Server Monitoring using Prometheus and Grafana

CONTENTS

1. ABSTRACT

Server Monitoring is a process to monitor server's system resources like CPU Usage, Memory Consumption, I/O, Network, Disk Usage, Process etc. Server Monitoring helps understanding server's system resource usage which can help you better your capacity planning and provide a better end-user experience.

Your application's health depends on a large part on the health of the underlying server. Server Monitoring ensures that your server machine is capable of hosting your applications. Server Monitoring provides you with data relating to your operating system and when used in conjunction with other monitoring data from the application you get a true glimpse into the working of your system.

2. INTRODUCTION

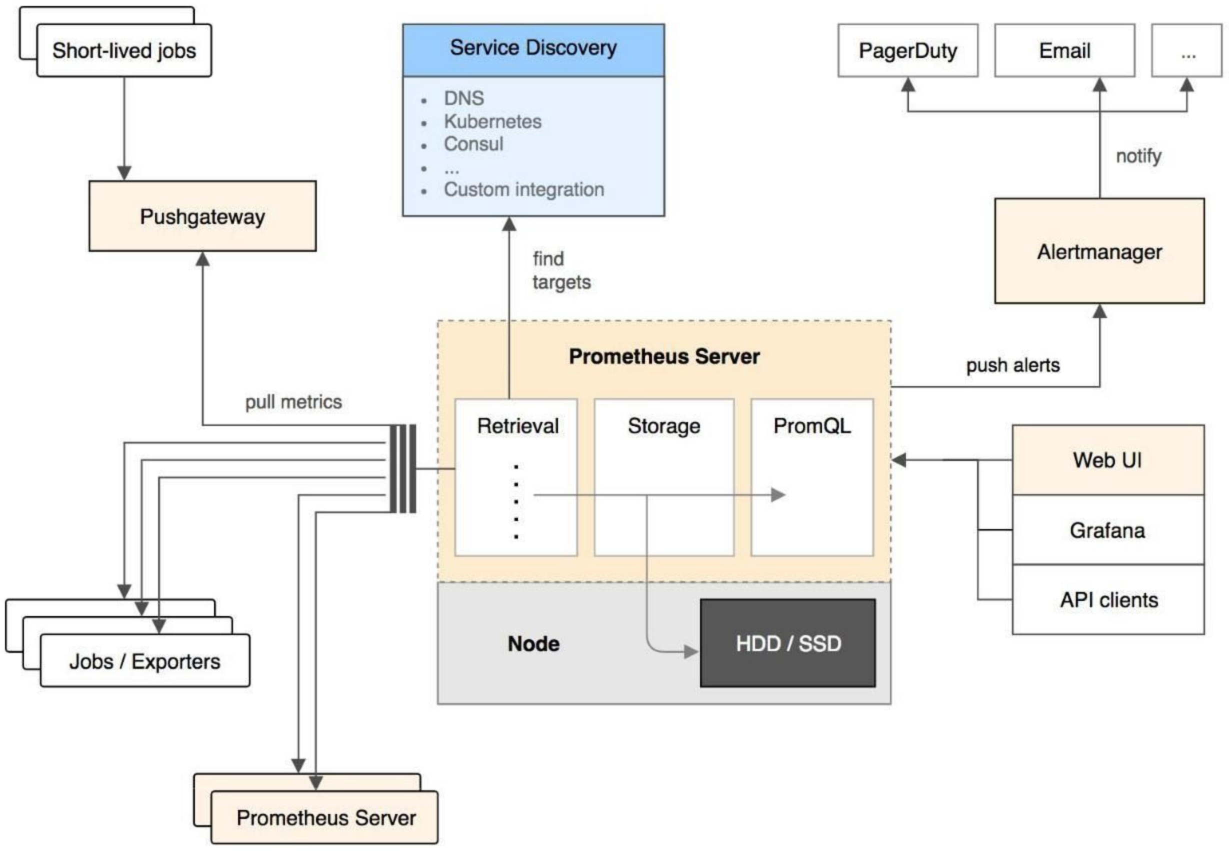
Prometheus is an open source monitoring system and time series database. It addresses many aspects of monitoring such as the generation and collection of metrics, graphing the resulting data on dashboards, and alerting on anomalies. To achieve this, it offers a variety of components that are run separately but used in combination.

Docker provides a way for you to encapsulate server processes using Linux containers (or other encapsulation technologies) so that they are more easily managed and isolated from each other. Each Docker container is a local instance of a Docker image. You can think of a Docker image as a complete Linux installation. In this tutorial, we will learn how to install three key components for using Prometheus on Docker. These are:

- A Prometheus server to collect metrics and query them
- A Node Exporter to export system metrics in a Prometheus-compatible format
- Grafana, a web-based graphical dashboard builder that supports Prometheus among other backends.

There are many more components in the Prometheus ecosystem, but these three provide a good starting point for using Prometheus.

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 record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.



Overall architecture of Prometheus and some of its ecosystem components

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 have to set up complex infrastructure to use it.

Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture.

