Monitoring and Logging with Prometheus and Grafana (AWS EC2)

A web application crashing due to CPU or memory exceeding 100% can be a frustrating problem, especially without knowing the root cause. The key to preventing and quickly resolving such issues is proactive monitoring. By consistently monitoring your web application's performance, you can identify problems early and address them before they lead to crashes. Tools like Prometheus and Grafana can be invaluable in this process.

1. Create an EC2 instance (or select an EC2 instance that you want to monitor)
2. Create Grafana repo

```
sudo nano /etc/yum.repos.d/grafana.repo
```

3. enter the following:

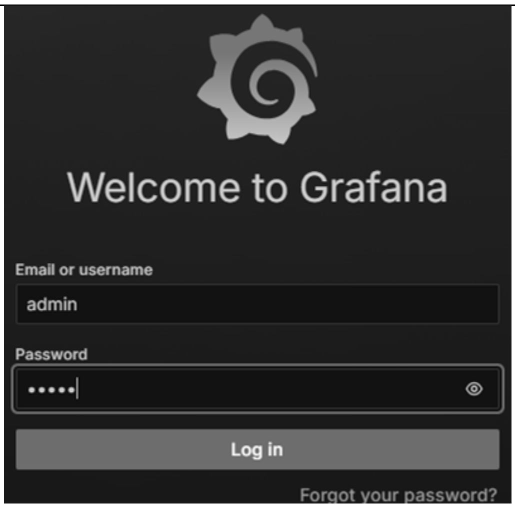
```
[grafana]
name=grafana
baseurl=https://packages.grafana.com/oss/rpm
repo_gpgcheck=1
enabled=1
gpgcheck=1
gpgkey=https://packages.grafana.com/gpg.key
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
```

4. Install and start services

```
sudo dnf install grafana
sudo systemctl daemon-reload
sudo systemctl start grafana-server
sudo systemctl status grafana-server
sudo systemctl enable --now grafana-server
```

5. Allow Ingress port 3000 in security group
6. Access the Grafana server using its public ip or public dns and port (3000)

```
http://<public ip>:3000
```



Welcome to Grafana

Email or username: admin

Password: •••••

7. Login using default username (admin) and password (admin)
8. Setup Prometheus and Node_Exporter
   https://docs.aws.amazon.com/en_us/lightsail/latest/userguide/amazon-lightsail-install-prometheus.html

9. Create two Linux user accounts, prometheus and exporter.

```
sudo useradd --no-create-home --shell /bin/false prometheus
sudo useradd --no-create-home --shell /bin/false exporter
```

10. Create local system directories.

```
sudo mkdir /etc/prometheus /var/lib/prometheus
sudo chown prometheus:prometheus /etc/prometheus
sudo chown prometheus:prometheus /var/lib/prometheus
```

11. Download Prometheus and node_exporter https://prometheus.io/download/

```
curl -LO https://github.com/prometheus/prometheus/releases/download/v2.53.3/prometheus-2.53.3.linux-amd64.tar.gz

curl -LO https://github.com/prometheus/node_exporter/releases/download/v1.8.2/node_exporter-1.8.2.linux-amd64.tar.gz
```

or try it out as a docker container (available publicly in docker hub)

```
docker run --name prometheus -d -p 127.0.0.1:9090:9090 prom/prometheus
```

12. After installing Prometheus, extract the Prometheus and Node_exporter downloaded file.

```
tar -xvf prometheus-2.53.3.linux-amd64.tar.gz
tar -xvf node_exporter-1.8.2.linux-amd64.tar.gz
```

13. Copy the prometheus and promtool extracted files to the /usr/local/bin programs directory.

```
sudo cp -p ./prometheus-2.53.3.linux-amd64/prometheus /usr/local/bin
sudo cp -p ./prometheus-2.53.3.linux-amd64/promtool /usr/local/bin
```

14. Enter the following command to change the ownership of the prometheus and promtool files to the prometheus user that you created.

```
sudo chown prometheus:prometheus /usr/local/bin/prom*
```

15. Enter the following commands one by one to copy the consoles and console_libraries subdirectories to /etc/prometheus. The -r option performs a recursive copy of all directories within the hierarchy.

```
sudo cp -r ./prometheus-2.53.3.linux-amd64/consoles /etc/prometheus
sudo cp -r ./prometheus-2.53.3.linux-amd64/console_libraries /etc/prometheus
```

16. Enter the following commands one by one to change the ownership of the copied files to the prometheus user that you created earlier. The -R option performs a recursive ownership change for all of the files and directories within the hierarchy.

```
sudo chown -R prometheus:prometheus /etc/prometheus/consoles
sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries
```

17. Enter the following commands one by one to copy the configuration file prometheus.yml to the /etc/prometheus directory and change the ownership of the copied file to the prometheus user that you created earlier in this tutorial.

```
sudo cp -p ./prometheus-2.53.3.linux-amd64/prometheus.yml /etc/prometheus
sudo chown prometheus:prometheus /etc/prometheus/prometheus.yml
```

18. Enter the following command to copy the node_exporter file from the ./node_exporter* subdirectory to the /usr/local/bin programs directory.

```
sudo cp -p ./node_exporter-1.8.2.linux-amd64/node_exporter /usr/local/bin
```

19. Enter the following command to change the ownership of the file to the exporter user that you created earlier in this tutorial.

```
sudo chown exporter:exporter /usr/local/bin/node_exporter
```

