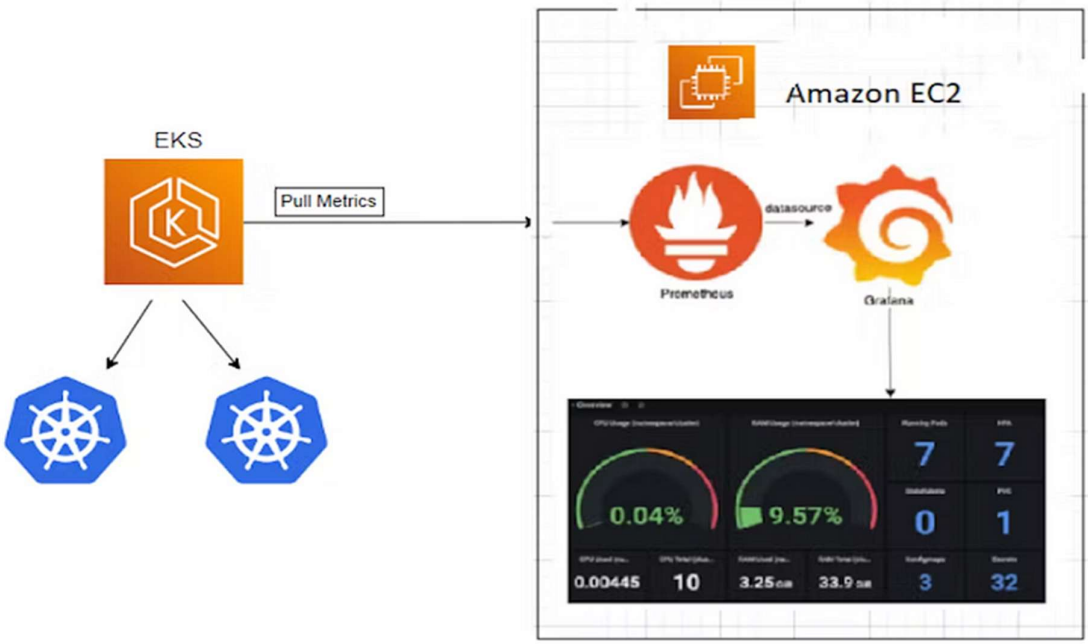
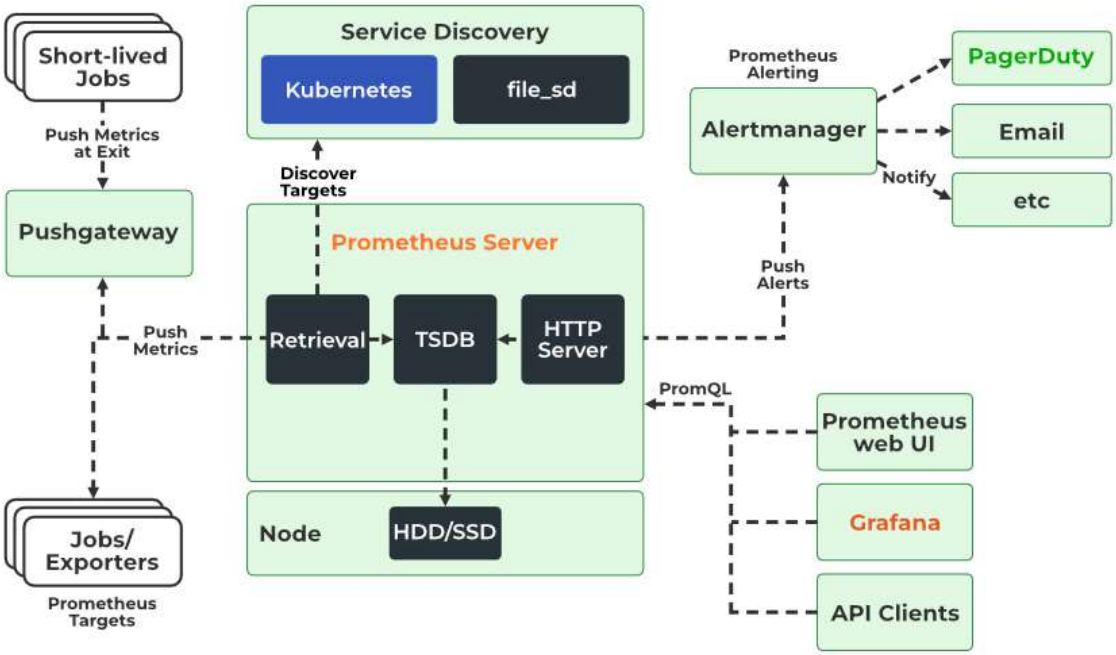
🌻 EKS monitoring using Helm, Prometheus and Grafana Dashboard 🌻





This blog explains how to set up Prometheus and Grafana in Amazon EKS.

1. What is Prometheus?

- Prometheus is an open-source monitoring tool
- Provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform. It can monitor servers and databases as well.
- Collects and stores metrics as time-series data, recording information with a timestamp
- It is based on pull and collects metrics from targets by scraping metrics HTTP endpoints.

2. What is Grafana?

- Grafana is an open-source visualization and analytics software.
- It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored.

3. Why use helm?

- Helm is a package manager for Kubernetes. Helm simplifies the installation of all components in one command. install using helm is recommended as you will not be missing any configuration steps and very efficient

Why do we need System Monitoring?

- To detect and prevent failures, it is very convenient to have a good monitoring tool that provides you with a solid monitoring system.
- Monitoring systems are responsible for controlling the technology used by companies in order to analyze their operation and performance.
- Understanding the state of your infrastructure and systems is essential for ensuring the reliability and stability of your services.
- Information about the health and performance of your deployments not only helps your team react to issues but also gives them the security to make changes with confidence.

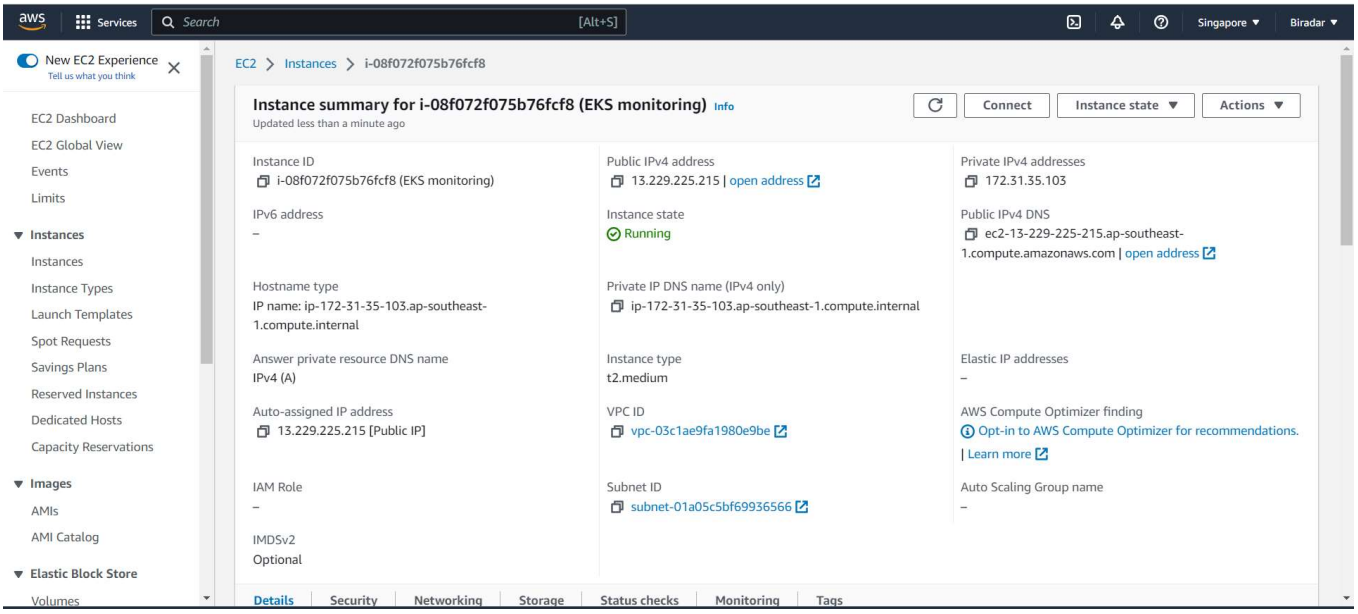
Steps:-

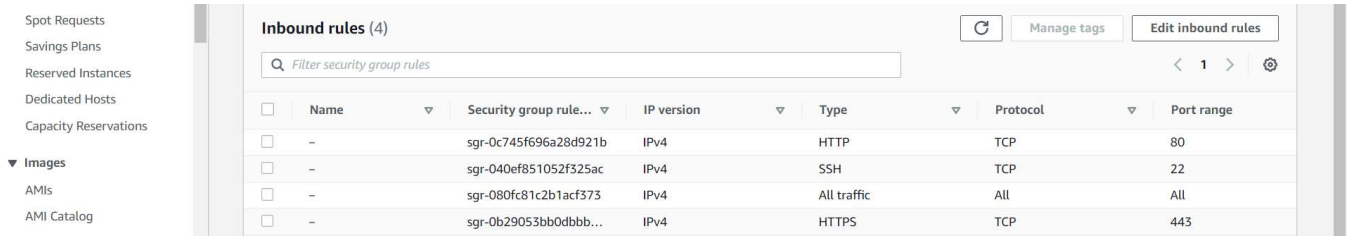
1. Set up an AWS EC2 Instance and
2. Install AWS CLI and Configure AWS Credentials
3. Install and setup kubectl
4. Install and setup eksctl
5. Install Helm Charts
6. Creating an Amazon EKS cluster using eksctl
7. We need to add the Helm Stable Charts for your local client
8. Add Prometheus Helm repo
9. Create Prometheus namespace
10. Install Prometheus
11. Install Node JS application and monitor it on Grafana
12. Cleanup/Deprovision

Now, lets get started in detail

Step 1 — Set up an AWS T2 Medium Ubuntu EC2 Instance.