3. IMPLEMENTATION

3.1: What do you need?

To follow this tutorial you will need:

- An Ubuntu EC2 instance on AWS;
- a non-root user with sudo access; and
- Docker installed on the server instance.

3.2: Initial Server Setup. Creating the non-root user and giving it sudo access

Login to your server using the following command:

```
$ ssh root@<IP_ADDRESS>
```

If this is the first time you are logging in, you will be prompted to change the password.

Create a new user using the following command:

```
$ adduser <username>
```

As root, run this command to add your new user to the ***sudo*** group

```
$ gpasswd -a <username> sudo
```

If you do not already have an SSH key pair, which consists of a public and private key, you need to generate one.

To generate a new key pair, enter the following command at the terminal of your local machine

```
aws-server$ ssh-keygen
```

After generating an SSH key pair, you will want to copy your public key to your new server. Run the ssh-copy-id script by specifying the user and IP address of the server that you want to install the key on, like this:

```
aws-server$ ssh-copy-id <username>@<IP_ADDRESS>
```

Then ssh into the non-root user: `$ ssh <username>@<IP_ADDRESS>`

3.3: Installing Docker

The quickest way to install Docker is to download and install their installation script.

```
$ sudo wget -qO- https://get.docker.com/ | sh
```

Working with Docker is a pain if your user is not configured correctly, so add your user to the docker group with the following command.

```
$ sudo usermod -aG docker <username>
```

3.4: Installing Prometheus

The Prometheus server is the central piece of the Prometheus ecosystem and is responsible for collecting and storing metrics as well as processing expression queries and generating alerts. Docker container images for all Prometheus components are hosted under the prom organization on Docker Hub. In this tutorial, we will choose to pass in a configuration file from the host system. First, create a minimal Prometheus configuration file on the host filesystem at ~/prometheus.yml:

```
$ touch ~/Prometheus.yml
```

Add the following contents to the file:

```
# A scrape configuration scraping a Node Exporter and the
Prometheus server itself.
scrape_configs:
 # Scrape Prometheus itself every 5 seconds.
 - job_name: "prometheus"
   scrape_interval: 5s
   static_configs:
     - targets: ["IP_ADDRESS:9090"]

# Scrape the Node Exporter every 5 seconds.
 - job_name: "node"
   scrape_interval: 5s
     static_configs:
       - targets: ["IP_ADDRESS:9100"]
```

Start the Prometheus Docker container with the external configuration file:

```
$ sudo docker run -d -p 9090:9090 \
-v/etc/prometheus/Prometheus.yml:/etc/prometheus/P
rometheus.yml: \
prom/prometheus \
--config.file="/etc/prometheus/Prometheus.yml \
--storage.tsdb.path="/Prometheus" \
```

The first time you run this command, it will pull the Docker image from the Docker Hub. This command is quite long and contains many command line options. Let's take a look at it in more detail:

- The -d option starts the Prometheus container in detached mode, meaning that the container will be started in the background and will not be terminated by pressing CTRL+C.
- The -p 9090:9090 option exposes Prometheus's web port (9090) and makes it reachable via the external IP address of the host system.
- The -v [...] option mounts the prometheus.yml configuration file from the host filesystem into the location within the container where Prometheus expects it (/etc/prometheus/Prometheus.yml).
- The -config.file option is set accordingly to the location of the Prometheus configuration file ***within the container***.
- The -storage.tsdb.path option configures the metrics storage location ***within the container.***

3.5: Setting up Node Exporter

The Node Exporter is a server that exposes Prometheus metrics about the host machine (node) it is running on. This includes metrics about the machine's filesystems, networking devices, processor usage, memory usage, and more. To start the Node Exporter on port 9100 using Docker:

```
$ sudo docker run -d -p 9100:9100 \
-v "/proc:/host/proc" \
-v "/sys:/host/sys" \
-v "/:/rootfs" \
--net="host" \
prom/node-exporter \
--path.procfs=/host/proc \
--path.sysfs=/host/proc \
--collector.filesystem.ignored-mount-points
"^/(sys|proc|dev|host|etc)($|/)"
```

The following Docker and Node Exporter flags are used to provide a reasonable approximation for the ***host*** metrics:

- On Linux, the Node Exporter gathers most of its metrics from the /proc and /sys filesystems. These filesystems are mounted from the host into the container underneath a /host directory, using Docker's -v flag.
- Via the Node Exporter's -path.procfs and -path.sysfs flags, we instruct the Node Exporter to look for the /proc and /sys filesystems in a non-standard location.
- To report host filesystem metrics, we also mount the entire root (/) filesystem into the container (at /rootfs), again using Docker's -v flag.
- Use Node Exporter's -collector.filesystem.ignored-mount-points flag to ignore any other file systems within the container that do not belong to the host system. This option takes a regular expression of mount points to exclude from the reported metrics.
- Using the --net=host Docker flag, we place the container into the same network stack as the host, so that reading from files such as /proc/net/dev will yield the same results as on the host (reading from the /proc filesystem mounted in from the host is not sufficient).

