**Monitoring using Prometheus and Grafana**

Table of Contents

1 Introduction to Prometheus 3

1.1 What is Prometheus?3

1.2 Why Prometheus?3

1.3 Components of Prometheus.3

2 Setup Prometheus 4

2.1 Installation using Docker 4

2.2 Configuration File 5

2.3 Dashboard 6

3 Monitoring server, container, cluster using Prometheus.8

3.1 Server Monitoring 8

3.2 Container Monitoring10

3.3 Service Monitoring11

3.4 Cluster Monitoring12

4 Introduction to Grafana 14

4.1 What is Grafana 14

4.2 Components of Grafana 14

5 Installation and Dashboard 15

5.1 Installation 15

5.2 Dashboard 15

5.2.1 Adding a data source ………………………………………………………………………………………….16

5.2.2 Dashboard creation ..………………………………………………………………………………………….16

* 1. What is Prometheus?

Prometheus, a [Cloud Native Computing Foundation](https://cncf.io/) project, is a systems and service monitoring system. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true.

* 1. **Why Prometheus?**

Prometheus' main distinguishing features as compared to other monitoring systems are:

* a **multi-dimensional** data model (timeseries defined by metric name and set of key/value dimensions)
* a **flexible query language** to leverage this dimensionality
* no dependency on distributed storage; **single server nodes are autonomous**
* timeseries collection happens via a **pull model** over HTTP
* **pushing timeseries** is supported via an intermediary gateway
* targets are discovered via **service discovery** or **static configuration**
* multiple modes of **graphing and dashboarding support**
* support for hierarchical and horizontal **federation**
  1. **Components** **-**

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

* the main [Prometheus server](https://github.com/prometheus/prometheus) which scrapes and stores time series data
* [client libraries](https://prometheus.io/docs/instrumenting/clientlibs/) for instrumenting application code
* a [push gateway](https://github.com/prometheus/pushgateway) for supporting short-lived jobs
* special-purpose [exporters](https://prometheus.io/docs/instrumenting/exporters/) for services like HAProxy, StatsD, Graphite, etc.
* an [alertmanager](https://github.com/prometheus/alertmanager) to handle alerts
* various support tools

**2) Installation -**

**2.1) Using Docker -**

Running Prometheus on Docker is as simple as docker run -p 9090:9090 prom/prometheus. This starts Prometheus with a sample configuration and exposes it on port 9090.

The Prometheus image uses a volume to store the actual metrics. For production deployments it is highly recommended to use the [Data Volume Container](https://docs.docker.com/engine/admin/volumes/volumes/) pattern to ease managing the data on Prometheus upgrades.

- Bind-mount your prometheus.yml from the host by running:

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

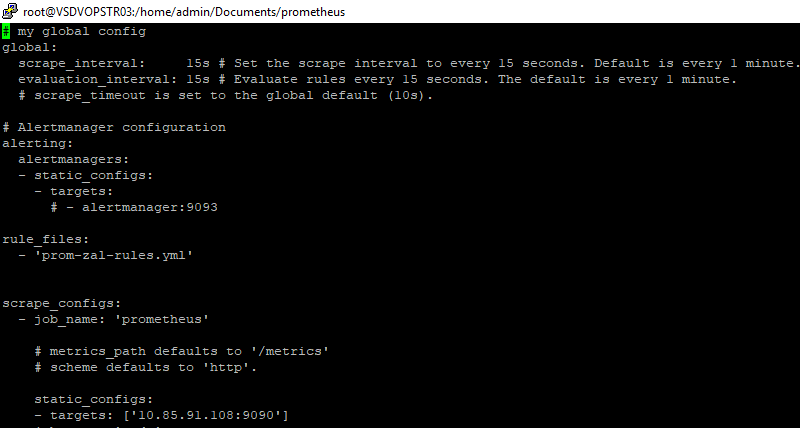
infydp.ad.infosys.com:8082/prom/prometheus

Here we are running Prometheus image inside the container by volume mounting the the main config file of Prometheus I.e Prometheus.yml file to the container. Below we have specified the default config file of Prometheus.

**2.2) Configuration file -**

- To specify which configuration file to load, use the --config.file flag.

The file is written in [YAML format](https://en.wikipedia.org/wiki/YAML),



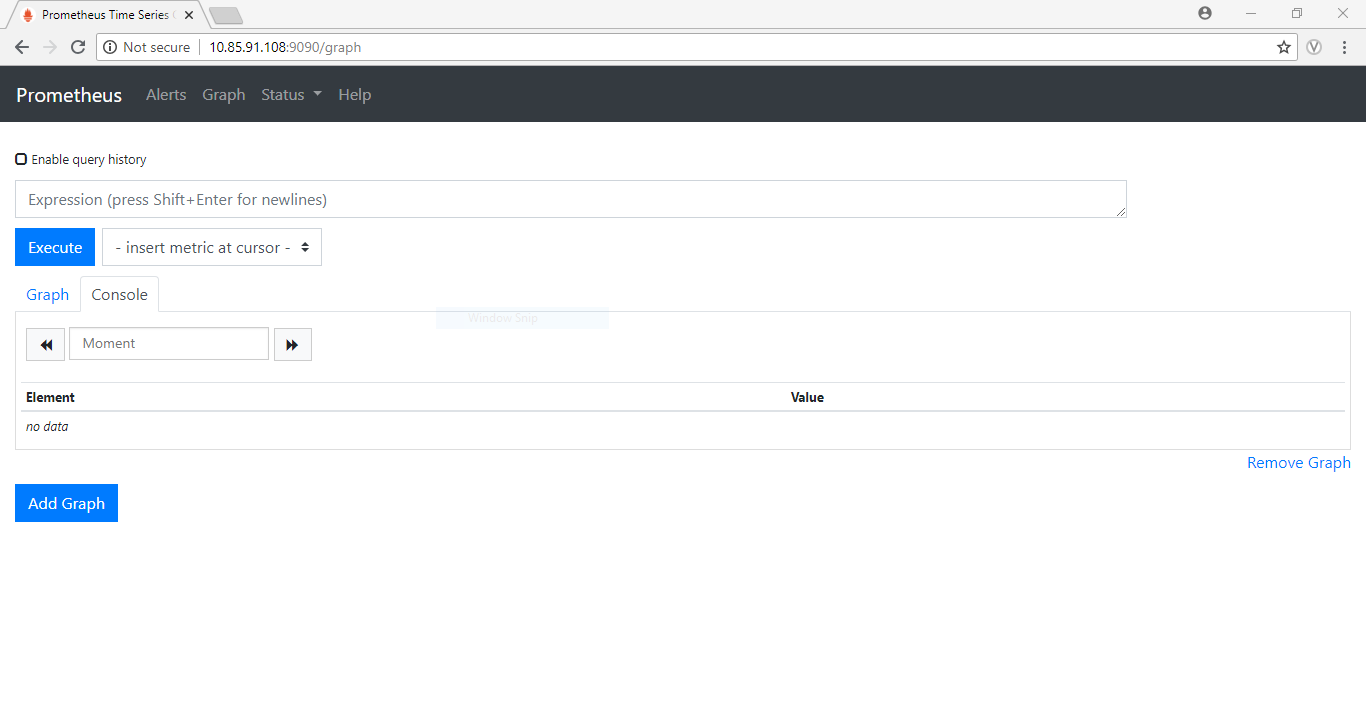
1. Above is the default configuration file of Prometheus which is retrieving metrics of promtheus itself.
2. The config file can be divided into 4 main tags-

* Global config- The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.
* Alertmanager configuration- An alertmanager\_config section specifies Alertmanager instances the Prometheus server sends alerts to. It also provides parameters to configure how to communicate with these Alertmanagers.
* Rule-files – a rule file is a yml file in which we specify a proper condition to which an alert will be generated
* scrape\_config- A scrape\_config section specifies a set of targets and parameters describing how to scrape them. We can specify multiple jobs in single scrape config tag.

