## Why Use Prometheus?

Due to the emergence of DevOps culture and Container based infrastructure created a need for a new type of monitoring framework.

Why Prometheus Is the Right Tool for monitoring containerized environment?

Multi-dimensional data model: The model is based on key-value pairs, similar to how Kubernetes itself organizes infrastructure metadata using labels.

Accessible format and protocols: Exposing Prometheus metrics is a pretty straightforward task. Metrics are human readable, are in a self-explanatory format, and are published using a standard HTTP transport.

Service discovery: The Prometheus server is in charge of periodically scraping the targets, so that applications and services don’t need to worry about emitting data (metrics are pulled, not pushed). These Prometheus servers have several methods to auto-discover scrape targets, some of them can be configured to filter and match container metadata, making it an excellent fit for ephemeral Kubernetes workloads.

Modular and highly available components: Metric collection, alerting, graphical visualization, etc, are performed by different composable services.

How to install Prometheus?

You can directly download and run the Prometheus binary in your host:

prometheus-2.3.1.linux-amd64$ ./prometheus

A better option is to deploy the Prometheus server inside a container:

docker run -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus

Deploy Prometheus to kubernetes cluster using the following menifist file

## Preparing your Environment

In this tutorial, we use an instance running on [Ubuntu Xenial (16.04)](https://www.ubuntu.com/).

1 . To run Prometheus safely on our server, we have to create a user for Prometheus and Node Exporter without the possibility to log in. To achieve this, we use the parameter --no-create-home which skips the creation of a home directory and disable the shell with --shell /usr/sbin/nologin.

sudo useradd --no-create-home --shell /usr/sbin/nologin prometheus

sudo useradd --no-create-home --shell /bin/false node\_exporter

2 . Create the folders required to store the binaries of Prometheus and its configuration files:

sudo mkdir /etc/prometheus

sudo mkdir /var/lib/prometheus

3 . Set the ownership of these directories to our prometheus user, to make sure that Prometheus can access to these folders:

sudo chown prometheus:prometheus /etc/prometheus

sudo chown prometheus:prometheus /var/lib/prometheus

## Downloading and Installing Node Exporter

As your Prometheus is only capable of collecting metrics, we want to extend its capabilities by adding Node Exporter, a tool that collects information about the system including [CPU, disk, and memory usage](https://github.com/prometheus/node_exporter" \l "enabled-by-default) and exposes them for scraping.

1 . Download the latest version of Node Exporter:

wget https://github.com/prometheus/node\_exporter/releases/download/v0.16.0/node\_exporter-0.16.0.linux-amd64.tar.gz

2 . Unpack the downloaded archive. This will create a directory node\_exporter-0.16.0.linux-amd64, containing the executable, a readme and license file:

tar xvf node\_exporter-0.16.0.linux-amd64.tar.gz

3 . Copy the binary file into the directory /usr/local/bin and set the ownership to the user you have created in step previously:

sudo cp node\_exporter-0.16.0.linux-amd64/node\_exporter /usr/local/bin

sudo chown node\_exporter:node\_exporter /usr/local/bin/node\_exporter

4 . Remove the leftover files of Node Exporter, as they are not needed any longer:

rm -rf node\_exporter-0.16.0.linux-amd64.tar.gz node\_exporter-0.16.0.linux-amd64

5 . To run Node Exporter automatically on each boot, a Systemd service file is required. Create the following file by opening it in Nano:

sudo nano /etc/systemd/system/node\_exporter.service

6 . Copy the following information in the service file, save it and exit Nano:

[Unit]

Description=Node Exporter

Wants=network-online.target

After=network-online.target

[Service]

User=node\_exporter

Group=node\_exporter

Type=simple

ExecStart=/usr/local/bin/node\_exporter

[Install]

WantedBy=multi-user.target

7 . Collectors are used to gather information about the system. By default a set of collectors is activated. You can see the details about the set in the [README-file](https://github.com/prometheus/node_exporter/blob/master/README.md" \l "enabled-by-default). If you want to use a specific set of collectors, you can define them in the ExecStart section of the service. Collectors are enabled by providing a--collector.<name> flag. Collectors that are enabled by default can be disabled by providing a --no-collector.<name> flag.

8 . Reload Systemd to use the newly defined service:

sudo systemctl daemon-reload

9 . Run Node Exporter by typing the following command:

sudo systemctl start node\_exporter

10 . Verify that the software has been started successfully:

sudo systemctl status node\_exporter

You will see an output like this, showing you the status active (running) as well as the main PID of the application:

● node\_exporter.service - Node Exporter

Loaded: loaded (/etc/systemd/system/node\_exporter.service; disabled; vendor preset: enabled)

Active: active (running) since Mon 2018-06-25 11:47:06 UTC; 4s ago

Main PID: 1719 (node\_exporter)

CGroup: /system.slice/node\_exporter.service

└─1719 /usr/local/bin/node\_exporter

11 . If everything is working, enable Node Exporter to be started on each boot of the server:

sudo systemctl enable node\_exporter

## Downloading and Installing Prometheus

1 . Download and Unpack [Prometheus](https://prometheus.io/download/) latest release of Prometheus. As exemplified, the version is 2.2.1:

sudo apt-get update && apt-get upgrade

wget https://github.com/prometheus/prometheus/releases/download/v2.2.1/prometheus-2.2.1.linux-amd64.tar.gz

tar xfz prometheus-\*.tar.gz

cd prometheus-\*

The following two binaries are in the directory:

* Prometheus - Prometheus main binary file
* promtool

The following two folders (which contain the web interface, configuration files examples and the license) are in the directory:

* consoles
* console\_libraries

2 . Copy the binary files into the /usr/local/bin/directory:

sudo cp ./prometheus /usr/local/bin/

sudo cp ./promtool /usr/local/bin/

3 . Set the ownership of these files to the prometheus user previously created:

sudo chown prometheus:prometheus /usr/local/bin/prometheus

sudo chown prometheus:prometheus /usr/local/bin/promtool

4 . Copy the consoles and console\_libraries directories to /etc/prometheus:

sudo cp -r ./consoles /etc/prometheus

sudo cp -r ./console\_libraries /etc/prometheus

5 . Set the ownership of the two folders, as well as of all files that they contain, to our prometheus user:

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

sudo chown -R prometheus:prometheus /etc/prometheus/console\_libraries

6 . In our home folder, remove the source files that are not needed anymore:

cd .. && rm -rf prometheus-\*

## Configuring Prometheus

Prior to using Prometheus, it needs basic configuring. Thus, we need to create a configuration file named prometheus.yml

Note: The configuration file of Prometheus is written in [YAML](http://www.yaml.org/start.html) which strictly forbids to use tabs. If your file is incorrectly formatted, Prometheus will not start. Be careful when you edit it.

1 . Open the file prometheus.yml in a text editor:

sudo nano /etc/prometheus/prometheus.yml

Prometheus’ configuration file is divided into three parts: global, rule\_files, and scrape\_configs.

In the global part we can find the general configuration of Prometheus: scrape\_interval defines how often Prometheus scrapes targets, evaluation\_interval controls how often the software will evaluate rules. Rules are used to create new time series and for the generation of alerts.

The rule\_files block contains information of the location of any rules we want the Prometheus server to load.

The last block of the configuration file is named scape\_configs and contains the information which resources Prometheus monitors.

Our file should look like this example:

---

global:

evaluation\_interval: 15s

external\_labels:

monitor: codelab-monitor

scrape\_interval: 15s

rule\_files: ~

scrape\_configs:

-

job\_name: prometheus

scrape\_interval: 5s

static\_configs:

-

targets:

- "localhost:9090"

-

job\_name: node\_exporter

scrape\_interval: 5s

static\_configs:

-

targets:

- "localhost:9100"

The global scrape\_interval is set to 15 seconds which is enough for most use cases.

We do not have any rule\_files yet, so the lines are commented out and start with a #.

In the scrape\_configs part we have defined our first exporter. It is Prometheus that monitors itself. As we want to have more precise information about the state of our Prometheus server we reduced the scrape\_interval to 5 seconds for this job. The parameters static\_configsand targets determine where the exporters are running. In our case it is the same server, so we use localhost and the port 9090.

As Prometheus scrapes only exporters that are defined in the scrape\_configs part of the configuration file, we have to add Node Exporter to the file, as we did for Prometheus itself.

We add the following part below the configuration for scrapping Prometheus:

- job\_name: 'node\_exporter'

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:9100']

Overwrite the global scrape interval again and set it to 5 seconds. As we are scarping the data from the same server as Prometheus is running on, we can use localhost with the default port of Node Exporter: 9100.

If you want to scrape data from a remote host, you have to replace localhost with the IP address of the remote server.

For all information about the configuration of Prometheus, you may check the [configuration documentation](https://prometheus.io/docs/prometheus/latest/configuration/configuration/).

2 . Set the ownership of the file to our Prometheus user:

sudo chown prometheus:prometheus /etc/prometheus/prometheus.yml

Our Prometheus server is ready to run for the first time.

## Running Prometheus

1 . Start Prometheus directly from the command line with the following command, which executes the binary file as our Prometheus user:

sudo -u prometheus /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

The server starts displaying multiple status messages and the information that the server has started:

level=info ts=2018-04-12T11:56:53.084000977Z caller=main.go:220 msg="Starting Prometheus" version="(version=2.2.1, branch=HEAD, revision=bc6058c81272a8d938c05e75607371284236aadc)"

level=info ts=2018-04-12T11:56:53.084463975Z caller=main.go:221 build\_context="(go=go1.10, user=root@149e5b3f0829, date=20180314-14:15:45)"

level=info ts=2018-04-12T11:56:53.084632256Z caller=main.go:222 host\_details="(Linux 4.4.127-mainline-rev1 #1 SMP Sun Apr 8 10:38:32 UTC 2018 x86\_64 scw-041406 (none))"

level=info ts=2018-04-12T11:56:53.084797692Z caller=main.go:223 fd\_limits="(soft=1024, hard=65536)"

level=info ts=2018-04-12T11:56:53.09190775Z caller=web.go:382 component=web msg="Start listening for connections" address=0.0.0.0:9090

level=info ts=2018-04-12T11:56:53.091908126Z caller=main.go:504 msg="Starting TSDB ..."

level=info ts=2018-04-12T11:56:53.102833743Z caller=main.go:514 msg="TSDB started"

level=info ts=2018-04-12T11:56:53.103343144Z caller=main.go:588 msg="Loading configuration file" filename=/etc/prometheus/prometheus.yml

level=info ts=2018-04-12T11:56:53.104047346Z caller=main.go:491 msg="Server is ready to receive web requests."

