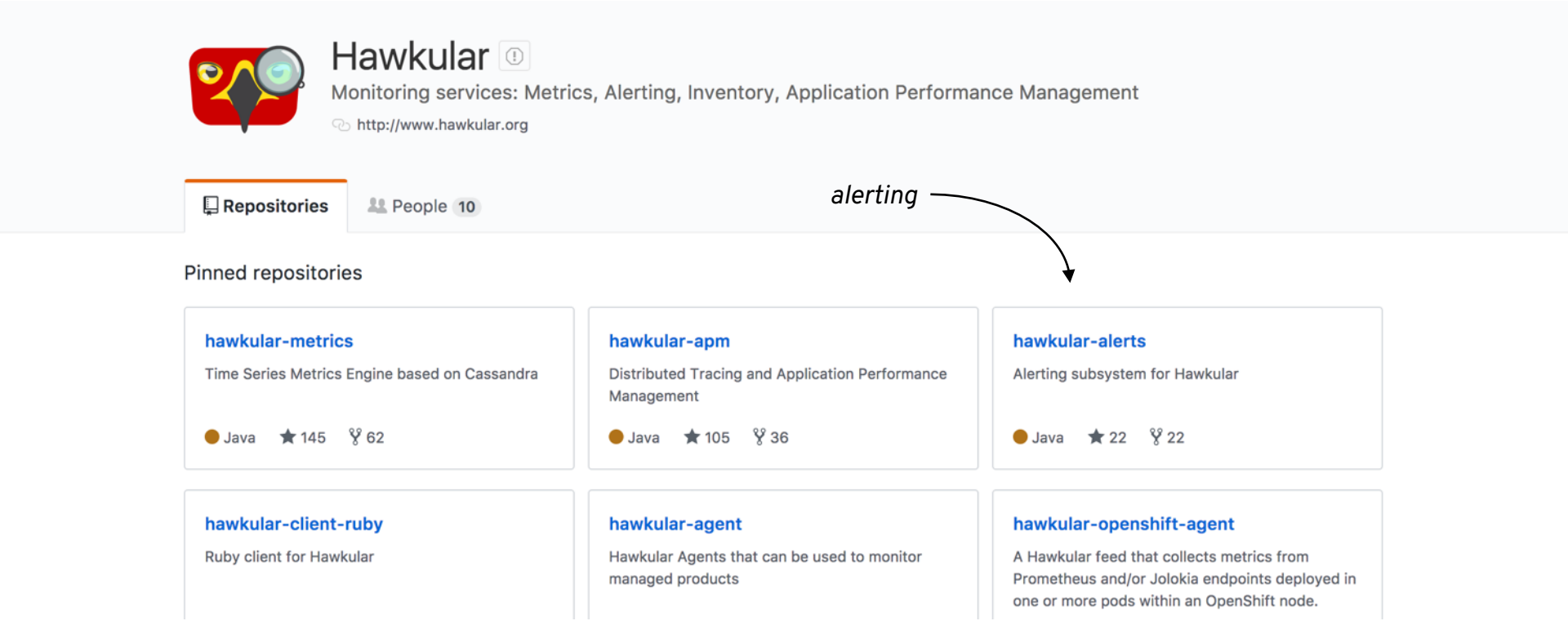
Hawkular

A collection of open source monitoring components by Red Hat



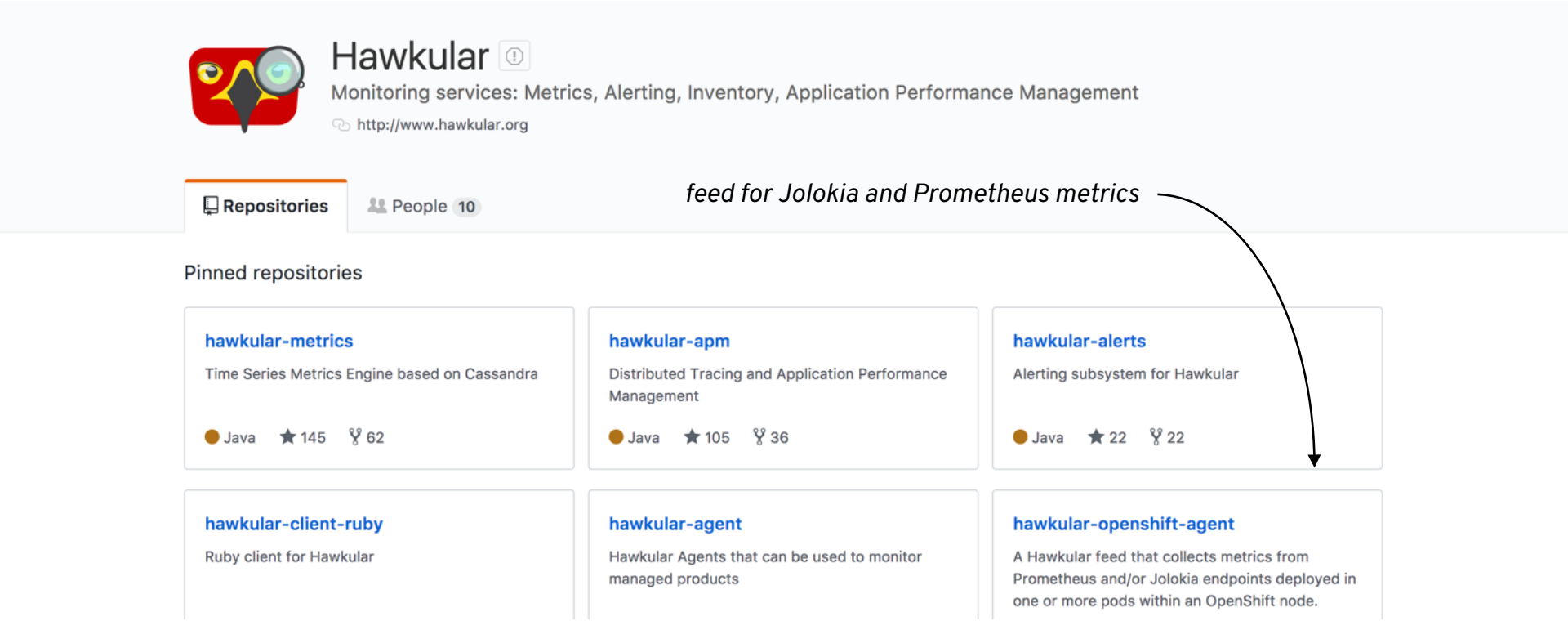
Hawkular

A collection of open source monitoring components by Red Hat



Hawkular

A collection of open source monitoring components by Red Hat



hawkular / hawkular-metrics

Watch 35 | Star 146 | Fork 62

Code | Pull requests 10 | Projects 0 | Pulse | Graphs

Time Series Metrics Engine based on Cassandra

monitoring metrics java alerting hawkular

README.adoc

Hawkular Metrics, a storage engine for metric data

build passing

coverity passed 25 new defects

a metrics storage engine

About

Hawkular Metrics is the metric data storage engine part of Hawkular community.

It relies on Apache Cassandra as a backend and is comprised of:

- a core library
- a REST/HTTP interface

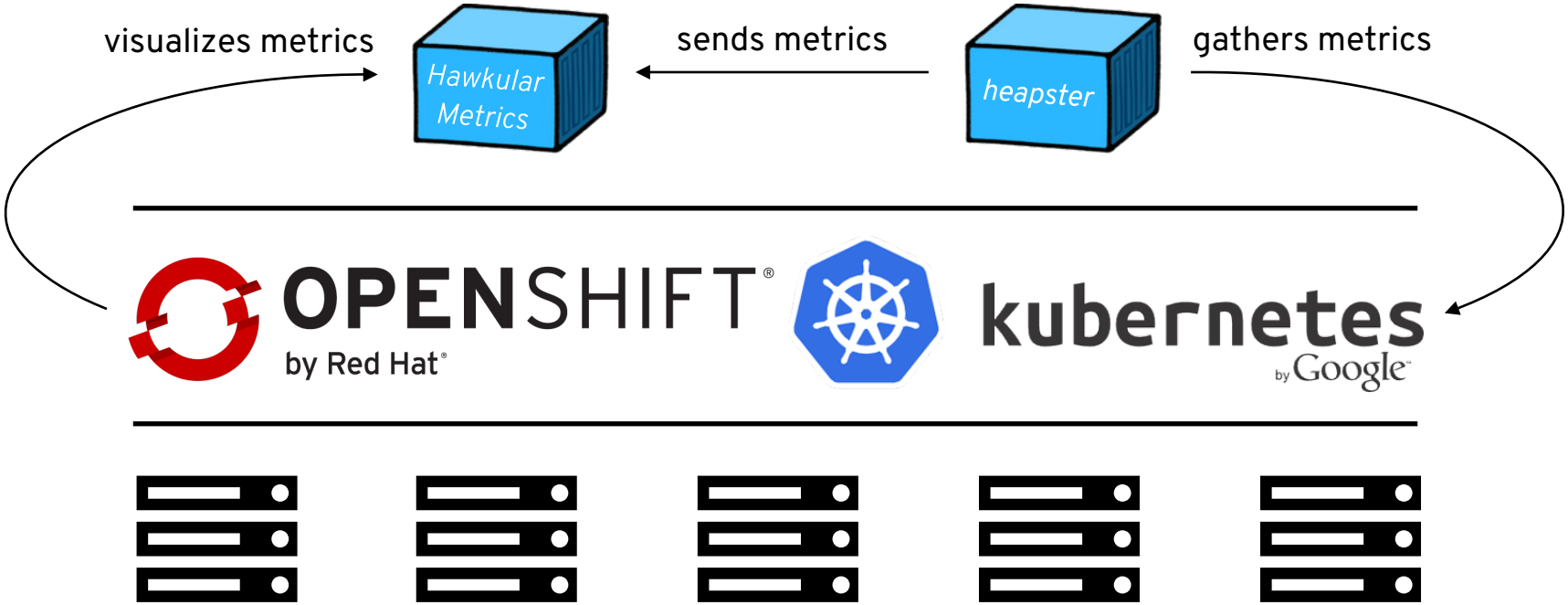
with a REST interface

Important	Cassandra 3.0.12 or later is required.

Demo

Monitoring

container, pod and node metrics with *Heapster and Hawkular Metrics*



Project
helloworld

Add to project

admin

APPLICATION
aloha

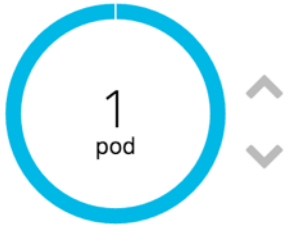
http://aloha-helloworld.54.91.241.179.nip.io

DEPLOYMENT
aloha, #1

CONTAINER: ALOHA

Image: metmajer/redhatmsa-aloha:hawkular-apm

Ports: 8080/TCP and 2 others



Networking

SERVICE Internal Traffic

aloha

8080/TCP (8080-tcp) → 8080 and 2 others

ROUTES External Traffic

http://aloha-helloworld.54.91.241.179.nip.io

Route aloha, target port 8080

http://jolokia-aloha-helloworld.54.91.241.179.nip.io/jolokia/

Route jolokia-aloha, target port 8778

Project
helloworld

Add to project

admin

APPLICATION
aloha

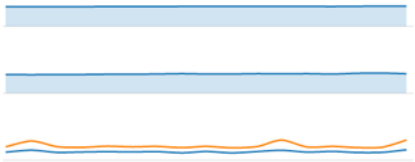
http://aloha-helloworld.54.91.241.179.nip.io

DEPLOYMENT
aloha, #1

CONTAINER: ALOHA

Image: metmajer/redhatmsa-aloha:hawkular-apm

Ports: 8080/TCP and 2 others

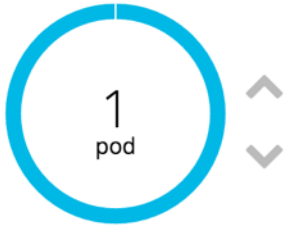


500
MiB Memory

0.03
Cores CPU

2.0
KiB/s Network

Average Usage Last 15 Minutes



Networking

SERVICE Internal Traffic

aloha

8080/TCP (8080-tcp) → 8080 and 2 others

ROUTES External Traffic

http://aloha-helloworld.54.91.241.179.nip.io

Route aloha, target port 8080

http://jolokia-aloha-helloworld.54.91.241.179.nip.io/jolokia/

Route jolokia-aloha, target port 8778

Pods » aloha-1-4t9m2

aloha-1-4t9m2 created 4 hours ago

Actions

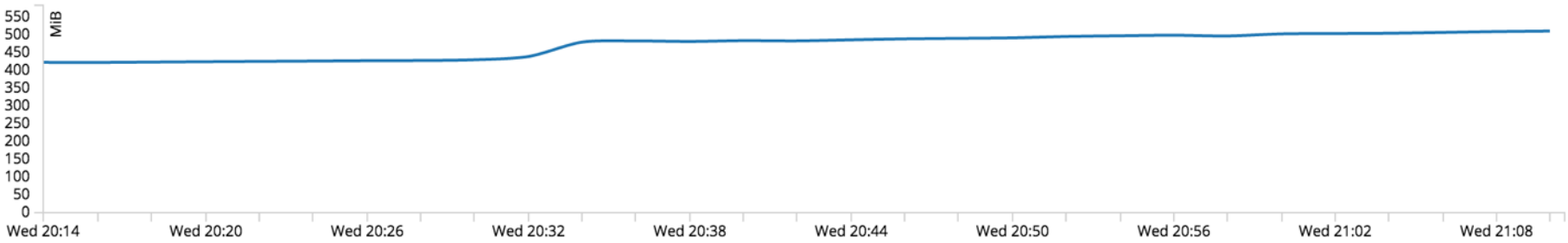
app aloha | deployment aloha-1 | deploymentconfig aloha | More labels...

