Prometheus

Prometheus is an open-source system monitoring and alerting toolkit originally built at SoundCloud. It is now a standalone open source project . Prometheus joined the **Cloud Native Computing Foundat**ion in 2016 as the second hosted project, after Kubernetes.

Features

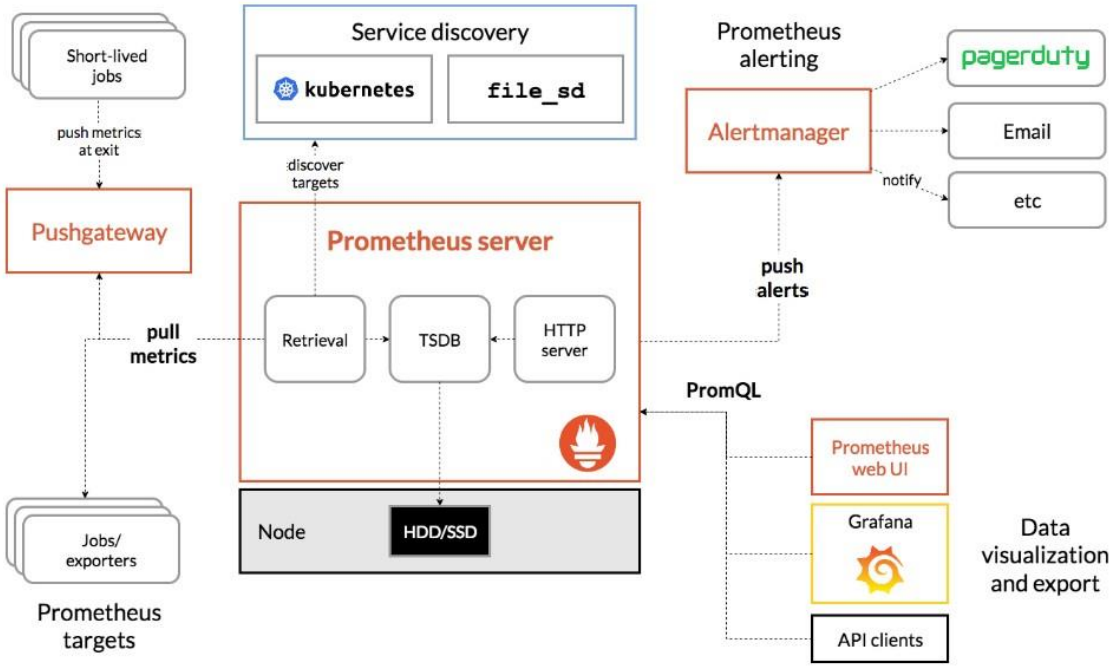
1. a multi-dimensional data model with time series data identified by metric name and **key/value** pairs
2. **PromQL**, a flexible query language to leverage this dimensionality
3. no reliance on distributed storage; single server nodes are autonomous
4. time series collection happens via a **pull model** over HTTP
5. pushing time series is supported via an intermediary gateway
6. targets are discovered via service discovery or static configuration
7. multiple modes of **graphing** and **dashboarding** support

Components

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

1. the main **Prometheus serve**r which scrapes and stores time series data
2. **client libraries** for instrumenting application code
3. a **push gateway** for supporting short-lived jobs
4. special-purpose **exporters** for services like HAProxy, StatsD, Graphite, etc.
5. an **alertmanager** to handle alerts
6. various support tools.

Architecture



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** 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.

Prometheus Installation.

Prometheus is a monitoring platform that collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets.

Download the latest release of Prometheus for your platform, then extract it:

https://prometheus.io/download/

download Prometheus

curl -LO https://github.com/prometheus/prometheus/releases/download/v2.9.2/prometheus-2.9.2.linux-amd64.tar.gz

Next, use the **sha256sum** command to generate a checksum of the downloaded file:

$ sha256sum prometheus-2.9.2.linux-amd64.tar.gz

```
root@vuser-HVM-domU-215:~# sha256sum prometheus-2.9.2.linux-amd64.tar.gz
19d29910fd0e51765d47b59b9276df016441ad4c6c48e3b27e5aa9acb5d1da26  prometheus-2.9.2.linux-amd64.tar.gz
```

then extract tar file

$ tar xvf prometheus-2.9.2.linux-amd64.tar.gz

This will create a directory called **prometheus-2.9.2.linux-amd64** containing two binary files **(prometheus and promtool), consoles** and **console_libraries** directories containing the **web interface files, a license, a notice**, and several example files.

```
root@vuser-HVM-domU-215:~# tar xvf prometheus-2.9.2.linux-amd64.tar.gz
prometheus-2.9.2.linux-amd64/
prometheus-2.9.2.linux-amd64/consoles/
prometheus-2.9.2.linux-amd64/consoles/node-cpu.html
prometheus-2.9.2.linux-amd64/consoles/prometheus.html
prometheus-2.9.2.linux-amd64/consoles/node.html
prometheus-2.9.2.linux-amd64/consoles/prometheus-overview.html
prometheus-2.9.2.linux-amd64/consoles/node-overview.html
prometheus-2.9.2.linux-amd64/consoles/index.html.example
prometheus-2.9.2.linux-amd64/consoles/node-disk.html
prometheus-2.9.2.linux-amd64/console_libraries/
prometheus-2.9.2.linux-amd64/console_libraries/menu.lib
prometheus-2.9.2.linux-amd64/console_libraries/prom.lib
prometheus-2.9.2.linux-amd64/promtool
prometheus-2.9.2.linux-amd64/prometheus.yml
prometheus-2.9.2.linux-amd64/LICENSE
prometheus-2.9.2.linux-amd64/NOTICE
prometheus-2.9.2.linux-amd64/prometheus
```

Copy the two binaries to the **/usr/local/bin** directory.

$ sudo cp prometheus-2.9.2.linux-amd64/prometheus /usr/local/bin/

$sudo cp prometheus-2.9.2.linux-amd64/promtool /usr/local/bin/

Configuring Prometheus

Open the **prometheus.yml** will contain just enough information to run Prometheus for the first time.

```
# 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).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# 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']
```

In the **global** settings, define the default interval for scraping metrics. Note that Prometheus will apply these settings to every exporter unless an individual exporter's own settings override the globals.

This **scrape_interval** value tells Prometheus to collect metrics from its exporters every 15 seconds, which is long enough for most exporters.

The **evaluation_interval** option controls how often Prometheus will evaluate rules. Prometheus uses rules to create new time series and to generate alerts.

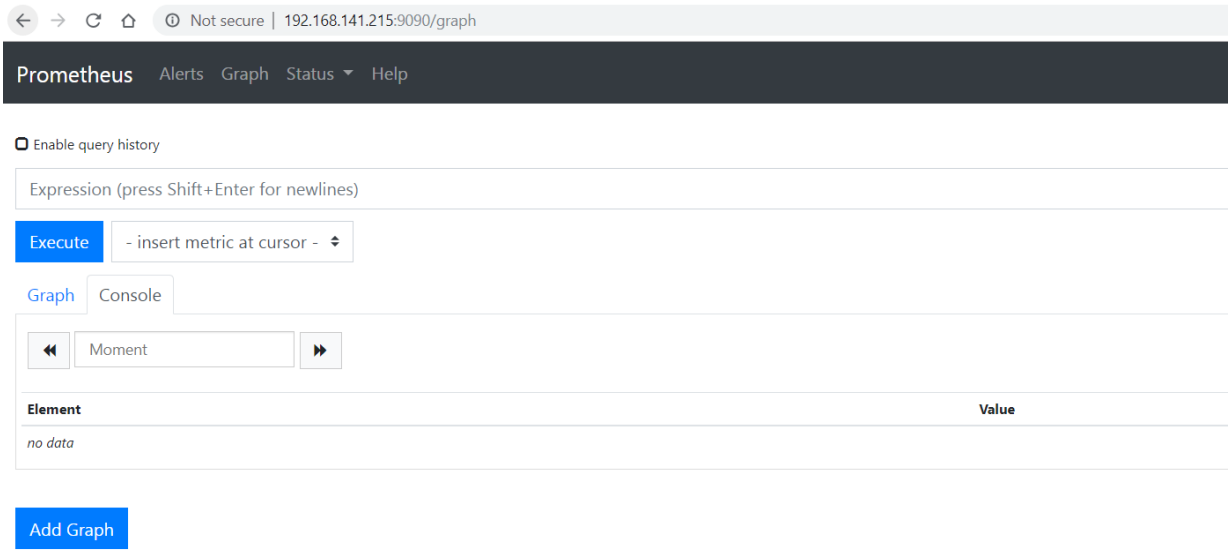
The **rule_files** block specifies the location of any rules we want the Prometheus server to load. For now we've got no rules.

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

~/prometheus-2.9.2.linux-amd64# ./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.