You can select an existing key pair, and enable HTTP and HTTPS Traffic. Launch the instance and once it is launched you can connect to it using the key pair. Name it as Main-Server





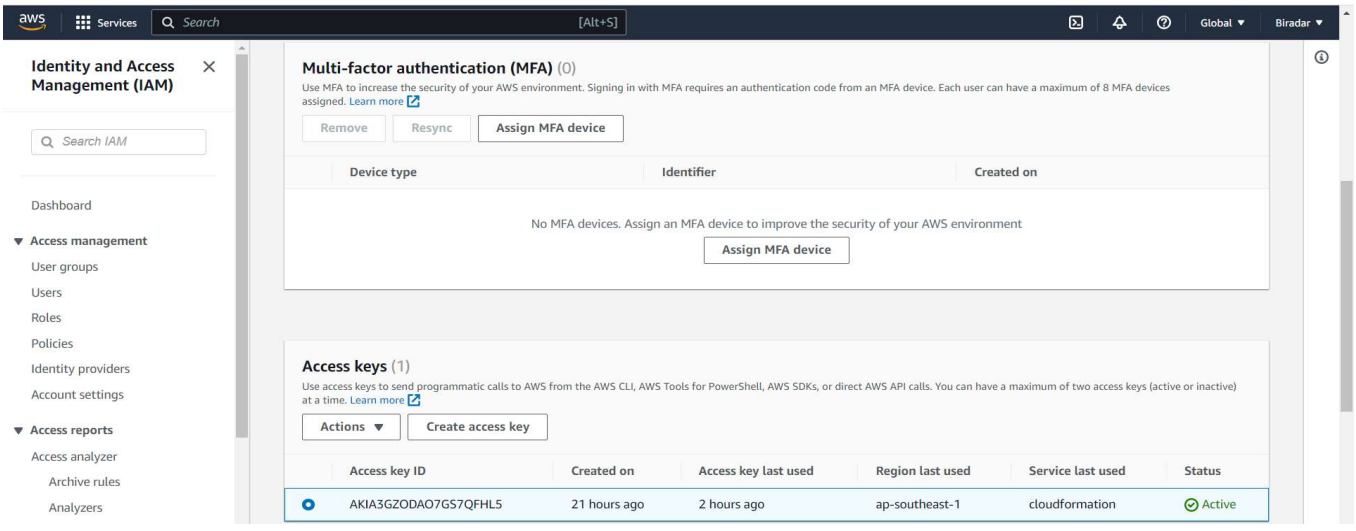
Once it is launched, connect to the instance via console or using SSH Key Pair. Once you have connected to your EC2 Instance, you need to install and configure AWS CLI using the below command

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install unzip
unzip awscliv2.zip
sudo ./aws/install
aws --version
```

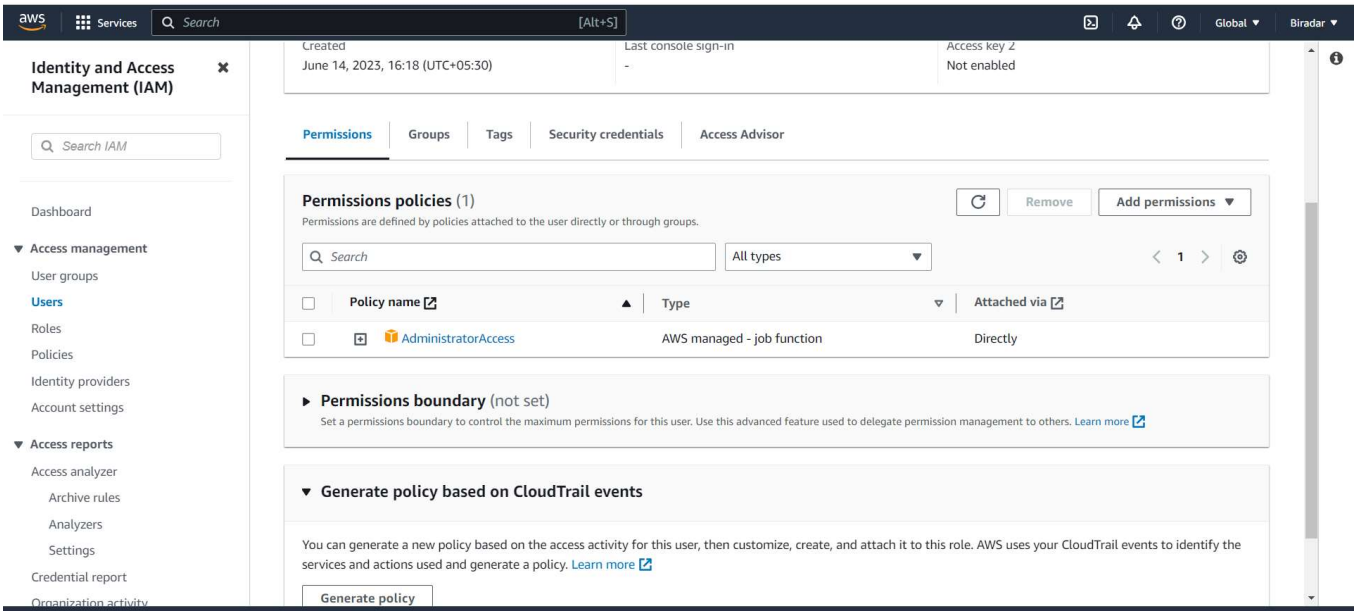
Step 2— Now, let's configure AWS Credentials so that it can authenticate and communicate with AWS Environment.
Here are the instructions for fetching the AWS credentials -

A) To set up the AWS credentials you first need to fetch them from your AWS account.

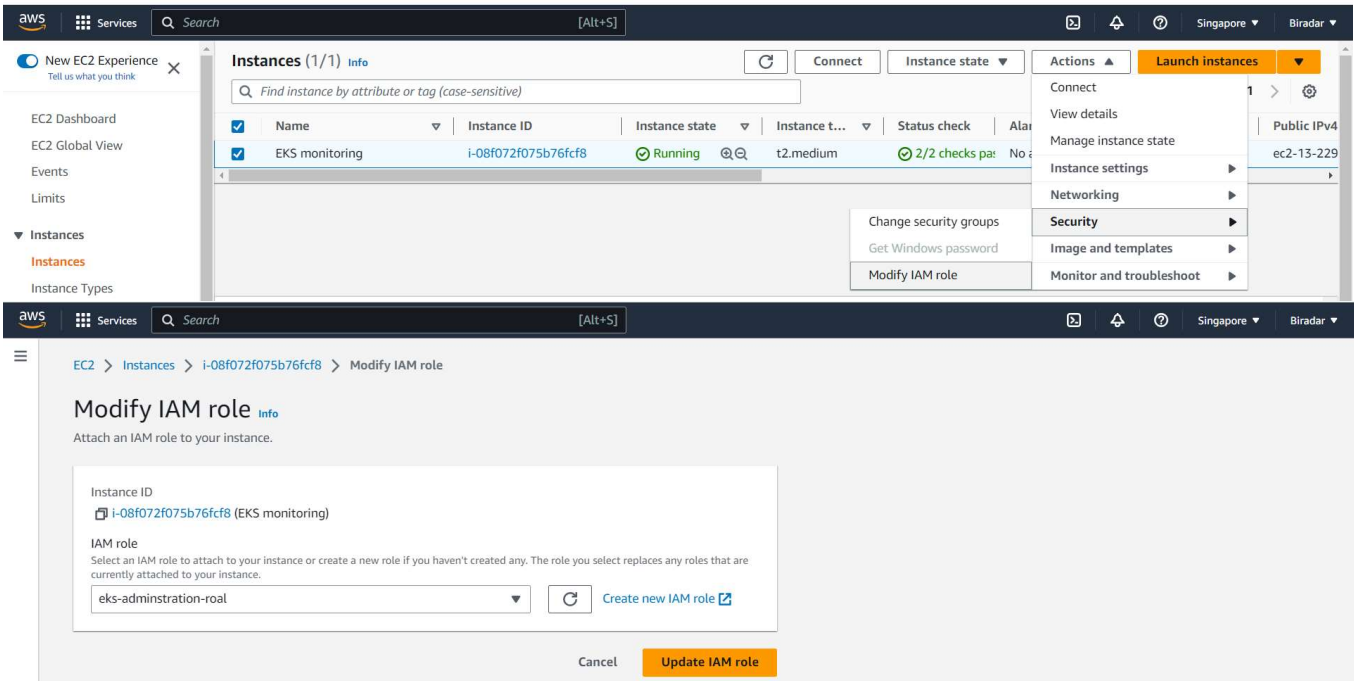
1. Goto your AWS account
2. Then Goto Security Credentials
3. Click on *Access Keys*
4. Then *Create New Access Key*



Creta IAM Rol with administration access



Go to AWS console, click on EC2, select EC2 instance, Choose Security. Click on Modify IAM Role



Setup the AWS Credentials -

1. Open terminal/command prompt
2. Run the command ***aws configure*** and then enter ***AWS Access key ID*** and ***AWS secret access key***

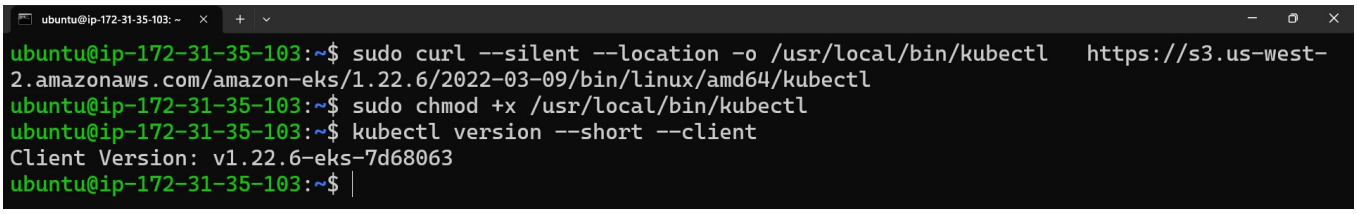
Step 3 — Install and setup kubectl

```
sudo curl --silent --location -o /usr/local/bin/kubectl   https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl

sudo chmod +x /usr/local/bin/kubectl

kubectl version --short --client
```

It should return you back with the Kubernetes version -



Step 4— Install and set up eksctl

Download and extract the latest release of eksctl with below command

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

Move the extracted binary to /usr/local/bin

```
sudo mv /tmp/eksctl /usr/local/bin
```

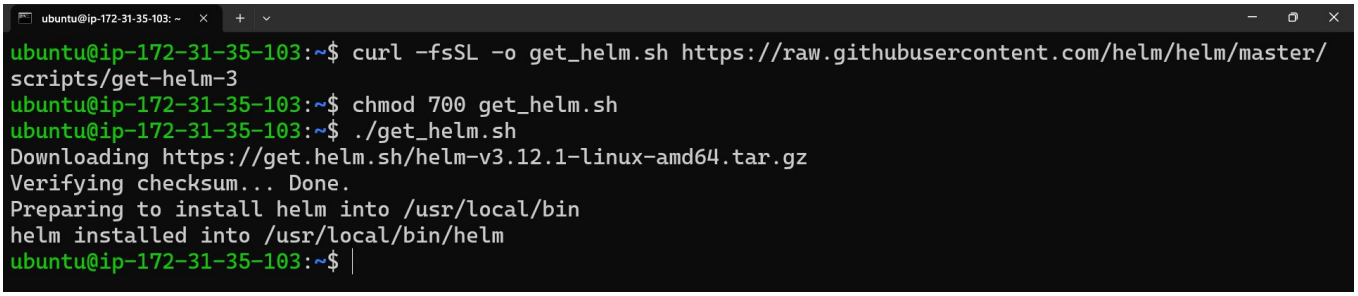
Check whether the installation has been successfully done using

