**Team Assignment**

**Prometheus & Grafana**

**Scope:**

*Container performance and who is doing what? YML file, Grafana Prometheus, load, performance and everything can be done using Grafana & Prometheus; pulling it together through Docker Compose for observability*

## What is Prometheus?

[Prometheus](https://github.com/prometheus) is an **open-source systems monitoring and alerting toolkit** originally built at [SoundCloud](https://soundcloud.com/). Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user [community](https://prometheus.io/community). It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the [Cloud Native Computing Foundation](https://cncf.io/) in 2016 as the second hosted project, after [Kubernetes](https://kubernetes.io/).

### Features

Prometheus's main features are:

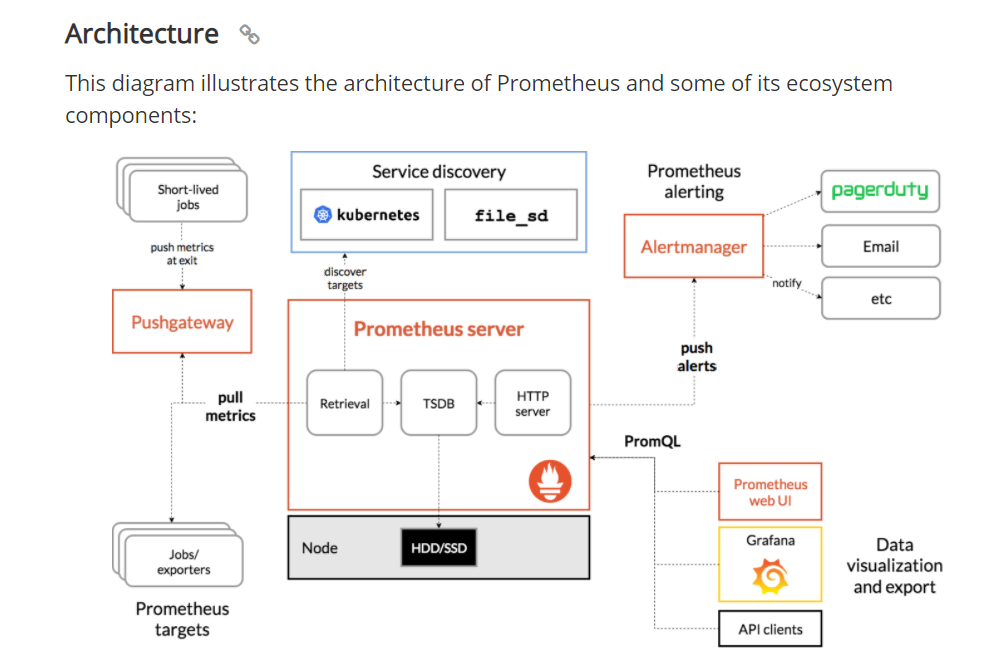
* a multi-dimensional [data model](https://prometheus.io/docs/concepts/data_model/) with time series data identified by metric name and key/value pairs
* PromQL, a [flexible query language](https://prometheus.io/docs/prometheus/latest/querying/basics/) to leverage this dimensionality
* no reliance on distributed storage; single server nodes are autonomous
* time series collection happens via a pull model over HTTP
* [pushing time series](https://prometheus.io/docs/instrumenting/pushing/) is supported via an intermediary gateway
* targets are discovered via service discovery or static configuration
* multiple modes of graphing and dashboarding support

### Components

The Prometheus ecosystem consists of multiple components, many of which are optional:

* the main [Prometheus server](https://github.com/prometheus/prometheus) which scrapes and stores time series data
* [client libraries](https://prometheus.io/docs/instrumenting/clientlibs/) for instrumenting application code
* a [push gateway](https://github.com/prometheus/pushgateway) for supporting short-lived jobs
* special-purpose [exporters](https://prometheus.io/docs/instrumenting/exporters/) for services like HAProxy, StatsD, Graphite, etc.
* an [alertmanager](https://github.com/prometheus/alertmanager) to handle alerts
* various support tools

Most Prometheus components are written in [Go](https://golang.org/), making them easy to build and deploy as static binaries.

