PROMETHEUS AND GRAFANA

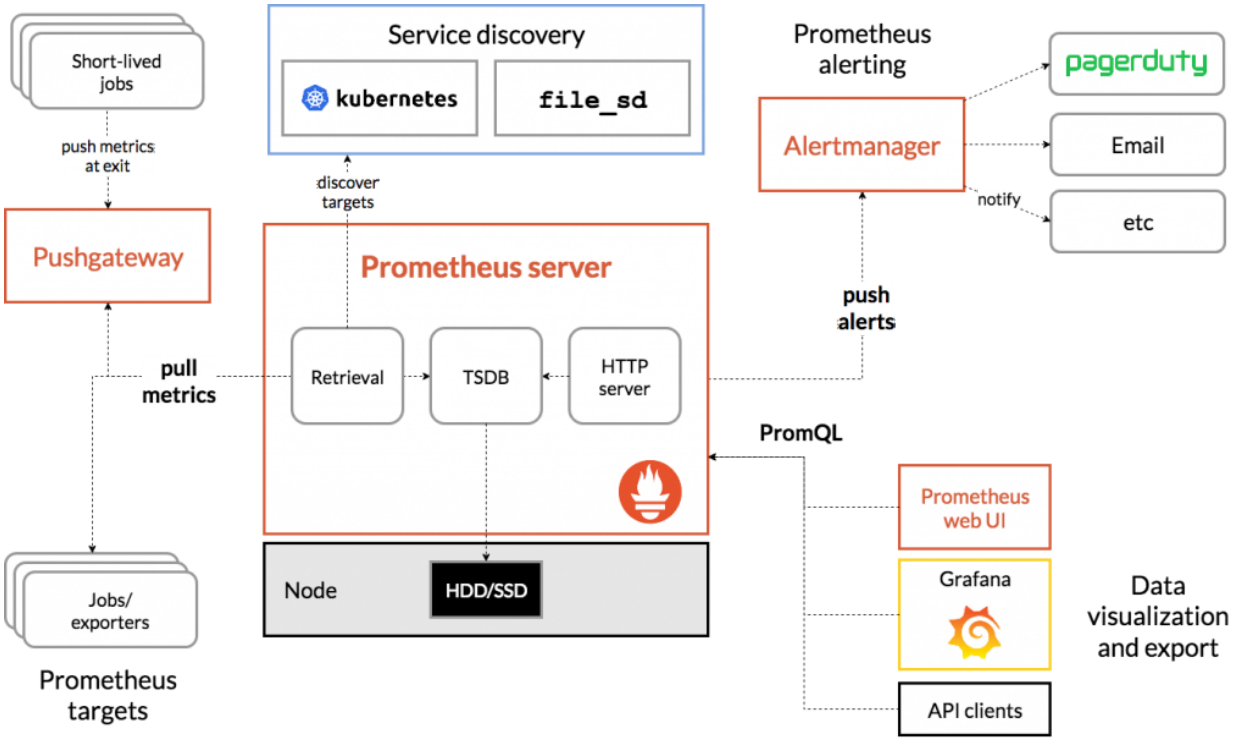
What is Prometheus?

Prometheus is an open-source monitoring solution written in Go that collects metrics data and stores that data in a time series database. It was originally built by SoundCloud in 2012 and became part of the Cloud Native Computing Foundation (CNCF) in 2016. It uses PromQL, a powerful query language for querying your time series data.

Features of Prometheus

- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection happens via a pull model over HTTP
- Pushing time series is supported via an intermediary gateway
- Targets are discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

Architecture of Prometheus



What is Grafana?

Grafana helps us by allowing us to query, visualize, alert and explore our metrics no matter where they are stored, it also helps in providing us with the tools for turning our time series database data into beautiful graphs and visualizations.

Grafana Dashboard

Grafana Dashboard contains data form the plugged-in data sources like Graphite, Prometheus, Influx DB, ElasticSearch, MySQL, PostgreSQL, etc. The dashboard contains a gamut of visualization options such as geo maps, heat maps, histograms, all the variety of charts & graphs which a business typically requires to study data.

Uses of Grafana in Devops

Grafana is an open source, feature-rich metrics dashboard and graph editor. It supports Graphite, Elasticsearch, OpenTSDB, Prometheus, and InfluxDB analytics services. Use the Grafana service hook to annotate Grafana dashboards upon completion of Azure Pipelines deployments.

Need Prometheus and Grafana

Prometheus collects rich metrics and provides a powerful querying language; Grafana transforms metrics into meaningful visualizations. Both are compatible with many, if not most, data source types.

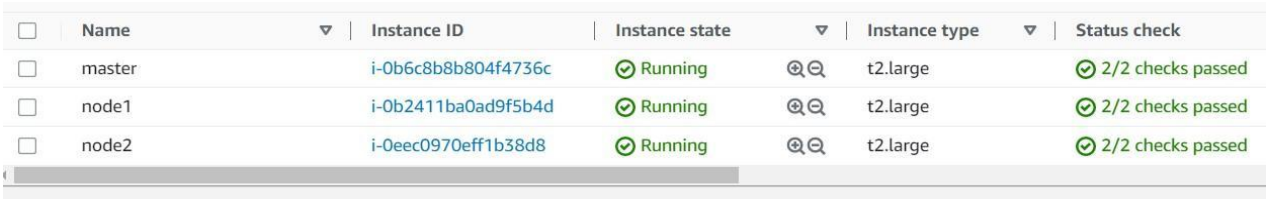
Prometheus is an open-source data monitoring tool. The combination of Prometheus and Grafana is the de-facto combination leveraged in the industry for deploying a data visualization setup. Grafana dashboard is used for visualizing the data whereas the backend is powered by Prometheus.

How to install Prometheus and Grafana in Amazon linux 2

Here First we will launch the three AWS ec2-instances and we will create Kubernetes cluster and then in the next step will install Prometheus and Grafana to monitor the ec2 machines.

Step1 : Launch 3 Amazon Linux 2 ec2 machines(Master, node1 and node2) with t2.large Instance Type

Lanch the required ec2-instances with required configurations



Name	Instance ID	Instance state	Instance type	Status check
master	i-0b6c8b8b804f4736c	Running	t2.large	2/2 checks passed
node1	i-0b2411ba0ad9f5b4d	Running	t2.large	2/2 checks passed
node2	i-0eec0970eff1b38d8	Running	t2.large	2/2 checks passed

Step2 : Installing Docker and Kubernetes on master

Connect to the master machine shell execute the following commands

- Switch to root user and create a file called master.sh with docker and Kubernetes installation commands in it.
- Create a new file with a name of master.sh

```
vi master.sh
```

- Paste following content in the file

```
hostnamectl set-hostname K8s-Master
yum update -y
yum install git -y
amazon-linux-extras install java-openjdk11 -y
yum install docker -y
systemctl enable docker
systemctl restart docker
systemctl status docker
free -mh
sestatus

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
```

```
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable kubelet
systemctl restart kubelet
kubeadm init
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
export KUBECONFIG=/etc/kubernetes/admin.conf
kubeadm token create --print-join-command
```

- Give execution permission to master.sh file

```
chmod +x master.sh
```

- Run the above master.sh file

```
./master.sh
```

- After running master.sh file you will get kubeadm join command copy and keep, we will use that join command to join our nodes to master.

```
Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.19.170:6443 --token 8n27fg.9xa3es5gh1ebfo2p \
        --discovery-token-ca-cert-hash sha256:9f43111525ee0c5a65c141dfc77b29b42e4cf7c0259963d885a5953c53e65743
kubeadm join 172.31.19.170:6443 --token 004niy.1nhnedwcw6apeih2 --discovery-token-ca-cert-hash sha256:9f43111525ee0c5a65c141dfc77b29b42e
[root@ip-172-31-19-170 ~]#
```

Check git version

```
git –version
```

Check java version

```
java --version
```

```
[root@ip-172-31-19-170 ~]# git --version
git version 2.39.2
[root@ip-172-31-19-170 ~]# java --version
openjdk 11.0.18 2023-01-17 LTS
OpenJDK Runtime Environment (Red_Hat-11.0.18.0.10-1.amzn2.0.1) (build 11.0.18+10-LTS)
OpenJDK 64-Bit Server VM (Red_Hat-11.0.18.0.10-1.amzn2.0.1) (build 11.0.18+10-LTS, mixed mode, sharing)
[root@ip-172-31-19-170 ~]#
```

Step 3 : Adding node1 and node2 to Kubernetes master

- Connect to node1 ec2 machine and Switch to root user

```
sudo -i
```

- Create a file called node1.sh and put following commands in it. And give execute permission to node1.sh file.
- Create a file name with node1.sh

```
vi node1.sh
```

- Insert the following commands in it.

```
hostnamectl set-hostname node1
yum update -y
amazon-linux-extras install java-openjdk11 -y
yum install git -y
yum install docker -y
systemctl enable docker
systemctl restart docker
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable kubelet
systemctl restart kubelet
```

- Your file will look like this. Give execute permission to this file and run it.

```
chmod +x node1.sh
```

```
hostnamectl set-hostname node1
yum update -y
amazon-linux-extras install java-openjdk11 -y
yum install docker -y
systemctl enable docker
systemctl restart docker

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable kubelet
systemctl restart kubelet


~
~
~
-- INSERT --
```

- Execute the following command

```
./node1.sh
```

Step4: Add node2 to Kubernetes Cluster by following same steps.

