# Configuring and Visualizing Metrics with Prometheus and Grafana

Estimated Time: 30 minutes

Welcome to the **Configuring and Visualizing Metrics with Prometheus and Grafana** lab. In this lab, you will become familiar with using Prometheus to monitor sample servers simulated with node exporters. Node exporters simulate server applications that serve metrics through the **/metrics** endpoint. You will use Prometheus to monitor the target node_exporter application by scraping its metrics endpoints, and you will also understand the Grafana UI.

## Learning Objectives

After completing this exercise, you should be able to perform the following tasks:

- Create queries to get the metrics about the target
- Find the status of the targets
- Find some information about the targets and visualize it with graphs
- Create and customize graphs in Prometheus to visualize the metrics and status of monitored targets.
- Configure and launch Grafana to source data from Prometheus.

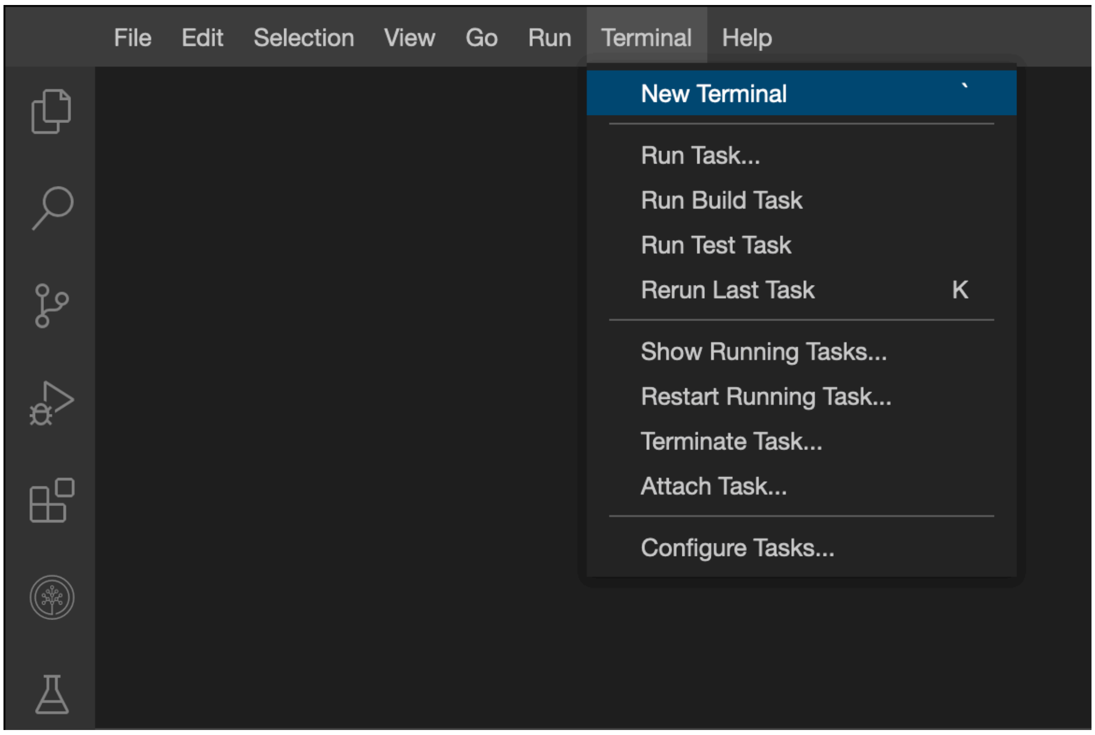
# Prerequisites

## Pre-requisites

This lab uses Docker to run special Node Exporters which will behave like servers that you can monitor. As a prerequisite, you will pull down the `bitnami/node-exporter` image from Docker Hub. You will use this image to make three instances of node exporters.

## Your Task

1. Let's start by opening up a new terminal. From the top menu, go to **Terminal** and choose **New Terminal** to open a new terminal window.



2. Use the following `docker pull` command to pull down the `bitnami/node-exporter` image from Docker Hub that you will use to simulate 3 servers being monitored.

```
docker pull bitnami/node-exporter:latest
```

You are now ready to start the lab.

# Step 1: Start the first node exporter

Now you need some server nodes to monitor. You will start up three node exporters listening on ports `9101`, `9102`, and `9103` respectively. In this step you will just start the first one and ensure that it is working correctly.

## Your Task

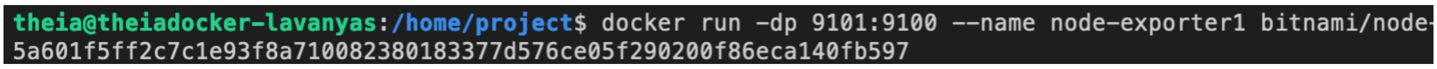
1. Run the following command to create a docker network within which we will run all the docker instances.

```
docker network create monitor
```

2. Run the following `docker run` command to start a node exporter instance listening at port `9101`

```
docker run -d -p 9101:9100 --name node-exporter1 --network monitor bitnami/node-exporter:latest
```

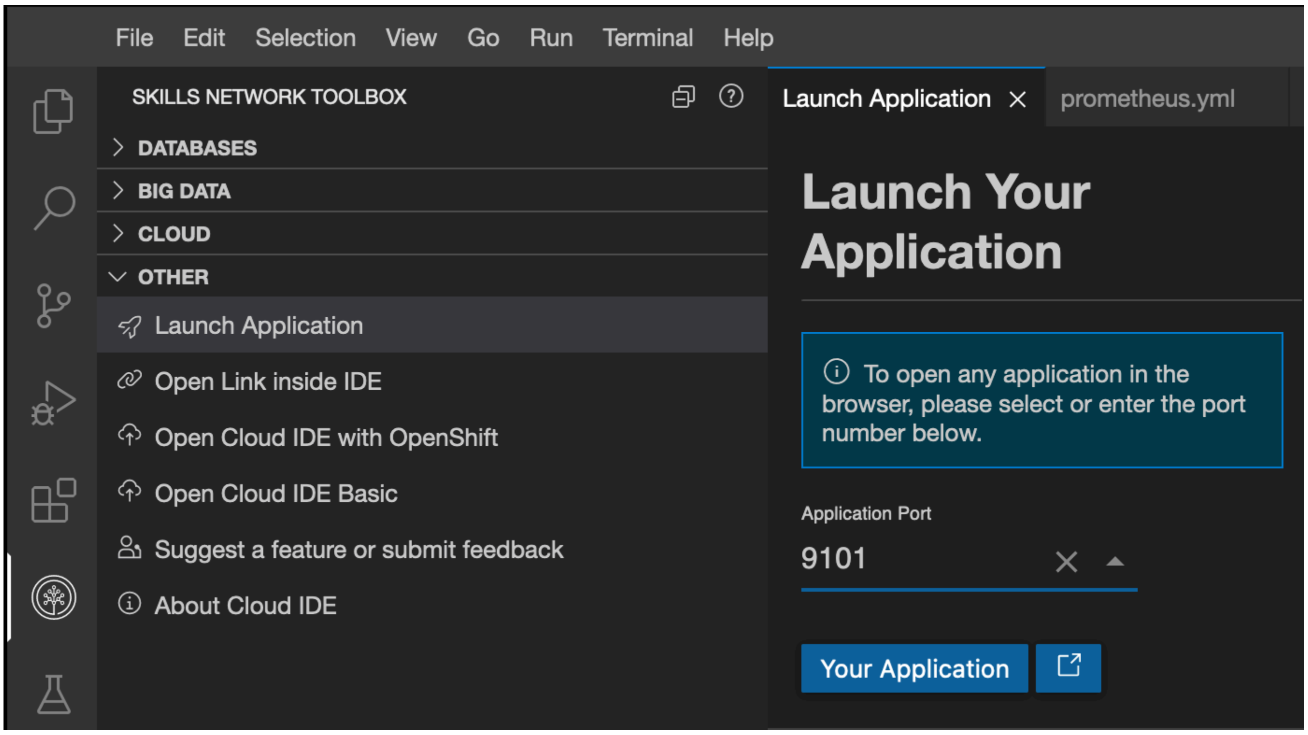
This will start a instance named node_exporter1 of node-exporter. The output should look like this:



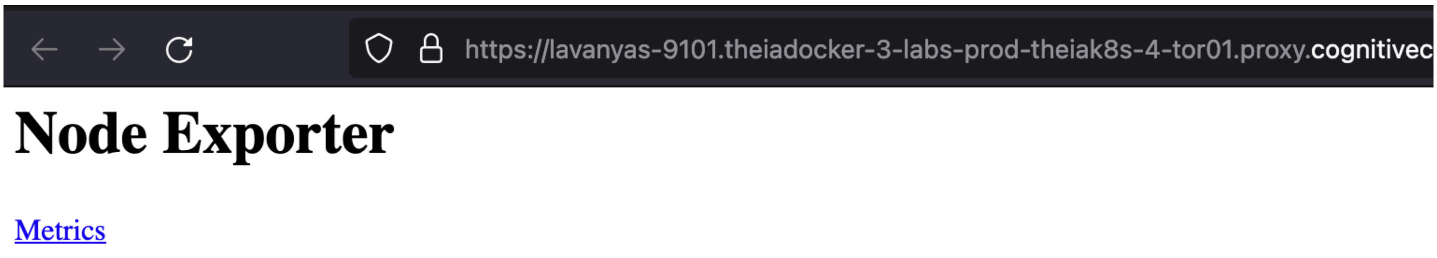
```
theia@theiadocker-lavanyas:/home/project$ docker run -dp 9101:9100 --name node-exporter1 bitnami/node-
5a601f5ff2c7c1e93f8a710082380183377d576ce05f290200f86eca140fb597
```

3. Next, check if the instance is running by either clicking on the **Skills Network Toolbox** and under **Others** choose **Launch Application**, enter the port number **9101** and click on the launch URL button. Or press the `[Launch Application]` button below:

Launch Application



4. The node exporter page opens up as below with a hyper link to **Metrics**. These are the metrics the Prometheus instance is going to monitor.