Using the expression browser

As you can gather from **http://localhost:9090/metrics**, one metric that Prometheus exports about itself is called ***promhttp_metric_handler_requests_total*** (the total number of ***/metrics*** requests the Prometheus server has served). Go ahead and enter this into the expression console

promhttp_metric_handler_requests_total

If we were only interested in requests that resulted in HTTP code 200, we could use this query to retrieve that information:

promhttp_metric_handler_requests_total{code="200"}

To count the number of returned time series, you could write:

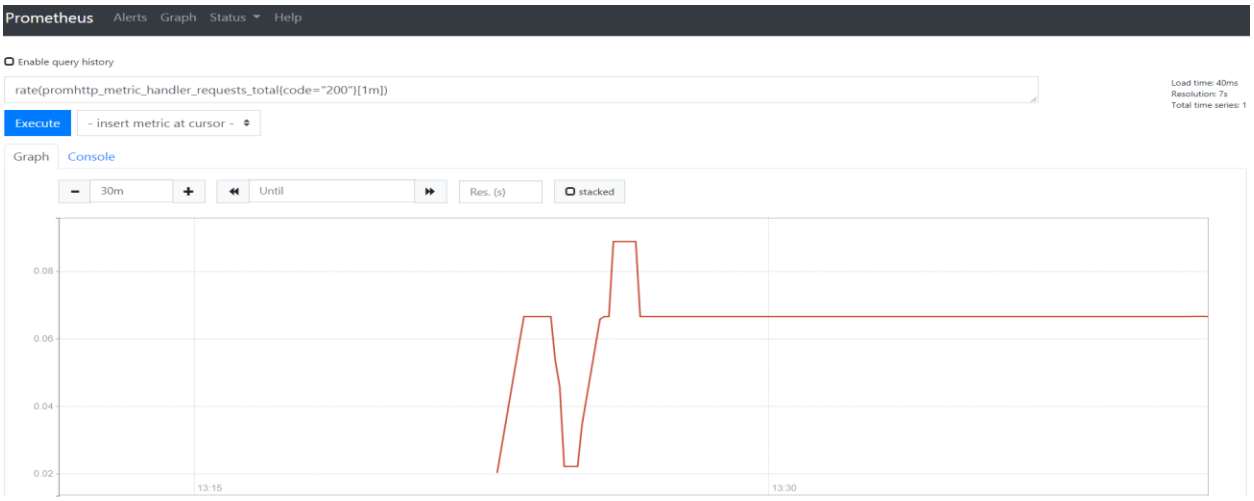
count(promhttp_metric_handler_requests_total)

Using the graphing interface

To graph expressions, navigate to **http://localhost:9090/graph** and use the "Graph" tab.

For example, enter the following expression to graph the per-second HTTP request rate returning status code 200 happening in the self-scraped Prometheus.

rate(promhttp_metric_handler_requests_total{code="200"}[1m])



MONITORING LINUX HOST METRICS WITH THE NODE EXPORTER

The Prometheus Node Exporter exposes a wide variety of hardware- and kernel-related metrics.

NOTE: While the Prometheus Node Exporter is for ***nix** systems, there is a ***WMI exporter*** for Windows that serves an analogous purpose.

Installing and running the Node Exporter

The Prometheus Node Exporter is a single static binary that you can install via tarball.

https://prometheus.io/download/#node_exporter download from here.

$ wget https://github.com/prometheus/node_exporter/releases/download/v0.17.0/node_exporter-0.17.0.linux-amd64.tar.gz

then extract

$ tar xvzf node_exporter-0.17.0.linux-amd64.tar.gz

$ cd node_exporter-0.17.0.linux-amd64/

$ ./node_exporter &

You should see output like this indicating that the Node Exporter is now running and exposing metrics on port 9100

```
INFO[0000]  - netstat                                        source="node_exporter.go:97"
INFO[0000]  - nfs                                            source="node_exporter.go:97"
INFO[0000]  - nfsd                                           source="node_exporter.go:97"
INFO[0000]  - sockstat                                       source="node_exporter.go:97"
INFO[0000]  - stat                                           source="node_exporter.go:97"
INFO[0000]  - textfile                                       source="node_exporter.go:97"
INFO[0000]  - time                                           source="node_exporter.go:97"
INFO[0000]  - timex                                          source="node_exporter.go:97"
INFO[0000]  - uname                                          source="node_exporter.go:97"
INFO[0000]  - vmstat                                         source="node_exporter.go:97"
INFO[0000]  - xfs                                            source="node_exporter.go:97"
INFO[0000]  - zfs                                            source="node_exporter.go:97"
INFO[0000] Listening on :9100                                source="node_exporter.go:111"
```

Node Exporter metrics

Once the Node Exporter is installed and running, you can verify that metrics are being exported by cURLing the **/metrics** endpoint.

