# OVERVIEW

## 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/).

For more elaborate overviews of Prometheus, see the resources linked from the [media](https://prometheus.io/docs/introduction/media/) section.

### 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

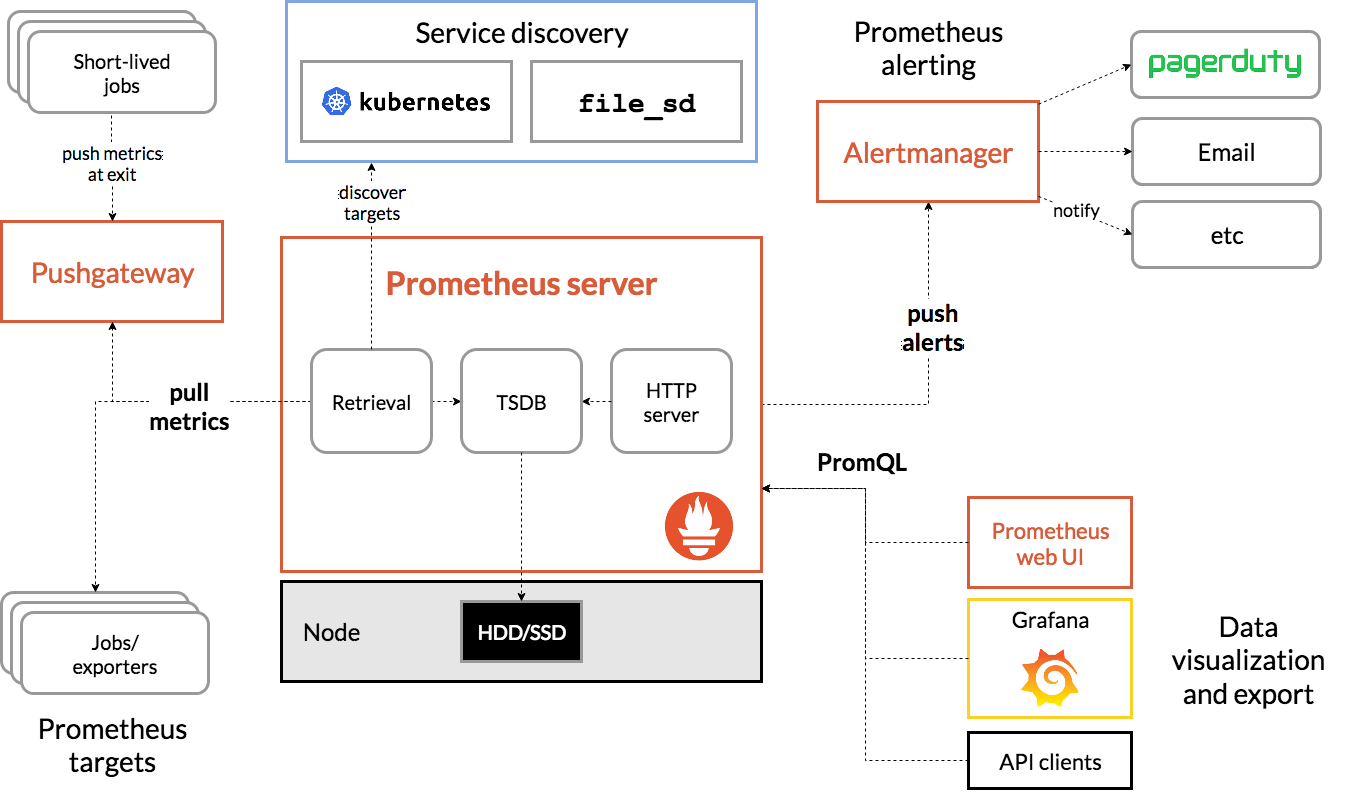
The Prometheus ecosystem consists of multiple components, many of which are optional:

* 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.

### Architecture

This diagram illustrates the architecture of Prometheus and some of its ecosystem components:



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.

## 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.

Collect Docker metrics with Prometheus

*Estimated reading time: 8 minutes*

[Prometheus](https://prometheus.io/) is an open-source systems monitoring and alerting toolkit. You can configure Docker as a Prometheus target. This topic shows you how to configure Docker, set up Prometheus to run as a Docker container, and monitor your Docker instance using Prometheus.

