**Configuring, Provisioning, and Monitoring Node.js Application - Capstone Project-NEW**

**Step1 : Creating Node.js application**

1.Create a EC2 machine in AWS and connect to it.

2.Create a directory “**app**”

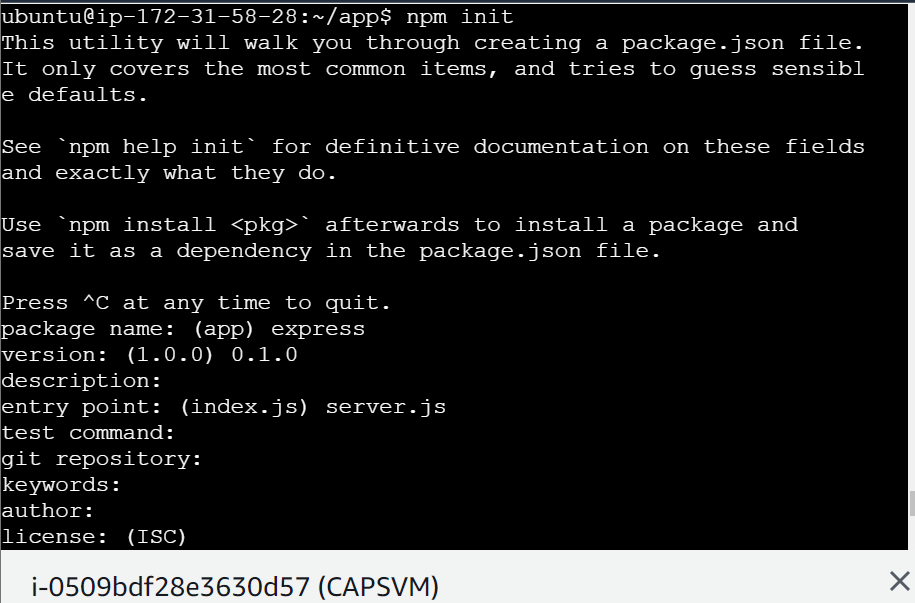
3.Navigate inside the app directory **cd app**

4. Install npm package:

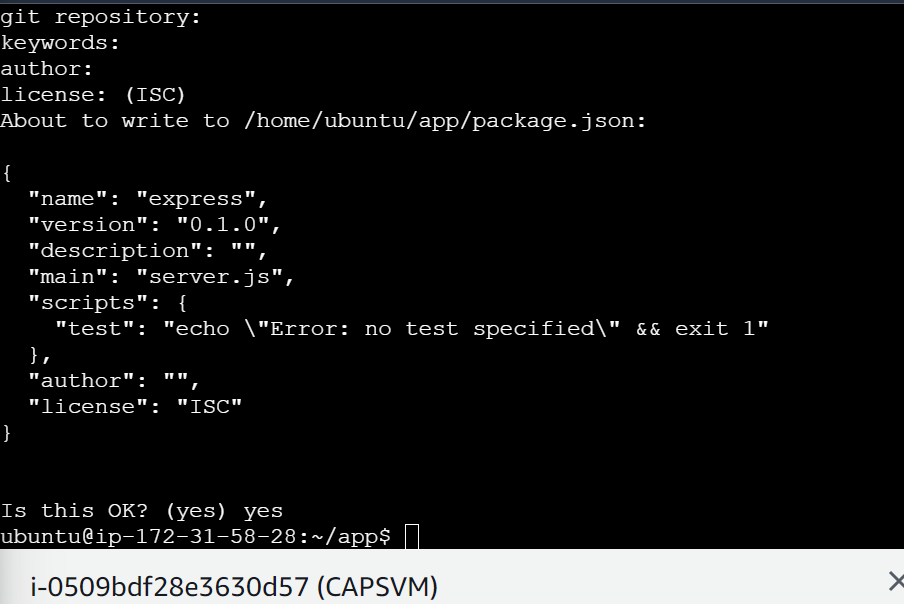
**sudo apt-get update**

**sudo apt install npm**

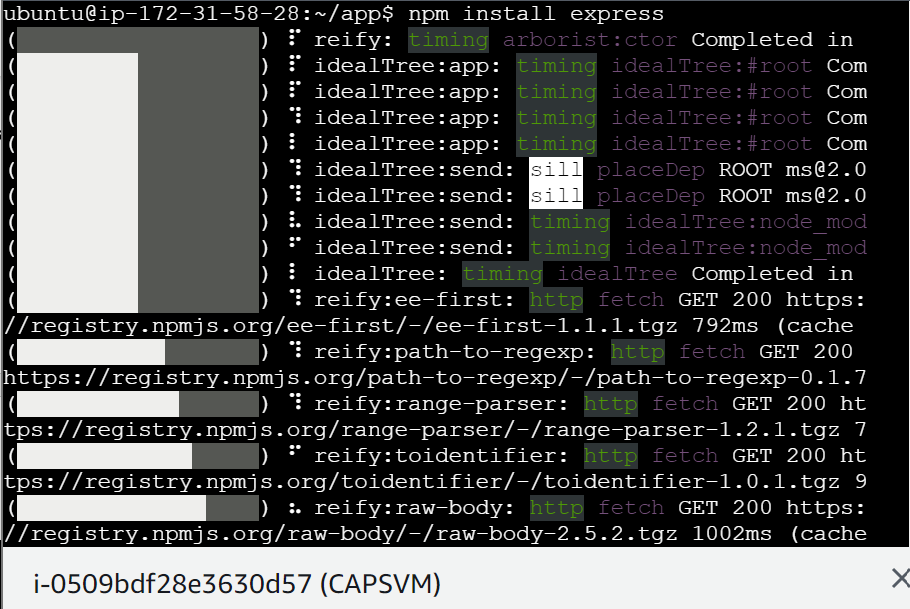
5. Initialize a Node.js project (using npm): npm init

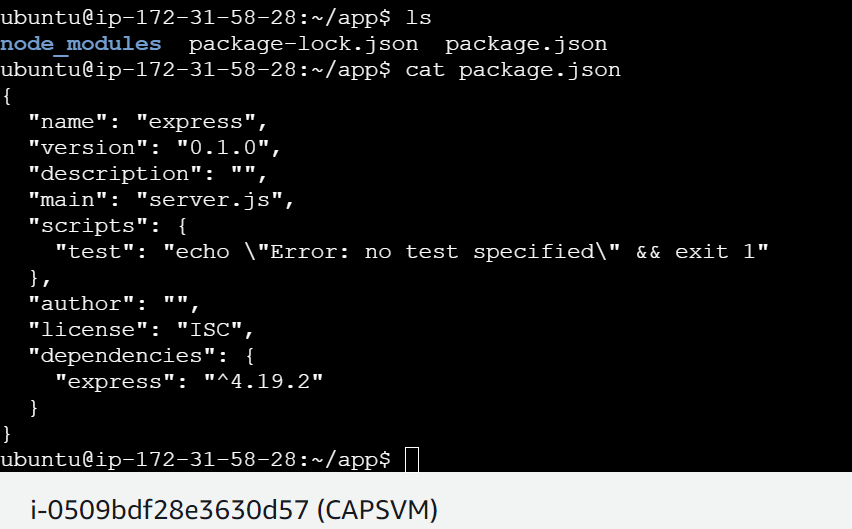


6. Then give yes



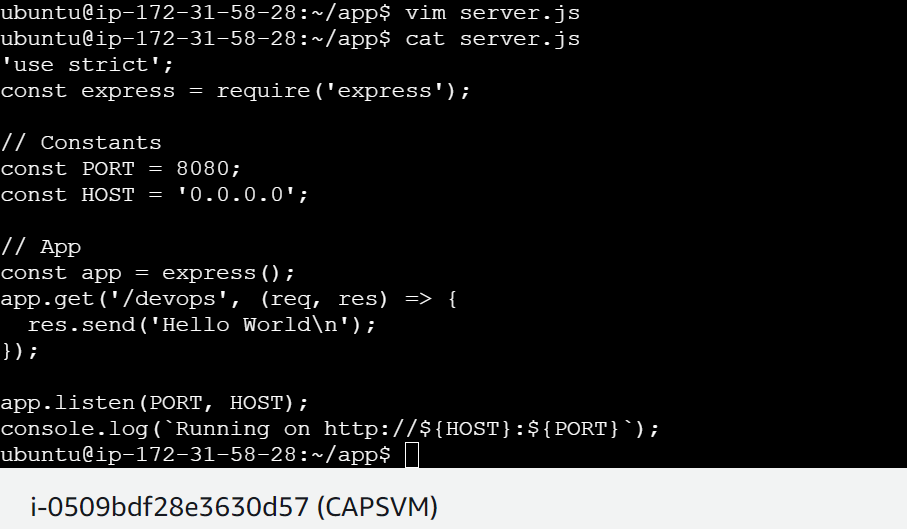
7. Create a Simple REST API: Install Express.js: **npm install express**