$ curl http://localhost:9100/metrics

You will the output like below

```
process_cpu_seconds_total 0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1024
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 7
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.0063872e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.55669324613e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.1497472e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes -1
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 0
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

Done…! We are successfully installed node exporter. The Node Exporter is now exposing metrics that Prometheus can scrape, including a wide variety of system metrics further down in the output (prefixed with **node_**). To view those metrics (along with help and type information)

$ curl http://localhost:9100/metrics | grep "node_"

Configuring your Prometheus instances

Your locally running Prometheus instance needs to be properly configured in order to access Node Exporter metrics. The following **scrape_config** block (in a **prometheus.yml** configuration file) will tell the Prometheus instance to scrape from the Node Exporter via localhost:9100

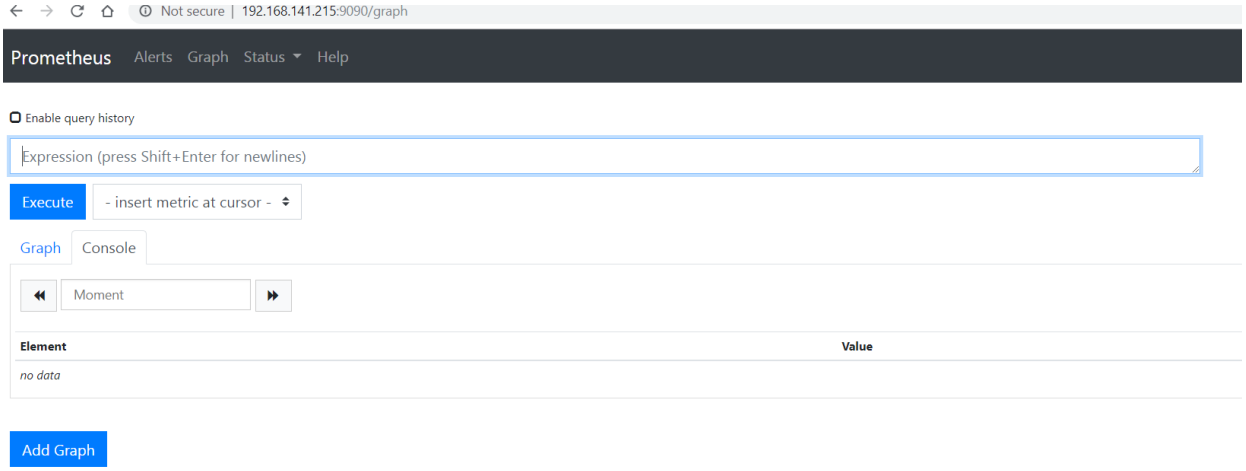
```
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'node'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9100']
~
~
```

Go to Prometheus installation directory and modify Prometheus.yml and run

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

Exploring Node Exporter metrics through the Prometheus expression browser

Now that Prometheus is scraping metrics from a running Node Exporter instance, you can explore those metrics using the Prometheus UI (aka the expression browser). Navigate to **localhost:9090/graph** in your browser and use the main expression bar at the top of the page to enter expressions. The expression bar looks like this



