**PROMETHEUS**

[Prometheus](https://github.com/prometheus) is an open-source systems monitoring and alerting toolkit originally built at [SoundCloud](https://soundcloud.com).

**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

**ARCHITECTURE**

This diagram illustrates the architecture of Prometheus and some of its ecosystem components:Graphical user interface, PowerPoint

Description automatically generated

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.

**DOWNLOADING PROMETHEUS**

[Download the latest release](https://prometheus.io/download) of Prometheus for your platform, then extract it:

tar xvfz prometheus-\*.tar.gz

cd prometheus-\*

The Prometheus server is a single binary called prometheus (or prometheus.exe on Microsoft Windows). We can run the binary and see help on its options by passing the --help flag.

./prometheus --help

usage: prometheus [<flags>]

The Prometheus monitoring server

. . .

**CONFIGURING PROMETHEUS**

Prometheus configuration is [YAML](https://yaml.org/). The Prometheus download comes with a sample configuration in a file called prometheus.yml that is a good place to get started.

We've stripped out most of the comments in the example file to make it more succinct (comments are the lines prefixed with a #).

global:

scrape\_interval: 15s

evaluation\_interval: 15s

rule\_files:

# - "first.rules"

# - "second.rules"

scrape\_configs:

- job\_name: prometheus

static\_configs:

- targets: ['localhost:9090']

There are three blocks of configuration in the example configuration file: global, rule\_files, and scrape\_configs.

The global block controls the Prometheus server's global configuration. We have two options present. The first, scrape\_interval, controls how often Prometheus will scrape targets. You can override this for individual targets. In this case the global setting is to scrape every 15 seconds. The evaluation\_interval option controls how often Prometheus will evaluate rules. Prometheus uses rules to create new time series and to generate alerts.

The rule\_files block specifies the location of any rules we want the Prometheus server to load. For now we've got no rules.

The last block, scrape\_configs, controls what resources Prometheus monitors. Since Prometheus also exposes data about itself as an HTTP endpoint it can scrape and monitor its own health. In the default configuration there is a single job, called prometheus, which scrapes the time series data exposed by the Prometheus server. The job contains a single, statically configured, target, the localhost on port 9090. Prometheus expects metrics to be available on targets on a path of /metrics. So this default job is scraping via the URL: <http://localhost:9090/metrics>.

The time series data returned will detail the state and performance of the Prometheus server.

**STARTING PROMETHEUS**

To start Prometheus with our newly created configuration file, change to the directory containing the Prometheus binary and run:

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

Prometheus should start up. You should also be able to browse to a status page about itself at <http://localhost:9090>. Give it about 30 seconds to collect data about itself from its own HTTP metrics endpoint.

You can also verify that Prometheus is serving metrics about itself by navigating to its own metrics endpoint: <http://localhost:9090/metrics>.

**GETTING PROMETHEUS SERVICE**

1. Open a terminal in your server or operating system.
2. Paste the given commands below to terminal.

$ apt install Prometheus

1. For getting default service procedure opening after any boot process:

$ systemctl enable prometheus.service

**GRAFANA**

Grafana is an open-source platform for monitoring and observability that lets you visualize and explore the state of your systems.

**INSTALING GRAFANA**

1. Open a terminal in your server or operating system.
2. Paste the given commands below to terminal.

sudo apt-get install -y apt-transport-https

sudo apt-get install -y software-properties-common wget

wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -

echo "deb https://packages.grafana.com/enterprise/deb stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

sudo apt-get update

sudo apt-get install grafana-enterprise

1. For getting default service procedure opening after any boot process:

systemctl enable grafana-server.service

**For open the Grafana server in your platform:**

1. Browse to localhost:3000.
2. In email or username, enter admin.
3. In password, enter admin.
4. Click Log In.
5. The first time you log in, you’re asked to change your password. (If you don’t want change the password you can click skip button.)
6. In New password, enter your new password.
7. In Confirm new password, enter the same password.
8. Click Save.

The first thing you see is the Home dashboard, which helps you get started.

To the far left you can see the sidebar, a set of quick access icons for navigating Grafana.

**ADD A METRICS DATA SOURCE**

The sample application exposes metrics which are stored in [Prometheus](https://prometheus.io/), a popular time series database (TSDB).