There are two main tags inside the scrape\_config.

1. Job\_name- we can specify the name of the service/server we want to monitor
2. Static\_configs: inside the static\_config we can specify the target from which we want metrics from. For example, here we have to specify the external ip and node port of the Prometheus, so that Prometheus can get metrics of itself.

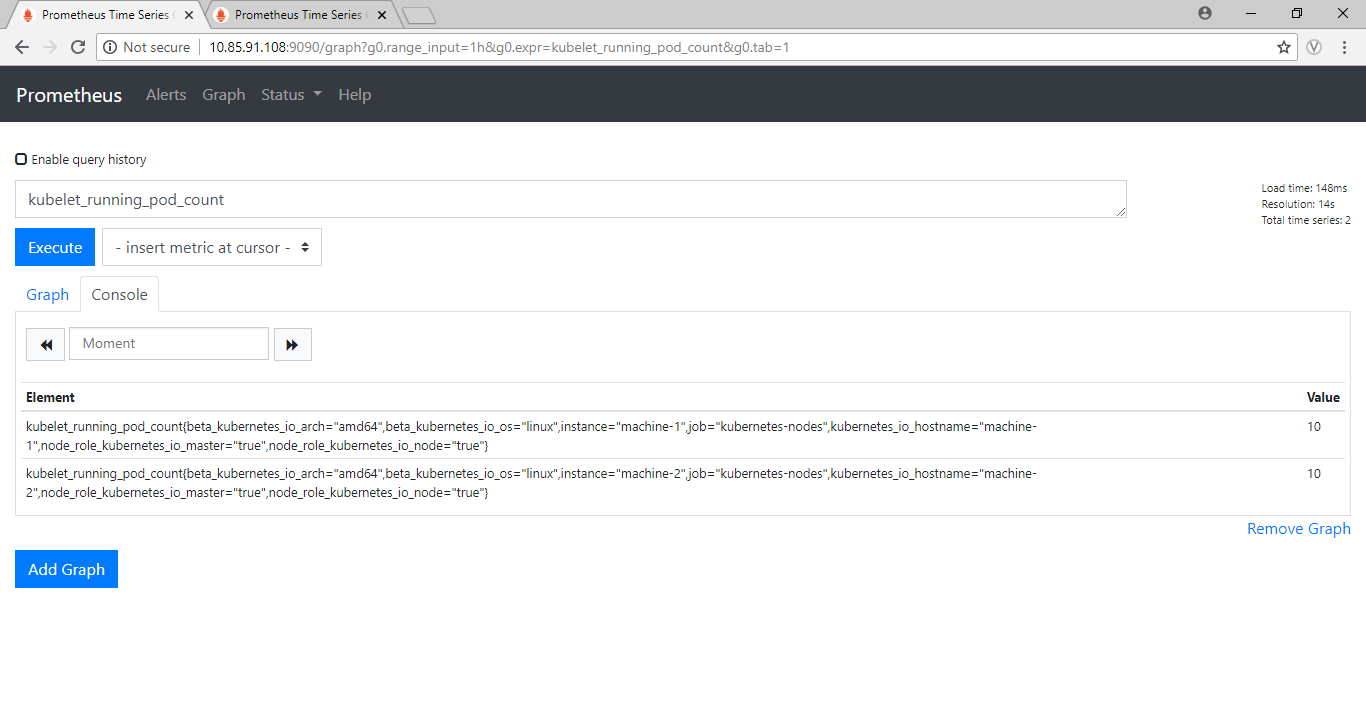
**2.3) Dashboard -**



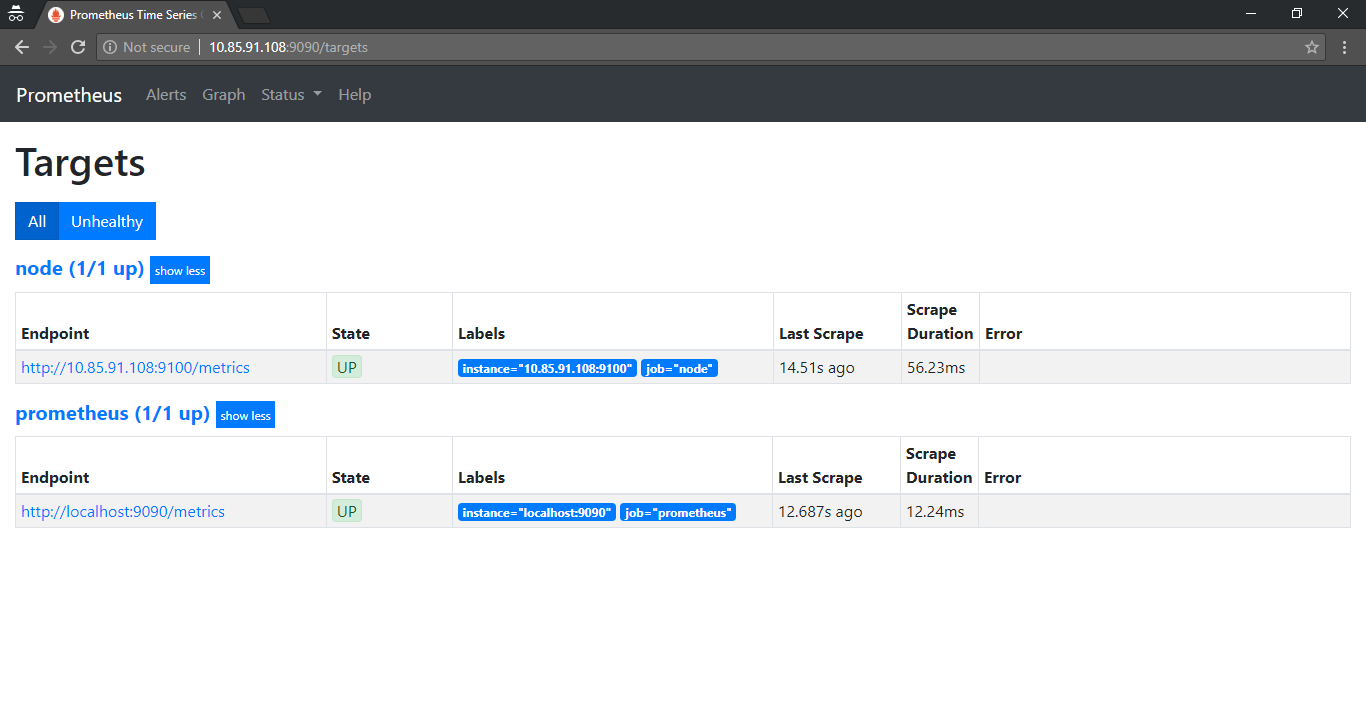
This is the promethues dashboard, we can provide the query according our requirement.

Promethues uses its own query language i.e PromQL.