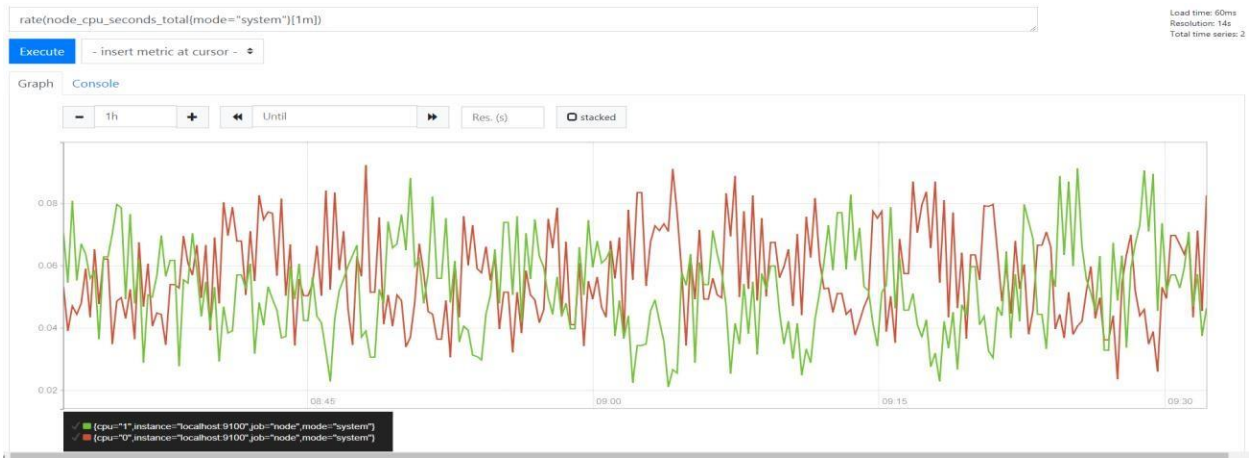
Metrics specific to the Node Exporter are prefixed with node_ and include metrics like **node_cpu_seconds_total** and **node_exporter_build_info.**

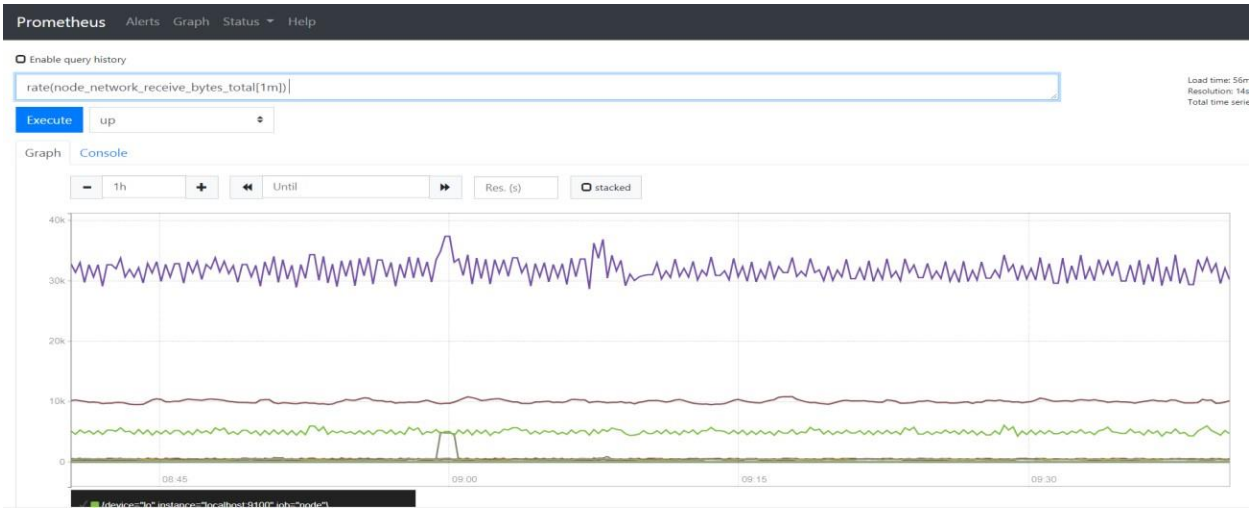
rate(node_cpu_seconds_total{mode="system"}[1m]) → The average amount of CPU time spent in system mode, per second, over the last minute (in seconds)

node_filesystem_avail_bytes → The filesystem space available to non-root users (in bytes)

rate(node_network_receive_bytes_total[1m]) → The average network traffic received, per second, over the last minute (in bytes)

node_load15 → load average in 15 seconds


rate(node_cpu_seconds_total{mode="system"}[1m])
Load time: 60ms
Resolution: 14s
Total time series: 2
Execute
- insert metric at cursor -
Graph
Console
1h
Until
Res. (s)
stacked
0.08
0.06
0.04
0.02
08:45
09:00
09:15
09:30
{cpu="1",instance="localhost:9100",job="node",mode="system"}
{cpu="0",instance="localhost:9100",job="node",mode="system"}


Prometheus
Alerts
Graph
Status
Help
Enable query history
rate(node_network_receive_bytes_total[1m])
Execute
up
Graph
Console
1h
Until
Res. (s)
stacked
40k
30k
20k
10k
0
08:45
09:00
09:15
09:30

If you want configure multiple hosts we can add in prometheus.yml like below

---

scrape_configs:

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

- job_name: 'node'

metrics_path defaults to '/metrics'

scheme defaults to 'http'.

static_configs:

- targets: ['192.168.x.x:9100']

- targets: ['192.168.x.y:9100']

- targets: ['192.168.x.z:9100']

GRAFANA

Grafana supports querying Prometheus. The Grafana data source for Prometheus is included since Grafana 2.5.0 (2015-10-28).

Installing Grafana

Download and unpack Grafana from binary tar.

$ wget https://dl.grafana.com/oss/release/grafana-6.1.6.linux-amd64.tar.gz

$ tar -zxvf grafana-6.1.6.linux-amd64.tar.gz

Start Grafana.

cd grafana-2.5.0/

./bin/grafana-server web

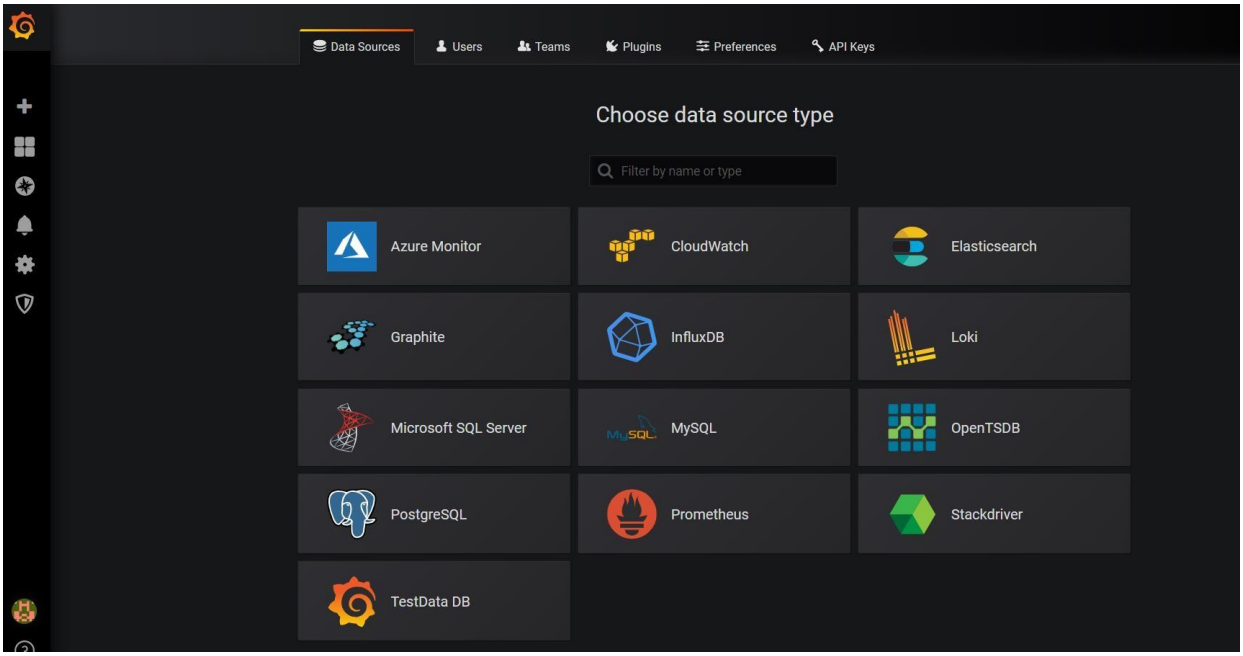