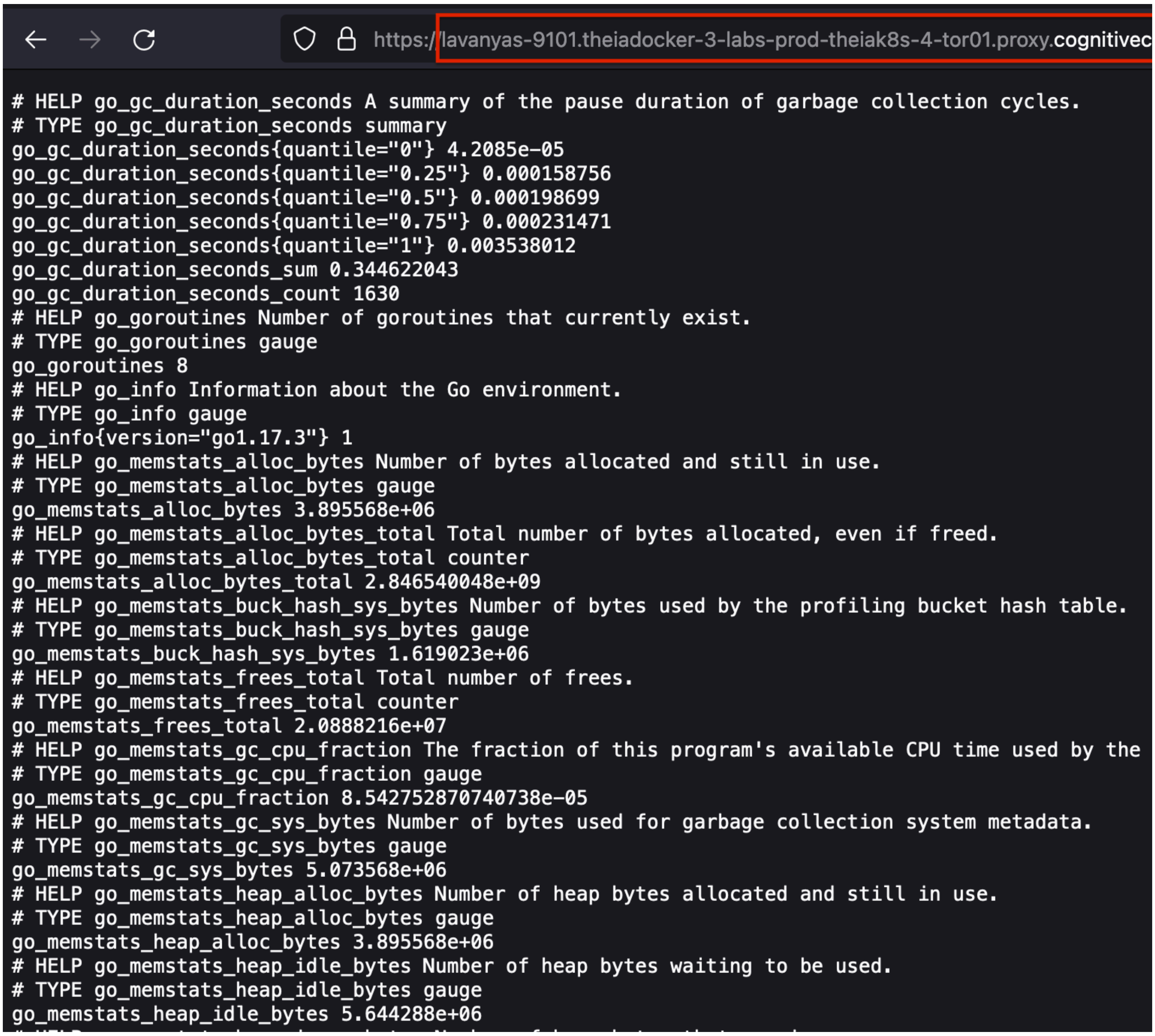


https://lavanyas-9101.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitivec

Node Exporter

Metrics

5. Click on the **Metrics** link to have a glimpse of the metrics.


https://lavanyas-9101.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitivec

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.2085e-05
go_gc_duration_seconds{quantile="0.25"} 0.000158756
go_gc_duration_seconds{quantile="0.5"} 0.000198699
go_gc_duration_seconds{quantile="0.75"} 0.000231471
go_gc_duration_seconds{quantile="1"} 0.003538012
go_gc_duration_seconds_sum 0.344622043
go_gc_duration_seconds_count 1630
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 3.895568e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 2.846540048e+09
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.619023e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 2.0888216e+07
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 8.542752870740738e-05
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 5.073568e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 3.895568e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 5.644288e+06
```

6. Copy the URL marked in red in the above image and save it in a notepad. You will use this URL to configure prometheus to monitor the node exporter.

## Step 2: Start two more node exporters

Now that you have one node exporter working, let's start two more so that Prometheus has three nodes to monitor in total. You will do this exactly the same way as you did the first node exporter except that you will change the port numbers to `9102` and `9103` respectively.

### Your Task

1. In the terminal, now run the following commands to start two more instances of node exporter.

```
docker run -d -p 9102:9100 --name node-exporter2 --network monitor bitnami/node-exporter:latest
```

and

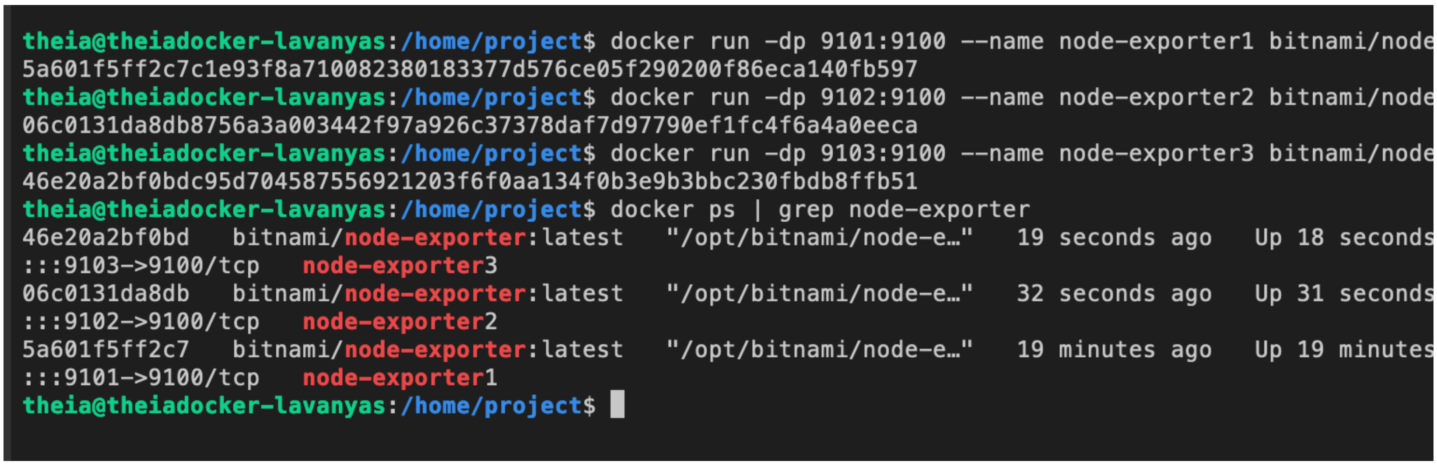
```
docker run -d -p 9103:9100 --name node-exporter3 --network monitor bitnami/node-exporter:latest
```

2. Now, check if all the instances of node exporter are running by using the `docker ps` command and grepping for `node-exporter`.

```
docker ps | grep node-exporter
```

### Results

If everything started correctly, you should see output similar to the following coming back from the `docker` commands:



```
theia@theiadocker-lavanyas:/home/project$ docker run -dp 9101:9100 --name node-exporter1 bitnami/node
5a601f5ff2c7c1e93f8a710082380183377d576ce05f290200f86eca140fb597
theia@theiadocker-lavanyas:/home/project$ docker run -dp 9102:9100 --name node-exporter2 bitnami/node
06c0131da8db8756a3a003442f97a926c37378daf7d97790ef1fc4f6a4a0eeca
theia@theiadocker-lavanyas:/home/project$ docker run -dp 9103:9100 --name node-exporter3 bitnami/node
46e20a2bf0bdc95d704587556921203f6f0aa134f0b3e9b3bbc230fbdb8ffb51
theia@theiadocker-lavanyas:/home/project$ docker ps | grep node-exporter
46e20a2bf0bd   bitnami/node-exporter:latest   "/opt/bitnami/node-e…"   19 seconds ago   Up 18 seconds
:::9103->9100/tcp   node-exporter3
06c0131da8db   bitnami/node-exporter:latest   "/opt/bitnami/node-e…"   32 seconds ago   Up 31 seconds
:::9102->9100/tcp   node-exporter2
5a601f5ff2c7   bitnami/node-exporter:latest   "/opt/bitnami/node-e…"   19 minutes ago   Up 19 minutes
:::9101->9100/tcp   node-exporter1
theia@theiadocker-lavanyas:/home/project$
```

You are now ready to install and start Prometheus.

## Step 3: Download, configure and run Prometheus

Now that we have some nodes to monitor, it's time to run Prometheus to monitor the nodes. In this step you will also create a custom configuration file to tell Prometheus what nodes to monitor. Then you will start Prometheus passing it the configuration file to use.

### Your Task

1. First, pull the Prometheus into your local environment, by running the following `docker pull` command on your terminal.

   ```
   docker pull bitnami/prometheus:latest
   ```

   This may take a few seconds depending upon your internet connectivity

1. Create a file named **prometheus.yml** in the current directory. This is the file where you will configure the prometheus to monitor the node exporter instances.

   ```
   touch prometheus.yml
   ```

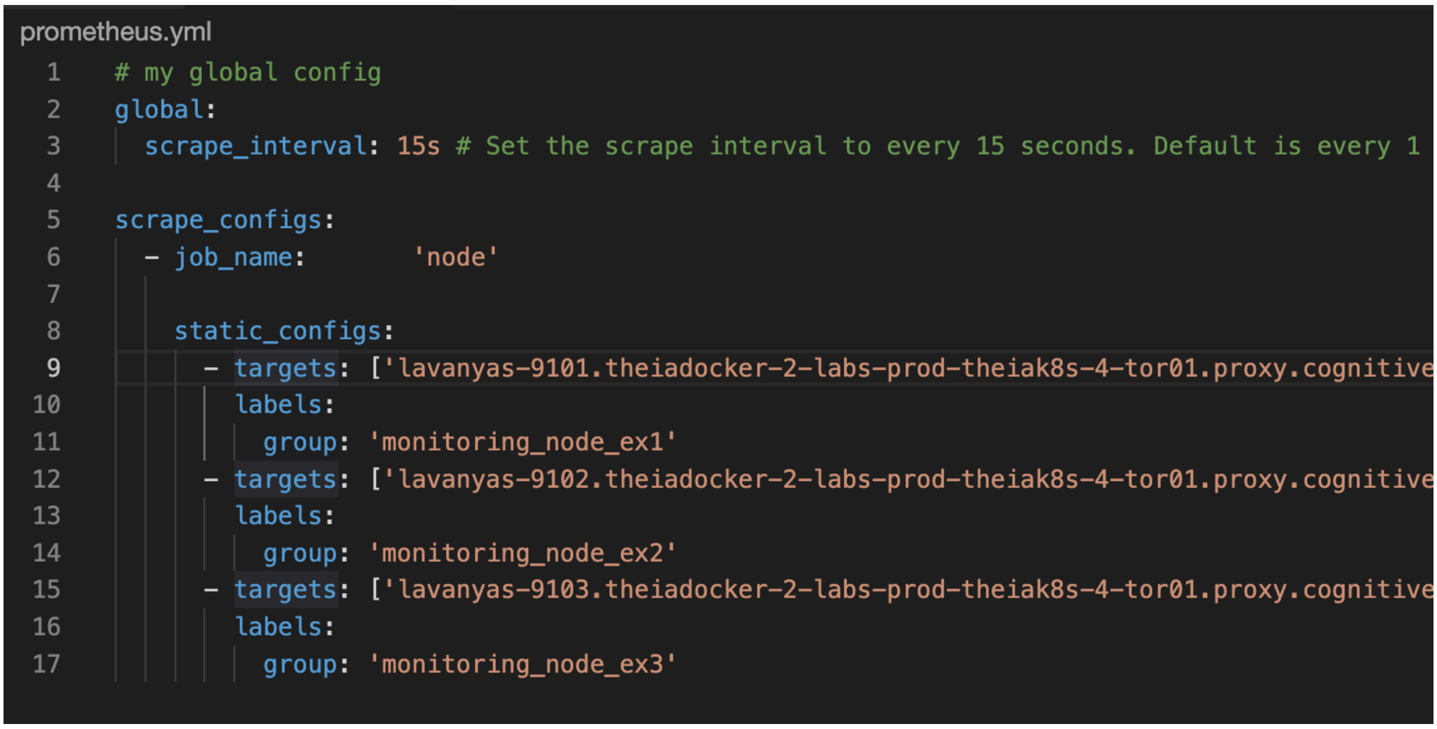
2. From the explorer, navigate to **prometheus.yml** to edit the file or press the `[Open prometheus.yml in IDE]` button below:

Open **prometheus.yml** in IDE

3. Add the content below to the file. Update the URLs appropriately. The node exporter 1 URL will be the one that you copied into a notepad in the previous task. `node exporter 2` will be the same URL with port number changed to 9102. `node exporter 3` will be the same URL with port number changed to 9103.

   ```
   # my global config
   global:
     scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
   scrape_configs:
     - job_name:       'node'
       static_configs:
         - targets: ['node-exporter1:9100']
           labels:
             group: 'monitoring_node_ex1'
         - targets: ['node-exporter2:9100']
           labels:
             group: 'monitoring_node_ex2'
         - targets: ['node-exporter2:9100']
           labels:
             group: 'monitoring_node_ex3'
   ```

A sample configuration can be seen in the image below.



prometheus.yml

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1

scrape_configs:
  - job_name:       'node'

    static_configs:
      - targets: ['lavanyas-9101.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitive
        labels:
          group: 'monitoring_node_ex1'
      - targets: ['lavanyas-9102.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitive
        labels:
          group: 'monitoring_node_ex2'
      - targets: ['lavanyas-9103.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitive
        labels:
          group: 'monitoring_node_ex3'
```

4. Now you can launch the prometheus monitor by passing this yaml file as a parameter.

```
docker run --rm --name prometheus -p 9090:9090 --network monitor \
-v $(pwd)/prometheus.yml:/opt/bitnami/prometheus/conf/prometheus.yml \
bitnami/prometheus:latest
```

## Results

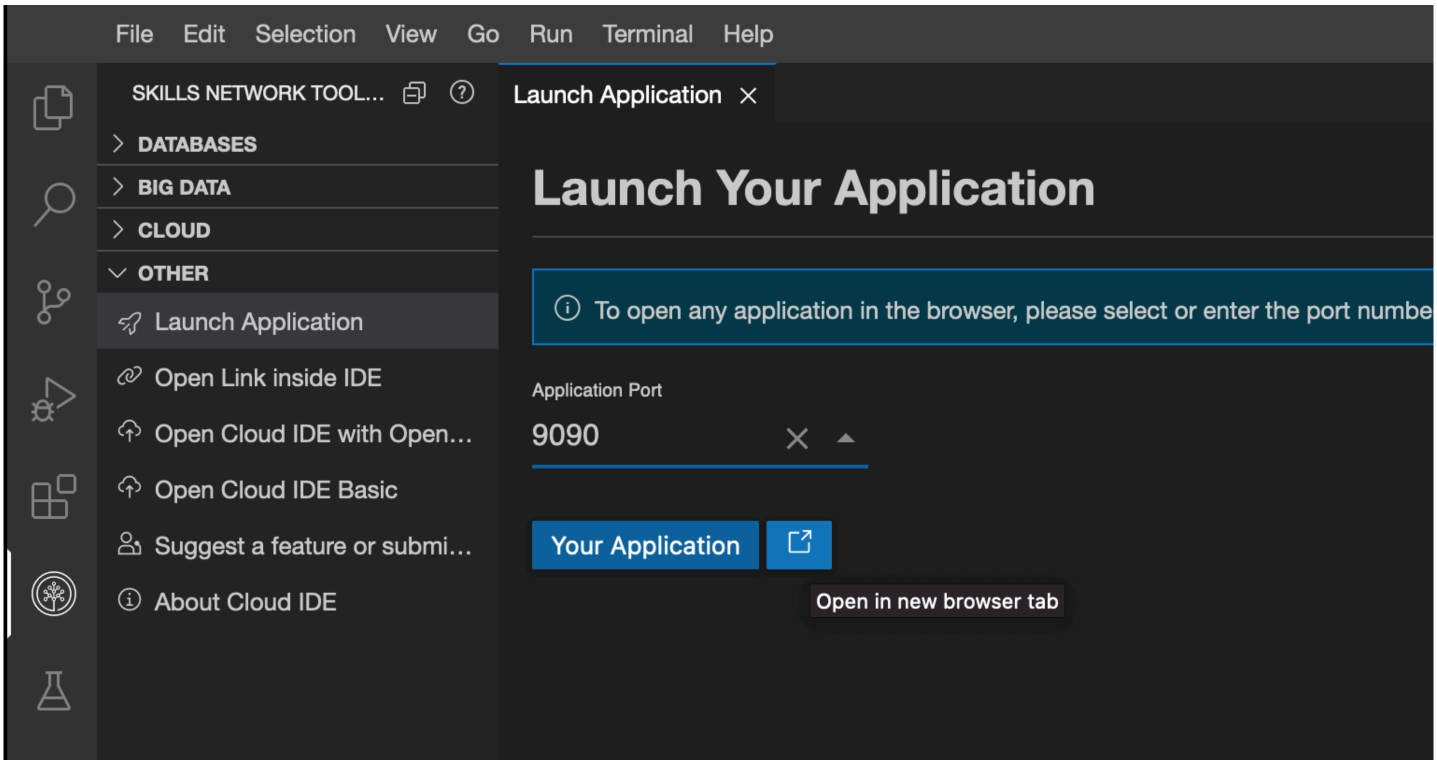
You should see the Prometheus logs scroll up the screen which indicate that the monitoring has started.

## Step 4: Open the Prometheus UI

In this step you will launch the Prometheus web UI and navigate to the page where you an start executing queries.