Details | Environment | Metrics | Logs | Terminal | Events

Container: aloha **Time Range:** Last hour

About Compute Resources

Memory



aloha-1-4t9m2 created 4 hours ago

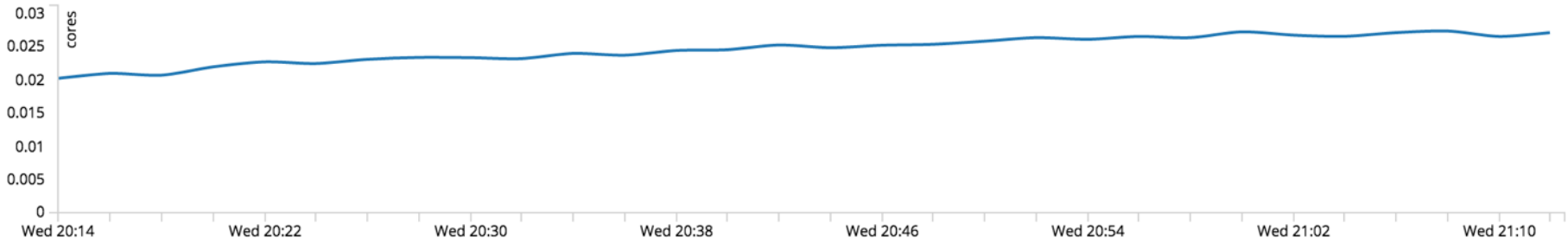
Actions

app aloha | deployment aloha-1 | deploymentconfig aloha | More labels...

Container: aloha **Time Range:** Last hour

About Compute Resources

CPU



aloha-1-4t9m2 created 4 hours ago

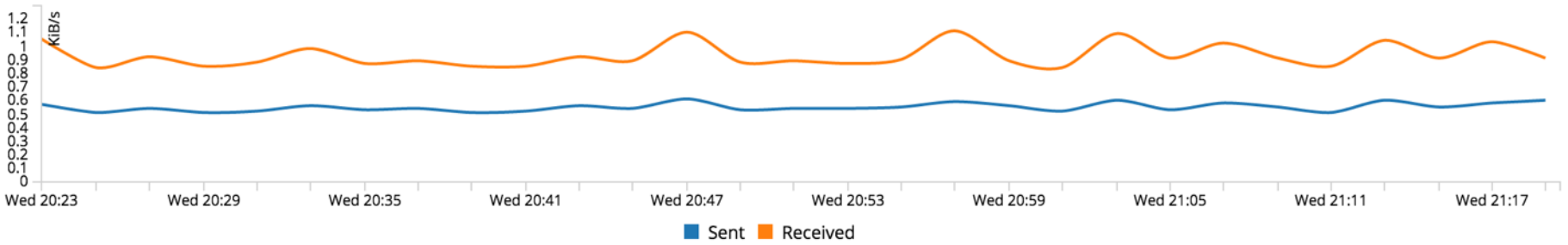
Actions

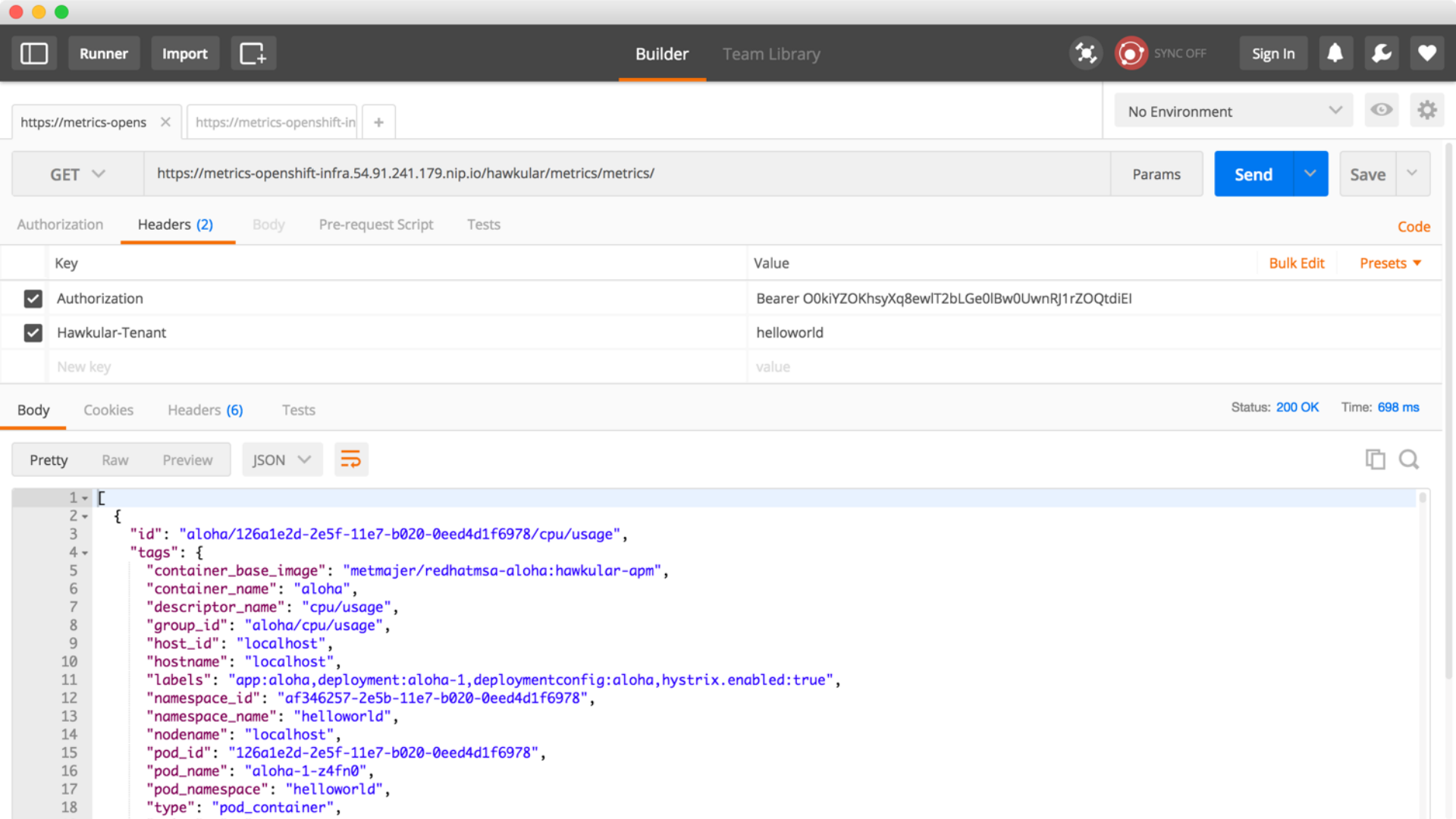
app aloha | deployment aloha-1 | deploymentconfig aloha | More labels...

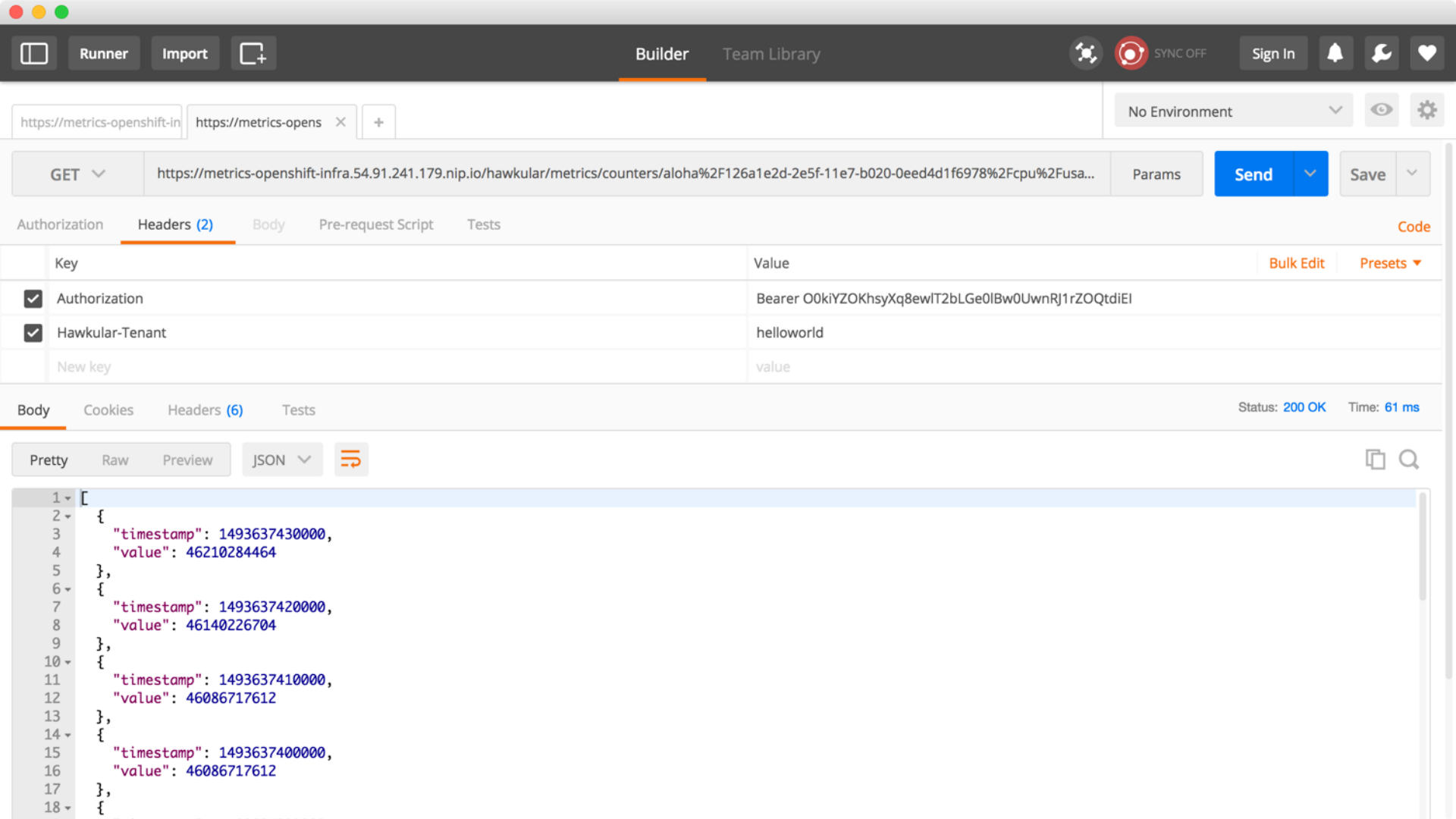
Container: aloha **Time Range:** Last hour

About Compute Resources

Network




Runner
Import
Builder
Team Library
SYNC OFF
Sign In
https://metrics-opens
https://metrics-openshift-in
No Environment
GET
https://metrics-openshift-infra.54.91.241.179.nip.io/hawkular/metrics/metrics/
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Key
Value
Bulk Edit
Presets
Authorization
Bearer O0kiYZOKhsyXq8ewlT2bLGe0lBw0UwnRJ1rZOQtdiEI
Hawkular-Tenant
helloworld
New key
value
Body
Cookies
Headers (6)
Tests
Status: 200 OK
Time: 698 ms
Pretty
Raw
Preview
JSON
[
{
"id": "aloha/126a1e2d-2e5f-11e7-b020-0eed4d1f6978/cpu/usage",
"tags": {
"container_base_image": "metmajer/redhatmsa-aloha:hawkular-apm",
"container_name": "aloha",
"descriptor_name": "cpu/usage",
"group_id": "aloha/cpu/usage",
"host_id": "localhost",
"hostname": "localhost",
"labels": "app:aloha,deployment:aloha-1,deploymentconfig:aloha,hystrix.enabled:true",
"namespace_id": "af346257-2e5b-11e7-b020-0eed4d1f6978",
"namespace_name": "helloworld",
"nodename": "localhost",
"pod_id": "126a1e2d-2e5f-11e7-b020-0eed4d1f6978",
"pod_name": "aloha-1-z4fn0",
"pod_namespace": "helloworld",
"type": "pod_container",


Runner
Import
Builder
Team Library
SYNC OFF
Sign In
https://metrics-openshift-in
https://metrics-opens
No Environment
GET
https://metrics-openshift-infra.54.91.241.179.nip.io/hawkular/metrics/counters/aloha%2F126a1e2d-2e5f-11e7-b020-0eed4d1f6978%2Fcpu%2Fusa...
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Key
Value
Bulk Edit
Presets
Authorization
Bearer O0kiYZOKhsyXq8ewlT2bLGe0lBw0UwnRJ1rZOQtdiEI
Hawkular-Tenant
helloworld
New key
value
Body
Cookies
Headers (6)
Tests
Status: 200 OK
Time: 61 ms
Pretty
Raw
Preview
JSON
[
{
"timestamp": 1493637430000,
"value": 46210284464
},
{
"timestamp": 1493637420000,
"value": 46140226704
},
{
"timestamp": 1493637410000,
"value": 46086717612
},
{
"timestamp": 1493637400000,
"value": 46086717612
},
{

Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*

Hawkular OpenShift Agent

Jolokia

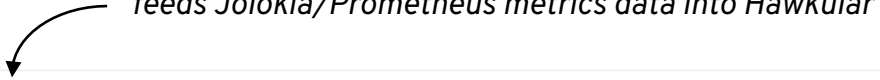
Prometheus

A Hawkular feed that collects metrics from Prometheus and/or Jolokia endpoints deployed in one or more pods within an OpenShift node.