****

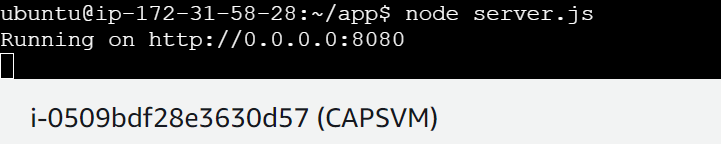
****

8. Now create a Server.js file: **vim server.js** inside app directory

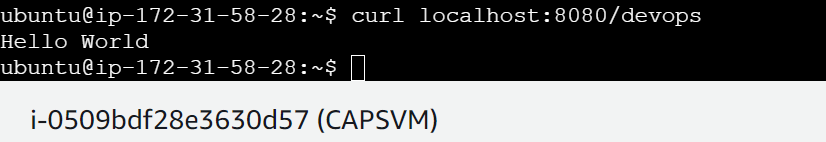
We will write a simple node.js application as below Screenshot to print “Hello World”



9.Then run node.js application: **node server.js**



10.You will get output as Hello world to check that use command **curl localhost:8080/devops**

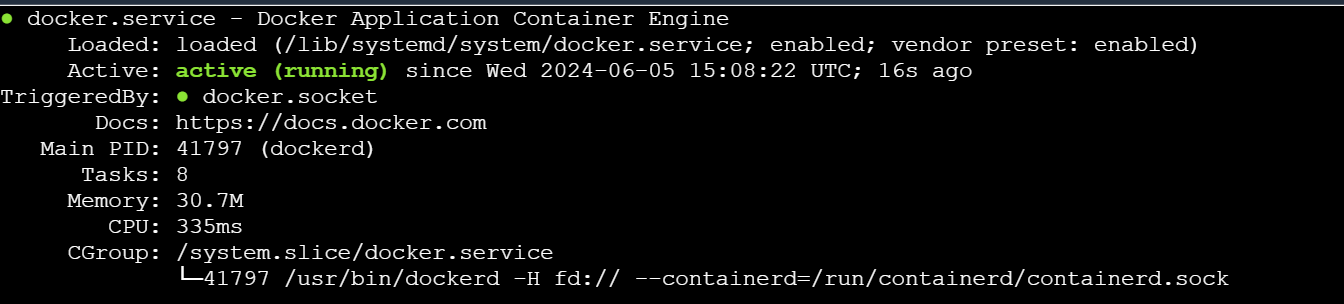


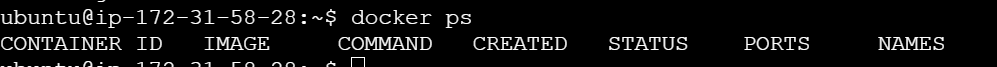
**Step2 : Installing Docker, create file and image**

**Reference: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04**

Installed Docker on the EC2 machine using below commands.

* sudo apt update
* sudo apt install apt-transport-https ca-certificates curl software-properties-common
* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
* sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
* apt-cache policy docker-ce
* sudo apt install docker-ce
* sudo usermod -aG docker ${USER}

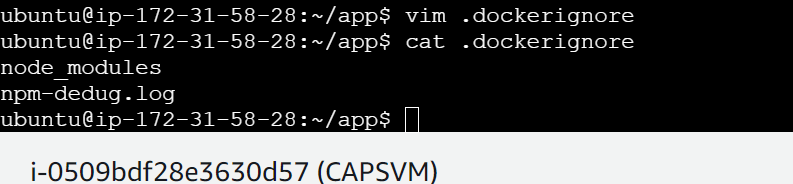


****

1.Under app Directory create docker file: **vim dockerfile**

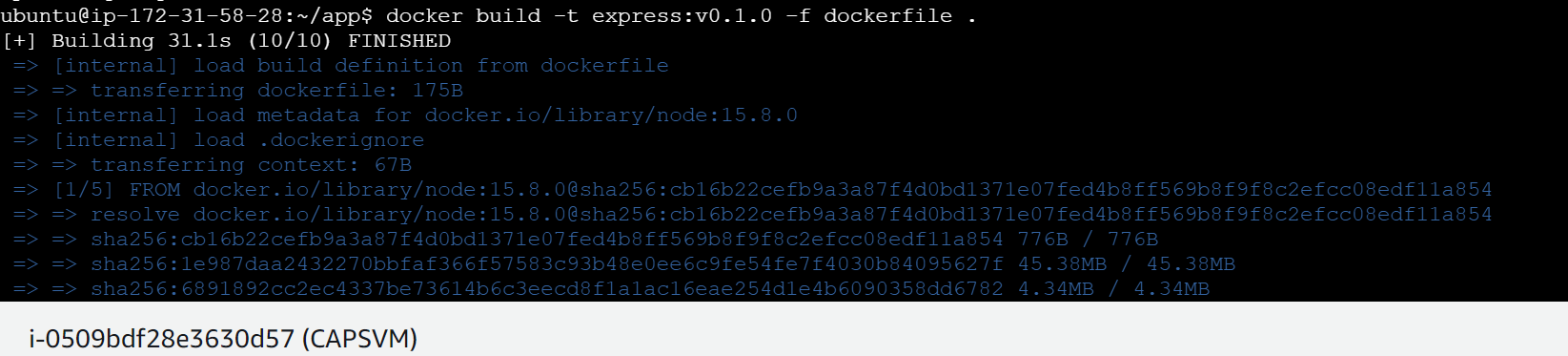
****

2.Create .dockerignore file



3.Building a image using below command:

**docker build -t express:v0.1.0 -f dockerfile .**

****

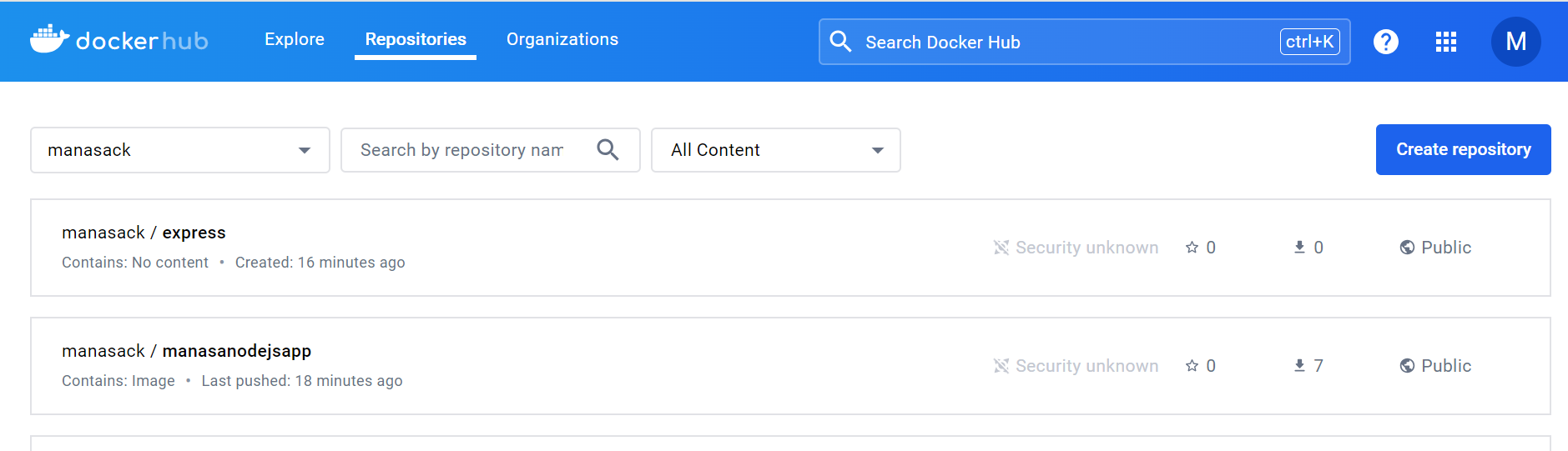
**Docker image will be created.**

Graphical user interface, text

Description automatically generated

4.Pushing the docker image to docker hub

First, create an account in dockerhub and create repository in dockerhub



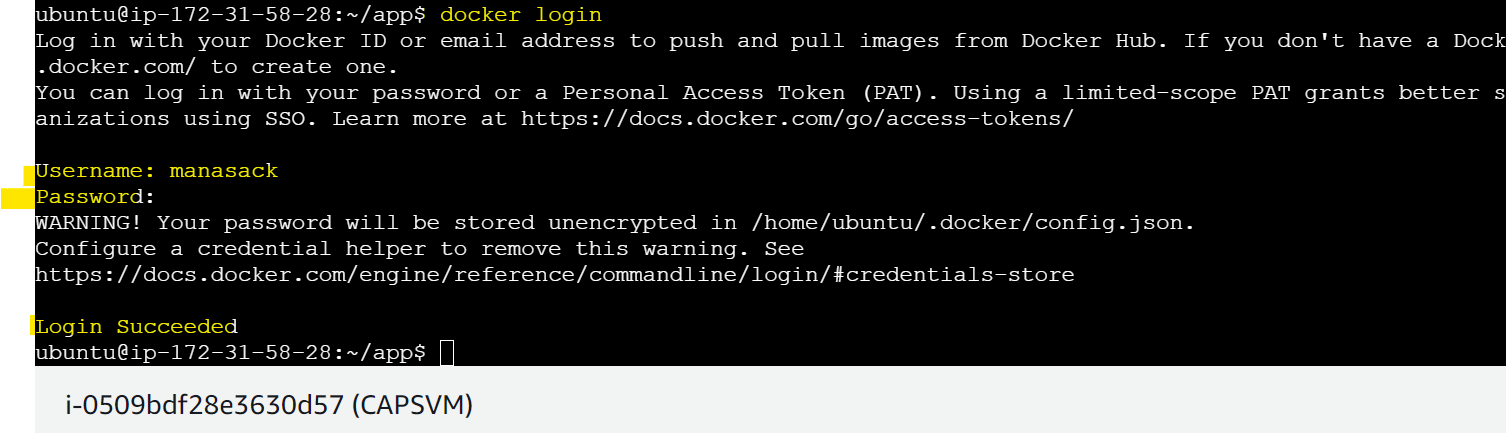
Type docker images to see the docker images that we have created.

Graphical user interface, text

Description automatically generated

Here we will push the “express” image to dockerhub, Before doing that we need to tag the image using below commands.

**docker tag 56c3d10a57a6 manasack/manasanodejsapp:v3**

****

Then push the image using below command:

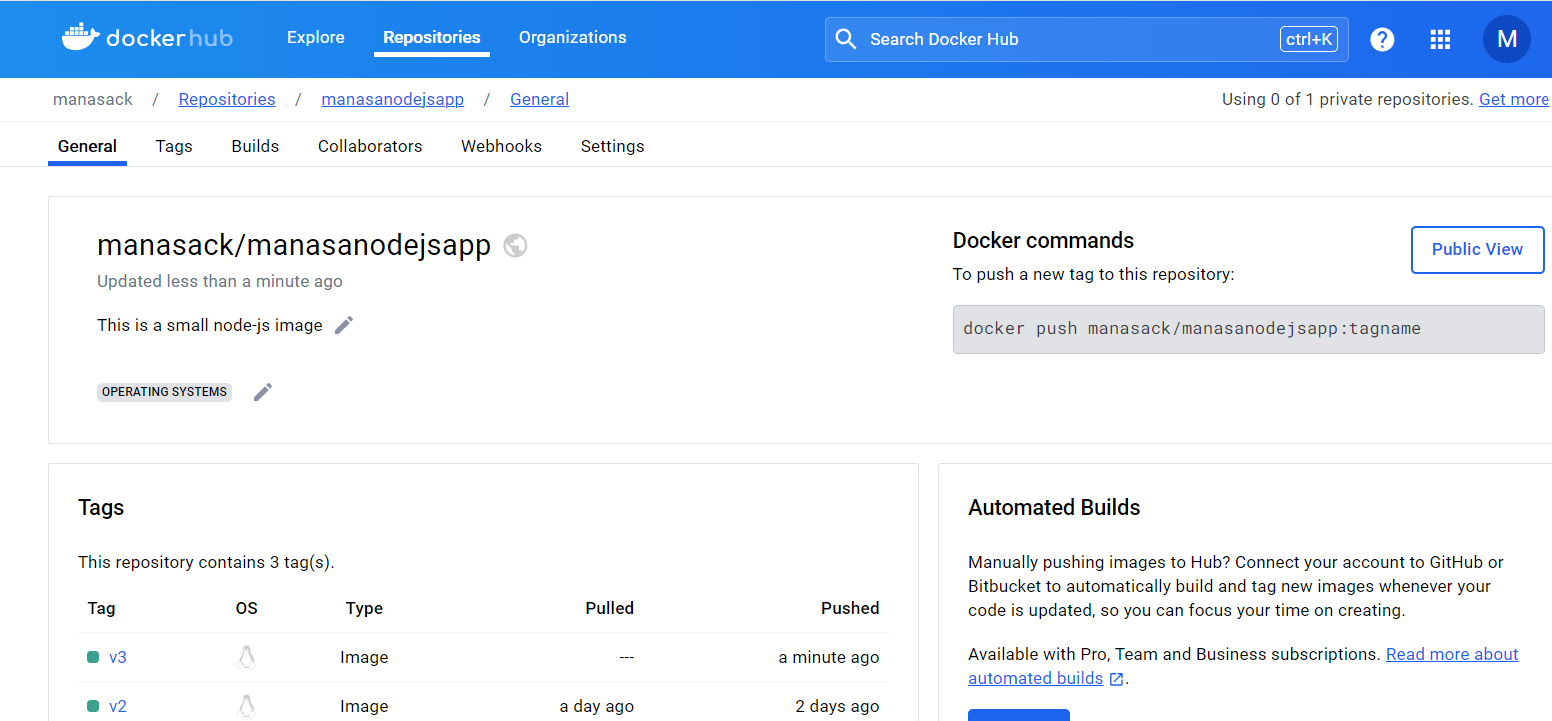
**docker push manasack/manasanodejsapp:v3**

you will be able to see below status as pushed:

Text

Description automatically generated

We will be able to see the image in dockerhub with the tag v3.



**Step3: Deploying the image to minikube and Kubernetes**

We need to install kubectl

Minikube has been executed on different server

Kubernetes has been executed on different server

But we need to install on both the servers.