1. Open the Prometheus web UI by click on the **Skills Network Toolbox** and under **Others** choose **Launch Application**, enter the port number **9090** and click on the launch URL button or press the Launch Prometheus button below to launch in an external browser:

Launch Prometheus



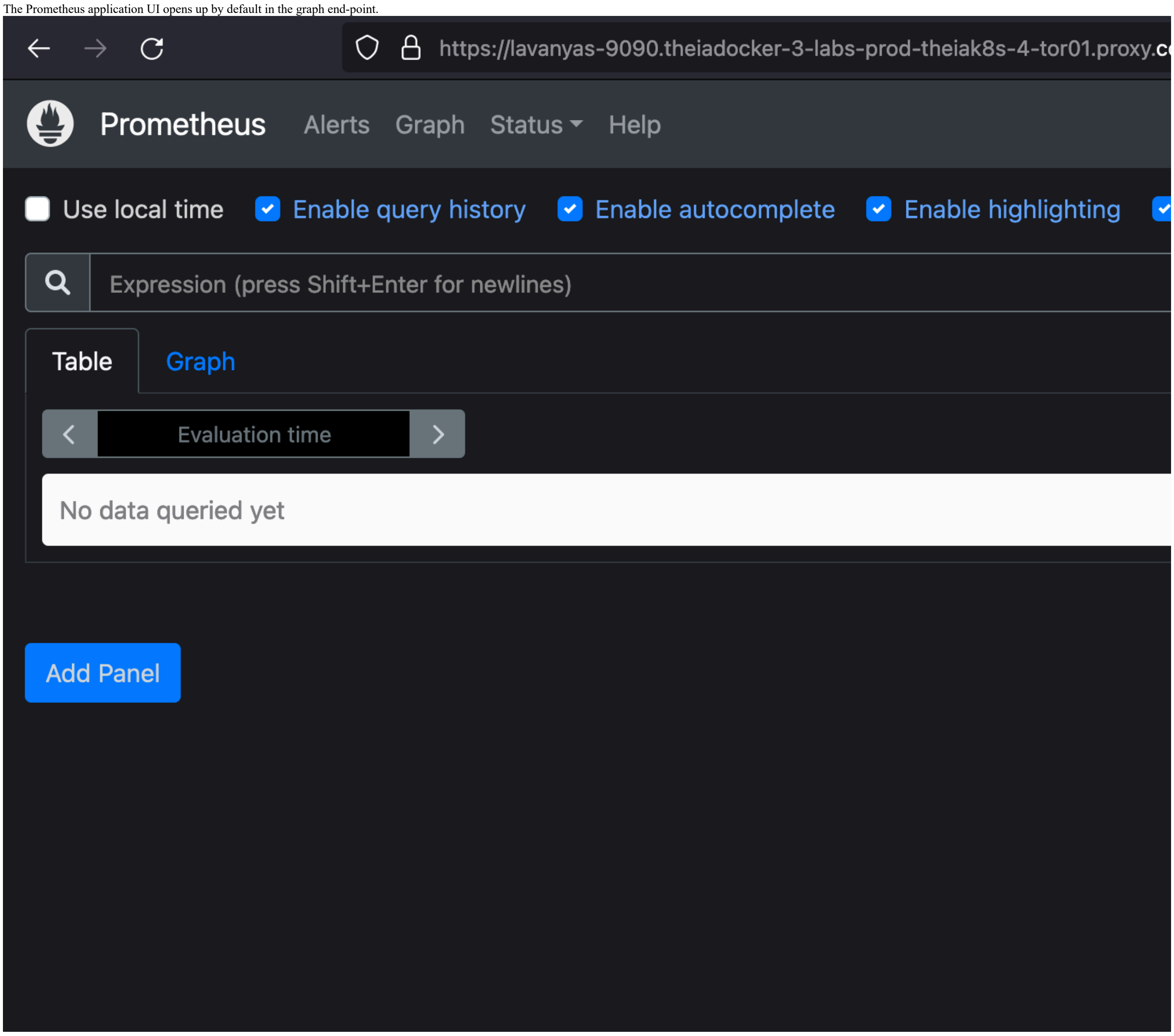
Launch Your Application

To open any application in the browser, please select or enter the port numbe

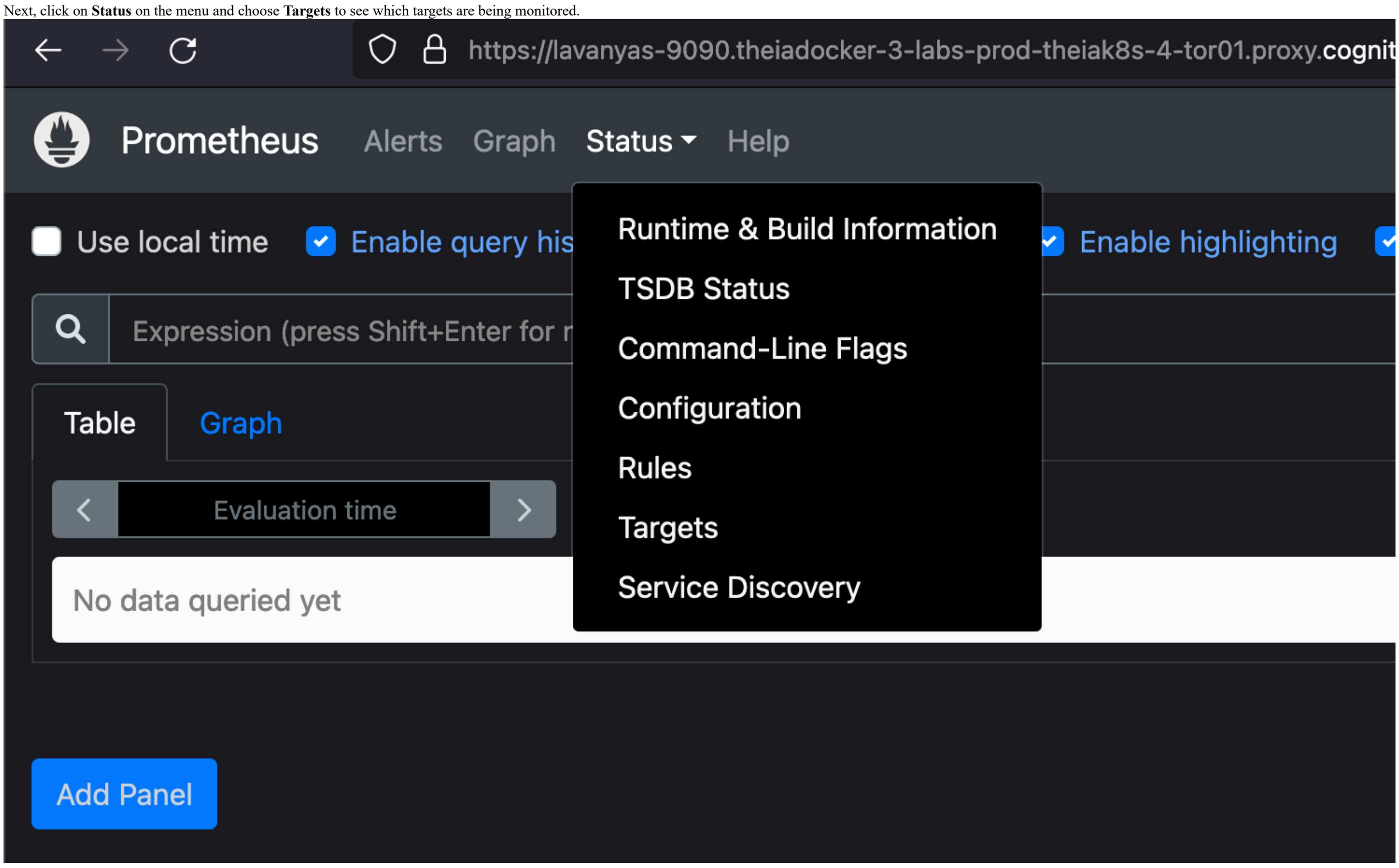
Application Port

9090

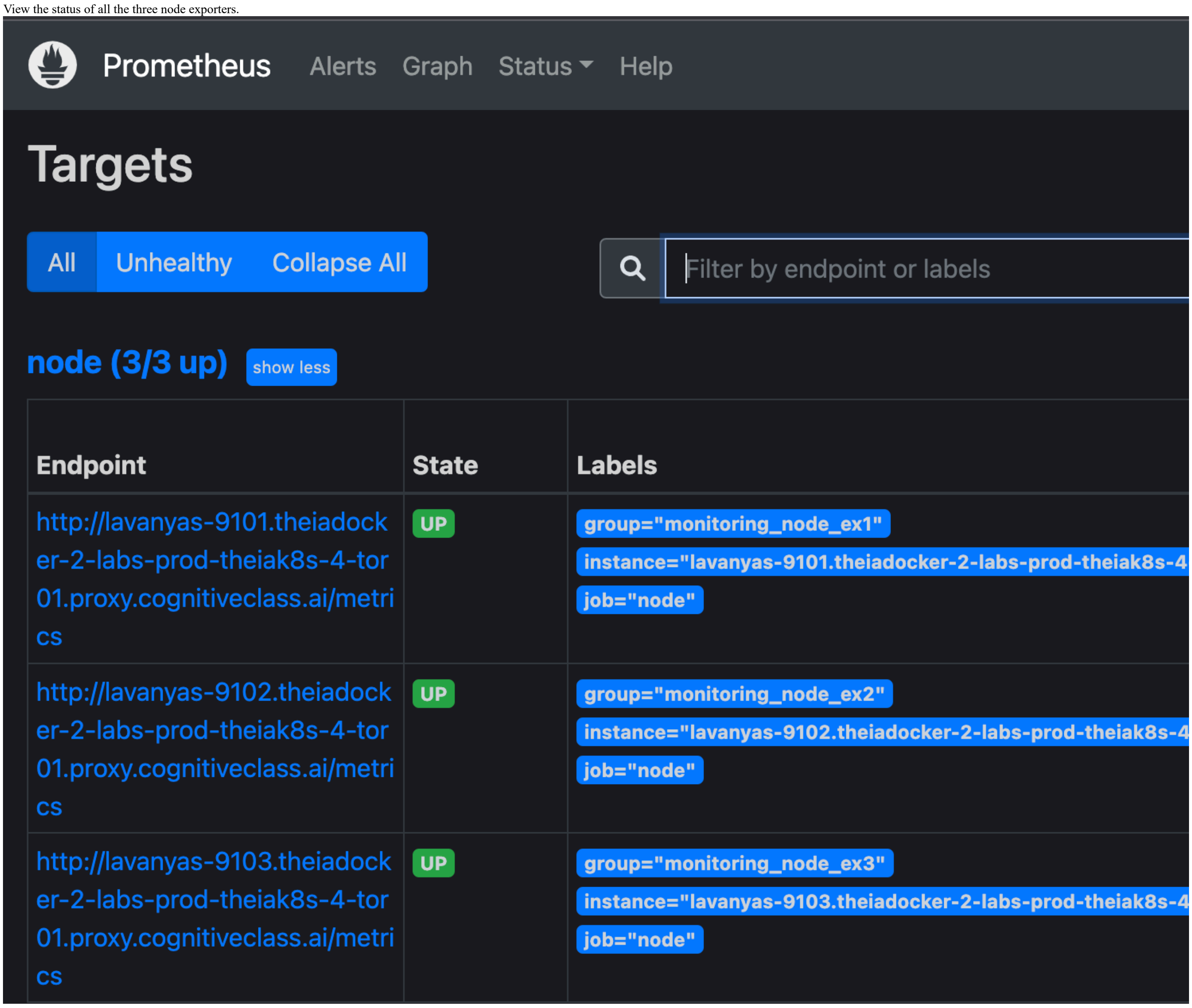
2. The Prometheus application UI opens up by default in the graph end-point.