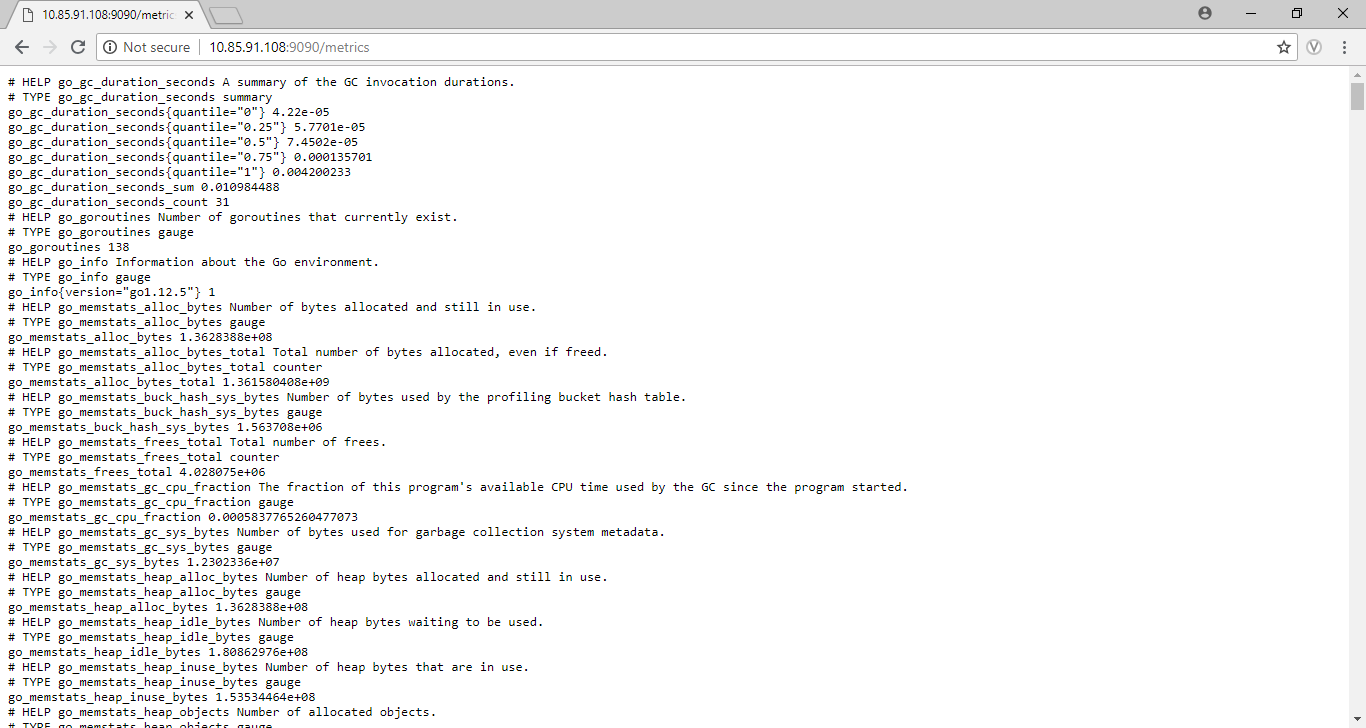
You can learn more about PromQL from this following [link](https://prometheus.io/docs/prometheus/latest/querying/basics/).



Here we want to retrieve data of running pods count inside kubernetes cluster, so we are executing kubelet\_running\_pod\_count command to retrieve the data.



Here we can all the target we specified inside the Prometheus config file, for example you can see there are two target i.e node and Prometheus , where node is used get the metrics about the server in which the node exporter is running and Prometheus which retrieves metrics about Prometheus itself.



This is some of the metrics provided by Prometheus, we can access it using with the http://ip:port/metrics url where /**metrics** will provide you with all the metrics.

**3)Monitoring server, container, cluster using Prometheus.**

* 1. **Server monitoring –**

To monitor a specific server, we need to use node-exporter of Prometheus which help us to retrieve metrics of that server in which the node exporter is running.

Usually the node exporter is exposed on port **9100**.

We can use official docker image provided by Prometheus itself to run the node exporter on a specific server machine. You can use the [Github](https://github.com/prometheus/node_exporter) link to understand more about this.

docker run -d \

--net="host" \

--pid="host" \

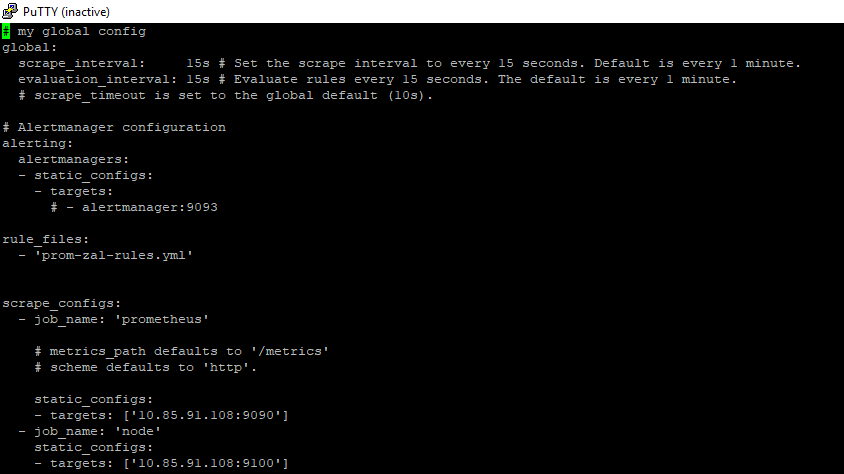
-v "/:/host:ro,rslave" \

Infydp.ad.infosys.com:8082/prom/node-exporter \

--path.rootfs /host

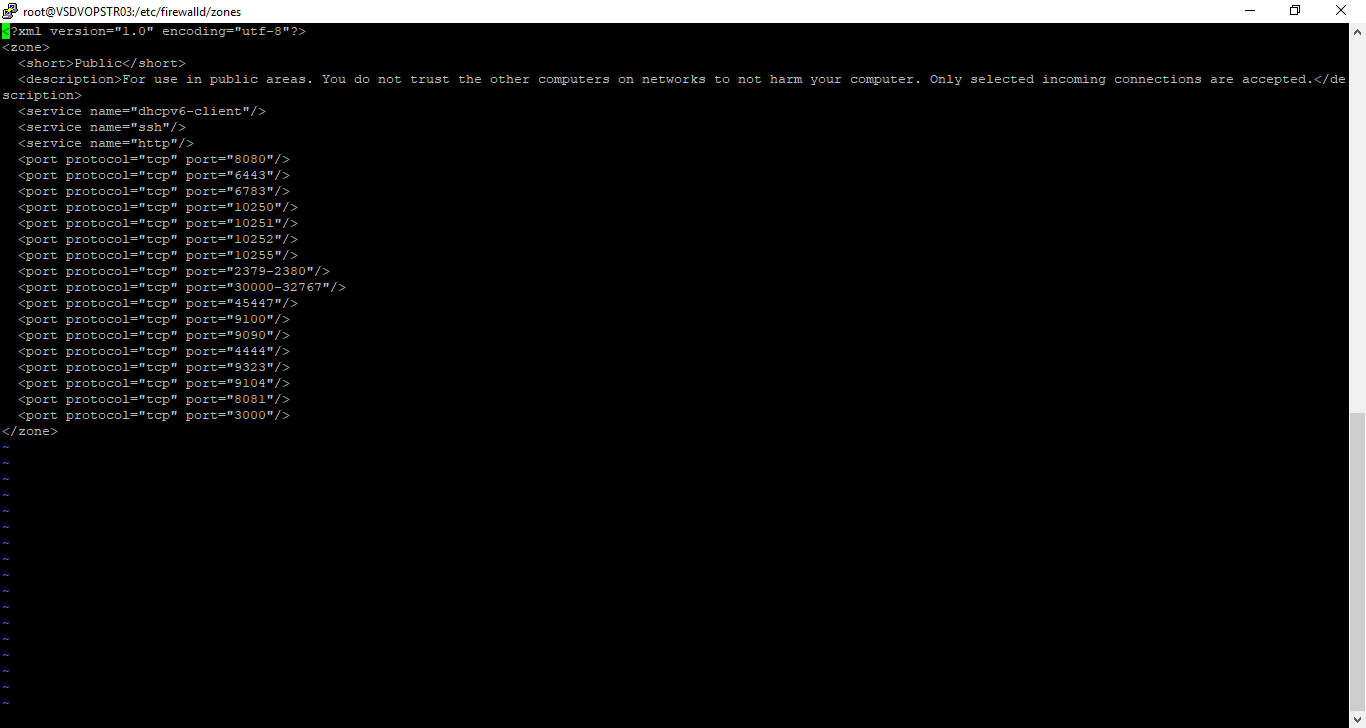
This will help us to retrieve metrics from the server machine in which we are running the container.