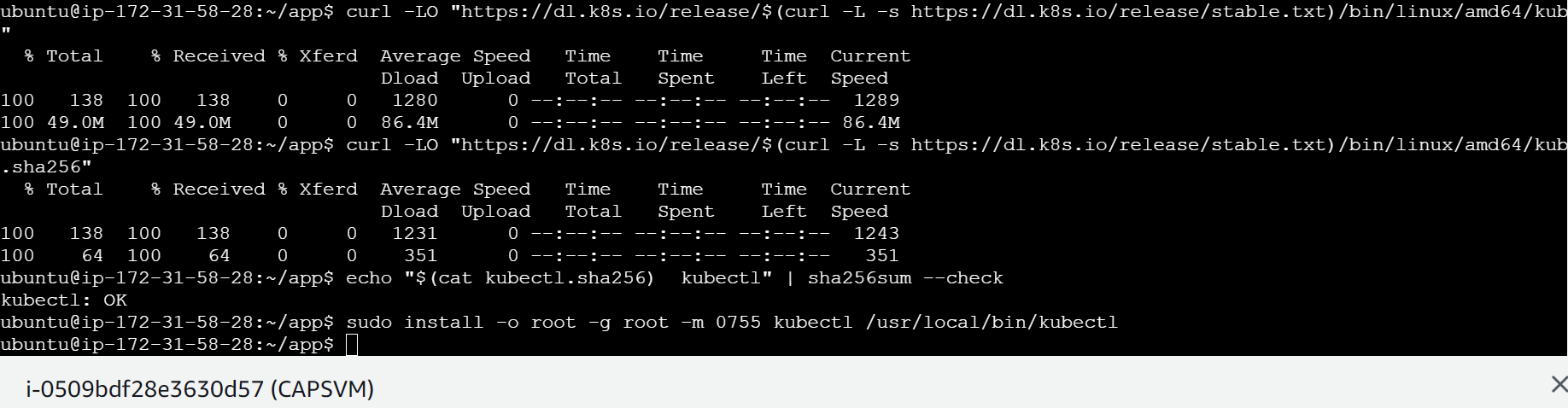
**Installing kubectl**

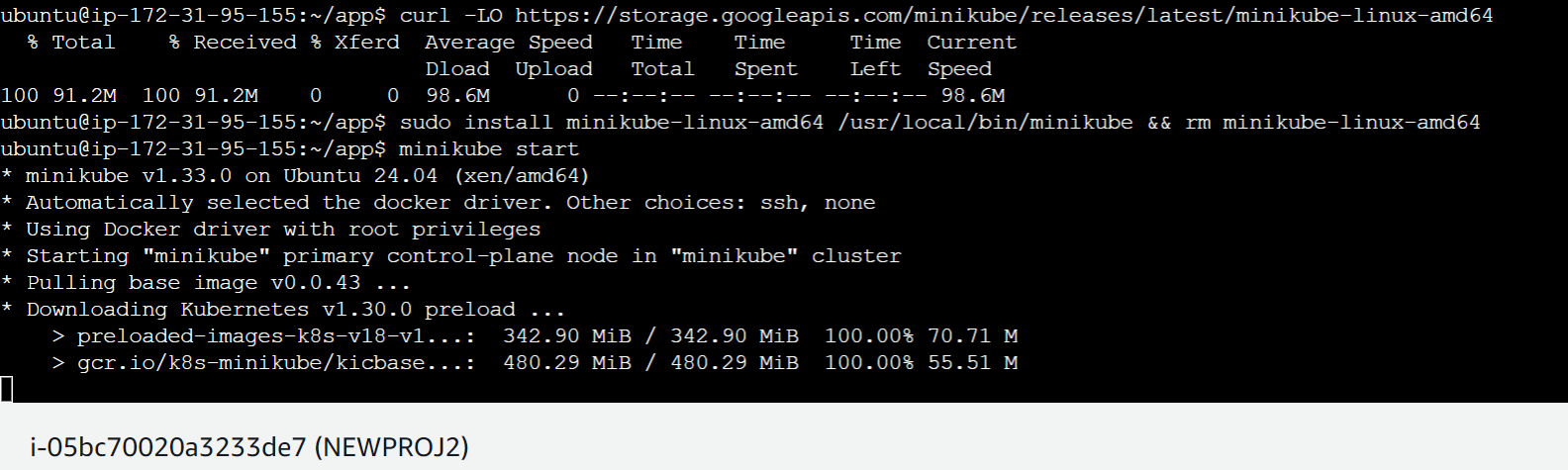
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

echo "$(cat kubectl.sha256) kubectl" | sha256sum --check

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl





Creating express.yaml with Deployment and services configuration in it k8s directory on EC2 machine

creating a directory----------**mkdir k8s**

creating yaml file under k8s directory--------------**vim express.yaml**

Execute the express.yaml using below command

**kubectl apply -f k8s/express.yaml**

we can see namespace,deployment,service configuration is created

Text

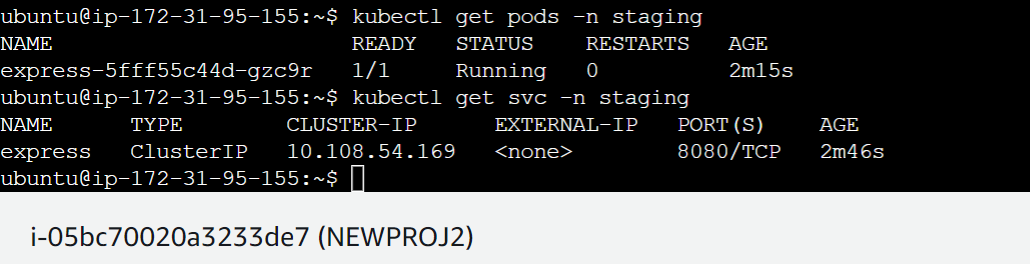
Description automatically generated

We can check pods by using command: (Reference below screenshot)

Kubectl get pods -n staging

We can check services by using command:

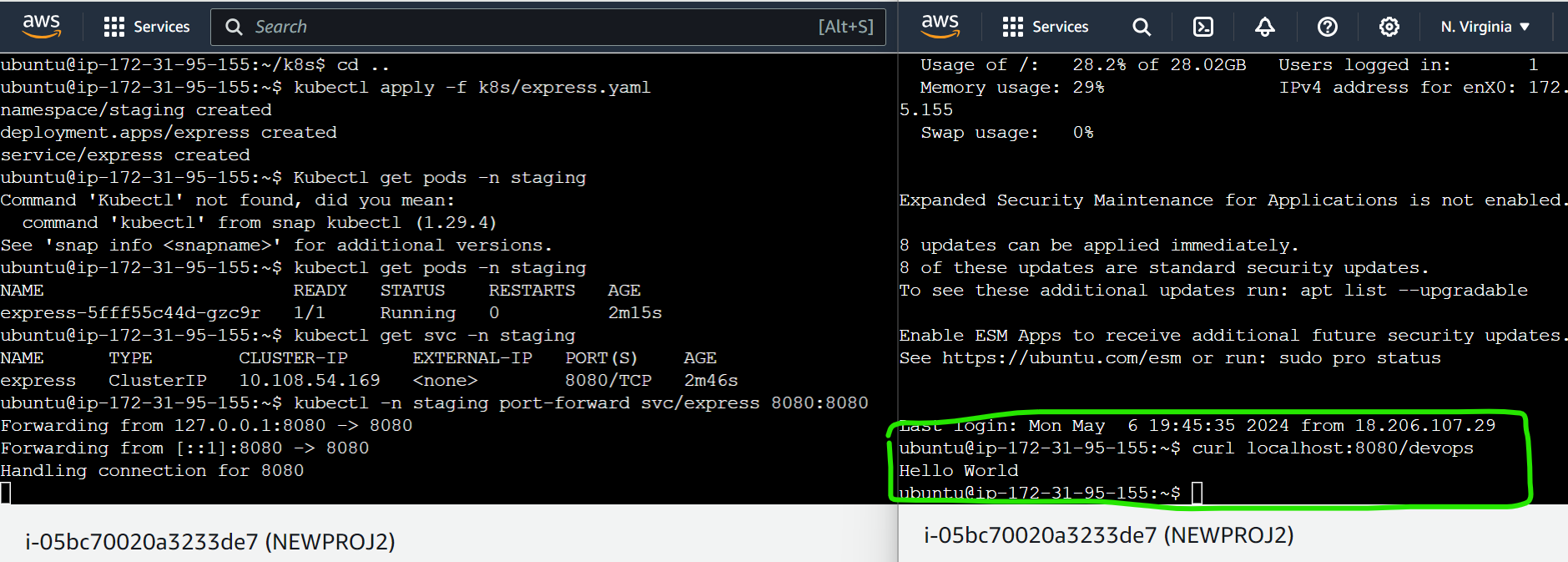
Kubectl get svc -n staging



Then we will check the output whether our nodejs application which displays “Hello World” is working or not.

**kubectl -n staging port-forward svc/express 8080:8080**

We can see **Hello world** output successfully.