```
eksctl version
```

Step 5— Install Helm Chart

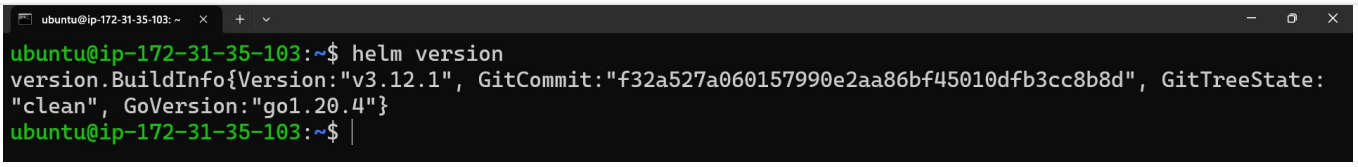
The next tool we need is Helm Chart. Helm is a package manager for Kubernetes, an open-source container orchestration platform. Helm helps you manage Kubernetes applications by making it easy to install, update, and delete them. Use the below script to install helm

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```



You can verify the helm installation using

```
helm version
```



Now, here we are done with the installation of **AWS, CLI**, **kubectl**, **eksctl** and **Helm**

Step 6— Now we will configure the Amazon EKS cluster using eksctl. You can refer to the detailed documentation here https://docs.aws.amazon.com/eks/latest/userguide/clusters.html

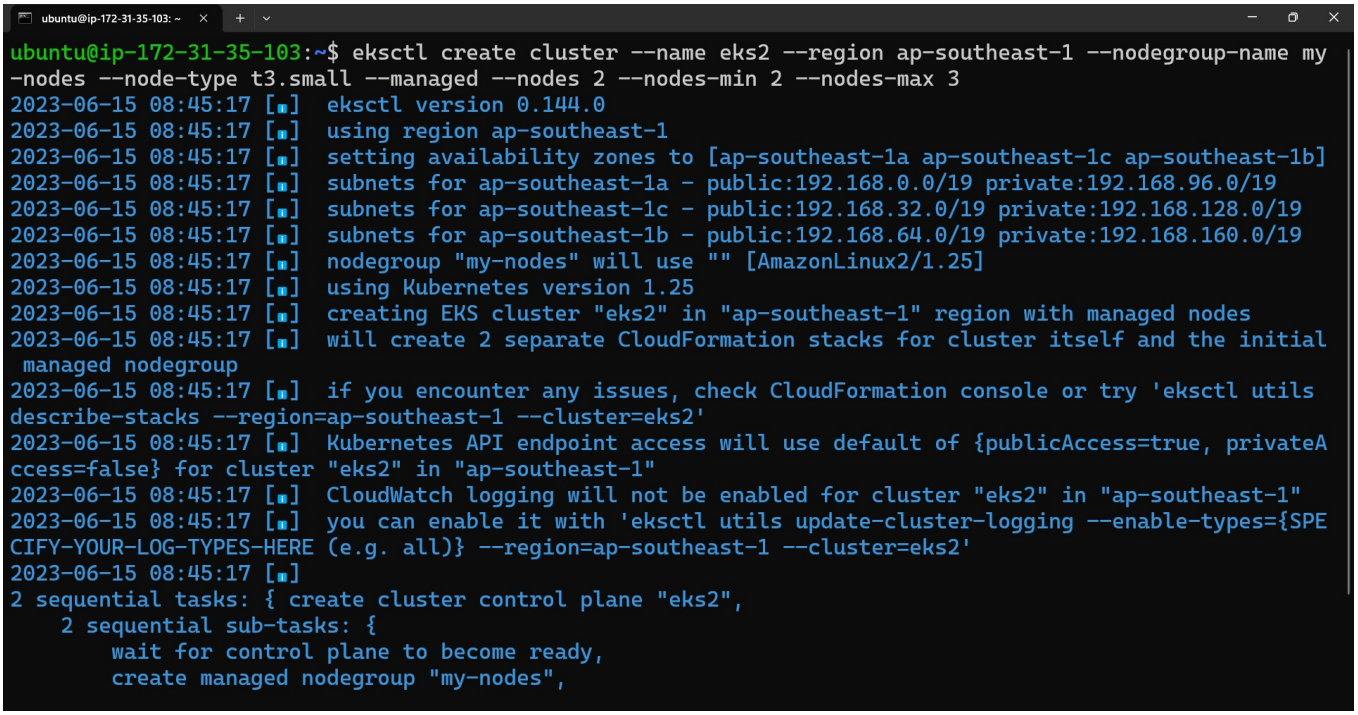
You need the following in order to run the eksctl command

1. Name of the cluster : — eks4
2. Version of Kubernetes : — version 1.24
3. Region : — region us-east-1
4. Nodegroup name/worker nodes : — nodegroup-name worker-nodes
5. Node Type : — nodegroup-type t3.small
6. Number of nodes: — nodes 2
7. Minimum Number of nodes: — nodes-min 2
8. Maximum Number of nodes: — nodes-max 3

Here is the eksctl command

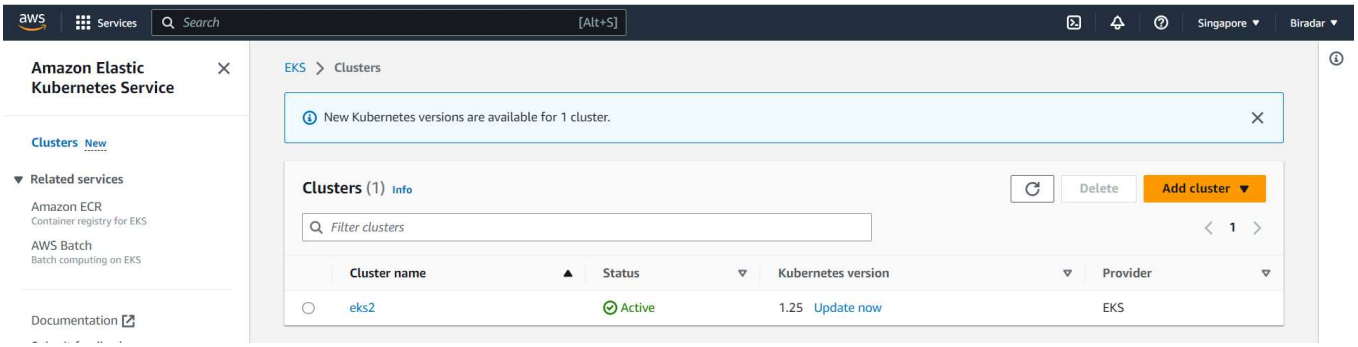
```
eksctl create cluster --name eks2 --region us-east-1 --nodegroup-name my-nodes --node-type t3.small --managed --nodes 2 --nodes-min 2 --nodes-max 3
```

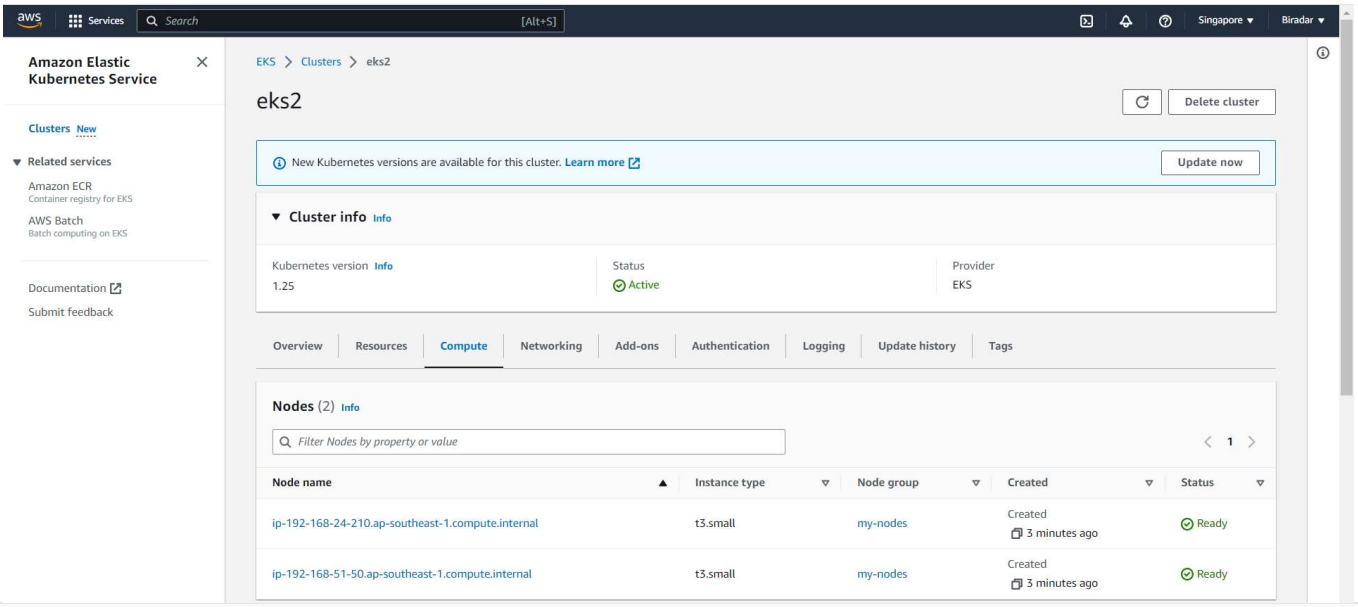
Kindly note that it would take 15–20 minutes for this installation to complete. Once it is done, you can go to your AWS Console and look for the eksctl clusters



```
2023-06-15 08:51:17 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-cluster"
2023-06-15 08:52:17 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-cluster"
2023-06-15 08:53:17 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-cluster"
2023-06-15 08:55:18 [ℹ]  building managed nodegroup stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:55:18 [ℹ]  deploying stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:55:18 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:55:49 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:56:46 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:57:19 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:58:51 [ℹ]  waiting for CloudFormation stack "eksctl-eks2-nodegroup-my-nodes"
2023-06-15 08:58:51 [ℹ]  waiting for the control plane to become ready
2023-06-15 08:58:52 [✔]  saved kubeconfig as "/home/ubuntu/.kube/config"
2023-06-15 08:58:52 [ℹ]  no tasks
2023-06-15 08:58:52 [✔]  all EKS cluster resources for "eks2" have been created
2023-06-15 08:58:52 [ℹ]  nodegroup "my-nodes" has 2 node(s)
2023-06-15 08:58:52 [ℹ]  node "ip-192-168-24-210.ap-southeast-1.compute.internal" is ready
2023-06-15 08:58:52 [ℹ]  node "ip-192-168-51-50.ap-southeast-1.compute.internal" is ready
2023-06-15 08:58:52 [ℹ]  waiting for at least 2 node(s) to become ready in "my-nodes"
2023-06-15 08:58:52 [ℹ]  nodegroup "my-nodes" has 2 node(s)
2023-06-15 08:58:52 [ℹ]  node "ip-192-168-24-210.ap-southeast-1.compute.internal" is ready
2023-06-15 08:58:52 [ℹ]  node "ip-192-168-51-50.ap-southeast-1.compute.internal" is ready
2023-06-15 08:58:54 [ℹ]  kubectl command should work with "/home/ubuntu/.kube/config", try 'kubectl get
nodes'
2023-06-15 08:58:54 [✔]  EKS cluster "eks2" in "ap-southeast-1" region is ready
ubuntu@ip-172-31-35-103:~$
```

We can verify the cluster by logging into the AWS Console






```
eksctl get cluster --name eks2 --region ap-southeast-1
```

This should confirm that EKS cluster is up and running.