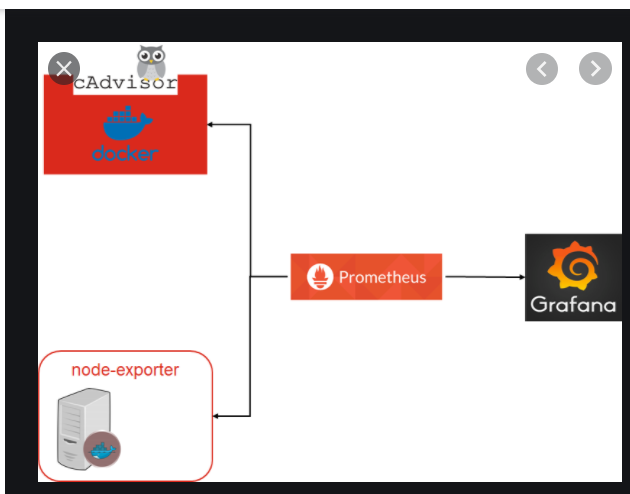


Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. [Grafana](https://grafana.com/) or other API consumers can be used to visualize the collected data.

Prometheus includes a local on-disk **time series database**, but also optionally integrates with remote storage systems

* **The Prometheus Node Exporter exposes a wide variety of hardware- and kernel-related metrics (Linux host related)**
* [**cAdvisor**](https://github.com/google/cadvisor) (short for **c**ontainer **Advisor**) analyzes and exposes resource usage and **performance data from running containers**
* The **Alertmanager** handles alerts sent by client applications such as the **Prometheus** server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts
* The **Pushgateway** is an intermediary service which allows you to push metrics from jobs which cannot be scraped

**Container specific monitoring – Node Exporter & cAdvisor**

* ****

## When does it fit?

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength.

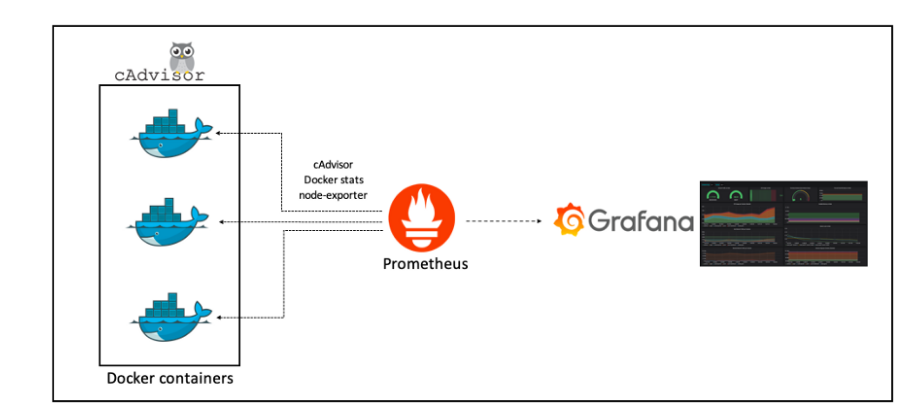
Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems. Each Prometheus server is standalone, not depending on network storage or other remote services. You can rely on it when other parts of your infrastructure are broken, and you do not need to setup extensive infrastructure to use it.

## When does it not fit?

Prometheus values reliability. You can always view what statistics are available about your system, even under failure conditions. If you need 100% accuracy, such as for per-request billing, Prometheus is not a good choice as the collected data will likely not be detailed and complete enough. In such a case you would be best off using some other system to collect and analyze the data for billing, and Prometheus for the rest of your monitoring.

Grafana

Grafana has become the world’s most popular technology used to compose observability dashboards with everything from Prometheus & Graphite metrics, to logs and application data to power plants and beehives.