Now we have to specify the ip of the machine and the exposed port number of the node exporter in the Prometheus config file.

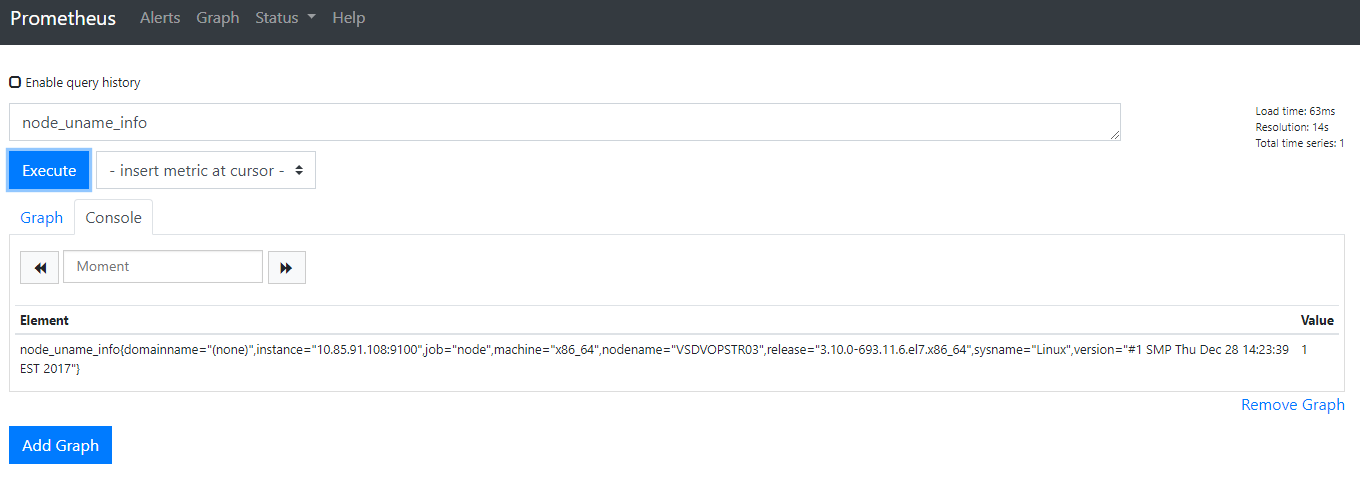


If firewalld service is running on your machine then do the following steps –

* After specifying the port in the Prometheus config file, we have to mention the same port number in the public.xml file of firewalld of linux for routing purpose.



* Path of this file is - /etc/firewalld/zones/public.xml
* Inside this we have to give the port number we want to route and after editing and saving this file we have to restart the firewall.
* **systemctl restart firewalld**

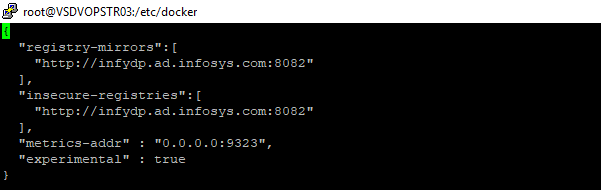


Now to retrieve specific metrics we can provide the required PromQL expression.

Here to get the detailed info about the server, we are executing the node\_uname\_info command which in return gives us the basic info of our server.

* 1. **Container Monitoring –**

To monitor our docker container, we have to some changes in our docker’s daemon.json file.



Here we have to add some extra parameters – **metrics-addr** – which will help docker to expose its metrics on the given port number and Prometheus can access those metrics by providing the target with this port number. You can follow this [link](https://docs.docker.com/config/thirdparty/prometheus/) for in-depth understanding of this.

After specifying the metrics-addr flag inside the daemon.json file we have to restart docker so that it can be implemented.

1. systemctl daemon-reload
2. systemctl restart docker

Then we have to specify this port inside the config of Prometheus so that It can get the metrics of the container.



Here we providing the name as docker-monitor and specifying target with ip and the port number we specified in the daemon.json file of docker.

**Using cAdvisor** –

cAdvisor (Container Advisor) provides container users an understanding of the resource usage and performance characteristics of their running containers. It is a running daemon that collects, aggregates, processes, and exports information about running containers.

We can use cadvisor to get more metrics about the containers.

We can implement cadvisor using docker image.

You can use this [GitHub](https://github.com/google/cadvisor) link for more information.

docker run \

--volume=/:/rootfs:ro \

--volume=/var/run:/var/run:ro \

--volume=/sys:/sys:ro \

--volume=/var/lib/docker/:/var/lib/docker:ro \

--volume=/dev/disk/:/dev/disk:ro \

--publish=8080:8080 \

--detach=true \

--name=cadvisor \

Infydp.ad.infosys.com:8082/google/cadvisor:latest

It will be exposed on 8080 port and we can specify this port number inside the Prometheus config file so that Prometheus can access the metrics provided by cadvisor.

* 1. **Service Monitoring** –

Here we are going to monitor zalenium as a service.

What is zalenium-

This is a Selenium Grid extension to scale your local grid dynamically with docker containers. It uses [docker-selenium](https://github.com/elgalu/docker-selenium) to run your tests in Firefox and Chrome locally.

Zalenium is exposed on 4444 port, so we can specify the port number inside Prometheus config file so it can retrieve zalenium metrics.



* 1. **Cluster Monitoring –**

For cluster monitoring, we were not able to get metrics as Prometheus was deployed outside the cluster and we were facing some authentication issues so in-order to overcome this we deployed a new Prometheus instance inside the cluster. Now we can retrieve metrics from cluster with no authentication issues. So let us deploy a new Prometheus instance inside the cluster machine.