https://lavanyas-9090.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.c

3. Next, click on **Status** on the menu and choose **Targets** to see which targets are being monitored.



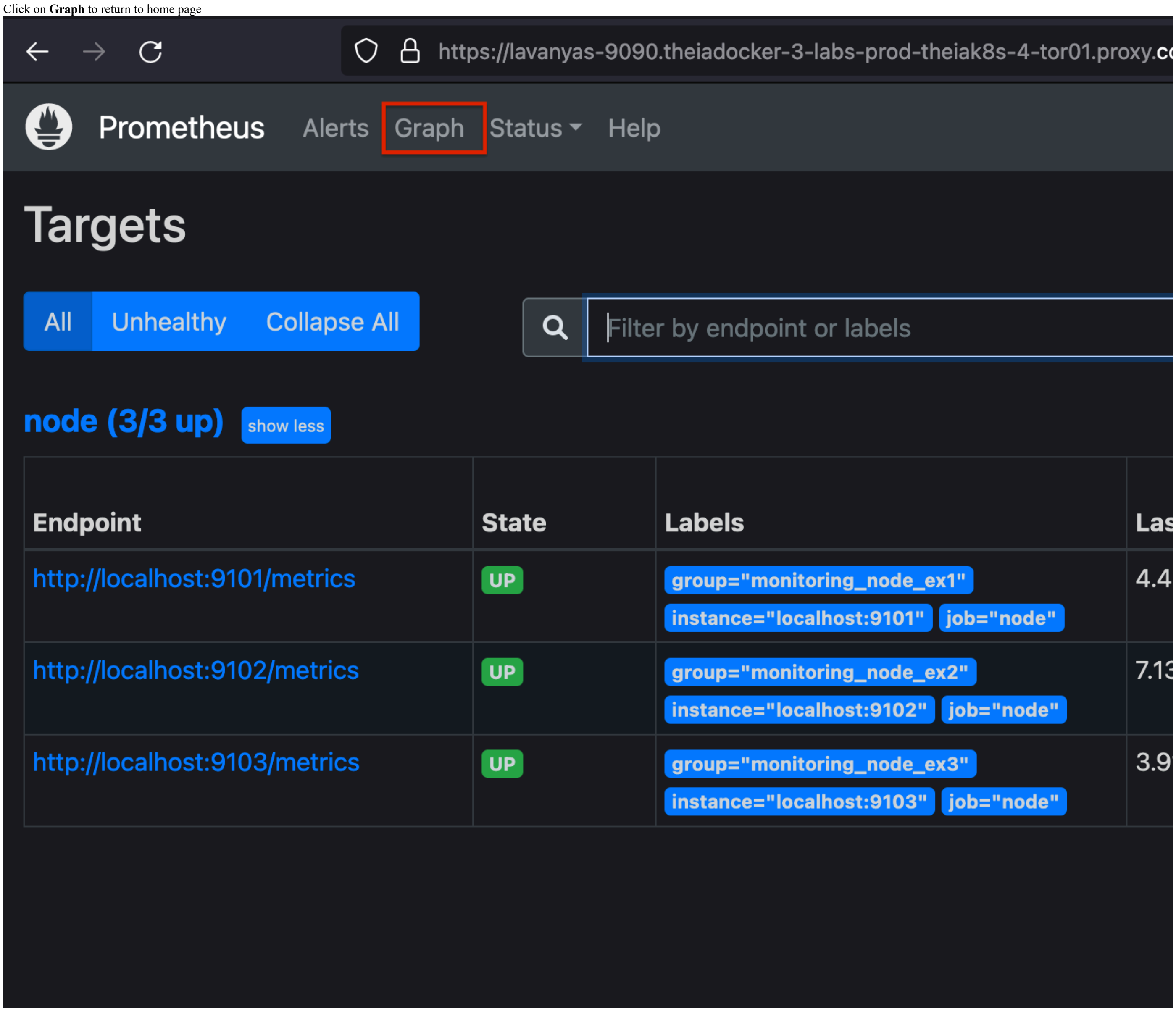
4. View the status of all the three node exporters.



node (3/3 up)

| Endpoint | State | Labels |
|---|---|---|
| http://lavanyas-9101.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/metrics | UP | group="monitoring_node_ex1" instance="lavanyas-9101.theiadocker-2-labs-prod-theiak8s-4 job="node" |
| http://lavanyas-9102.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/metrics | UP | group="monitoring_node_ex2" instance="lavanyas-9102.theiadocker-2-labs-prod-theiak8s-4 job="node" |
| http://lavanyas-9103.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/metrics | UP | group="monitoring_node_ex3" instance="lavanyas-9103.theiadocker-2-labs-prod-theiak8s-4 job="node" |

5. Click on **Graph** to return to home page



https://lavanyas-9090.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.c

Targets

node (3/3 up)

| Endpoint | State | Labels | Las |
|---|---|---|---|
| http://localhost:9101/metrics | UP | group="monitoring_node_ex1" instance="localhost:9101" job="node" | 4.4 |
| http://localhost:9102/metrics | UP | group="monitoring_node_ex2" instance="localhost:9102" job="node" | 7.13 |
| http://localhost:9103/metrics | UP | group="monitoring_node_ex3" instance="localhost:9103" job="node" | 3.9 |

You are now ready to execute queries.

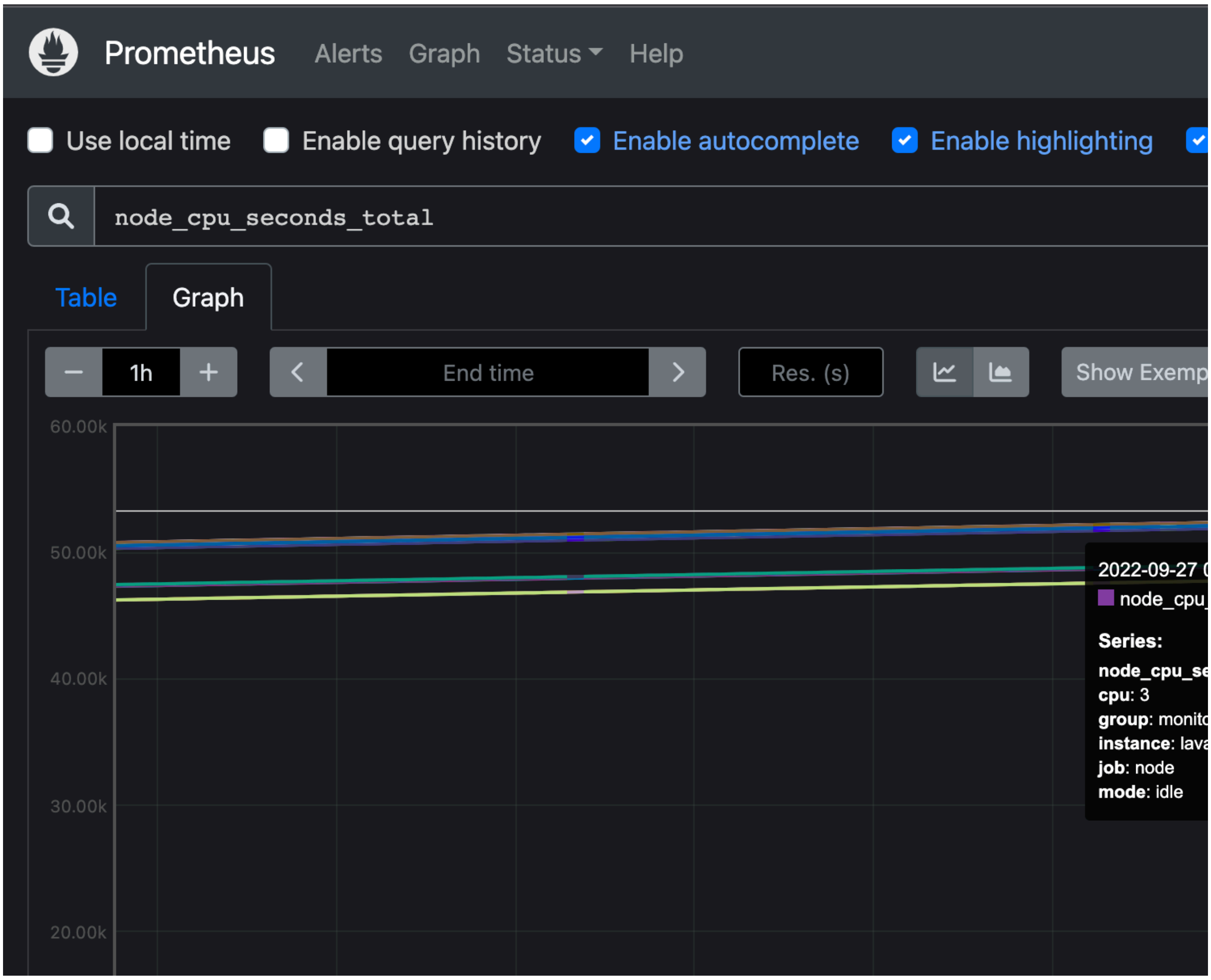
## Step 5: Execute your first query

You are now ready to execute your first query. Query for the total CPU second of the nodes by pasting the following query and executing it. It will show the graph as given in the image. You can observe the details for each instance by hovering the mouse over that instance.