20. Edit the prometheus.yml

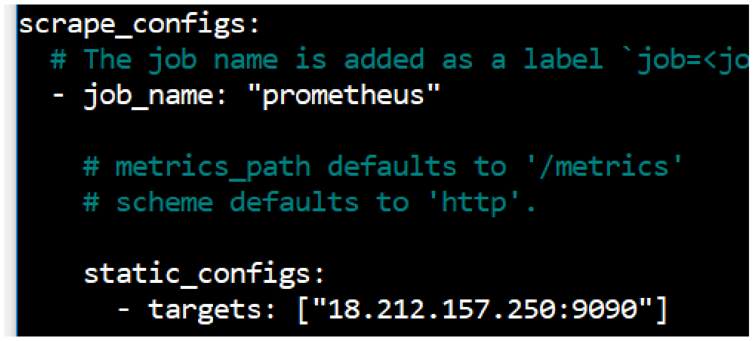
```
sudo nano /etc/prometheus/prometheus.yml
```

Following are a few important parameters that you might want to configure in the prometheus.yml file:
scrape_interval — Located under the global header, this parameter defines the time interval (in seconds) for how often Prometheus will collect or scrape metric data for a given target. As indicated by the global tag, this setting is universal for all resources that Prometheus monitors. This setting also applies for exporters, unless an individual exporter provides a different value that overrides the global value. You can keep this parameter set to its current value of 15 seconds.

job_name — Located under the scrape_configs header, this parameter is a label that identifies exporters in the result set of a data query or visual display. You can specify the value of a job name to best reflect the resources that are being monitored in your environment. For example, you can label a job for managing a website as business-web-app, or you can label a database as mysql-db-1. In this initial setup, you are only monitoring the Prometheus server, so you can keep the current prometheus value.

targets — Located under the static_configs header, the targets setting uses an ip_addr:port key-value pair to identify the location where a given exporter is running.

21. Check the targets, change localhost to a public IPv4 address on the EC2 instance that Prometheus installed.



```
scrape_configs:
  # The job name is added as a label `job=<jo
  - job_name: "prometheus"

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

    static_configs:
      - targets: ["18.212.157.250:9090"]
```

22. Edit the prometheus.service

```
sudo nano /etc/systemd/system/prometheus.service
```

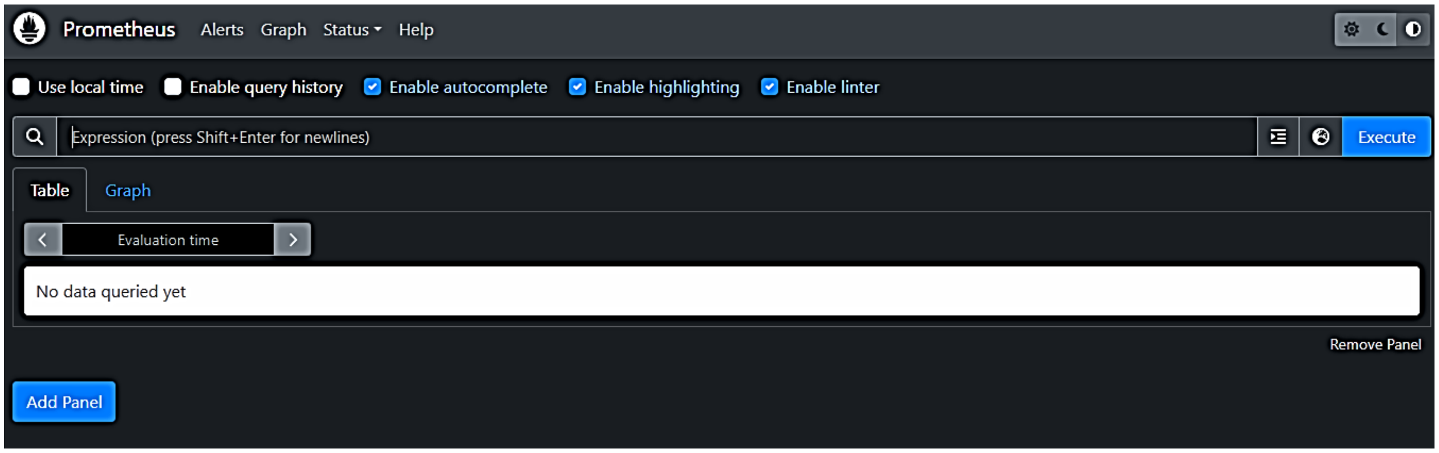
```
[Unit]
Description=PromServer
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
--config.file /etc/prometheus/prometheus.yml \
--storage.tsdb.path /var/lib/prometheus/ \
--web.console.templates=/etc/prometheus/consoles \
--web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
```

23. After Prometheus has already been installed, start Prometheus and check Prometheus status.

```
sudo systemctl daemon-reload
sudo systemctl start prometheus
sudo systemctl status prometheus
sudo systemctl enable --now prometheus
```

24. Open a web browser on your local computer and go to the following web address to view the Prometheus management interface.

```
http:<ip_addr>:9090
```



25. To start Node_exporter:

```
sudo nano /etc/systemd/system/node_exporter.service
```

```
[Unit]
Description=NodeExporter
Wants=network-online.target
After=network-online.target

[Service]
User=exporter
Group=exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter --collector.disable-defaults \
--collector.meminfo \
--collector.loadavg \
--collector.filesystem

[Install]
WantedBy=multi-user.target
```

26. Start and enable the services

```
sudo systemctl daemon-reload
sudo systemctl start node_exporter
sudo systemctl status node_exporter
sudo systemctl enable node_exporter
```