1. Create a namespace:

kubectl create namespace monitoring

1. Create a file named clusterRole.yaml:

You can find all the yml files in this git repo - <https://infygit.ad.infosys.com/solutions/promethus>

kubectl create -f clusterRole.yaml

1. Create a file called config-map.yaml:

kubectl create -f config-map.yaml -n monitoring

1. Create A Prometheus Deployment

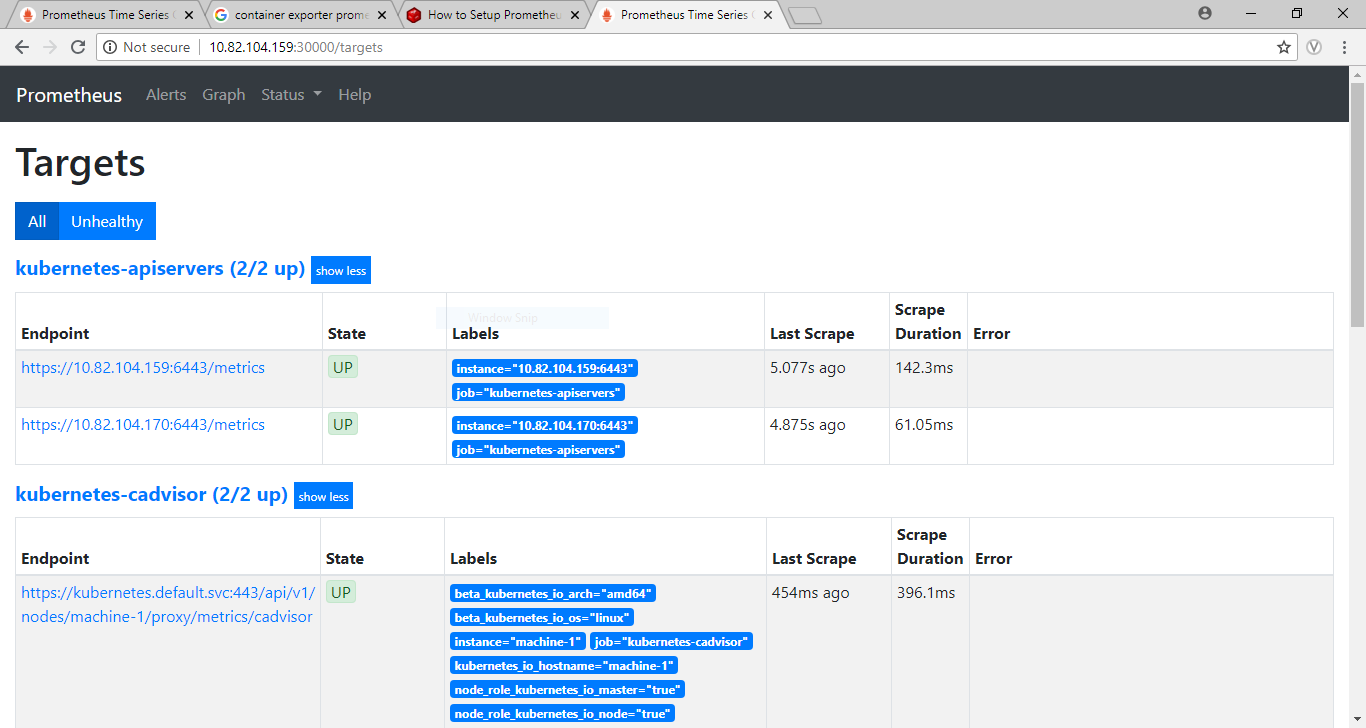
kubectl create -f prometheus-deployment.yaml --namespace=monitoring

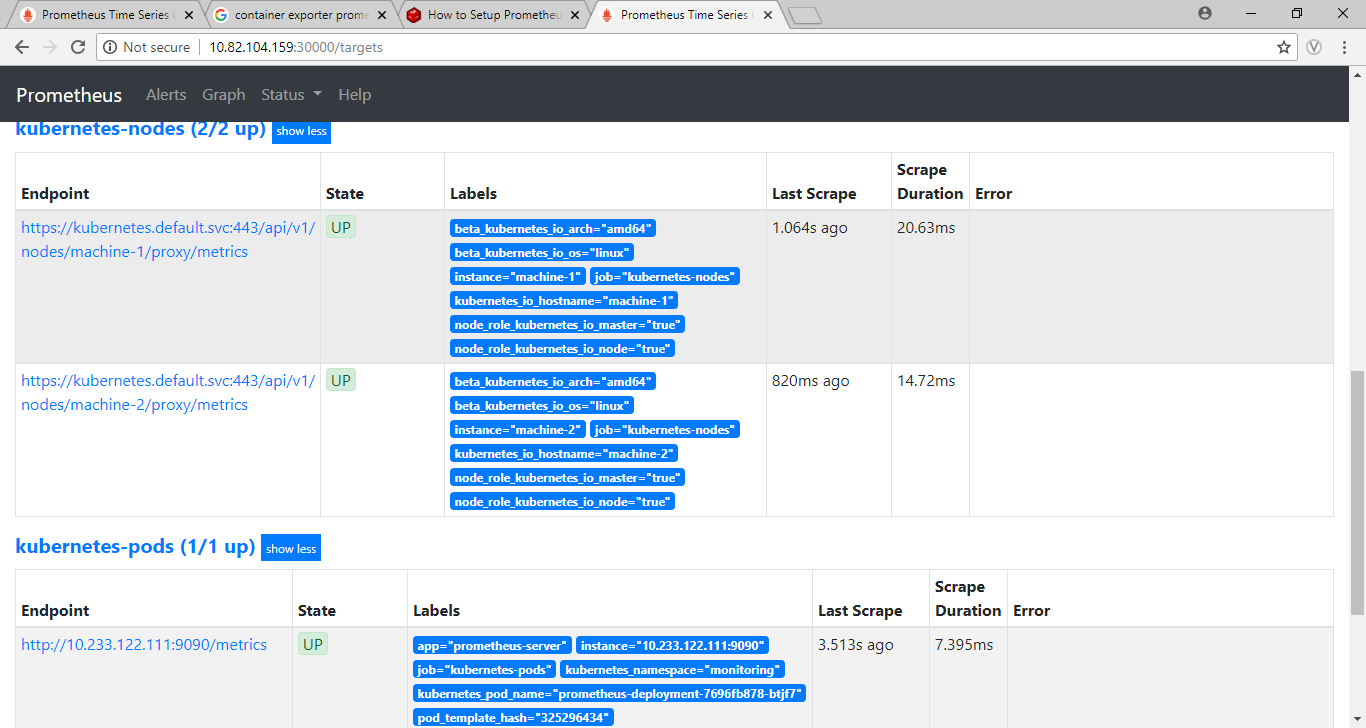
1. Exposing Prometheus as a Service

To access the Prometheus dashboard over a IP or a DNS name, you need to expose it as kubernetes service.

kubectl create -f prometheus-service.yaml --namespace=monitoring

Once created, you can access the Prometheus dashboard using any Kubernetes node IP on port 30000.





Grafana –

**4.1) What is grafana?**

Grafana is an open-source platform for data visualization, monitoring and analysis. Grafana also supports data sources like Prometheus, MySQL, Postgres and many more. For each data source, Grafana has a customized query editor and specific syntax.

In Grafana, users can to create dashboards with panels, each representing specific metrics over a set time-frame. Grafana supports graph, singlestat, table, heatmap and freetext panels as well as integration with official and community-built plugins (like world map or clock) and apps that could be visualized too. Every dashboard and panel is versatile, so it could be custom-tailored for a specific project or any development and/or business needs. Also, users could add annotations into the panels to track specific events and reference to their consequences, e.g. new releases, runtime errors etc.