### Your Task

1. Pasting the following query and executing it. It will show the graph as given in the image. You can observe the details for each instance by hovering the mouse over that instance.

```
node_cpu_seconds_total
```

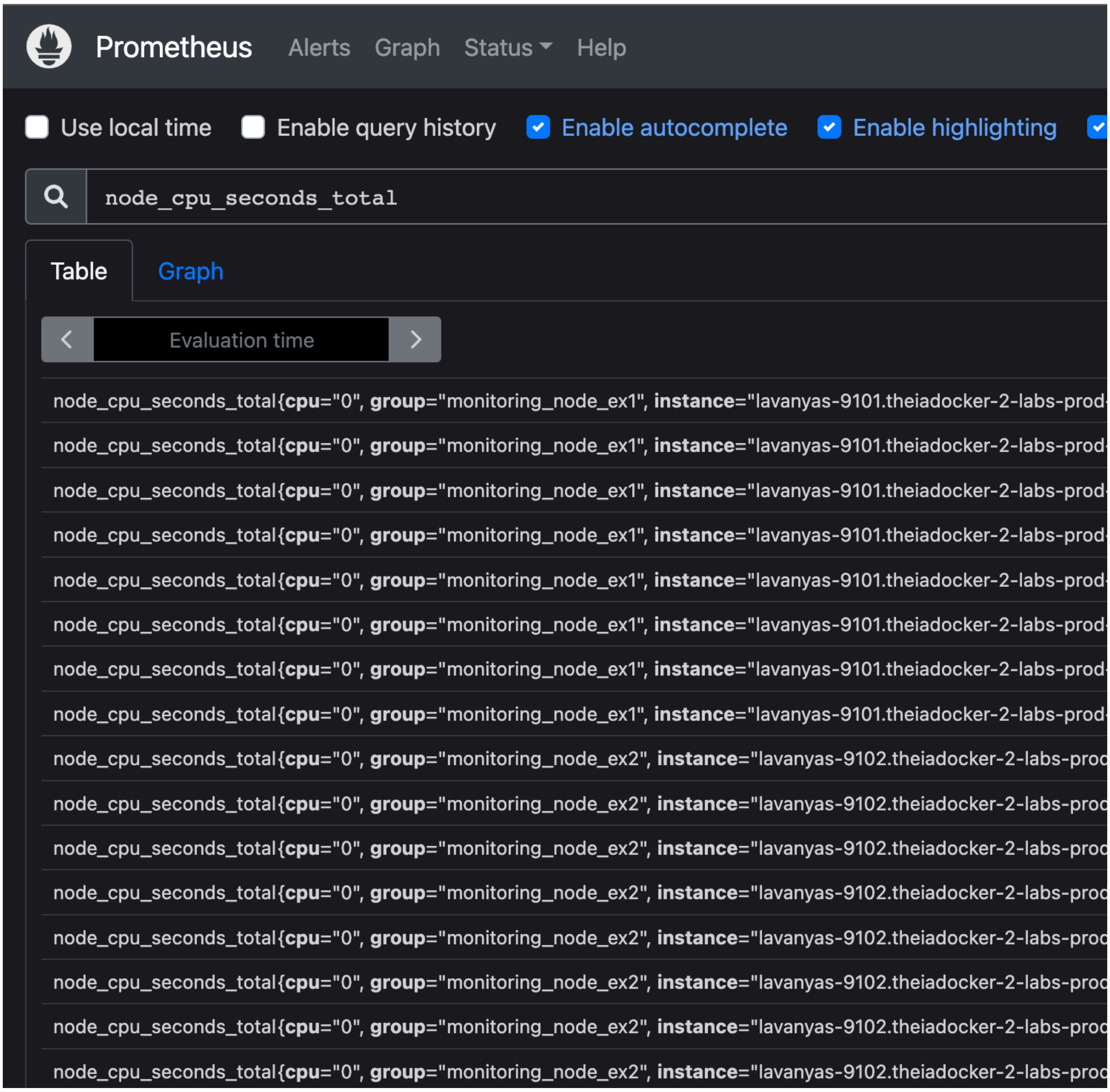


```
node_cpu_seconds_total
```

2022-09-27 0
node_cpu_

**Series:**
**node_cpu_se**
**cpu**: 3
**group**: monito
**instance**: lava
**job**: node
**mode**: idle

2. Next on **Table** and query the cpu second for all the targets in tabular format.


Use local time Enable query history Enable autocomplete Enable highlighting

```
node_cpu_seconds_total
```

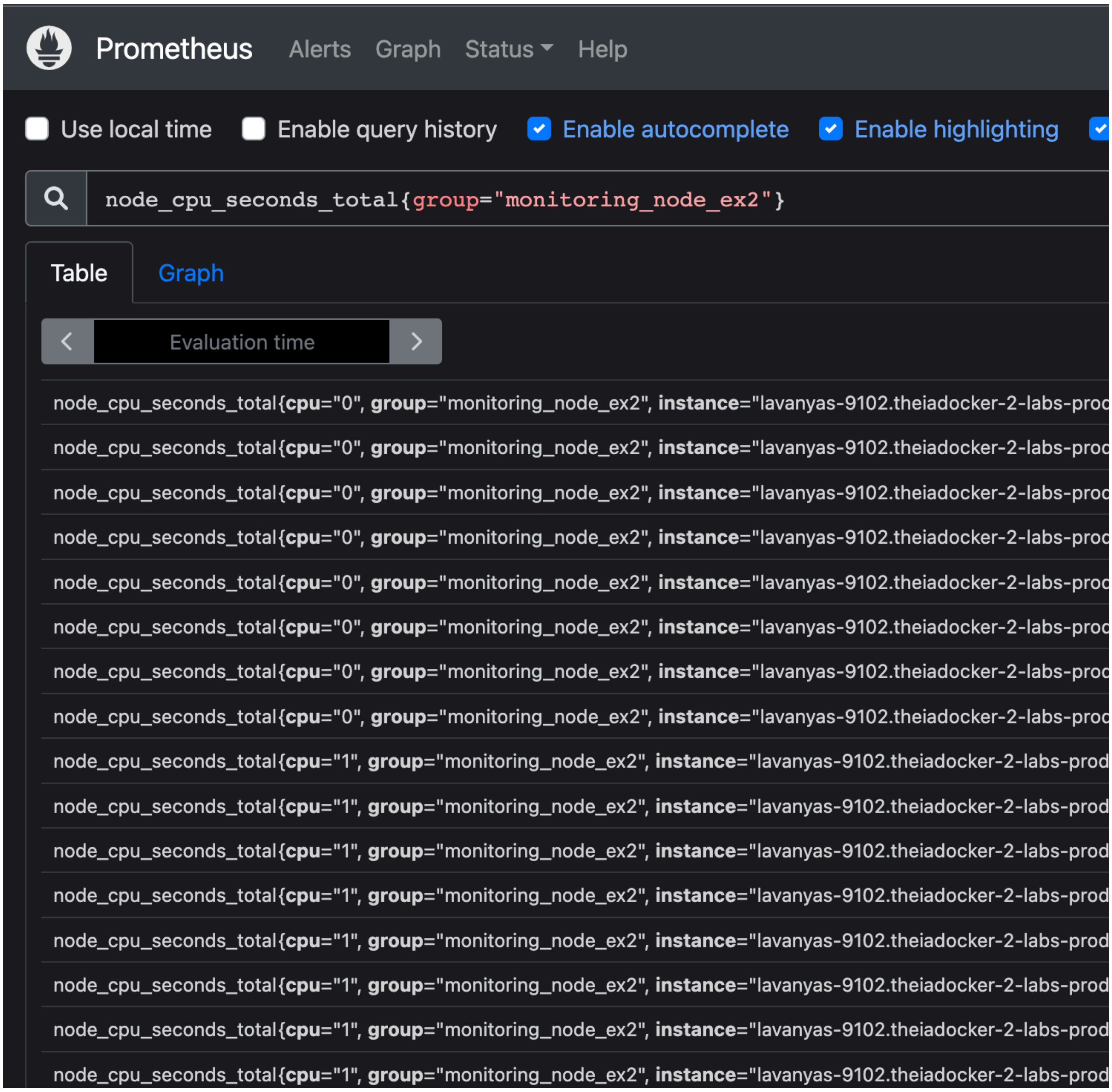
Table Graph

Evaluation time

```
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
```

3. Now, filter the query to get the details for only one instance using the following query.

```
node_cpu_seconds_total{group="monitoring_node_ex2"}
```