README.adoc

Hawkular OpenShift Agent build passing

Introduction



Hawkular OpenShift Agent is a Hawkular feed implemented in the Go Programming Language. Its main purpose is to monitor a node within an OpenShift environment, collecting metrics from Prometheus and/or Jolokia endpoints deployed in one or more pods within the node. It can also be used to collect metrics from endpoints outside of OpenShift. The agent can be deployed inside the OpenShift node it is monitoring, or outside of OpenShift completely.

Watch this quick 10-minute demo to see the agent in action.

Note that the agent does not collect or store inventory at this time - this is strictly a metric collection and storage agent that integrates with Hawkular Metrics.

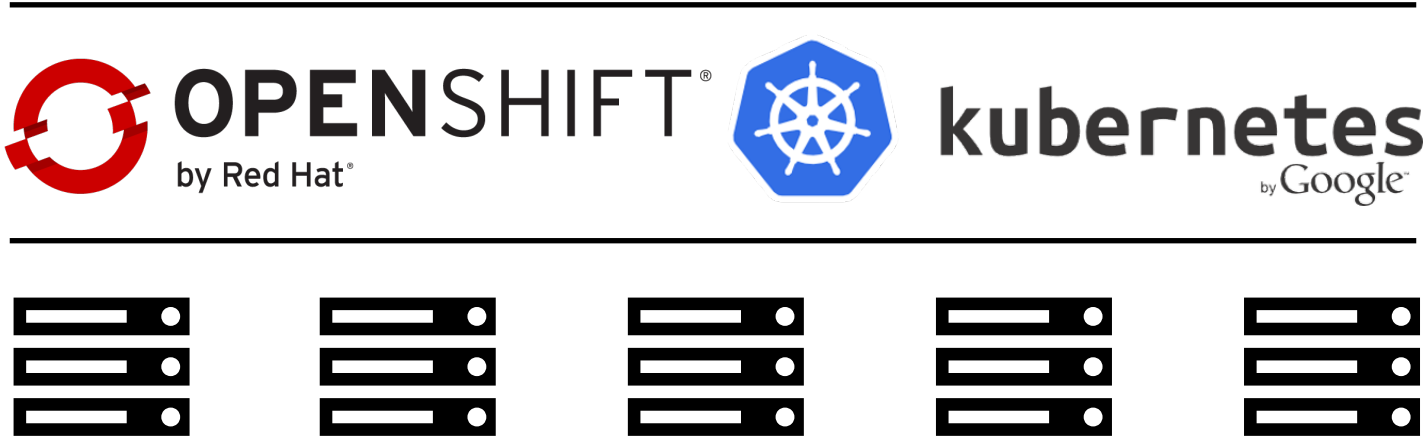
Docker Image

Hawkular OpenShift Agent is published as a docker image on Docker hub at hawkular/hawkular-openshift-agent

Monitoring

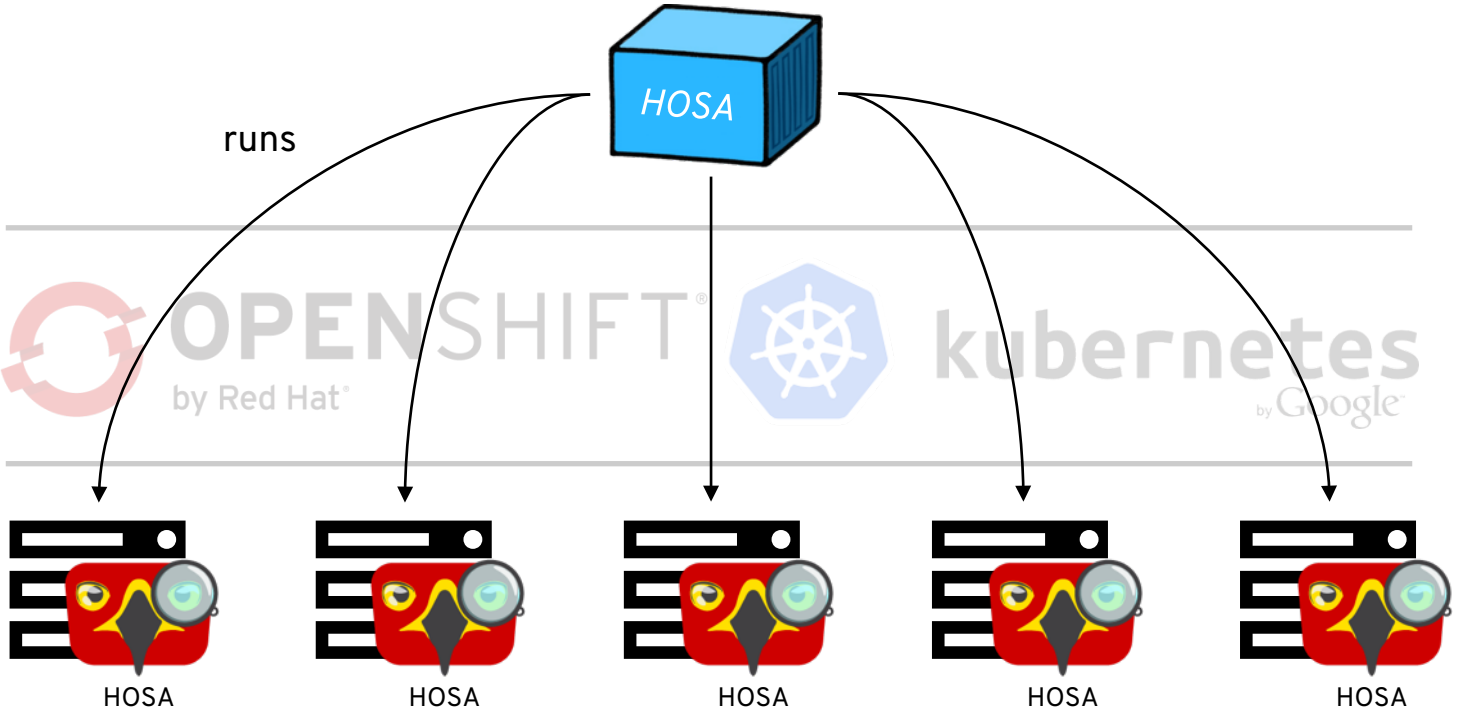
application level metrics with *Hawkular OpenShift Agent (HOSA)*

is deployed via a DaemonSet



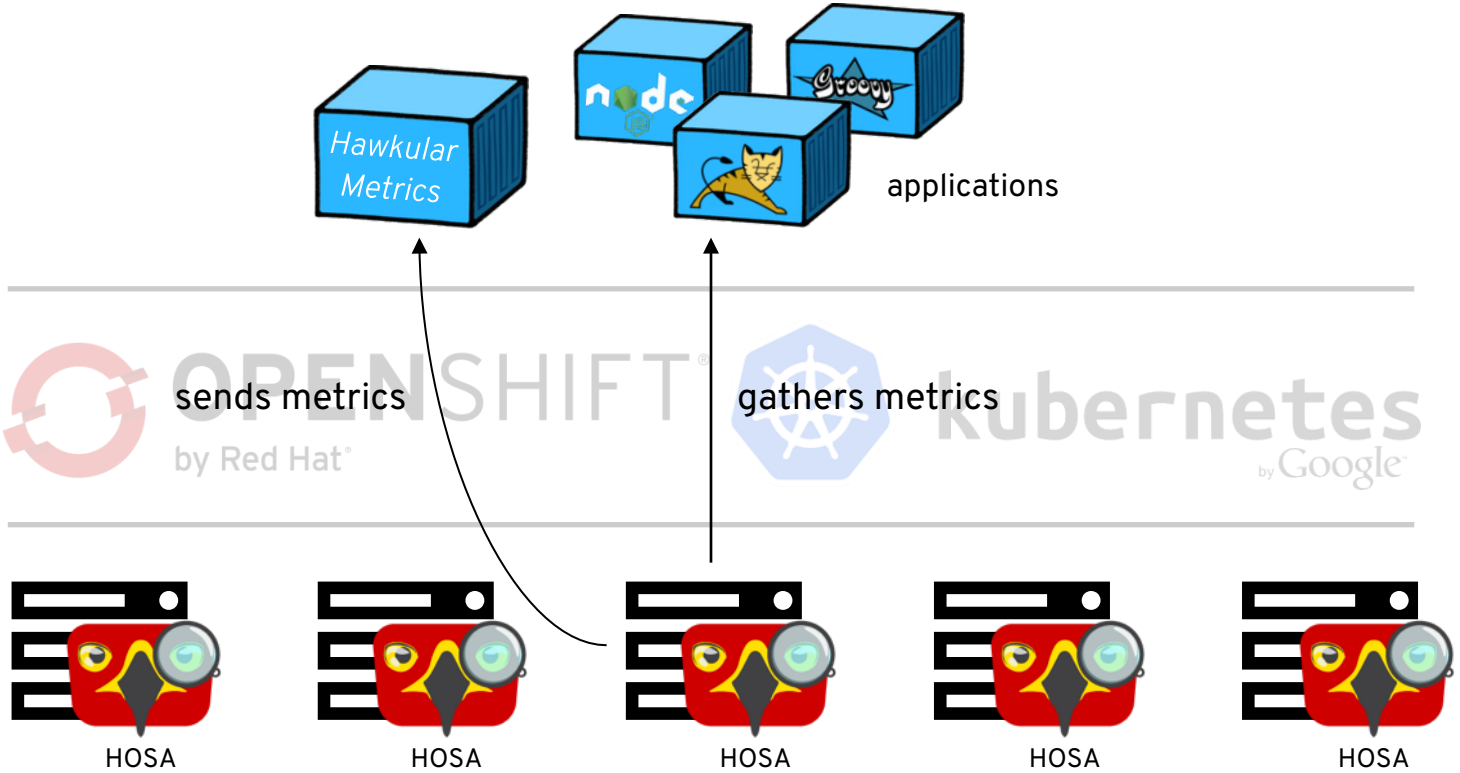
Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*



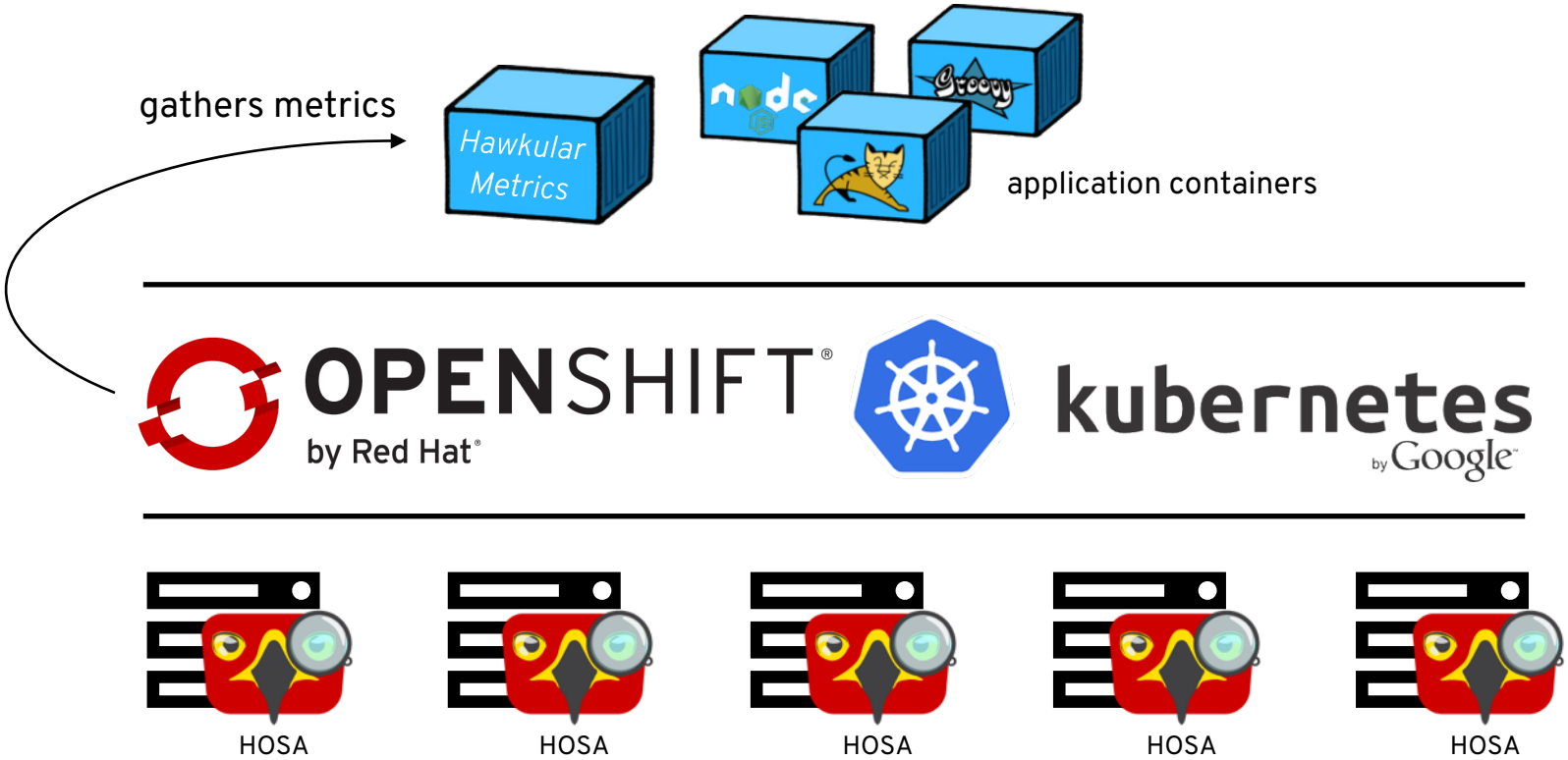
Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*



Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*



api-gateway-2-55plk created 4 hours ago

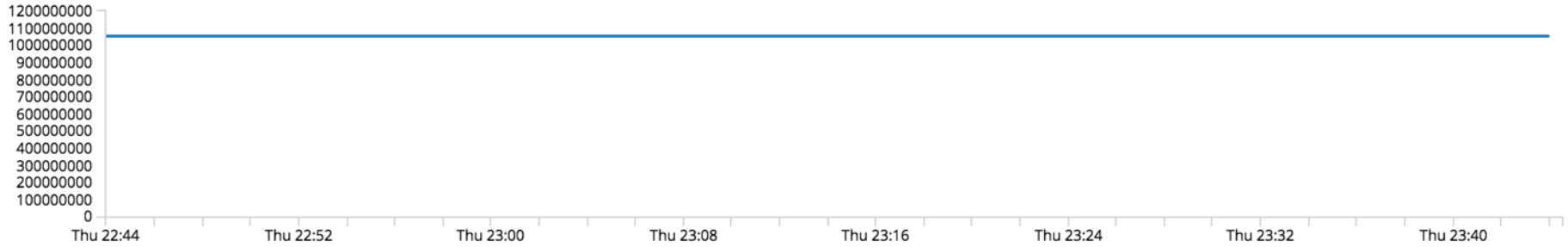
Actions

app api-gateway deployment api-gateway-2 deploymentconfig api-gateway More labels...