3.6: Setting up Grafana

Finally, we will set up Grafana. Grafana is a graphical dashboard builder that supports Prometheus as a backend to query for data to graph.
Grafana stores its dashboard metadata (such as which dashboards exist and what graphs they should show) in a configurable SQL-based database. Grafana supports using a local file-backed SQLite3 database as well as external database servers such as MySQL or PostgreSQL for this.In this tutorial, we will use a SQLite3 database backed by a Docker data volume.Launch Grafana as a Docker container with an administrator password of your choosing:

```
$ sudo docker run -d -p 3000:3000 \
-e "GF_SECURITY_ADMIN_PASSWORD=admin" \
-v "~/grafana_db:/var/lib/grafana" \
grafana/grafana
```

This will download the Grafana Docker image from the Docker Hub and create a new Docker volume placed at ~/grafana_db on the host system and at `/var/lib/grafana` in the container filesystem.

- In the container, Grafana will then automatically create and initialize its SQLite3 database at /var/lib/grafana/grafana.db.
- The -e flag allows passing environment variables to the process launched inside the Docker container.
- Here, we use it to set the GF_SECURITY_ADMIN_PASSWORD environment variable to the desired dashboard administrator password, overriding the default password of admin.

3.7: Checking System Status

One can check all running docker containers using the following command:

```
$ sudo docker ps
```

You should now be able to reach your Prometheus server at `http://<IP_ADDRESS>:9090/`. Verify that it is collecting metrics about itself by heading to `http://<IP_ADDRESS>:9090/status`

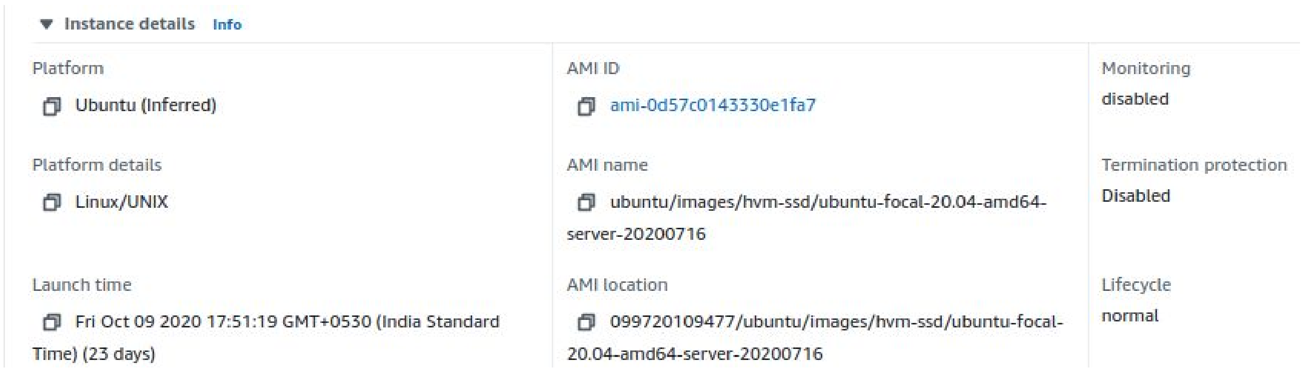
To verify that Grafana is running correctly, head to `http://<IP_ADDRESS>:3000/`

The administrator username is admin and the password is the one you chose when starting the Docker container previously.