# MONITORING DOCKER CONTAINER METRICS USING CADVISOR

* [Prometheus configuration](https://prometheus.io/docs/guides/cadvisor/#prometheus-configuration)
* [Docker Compose configuration](https://prometheus.io/docs/guides/cadvisor/#docker-compose-configuration)
* [Exploring the cAdvisor web UI](https://prometheus.io/docs/guides/cadvisor/#exploring-the-cadvisor-web-ui)
* [Exploring metrics in the expression browser](https://prometheus.io/docs/guides/cadvisor/#exploring-metrics-in-the-expression-browser)
* [Other expressions](https://prometheus.io/docs/guides/cadvisor/#other-expressions)
* [Summary](https://prometheus.io/docs/guides/cadvisor/#summary)

[cAdvisor](https://github.com/google/cadvisor) (short for **c**ontainer **Advisor**) analyzes and exposes resource usage and performance data from running containers. cAdvisor exposes Prometheus metrics out of the box. In this guide, we will:

* create a local multi-container [Docker Compose](https://docs.docker.com/compose/) installation that includes containers running Prometheus, cAdvisor, and a [Redis](https://redis.io/) server, respectively
* examine some container metrics produced by the Redis container, collected by cAdvisor, and scraped by Prometheus

## Prometheus configuration

First, you'll need to [configure Prometheus](https://prometheus.io/docs/prometheus/latest/configuration/configuration) to scrape metrics from cAdvisor. Create a prometheus.yml file and populate it with this configuration:

scrape\_configs:

- job\_name: cadvisor

scrape\_interval: 5s

static\_configs:

- targets:

- cadvisor:8080

## Docker Compose configuration

Now we'll need to create a Docker Compose [configuration](https://docs.docker.com/compose/compose-file/) that specifies which containers are part of our installation as well as which ports are exposed by each container, which volumes are used, and so on.

In the same folder where you created the [prometheus.yml](https://prometheus.io/docs/guides/cadvisor/#prometheus-configuration) file, create a docker-compose.yml file and populate it with this Docker Compose configuration:

version: '3.2'

services:

prometheus:

image: prom/prometheus:latest

container\_name: prometheus

ports:

- 9090:9090

command:

- --config.file=/etc/prometheus/prometheus.yml

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml:ro

depends\_on:

- cadvisor

cadvisor:

image: gcr.io/google-containers/cadvisor:latest

container\_name: cadvisor

ports:

- 8080:8080

volumes:

- /:/rootfs:ro

- /var/run:/var/run:rw

- /sys:/sys:ro

- /var/lib/docker/:/var/lib/docker:ro

depends\_on:

- redis

redis:

image: redis:latest

container\_name: redis

ports:

- 6379:6379

This configuration instructs Docker Compose to run three services, each of which corresponds to a [Docker](https://docker.com/) container:

1. The prometheus service uses the local prometheus.yml configuration file (imported into the container by the volumes parameter).
2. The cadvisor service exposes port 8080 (the default port for cAdvisor metrics) and relies on a variety of local volumes (/, /var/run, etc.).
3. The redis service is a standard Redis server. cAdvisor will gather container metrics from this container automatically, i.e. without any further configuration.

To run the installation:

docker-compose up

If Docker Compose successfully starts up all three containers, you should see output like this:

prometheus | level=info ts=2018-07-12T22:02:40.5195272Z caller=main.go:500 msg="Server is ready to receive web requests."

You can verify that all three containers are running using the [ps](https://docs.docker.com/compose/reference/ps/) command:

docker-compose ps

Your output will look something like this:

Name Command State Ports

----------------------------------------------------------------------------

cadvisor /usr/bin/cadvisor -logtostderr Up 8080/tcp

prometheus /bin/prometheus --config.f ... Up 0.0.0.0:9090->9090/tcp

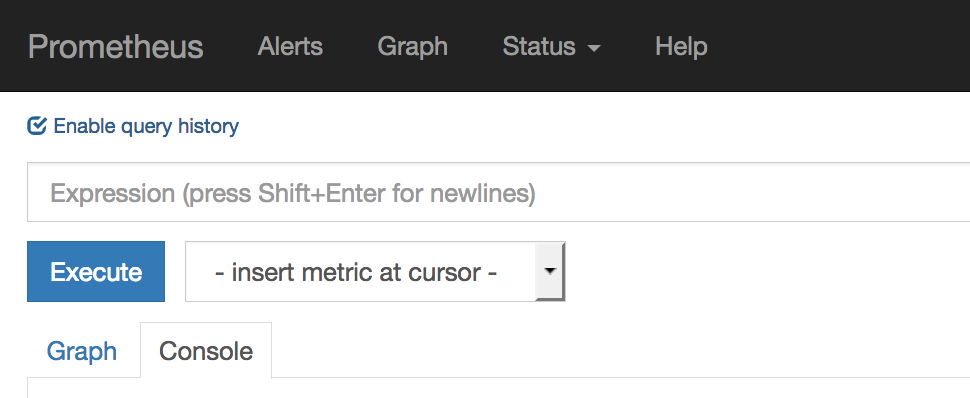
redis docker-entrypoint.sh redis ... Up 0.0.0.0:6379->6379/tcp

## Exploring the cAdvisor web UI

You can access the cAdvisor [web UI](https://github.com/google/cadvisor/blob/master/docs/web.md) at http://localhost:8080. You can explore stats and graphs for specific Docker containers in our installation at http://localhost:8080/docker/<container>. Metrics for the Redis container, for example, can be accessed at http://localhost:8080/docker/redis, Prometheus at http://localhost:8080/docker/prometheus, and so on.

## Exploring metrics in the expression browser

cAdvisor's web UI is a useful interface for exploring the kinds of things that cAdvisor monitors, but it doesn't provide an interface for exploring container metrics. For that we'll need the Prometheus [expression browser](https://prometheus.io/docs/visualization/browser), which is available at http://localhost:9090/graph. You can enter Prometheus expressions into the expression bar, which looks like this:



Let's start by exploring the container\_start\_time\_seconds metric, which records the start time of containers (in seconds). You can select for specific containers by name using the name="<container\_name>" expression. The container name corresponds to the container\_name parameter in the Docker Compose configuration. The [container\_start\_time\_seconds{name="redis"}](http://localhost:9090/graph?g0.range_input=1h&g0.expr=container_start_time_seconds%7Bname%3D%22redis%22%7D&g0.tab=1) expression, for example, shows the start time for the redis container.

System metric

A combination of Grafana & Prometheus can be used for monitoring system metrics – For e.g..

* Time up
* Memory usage/swap
* Disk usage
* Load
* Network
* CPU usage
* Disk I/O

*Alerts can be set on disk usage, memory usage and load usage to warn when the metric are critics.*

Docker metric

It can also be used for monitoring Docker container performance metrics like -

* CPU usage per container
* Sent network per container
* Received network per container
* Memory usage/swap per container
* Remaining memory for each container (if men\_limit defined in docker-compose.yml)

Conclusion

It has emerged as the GO TO platform for monitoring.

Useful links

[**https://sysdig.com/blog/kubernetes-monitoring-prometheus/**](https://sysdig.com/blog/kubernetes-monitoring-prometheus/)

[**https://prometheus.io/docs/guides/cadvisor/#prometheus-configuration**](https://prometheus.io/docs/guides/cadvisor/#prometheus-configuration)

[**https://phoenixnap.com/kb/prometheus-kubernetes-monitoring**](https://phoenixnap.com/kb/prometheus-kubernetes-monitoring)

**Prometheus -** [**https://prometheus.io/**](https://prometheus.io/)

**Grafana -** [**https://grafana.com/**](https://grafana.com/)

**cAdvisor -** [**https://github.com/google/cadvisor**](https://github.com/google/cadvisor)

**Node Exporter -** [**https://github.com/prometheus/node\_exporter**](https://github.com/prometheus/node_exporter)