Container: api-gateway **Time Range:** Last hour

About Compute Resources

java.lang:type=Memory {1=HeapMemoryUsage,2=init}



Pods » api-gateway-2-55plk

api-gateway-2-55plk created 4 hours ago

Actions

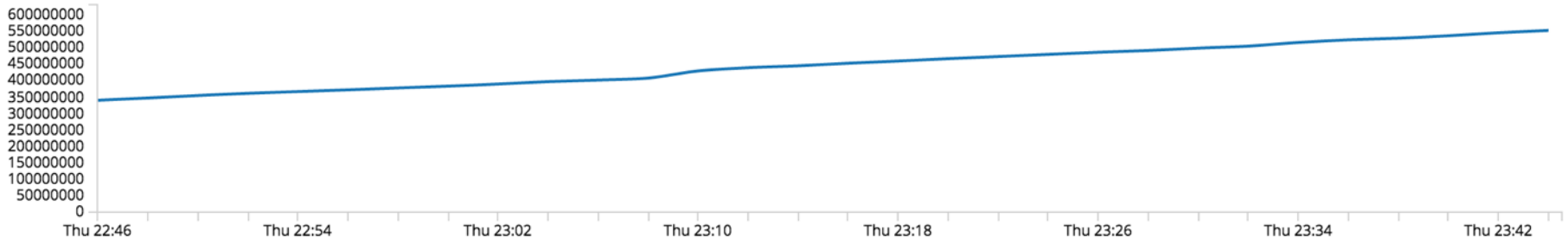
app api-gateway deployment api-gateway-2 deploymentconfig api-gateway More labels...

Details Environment Metrics Logs Terminal Events

Container: api-gateway **Time Range:** Last hour

About Compute Resources

java.lang:type=Memory {1=HeapMemoryUsage,2=used}



Pods » api-gateway-2-55plk

api-gateway-2-55plk created 4 hours ago

Actions

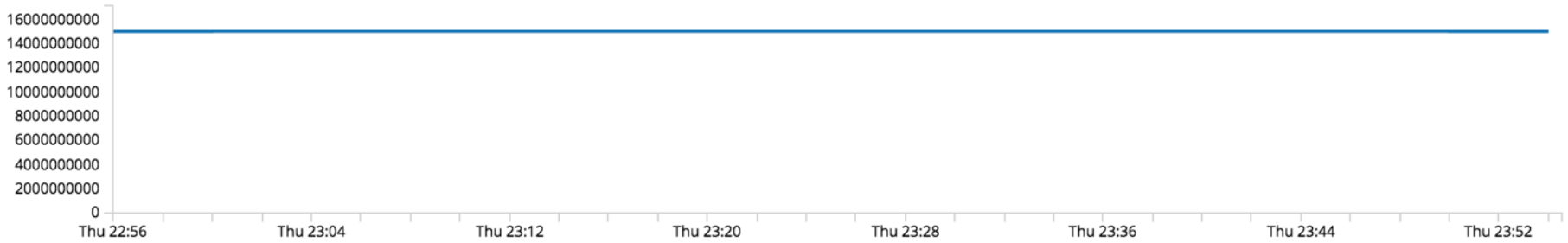
app api-gateway deployment api-gateway-2 deploymentconfig api-gateway More labels...

Details Environment Metrics Logs Terminal Events

Container: api-gateway **Time Range:** Last hour

About Compute Resources

java.lang:type=Memory {1=HeapMemoryUsage,2=max}



Pods » api-gateway-2-55plk

api-gateway-2-55plk created 4 hours ago

Actions

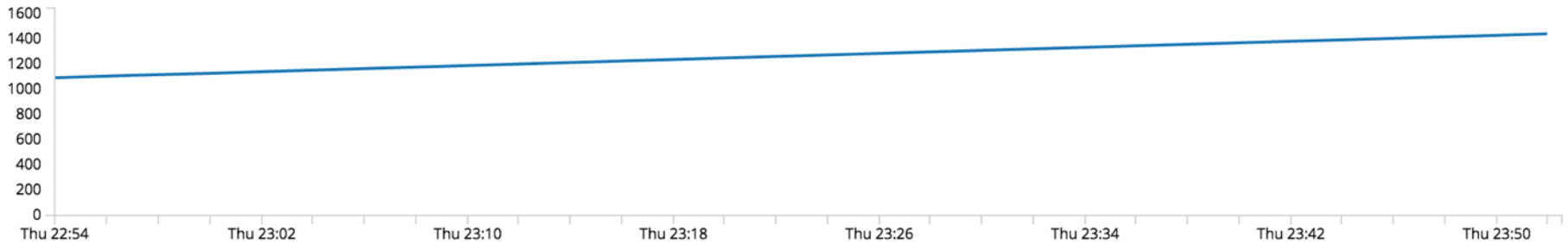
app api-gateway deployment api-gateway-2 deployment config api-gateway More labels...

Details Environment Metrics Logs Terminal Events

Container: api-gateway **Time Range:** Last hour

About Compute Resources

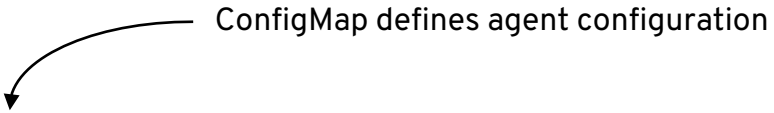
java.lang:type=Threading {1=ThreadCount}



Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*

How?




```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-hosa-config
data:
  hawkular-openshift-agent: |
    endpoints:
    - type: jolokia
      protocol: http
      port: 8778
      path: /jolokia/
      metrics
      - name: java.lang:type=Memory
        type: gauge
    - type: prometheus
      protocol: http
      port: 8080
      path: /metrics/
      metrics:
      - name: process_start_time_seconds
        type: gauge
```

Monitoring

Demo

application level metrics with *Hawkular OpenShift Agent (HOSA)*

How?

a volume named „*hawkular-openshift-agent*“ tells HOSA to monitor this Pod

```
    …
1   spec:
2     volumes:
3     - name: hawkular-openshift-agent
4       configMap:
5         name: my-hosa-config
    …
```

Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*

How?

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-hosa-config
data:
  hawkular-openshift-agent: |
    endpoints:
    - type: jolokia
      protocol: http
      port: 8778
      path: /jolokia/
      metrics
      - name: java.lang:type=Memory
        type: gauge
```

Jolokia ?

```
    - type: prometheus
      protocol: http
      port: 8080
      path: /metrics/
      metrics:
      - name: process_start_time_seconds
        type: gauge
```

Jolokia

JMX on Capsaicin

Jolokia

JMX on Capsaicin

installs a Jolokia WAR agent into Apache Tomcat

2. bash

```
$ mv $JOLOKIA_HOME/agents/jolokia.war $TOMCAT_HOME/webapps
$ $TOMCAT_HOME/bin/catalina.sh start
```

Monitoring

application level metrics with *Hawkular OpenShift Agent (HOSA)*

How?

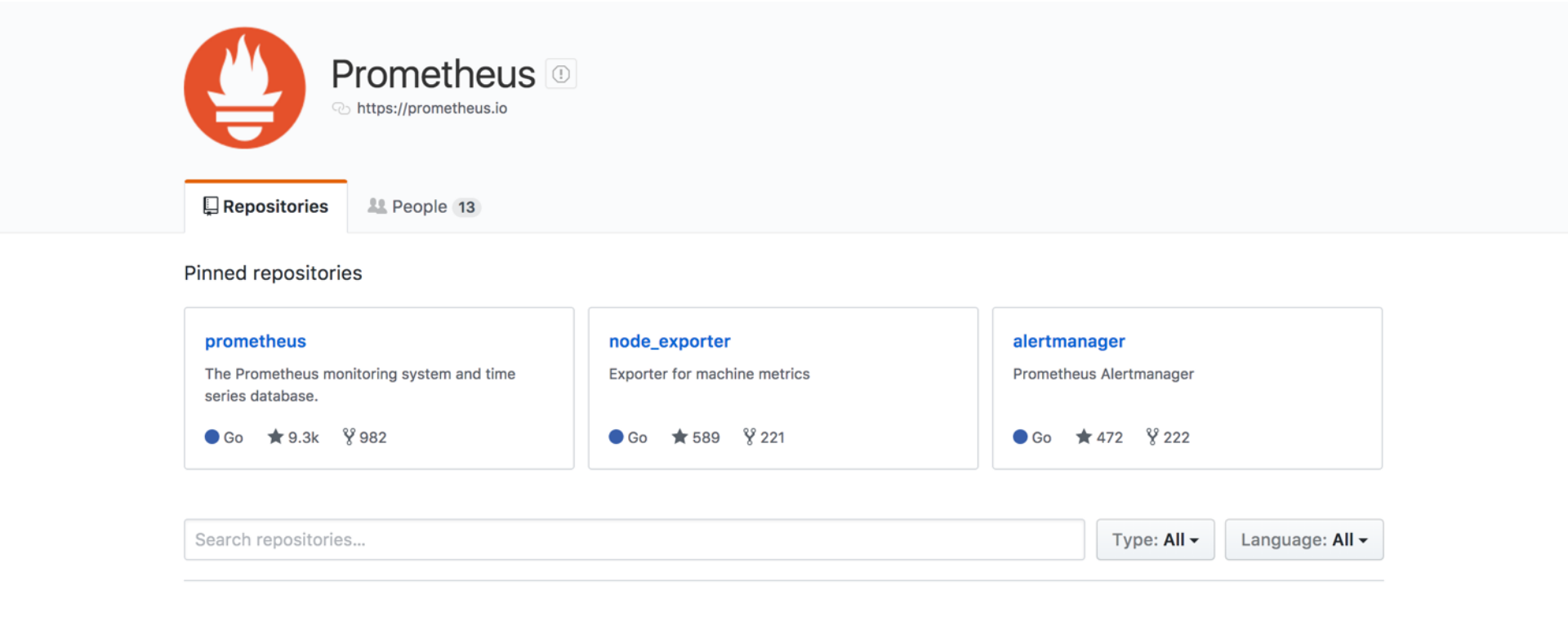
```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-hosa-config
data:
  hawkular-openshift-agent: |
    endpoints:
    - type: jolokia
      protocol: http
      port: 8778
      path: /jolokia/
      metrics
      - name: java.lang:type=Memory
        type: gauge
```

Prometheus ?