- Open node2 ec2 machine and Switch to root user

```
sudo -i
```

- Create a file called node2.sh and put following commands in it. And give execute permission to node2.sh file.

```
vi node2.sh
```

- Put these commands in the file and run it.

```
hostnamectl set-hostname node2
yum update -y
amazon-linux-extras install java-openjdk11 -y
yum install git -y
yum install docker -y
systemctl enable docker
systemctl restart docker
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearchenabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable kubelet
systemctl restart kubelet
```

- Now run that Kubeadm join command on node1 and node2. **(Execute the join command only on node1 and node 2)**
- After running join command on node1 you will get this.

```
Complete!
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/kubelet.service.
[root@ip-172-31-21-121 ~]# kubeadm join 172.31.19.170:6443 --token 46ho5s.812fk37vdcfo25kk --discovery-token-ca-cert-hash sha2
963d885a5953c53e65743
[preflight] Running pre-flight checks
        [WARNING FileExisting-tc]: tc not found in system path
        [WARNING Hostname]: hostname "node1" could not be reached
        [WARNING Hostname]: hostname "node1": lookup node1 on 172.31.0.2:53: no such host
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

[root@ip-172-31-21-121 ~]#
```

i-0b2411ba0ad9f5b4d (node1)

PublicIPs: 3.15.216.32 PrivateIPs: 172.31.21.121

- After running join command on node2 you will get this.

```
Complete!
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/k
[root@ip-172-31-22-35 ~]# kubeadm join 172.31.19.170:6443 --token 46ho5s.812fk37vdcfo25kk --discovery-token-c
63d885a5953c53e65743
[preflight] Running pre-flight checks
        [WARNING FileExisting-tc]: tc not found in system path
        [WARNING Hostname]: hostname "node2" could not be reached
        [WARNING Hostname]: hostname "node2": lookup node2 on 172.31.0.2:53: no such host
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

[root@ip-172-31-22-35 ~]#
```

i-0eec0970eff1b38d8 (node2)

PublicIPs: 18.116.73.25 PrivateIPs: 172.31.22.35

- Execute the command to get the nodes but it shows nodes are **NotReady**

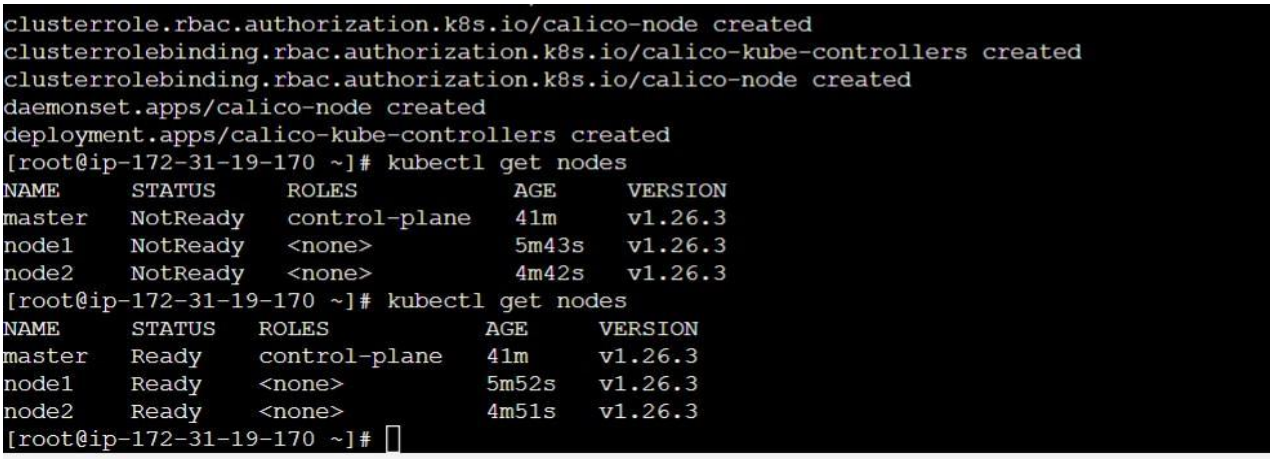
```
kubectl get nodes
```

- Run the following commands on master node only. By adding this CNI the all the nodes get ready in our cluster

```
curl https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml -O

kubectl apply -f calico.yaml
```

- If all above steps followed correctly the cluster will be created and status will be changed to ready.

```
kubectl get nodes
```



Step5 : Now lets clone our project and deploy it on Kubernetes cluster.

- On Master node clone the git repository using the command
- Create a project directory and run git clone command inside it

```
git clone https://github.com/ramdassbhanage/fitpro-k8s-project-new-git2.git
```

- After cloning you will get ready microservices yaml files. You just need to run them.

- Run the following command to deploy all the microservices to Kubernetes cluster

```
kubectl apply -f .
```

- After running apply command project is deployed

```
[root@ip-172-31-19-170 fitpro-k8s-project-new-git2]# kubectl apply -f .
deployment.apps/api-gateway created
service/apigatewaysvc created
deployment.apps/appointmentservice created
service/appointmentservicesvc created
deployment.apps/authenticationservice created
service/authenticationservicesvc created
deployment.apps/configserver created
service/configserversvc created
deployment.apps/emailservice created
service/emailservicesvc created
statefulset.apps/eurekaserver created
service/eurekaserversvc created
statefulset.apps/mongodbservice created
service/mongodbservicesvc created
statefulset.apps/mysqlservice created
service/mysqlservicesvc created
deployment.apps/paymentservice created
service/paymentservicesvc created
deployment.apps/product-webapp created
service/productwebappsvc created
statefulset.apps/rabbitmq created
service/rabbitmqsvc created
deployment.apps/userservice created
service/userservicesvc created
[root@ip-172-31-19-170 fitpro-k8s-project-new-git2]#
```

- Run the kubectl get pods command to list all pods

```
kubectl get pods
```

- Run the kubectl get all command to list all the pods and services

```
kubectl get all
```

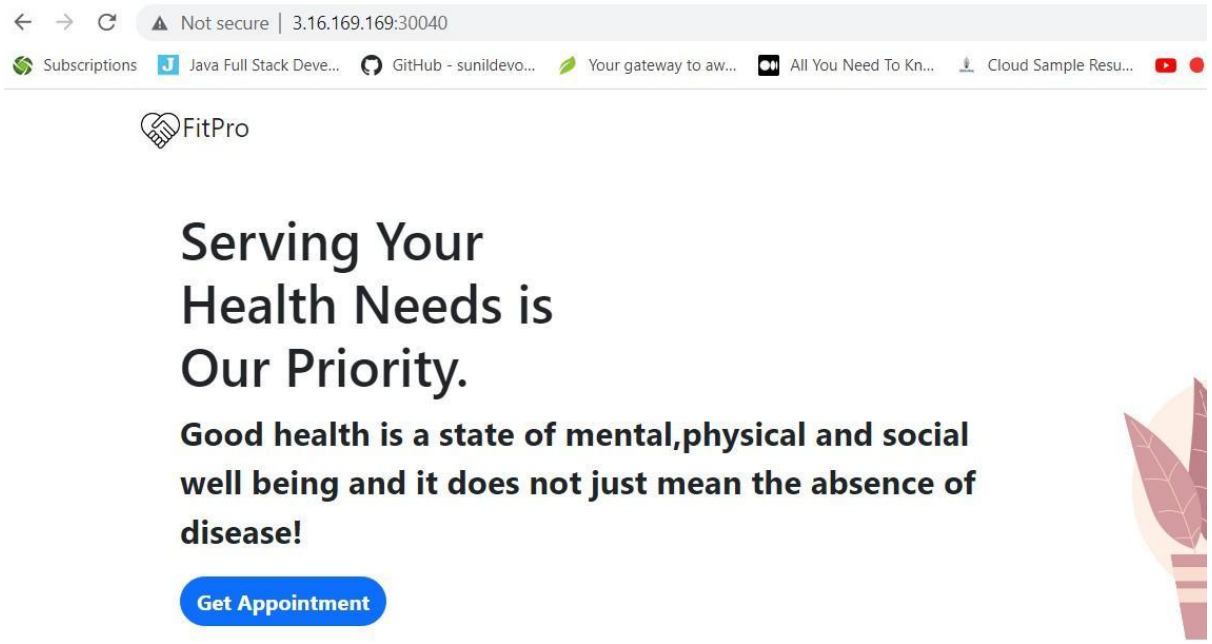
- Now run kubecl get svc command to list all ports information

```
kubectl get svc
```

- Copy 5 digit port of apigateway service and paste it in the browser with <public ip> of your master machine.

  Example : (http://3.16.169.169:30040/)

- You will get this output in the browser



Installing Prometheus and Grafana

Steps to install Prometheus

Step 1: Run these commands on master machine to install Prometheus

- Add the repository

```
sudo tee /etc/yum.repos.d/prometheus.repo <<EOF
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rp
m
gpgcheck=1
metadata_expire=300
EOF
```

- We will the get the output like this after adding the repo

```
[root@ip-172-31-19-170 ~]# sudo tee /etc/yum.repos.d/prometheus.repo <<EOF
> [prometheus]
> name=prometheus
> baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
> repo_gpgcheck=1
> enabled=1
> gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
>        https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
> gpgcheck=1
> metadata_expire=300
> EOF
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
       https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
gpgcheck=1
metadata_expire=300
[root@ip-172-31-19-170 ~]#
```

Run these commands

- Install Prometheus service

```
sudo yum -y install prometheus2
```

- Enable the Prometheus service

```
systemctl enable prometheus
```

- Start the Prometheus service

```
systemctl restart Prometheus
```

- Check the Prometheus service

```
systemctl status prometheus
```

- You can use rpm command to check for the version of Prometheus installed

```
rpm -qi prometheus2
```

- Execute the command to edit the file

```
vim /etc/prometheus/prometheus.yml
```

write this all in the file
(paste the below lines in the file)

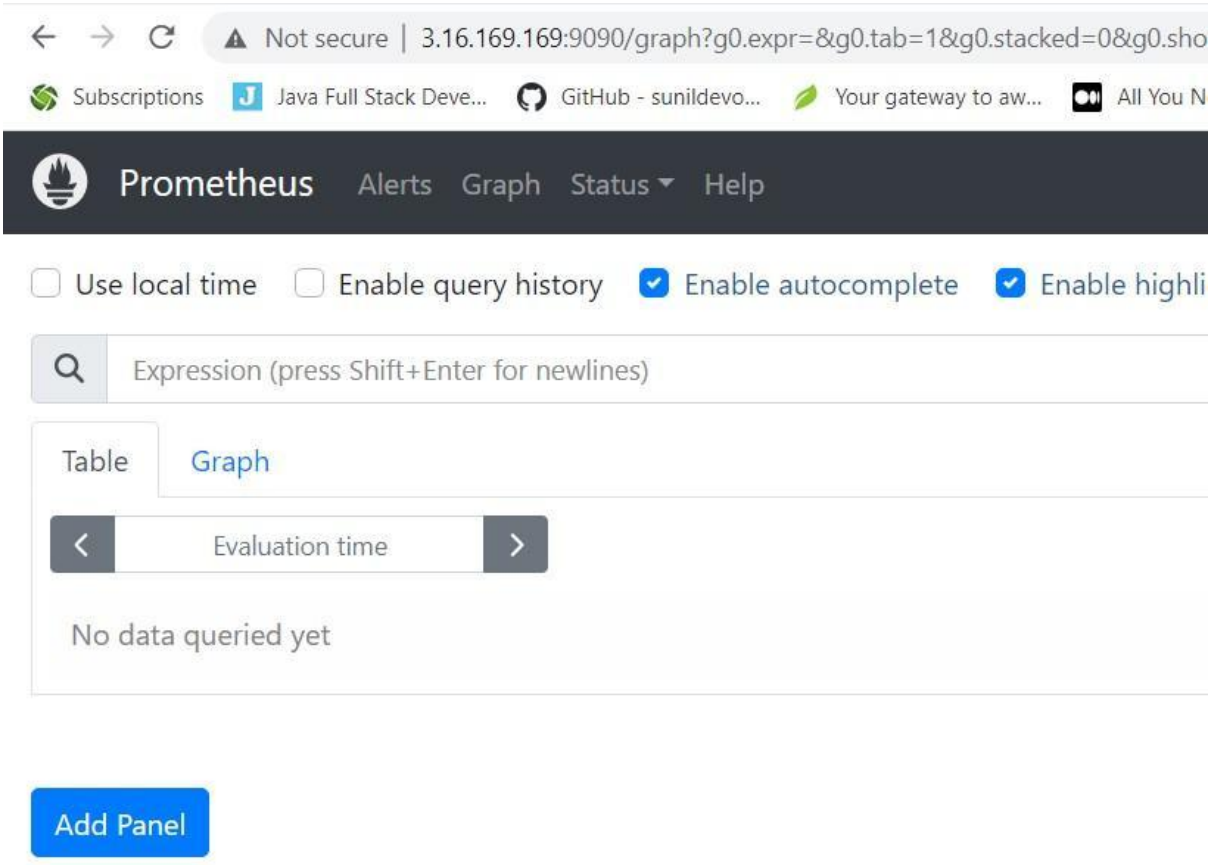
```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'K8s-master'

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

    static_configs:
    - targets: ['localhost:9090']

  # Pull host metrics with node exporter
  - job_name: ‘Worker-node-1’
    static_configs:
      - targets: ['<Replace with node1 public IP>:9100']
  - job_name: ‘Worker-node-1’
    static_configs:
      - targets: ['<Replace with node2 public IP>:9100']
```