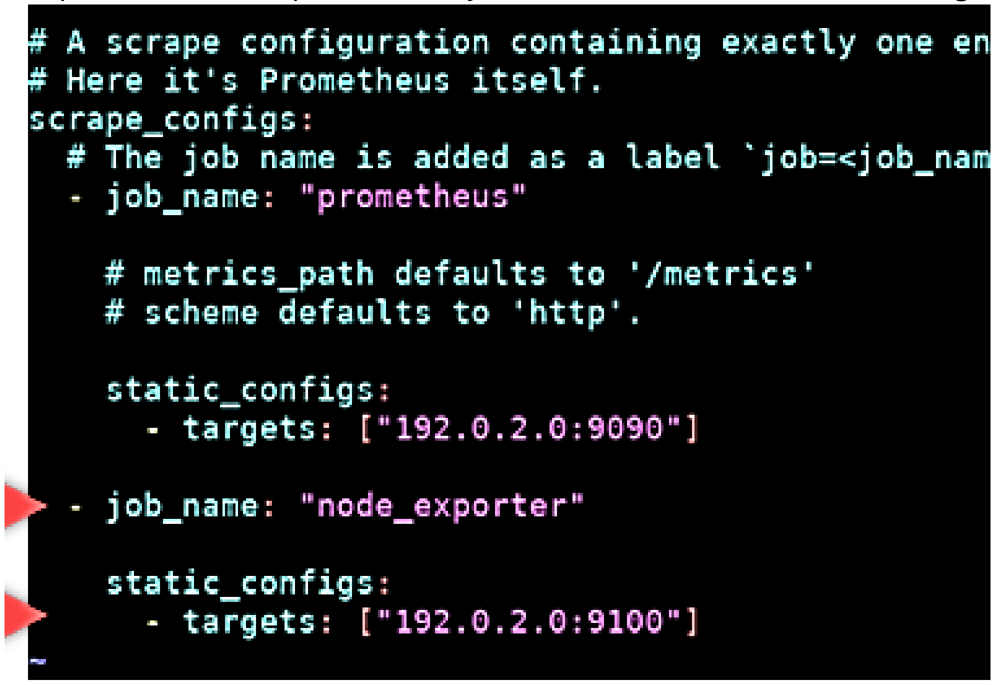
27. Configure Prometheus with the Node Exporter data collector

```
sudo nano /etc/prometheus/prometheus.yml
```

28. Add the following lines of text into the file, below the existing - targets: ["<ip_addr>:9090"] parameter.

```
- job_name: "node_exporter"

static_configs:
- targets: ["<ip_addr>:9100"]
```

The modified parameter in the prometheus.yml file should look like the following example.



```
# A scrape configuration containing exactly one en
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_nam
  - job_name: "prometheus"

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

    static_configs:
      - targets: ["192.0.2.0:9090"]

  - job_name: "node_exporter"

    static_configs:
      - targets: ["192.0.2.0:9100"]
~
```

29. Note the following:
    - Node Exporter listens to port 9100 for the prometheus server to scrape the data.
    - As with the configuration of the prometheus job_name, replace <ip_addr> with the static IP address that's attached to your ec2 instance.

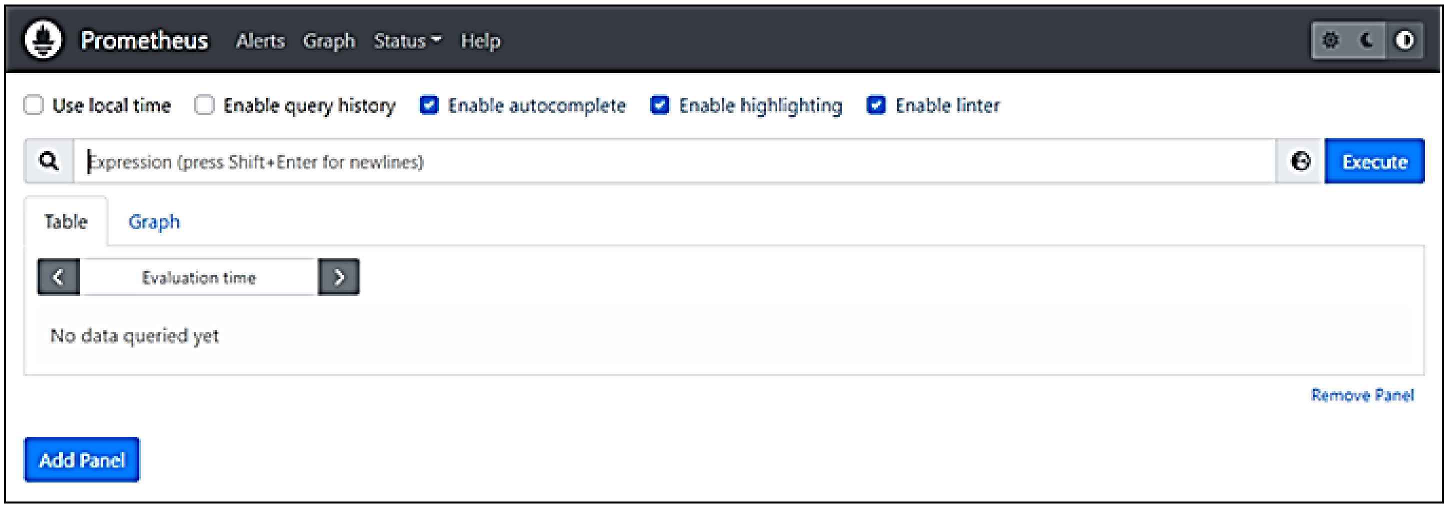
30. restart the Prometheus service so that the changes to the configuration file can take effect.

```
sudo systemctl restart prometheus
sudo systemctl status prometheus
```

31. Open a web browser on your local computer and go to the following web address to view the Prometheus management interface.

```
http:<ip_addr>:9090
```

32. Replace <ip_addr> with the static IP address of your Lightsail instance. You should see a dashboard similar to the following example.