Here we will use grafana as a data visualization tool for our Prometheus data source.

**4.2) Components –**

1. **Data Source -** Grafana supports many different storage backends for your time series data (Data Source). Each Data Source has a specific Query Editor that is customized for the features and capabilities that the particular Data Source exposes.
2. **Panel -** The Panel is the basic visualization building block in Grafana. Each Panel provides a Query Editor (dependent on the Data Source selected in the panel) that allows you to extract the perfect visualization to display on the Panel by utilizing the Query Editor
3. **Query Editor -** The Query Editor exposes capabilities of your Data Source and allows you to query the metrics that it contains.

Use the Query Editor to build one or more queries (for one or more series) in your time series database. The panel will instantly update allowing you to effectively explore your data in real time and build a perfect query for that particular Panel.

1. **Dashboard** - The Dashboard is where it all comes together. Dashboards can be thought of as of a set of one or more Panels organized and arranged into one or more Rows.
2. **User -** A User is a named account in Grafana. A user can belong to one or more Organizations, and can be assigned different levels of privileges through roles

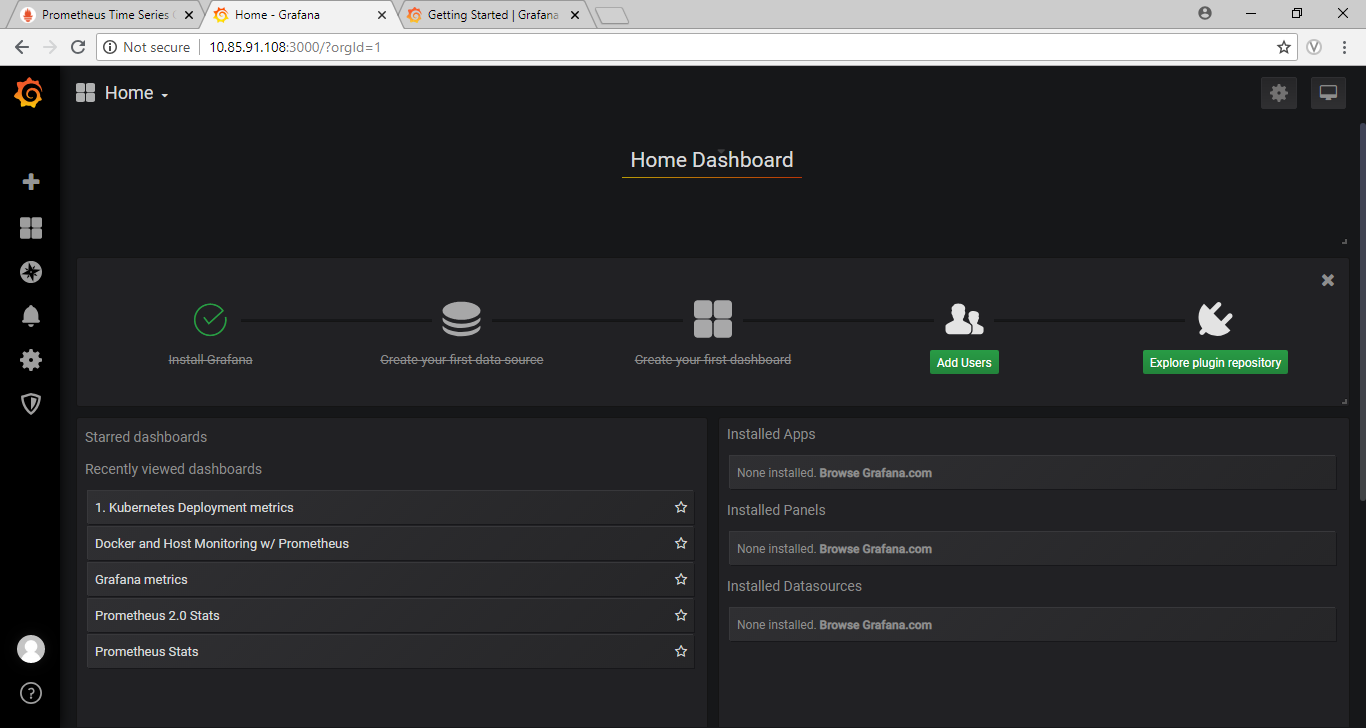
**5.1) Installation –**

We can install grafana with docker:

$ docker run -d -p 3000:3000 infydp.ad.infosys.com:8082/grafana/grafana

Now grafana is exposed on port 3000.  Default username is admin and default password is admin.

**5.2) Dashboard -**



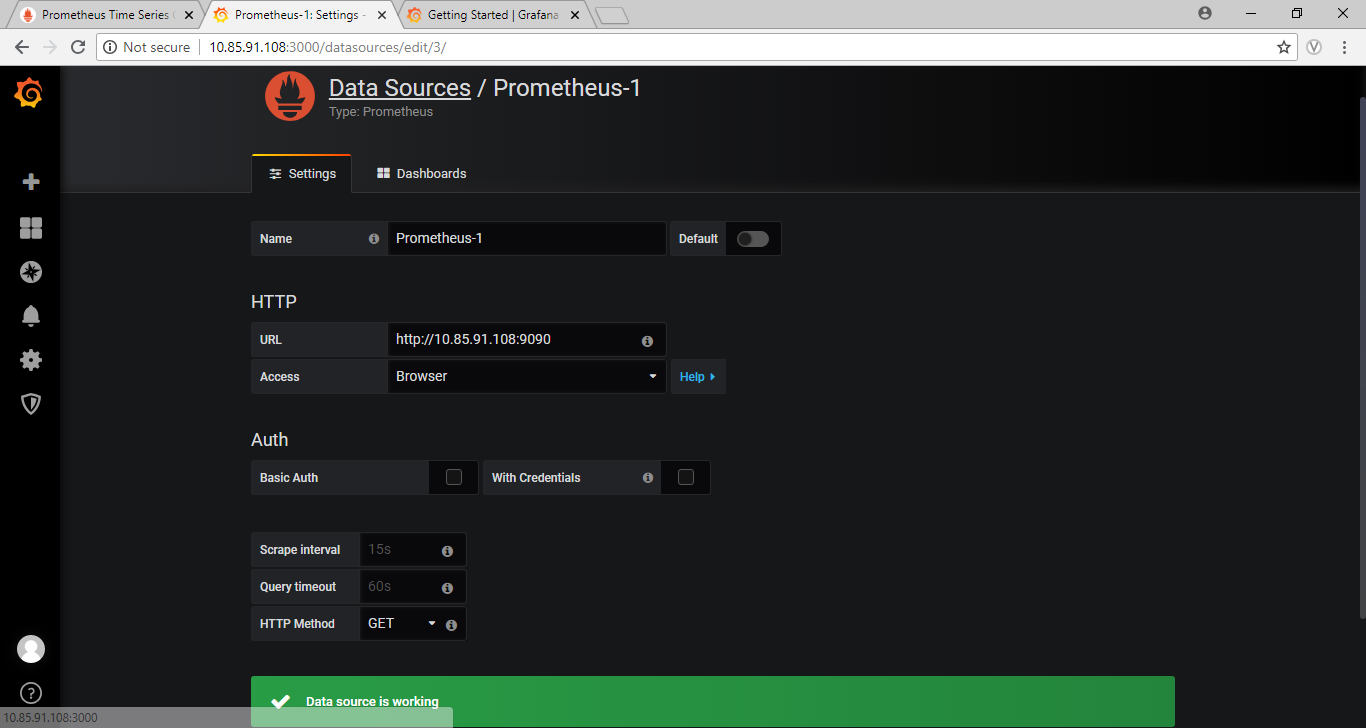
**5.2.1) Adding a data source –**

Here we are using Prometheus as our data source for grafana.