4. RESULTS

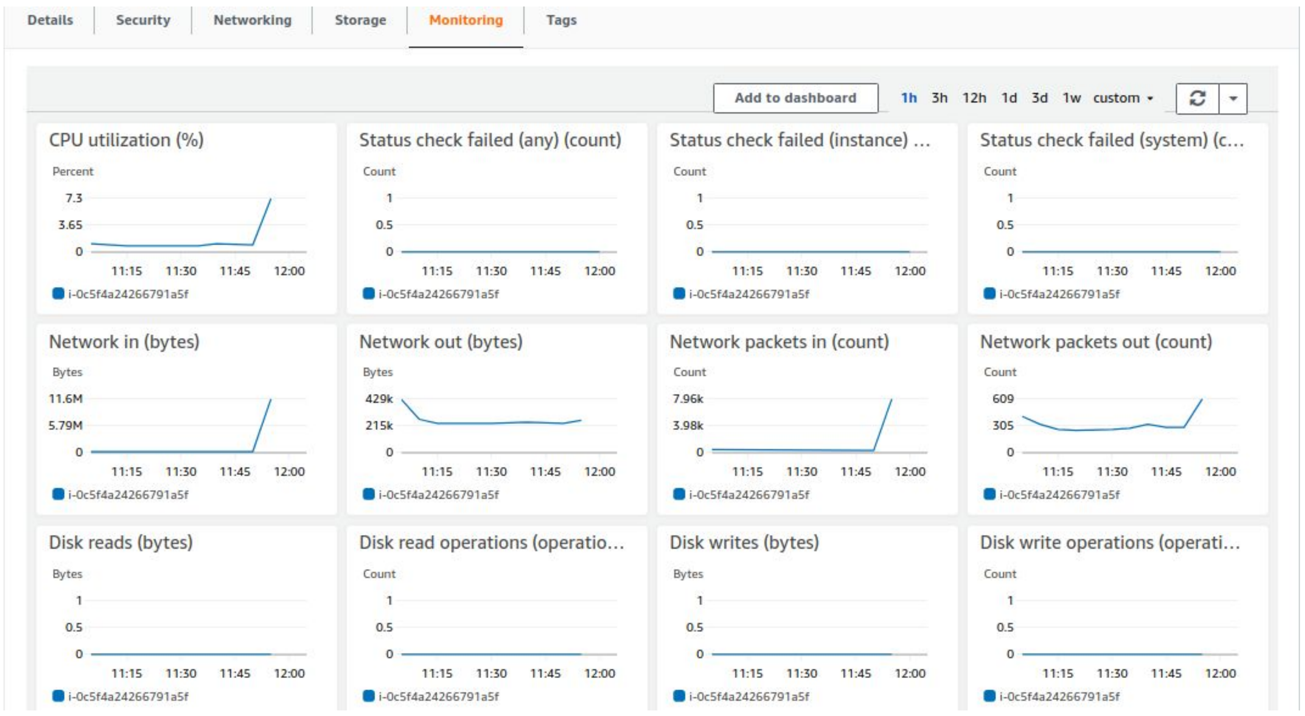
We have set up a Prometheus server, a Node Exporter, and Grafana — all using Docker. Even though these are currently all running on the same machine, this is only for demonstration purposes. In production setups, one would usually run the Node Exporter on every monitored machine, multiple Prometheus servers (as needed by the organization), as well as a single Grafana server to graph the data from these servers.