Prometheus Alerts Graph Status Help

☐ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

Expression (press Shift+Enter for newlines) Execute

Table Graph

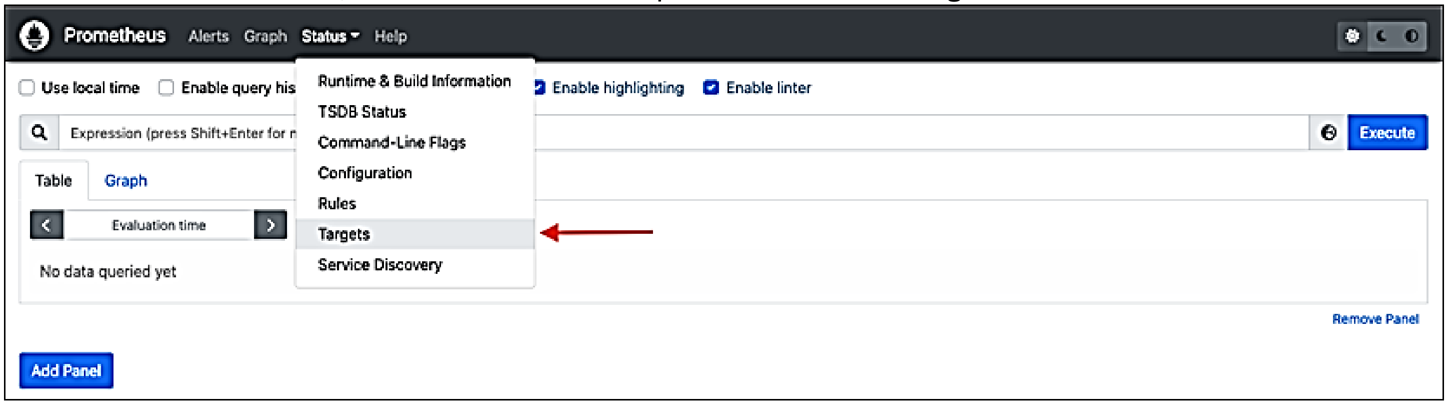
Evaluation time

No data queried yet

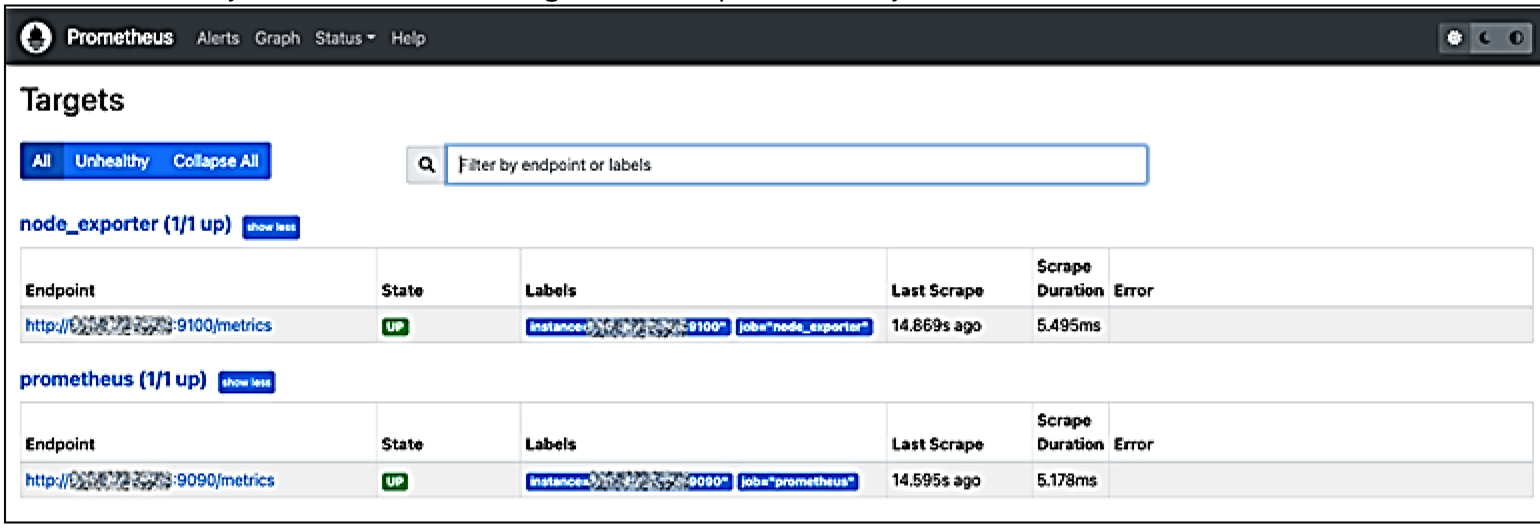
Remove Panel

Add Panel

33. In the main menu, choose the Status dropdown and select Targets.



34. On the next screen, you should see two targets. The first target is for the node_exporter metrics collector job, and the second target is for the prometheus job.


Prometheus Alerts Graph Status Help

Targets

All Unhealthy Collapse All

Filter by endpoint or labels

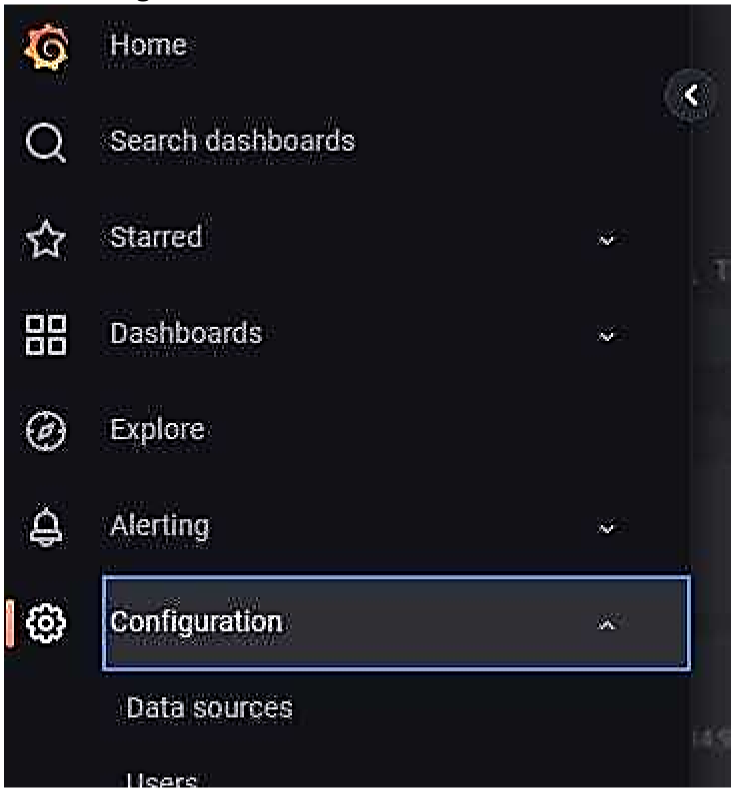
node_exporter (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://[redacted]:9100/metrics	UP	instance="[redacted]:9100" job="node_exporter"	14.869s ago	5.495ms	