```
ubuntu@ip-172-31-35-103:~$ eksctl get cluster --name eks2 --region ap-southeast-1
NAME    VERSION STATUS  CREATED                 VPC                     SUBNETS                         S
ECURITYGROUPS           PROVIDER
eks2    1.25    ACTIVE  2023-06-15T08:45:54Z    vpc-0964b7eab5c60e6a3   subnet-006c9c11486693721,subnet-
02a52be97875a08a5,subnet-04ffd66d93a838da9,subnet-074356b6ac6c00728,subnet-08be8563916152f34,subnet-0d07
5b3bae70c3263   sg-02484972dc9eaf508    EKS
ubuntu@ip-172-31-35-103:~$
```

Update Kube config by entering below command:

```
aws eks update-kubeconfig --name eks2 --region us-east-1
```

```
ubuntu@ip-172-31-35-103:~$ aws eks update-kubeconfig --name eks2 --region ap-southeast-1
Updated context arn:aws:eks:ap-southeast-1:770506425278:cluster/eks2 in /home/ubuntu/.kube/config
ubuntu@ip-172-31-35-103:~$
```

Connect to EKS cluster using kubectl commands

To view the list of worker nodes as part of EKS cluster.

```
Kubectl get nodes

kubectl get ns
```

```
ubuntu@ip-172-31-35-103:~$ kubectl get nodes
NAME                                            STATUS   ROLES    AGE   VERSION
ip-192-168-24-210.ap-southeast-1.compute.internal   Ready    <none>   13m   v1.25.9-eks-0a21954
ip-192-168-51-50.ap-southeast-1.compute.internal    Ready    <none>   13m   v1.25.9-eks-0a21954
ubuntu@ip-172-31-35-103:~$
```

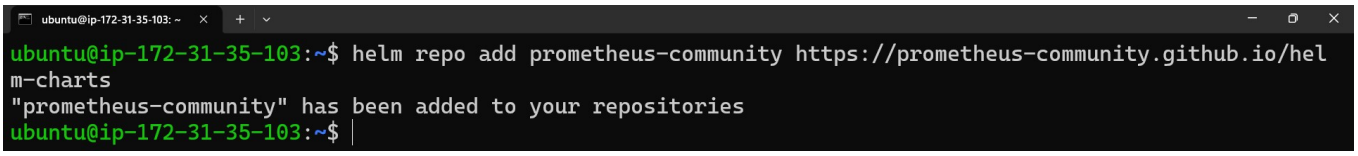
```
ubuntu@ip-172-31-35-103:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   19m
kube-node-lease   Active   20m
kube-public       Active   20m
kube-system       Active   20m
ubuntu@ip-172-31-35-103:~$
```

Step 7— We need to add the Helm Stable Charts for your local client. Execute the below command:

```
helm repo add stable https://charts.helm.sh/stable
```

Step 8— Add Prometheus Helm repo

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

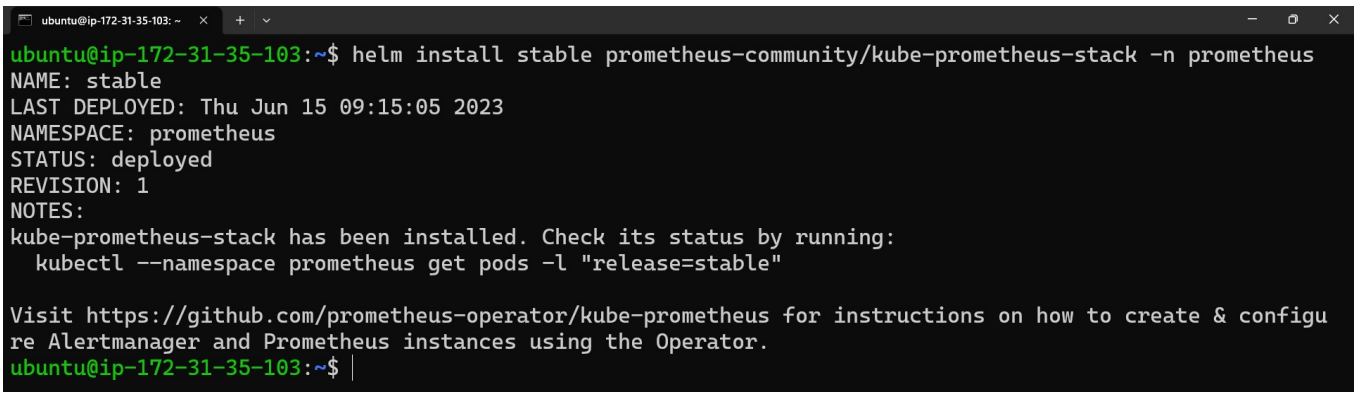


Step 9— Create Prometheus namespace

```
kubectl create namespace prometheus
```

Step 10— Install Prometheus

```
helm install stable prometheus-community/kube-prometheus-stack -n prometheus
```

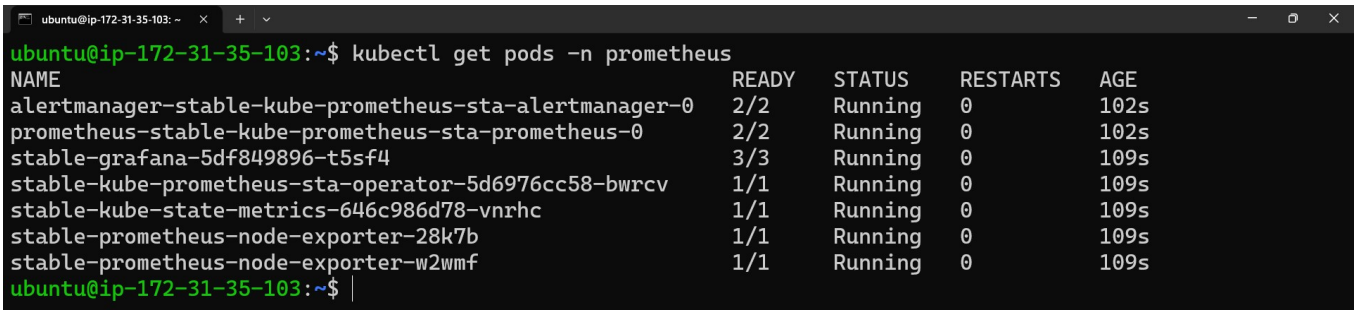


- above command is used to install kube-Prometheus-stack. The helm repo kube-stack-Prometheus comes with a Grafana deployment embedded (as the default one).

Now Prometheus is installed using helm in the ec2 instance

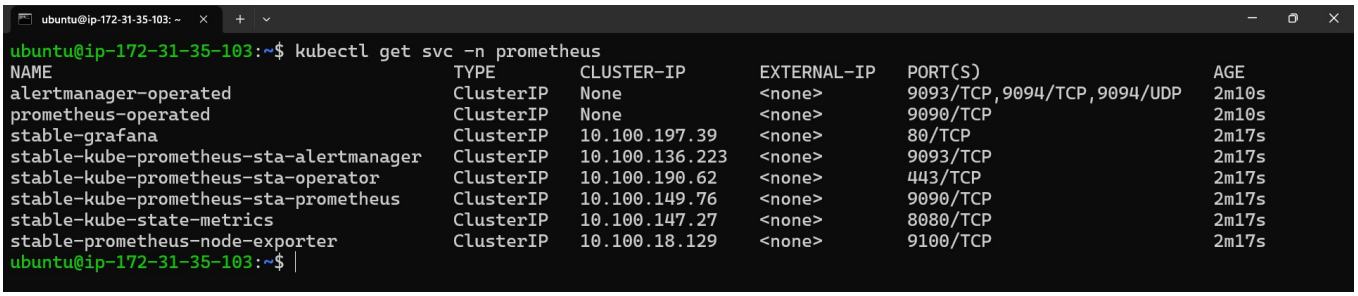
- To check whether Prometheus is installed or not use the below command

```
kubectl get pods -n prometheus
```



to check the services file (svc) of the Prometheus

```
kubectl get svc -n prometheus
```



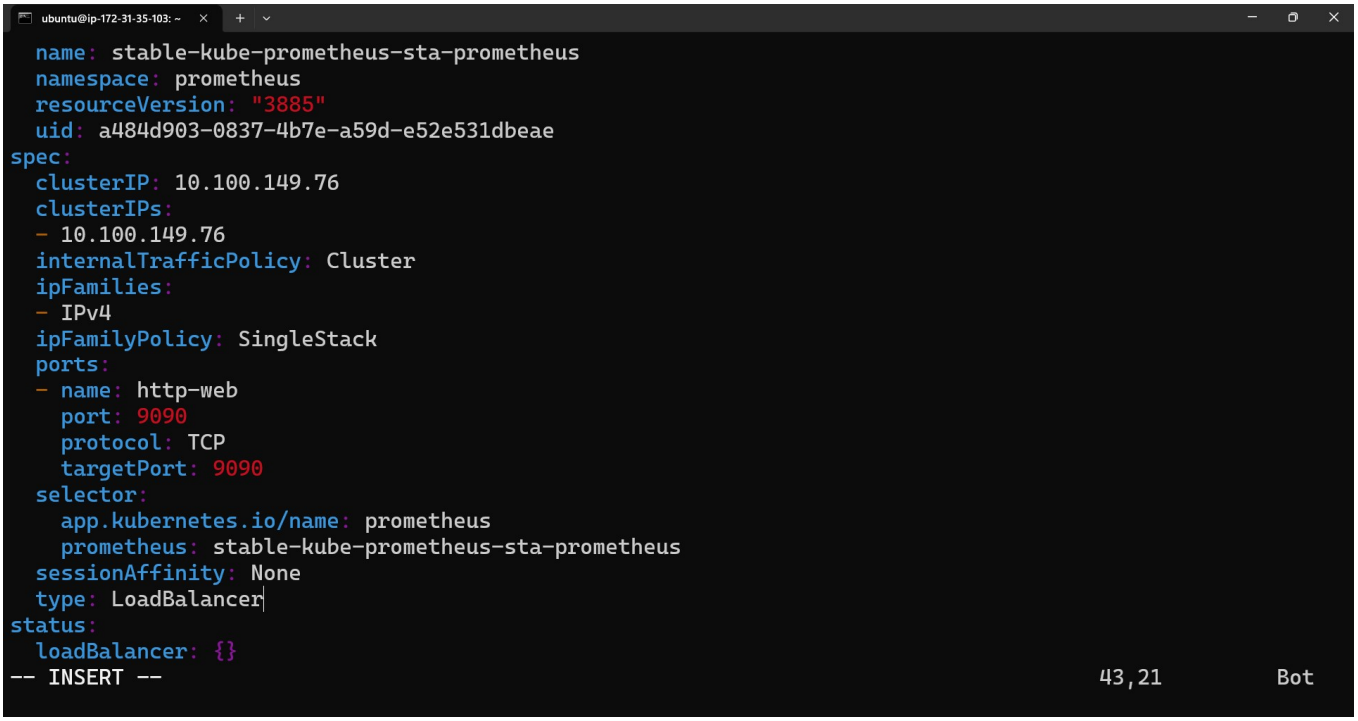
Grafana will be coming along with Prometheus as the stable version
This output is conformation that our Prometheus is installed successfully
there is no need of installing Grafana as a separate tool it comes along with Prometheus

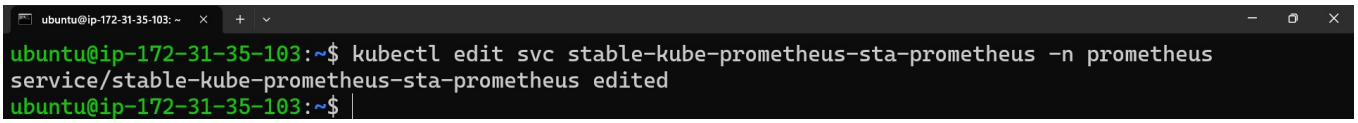
let's expose Prometheus and Grafan to the external world
there are 2 ways to expose

1. through Node Port
2. through LoadBalancer

let's go with the LoadBalancer
to attach the load balancer we need to change from ClusterIP to LoadBalancer
command to get the svc file