5. SCREENSHOTS

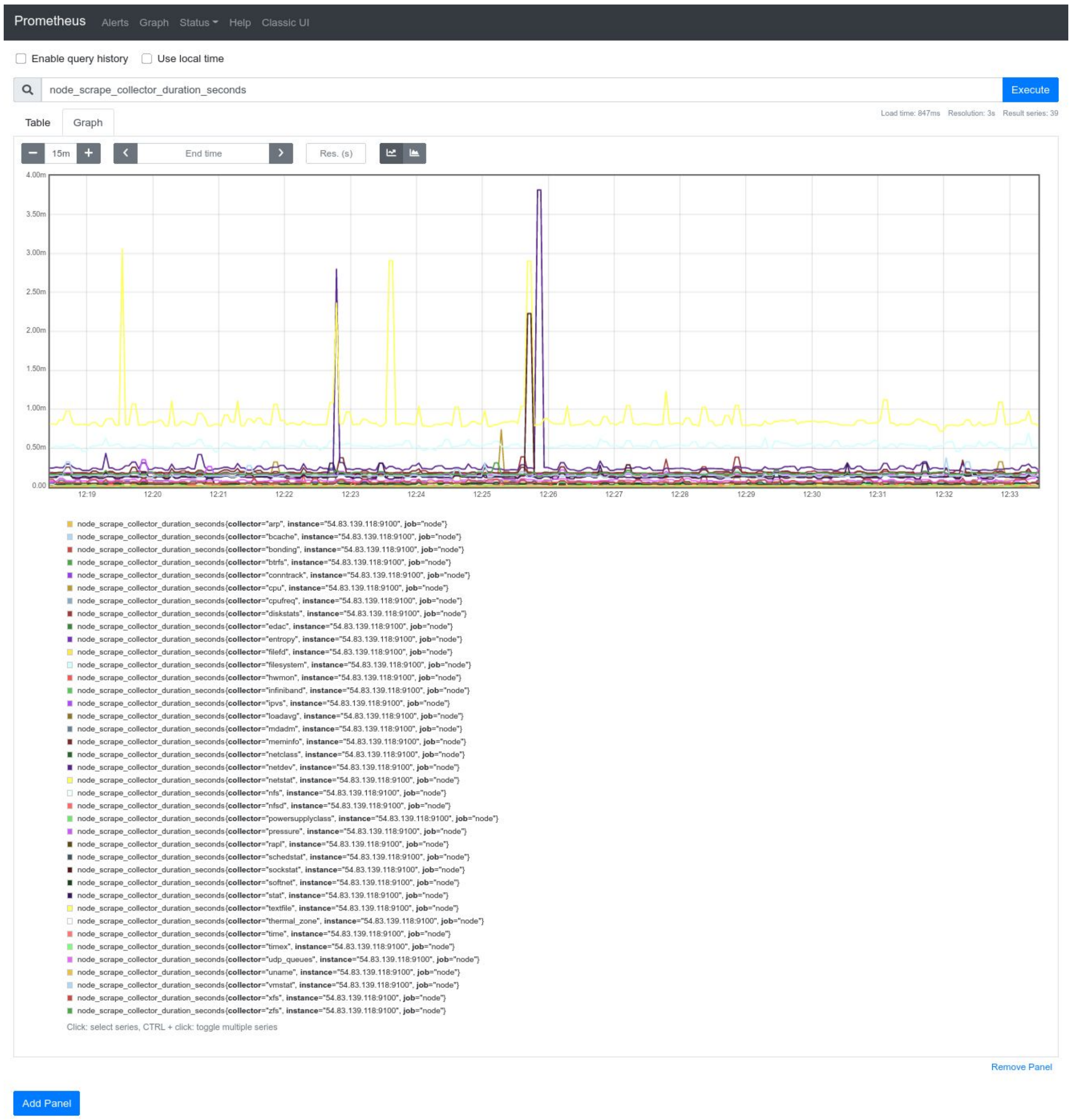


▼ Instance details Info

Platform	AMI ID	Monitoring
Ubuntu (Inferred)	ami-0d57c0143330e1fa7	disabled
Platform details	AMI name	Termination protection
Linux/UNIX	ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-20200716	Disabled
Launch time	AMI location	Lifecycle
Fri Oct 09 2020 17:51:19 GMT+0530 (India Standard Time) (23 days)	099720109477/ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-20200716	normal


Details
Security
Networking
Storage
Monitoring
Tags
Add to dashboard
1h 3h 12h 1d 3d 1w custom
CPU utilization (%)
Status check failed (any) (count)
Status check failed (instance) ...
Status check failed (system) (c...
Network in (bytes)
Network out (bytes)
Network packets in (count)
Network packets out (count)
Disk reads (bytes)
Disk read operations (operatio...
Disk writes (bytes)
Disk write operations (operati...
Percent
Count
Bytes
11:15 11:30 11:45 12:00
i-0c5f4a24266791a5f

Active view of the dashboard for the AWS EC2 instance



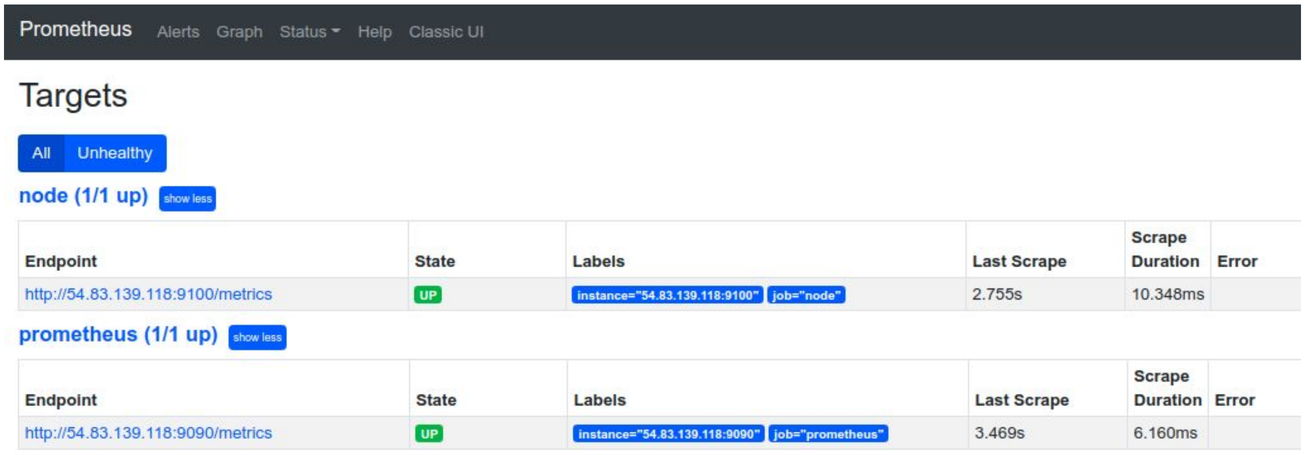
```
node_scrape_collector_duration_seconds
```