By default, Grafana will be listening on http://localhost:3000. The default login is "admin" / "admin".

Creating a Prometheus data source

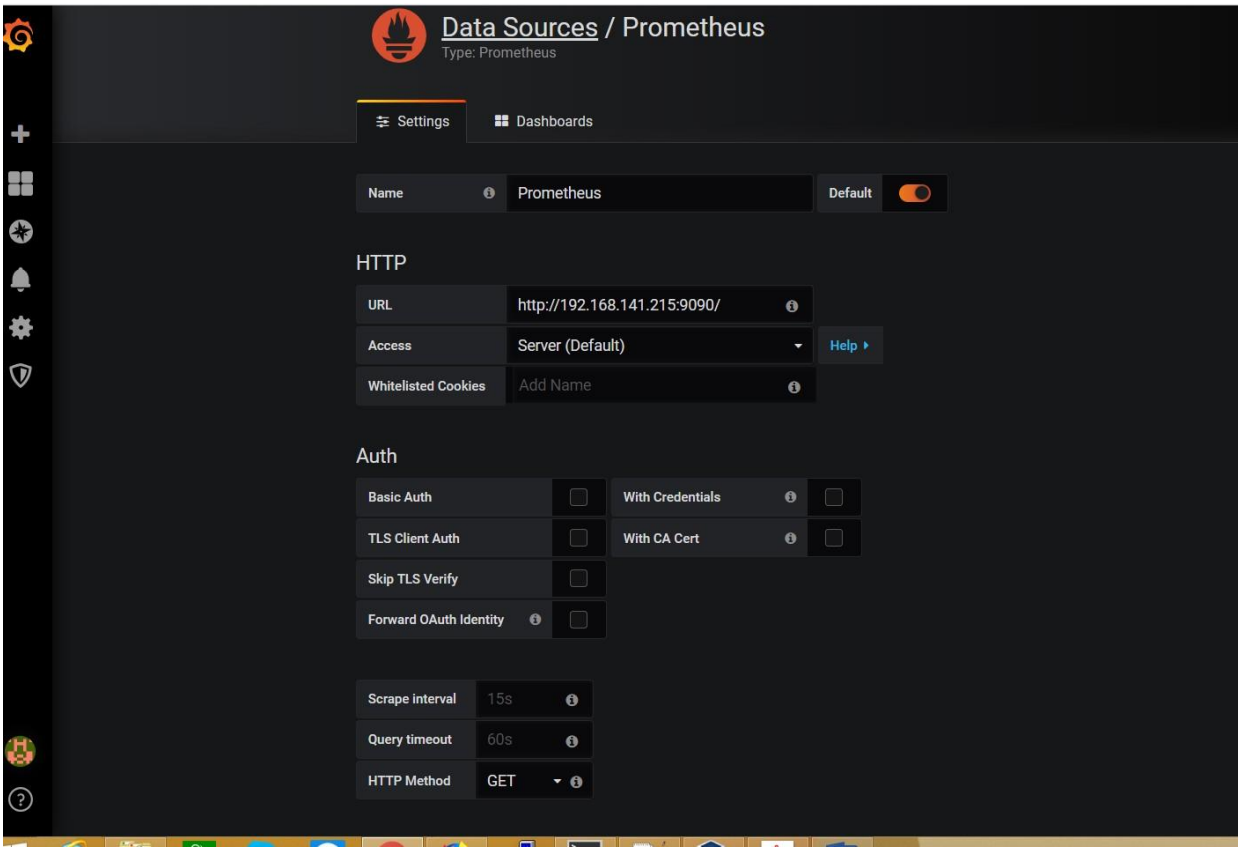
To create a Prometheus data source:

1. Click on the Grafana logo to open the sidebar menu.
2. Click on "Data Sources" in the sidebar.
3. Click on "Add New".
4. Select "Prometheus" as the type.
5. Set the appropriate Prometheus server URL (for example, http://localhost:9090/)
6. Adjust other data source settings as desired (for example, turning the proxy access off).
7. Click "Add" to save the new data source.

The following shows an example data source configuration:



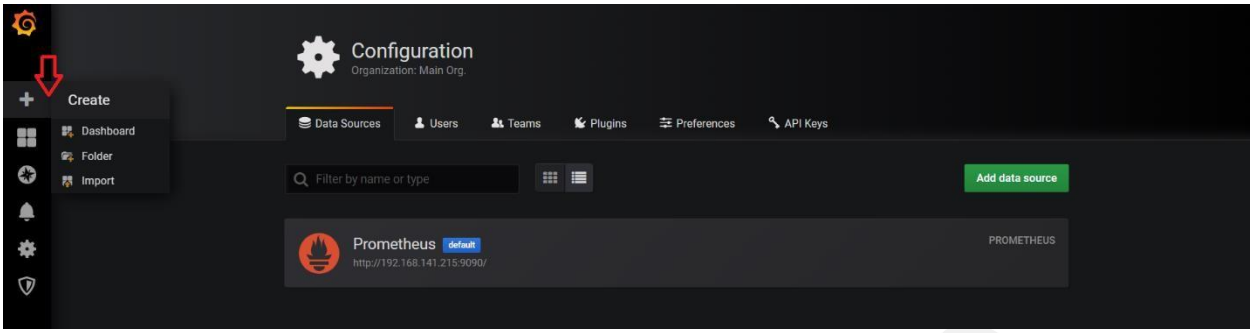
Once you select the Prometheus it will redirect the configuration page. Add necessary details click **save and test**.

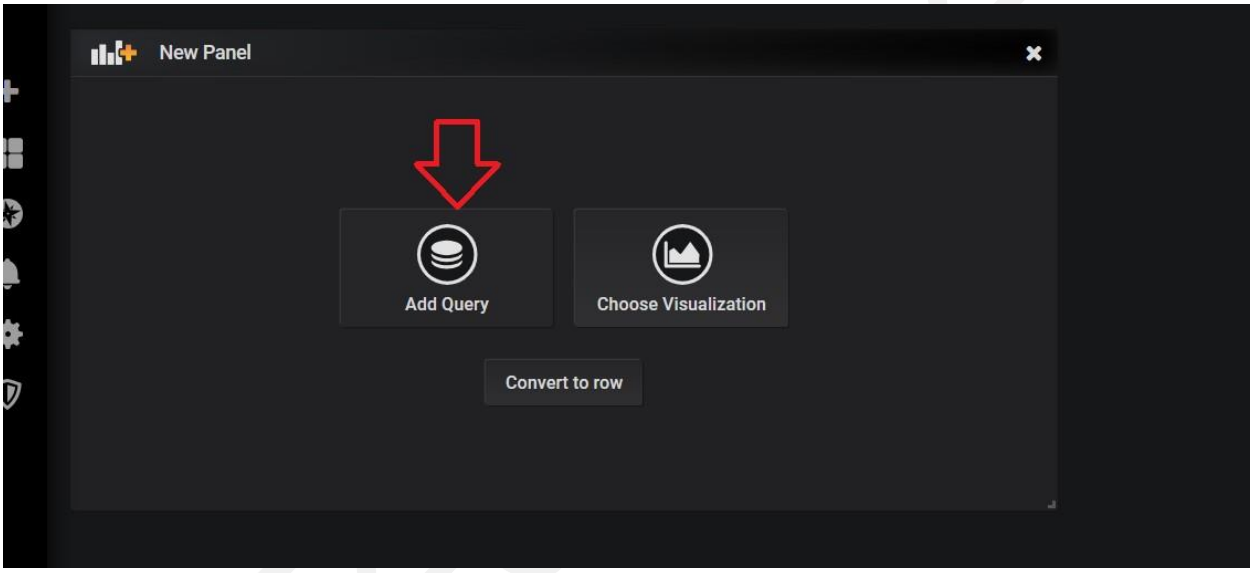


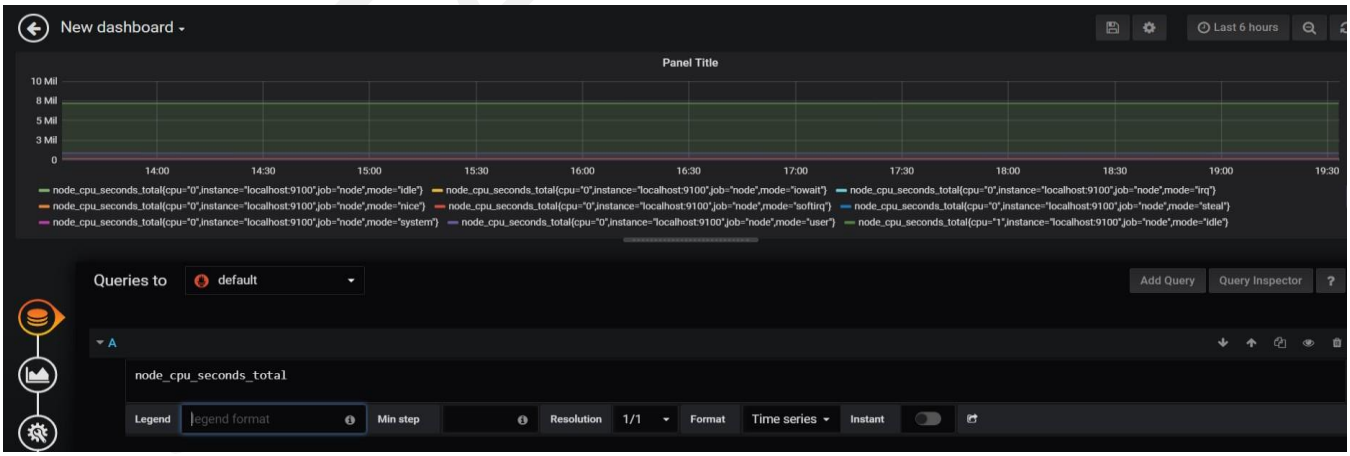
Creating a Prometheus graph

Follow the standard way of adding a new Grafana graph. Then:

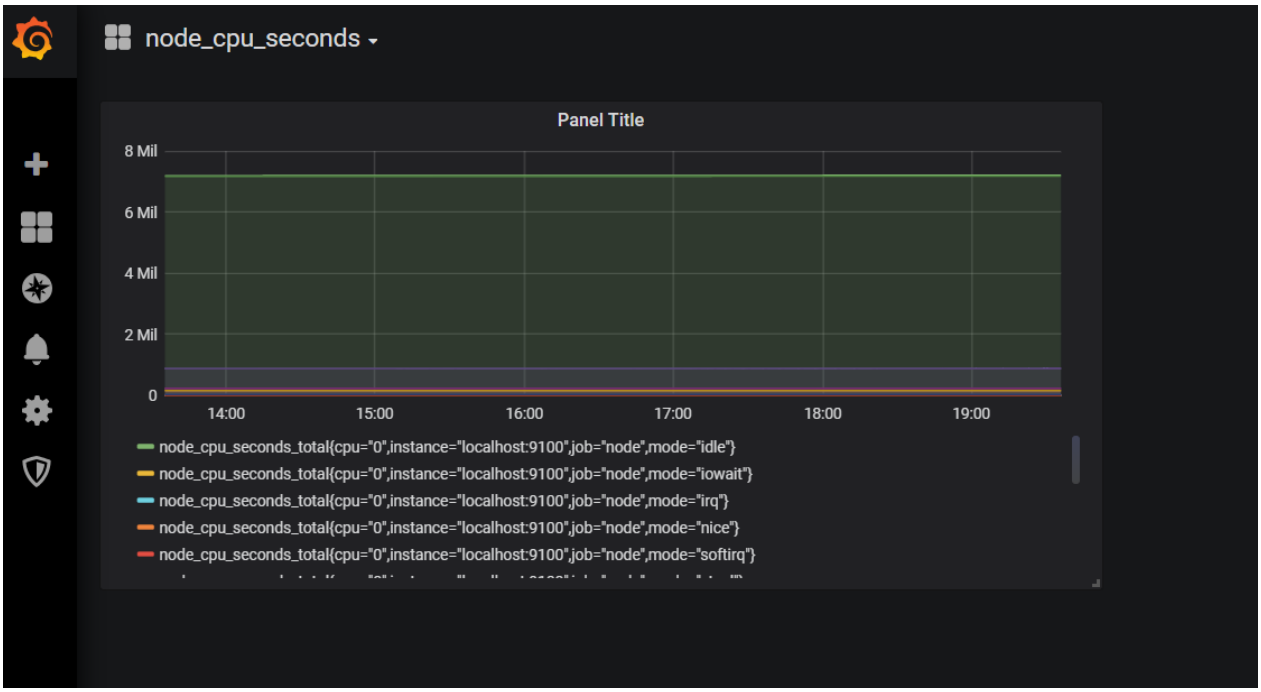
1. Go to panel bar click **+** mark create dashboard.
2. Check on query.
3. Enter a query in query panel.
4. Under the "Metrics" tab, select your Prometheus data source (bottom right).
5. To format the legend names of time series, use the "Legend format" input. For example, to show only the method and status labels of a returned query result, separated by a dash, you could use the legend format string {{method}} - {{status}}.
6. Tune other graph settings until you have a working graph.







Then save the dash board



Now if you want node exporter dash boards into Grafana serach in google like "**grafana node exporter metrics**". You will get some default dash boards like below.

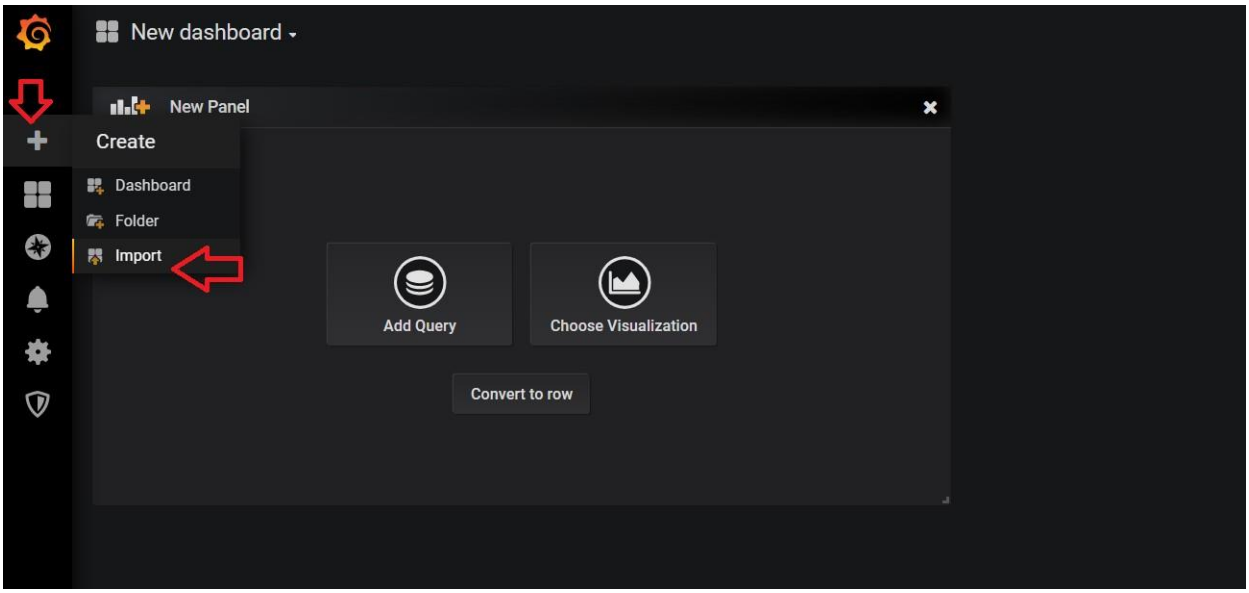
https://grafana.com/dashboards/405

https://grafana.com/dashboards/1860

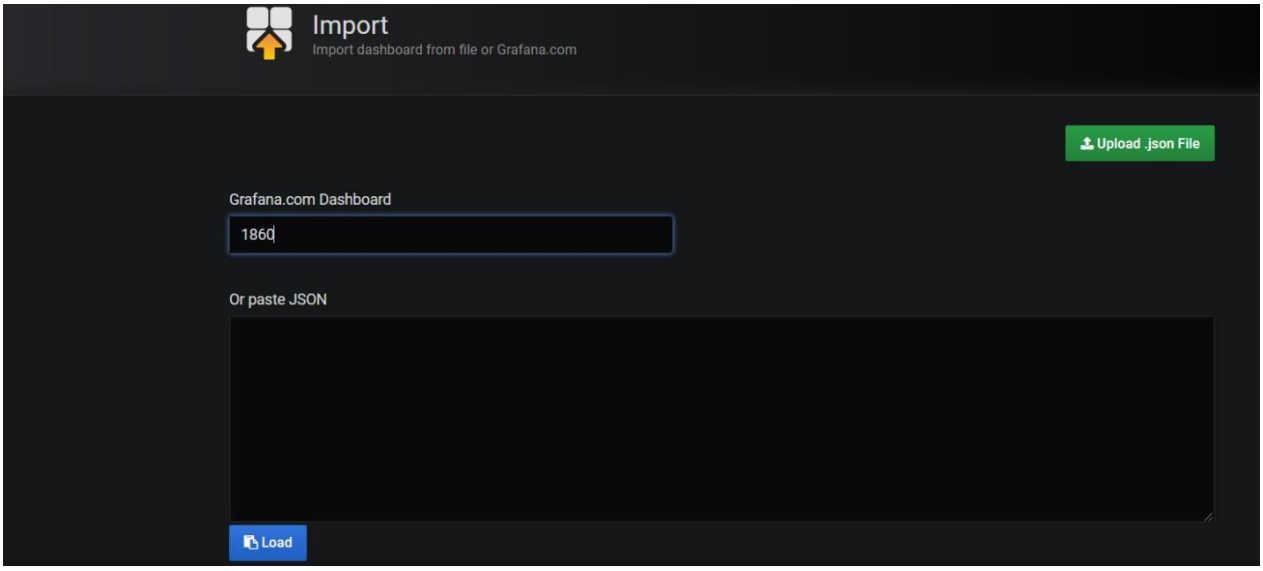
https://grafana.com/dashboards/5174

