**INSTALLING**

**Grafana**

**STARTING PROMETHEUS**

$ apt install Prometheus

Server’ı boot sonrası otomatik açılması adına:

$ systemctl enable prometheus.service

Adress = <http://localhost:9090>

**STARTING GRAFANA**

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

Server’ı boot sonrası otomatik açılması adına:

systemctl enable grafana-server.service

Adress = <http://localhost:3000>

Belirtilen adrese girdikten sonra:

Username: admin

Password: admin

Ile arayüze giriş sağlanır.

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

Description automatically generated

Şekil-1

**Graphical user interface, application, Teams

Description automatically generated**

Şekil-2

**Template id: 11074**

**A screenshot of a computer

Description automatically generated with medium confidence**

Şekil-3

**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>