Load time: 847ms Resolution: 3s Result series: 39

```
🟧 node_scrape_collector_duration_seconds{collector="arp", instance="54.83.139.118:9100", job="node"}
🟦 node_scrape_collector_duration_seconds{collector="bcache", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="bonding", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="btrfs", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="conntrack", instance="54.83.139.118:9100", job="node"}
🟨 node_scrape_collector_duration_seconds{collector="cpu", instance="54.83.139.118:9100", job="node"}
🟦 node_scrape_collector_duration_seconds{collector="cpufreq", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="diskstats", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="edac", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="entropy", instance="54.83.139.118:9100", job="node"}
🟨 node_scrape_collector_duration_seconds{collector="filefd", instance="54.83.139.118:9100", job="node"}
⬜ node_scrape_collector_duration_seconds{collector="filesystem", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="hwmon", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="infiniband", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="ipvs", instance="54.83.139.118:9100", job="node"}
🟫 node_scrape_collector_duration_seconds{collector="loadavg", instance="54.83.139.118:9100", job="node"}
⬛ node_scrape_collector_duration_seconds{collector="mdadm", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="meminfo", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="netclass", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="netdev", instance="54.83.139.118:9100", job="node"}
🟨 node_scrape_collector_duration_seconds{collector="netstat", instance="54.83.139.118:9100", job="node"}
⬜ node_scrape_collector_duration_seconds{collector="nfs", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="nfsd", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="powersupplyclass", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="pressure", instance="54.83.139.118:9100", job="node"}
🟫 node_scrape_collector_duration_seconds{collector="rapl", instance="54.83.139.118:9100", job="node"}
⬛ node_scrape_collector_duration_seconds{collector="schedstat", instance="54.83.139.118:9100", job="node"}
🟫 node_scrape_collector_duration_seconds{collector="sockstat", instance="54.83.139.118:9100", job="node"}
⬛ node_scrape_collector_duration_seconds{collector="softnet", instance="54.83.139.118:9100", job="node"}
⬛ node_scrape_collector_duration_seconds{collector="stat", instance="54.83.139.118:9100", job="node"}
🟨 node_scrape_collector_duration_seconds{collector="textfile", instance="54.83.139.118:9100", job="node"}
⬜ node_scrape_collector_duration_seconds{collector="thermal_zone", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="time", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="timex", instance="54.83.139.118:9100", job="node"}
🟪 node_scrape_collector_duration_seconds{collector="udp_queues", instance="54.83.139.118:9100", job="node"}
🟧 node_scrape_collector_duration_seconds{collector="uname", instance="54.83.139.118:9100", job="node"}
🟦 node_scrape_collector_duration_seconds{collector="vmstat", instance="54.83.139.118:9100", job="node"}
🟥 node_scrape_collector_duration_seconds{collector="xfs", instance="54.83.139.118:9100", job="node"}
🟩 node_scrape_collector_duration_seconds{collector="zfs", instance="54.83.139.118:9100", job="node"}
```

Click: select series, CTRL + click: toggle multiple series

Graph view of an attribute in Prometheus



Targets

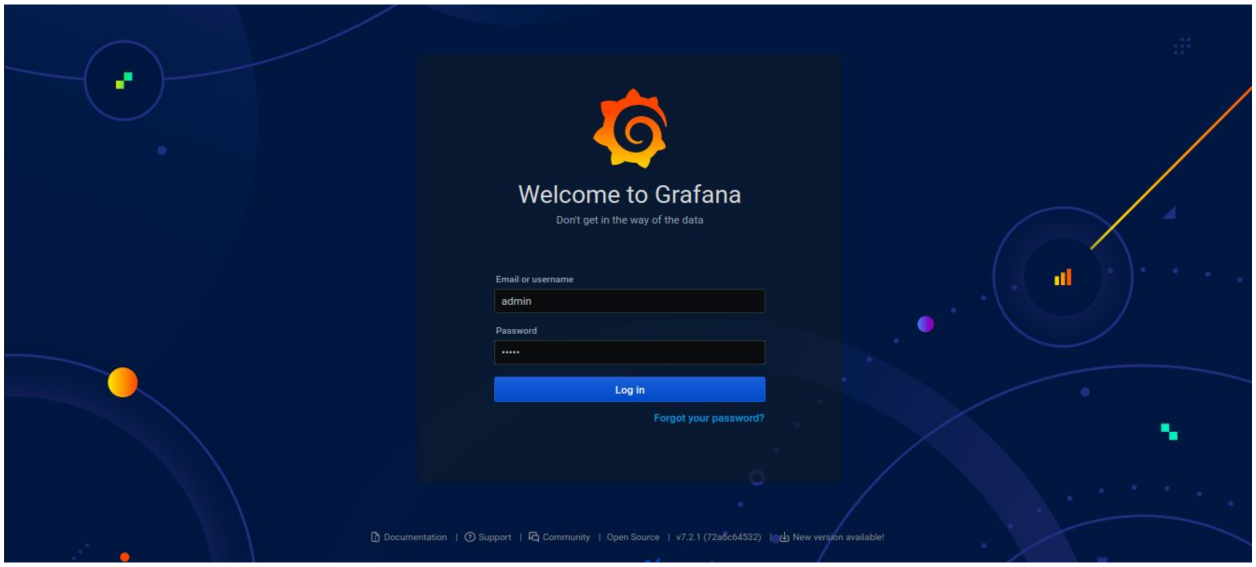
node (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://54.83.139.118:9100/metrics	UP	instance="54.83.139.118:9100" job="node"	2.755s	10.348ms	

prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://54.83.139.118:9090/metrics	UP	instance="54.83.139.118:9090" job="prometheus"	3.469s	6.160ms	