2 . Open your browser and type http://IP.OF.YOUR.SERVER:9090 to access the Prometheus interface. If everything is working, we end the task by pressing on CTRL + C on our keyboard.

Note: If you get an error message when you start the server, double check your configuration file for possible YAML syntax errors. The error message will tell you what to check.

3 . The server is working now, but it cannot yet be launched automatically at boot. To achieve this, we have to create a new systemd configuration file that will tell your OS which services should it launch automatically during the boot process.

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

The service file tells systemd to run Prometheus as prometheus and specifies the path of the configuration files.

4 . Copy the following information in the file and save it, then exit the editor:

[Unit]

Description=Prometheus Monitoring

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

ExecReload=/bin/kill -HUP $MAINPID

[Install]

WantedBy=multi-user.target

5 . To use the new service, reload systemd:

sudo systemctl daemon-reload

We enable the service so that it will be loaded automatically during boot:

sudo systemctl enable prometheus

6 . Start Prometheus:

sudo systemctl start prometheus

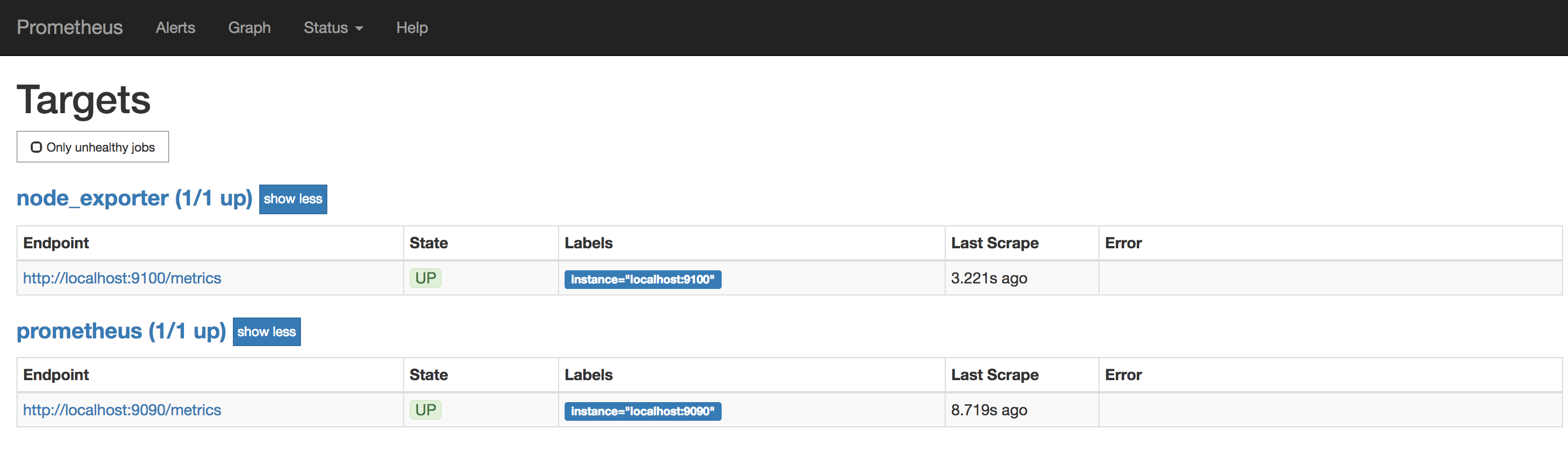
Your Prometheus server is ready to be used.

We have now installed Prometheus to monitor your instance.