```
    - type: prometheus
      protocol: http
      port: 8080
      path: /metrics/
      metrics:
      - name: process_start_time_seconds
        type: gauge
```

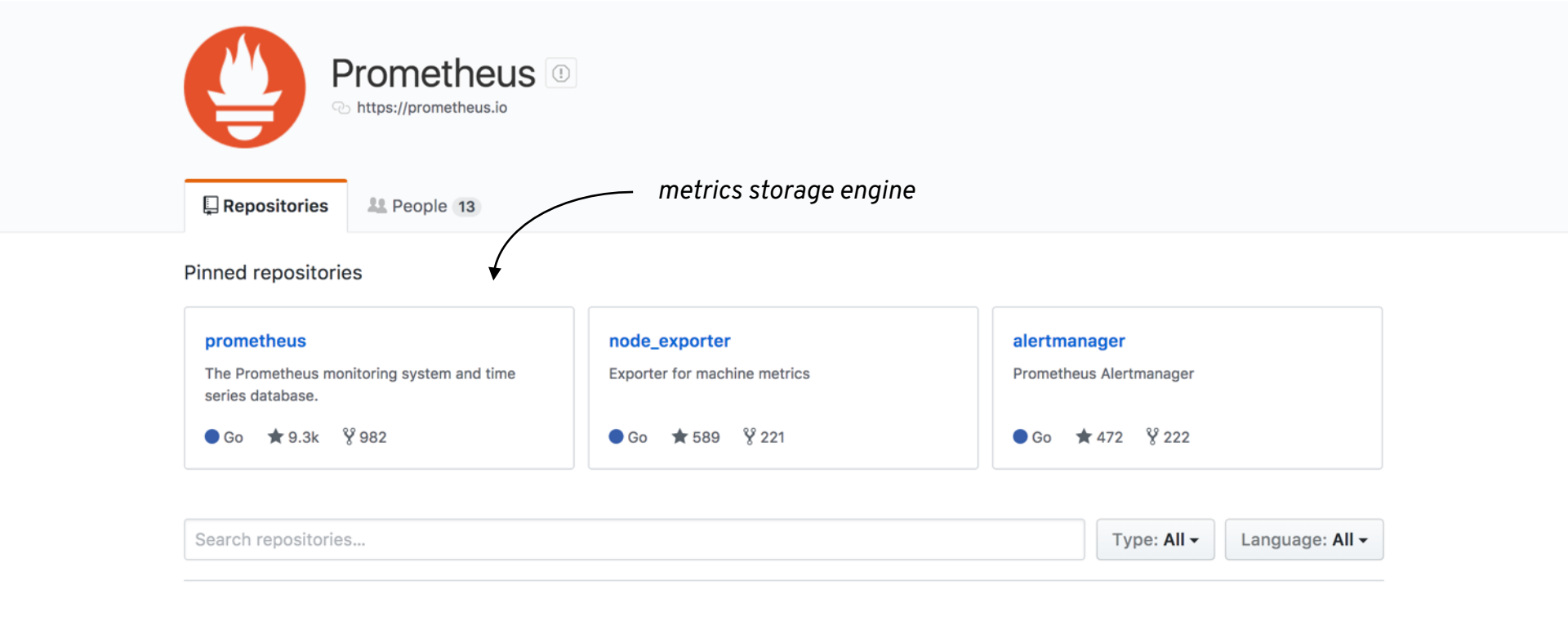
Prometheus

A monitoring system for metrics data



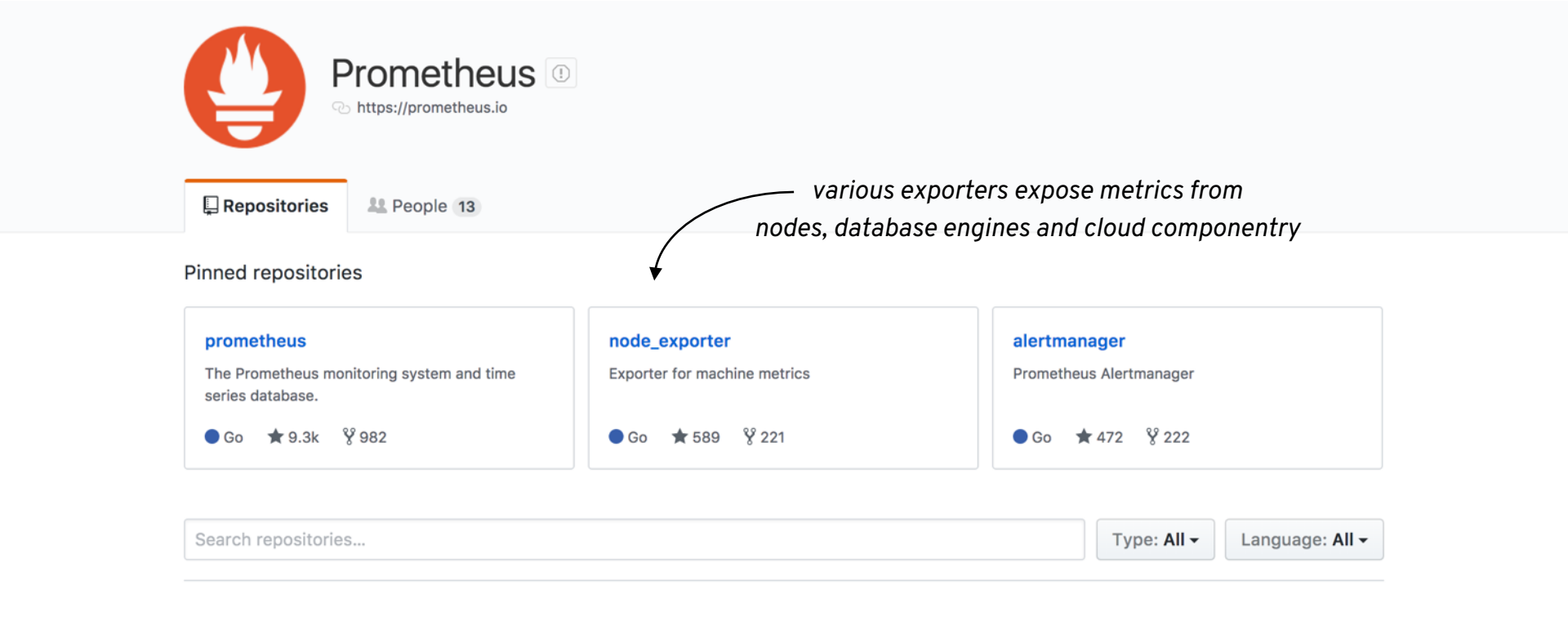
Prometheus

A monitoring system for metrics data



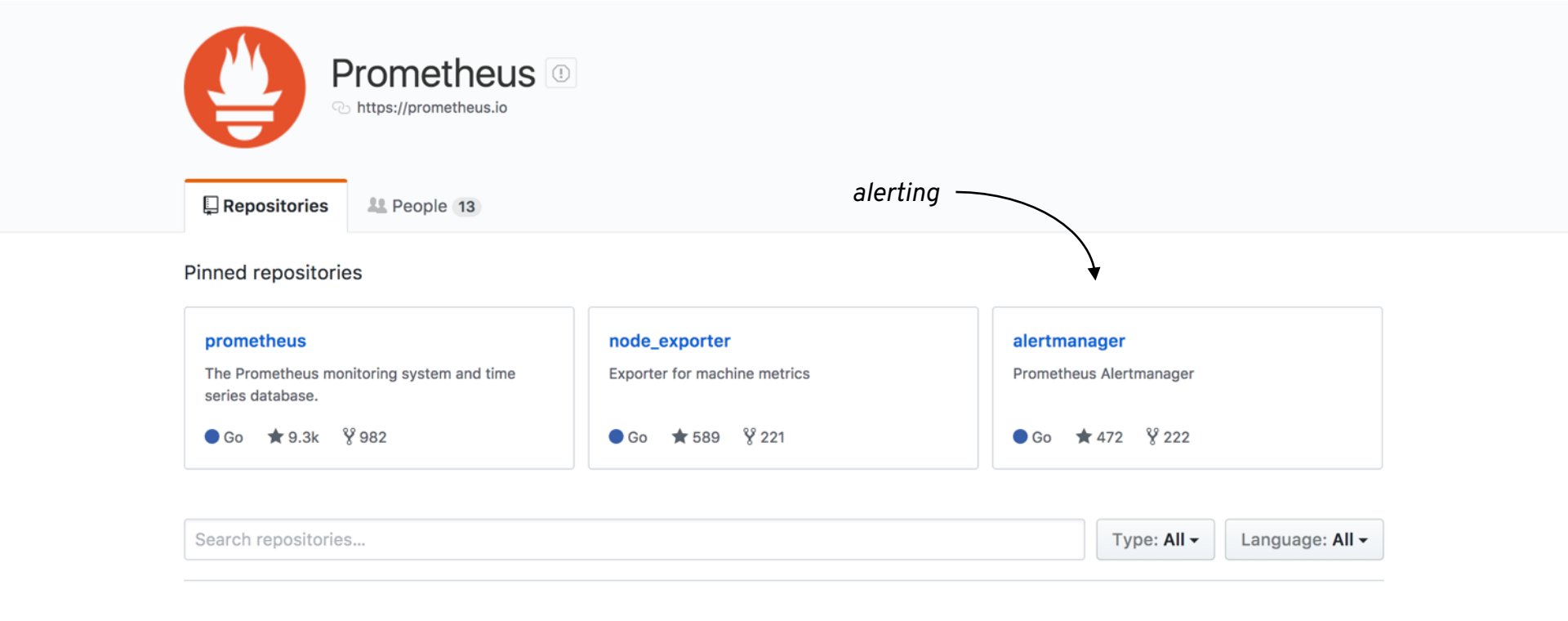
Prometheus

A monitoring system for metrics data



Prometheus

A monitoring system for metrics data



Prometheus

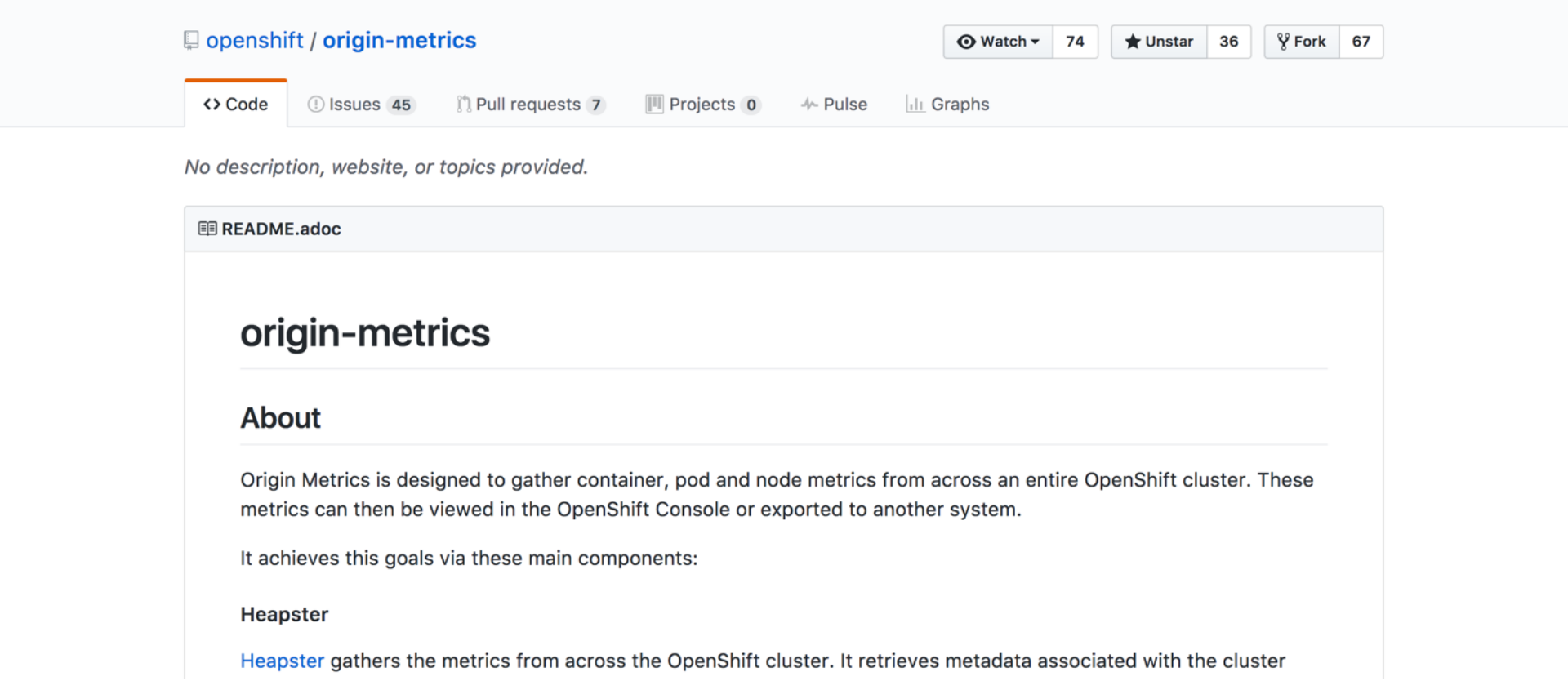
A monitoring system for metrics data

```
# build.gradle
dependencies {
  compile 'io.prometheus.simpleclient.0.0.21'
  compile 'io.prometheus.simpleclient_hotspot.0.0.21'
  compile 'io.prometheus.simpleclient_servlet.0.0.21'
}
```

```
# Application.java
public void registerMetricServlet() {
  // Expose Promtheus metrics.
 context.addServlet(new ServletHolder
    (new MetricsServlet()), "/metrics");
 // Add metrics about CPU, JVM memory etc.
 DefaultExports.initialize();
}
```

exposes default JMX metrics: CPU, Memory, Garbage Collection, etc.

Where to start?

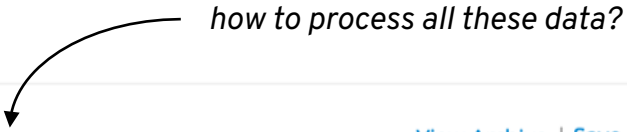


LOGGING

turbine-server-1-4cbb2 created a day ago

Actions

app turbine-server | deployment turbine-server-1 | deploymentconfig turbine-server



Details | Environment | Metrics | Logs | Terminal | Events

Container:turbine-server — Running Log from Apr 28, 2017 1:04:25 PM

View Archive | Save | Expand

Only the previous 5000 log lines and new log messages will be displayed because of the large log size.

Follow