Additional Information if implementing on Kubernetes:

**Installing EKSCTL**

ARCH=amd64

PLATFORM=$(uname -s)\_$ARCH

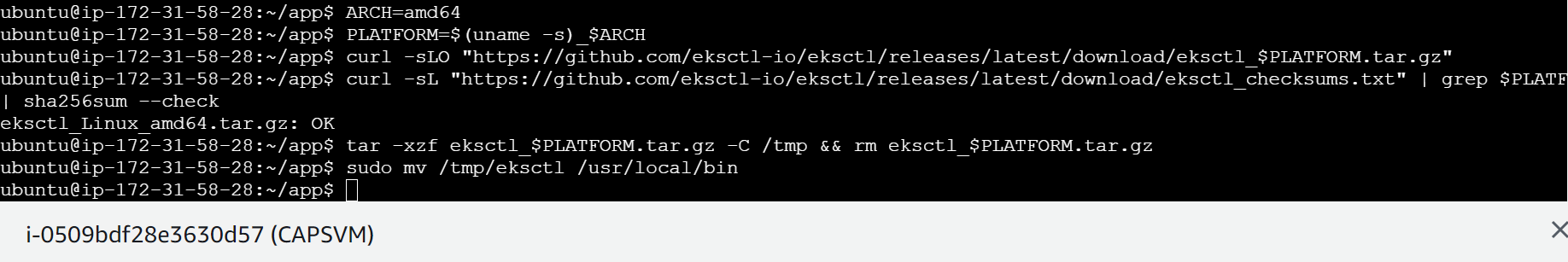
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"

# (Optional) Verify checksum

curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_checksums.txt" | grep $PLATFORM | sha256sum --check

tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz

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



**Installing Python**

sudo apt-get install python3-pip

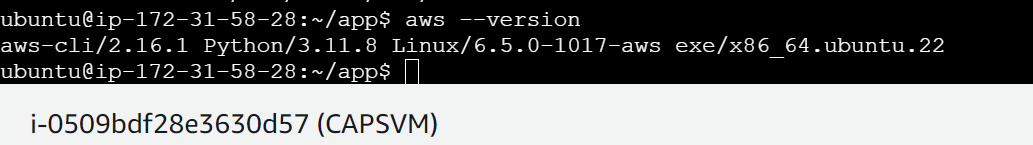
**Installing AWS CLI**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo apt install unzip

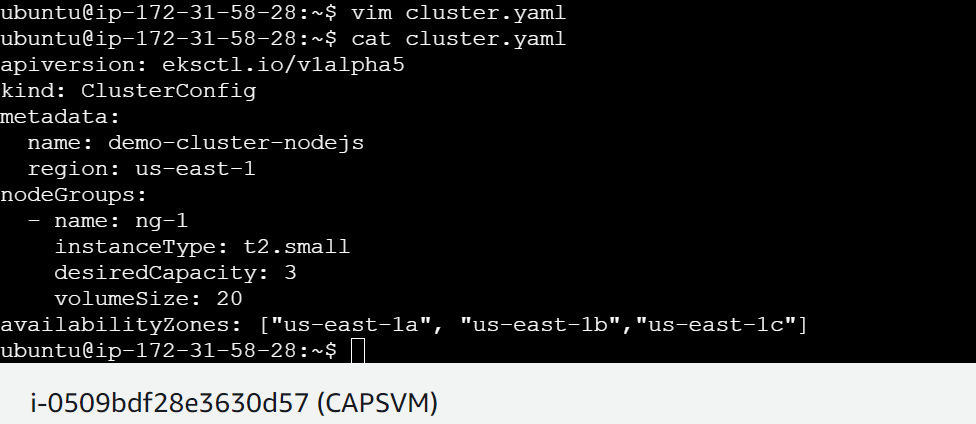
unzip awscliv2.zip

sudo ./aws/install

****

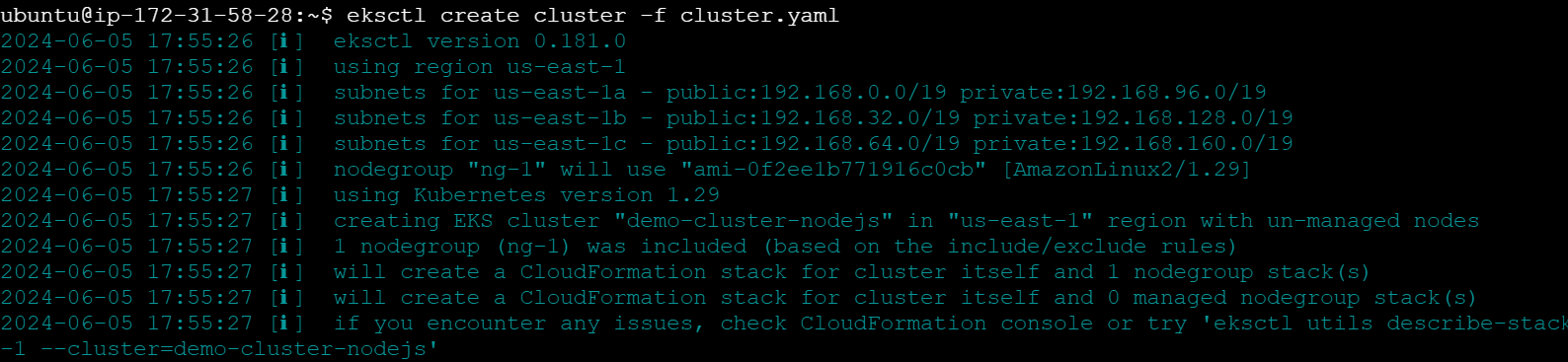
After installing eksctl we need to create Kubernetes cluster using below command:

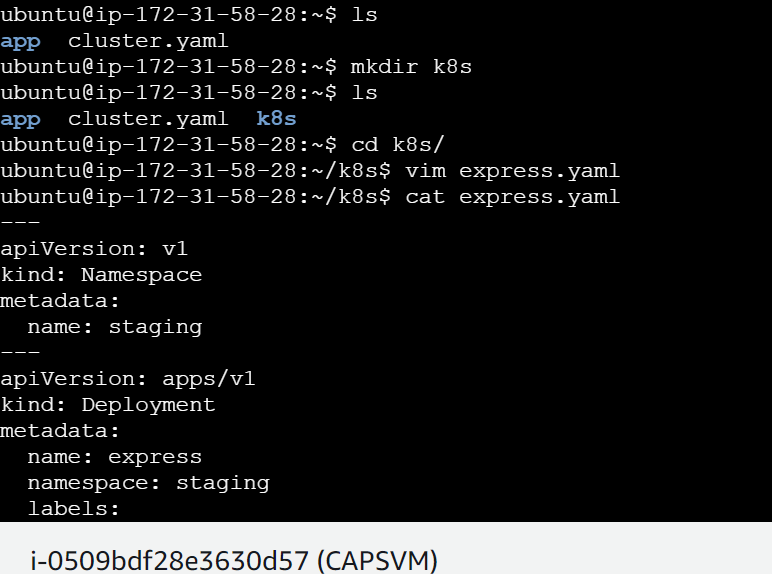
We need to create a file **vim cluster.yaml**