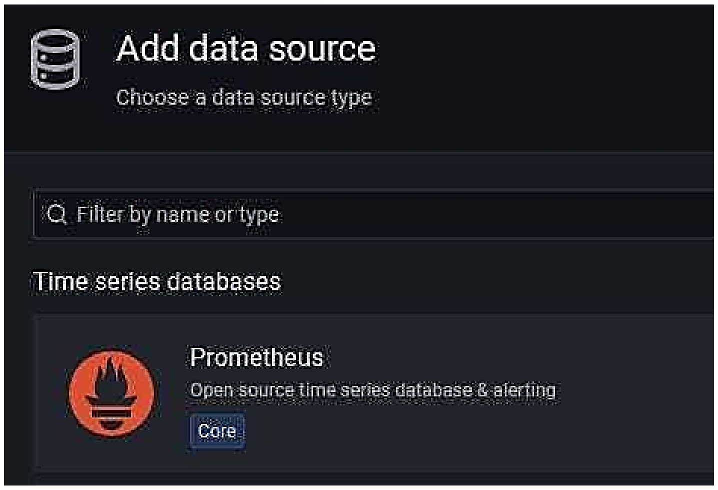
prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://[redacted]:9090/metrics	UP	instance="[redacted]:9090" job="prometheus"	14.595s ago	5.178ms	

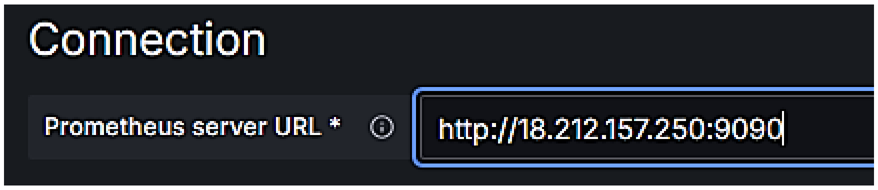
35. Back to Grafana. Go to Configuration and click Data sources.



36. Click Add data source, search Prometheus, and click Prometheus to settings Prometheus.



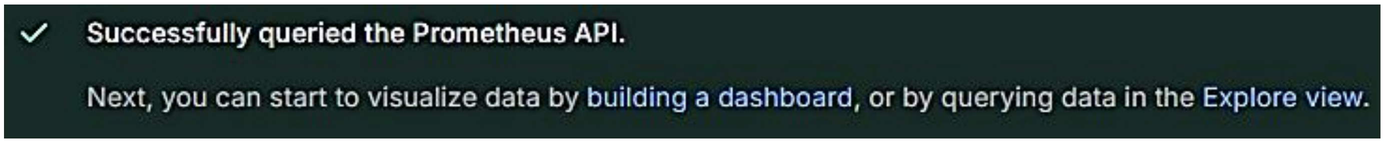
37. Fill in the URL-based Prometheus link



Connection

Prometheus server URL * http://18.212.157.250:9090

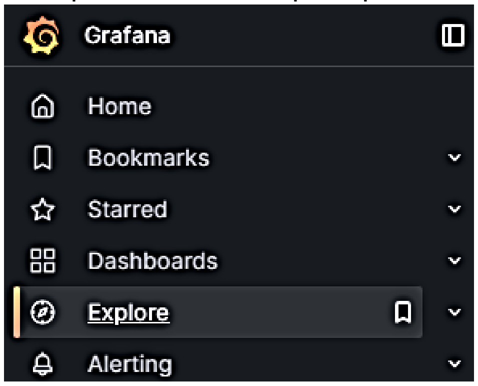
38. Click Save & test. If successful, show a notification Data source is working. Then go to Explore section.