```
1  2017-04-29 08:52:57.348  INFO [bootstrap,,,] 1 --- [       Timer-0] c.n.t.monitor.instance.InstanceMonitor   : Url for host:
   http://172.17.0.19:8080/hystrix.stream default
2  2017-04-29 08:52:57.353 ERROR [bootstrap,,,] 1 --- [InstanceMonitor] c.n.t.monitor.instance.InstanceMonitor   : Could not initiate
   connection to host, giving up: [<!DOCTYPE html>, <html lang="en">, <head>, <meta charset="utf-8">, <title>Error</title>, </head>,
   <body>, <pre>Cannot GET /hystrix.stream</pre>, </body>, </html>]
3  2017-04-29 08:52:57.353  WARN [bootstrap,,,] 1 --- [InstanceMonitor] c.n.t.monitor.instance.InstanceMonitor   : Stopping
   InstanceMonitor for: 172.17.0.19 default
4
5  com.netflix.turbine.monitor.instance.InstanceMonitor$MisconfiguredHostException: [<!DOCTYPE html>, <html lang="en">, <head>, <meta
   charset="utf-8">, <title>Error</title>, </head>, <body>, <pre>Cannot GET /hystrix.stream</pre>, </body>, </html>]
6          at com.netflix.turbine.monitor.instance.InstanceMonitor.init(InstanceMonitor.java:318) ~[turbine-core-1.0.0.jar!/:na]
7          at com.netflix.turbine.monitor.instance.InstanceMonitor.access$100(InstanceMonitor.java:103) ~[turbine-core-1.0.0.jar!/:na]
8          at com.netflix.turbine.monitor.instance.InstanceMonitor$2.call(InstanceMonitor.java:235) [turbine-core-1.0.0.jar!/:na]
9          at com.netflix.turbine.monitor.instance.InstanceMonitor$2.call(InstanceMonitor.java:229) [turbine-core-1.0.0.jar!/:na]
```

Logging

application events with *Elasticsearch*, *Fluentd* and *Kibana (EFK)*

Elasticsearch

Fluentd

Kibana

fluent / fluentd

Watch 305 | Star 5,196 | Fork 631

Code | Issues 63 | Pull requests 5 | Projects 0 | Wiki | Pulse | Graphs

Fluentd: Unified Logging Layer (project under CNCF) http://www.fluentd.org/

fluentd logging data-collector ruby log-collector cncf

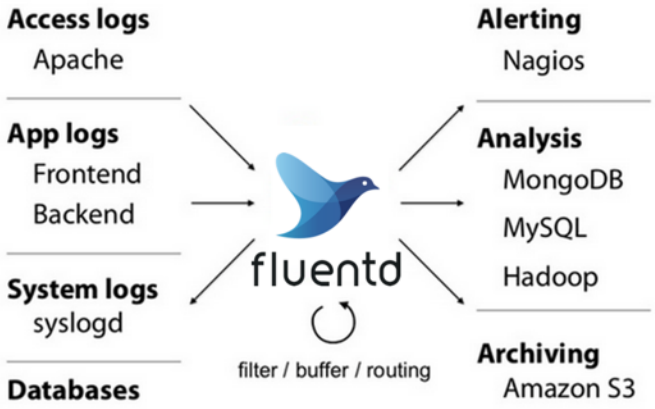
README.md

Fluentd: Open-Source Log Collector

build failing code climate 3.3

unifies your logging infrastructure; supports almost 200 plugins

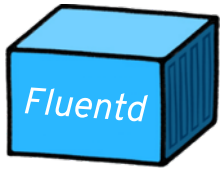
Fluentd collects events from various data sources and writes them to files, RDBMS, NoSQL, IaaS, SaaS, Hadoop and so on. Fluentd helps you unify your logging infrastructure (Learn more about the Unified Logging Layer).

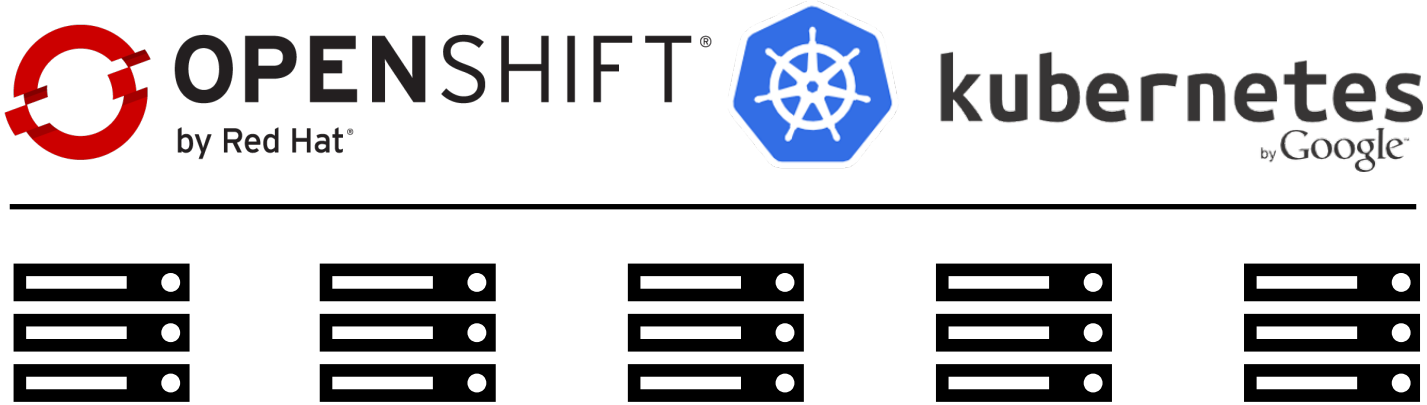


Logging

application events with *Elasticsearch*, *Fluentd* and *Kibana (EFK)*

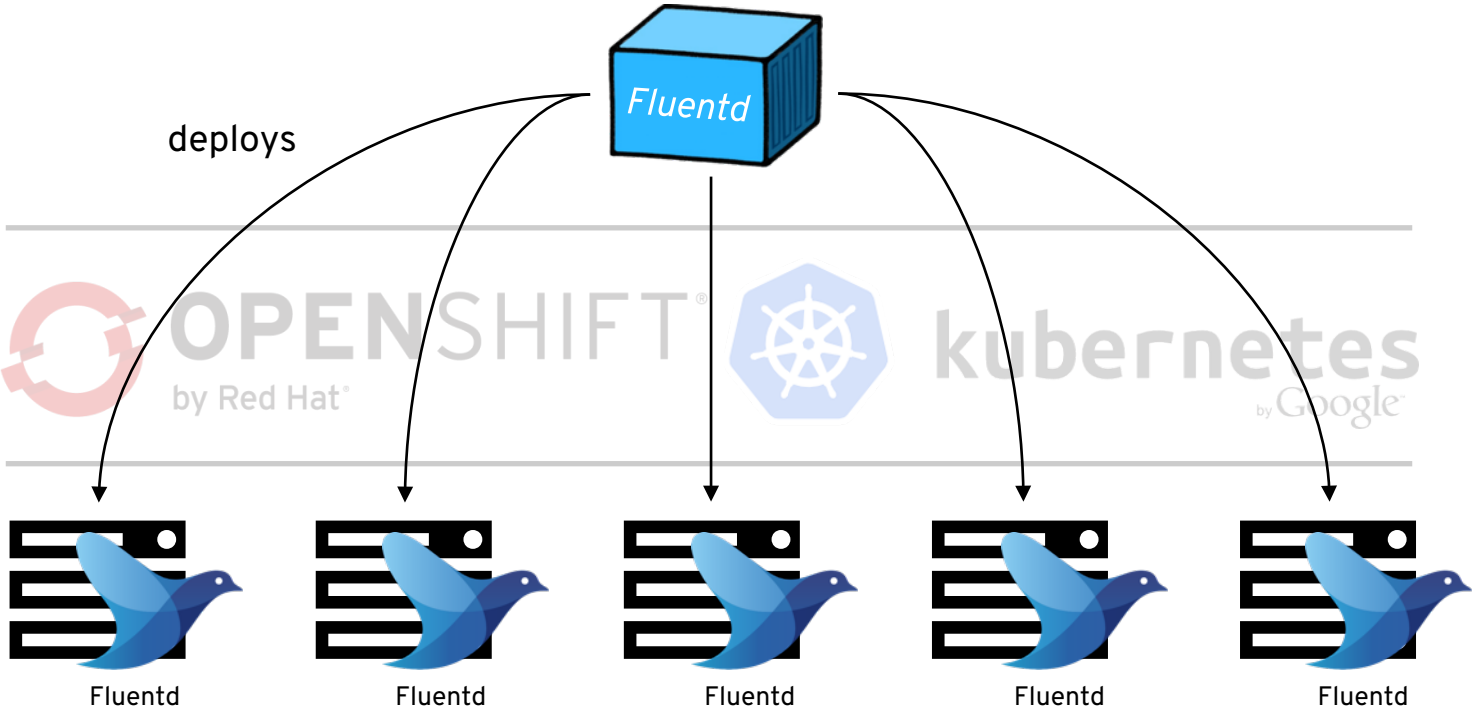
is deployed via a DaemonSet





Logging

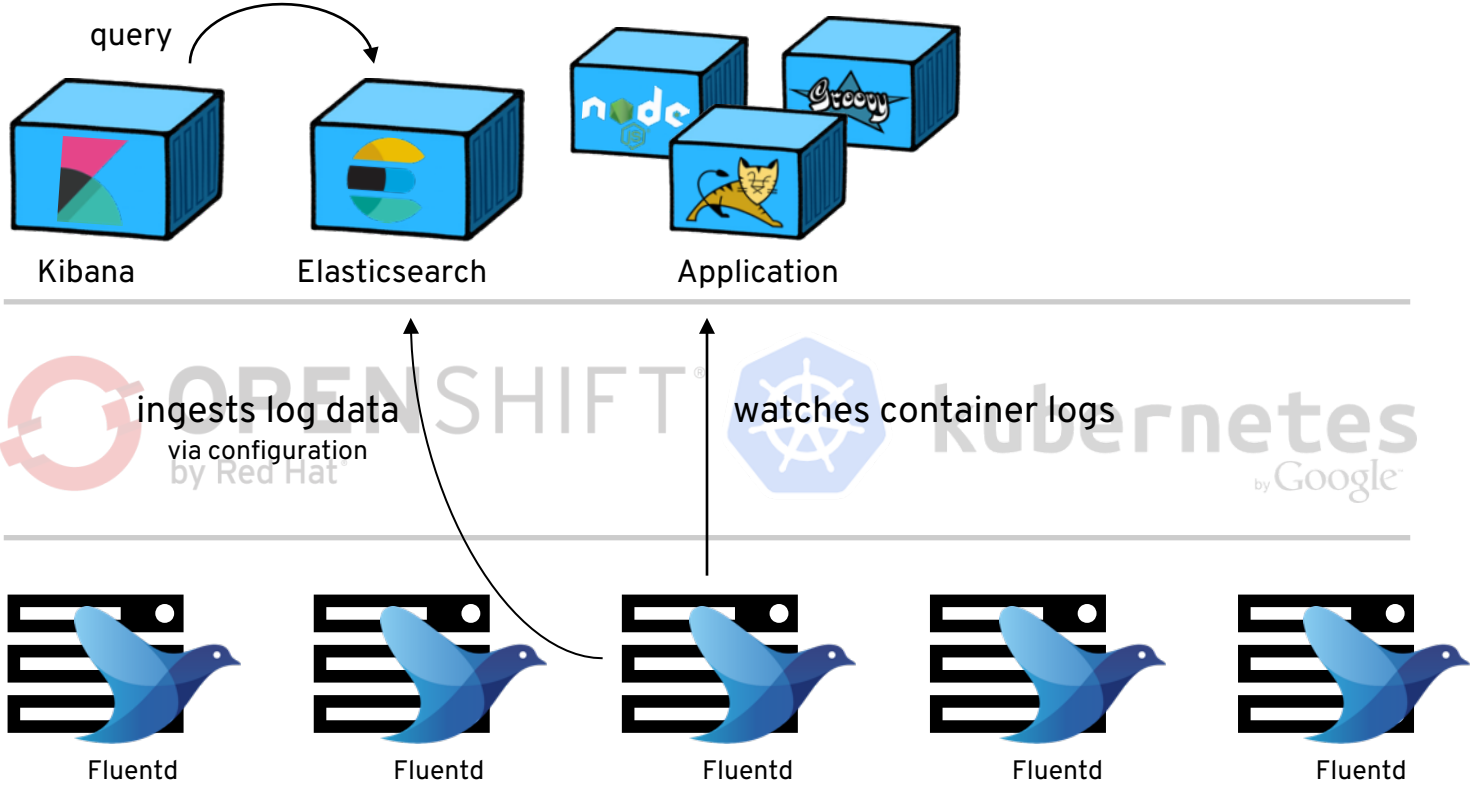
application events with *Elasticsearch*, *Fluentd* and *Kibana (EFK)*



Demo

Logging

application events with *Elasticsearch*, *Fluentd* and *Kibana (EFK)*



X-Pack

One Pack. Loads of Possibilities.

On its own, the Elastic Stack is a force to be reckoned with. X-Pack takes it to a new level by bundling powerful features into a single pack.

Security
(formerly Shield)

Alerting
(via Watcher)

Monitoring
(formerly Marvel)

Reporting

Graph

Machine Learning
(coming soon)

No description, website, or topics provided.

README.md

Origin-Aggregated-Logging

This repo contains the image definitions for the components of the logging stack as well as tools for building and deploying them. The logging subsystem consists of multiple components abbreviated as the "EFK" stack: Elasticsearch, Fluentd, Kibana.

The primary features this integration provides:

- Multitenant support to isolate logs from various project namespaces
- Openshift OAuth2 integration
- Historical log discovery and visualization
- Log aggregation of pod and node logs

Information to build the images from github source using an OpenShift Origin deployment is found here. To deploy the components from built or supplied images, see the deployer.

NOTE: If you are running OpenShift Origin using the All-In-One docker container method, you MUST add `-v /var/log:/var/log` to the `docker` command line. OpenShift must have access to the container logs in order for Fluentd to read and process them.

DISTRIBUTED TRACING

Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson,
Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag

Abstract