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeserie
  - job_name: "Kubernetes Master"

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


    static_configs:
    - targets: ['localhost:9090']
  # Pull host metrics with node exporter
  - job_name: 'node1'
    static_configs:
    - targets: ['3.15.216.32:9100']
  - job_name: 'node2'
    static_configs:
    - targets: ['18.116.73.25:9100']

-- INSERT --
```

- Put public Ip address of node1 and node2 in local host
- After editing Prometheus.yml file restart it.

```
systemctl restart Prometheus
```

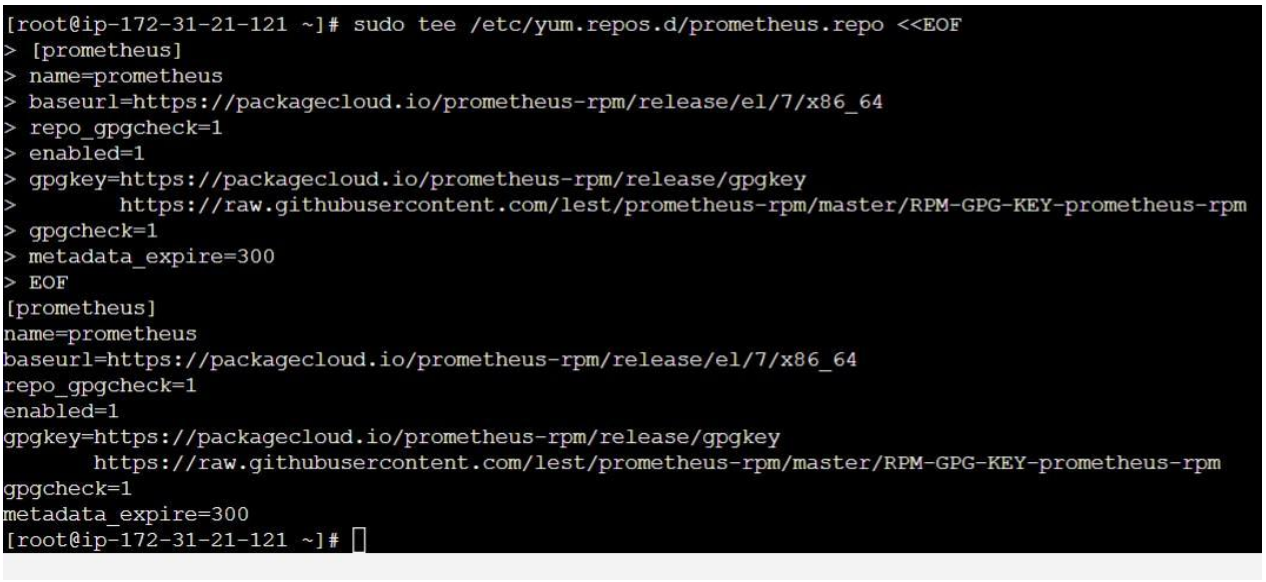
- Copy master node Public IP : 9090 and paste it in browser.



Step 2 : Install node exporter on node1 and node2

- Connect to node1 ec2 machine
- Add the repository

```
sudo tee /etc/yum.repos.d/prometheus.repo <<EOF
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey

https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
gpgcheck=1
metadata_expire=300
EOF
```



- Run following commands
- Install the node exporter

```
yum -y install node_exporter
```

- Enable the node exporter service

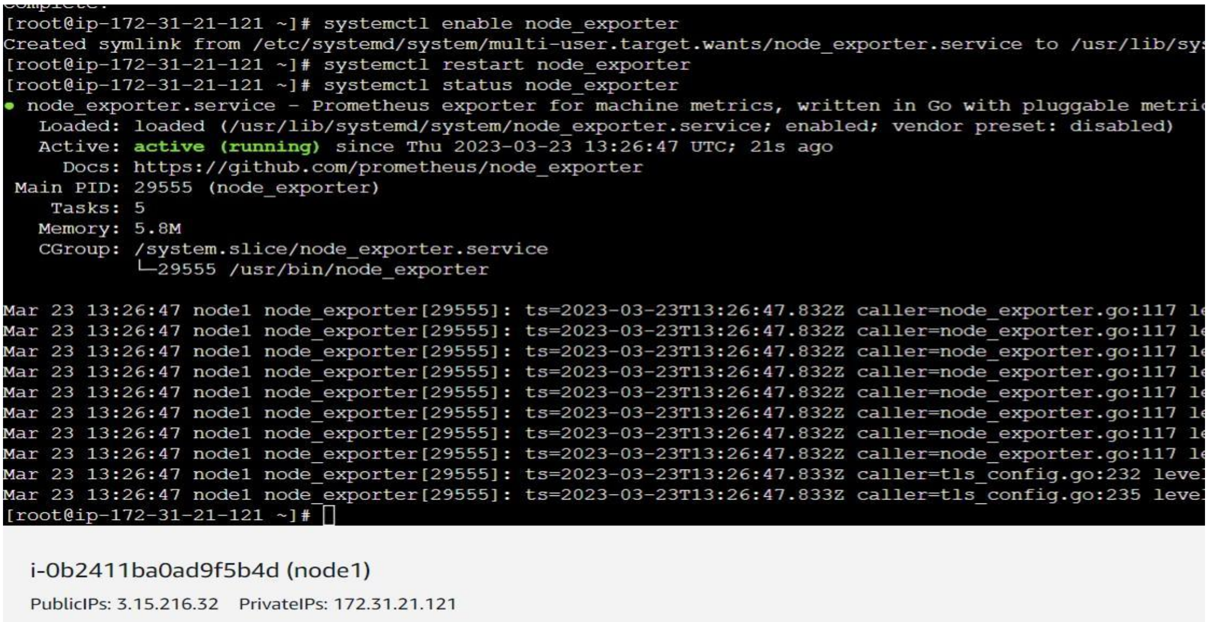
```
systemctl enable node_exporter
```

- Start the node exporter service

```
systemctl restart node_exporter
```

- Check the status of node exporter service

```
systemctl status node_exporter
```

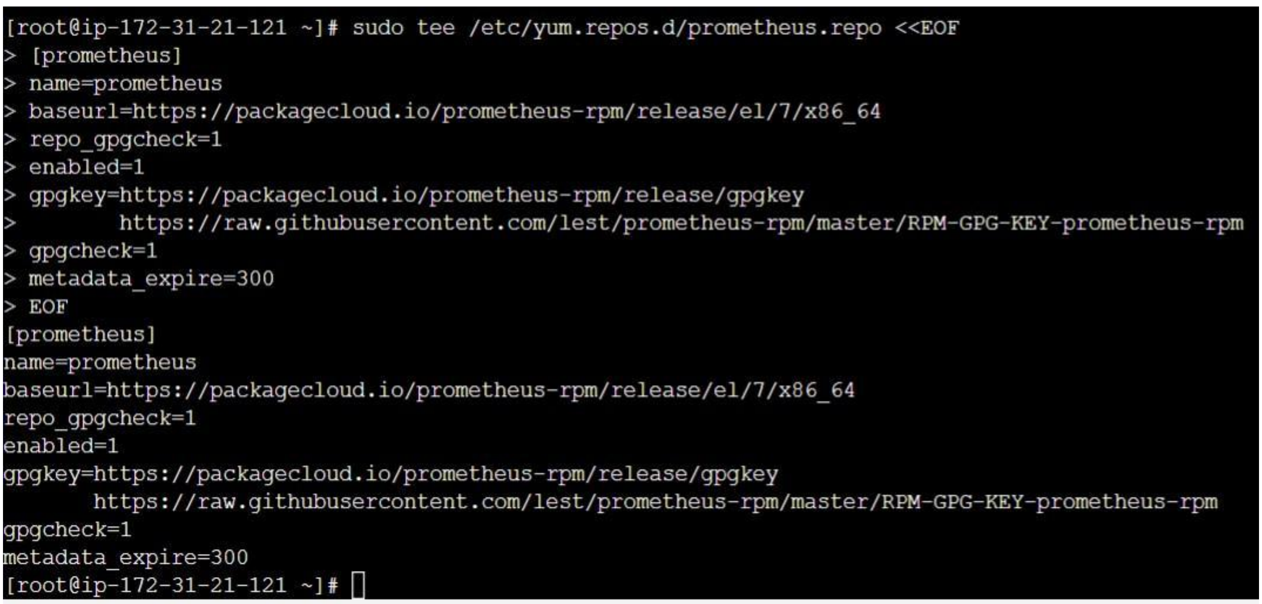


```
[root@ip-172-31-21-121 ~]# systemctl enable node_exporter
Created symlink from /etc/systemd/system/multi-user.target.wants/node_exporter.service to /usr/lib/sy
[root@ip-172-31-21-121 ~]# systemctl restart node_exporter
[root@ip-172-31-21-121 ~]# systemctl status node_exporter
● node_exporter.service - Prometheus exporter for machine metrics, written in Go with pluggable metri
   Loaded: loaded (/usr/lib/systemd/system/node_exporter.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2023-03-23 13:26:47 UTC; 21s ago
     Docs: https://github.com/prometheus/node_exporter
 Main PID: 29555 (node_exporter)
    Tasks: 5
   Memory: 5.8M
   CGroup: /system.slice/node_exporter.service
           └─29555 /usr/bin/node_exporter

Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.832Z caller=node_exporter.go:117 le
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.833Z caller=tls_config.go:232 level
Mar 23 13:26:47 node1 node_exporter[29555]: ts=2023-03-23T13:26:47.833Z caller=tls_config.go:235 level
[root@ip-172-31-21-121 ~]#
```

i-0b2411ba0ad9f5b4d (node1)
PublicIPs: 3.15.216.32   PrivateIPs: 172.31.21.121

- Connect to node2 ec2 machine
- Add the repository

```
sudo tee /etc/yum.repos.d/prometheus.repo <<EOF
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey

https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
gpgcheck=1
metadata_expire=300
EOF
```



```
[root@ip-172-31-21-121 ~]# sudo tee /etc/yum.repos.d/prometheus.repo <<EOF
> [prometheus]
> name=prometheus
> baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
> repo_gpgcheck=1
> enabled=1
> gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
>        https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
> gpgcheck=1
> metadata_expire=300
> EOF
[prometheus]
name=prometheus
baseurl=https://packagecloud.io/prometheus-rpm/release/el/7/x86_64
repo_gpgcheck=1
enabled=1
gpgkey=https://packagecloud.io/prometheus-rpm/release/gpgkey
       https://raw.githubusercontent.com/lest/prometheus-rpm/master/RPM-GPG-KEY-prometheus-rpm
gpgcheck=1
metadata_expire=300
[root@ip-172-31-21-121 ~]#
```

- Run following commands
- Install the node exporter

```
yum -y install node_exporter
```

- Enable the node exporter service

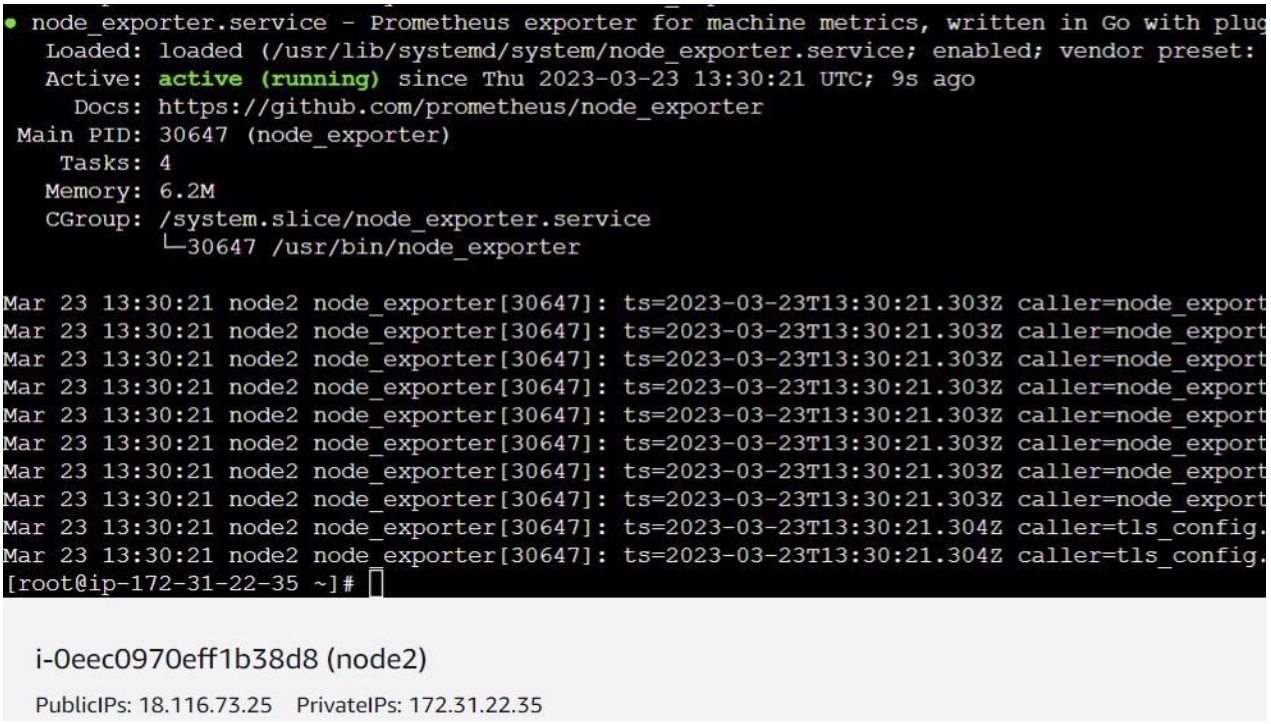
```
systemctl enable node_exporter
```

- Start the node exporter service

```
systemctl restart node_exporter
```

- Check the status of node exporter service

```
systemctl status node_exporter
```

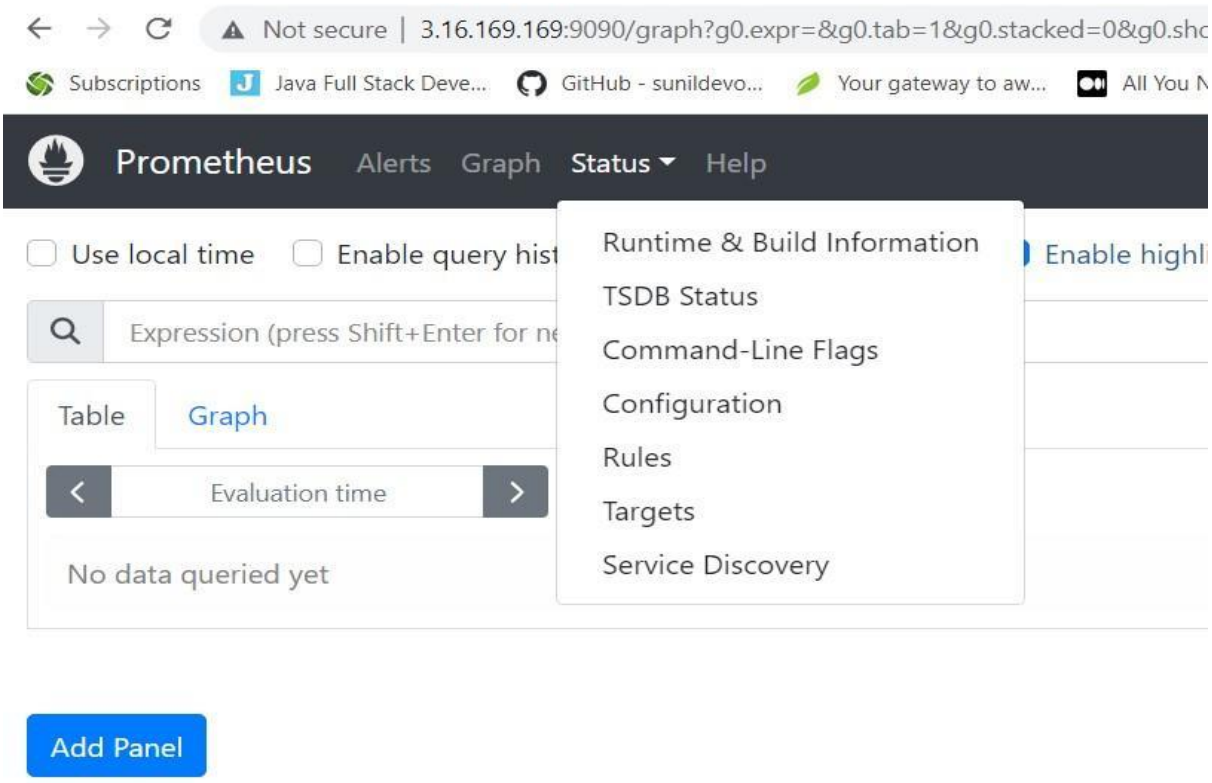


- Copy Node1 Public Ip and open it with 9100 port

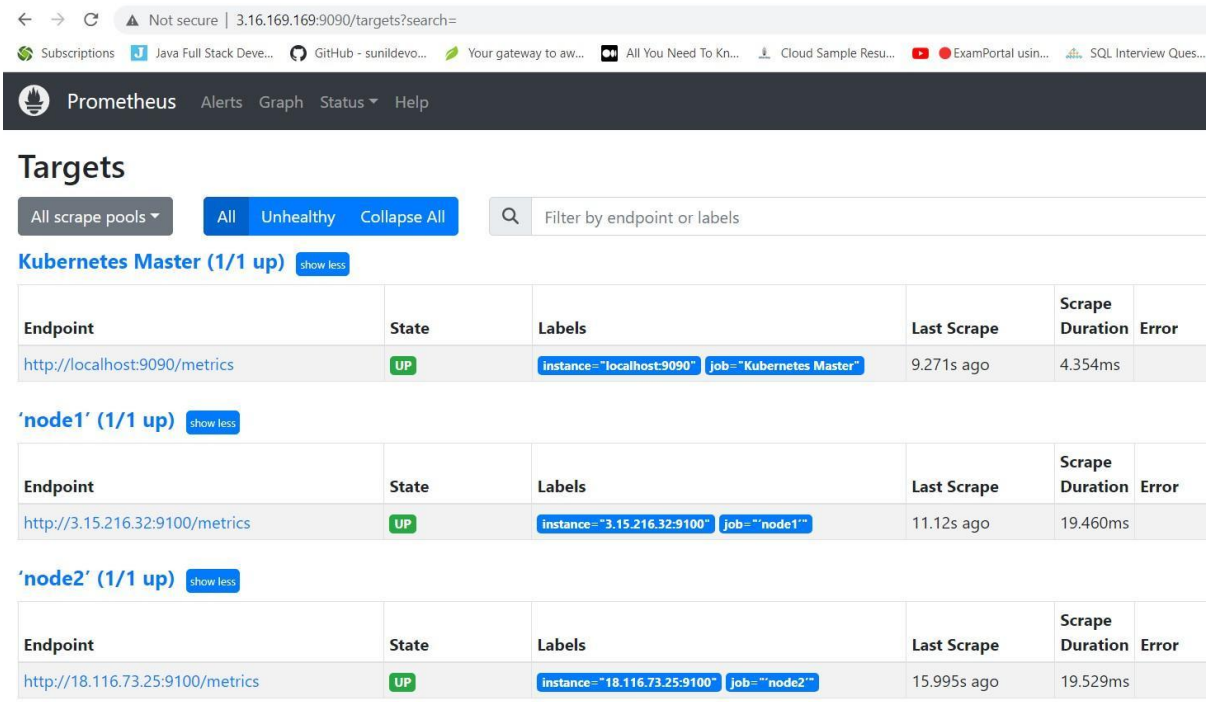
```
Example: http://3.15.216.32:9100/
```

- You will get this in browser

- Now copy master node puplic ip and open it with 9090 port and goto to targets



- Click on targets. It will display you node1 and node2 machine



Step 4: Installing Grafana on Master machine

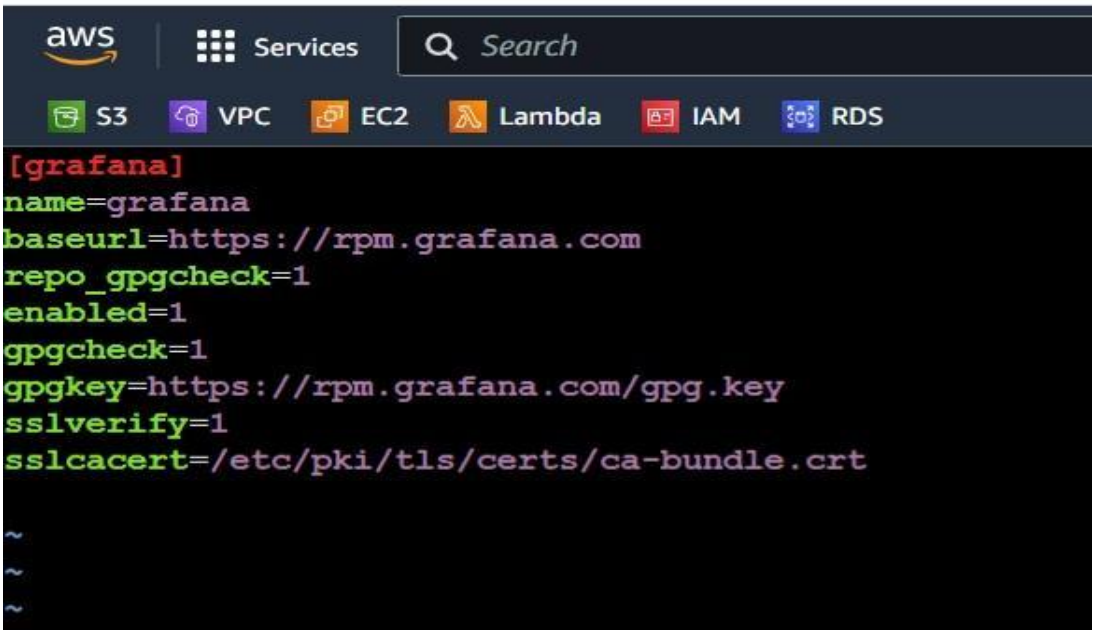
- Create **grafana.repo** file to install grafana in amazon linux 2

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

- Add following content in **grafana.repo** file

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

- After adding the data file look like this



Run following commands to install grafana

- Install the Grafana service on master machine