To be able to visualize the metrics from Prometheus, you first need to add it as a data source in Grafana.

1. In the side bar, hover your cursor over the **Configuration** (gear) icon, and then click **Data Sources**.
2. Click **Add data source**.
3. In the list of data sources, click **Prometheus**.
4. In the URL box, enter **http://prometheus:9090**.
5. Click **Save & Test**.

Prometheus is now available as a data source in Grafana.

**BUILD A DASHBOARD**

A *dashboard* gives you an at-a-glance view of your data and lets you track metrics through different visualizations.

Dashboards consist of *panels*, each representing a part of the story you want your dashboard to tell.

Every panel consists of a *query* and a *visualization*. The query defines *what* data you want to display, whereas the visualization defines *how* the data is displayed.

**FROM ALREADY EXİSTS DASHBOARD:**

1. In the side bar, hover your cursor over the **Create** (plus sign) icon and then click **Import**.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. In the via Grafana.com, enter the dashboard ID or URL (I used dashboard which ID is 11074.)

Graphical user interface, application, Teams

Description automatically generated

1. Click **Load** in the right corner to import via Grafana.com section. And load the dashboard which you entered before.
2. In the coming page, you can give the name to the dashboard (What if you want)
3. In Folder section, choose the which folder where do store the dashboard.
4. You should give the unique identifier to your dashboard.

(Unique identifier of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs)

1. Choose Prometheus in VictorialMetric
2. Click **Import** in the bottom of the page.

**A screenshot of a computer

Description automatically generated with medium confidence**

**GRAPHS ON THE DASHBOARD(ID:11074)**

Resource Overview:

* Service resource overview
* Node: Overall total 5m load & average CPU used%
* Node: Overall total memory & average memory used%
* Node: Overall total disk & average disk used%

Resource Details:

* Disk space used basic(EXT?/XPS)
* Internet traffic per hour All
* CPU% Basic
* Memory Basic
* Network bandwidth usage per second All
* System Load
* Disk R/W Data
* Disk Space Used% Basic
* Disk IOps Completed (IOPS)
* Time Spent Doing I/Os
* Disk R/W Time
* Network Sockstat
* Open File Descriptor

**CPU BASIC:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**QUERIES:**

* avg(rate(node\_cpu\_seconds\_total{instance=~"$node",mode="system"}[$interval])) by (instance) \*100,
* avg(rate(node\_cpu\_seconds\_total{instance=~"$node",mode="user"}[$interval])) by (instance) \*100,
* avg(rate(node\_cpu\_seconds\_total{instance=~"$node",mode="iowait"}[$interval])) by (instance) \*100,
* (1 - avg(rate(node\_cpu\_seconds\_total{instance=~"$node",mode="idle"}[$interval])) by (instance))\*100

**MEMORY BASIC:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**QUERIES:**

* node\_memory\_MemTotal\_bytes{instance=~"$node"},
* node\_memory\_MemTotal\_bytes{instance=~"$node"} - node\_memory\_MemAvailable\_bytes{instance=~"$node"},
* node\_memory\_MemAvailable\_bytes{instance=~"$node"},

**DISK R/W DATA**

**Chart

Description automatically generated**

**QUERIES:**

* rate(node\_disk\_read\_bytes\_total{instance=~"$node"}[$interval]),
* rate(node\_disk\_written\_bytes\_total{instance=~"$node"}[$interval])

**SYSTEM LOAD**

**A screenshot of a computer

Description automatically generated with medium confidence**

**QUERIES:**

* node\_load1{instance=~"$node"},
* node\_load5{instance=~"$node"},
* node\_load15{instance=~"$node"},
* sum(count(node\_cpu\_seconds\_total{instance=~"$node", mode='system'}) by (cpu,instance)) by(instance)

**OVERALL DASHBOARD SCREENSHOTS(ID:11074)**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Graphical user interface

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Graphical user interface, application

Description automatically generated**

**CONFIGURATION**

Prometheus can reload its configuration at runtime.

We should write a configuration file with yml standards

To specify which configuration file to load, use the --config.file flag.