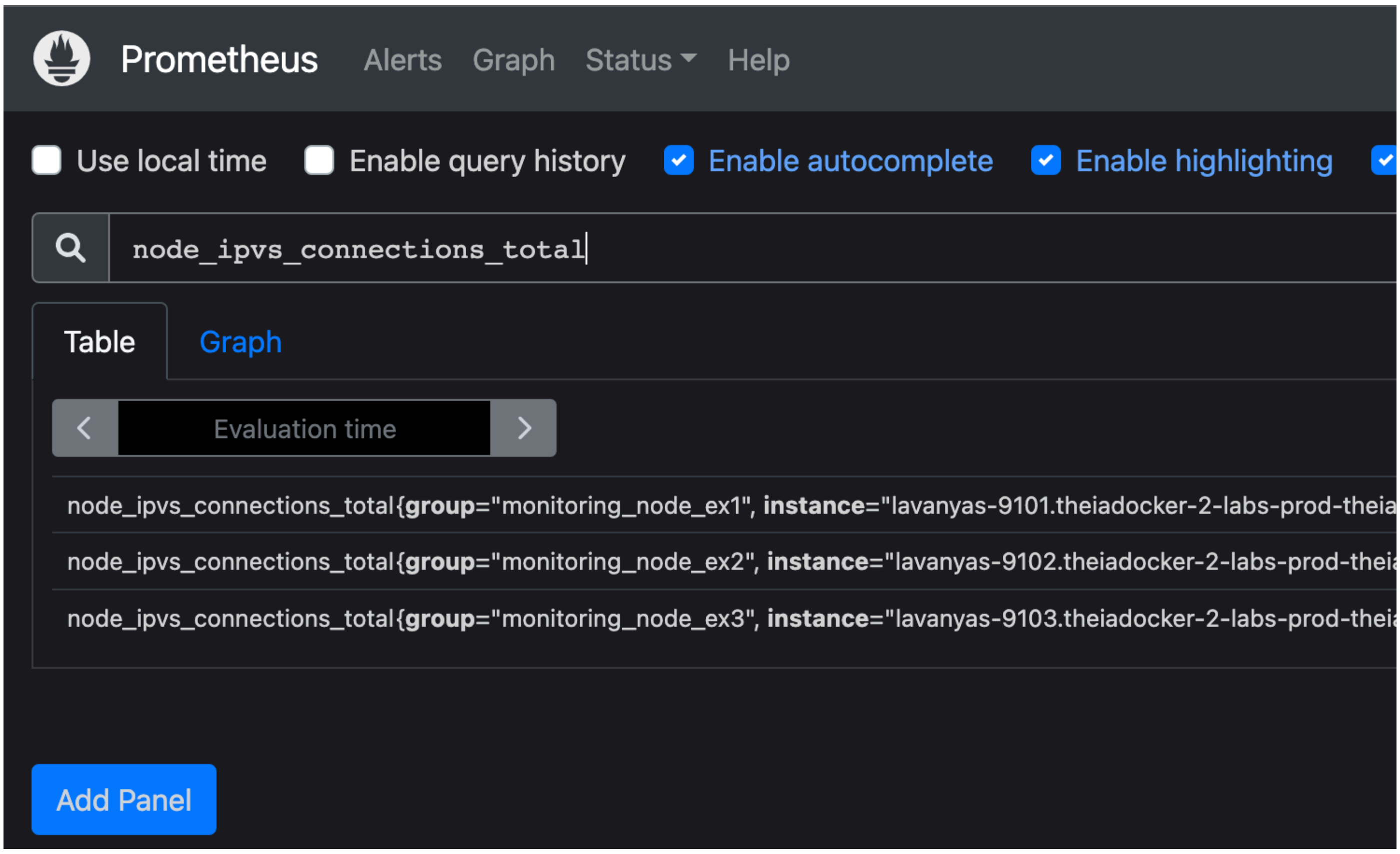
Use local time [✓ Enable autocomplete] [✓ Enable highlighting] Enable query history

```
node_cpu_seconds_total{group="monitoring_node_ex2"}
```

```
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="0", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-proc
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
node_cpu_seconds_total{cpu="1", group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod
```

4. Finally, query for the connections each node has using this query.

```
node_ipvs_connections_total
```



```
node_ipvs_connections_total
```

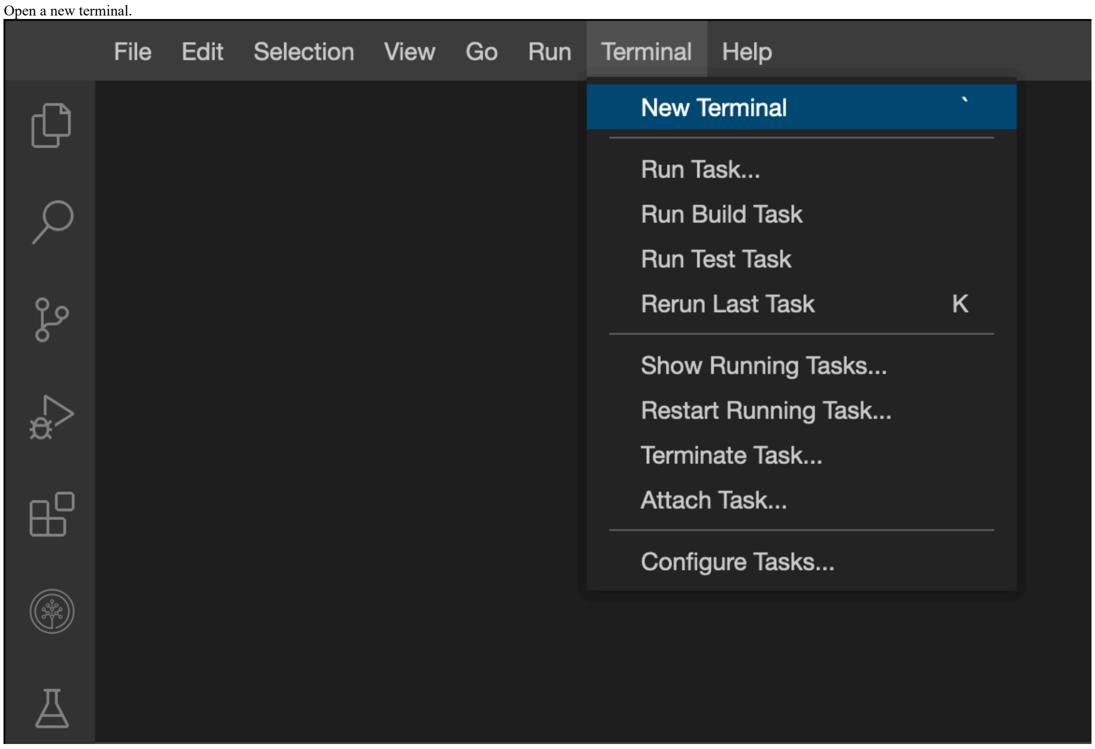
```
node_ipvs_connections_total{group="monitoring_node_ex1", instance="lavanyas-9101.theiadocker-2-labs-prod-theia
node_ipvs_connections_total{group="monitoring_node_ex2", instance="lavanyas-9102.theiadocker-2-labs-prod-thei
node_ipvs_connections_total{group="monitoring_node_ex3", instance="lavanyas-9103.theiadocker-2-labs-prod-thei
```

## Step 6: Stop and observe

In this step we will stop one of the node exporter instances and see how that is reflected in the Prometheus console.

### Your Task

1. Open a new terminal.



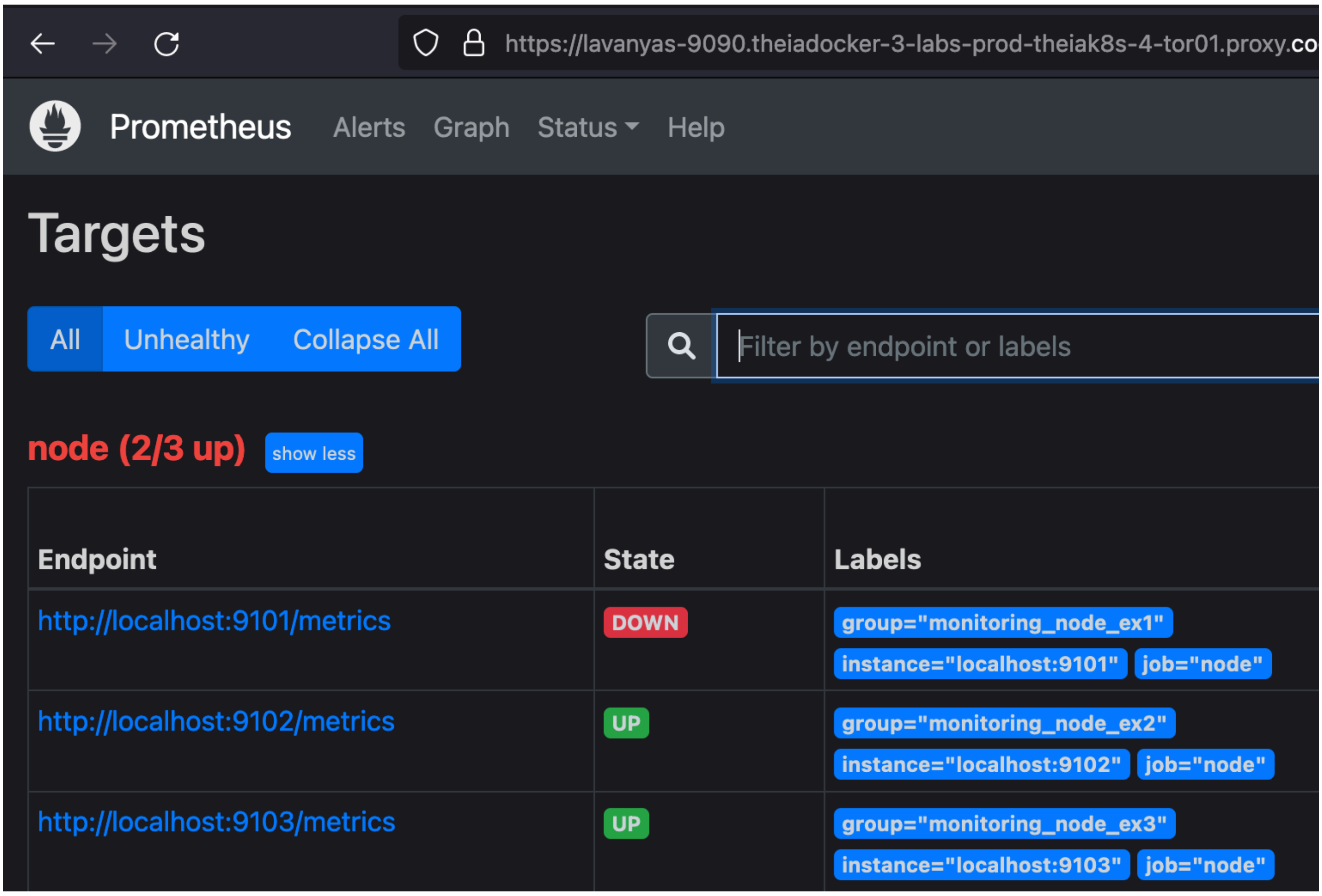
2. Stop the node-exporter1 instance by running the following `docker stop` command and then switch back to old terminal in which prometheus is running.

```
docker stop node-exporter1
```

3. Now go back to the prometheus UI on your browser and check the targets by selecting the menu item **Status -> Targets**

### Results

You should now see that one of the node exporters is being monitored is down. *Note: You may have to refresh your browser to see the new status*


https://lavanyas-9090.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.co
Prometheus
Alerts
Graph
Status
Help
All
Unhealthy
Collapse All
Filter by endpoint or labels
show less

Targets

node (2/3 up)

| Endpoint | State | Labels |
|---|---|---|
| http://localhost:9101/metrics | DOWN | group="monitoring_node_ex1" instance="localhost:9101" job="node" |
| http://localhost:9102/metrics | UP | group="monitoring_node_ex2" instance="localhost:9102" job="node" |
| http://localhost:9103/metrics | UP | group="monitoring_node_ex3" instance="localhost:9103" job="node" |

## Step 7: Start Grafana

1. Run the following command to pull Grafana from docker and run it. We will run it in the same network as node monitor and prometheus.