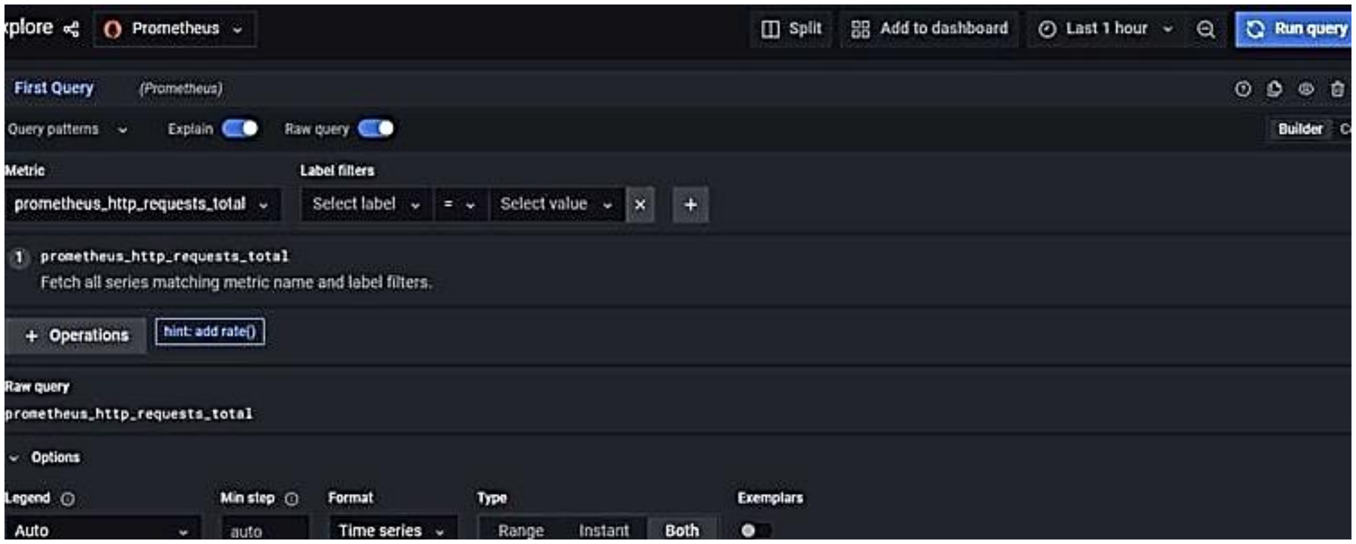


Successfully queried the Prometheus API.

Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

39. Explore means can focus on the query without thinking about visualization. Focus on the metrics you want to display. Try metric — prometheus_http_requests_total and click Run query.



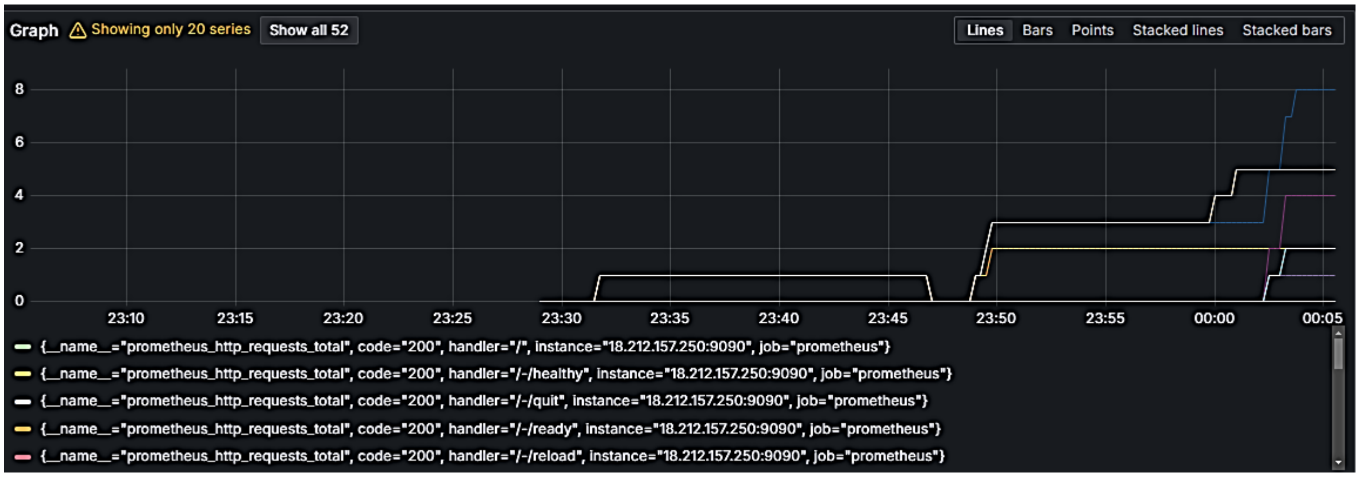


Metric: prometheus_http_requests_total

```
1 prometheus_http_requests_total
Fetch all series matching metric name and label filters.
```

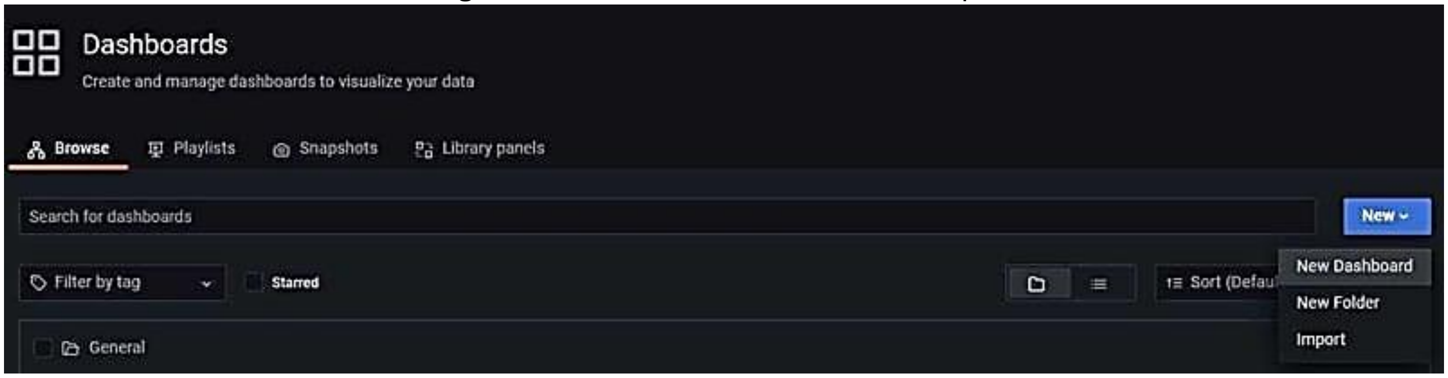
Raw query

```
prometheus_http_requests_total
```