```
kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus
```



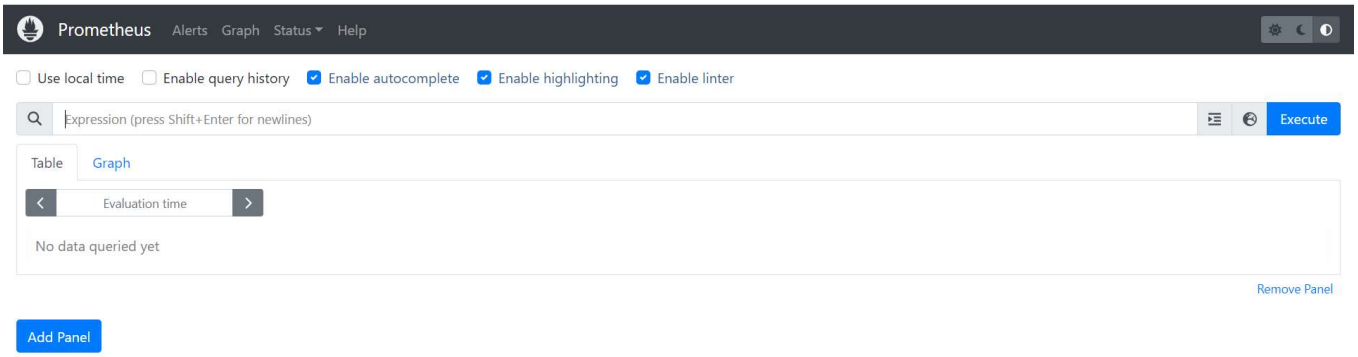


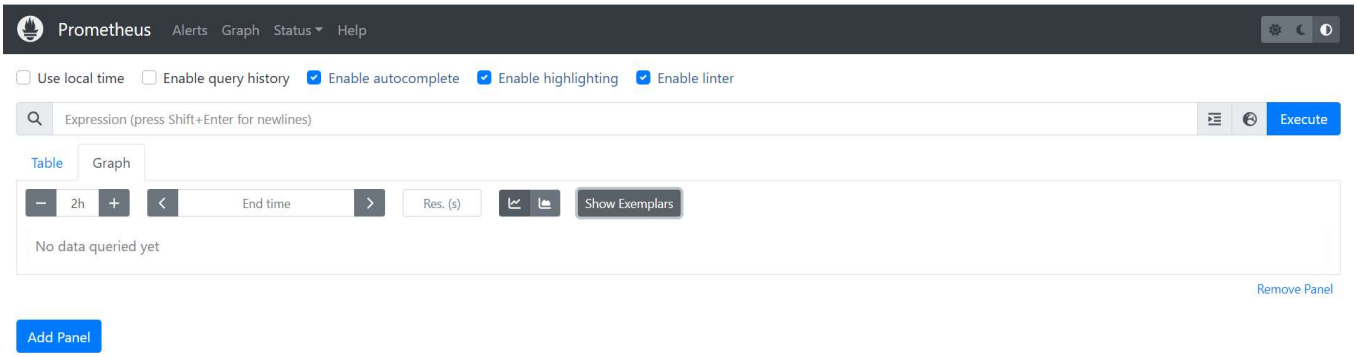
change it from Cluster IP to LoadBalancer after changing make sure you save the file

```
kubectl get svc -n prometheus
```

```
ubuntu@ip-172-31-35-103: ~
ubuntu@ip-172-31-35-103:~$ kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus
service/stable-kube-prometheus-sta-prometheus edited
ubuntu@ip-172-31-35-103:~$ kubectl get svc -n prometheus
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)                      AGE
alertmanager-operated                     ClusterIP      None             <none>                                                                   9093/TCP,9094/TCP,9094/UDP   5m38s
prometheus-operated                       ClusterIP      None             <none>                                                                   9090/TCP                     5m38s
stable-grafana                            ClusterIP      10.100.197.39    <none>                                                                   80/TCP                       5m45s
stable-kube-prometheus-sta-alertmanager   ClusterIP      10.100.136.223   <none>                                                                   9093/TCP                     5m45s
stable-kube-prometheus-sta-operator       ClusterIP      10.100.190.62    <none>                                                                   443/TCP                      5m45s
stable-kube-prometheus-sta-prometheus     LoadBalancer   10.100.149.76    aa484d90308374b7ea59de52e531dbea-555742178.ap-southeast-1.elb.amazonaws.com   9090:31258/TCP               5m45s
stable-kube-state-metrics                 ClusterIP      10.100.147.27    <none>                                                                   8080/TCP                     5m45s
stable-prometheus-node-exporter           ClusterIP      10.100.18.129    <none>                                                                   9100/TCP                     5m45s
ubuntu@ip-172-31-35-103:~$
```

you guys can see I have a load balancer for my Prometheus which I can access from that link port 9090





we can use a Prometheus UI for monitoring the EKSbut the UI of Prometheus is not a convent for the user Grafana will extract the matrix from the Prometheus UI and show it in a user-friendly manner

let's change the SVC file of the Grafana and expose it to the outer world

command to edit the SVC file of grafana

```
kubectl edit svc stable-grafana -n prometheus
```

```
spec:
  clusterIP: 10.100.197.39
  clusterIPs:
  - 10.100.197.39
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

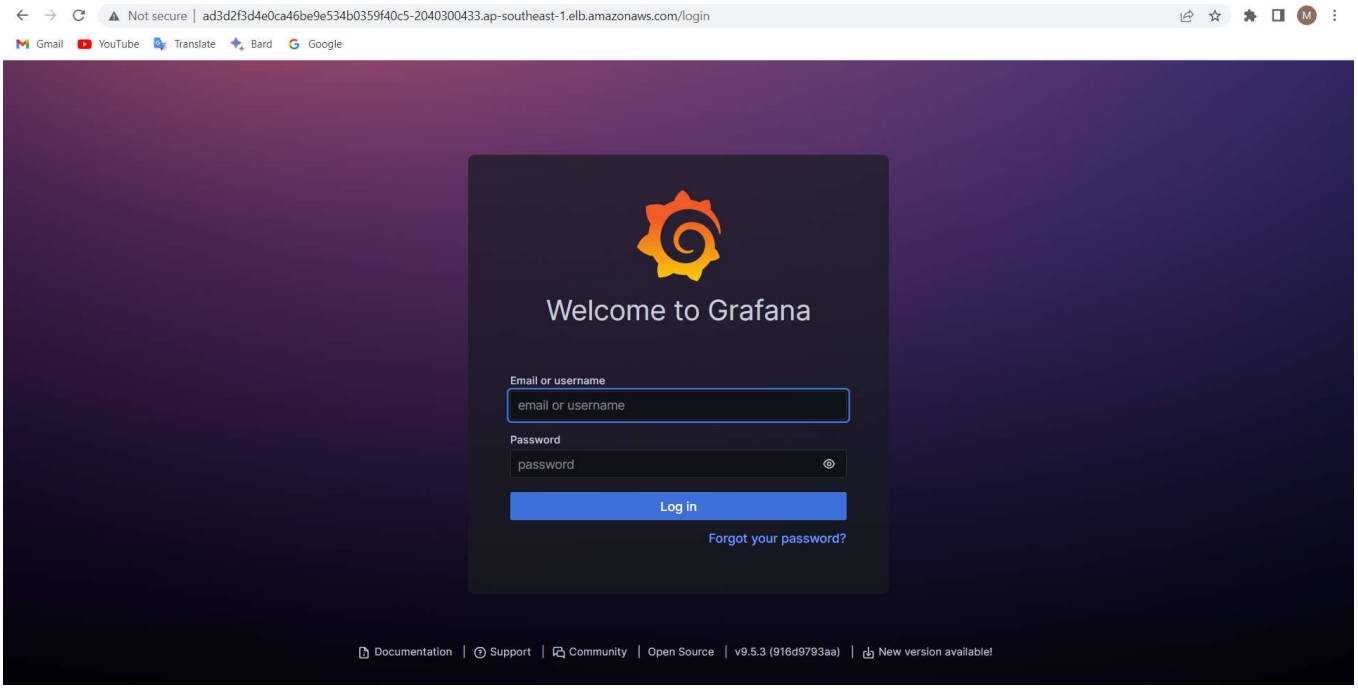
the Grafana LoadBalancer also exposed

```
ubuntu@ip-172-31-35-103:~$ kubectl edit svc stable-grafana -n prometheus
service/stable-grafana edited
```

```
kubectl get svc -n prometheus
```

```
ubuntu@ip-172-31-35-103:~$ kubectl get svc -n prometheus
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)                      AGE
alertmanager-operated                     ClusterIP      None             <none>                                                                   9093/TCP,9094/TCP,9094/UDP   17m
prometheus-operated                       ClusterIP      None             <none>                                                                   9090/TCP                     17m
stable-grafana                            LoadBalancer   10.100.197.39    ad3d2f3d4e0ca46be9e534b0359f40c5-2040300433.ap-southeast-1.elb.amazonaws.com   80:30362/TCP                 17m
stable-kube-prometheus-sta-alertmanager   ClusterIP      10.100.136.223   <none>                                                                   9093/TCP                     17m
stable-kube-prometheus-sta-operator       ClusterIP      10.100.190.62    <none>                                                                   443/TCP                      17m
stable-kube-prometheus-sta-prometheus     LoadBalancer   10.100.149.76    aa484d90308374b7ea59de52e531dbea-555742178.ap-southeast-1.elb.amazonaws.com   9090:31258/TCP               17m
stable-kube-state-metrics                 ClusterIP      10.100.147.27    <none>                                                                   8080/TCP                     17m
stable-prometheus-node-exporter           ClusterIP      10.100.18.129    <none>                                                                   9100/TCP                     17m
ubuntu@ip-172-31-35-103:~$
```