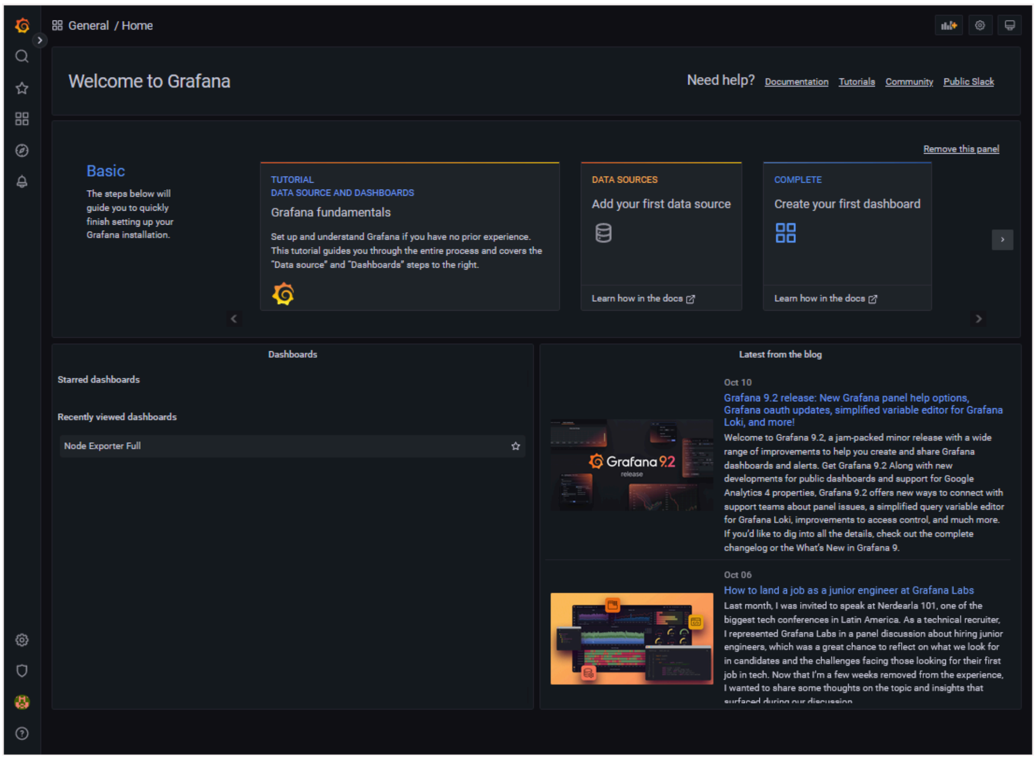
   ```
   docker run --name=grafana -dp 3000:3000 --network monitor  grafana/grafana
   ```

2. Now press `Launch Application` button below to open the Grafana browser by connection.

Launch Grafana

3. Log in with username as admin and password as admin. You may choose to set a new password. This will take you to the Grafana homepage.

   If you set a new password, it will only last for that docker instance. When you restart you need to reset it.



General / Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
DATA SOURCES
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Node Exporter Full
Latest from the blog

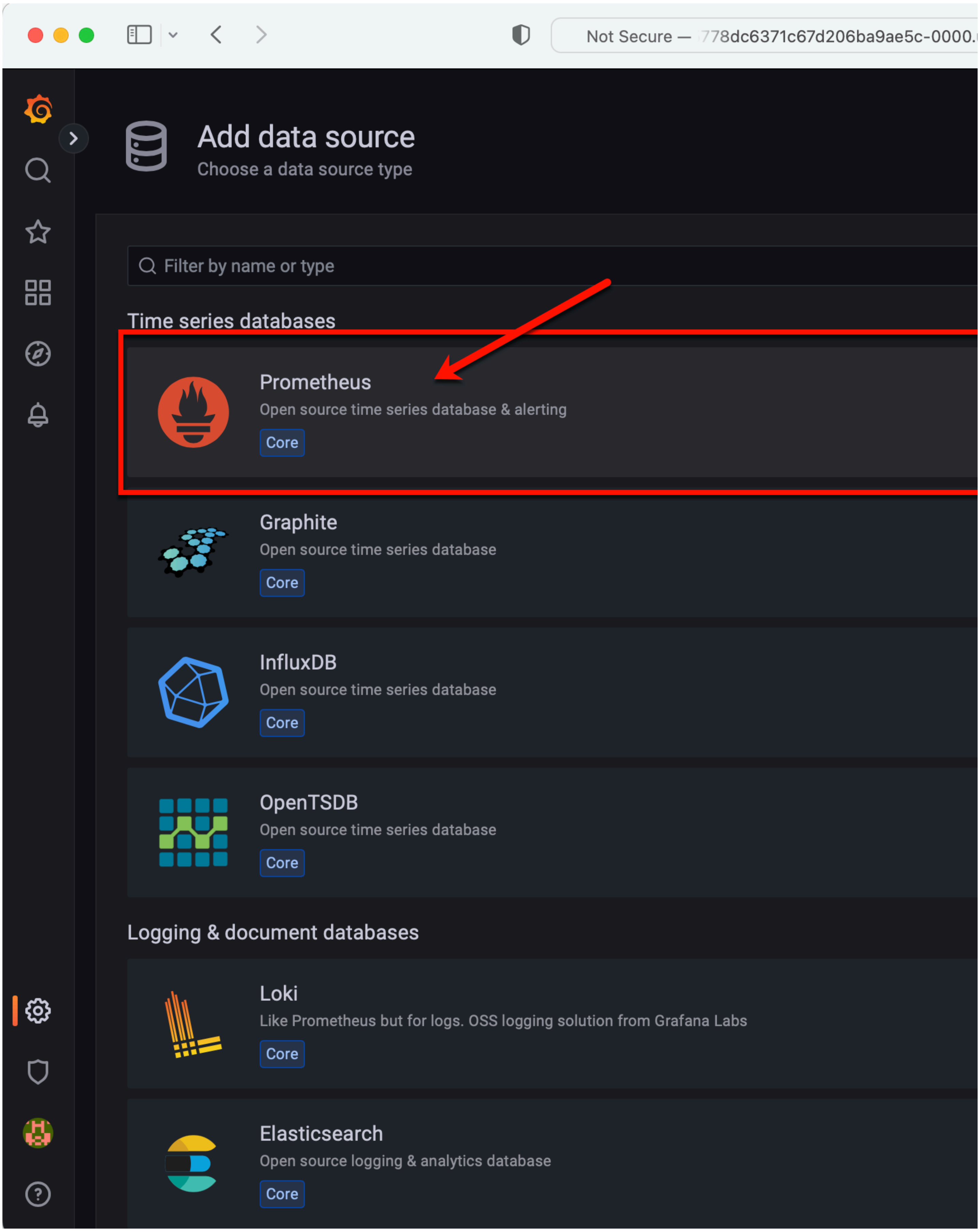
Oct 10
Grafana 9.2 release: New Grafana panel help options, Grafana oauth updates, simplified variable editor for Grafana Loki, and more!
Welcome to Grafana 9.2, a jam-packed minor release with a wide range of improvements to help you create and share Grafana dashboards and alerts. Get Grafana 9.2 Along with new developments for public dashboards and support for Google Analytics 4 properties, Grafana 9.2 offers new ways to connect with support teams about panel issues, a simplified query variable editor for Grafana Loki, improvements to access control, and much more. If you'd like to dig into all the details, check out the complete changelog or the What's New in Grafana 9.

Oct 06
How to land a job as a junior engineer at Grafana Labs
Last month, I was invited to speak at Nerdearla 101, one of the biggest tech conferences in Latin America. As a technical recruiter, I represented Grafana Labs in a panel discussion about hiring junior engineers, which was a great chance to reflect on what we look for in candidates and the challenges facing those looking for their first job in tech. Now that I'm a few weeks removed from the experience, I wanted to share some thoughts on the topic and insights that surfaced during our discussion

4. Click on `Data Sources` to add your first data source.

5. Choose `Prometheus` from the list of options available.

   Note: In this lab, we are focusing on using Prometheus to monitor the node exporters and visualize the metrics and graphs. Although Step 7 mentions starting Grafana, the lab is designed to demonstrate the capabilities of Prometheus in monitoring the node exporters. Therefore, we are not proceeding with configuring Grafana for this particular lab.


Not Secure — 778dc6371c67d206ba9ae5c-0000.

Add data source

Choose a data source type

Filter by name or type

Time series databases

Prometheus
Open source time series database & alerting
Core

Graphite
Open source time series database
Core

InfluxDB
Open source time series database
Core

OpenTSDB
Open source time series database
Core

Logging & document databases

Loki
Like Prometheus but for logs. OSS logging solution from Grafana Labs
Core

Elasticsearch
Open source logging & analytics database
Core

## Conclusion

Congratulations! You have completed this lab on Prometheus. You are now well on your way to monitoring your own applications to ensure they are running properly.

### Next Steps

Your next challenge is to set up Prometheus in your development environment to monitor your applications. You some of the queries you have learned in this lab to check on the health and performance of your applications.

### Author(s)

Lavanya T S

**Other Contributor(s)**

Pallavi Rai
John J. Rofrano