https://grafana.com/dashboards/9096

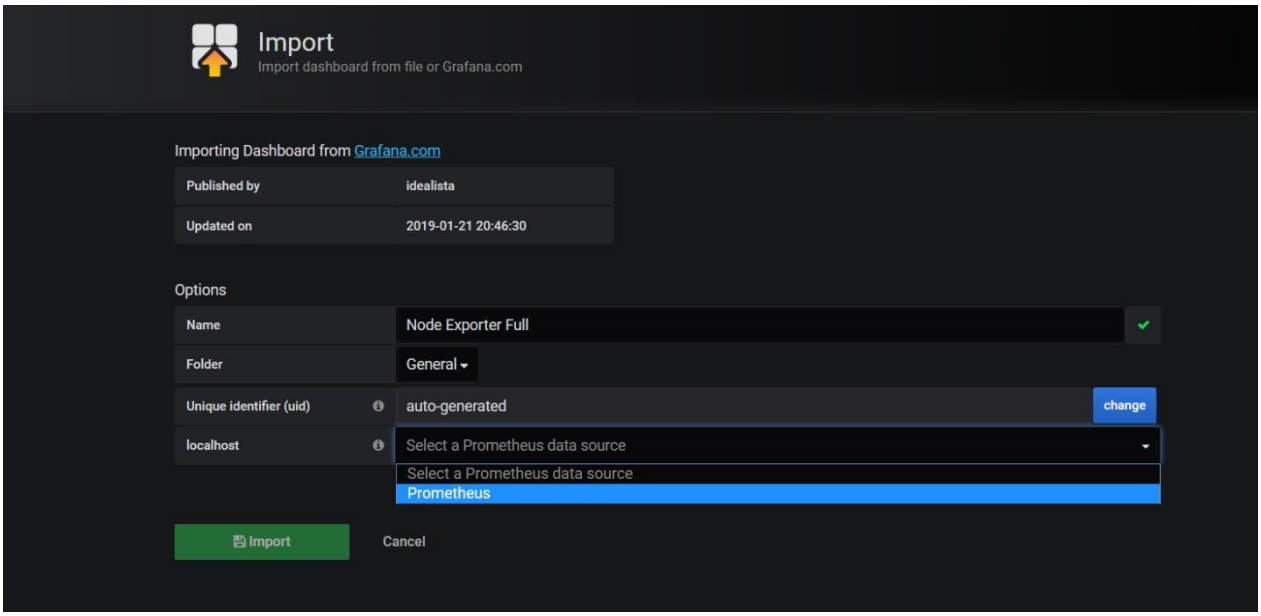
if we want add dash board of node exporter into Grafana goto dash board click + mark and click import



Then add dashboard id or url then click load.



And new page select data source to Prometheus and click import.



And see the dash board like below…

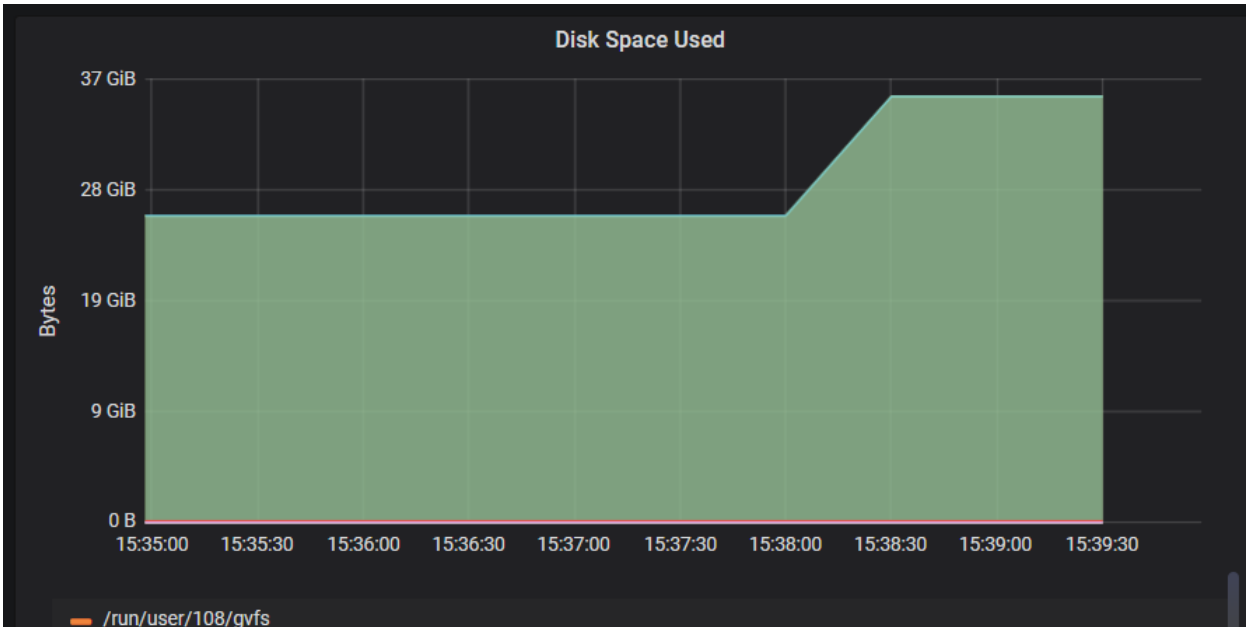
If want troubleshoot add big size file and check the graphs…

Run this command in your system .it will create 10gb file…

fallocate -l 10G gentoo_root.img

select last 5 min graph and go diskspace section and see the changes in disk space graph goto this **Basic Net / Disk Info** section**.**

.



Alerting

When an alert changes state it sends out notifications. Each alert rule can have multiple notifications. But in order to add a notification to an alert rule you first need to add and configure a notification channel (can be email, Pagerduty or other integration). This is done from the Notification Channels page.

Supported Notification Types.

- **Email**
- **Slack**
- **PagerDuty**
- **Webhook**

Other Supported Notification Channels

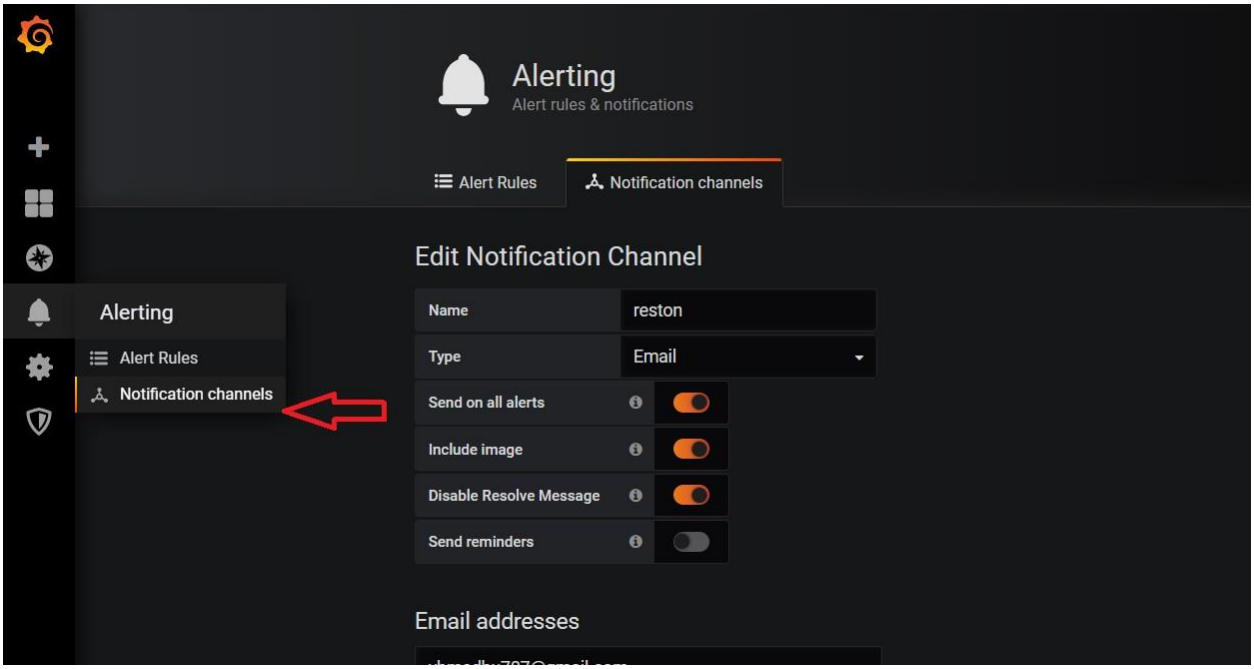
Grafana also supports the following Notification Channels:

HipChat
VictorOps
Sensu
OpsGenie

Threema
Pushover
Telegram
LINE
DingDing

Notification Channel Setup

On the Alert Notification Channels page hit the New Channel button to go the the page where you can configure and setup a new Notification Channel.



If you are selecting email as a alert type we need to configure SMTP server details inside Grafana/conf folder.

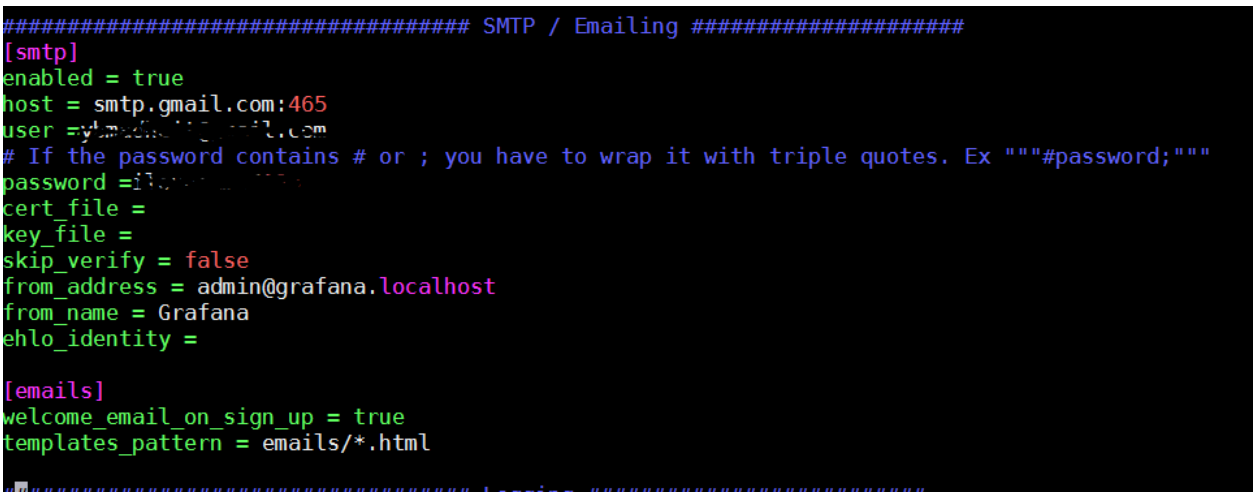
Go to "conf" directory of your Grafana distribution

Open your configuration file (as we did the setup using defaults so I am using "defaults.ini"). Navigate to SMTP/Emailing settings and update your SMTP details. As we have our fake-SMTP server running on localhost and on port 25. My "defaults.ini" has below configurations.