1. Once we are logged in, go to the configuration menu.

2. Select data source inside configuration menu.

1. Click add data source and choose Prometheus as your data source.

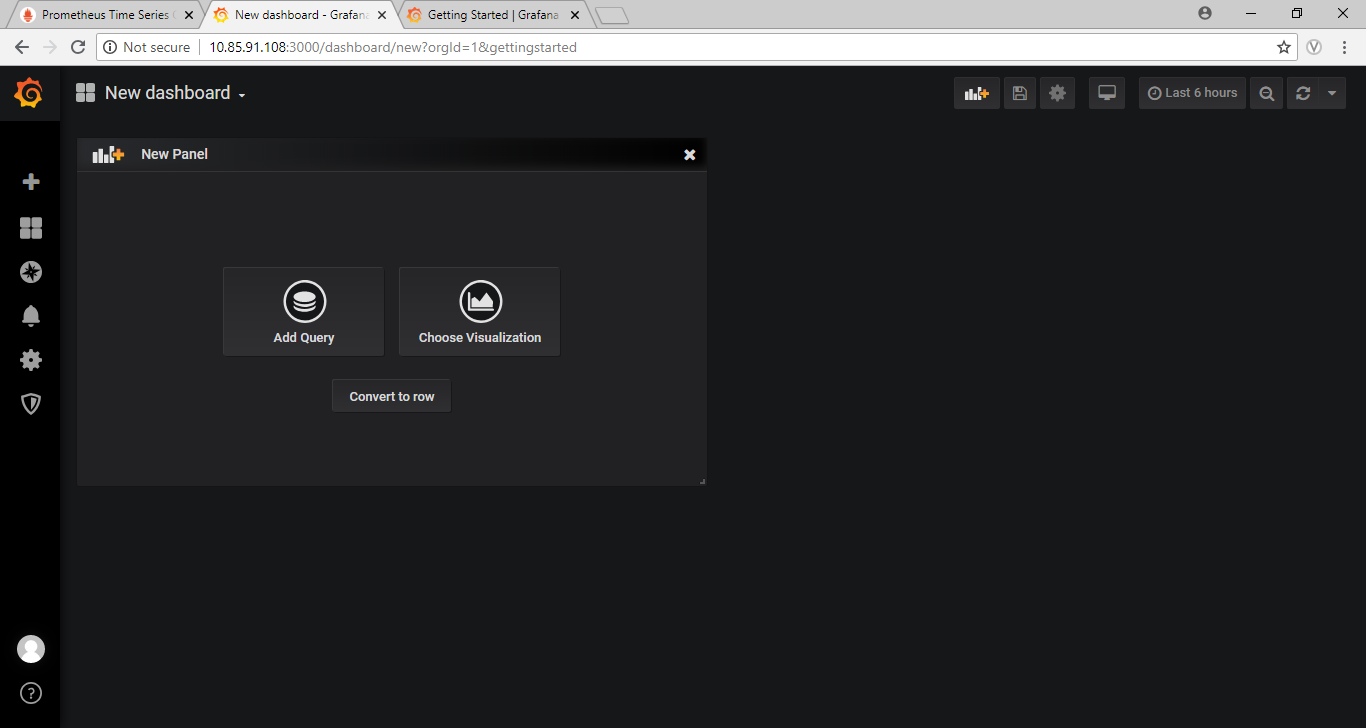


Enter your data source name, url of your Prometheus instance and change the access type to browser according to the image and then click save and test if it gives data source is working your connection to the Prometheus server is done.

**5.2.2) Dashboard Creation –**

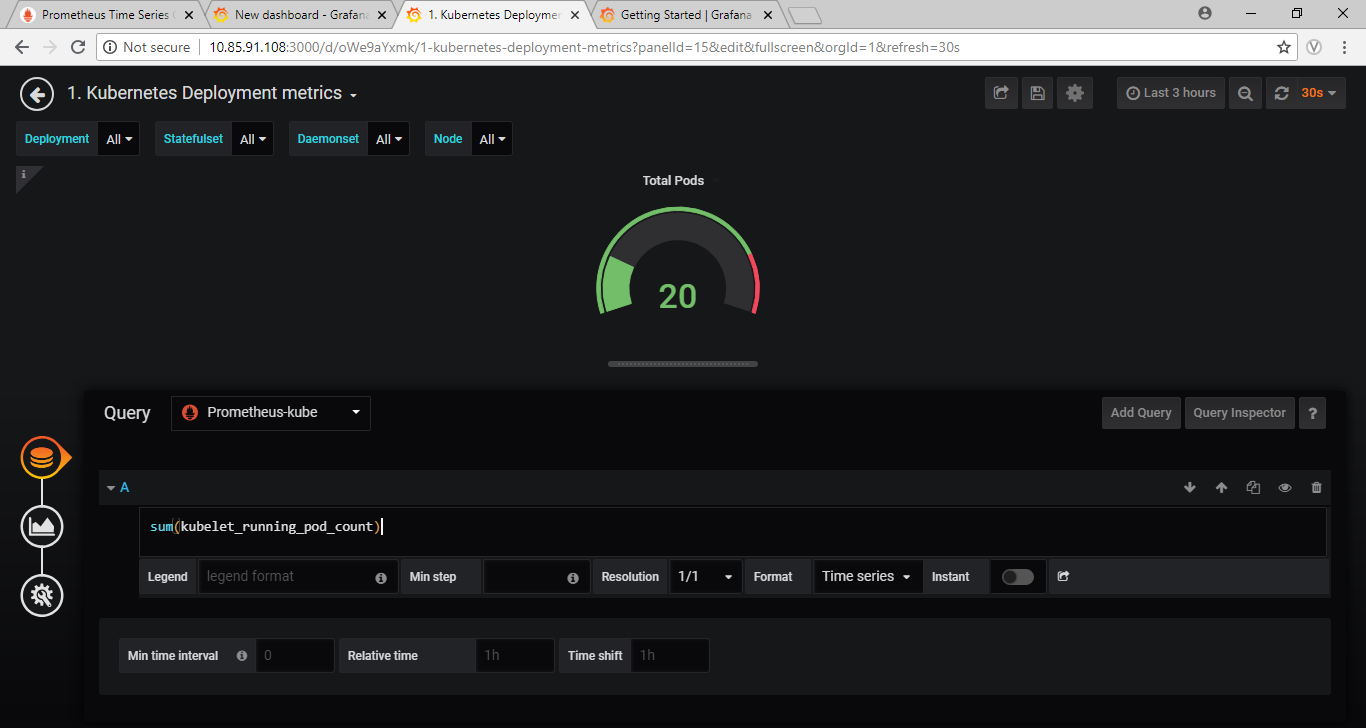
There are some default dashboard created by grafana itself i.e Grafana metrics, Prometheus stats you can import those dashboards and also you can create your own dashboard.

1. Click on create dashboard menu.



Here you get an option to add a query to your panel or choose a visualization i.e graph, heatmap etc.