**Warning**: The available metrics and the names of those metrics are in active development and may change at any time.

Currently, you can only monitor Docker itself. You cannot currently monitor your application using the Docker target.

Configure Docker

To configure the Docker daemon as a Prometheus target, you need to specify the metrics-address. The best way to do this is via the daemon.json, which is located at one of the following locations by default. If the file does not exist, create it.

* **Linux**: /etc/docker/daemon.json
* **Windows Server**: C:\ProgramData\docker\config\daemon.json
* **Docker Desktop for Mac / Docker Desktop for Windows**: Click the Docker icon in the toolbar, select **Preferences**, then select **Daemon**. Click **Advanced**.

If the file is currently empty, paste the following:

{

"metrics-addr" : "IP:9323",

"experimental" : true

}

If the file is not empty, add those two keys, making sure that the resulting file is valid JSON. Be careful that every line ends with a comma (,) except for the last line.

Save the file, or in the case of Docker Desktop for Mac or Docker Desktop for Windows, save the configuration. Restart Docker.

Docker now exposes Prometheus-compatible metrics on port 9323.

Configure and run Prometheus

Prometheus runs as a Docker service on a Docker swarm.

**Prerequisites**

1. One or more Docker engines are joined into a Docker swarm, using docker swarm init on one manager and docker swarm join on other managers and worker nodes.
2. You need an internet connection to pull the Prometheus image.

Copy one of the following configuration files and save it to /tmp/prometheus.yml (Linux or Mac) or C:\tmp\prometheus.yml (Windows). This is a stock Prometheus configuration file, except for the addition of the Docker job definition at the bottom of the file. Docker Desktop for Mac and Docker Desktop for Windows need a slightly different configuration.

# my global config

global:

scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# scrape\_timeout is set to the global default (10s).

# Attach these labels to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

monitor: 'codelab-monitor'

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

# - "first.rules"

# - "second.rules"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'prometheus'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'docker'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['IP:9323']

Next, start a single-replica Prometheus service using this configuration.

* Docker for Linux
* Docker Desktop for Mac
* Docker Desktop for Windows or Windows Server

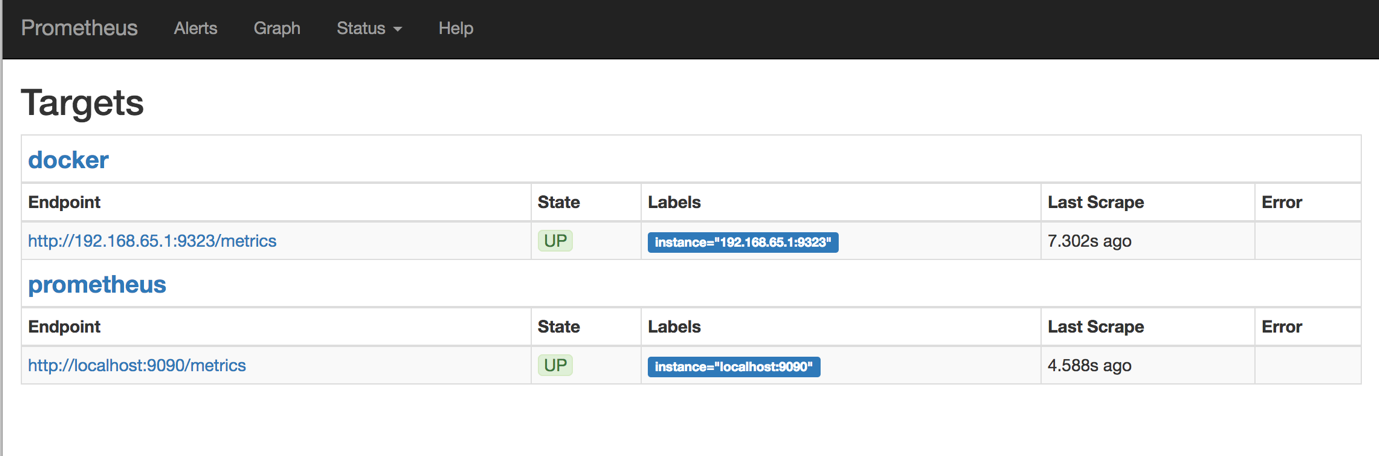
$ docker service create --replicas 1 --name my-prometheus \