```
sudo yum install grafana -y
```

- Start the Grafana service

```
sudo systemctl start grafana-server
```

- Enable the Grafana service

```
sudo systemctl enable grafana-server
```

- Check the Status Grafana service

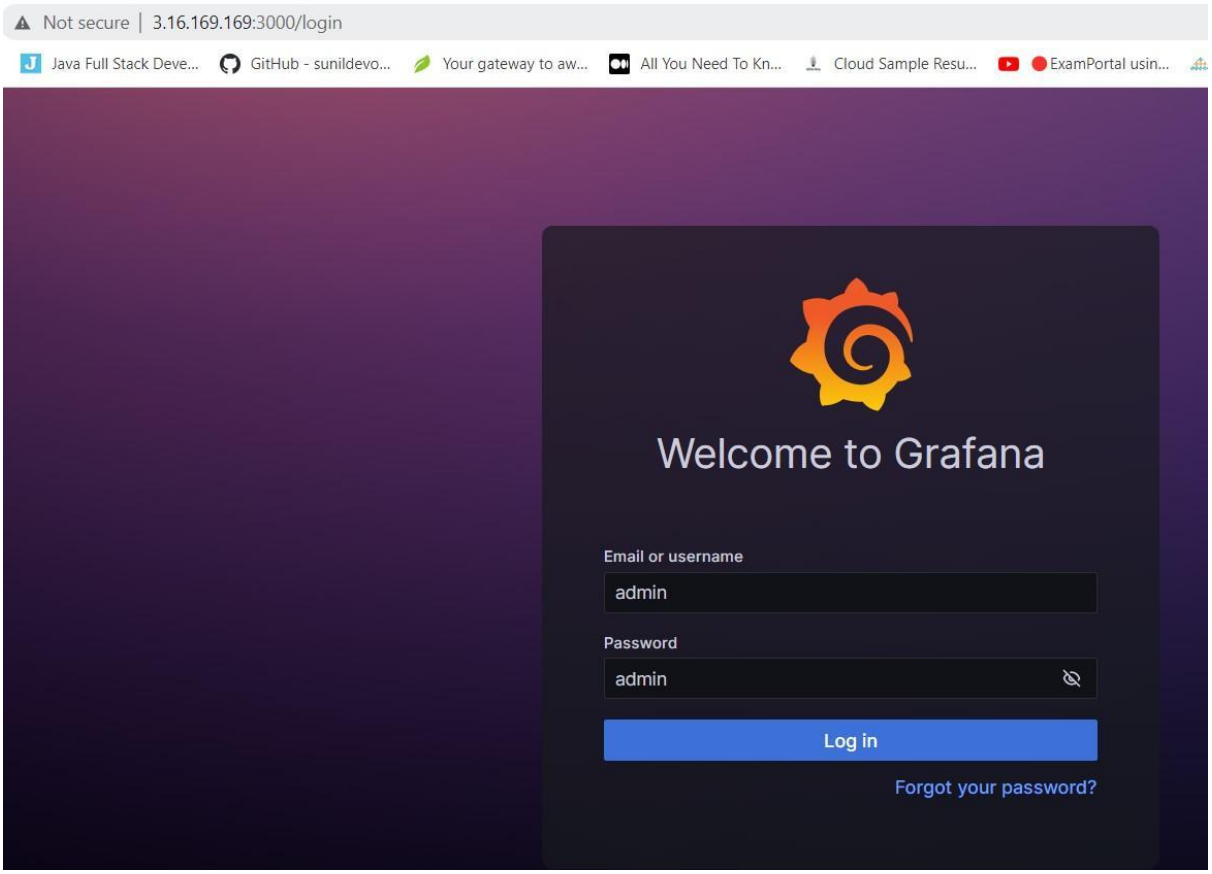
```
sudo systemctl status grafana-server
```

```
[root@ip-172-31-19-170 ~]# sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2023-03-23 13:44:29 UTC; 18s ago
     Docs: http://docs.grafana.org
 Main PID: 20435 (grafana)
   CGroup: /system.slice/grafana-server.service
           └─20435 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile=/v

Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:29] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:31] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:32] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:33] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:34] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:35] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:37] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:39] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:40] Unknown lvalue '
Mar 23 13:44:38 master systemd[1]: [/usr/lib/systemd/system/grafana-server.service:41] Unknown lvalue '
[root@ip-172-31-19-170 ~]#
```

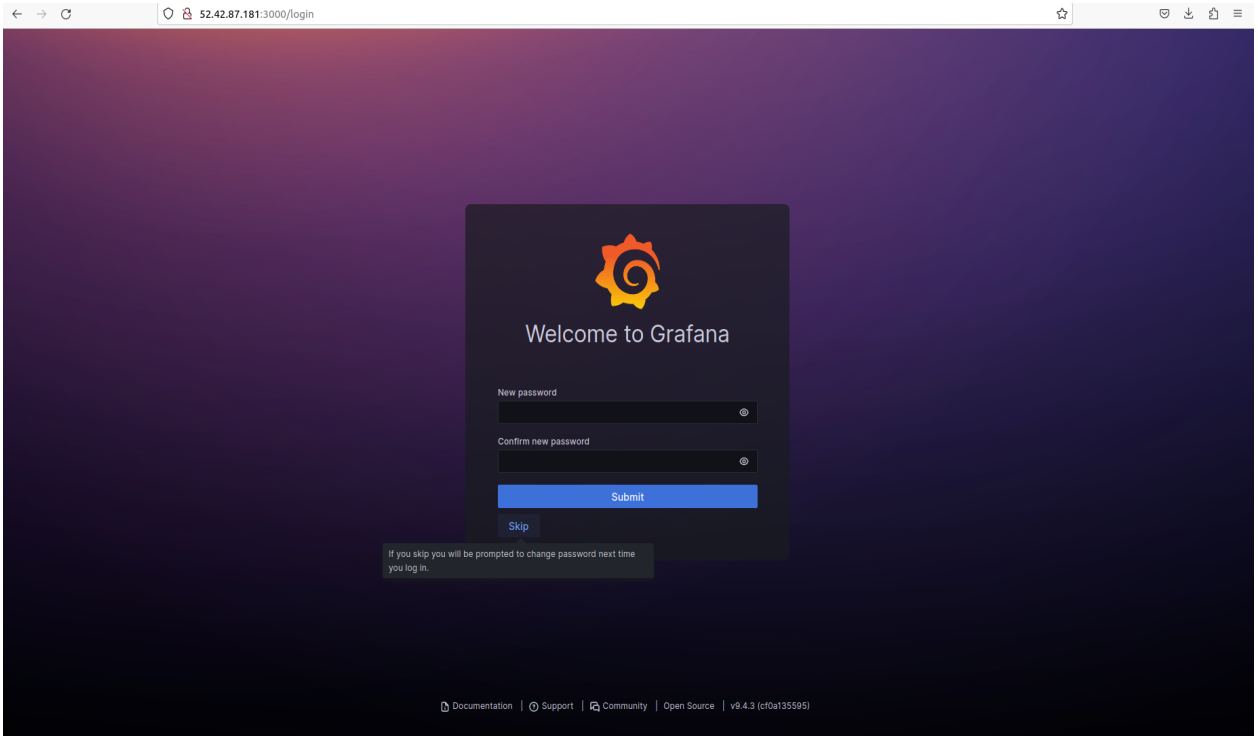
- Goto browser copy paste public IP of Master and 3000 port

Example: http://3.16.169.169:3000/

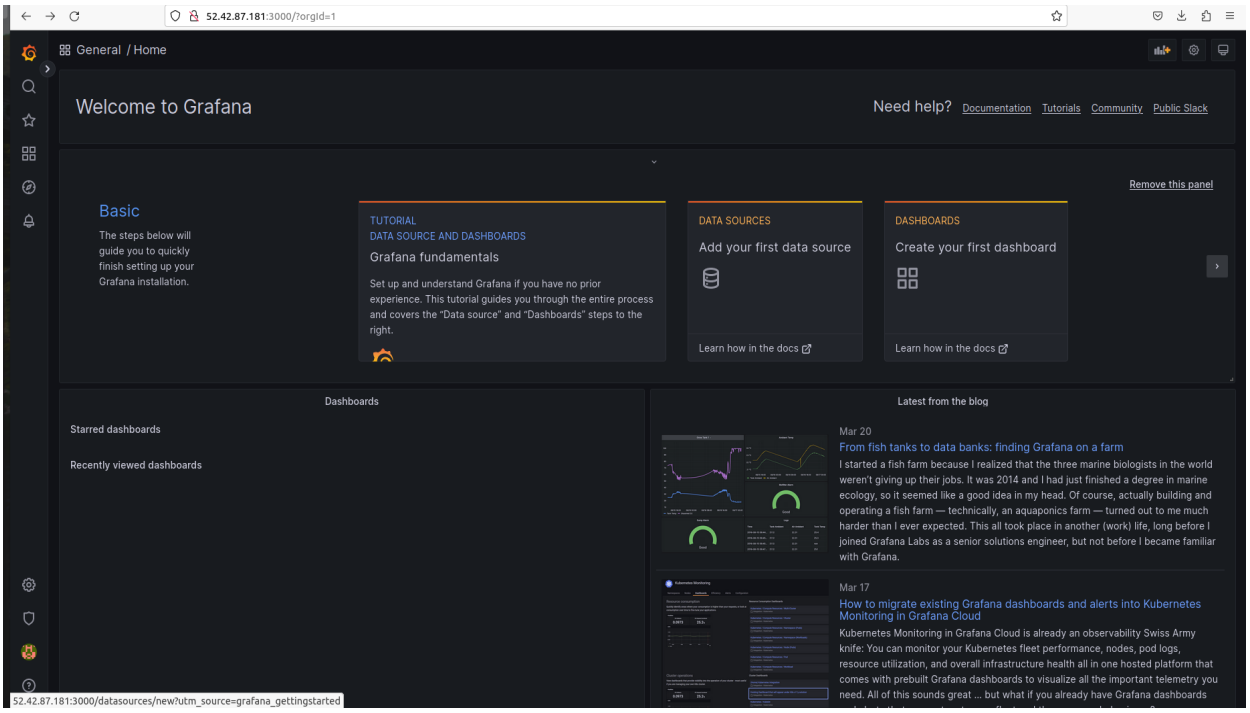
- You will see Grafana dashboard use admin and admin for username and password



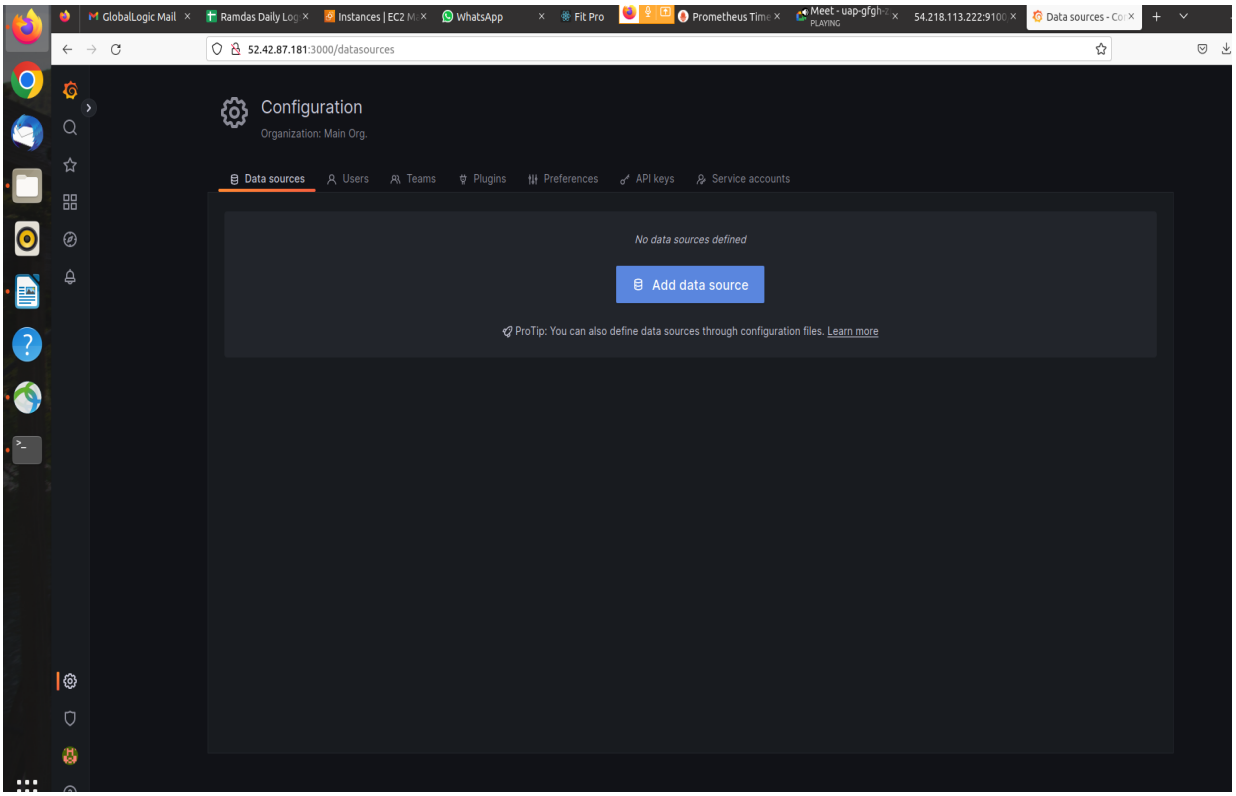
- Click skip if it ask to change password



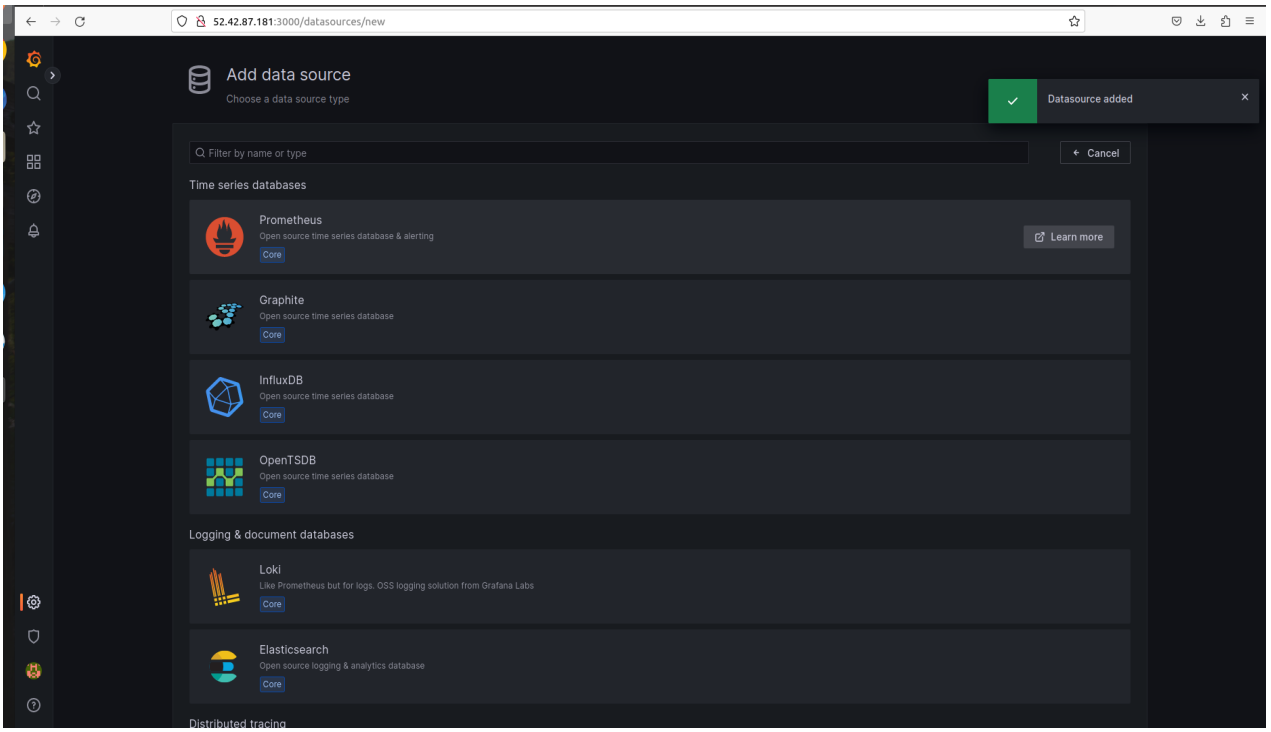
- You will see Grafana dashboard & Data Souces



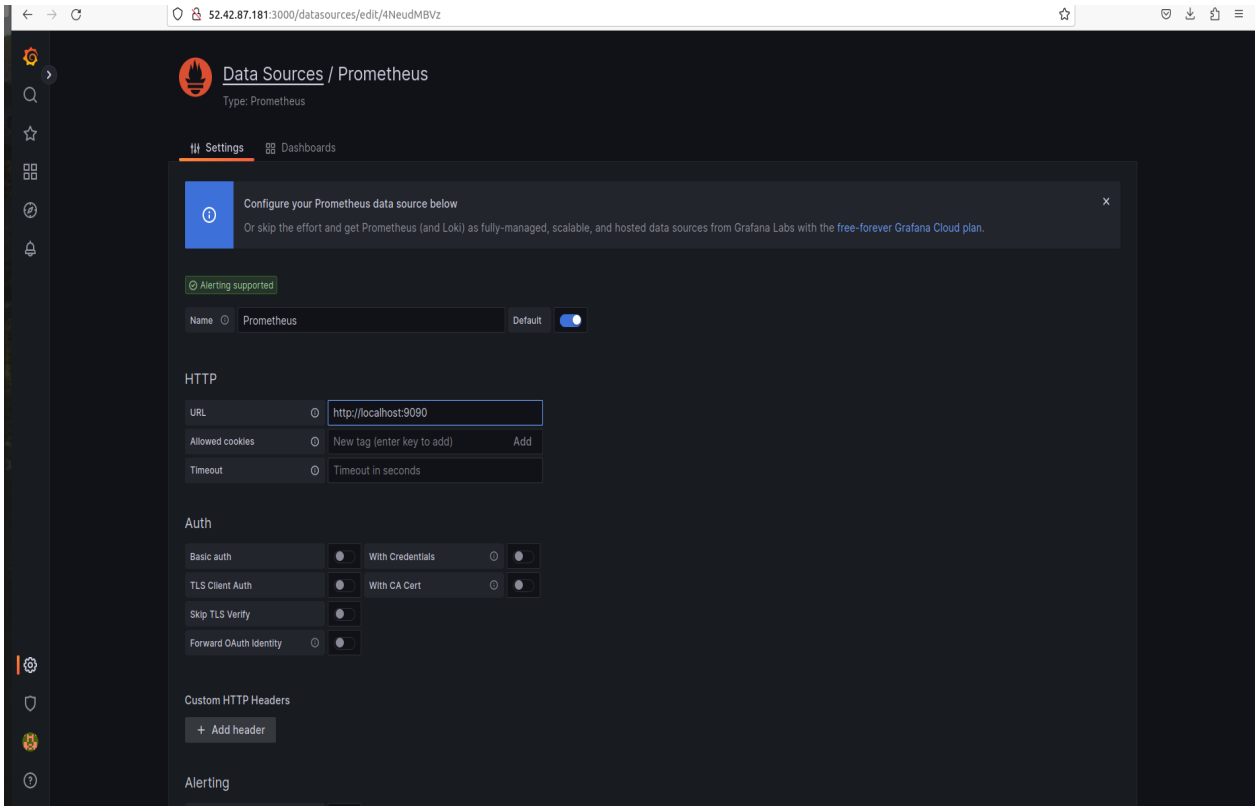
- You will see Add Data Source



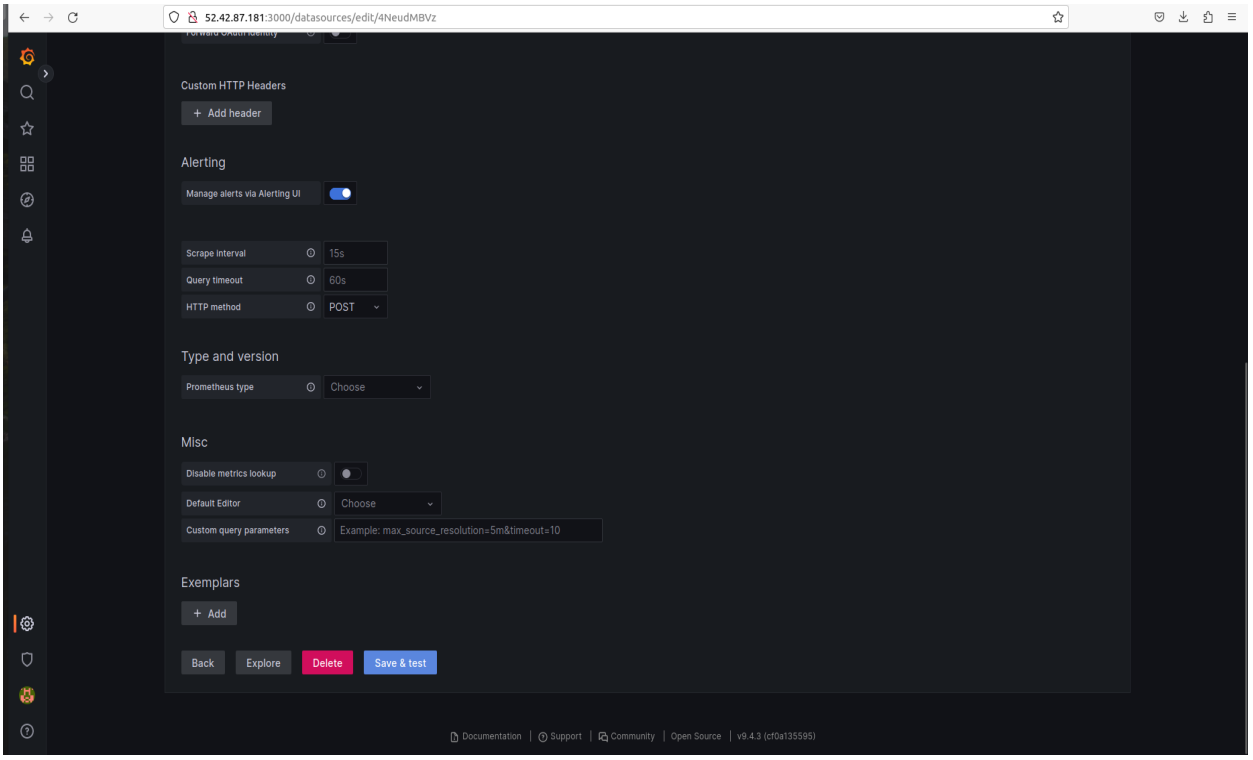
- You can select Prometheus Data Source and Click it.



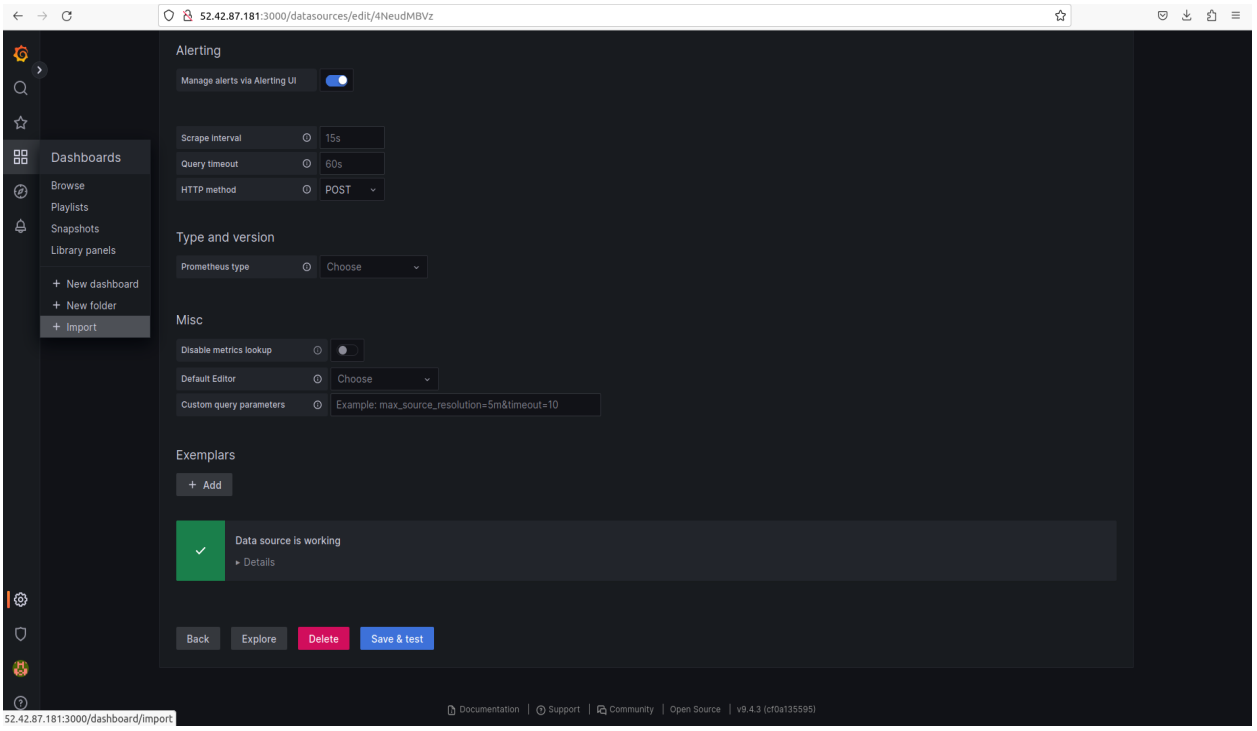