Status of Node Exporter and Prometheus Server


Welcome to Grafana

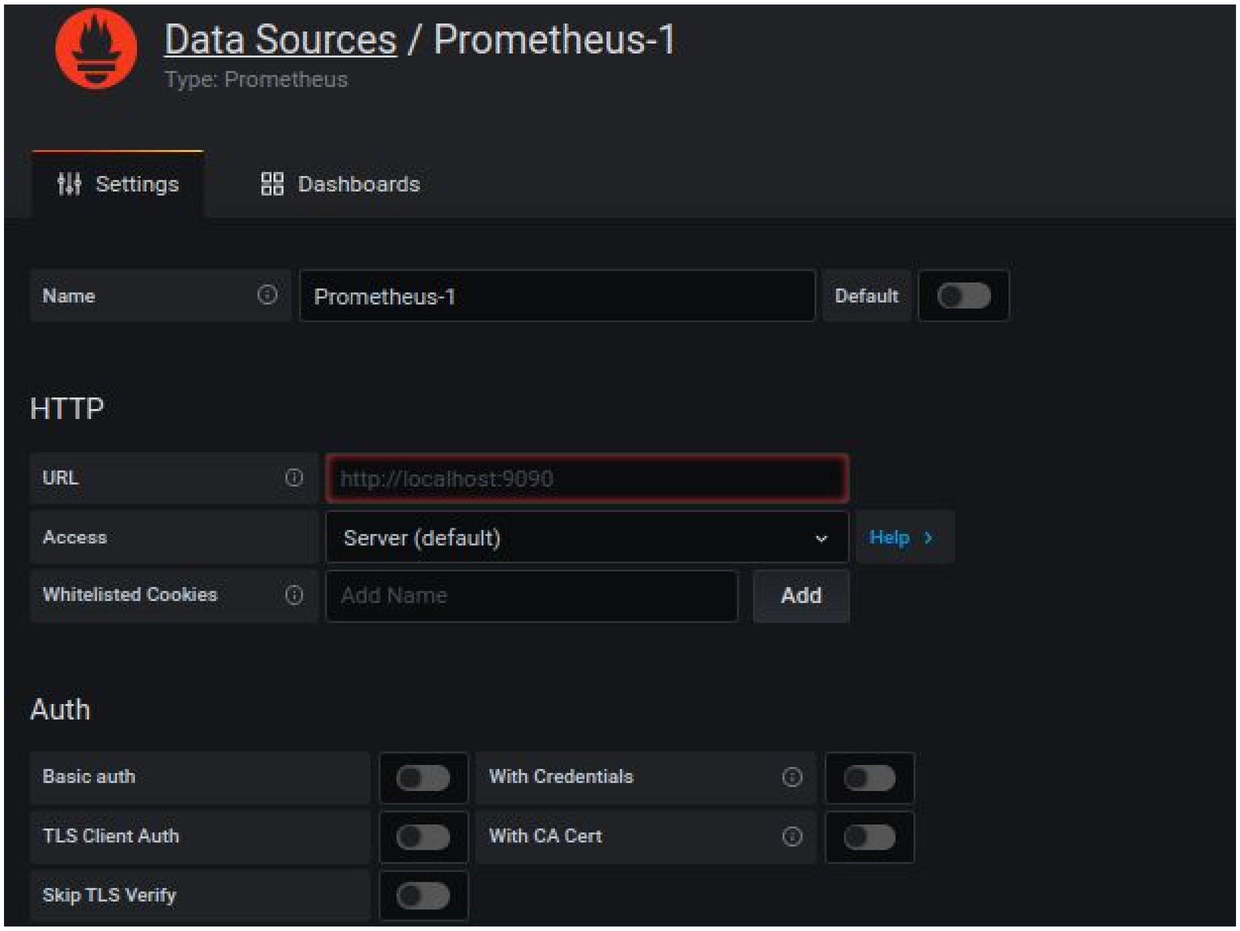
Don't get in the way of the data

Email or username: admin

Password: •••••

Log in
Forgot your password?
Documentation | Support | Community | Open Source | v7.2.1 (72a6c64532) | New version available!