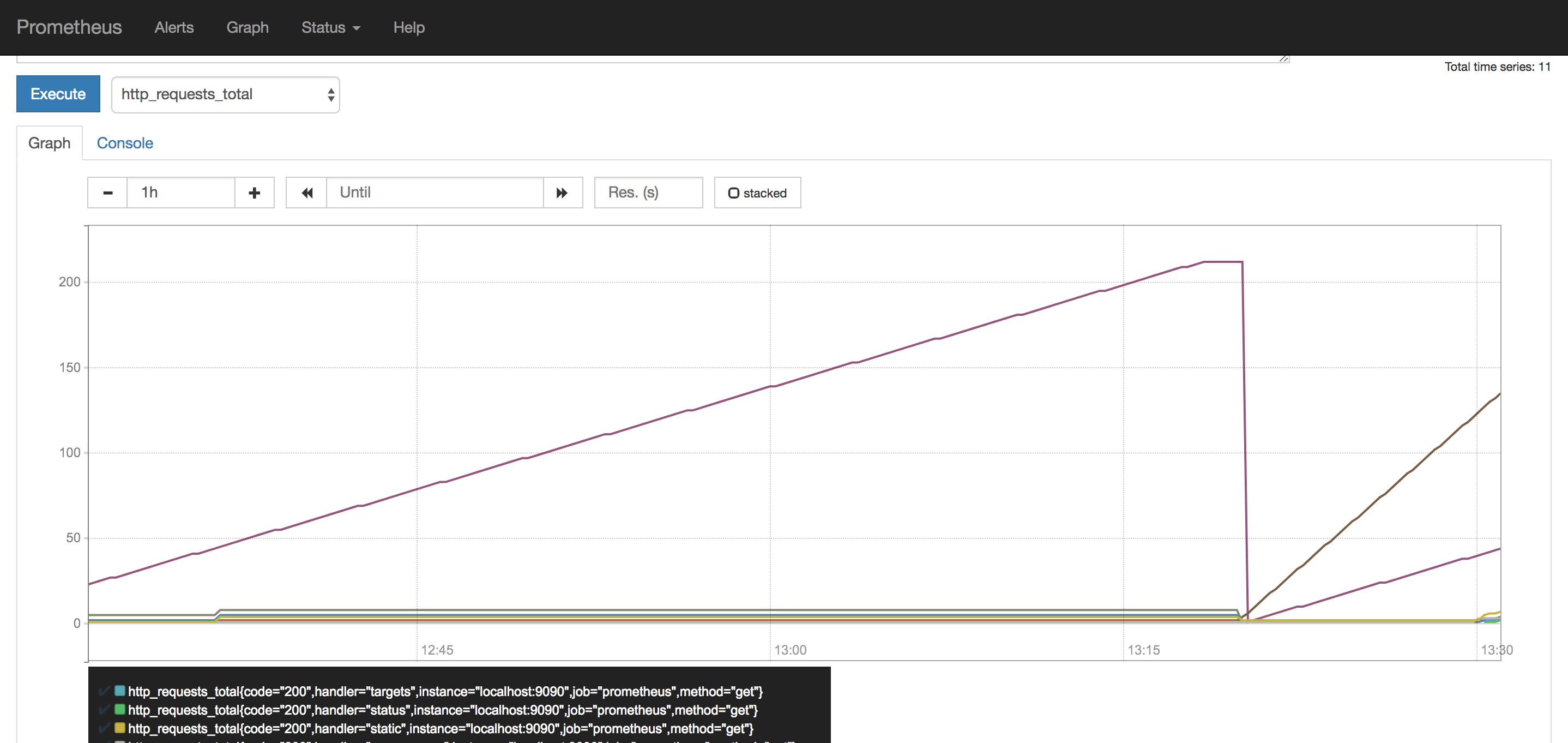
## Prometheus Web Interface

Prometheus provides a basic web server running on http://your.server.ip:9000 that provide access to the data collected by the software.

We can verify the status of our Prometheus server from the interface:



Moreover, do some queries in the data that has been collected.



The interface is very lightweight, and the Prometheus team recommend to use a tool like [Grafana](https://grafana.com/) if you want to do more than testing and debugging the installation.

## Installing of Grafana

1 . Install Grafana on our instance which queries our Prometheus server.

wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana\_5.0.4\_amd64.deb

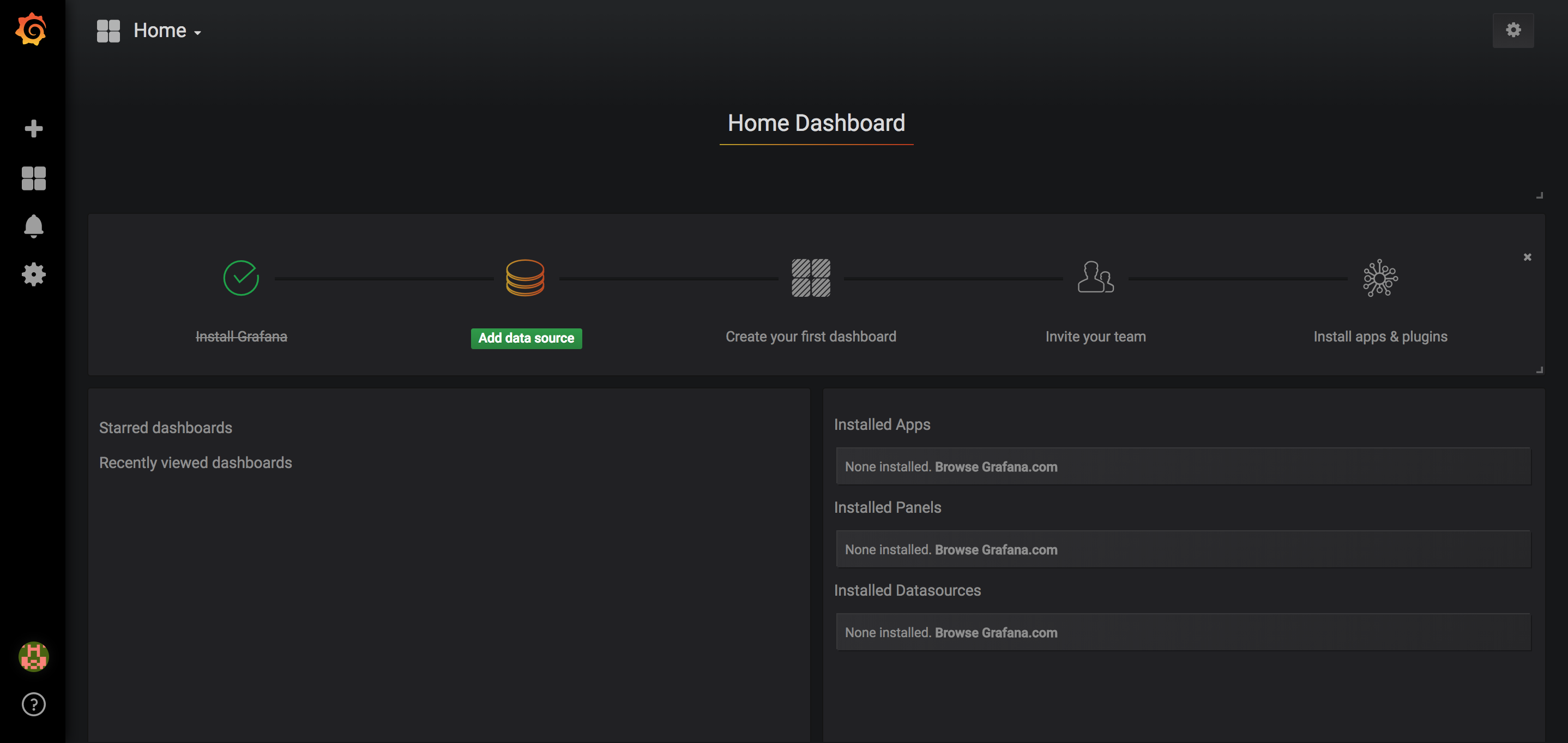
sudo apt-get install -y adduser libfontconfig