Modern Internet services are often implemented as complex, large-scale distributed systems. These applications are constructed from collections of software modules that may be developed by different teams, perhaps in different programming languages, and could span many thousands of machines across multiple physical facilities. Tools that aid in understanding system behavior and reasoning about performance issues are invaluable in such an environment.

Here we introduce the design of Dapper, Google's production distributed systems tracing infrastructure,

because large collections of small servers are a particularly cost-efficient platform for Internet services workloads [4]. Understanding system behavior in this context requires observing related activities across many different programs and machines.

A web-search example will illustrate some of the challenges such a system needs to address. A front-end service may distribute a web query to many hundreds of query servers, each searching within its own piece of the index. The query may also be sent to a number of other sub-systems that may process advertisements, check spelling, or look for specialized results, including images, videos, news, and so on. Results from all

design goals

impacted if even small parts of the system are not being monitored. In addition, monitoring should always be turned on, because it is often the case that unusual or otherwise noteworthy system behavior is difficult or impossible to reproduce. Three concrete design goals result from these requirements:

- *Low overhead*: the tracing system should have negligible performance impact on running services. In some highly optimized services even small monitoring overheads are easily noticeable, and might compel the deployment teams to turn the tracing system off.
- *Application-level transparency*: programmers should not need to be aware of the tracing system. A tracing infrastructure that relies on active collaboration from application-level developers in order to function becomes extremely fragile, and is often broken due to instrumentation bugs or omissions, therefore violating the ubiquity requirement. This is especially important in a fast-paced development environment such as ours.
- *Scalability*: it needs to handle the size of Google's services and clusters for at least the next few years.

An additional design goal is for tracing data to be available for analysis quickly after it is generated: ideally within a minute. Although a trace analysis system operating on hours-old data is still quite valuable, the availability of fresh information enables faster reaction to production anomalies.

True application-level transparency, possibly our most challenging design goal, was achieved by restricting Dapper's core tracing instrumentation to a small corpus

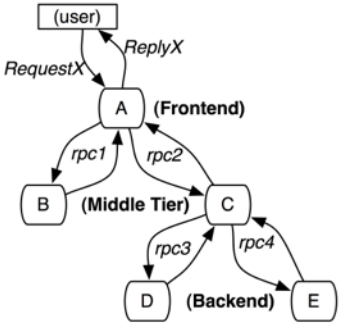


Figure 1: The path taken through a simple serving system on behalf of user request *X*. The letter-labeled nodes represent processes in a distributed system.

point in their development, before there is an opportunity to clearly evaluate important design choices. Since Dapper has been in production and operating at large scale for years now, we decided it would be most appropriate to focus this paper on what Dapper's deployment has taught us, how our design decisions played out, and in what ways it has been most useful. The value of Dapper as a platform for development of performance analysis tools, as much as a monitoring tool in itself, is one of a few unexpected outcomes we can identify in a retrospective assessment.

Although Dapper shares many of its high-level ideas with systems such as Pinpoint and Magpie, our implementation contains a number of new contributions in this space. For example, we have found sampling to be nec-

Google Dapper

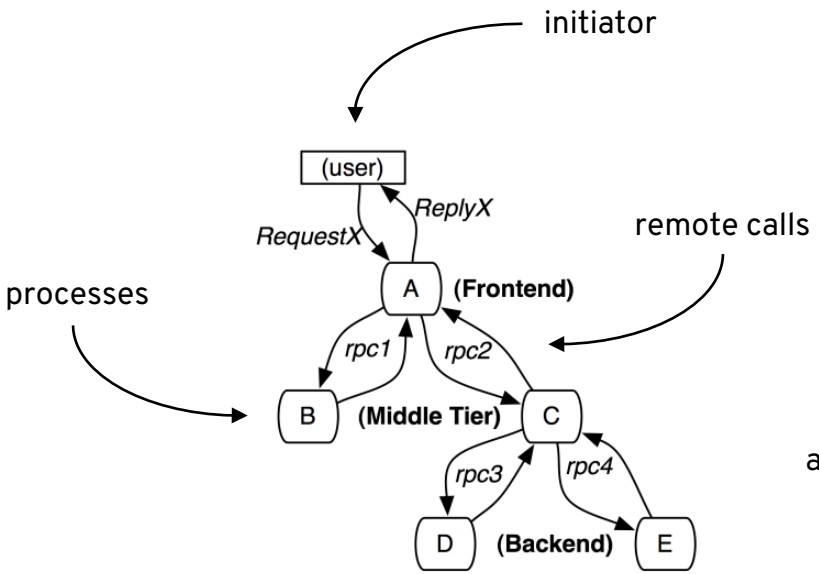


Figure 1: The path taken through a simple serving system on behalf of user request *X*. The letter-labeled nodes represent processes in a distributed system.

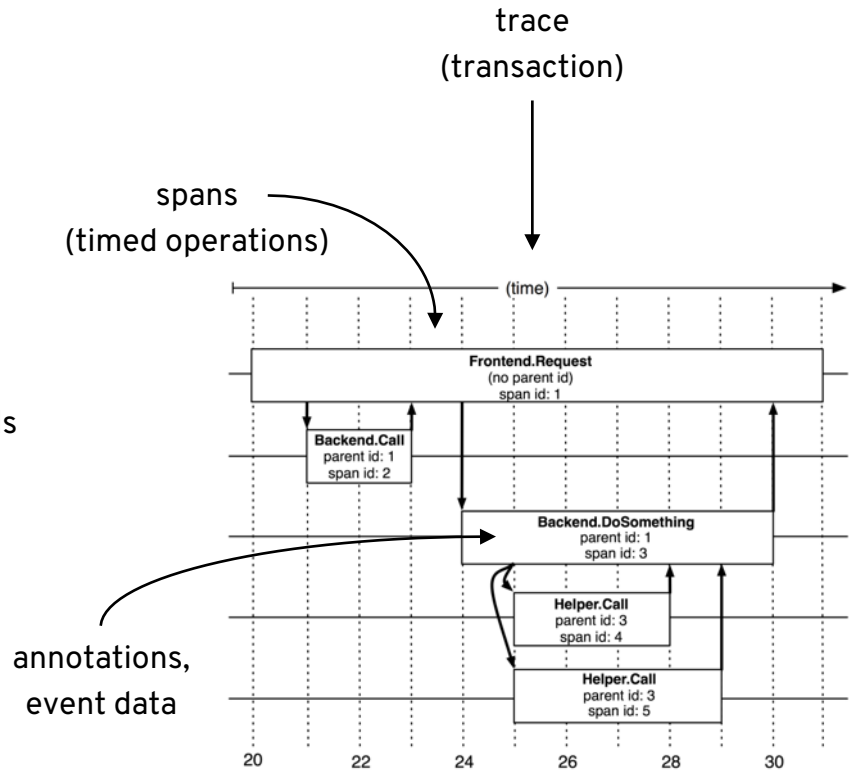
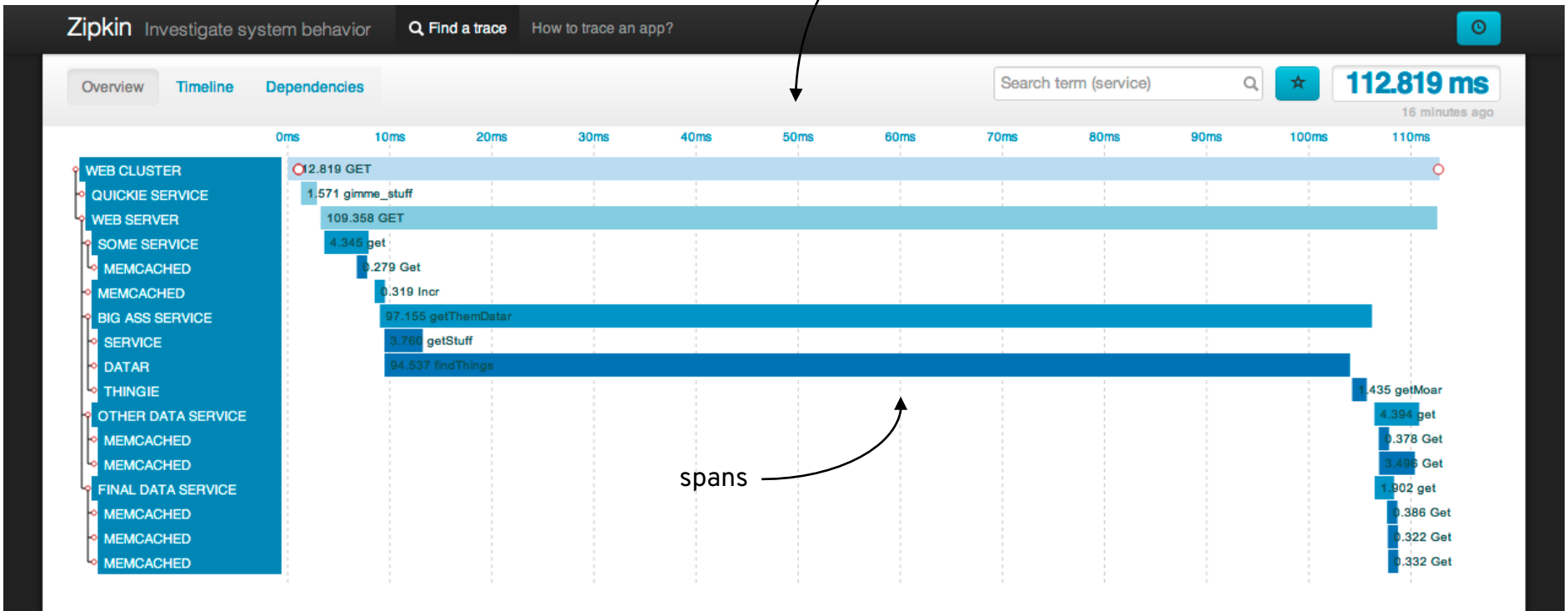


Figure 2: The causal and temporal relationships between five spans in a Dapper trace tree.

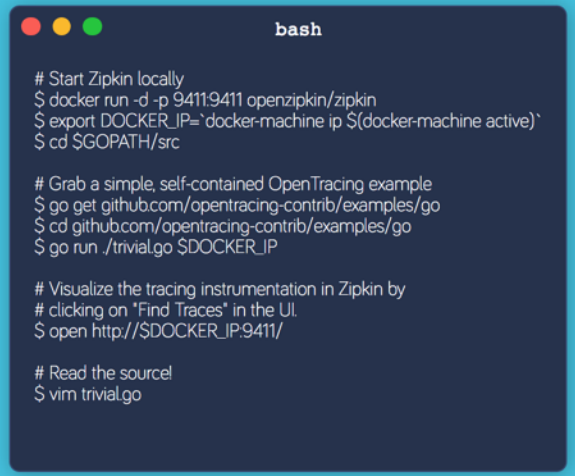
Source: Google Dapper Paper

ZipKin

A distributed tracing system



OPENTRACING DOCS GITHUB COMMUNITY GETTING STARTED BLOG -> JOIN THE MAILING LIST