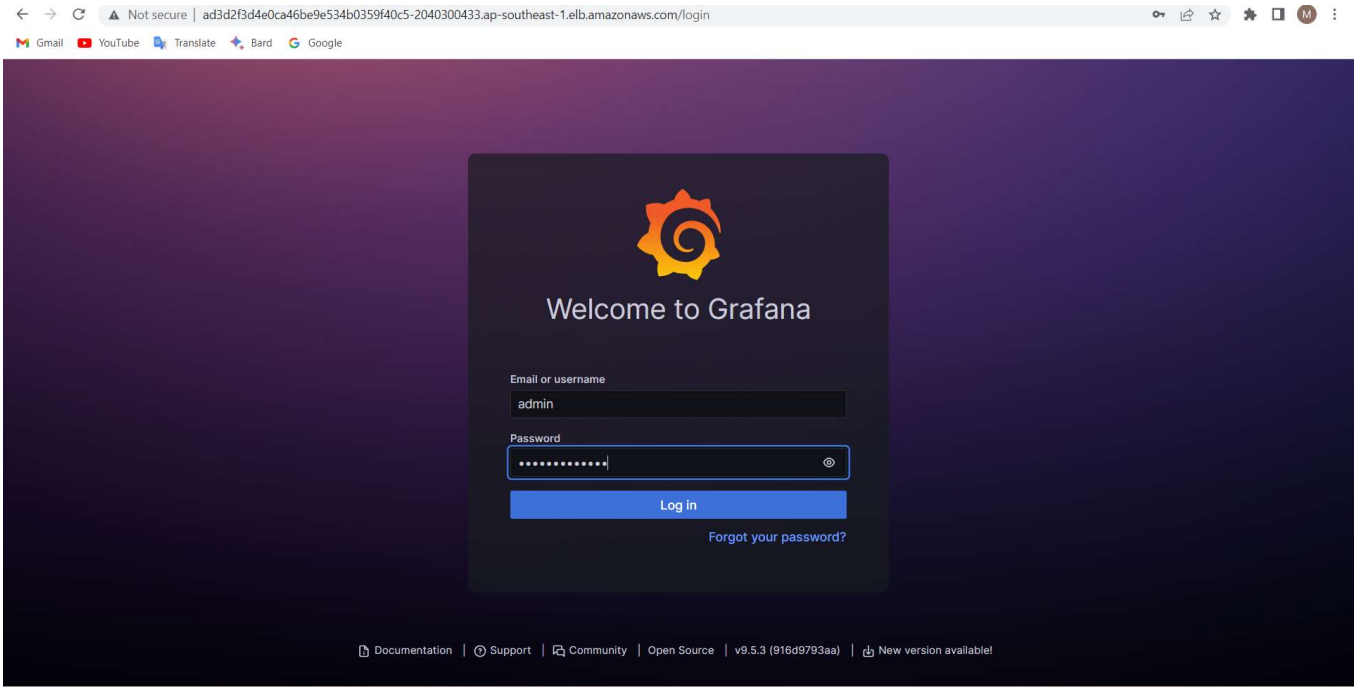
use the link of the LoadBalancer and access from the Browser