**Configuring Rules**

Prometheus supports two types of rules:

-Recording Rules

-Alerting Rules

To include rules in Prometheus, create a file containing the necessary rule statements and have Prometheus load the file via the rule\_files field in the [Prometheus configuration](https://prometheus.io/docs/prometheus/latest/configuration/configuration/). Rule files use YAML.

The rule files can be reloaded at runtime by sending SIGHUP to the Prometheus process. The changes are only applied if all rule files are well-formatted.

To quickly check whether a rule file is syntactically correct without starting a Prometheus server, you can use Prometheus's promtool command-line utility tool:

promtool check rules /path/to/example.rules.yml

If there are any syntax errors or invalid input arguments return 1, Otherwise return 0.

Recording and alerting rules exist in a rule group. Rules within a group are run sequentially at a regular interval, with the same evaluation time.

The names of recording rules must be [valid metric names](https://prometheus.io/docs/concepts/data_model/" \l "metric-names-and-labels).

The names of alerting rules must be [valid label values](https://prometheus.io/docs/concepts/data_model/" \l "metric-names-and-labels).

**Recording Rules**

The Prometheus Rule feature allows to collect and use long PromQL queries under an alias.

The syntax of a rule file is:

groups:

[ - <rule\_group> ]

**<rule\_group>**

name: <string> (The name of the group) (Must be unique within a file)

[ interval: <duration> | default = global.evaluation\_interval ] (Define how often rules in the group are evaluated)

[ limit: <int> | default = 0] (0 is no limit) (Limit the number of alerts an alerting rule and series a recording) (rule can produce)

rules:

[ - <rule> … ]

**<rule>**

The syntax for recording rules is:

record: <string> (The name of the time series to output to) (Must be valid metric name)

expr: <string> (The PromQL expression to evaluate)

labels:

[ <labelname>: <labelvalue> ] (Labels to add or overwrite before storing the result)

The syntax for alerting rules is:

alert: <string> ( The name of the alert) (Must be a valid label value)

expr: <string> (The PromQL expression to evaluate)

[ for: <duration> | default = 0s ] ( firing time)

labels:

[ <labelname>: <tmpl\_string> ] (Labels to add or overwrite for each alerts)

annotations:

[ <labelname>: <tmpl\_string> ] (annotations to add to each alert)

**Alerting Rules**

Alerting rules are configured in Prometheus in the same way as [recording rules](https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/).

**Template Reference**

The primary data structure for dealing with time series data is the sample, defined as:

type sample struct {

Labels map[string]string

Value float64

}

**Unit Testing For Rules**

You can use promtool to test your rules

For a single test file:

./promtool test rules test.yml

If you have multiple test files:

./promtool test rules test1.yml test2.yml test3.yml

Test file format

rules\_files:

[ - <file\_name> ](This is a list of rule files to consider for testing)

[ evaluation\_inteval: <duration> ­ default = 1m ]

group\_eval\_order: (The order of groups)

[ - <group\_name> ]

tests: (All the tests are listed here)

[ - <test\_group> ]

<test\_group>

interval: <duration>

input\_series:

[- <series> ]

[ name: <string> ] (Name of the test group)

alert\_rule\_test: (Unit tests for alerting rules)

[ - <alert\_test\_case> ]

promql\_expr\_test: (Unit tests for PromQL expressions)

[ - <promql\_test\_case> ]

external\_labels:

[ <labelname>: <string> ] (External labels accessible to the alert template)

[ external\_url: <string> ] (External URL accessible to the alert template)

<series>

series: <string> (Rules of the series)

values: <string>

<alert\_test\_case>

eval\_time: <duration> (The time elapsed from time=0s when alerts have to be checked)

alertname: <string> (Name of the alert to be tested)

exp\_alerts: (List of expected alerts which are firing under the given alertname at given evaluation time)

[ - <alert> ]

<alert> (These are the expanded labels and annotations of the expected alert)

exp\_labels:

[ - <labelname>: <string> ]

exp\_annotations:

[ <labelname>: <string> ]

<promql\_test\_case>

expr: <string> (Expression to evaluate)

eval\_time: <duration> (The time elapsed from time=0s when the expression has to be avaluated)

exp\_samples: (Expected samples at the given evaluation time)

[ - <sample> ]

<sample>

labels: <string>

value: <number>