sudo dpkg -i grafana\_5.0.4\_amd64.deb

2 . Enable the automatic start of Grafana by systemd:

sudo systemctl daemon-reload && sudo systemctl enable grafana-server && sudo systemctl start grafana-server

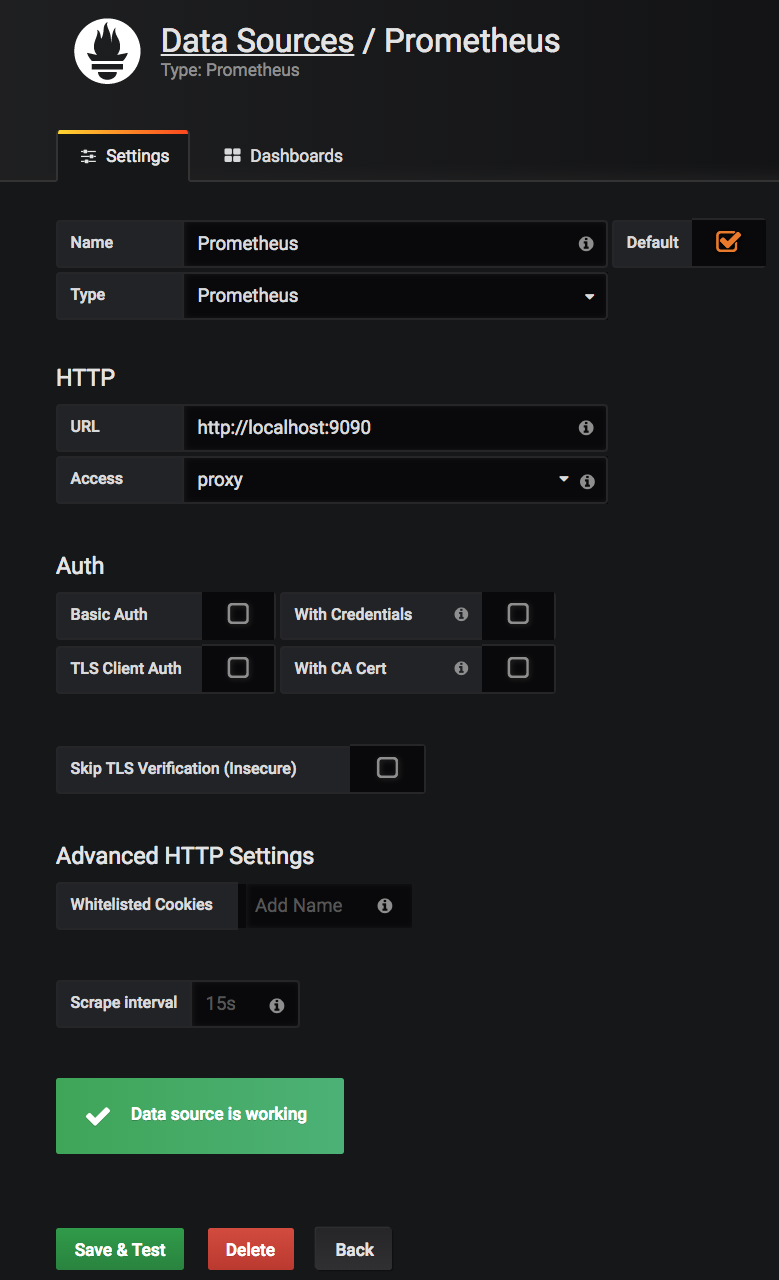
Grafana is running now, and we can connect to it at http://your.server.ip:3000. The default user and password is admin / admin.



Now you have to create a Prometheus data source:

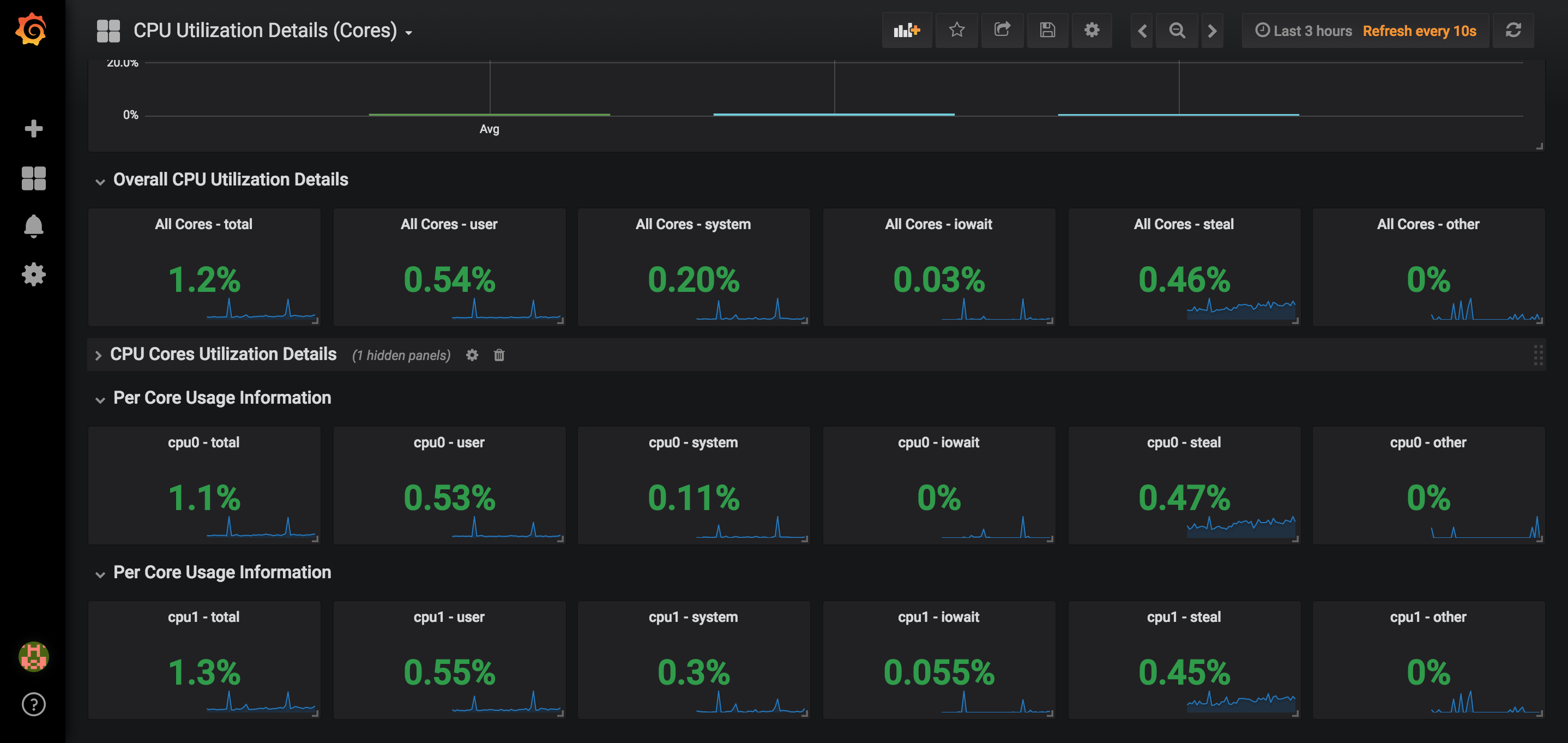
* Click on the Grafana logo to open the sidebar.
* Click on “Data Sources” in the sidebar.
* Choose “Add New”.
* Select “Prometheus” as the data source.
* Set the Prometheus server URL (in our case: http://localhost:9090/)
* Click “Add” to test the connection and to save the new data source.

Your settings should look like this:



You are now ready to create your first dashboard from the information collected by Prometheus. You can also import some dashboards from a collection of [shared dashboards](https://grafana.com/dashboards?dataSource=prometheus)

Here is an example of a Dashboard that uses the CPU usage of our node and presents it in Grafana:



In this tutorial, we were able to configure a Prometheus server with two data collectors that are scraped by our Prometheus server which provides the data to build Dashboards with Grafana. Don’t hesitate to consult the official documentation of [Prometheus](https://prometheus.io/docs/introduction/overview/) and [Grafana](http://docs.grafana.org/).

Installing Prometheus and Grafana Using HELM

Requirement:

Helm tiller with client.

Install Nginx-Ingress using helm:

$helm install stable/nginx-ingress --namespace demo

### Install Prometheus using helm:

$ git clone <https://github.com/helm/charts.git>

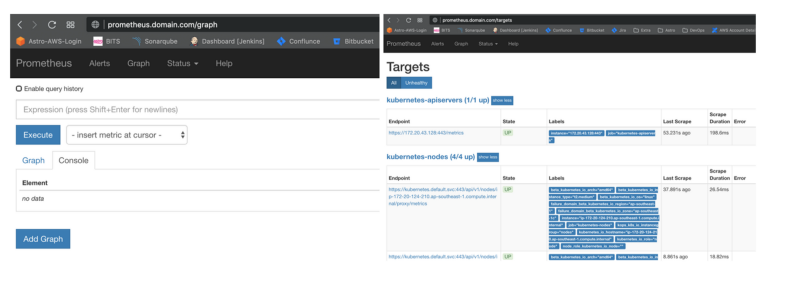
$ cd charts/stable/prometheus

Include the secret’s name, along with the desired hostnames, in the alertmanager/server Ingress TLS section of your custom values.yaml file:



$ helm install --name=prometheus . --namespace demo --set rbac.create=true

Open http://prometheus.domain.com on a browser.