```
#################################### SMTP / Emailing ##########################
[smtp]
enabled = true
host = localhost:25
user =
# If the password contains # or ; you have to wrap it with triple quotes. Ex
"""#password;"""
password =
cert_file =
key_file =
skip_verify = false
from_address = admin@grafana.localhost
from_name = Grafana
ehlo_identity =
```

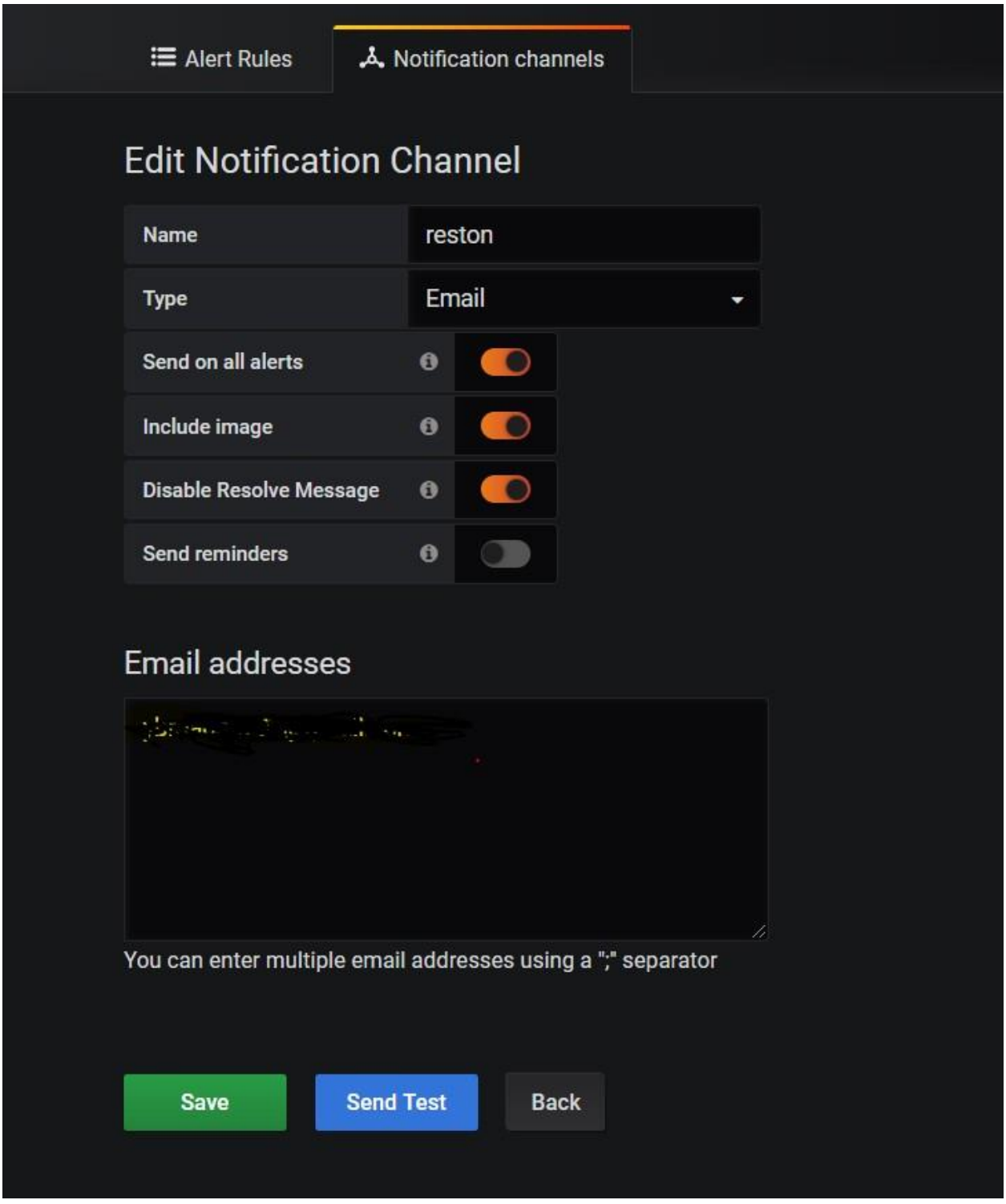
add the smtp confirmation like username and password host configurations. Once confirmation is done we need to restart the Grafana-server.

If you are using gmail as smtp server use below configuration



$ ./bin/Grafana-server web

Now ready to configure email alert type.



Fill appropriate values in the boxes name for alert and type of alert and add email address number of separated by ';' ….etc. and finally click send test button and check whether you receive mail or not.

Enable images in notifications

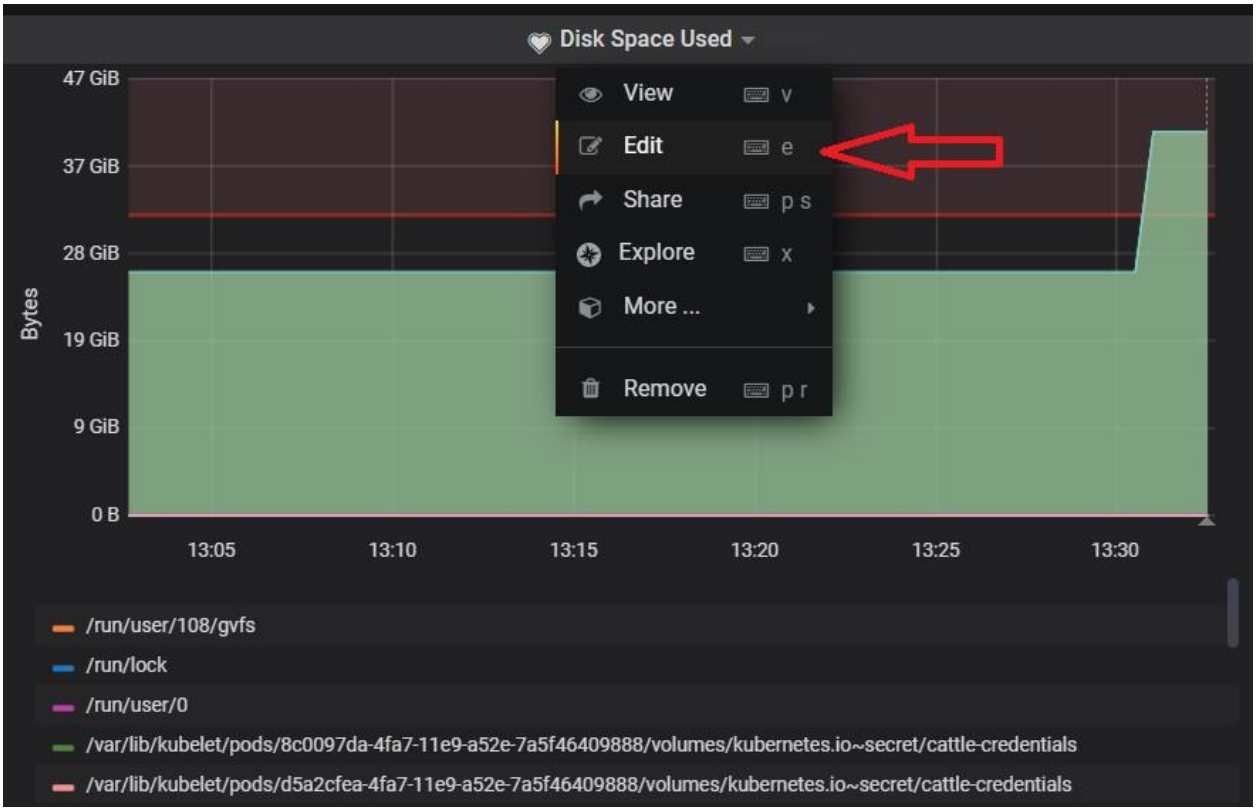
Grafana can render the panel associated with the alert rule and include that in the notification. Most Notification Channels require that this image be publicly accessable (Slack and PagerDuty for example). In order to include images in alert notifications, Grafana can upload the image to an image store. It currently supports Amazon S3 and Webdav for this. So to set that up you need to configure the external image uploader in your grafana-server ini config file.

Currently only the Email Channels attaches images if no external image store is specified. To include images in alert notifications for other channels then you need to set up an external image store.

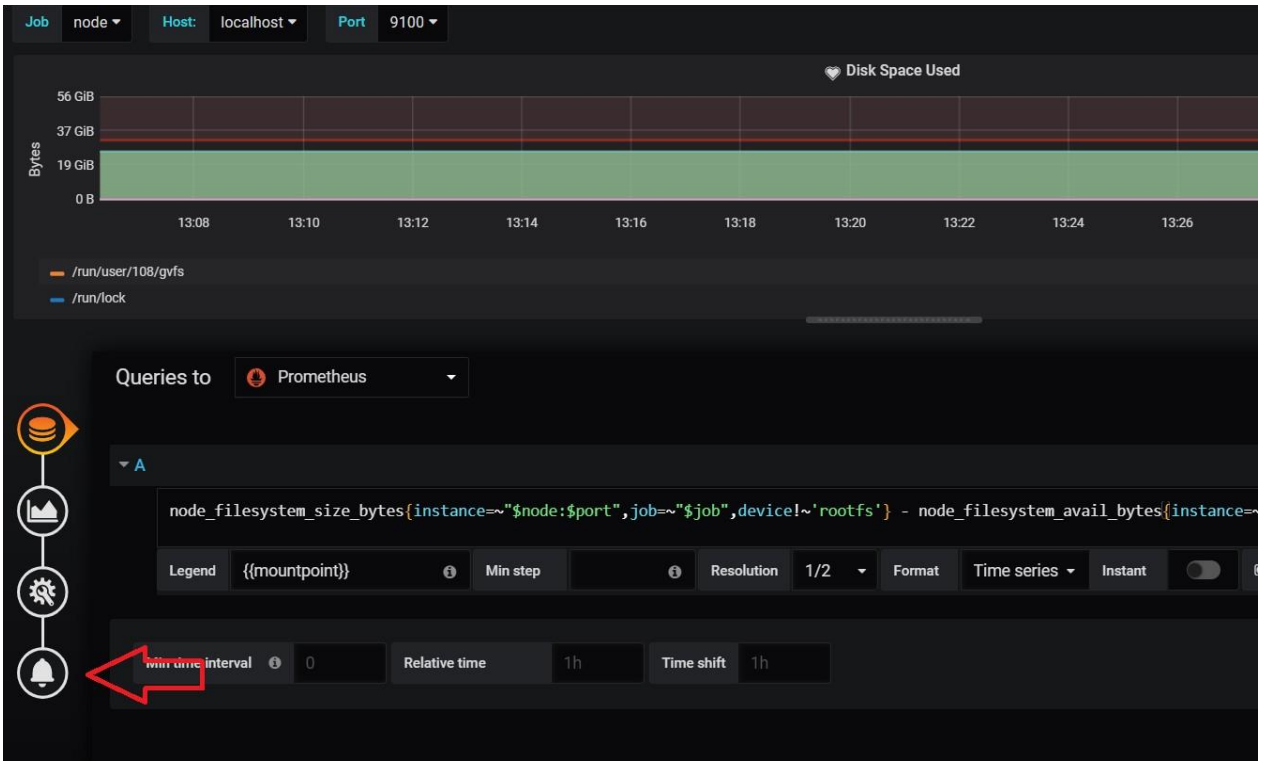
And now test email alert with real example….

Goto dash board panel and select a graph which we no need add alert.

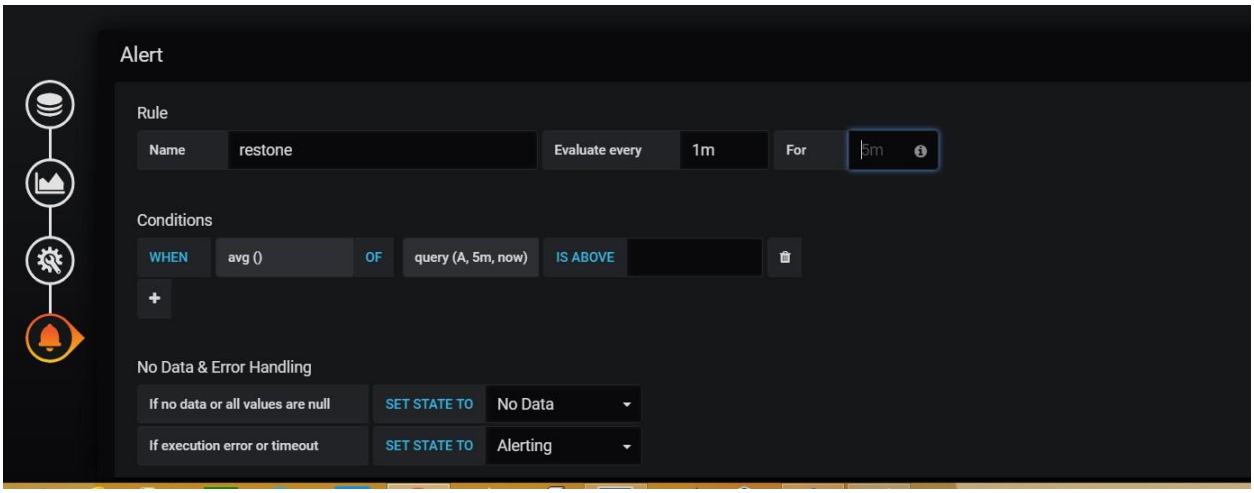
And go and edit



Once click the edit it will redirect into configuration tabs like below.



and if we want add alert configuration click bell icon. Then will see the configuration tab like below.