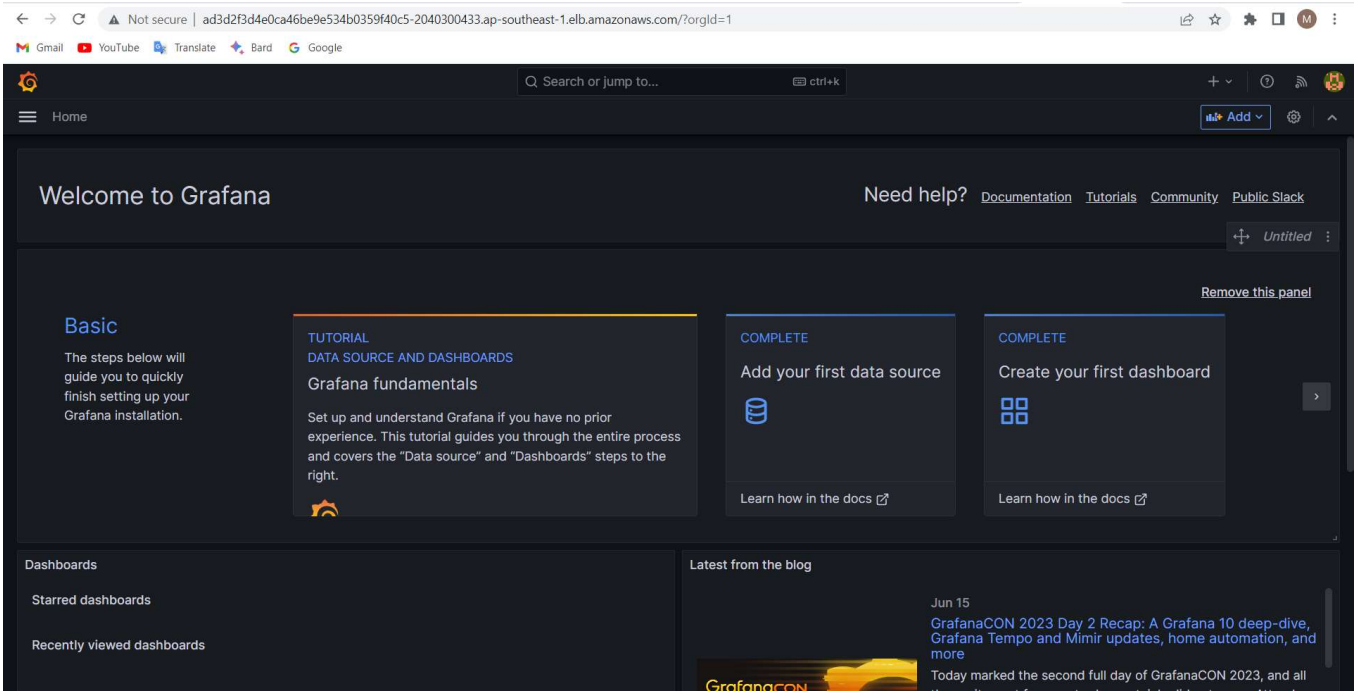
```
kubectl get secret --namespace prometheus stable-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

use the above command to get the password

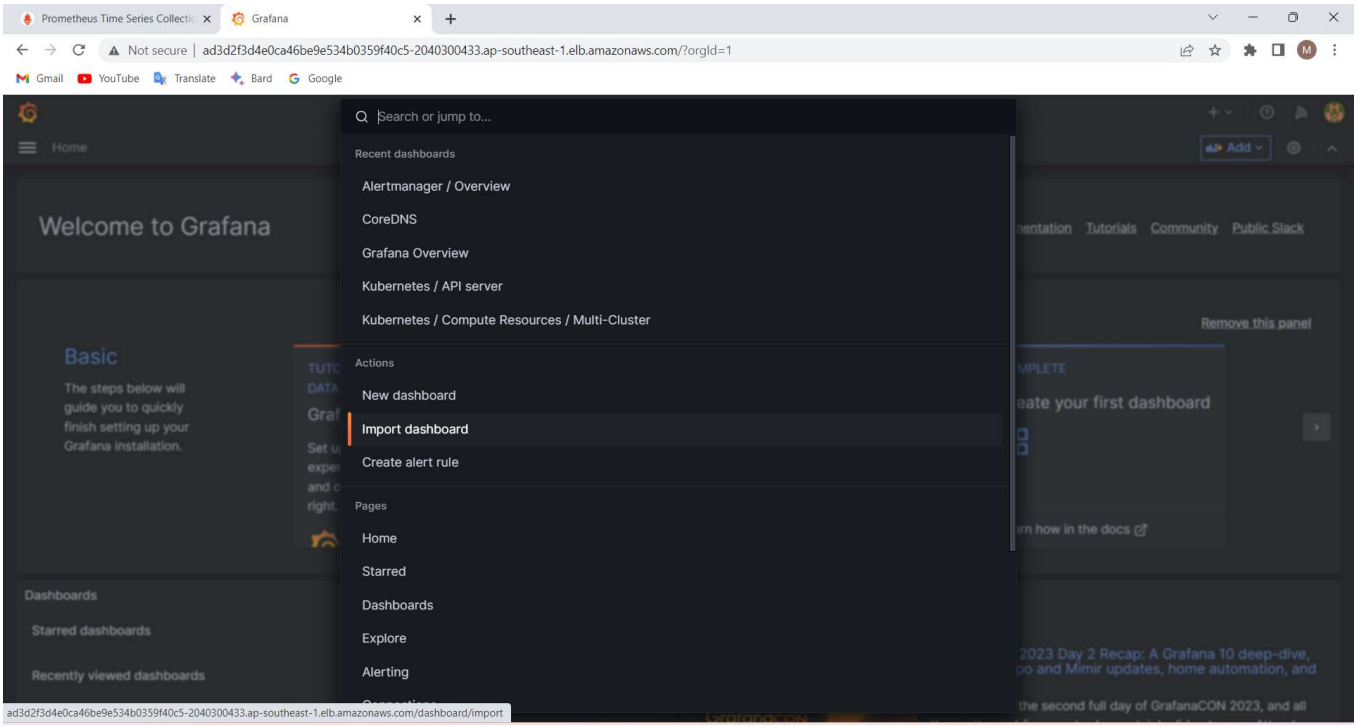
the user name is admin



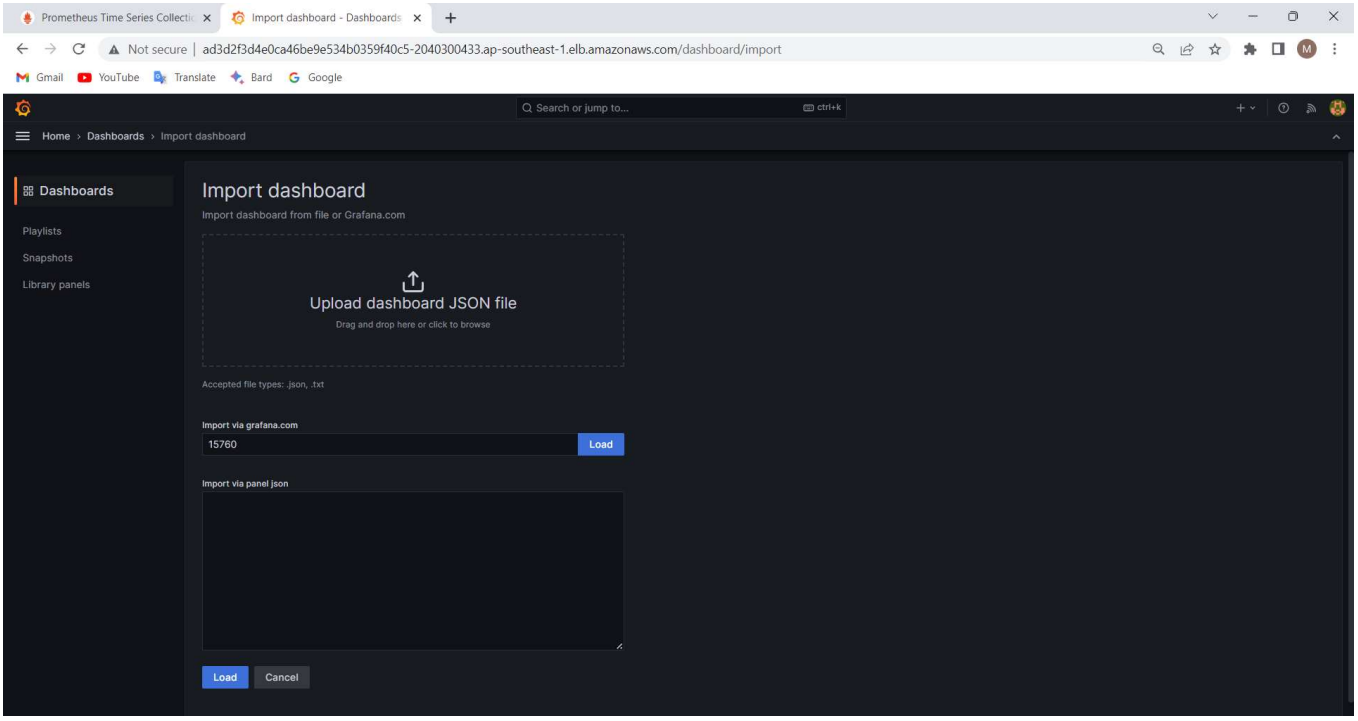
Create a Dashboard in Grafana

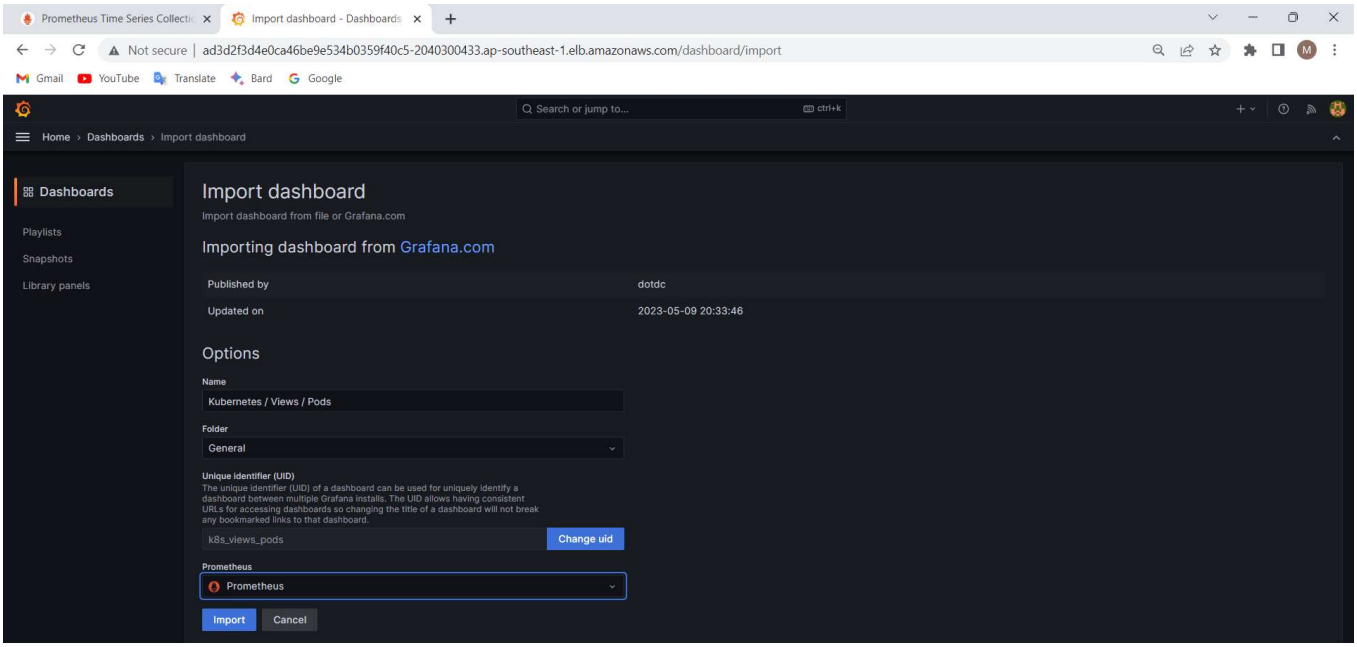


let’s create a Dashboard by importing
click on Import and import the dashboard with numbers

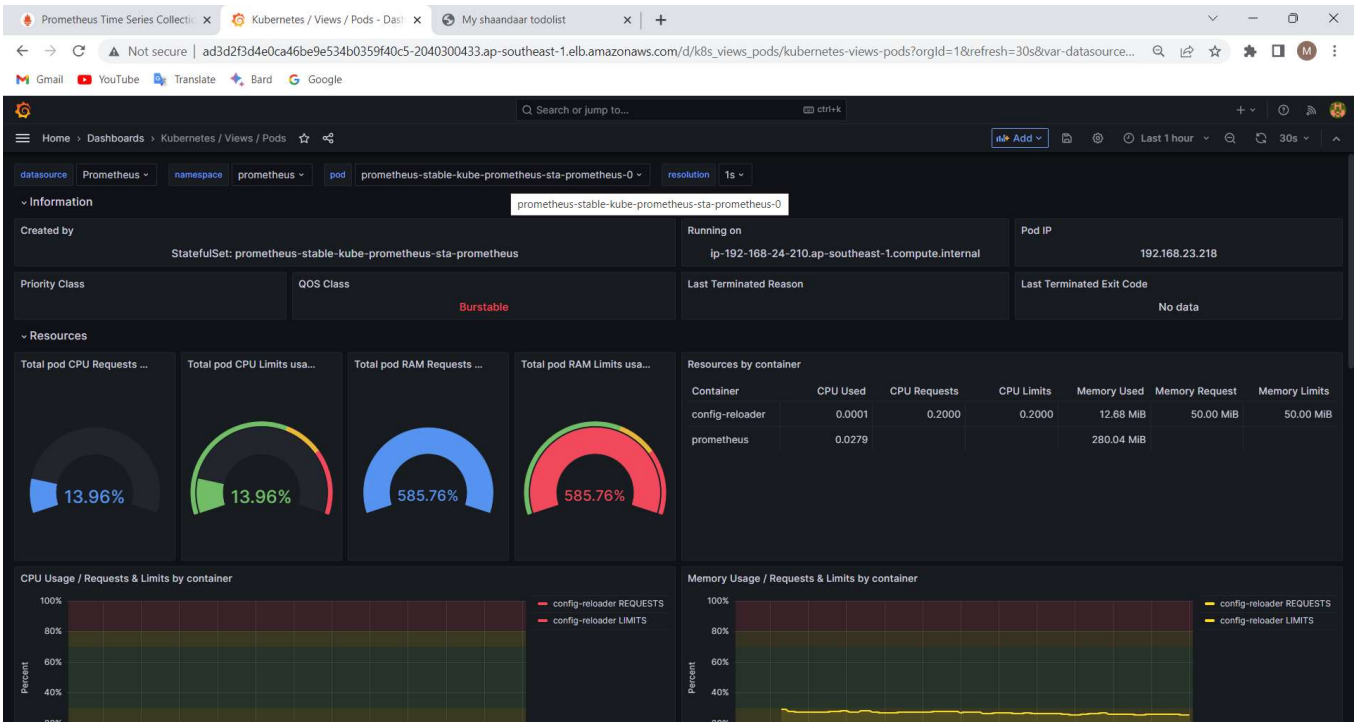


there are plenty of ready templates to use the pre-existing templates and modify based on our desired
it uses a Prometheus





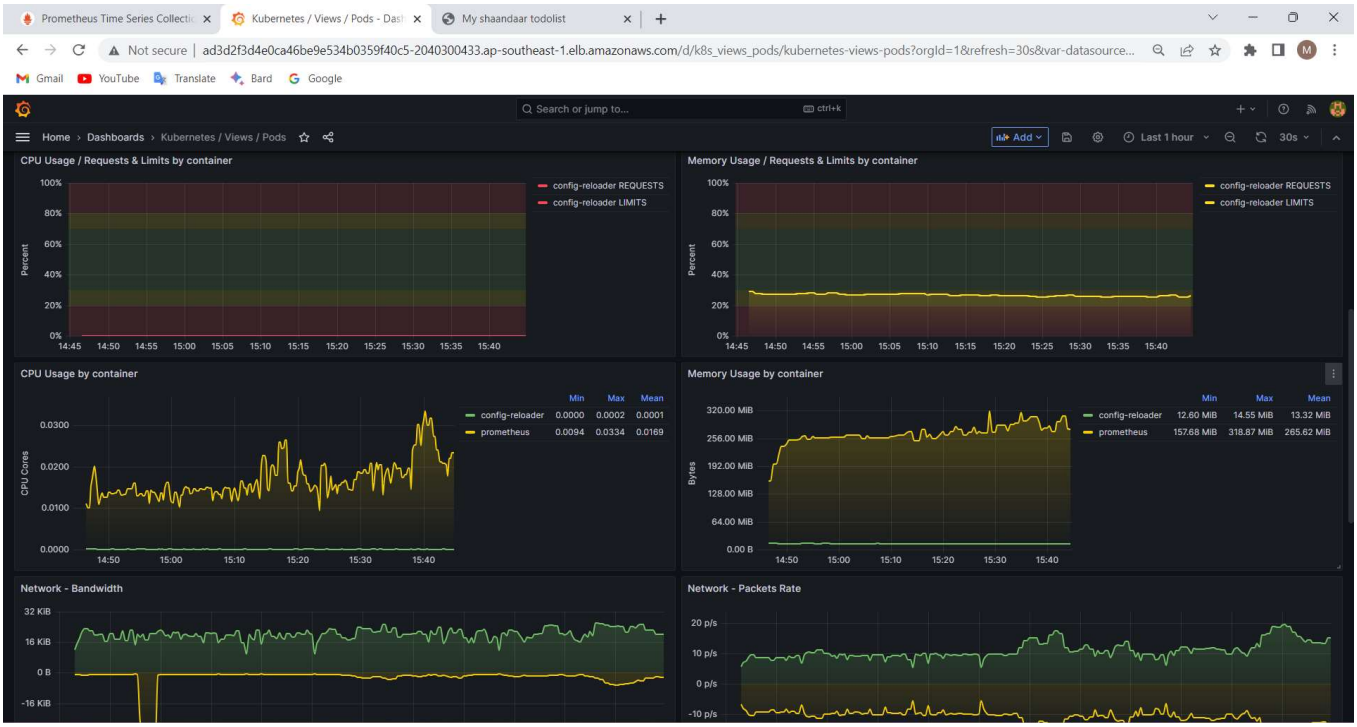
click on the Import



the Entire data of the cluster

where we can able to see the entire data of the EKS cluster

1. CPU and RAM use
2. pods in a specific namespace



3. Pod up history
    4. HPA
    5. Resources by Container

CPU used by container & limits

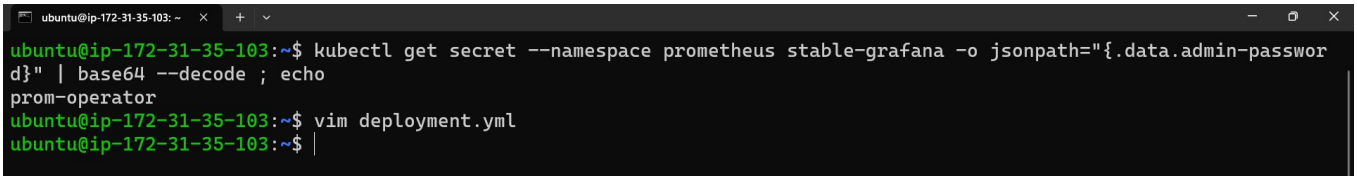
network bandwidth & packet rate

Step 11— Install Node JS application and monitor it on Grafana

To make use of Grafana dashboard, we will deploy Node.js application on Kubernetes. Download deployment.yml file from the below.

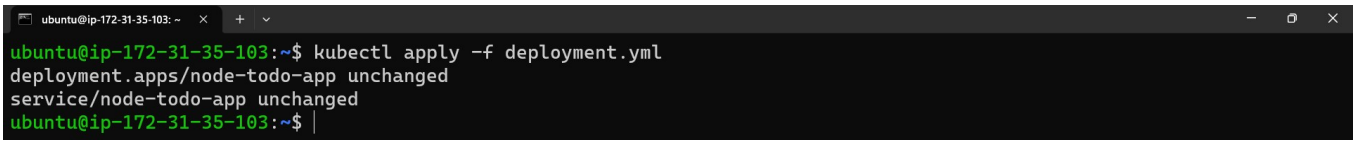
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-todo-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: node-todo-app
  template:
    metadata:
      labels:
        app: node-todo-app
    spec:
      containers:
```