--mount type=bind,source=/tmp/prometheus.yml,destination=/etc/prometheus/prometheus.yml \

--publish published=9090,target=9090,protocol=tcp \

prom/prometheus

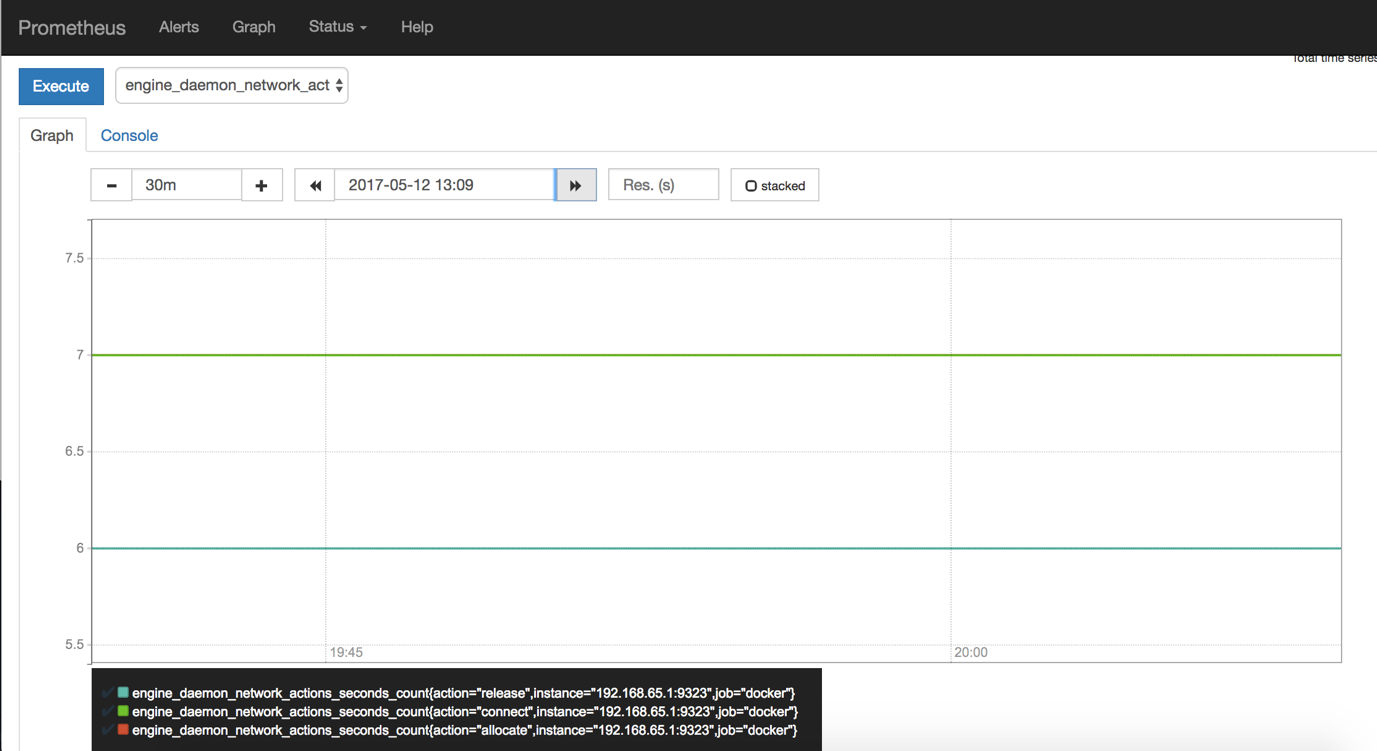
Verify that the Docker target is listed at http://localhost:9090/targets/.



You can’t access the endpoint URLs directly if you use Docker Desktop for Mac or Docker Desktop for Windows.

Use Prometheus

Create a graph. Click the **Graphs** link in the Prometheus UI. Choose a metric from the combo box to the right of the **Execute** button, and click **Execute**. The screenshot below shows the graph for engine\_daemon\_network\_actions\_seconds\_count.