****

**Before creating cluster we need to attach IAM role to EC2 machine with administrator access.**

**Eksctl create cluster -f cluster.yaml**

****

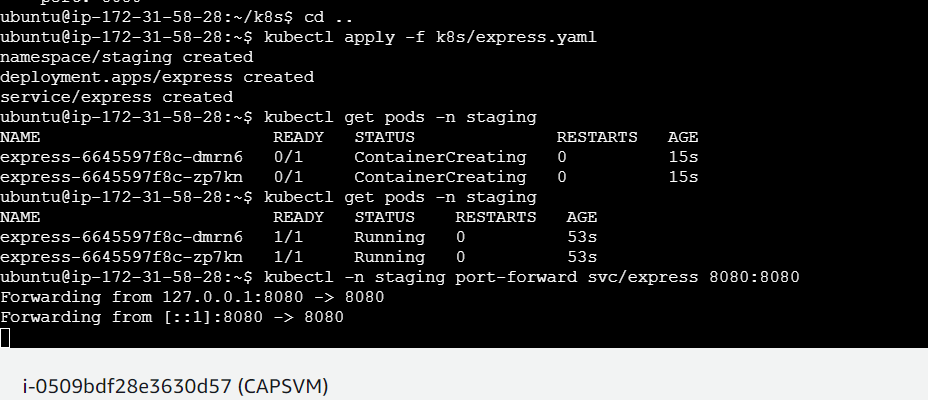
****

**Text

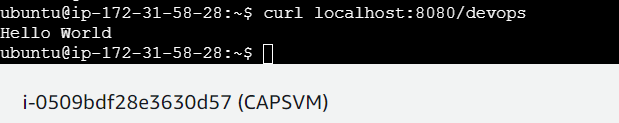
Description automatically generated**

**Text

Description automatically generated**

****

**We have implemented the same on Kubernetes instead of minikube and we are getting the output “Hello world”**



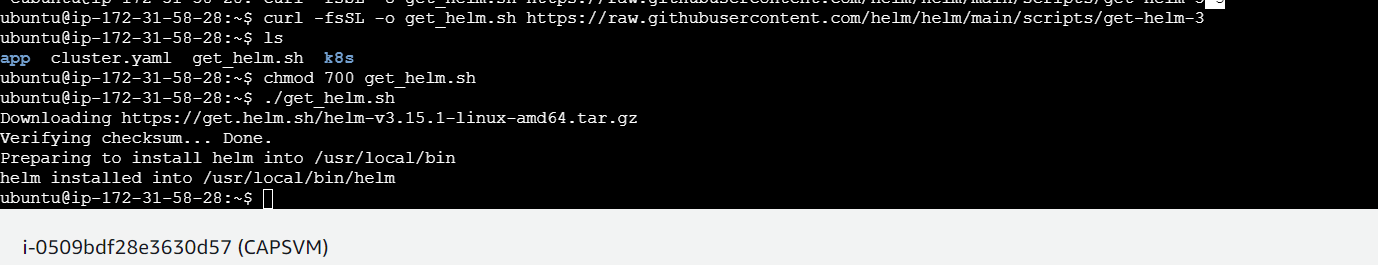
**Step 4: Integrating Prometheus and Grafana using helm-charts.**

**Installing helm:**

curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

chmod 700 get\_helm.sh

./get\_helm.sh

****

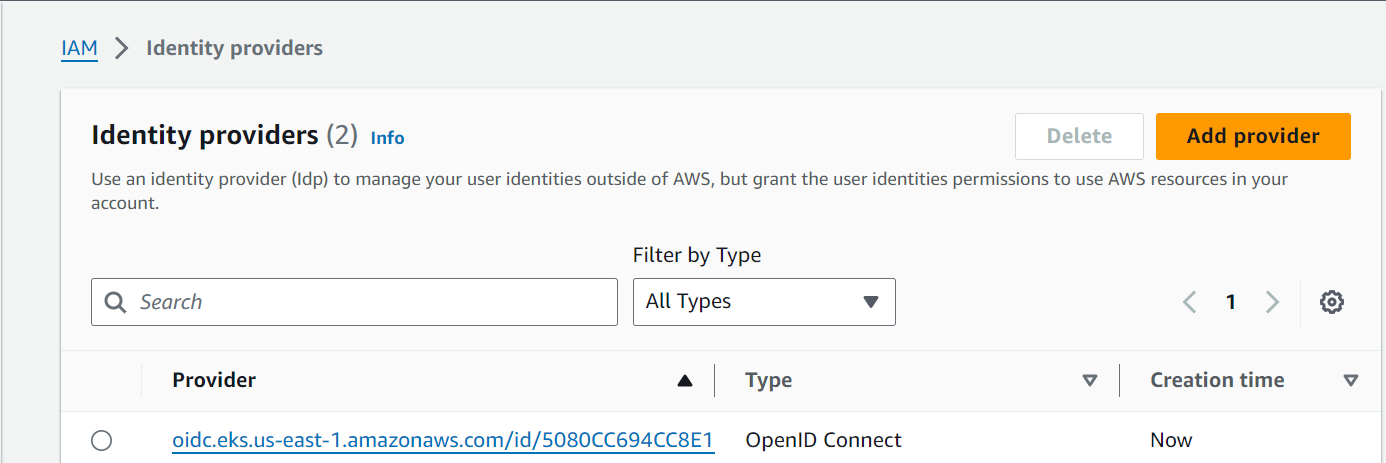
**Additional information if deploying in Kubernetes:**

**Before installing Prometheus and Grafana we need to create CSI driver for volume purpose**

1.Go to IAM role and select Identity providers >>>> then add provider>>OpenID connect

2.Then go to eks services select the Kubernetes cluster that we have created and copy the openID connect URL from there and paste it in IAM role. Then click on get thumbprint.

3.Then add “ sts.amazonaws.com” as audience>>>>>>add provider



4.Go to roles>>>create role>>>select web identity and then provide identity provider and audience that we just created and click next

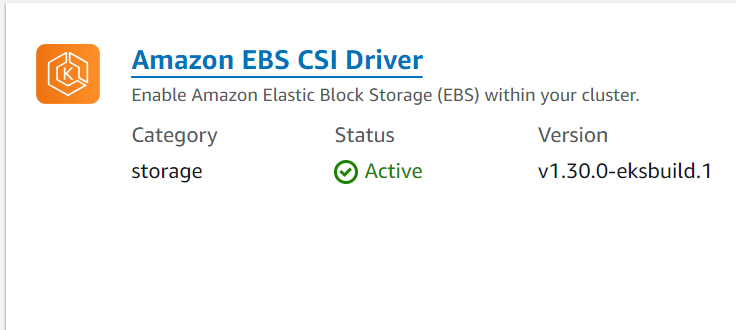
5. select the policy “AmazonEBSCSIDriverPolicy” and click next and add role name “AmazonEKS\_EBS\_CSI\_DriverRole” and create role.

6.Then go to newly created role and navigate to trust relationships and then edit trust policy. As mentioned in below link and update policy.

<https://docs.aws.amazon.com/eks/latest/userguide/csi-iam-role.html>

7. Then go to eks cluster and click on add-ons>>>>>Get more add-ons search for CSI driver and select amamzon EBS CSI driver >>Next

Select the role which we created >>>>>click on next and create. We should be getting below active state for CSI driver.



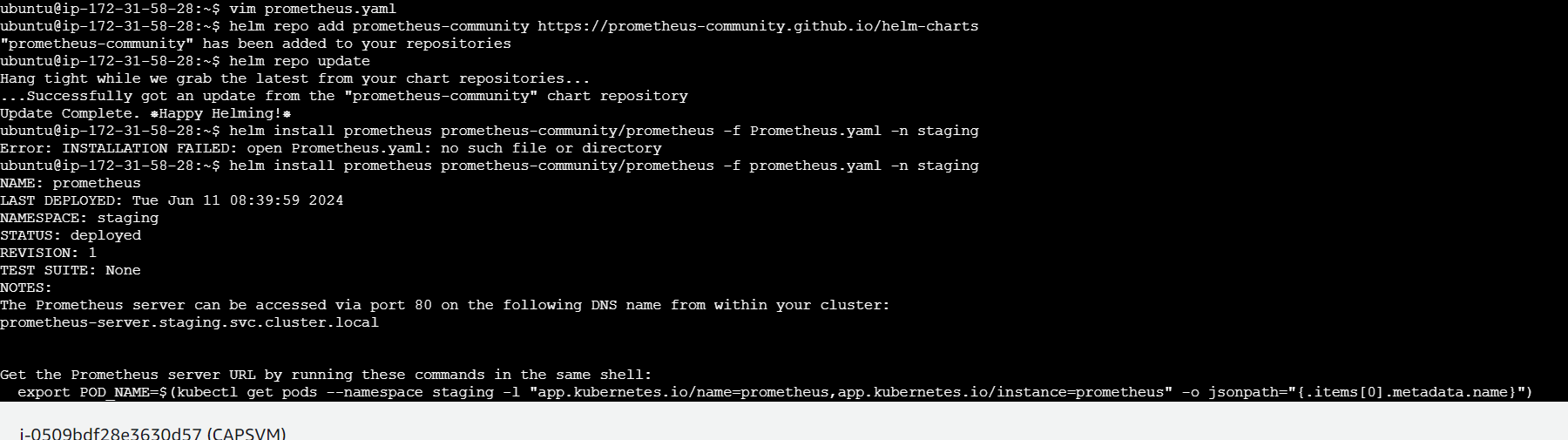
**PROMETHEUS:**

Execute the command:

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

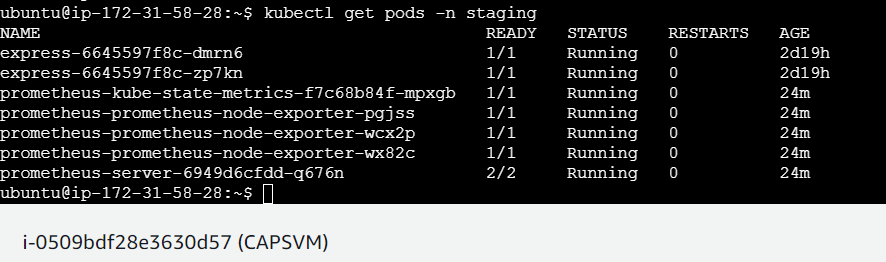
helm repo update

helm install prometheus prometheus-community/Prometheus -f Prometheus.yaml -n staging



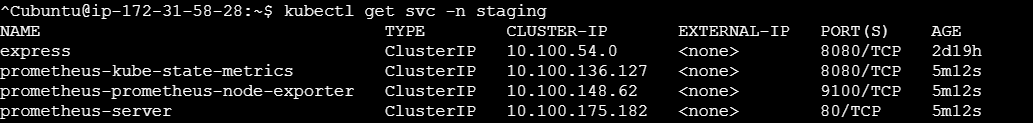
To see Prometheus pods

Kubectl get pods -n staging



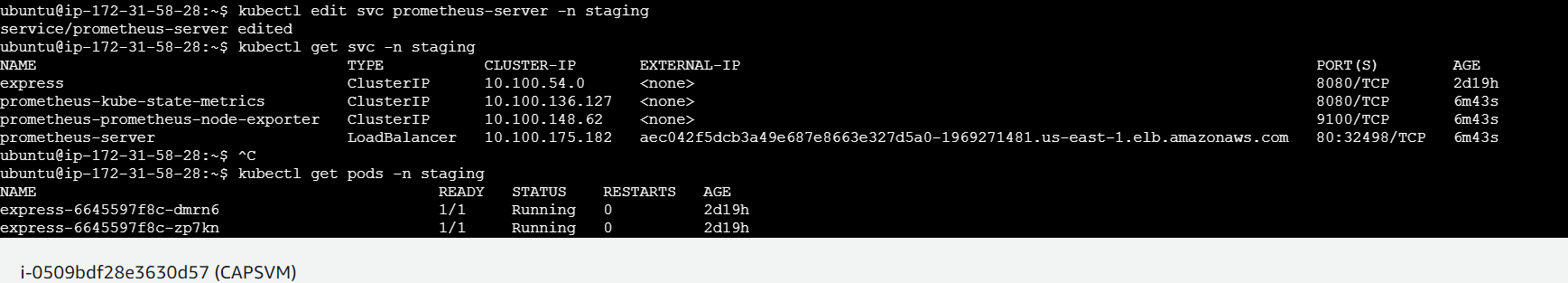
Let’s also see service using below command.

Kubectl get svc -n staging



We could see Prometheus server running and Kubernetes metrics also running.

But we could see Prometheus server service type ClusterIP so let’s change it to LoadBalancer to view the Prometheus page.