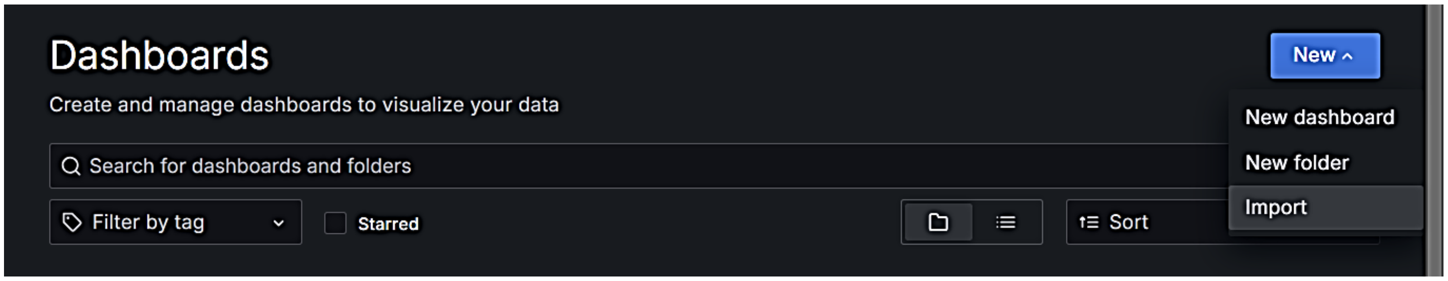


```
{__name__="prometheus_http_requests_total", code="200", handler="/", instance="18.212.157.250:9090", job="prometheus"}
{__name__="prometheus_http_requests_total", code="200", handler="/-/healthy", instance="18.212.157.250:9090", job="prometheus"}
{__name__="prometheus_http_requests_total", code="200", handler="/-/quit", instance="18.212.157.250:9090", job="prometheus"}
{__name__="prometheus_http_requests_total", code="200", handler="/-/ready", instance="18.212.157.250:9090", job="prometheus"}
{__name__="prometheus_http_requests_total", code="200", handler="/-/reload", instance="18.212.157.250:9090", job="prometheus"}
```

40. If want create dashboard, go to dashboard section and click Import.

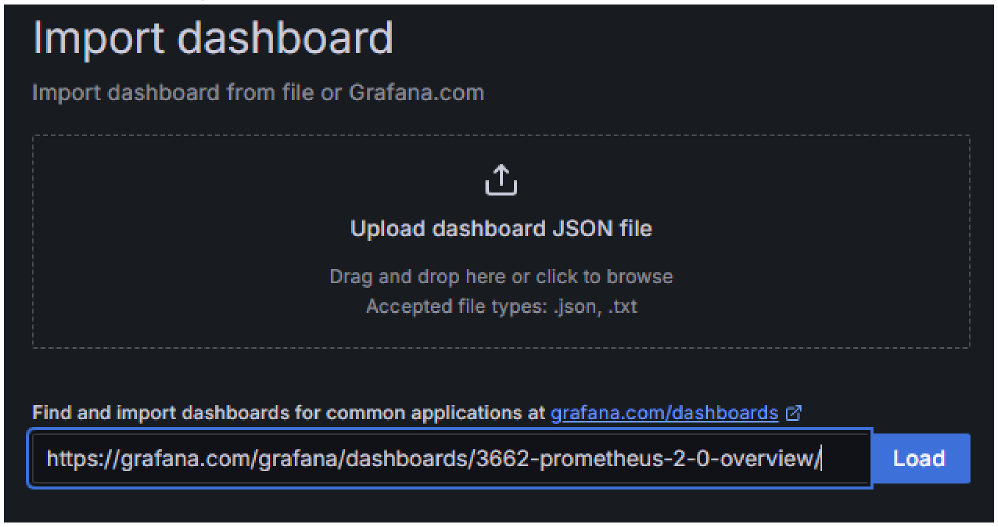


Importing a dashboard is fast step than creating a new dashboard, but instead, creating a new dashboard that can see the metrics that needs. Import dashboard can upload JSON files or via Grafana.com dashboard.