The above graph shows a pretty idle Docker instance. Your graph might look different if you are running active workloads.

To make the graph more interesting, create some network actions by starting a service with 10 tasks that just ping Docker non-stop (you can change the ping target to anything you like):

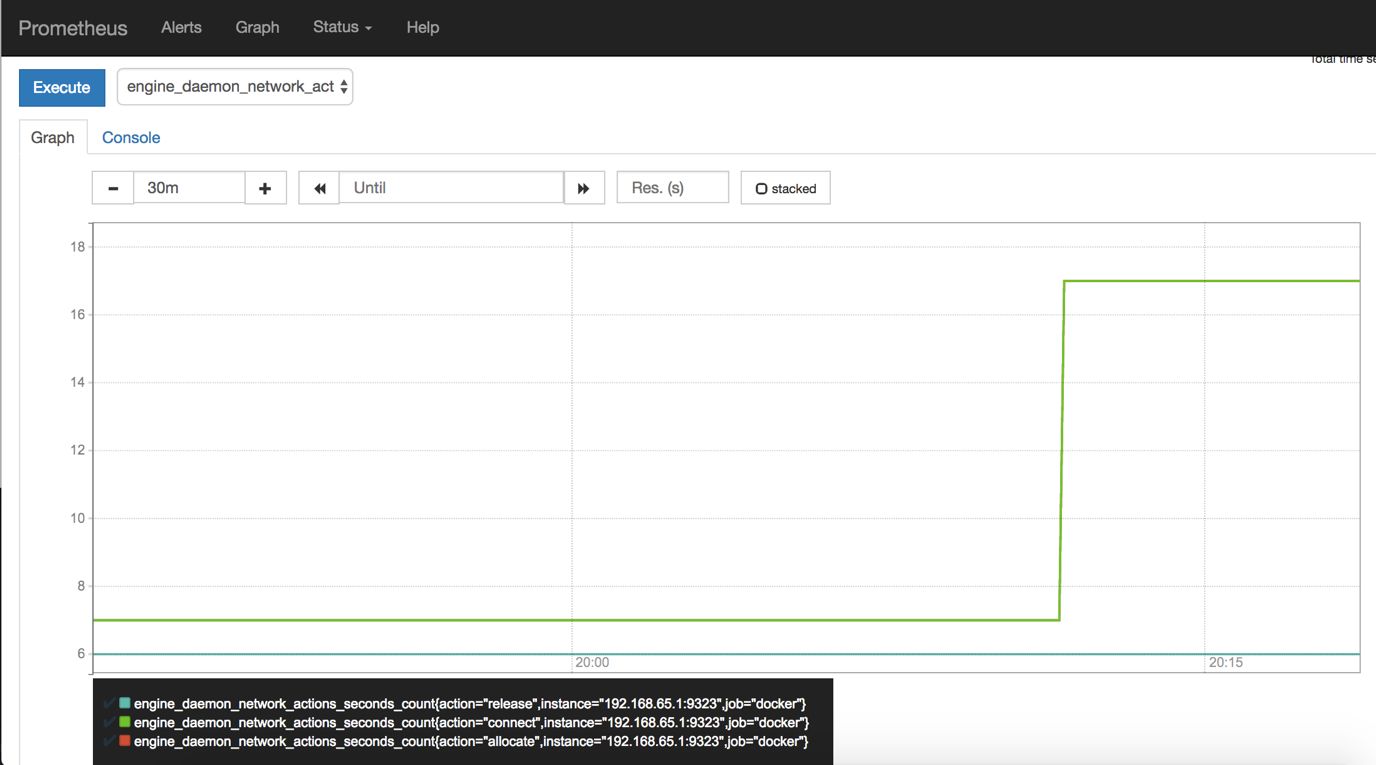
$ docker service create \

--replicas 10 \

--name ping\_service \

alpine ping docker.com

Wait a few minutes (the default scrape interval is 15 seconds) and reload your graph.



When you are ready, stop and remove the ping\_service service, so that you are not flooding a host with pings for no reason.

$ docker service remove ping\_service