a standard API for reporting distributed traces

A vendor-neutral open standard for distributed tracing.

Libraries available in 6 languages

Go, JavaScript, Java, Python, Objective-C, C++

with various client libraries

Supported Tracers

ZIPKIN LIGHTSTEP appdash TRACER

Supported Frameworks

GRPC Flask Go kit django

OpenTracing

A vendor-neutral open standard for distributed tracing

The problem?

- instrumentation is challenging: tracing context must propagate within and between processes
- unreasonable to *ask all vendor and FOSS software* to be intrumented by a single tracing vendor

The solution?

- allow developers to instrument their own code using OpenTracing client libraries
- have the application maintainer choose the tracing technology via configuration change

Hawkular APM

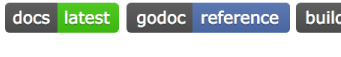
Distributed Tracing and Application Performance Management

The members of the Hawkular APM team have been contributing to the OpenTracing standard, getting involved in discussions around the API, as well as contributing Java framework integrations and a Java Agent. In April 2017, the team also began actively contributing to the Jaeger project, as this project provides a more "OpenTracing" native infrastructure. Further information can be found in this blog. This means that the focus on the Hawkular APM project has been de-emphasized in favour of Jaeger.

Jaeger, a Distributed Tracing System http://uber.github.io/jaeger/

distributed-tracing opentracing

README.md



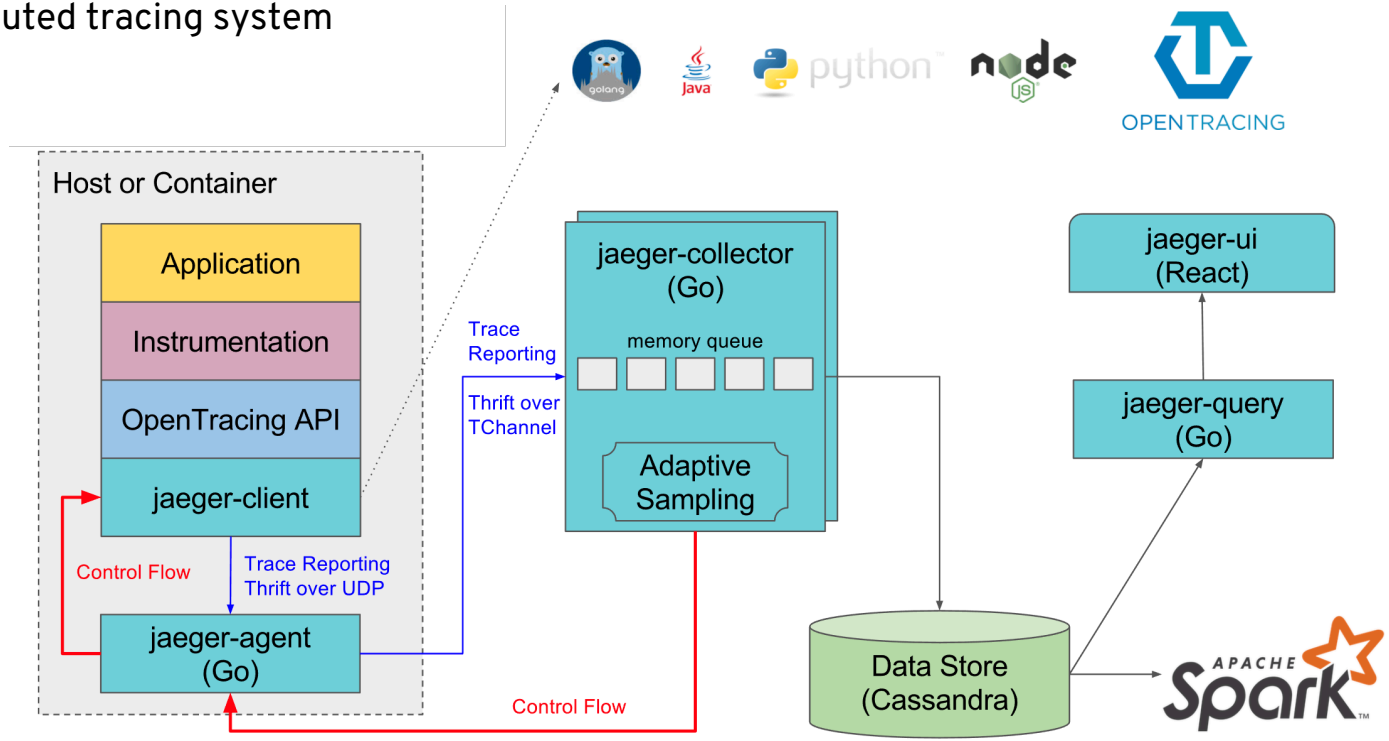
Jaeger - a Distributed Tracing System

Jaeger, inspired by Dapper and OpenZipkin, is a distributed tracing system released as open source by Uber Technologies. It can be used for monitoring microservice-based architectures:

- Distributed context propagation
- Distributed transaction monitoring
- Root cause analysis
- Service dependency analysis
- Performance / latency optimization

Jaeger

A distributed tracing system



Source: Jaeger Architecture – jaeger.readthedocs.io

SOLVED THE CHALLENGE?

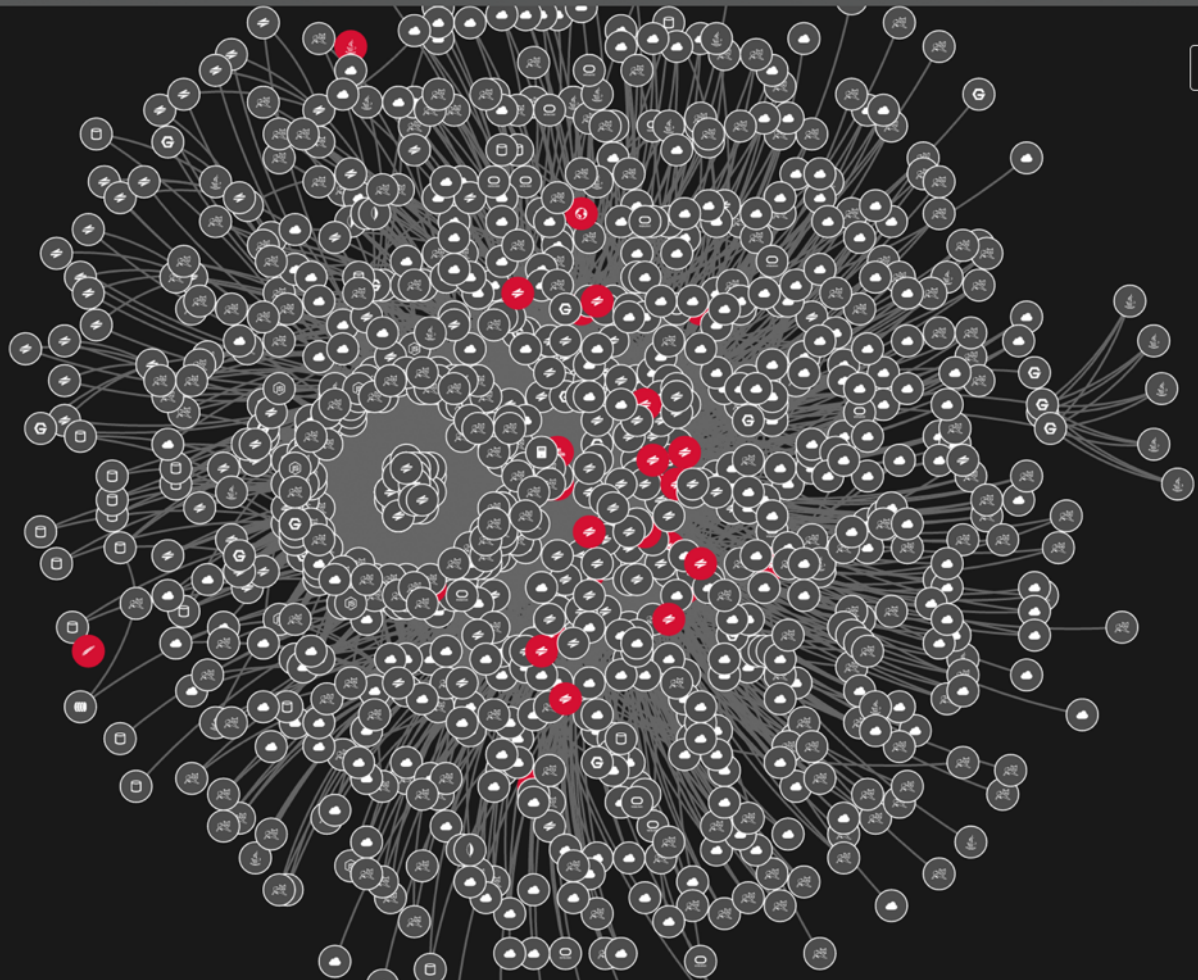
Applications
9

Services
30/3304

Processes
3/10424

Hosts
142

Datacenters
7



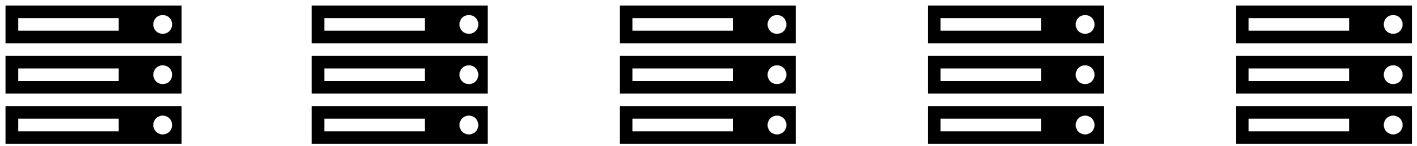
Dynatrace

- [x] All-in-one monitoring, now!

dynatrace/oneagent

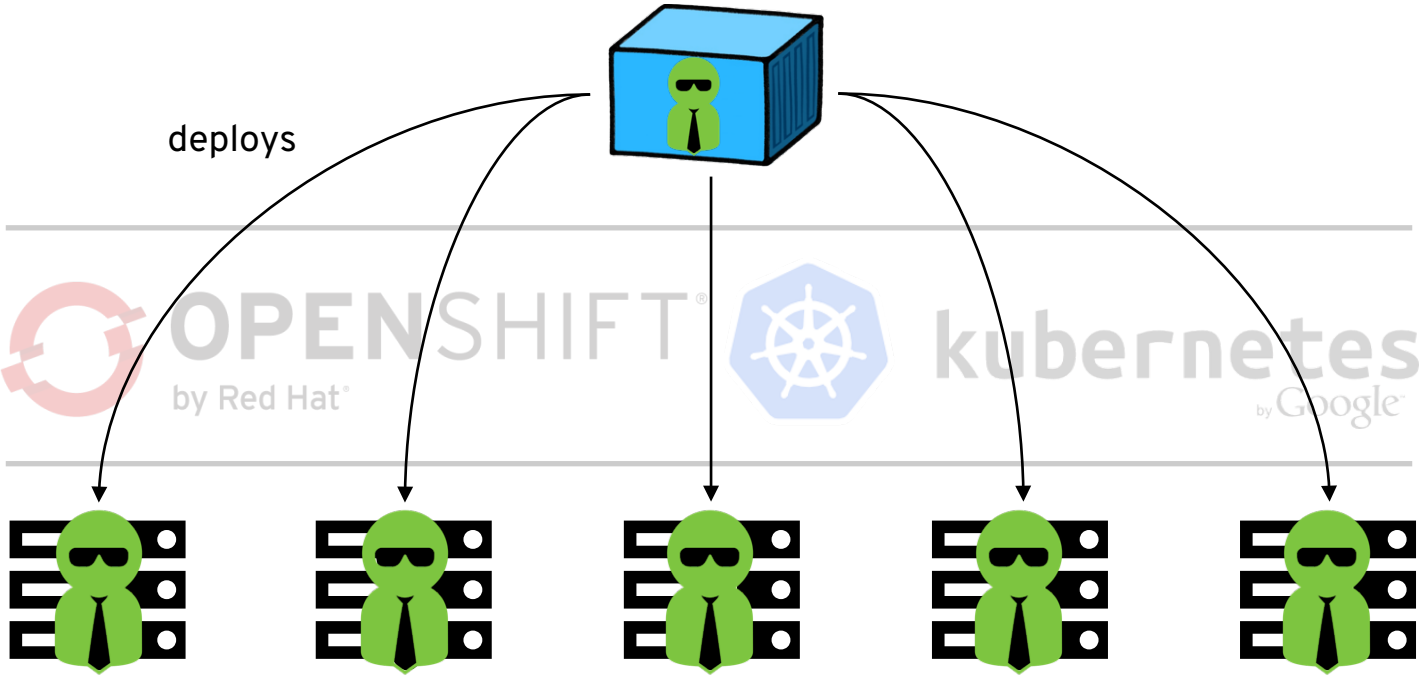
OPENSHIFT® by Red Hat®

kubernetes by Google



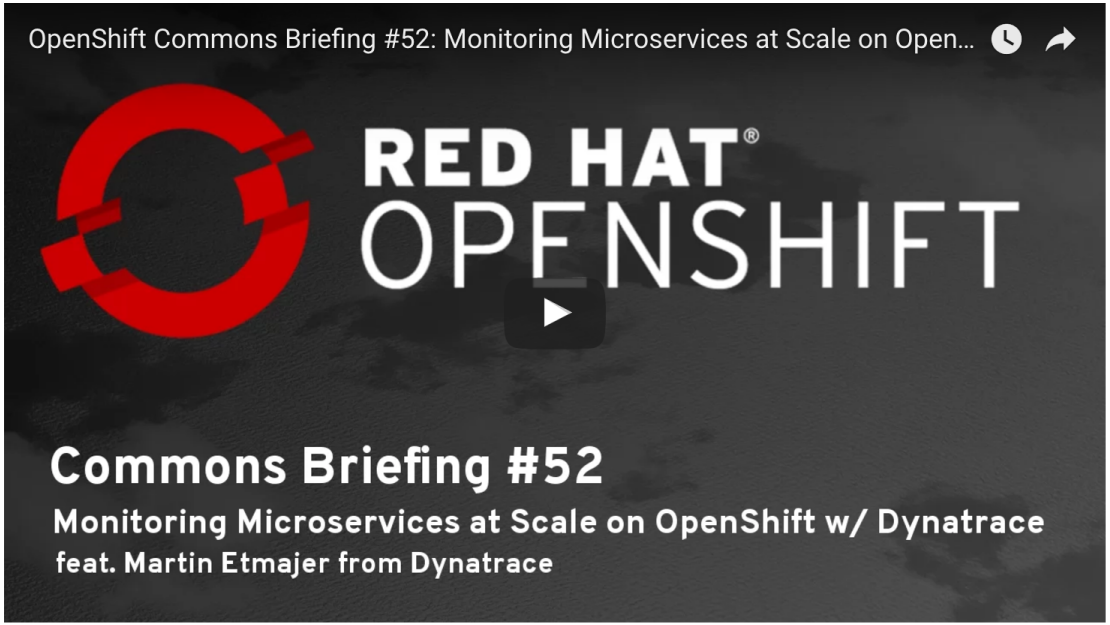
Dynatrace

- [x] All-in-one monitoring, now!



OpenShift Commons Briefing #52: Monitoring Microservices at Scale on OpenShift with Dynatrace

OCTOBER 24, 2016 BY DIANE MUELLER



Search the blog

CATEGORIES

- Educators (19)
- Events (410)
- News (556)
- OpenShift Ecosystem (93)
- Products (855)
  - OpenShift Container Platform (342)
  - OpenShift Dedicated (118)
  - OpenShift Online (545)
  - OpenShift Origin (386)
- Technologies (468)
  - .NET (12)
  - Java (132)
  - JBoss (77)
  - Jenkins (29)
  - MongoDB (85)
  - MySQL (37)
  - Node.js (75)
  - Perl (6)
  - PHP (70)

Author Archives: The Dynatrace Team

OpenShift Ecosystem: Monitoring OpenShift Apps with Dynatrace (Part 2)

OCTOBER 17, 2016 BY THE DYNATRACE TEAM

In this article, we'll explore various alternatives to monitoring your OpenShift applications when installing Dynatrace OneAgent on your cluster nodes isn't an option. This situation arises when using managed cloud offerings, such as OpenShift Dedicated or the OpenShift Online Developer Preview.

Read More...

OpenShift Ecosystem: Monitoring OpenShift Apps with Dynatrace

AUGUST 1, 2016 BY THE DYNATRACE TEAM

Dynatrace is an all-in-one, zero-config monitoring platform, powered by artificial intelligence that identifies performance problems and pinpoints their root causes in seconds.

Read More...

Search the blog

CATEGORIES

- Educators (19)
- Events (410)
- News (556)
- OpenShift Ecosystem (93)
- Products (855)
  - OpenShift Container Platform (342)
  - OpenShift Dedicated (118)
  - OpenShift Online (545)
  - OpenShift Origin (386)
- Technologies (468)
  - .NET (12)
  - Java (132)
  - JBoss (77)
  - Jenkins (29)
  - MongoDB (85)
  - MySQL (37)
  - Node.js (75)
  - Perl (6)
  - PHP (70)

RED HAT SUMMIT

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos

LEARN. NETWORK.
EXPERIENCE
OPEN SOURCE.