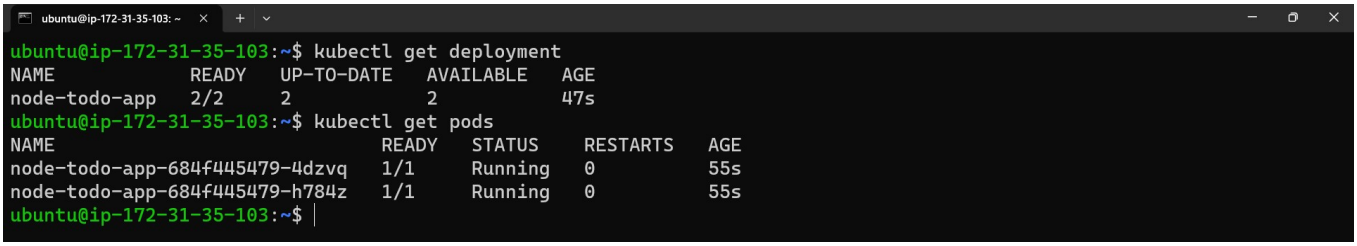
```
        - name: node-todo-app
          image: mahesh8887/node-todo-app:latest
          ports:
            - containerPort: 8000
          env:
            - name: PORT
              value: "8000"
---
apiVersion: v1
kind: Service
metadata:
  name: node-todo-app
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8000
  selector:
    app: node-todo-app
```



To deploy the Node.js application on kubernetes cluster user the following `kubectl` command. Verify the deployment by running the following `kubectl` command

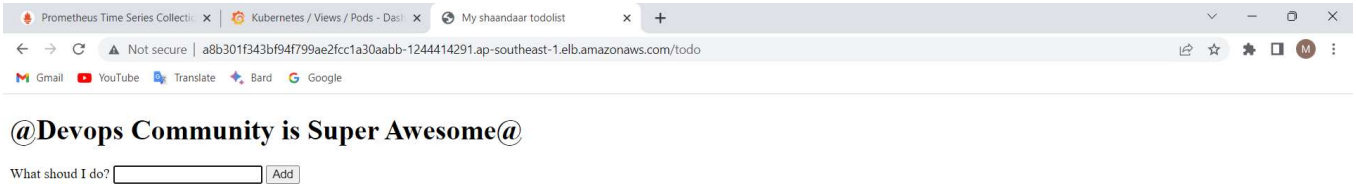
```
kubectl apply -f deployment.yml
kubectl get deployment
kubectl get pods
```



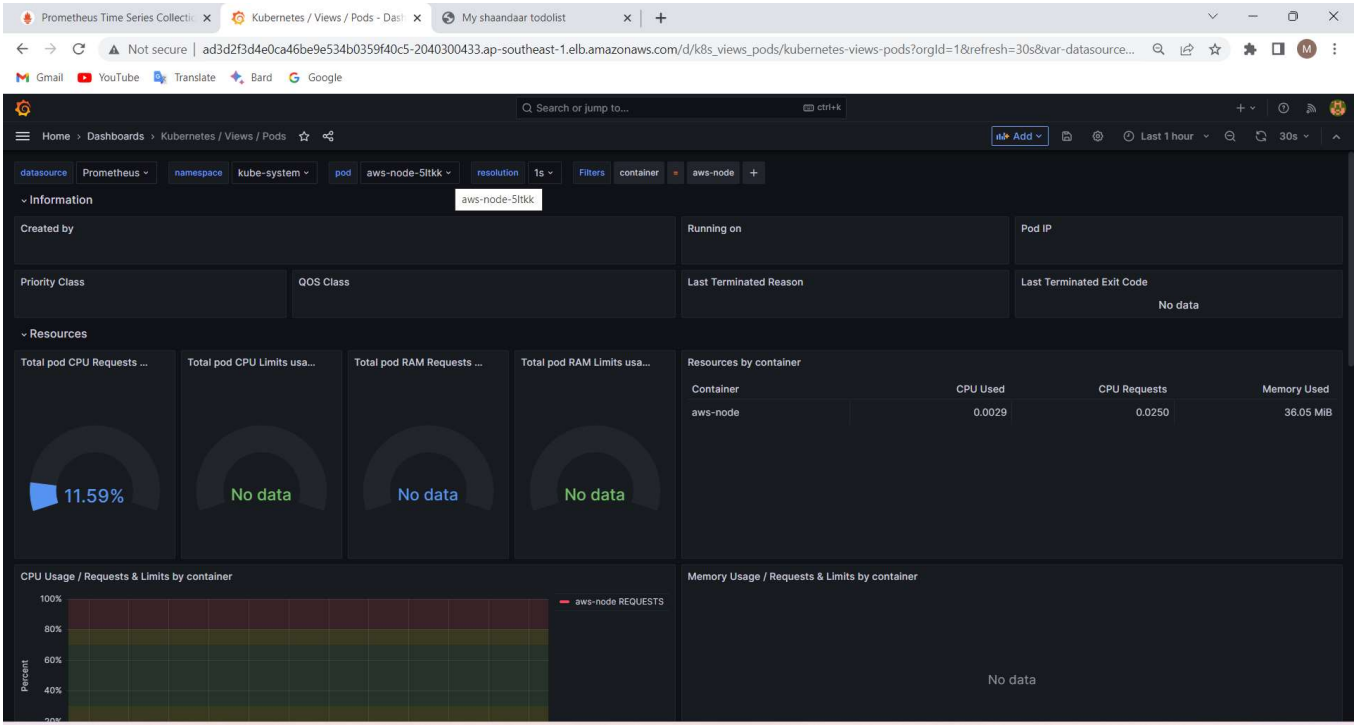


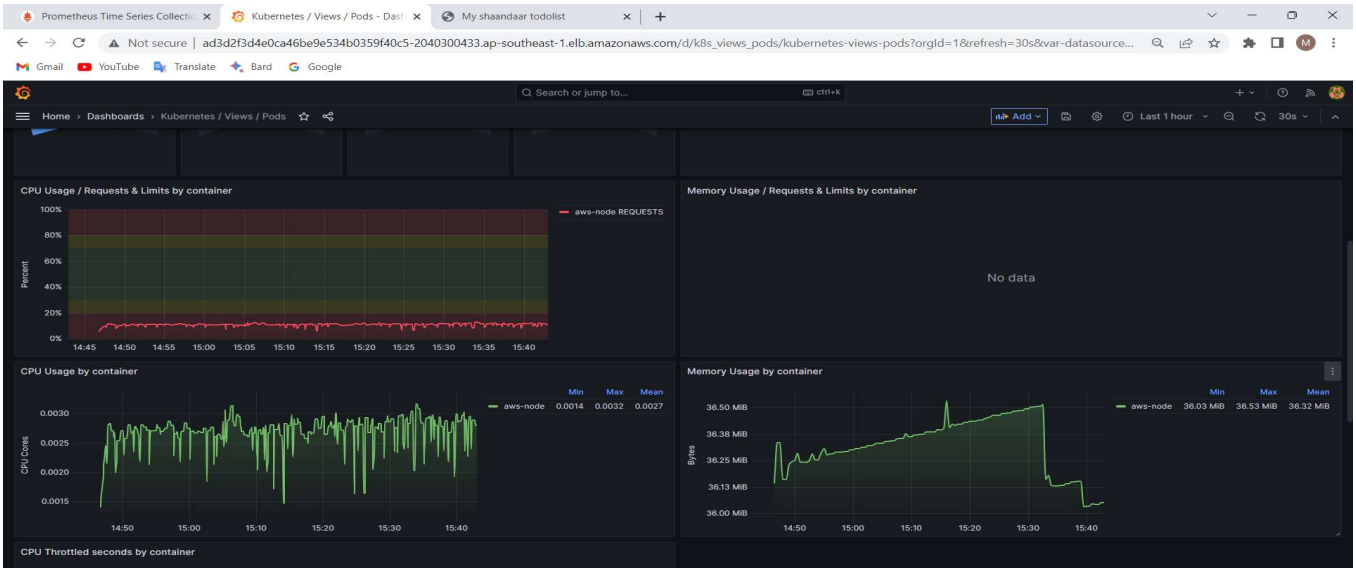
The Node.js Application is running successfully.

```
ubuntu@ip-172-31-35-103:~$ kubectl get svc
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP
          PORT(S)        AGE
kubernetes     ClusterIP      10.100.0.1       <none>
          443/TCP        73m
node-todo-app  LoadBalancer   10.100.239.252   a8b301f343bf94f799ae2fcc1a30aabb-1244414291.ap-southeast-1.elb.amazon
aws.com   80:30562/TCP   2m9s
ubuntu@ip-172-31-35-103:~$
```



Refresh the Grafana dashboard to verify the deployment





Step 12 — Clean up/Deprovision

Now we will deprovision all our resources. First, lets start with EKS cluster

```
eksctl delete cluster --name eks2
```

Note: we can deploy the Grafana and Prometheus with a different method

1. Create all configuration files of both Prometheus and Grafana and execute them in the right order.

2. Prometheus Operator — to simplify and automate the configuration and management of the Prometheus monitoring stack running on a Kubernetes cluster

3. Helm chart — Using helm to install Prometheus Operator including Grafana

Helm is the best Practice among all the methods.

Hope you found this useful. Please feel free to reach out to me if you need any assistance while doing EKS Monitoring using Prometheus and Grafana with the help of AWS CLI, kubectl, eksctl and helm utility.

Thank you 🙏 !