- after change Configuration Add url i.e.http://localhost:9090



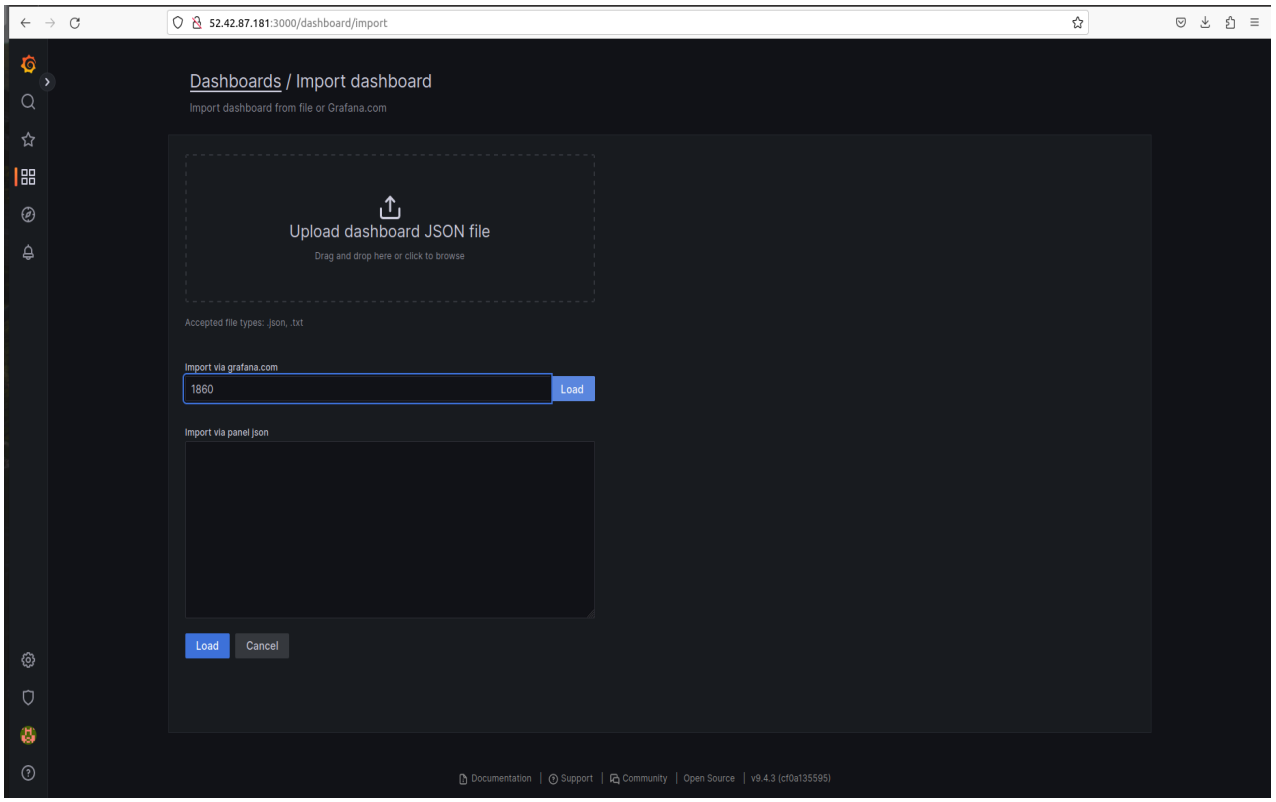
- Then click save & Test.



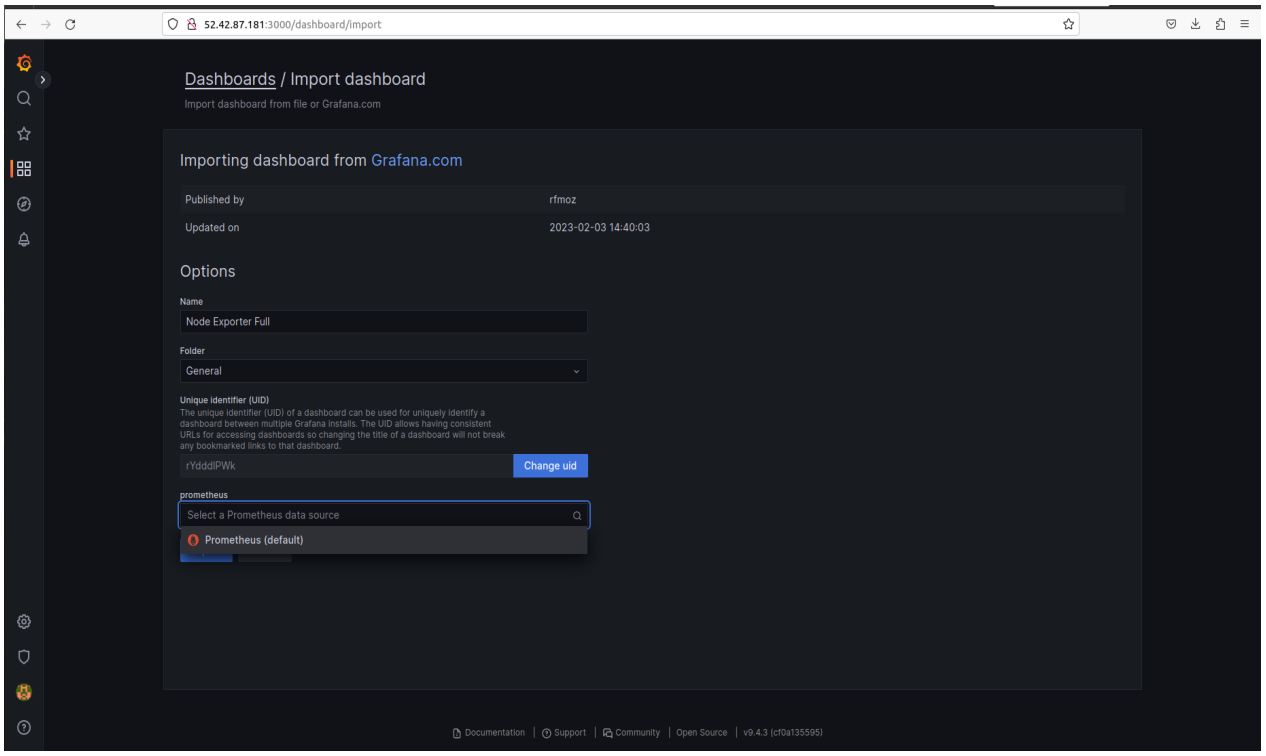
- You will see DialogBox DataSource Added and its working.



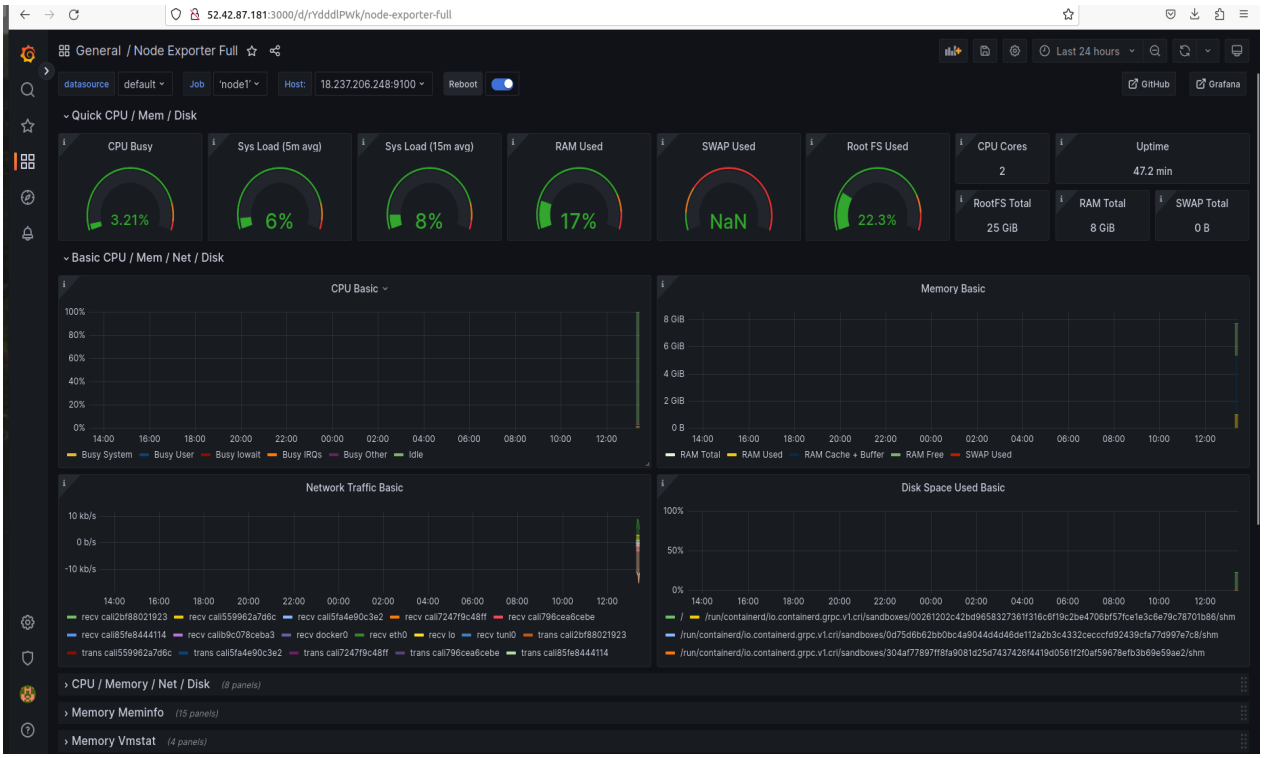
- you will see Dashobards /import dashboard add Grafana ID to import Prometheus 1860 click Load.



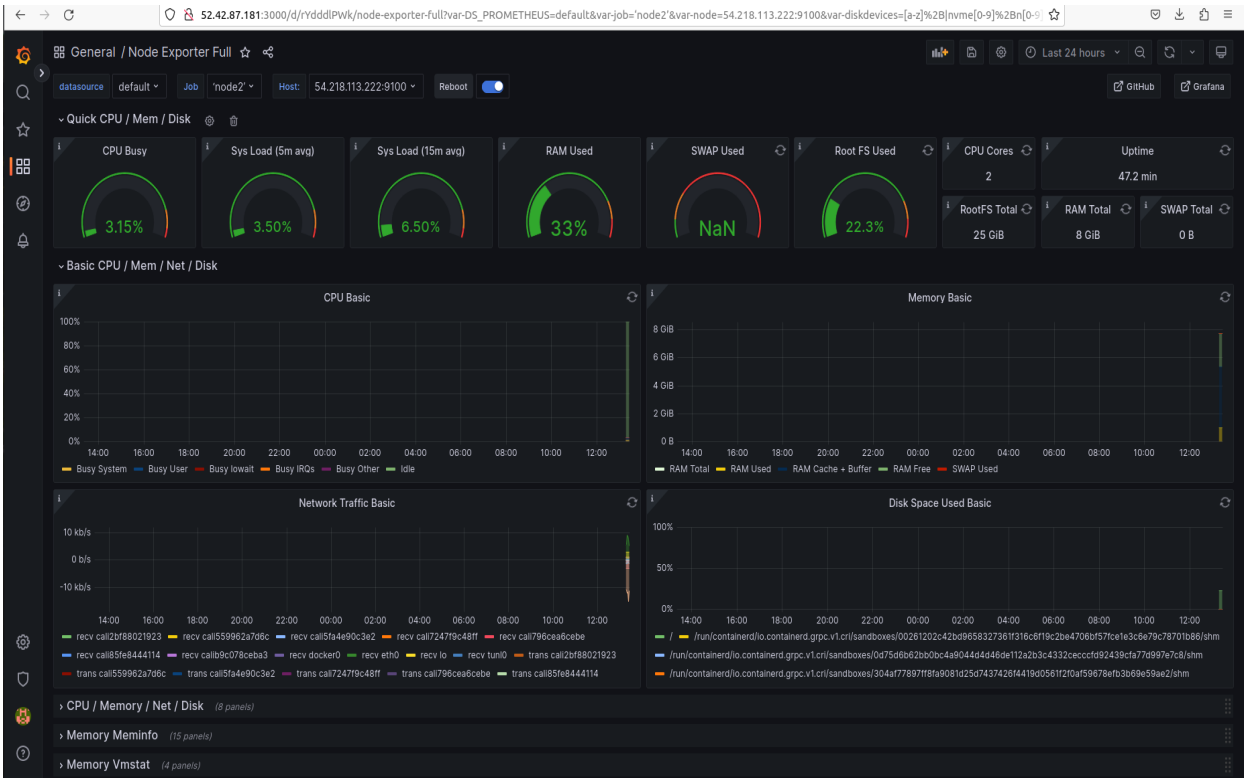
- you will see Dashboards /import dashboard select Data Source.



- you will see Node1 Dashboard Output.



- You will see Node2 Dashboard Output.



- You can see node1 and node2 Dashboard Output for Time series last 5 min.