Now we need to the rule name and evaluate every (minutes of time) for when we need to send mail in for section (for example sec or min –15s or 30s or 1m)

And need to add condition of threshold value here…

Name: Give a suitable name to this alert.

Evaluate Every: The time interval on which you want this rule to be evaluated, where "h" means hours, "m" means minutes, and "s" means seconds.

I am using 5s (5 seconds). In production, you might want to set this to higher values.

Click on the function name in the "Conditions" section and change it to the desired function. I am using "Avg()" as we want to validate our rule against the Average of disk usage.

In function "query (A, 5m, now)".

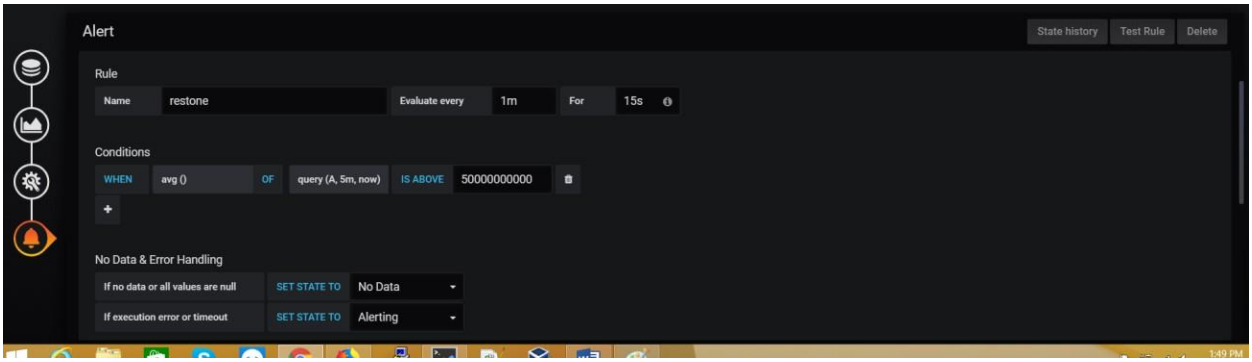
The first argument (A) is the query id to be used form all the queries which we have configured in "Metrics" tab. As we have the usage average query in section "A".

The second and third argument is used to form the time range. Our query means 5 minutes back from now.

You can validate your rule by clicking on 'Test Rule.' If the rule has any errors, then it will display the error. My final setup is as follows.

And write meaning full message in the message box and add the appropriate channel to send alert notification.

I do configuration like below for getting alert particular threshold of my disk usage..



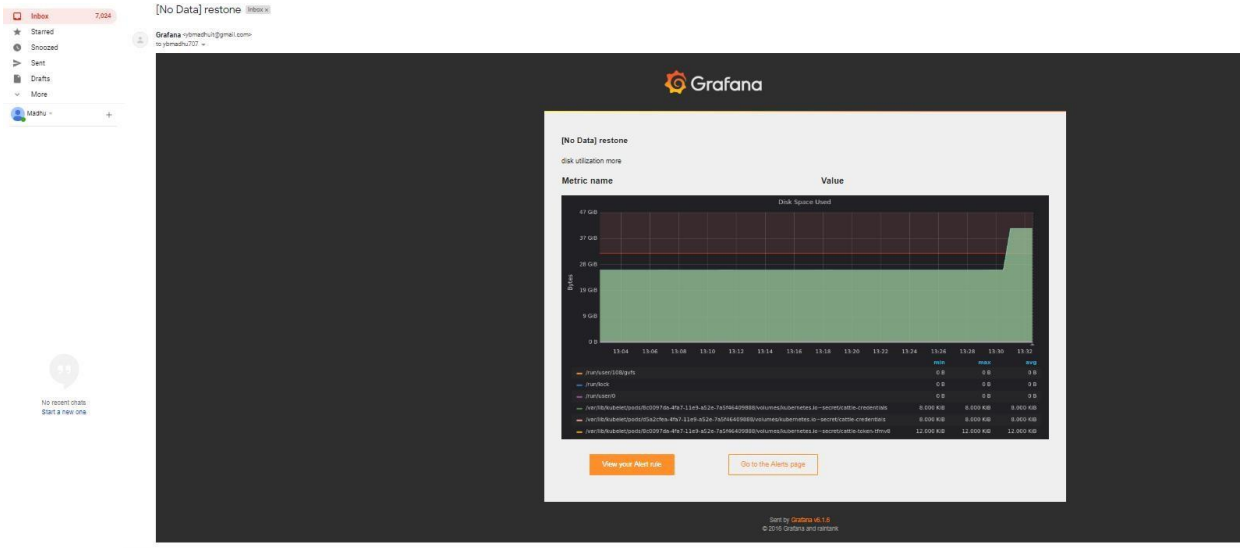
Once configuration is done and save it.

For this example I am creating a large file in my file system using below command.

$ fallocate -l 25G gentoo_root.img

Now I configured 50% usage in my disk utilization if its more than that I will get a alert notification with graph.


Job
node
Host:
localhost
Port
9100
Disk Space Used
56 GiB
37 GiB
19 GiB
0 B
Bytes
13:38
13:40
13:42
13:44
13:46
13:48
13:50
13:52
13:54
13:56
13:58
14:00
14:02
14:04
14:06
50000000000
min
max
avg
current
/run/user/108/gvfs
/run/lock
0 B
Alert
State history
Test Rule
Delete


Inbox
7,024
Starred
Snoozed
Sent
Drafts
More
Madhu
[No Data] restone
Grafana
Grafana
[No Data] restone
disk utilization more
Metric name
Value
Disk Space Used
View your Alert rule
Go to the Alerts page