Login view of Grafana



Data Sources / Prometheus-1

Type: Prometheus

Settings | Dashboards

Name	Prometheus-1	Default	

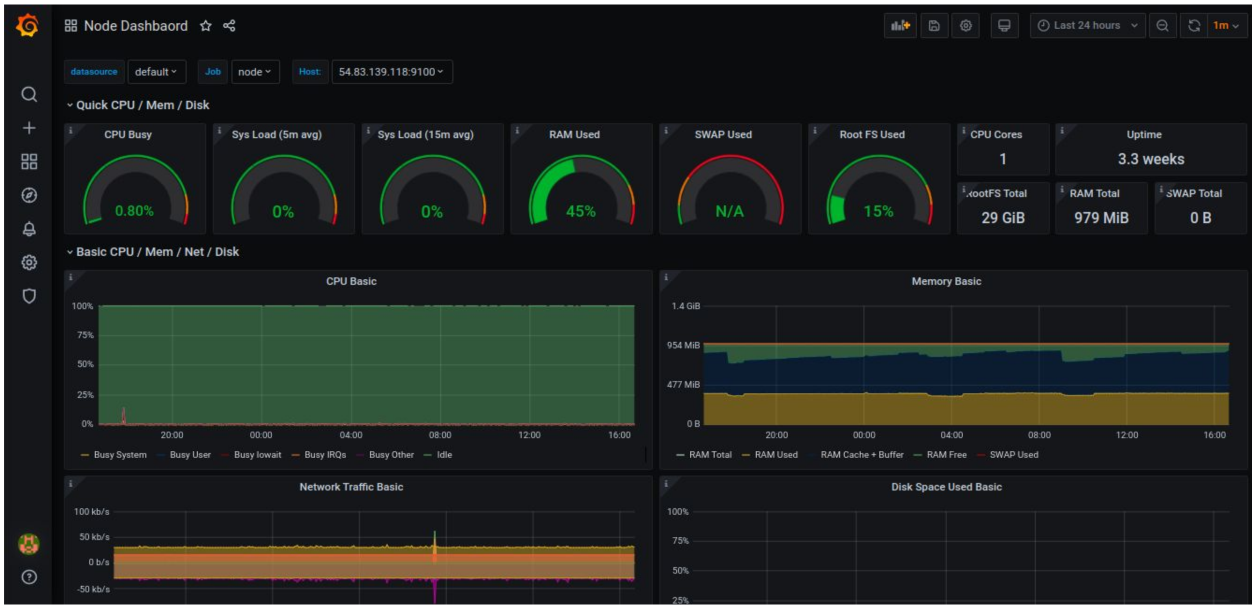
HTTP

URL	http://localhost:9090	
Access	Server (default)	Help
Whitelisted Cookies	Add Name	Add

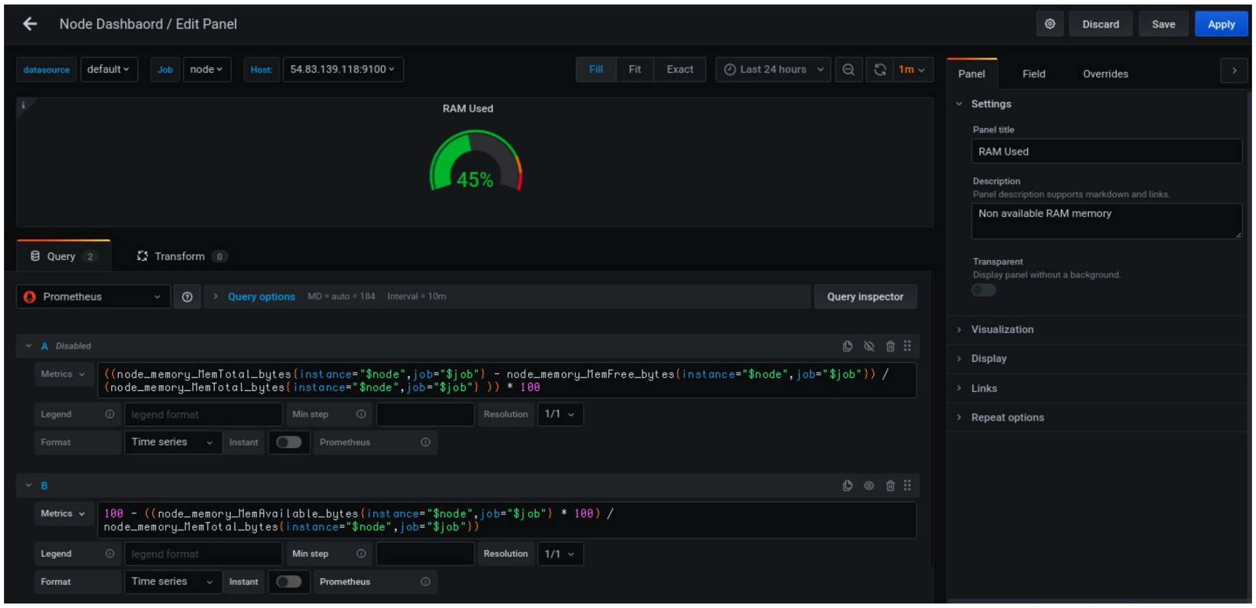
Auth

Basic auth		With Credentials	
TLS Client Auth		With CA Cert	
Skip TLS Verify			

Adding Prometheus as Data Source in Grafana



Dashboard view after setting up the Data Source



Settings

Panel title: RAM Used

Description: Panel description supports markdown and links.
Non available RAM memory

Transparent: Display panel without a background.

A Disabled

Metrics:
```
((node_memory_MemTotal_bytes{instance="$node",job="$job"} - node_memory_MemFree_bytes{instance="$node",job="$job"}) /
(node_memory_MemTotal_bytes{instance="$node",job="$job"} )) * 100
```
Legend: legend format | Min step | Resolution: 1/1
Format: Time series | Instant | Prometheus

B

Metrics:
```
100 - ((node_memory_MemAvailable_bytes{instance="$node",job="$job"} * 100) /
node_memory_MemTotal_bytes{instance="$node",job="$job"})
```
Legend: legend format | Min step | Resolution: 1/1
Format: Time series | Instant | Prometheus

Editing the attributes in dashboard in Grafana

6. REFERENCES

[1]https://www.digitalocean.com/community/tutorials/how-to-install-prometheus-using-docker-on-ubuntu-14-04

[2]https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-14-04

[3]https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-14-04