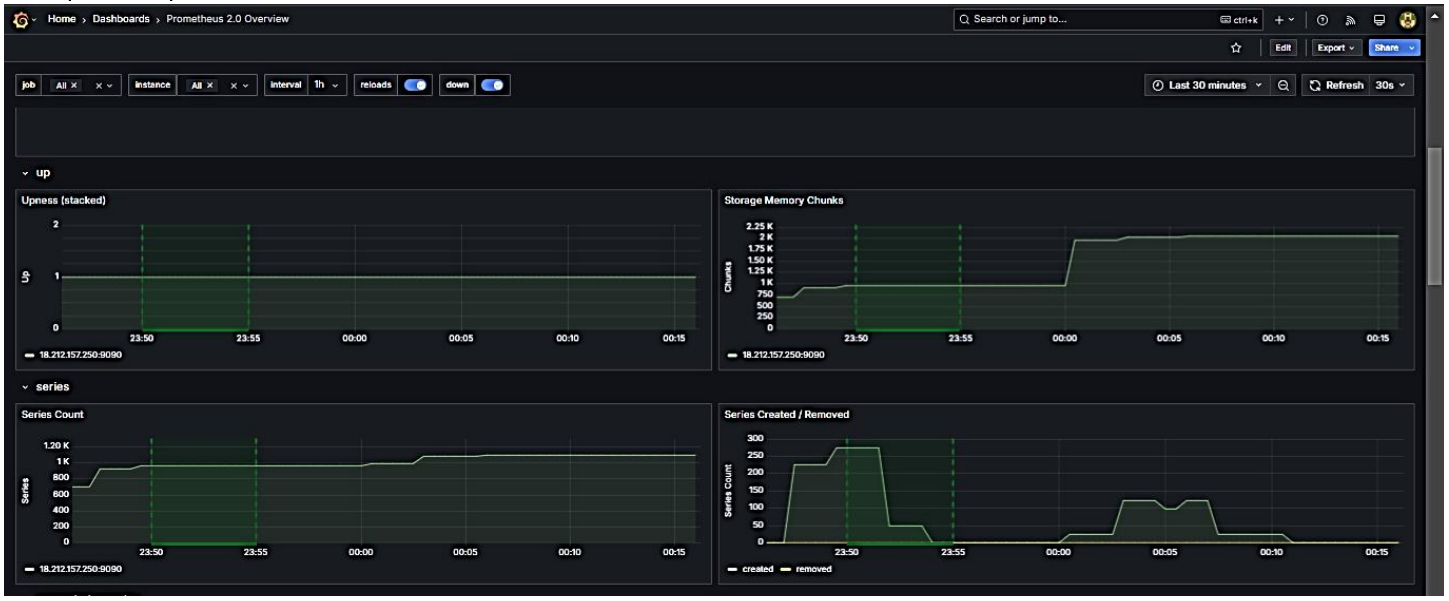

Dashboards
Create and manage dashboards to visualize your data
New
Search for dashboards and folders
New dashboard
New folder
Import
Filter by tag
Starred
Sort

Import via Grafana.com on this link:
https://grafana.com/grafana/dashboards/3662-prometheus-2-0-overview/

click Copy ID to the dashboard, and click Load.


Import dashboard
Import dashboard from file or Grafana.com
Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt
Find and import dashboards for common applications at grafana.com/dashboards
https://grafana.com/grafana/dashboards/3662-prometheus-2-0-overview/
Load

Sample Output:



- ✓ EC2 setup with Amazon Linux 2023
- ✓ Prometheus installed & configured (http://your-ec2-public-ip:9090)
- ✓ Node Exporter installed & running (http://your-ec2-public-ip:9100/metrics)
- ✓ Grafana installed & configured (http://your-ec2-public-ip:3000)
- ✓ Prometheus added as a Grafana data source
- ✓ Prometheus Dashboard imported in Grafana

Introduction to PromQL (Prometheus Query Language)

PromQL (Prometheus Query Language) is used to query and retrieve metrics from Prometheus. It allows you to filter, aggregate, and transform time-series data.

Basic Data Types in PromQL

PromQL supports four main types of queries:

✓ Instant Vector	A single timestamp with multiple metric values.
✓ Range Vector	A series of values over a time range.
✓ Scalar	A single numerical value.
✓ String	Not commonly used.

Querying Basic Metrics

Get All Available Metrics

Run this in Prometheus Web UI (http://your-ec2-public-ip:9090 → Graph tab):

```
{__name__=~".*"}
```

*This returns all available metric names in your Prometheus instance.

Get a Specific Metric

Retrieve CPU usage metrics from Node Exporter:

```
node_cpu_seconds_total
```

*This returns all time series for node_cpu_seconds_total.

Filter by Label

To get only CPU usage of core 0, use:

```
node_cpu_seconds_total{cpu="0"}
```

To get metrics for a specific instance (server):

```
node_cpu_seconds_total{instance="localhost:9100"}
```

Filter with Regular Expressions

Get CPU metrics excluding core 0:

```
node_cpu_seconds_total{cpu!="0"}
```

Match multiple values using regex:

```
node_cpu_seconds_total{cpu=~"0|1"}
```

Aggregation Functions

Prometheus allows aggregation of data over all instances, CPUs, or any other labels.

SUM: Total CPU Usage

```
sum(node_cpu_seconds_total)
```

*This returns the total CPU usage across all cores.

AVG: Average CPU Usage

```
avg(node_cpu_seconds_total)
```

MAX: Maximum CPU Usage

```
max(node_cpu_seconds_total)
```

MIN: Minimum CPU Usage

```
min(node_cpu_seconds_total)
```

COUNT: Number of Series Matching a Query

```
count(node_cpu_seconds_total)
```

*Returns the number of unique time series.

Rate and Increase Functions

RATE: Compute Per-Second Change
For CPU usage per second:

```
rate(node_cpu_seconds_total[5m])
```

*This calculates the per-second change over the last 5 minutes.

INCREASE: Compute the Total Increase Over Time
For total CPU usage increase over the last hour:

```
increase(node_cpu_seconds_total[1h])
```

Math Operations
PromQL allows basic math operations on metrics.

Convert CPU Time to Percentage

```
100 * (sum(rate(node_cpu_seconds_total[5m])) by (instance) / count(node_cpu_seconds_total))
```

Memory Usage in GB

```
If node_memory_MemTotal_bytes is total memory and node_memory_MemFree_bytes is free memory:
(node_memory_MemTotal_bytes - node_memory_MemFree_bytes) / 1024 / 1024 / 1024
```

*This converts bytes to GB.

Combining Metrics with by() and without()
Group by Instance

```
sum(rate(node_cpu_seconds_total[5m])) by (instance)
```

*Shows CPU usage per server.

Exclude Labels

```
sum(rate(node_cpu_seconds_total[5m])) without (cpu)
```

*Removes cpu labels and sums everything else.

Working with Time Ranges
Last 5 Minutes Data

```
node_load1[5m]
```

*Fetches 5-minute range vectors.

1-Hour Average CPU Usage

```
avg_over_time(node_cpu_seconds_total[1h])
```

Max Memory Usage Over 10 Minutes

```
max_over_time(node_memory_MemAvailable_bytes[10m])
```

Alerting with PromQL

To trigger an alert when CPU load is over 80%:

```
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
```

Useful Queries for Node Exporter

CPU Utilization per Instance

```
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

Memory Usage Percentage

```
100 * (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes))
```

Disk Space Used

```
100 * (1 - (node_filesystem_avail_bytes / node_filesystem_size_bytes))
```

Network Traffic Rate

```
rate(node_network_receive_bytes_total[5m])
```