1. Now let’s choose a visualization or a query for our panel.



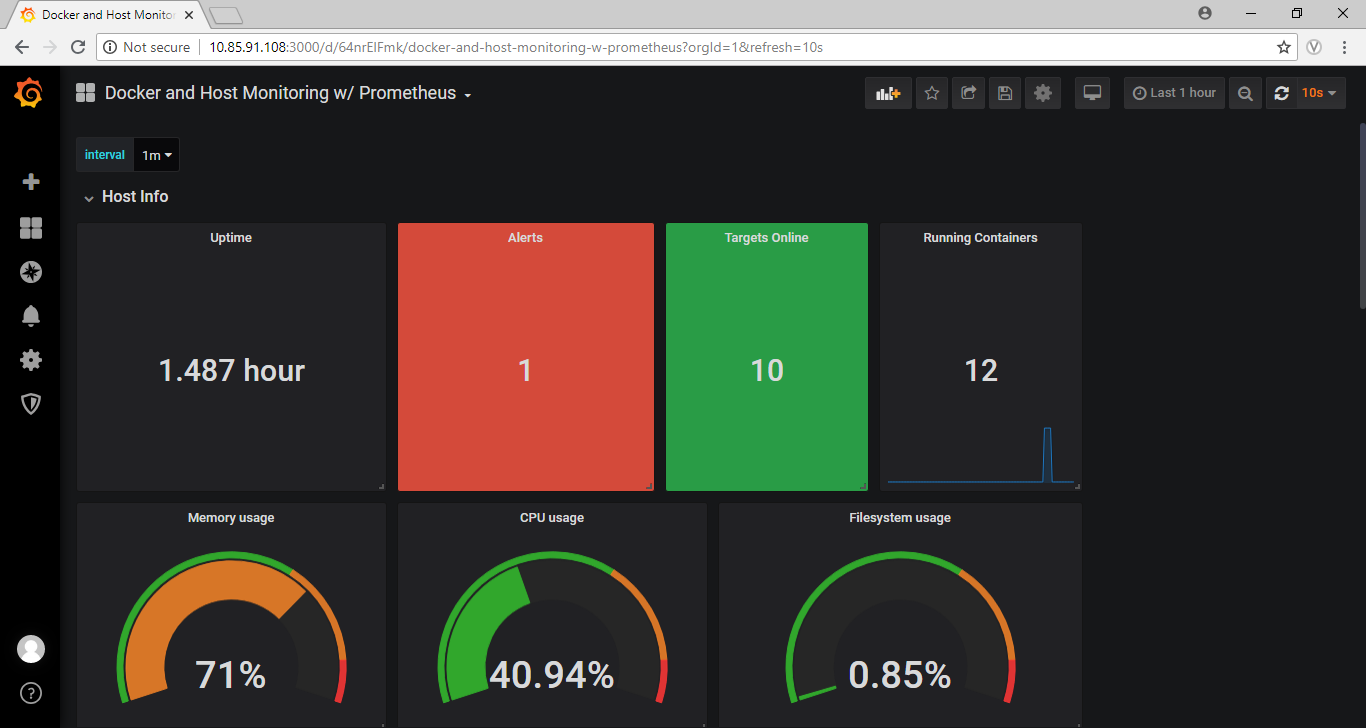
In this you can see that we have written a query to retrieve total number of pods running inside the kubernetes cluster and we have represented with the gauge visualization. You can change the visualization to anyform by go to the visualization menu I.e present below the query menu on the left side.

We can also download the dashboard created by the community i.e in <https://grafana.com/dashboards>

For example, we want a proper docker container monitoring dashboard. We can search it in the dashboard link above and download the dashboard.

To download the dashboard, first select the dashboard you like and then click on the download JSON on the right side, this will download the json format of the all dashboard and now to use the downloaded dashboard click on create menu of the grafana instance and the select import in which you will be provided with an option to upload the downloaded JSON file. Once you upload it your dashboard will be created.

**Container dashboard –**



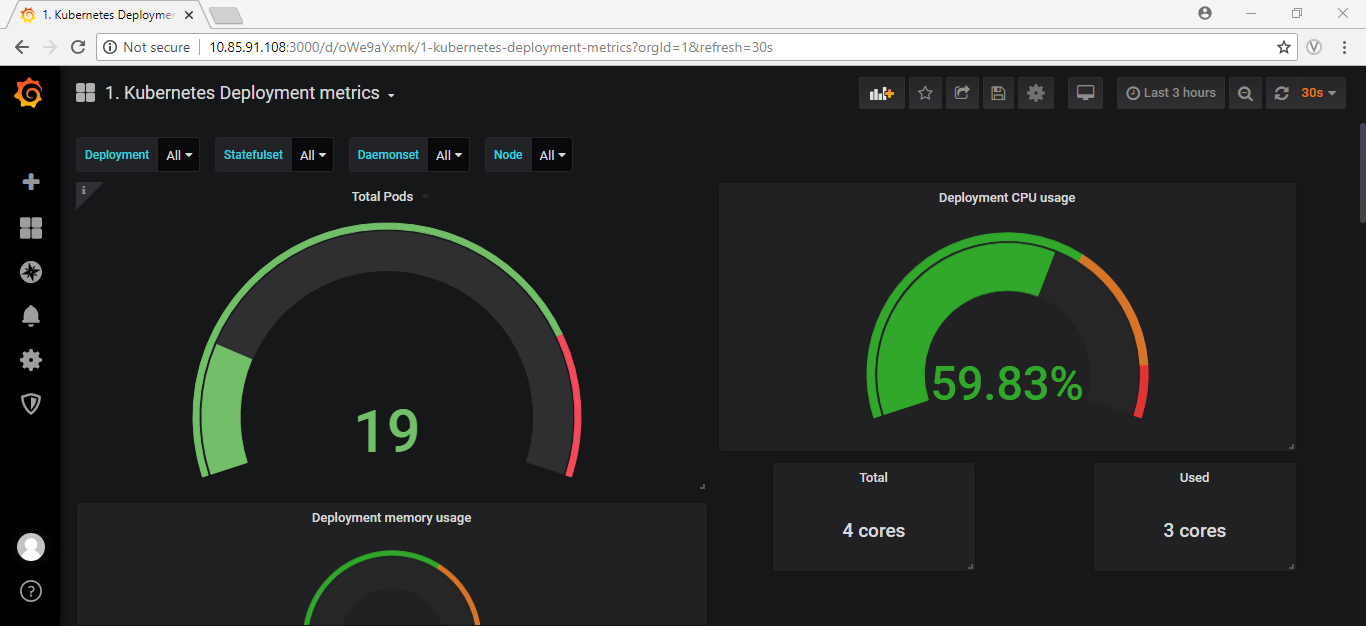
This contains all the important aspect of the container from running container to cpu and memory usage and many more. You can use this dashboard by downloading from the following link.- <https://grafana.com/dashboards/179>.

You can also create a dashboard for kubernetes cluster monitoring. But for that we have to add another data source as we have to deployed different Prometheus instance inside the cluster.

You can create the data source by following the same method as above and also we have a community made kubernetes dashboard for monitoring our cluster. You can go to this link to get the dashboard - <https://grafana.com/dashboards/8588> download and import by following the same method as above.

While importing your dashboard you will be redirected to a window where you will be allowed to select a data source for your dashboard, select the new data source you have created i.e Prometheus server deployed inside the cluster.

**Cluster Dashboard –**



This dashboard contains all the important panels regarding the network, cpu and memory usage. You can always add your own panel by providing the visualization model and the query according to your requirement.