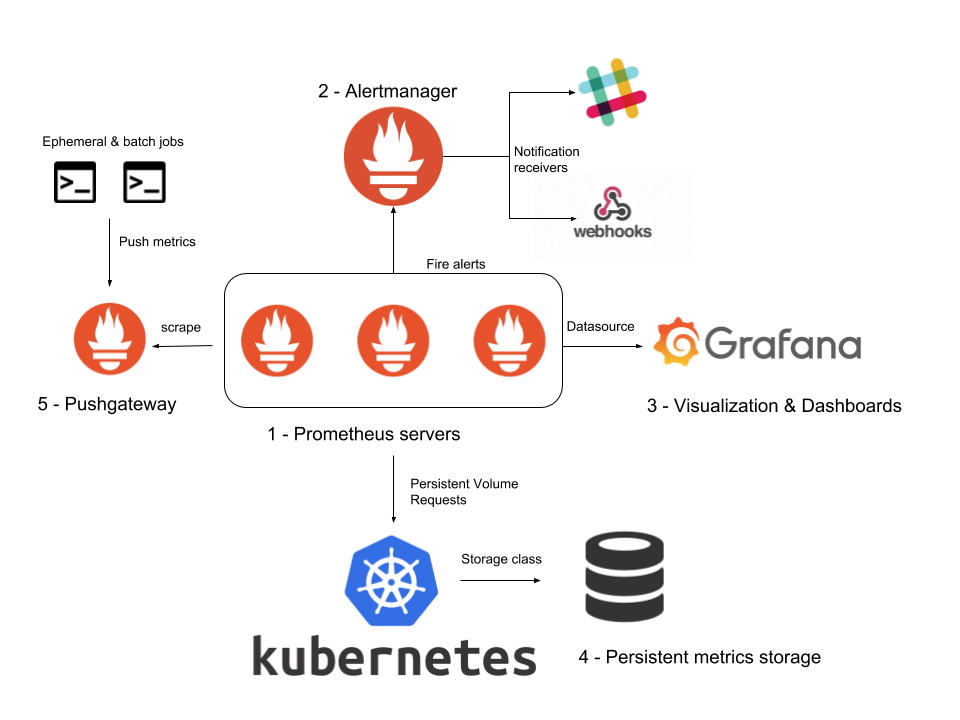
### Install Grafana using helm:

$ helm install --name grafana stable/grafana --set=ingress.enabled=True,ingress.hosts={grafana.domain.com} --namespace demo --set rbac.create=true

Get Admin Password:

kubectl get secret --namespace demo grafana-monit -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

Open [http://grafana.domain.com](http://grafana.domain.com/) a browser



### Before You Begin

1. Prometheus Node Exporter needs Prometheus server to be up and running. If you would like to setup Prometheus, please see the [Prometheus setup guide for Linux](https://devopscube.com/install-configure-prometheus-linux/).
2. Port 9100 opened in server firewall as Prometheus reads metrics on this port.

### Setup Node Exporter Binary

Step 1: Download the latest node exporter package. You should check the [Prometheus downloads section](https://prometheus.io/download/) for the latest version and update this command to get that package.

cd /tmp curl -LO https://github.com/prometheus/node\_exporter/releases/download/v0.16.0/node\_exporter-0.16.0.linux-amd64.tar.gz

|  |  |
| --- | --- |
| 1  2 | cd /tmp  curl -LO https://github.com/prometheus/node\_exporter/releases/download/v0.16.0/node\_exporter-0.16.0.linux-amd64.tar.gz |

Step 2: Unpack the tarball

tar -xvf node\_exporter-0.16.0.linux-amd64.tar.gz

|  |  |
| --- | --- |
| 1 | tar -xvf node\_exporter-0.16.0.linux-amd64.tar.gz |

Step 3: Move the node export binary to /usr/local/bin

sudo mv node\_exporter-0.16.0.linux-amd64/node\_exporter /usr/local/bin/

|  |  |
| --- | --- |
| 1 | sudo mv node\_exporter-0.16.0.linux-amd64/node\_exporter /usr/local/bin/ |

### Create a Custom Node Exporter Service

Step 1: Create a node\_exporter user to run the node exporter service.

sudo useradd -rs /bin/false node\_exporter

|  |  |
| --- | --- |
| 1 | sudo useradd -rs /bin/false node\_exporter |

Step 2: Create a node\_exporter service file under systemd.

sudo vi /etc/systemd/system/node\_exporter.service

|  |  |
| --- | --- |
| 1 | sudo vi /etc/systemd/system/node\_exporter.service |

Step 3: Add the following service file content to the service file and save it.

[Unit] Description=Node Exporter After=network.target [Service] User=node\_exporter Group=node\_exporter Type=simple ExecStart=/usr/local/bin/node\_exporter [Install] WantedBy=multi-user.target

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | [Unit]  Description=Node Exporter  After=network.target    [Service]  User=node\_exporter  Group=node\_exporter  Type=simple  ExecStart=/usr/local/bin/node\_exporter    [Install]  WantedBy=multi-user.target |

Step 4: Reload the system daemon and star the node exporter service.

[READ  How to Install and Configure Jenkins 2.0 - Getting Started](https://devopscube.com/install-configure-jenkins-2-0/)

sudo systemctl daemon-reload sudo systemctl start node\_exporter

|  |  |
| --- | --- |
| 1  2 | sudo systemctl daemon-reload  sudo systemctl start node\_exporter |

Step 5: check the node exporter status to make sure it is running in the active state.

sudo systemctl status node\_exporter

|  |  |
| --- | --- |
| 1 | sudo systemctl status node\_exporter |

Step 6: Enable the node exporter service to the system startup.

sudo systemctl enable node\_exporter

|  |  |
| --- | --- |
| 1 | sudo systemctl enable node\_exporter |

Now, node exporter would be exporting metrics on port 9100.

You can see all the server metrics by visiting your server URL on /metrics as shown below.

http://<server-IP>:9100/metrics

|  |  |
| --- | --- |
| 1 | http://<server-IP>:9100/metrics |

### Configure the Server as Target on Prometheus Server

Now that we have the node exporter up and running on the server, we have to add this server a target on the Prometheus server configuration.

Note: This configuration should be done on the Prometheus server.

Step 1: Login to the Prometheus server and open the prometheus.yml file.

sudo vi /etc/prometheus/prometheus.yml

|  |  |
| --- | --- |
| 1 | sudo vi /etc/prometheus/prometheus.yml |

Step 2: Under the scrape config section add the node exporter target as shown below. Change 10.142.0.3 with your server IP where you have setup node exporter. Job name can be your server hostname or IP for identification purposes.

- job\_name: 'node\_exporter\_metrics' scrape\_interval: 5s static\_configs: - targets: ['10.142.0.3:9100']

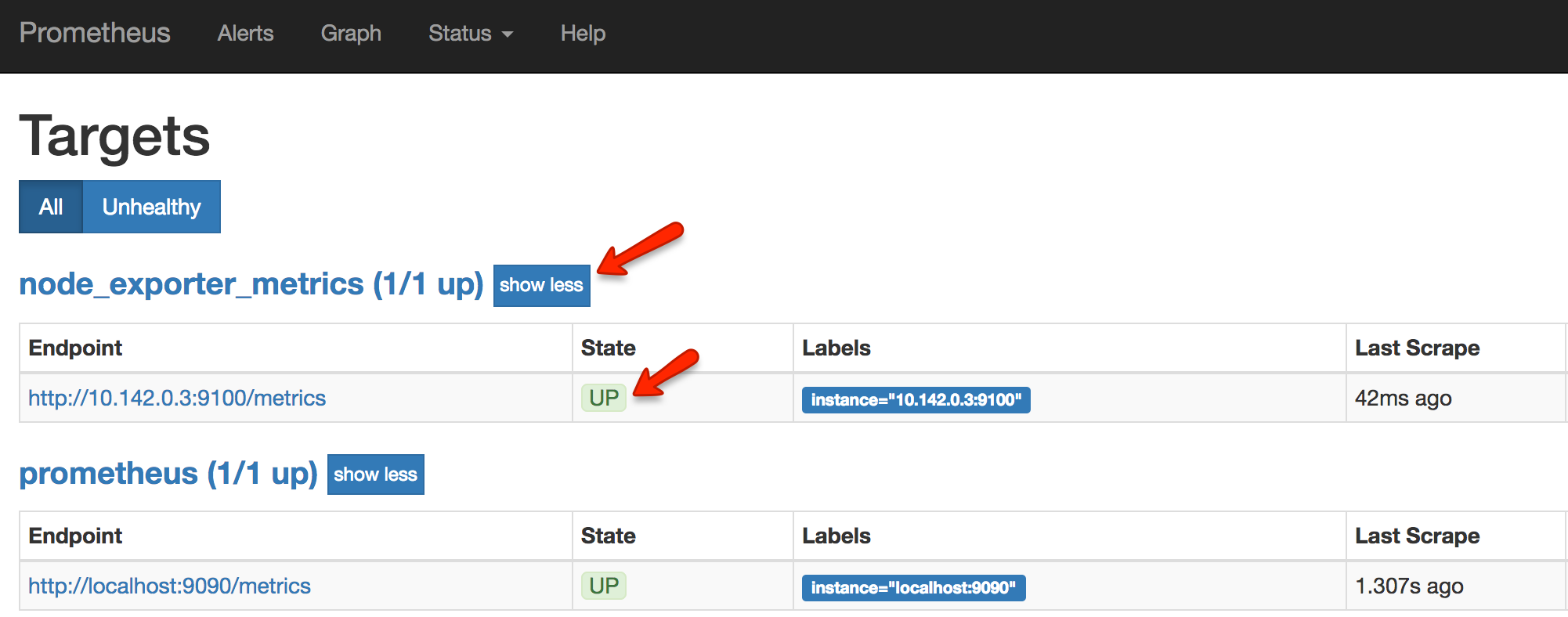
|  |  |
| --- | --- |
| 1  2  3  4 | - job\_name: 'node\_exporter\_metrics'      scrape\_interval: 5s      static\_configs:        - targets: ['10.142.0.3:9100'] |

Step 3: Restart the prometheus service for the configuration changes to take place.

sudo systemctl restart prometheus

|  |  |
| --- | --- |
| 1 | sudo systemctl restart prometheus |

Now, if you check the target in prometheus web UI (http://<prometheus-IP>:9090/targets) , you will be able to see the status as shown below.



Also, you can use the Prometheus expression browser to query for node related metrics. Following are the few key node metrics you can use to find its statistics.

[READ  Docker Multi-Host Networking Tutorial - Using Consul](https://devopscube.com/docker-multi-host-networking-tutorial/)

node\_memory\_MemFree\_bytes node\_cpu\_seconds\_total node\_filesystem\_avail\_bytes rate(node\_cpu\_seconds\_total{mode="system"}[1m]) rate(node\_network\_receive\_bytes\_total[1m])

|  |  |
| --- | --- |
| 1  2  3  4  5 | node\_memory\_MemFree\_bytes  node\_cpu\_seconds\_total  node\_filesystem\_avail\_bytes  rate(node\_cpu\_seconds\_total{mode="system"}[1m])  rate(node\_network\_receive\_bytes\_total[1m]) |