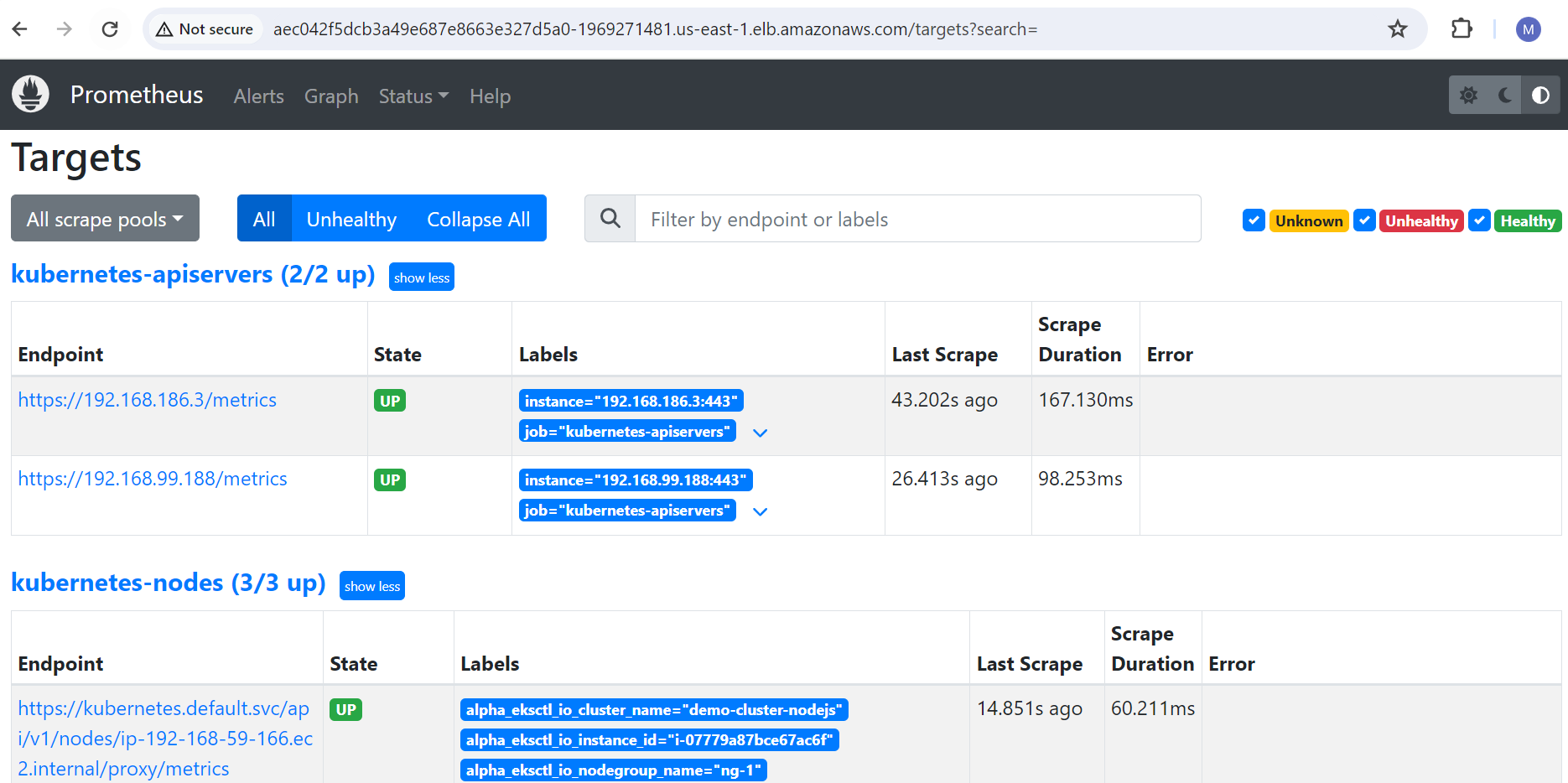


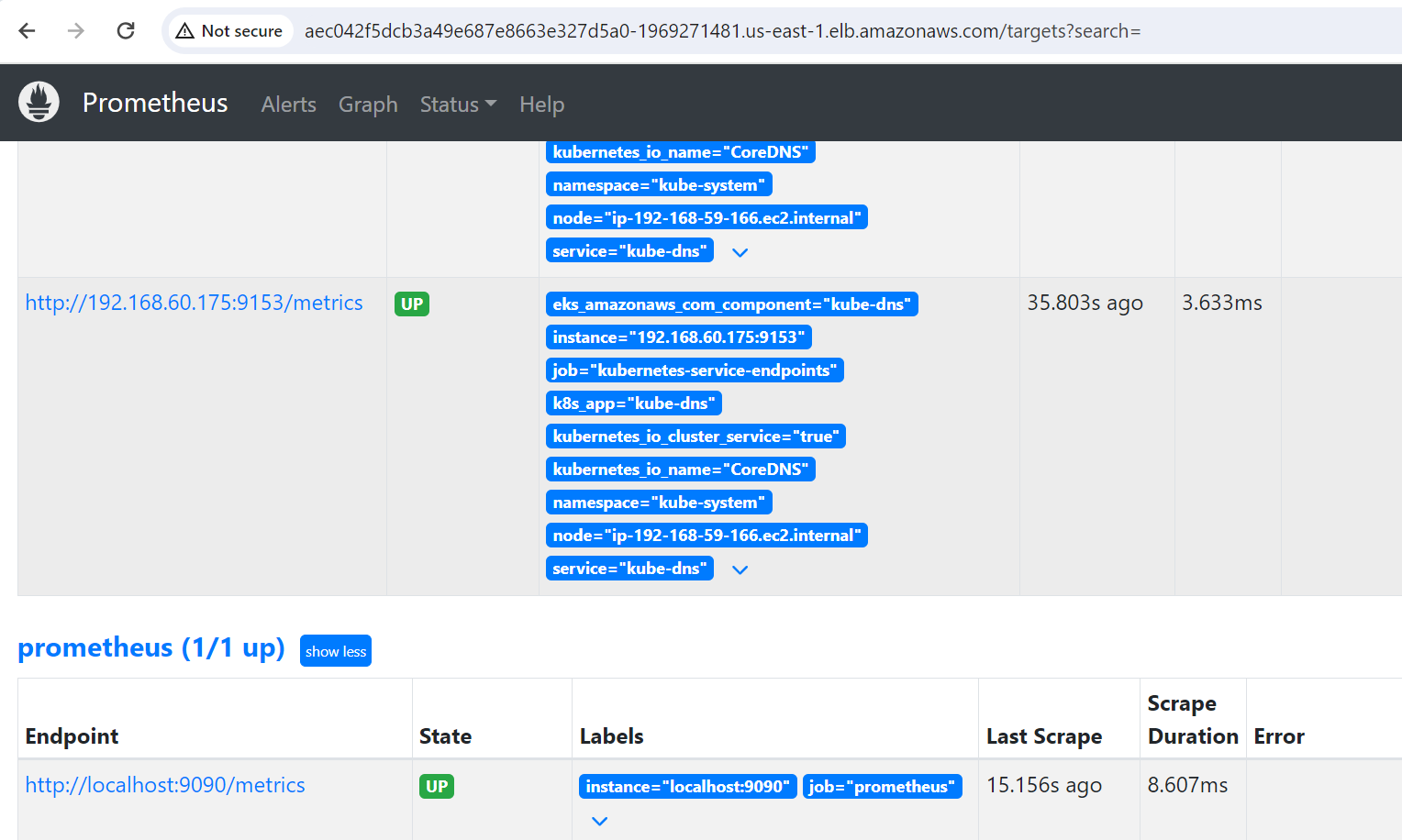
Let’s run command: kubectl edit svc prometheus-server -n staging

The above command will help us to edit service, after executing this command file will open up so there, we can edit type as type: LoadBalancer and save the file.

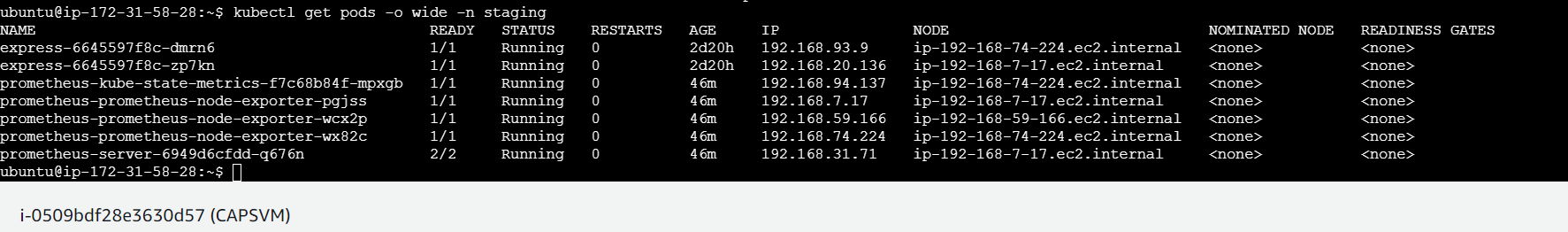
Again, if we check “kubectl get svc -n staging”, we will be able to see external IP where we can access that in the browser to view Prometheus page.

Ex: aec042f5dcb3a49e687e8663e327d5a0-1969271481.us-east-1.elb.amazonaws.com





Kubectl get pods -o wide -n staging to view complete details about nodes

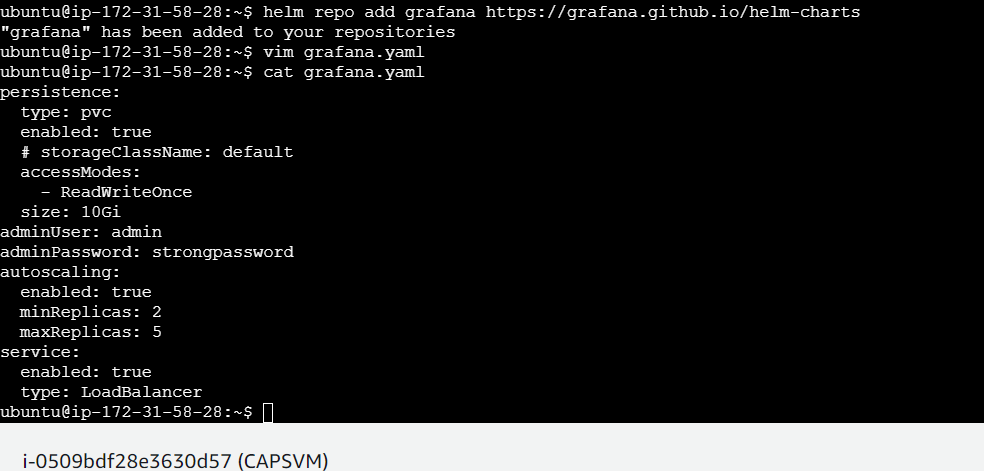


**Grafana using helm chart:**

Execute the command:

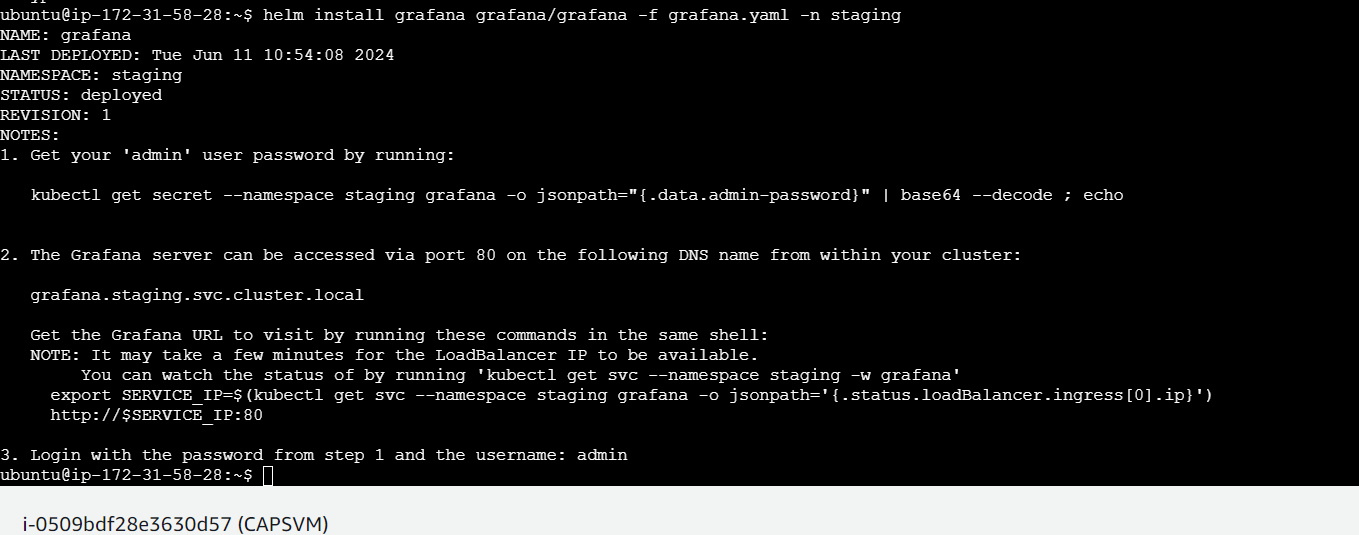
helm repo add grafana <https://grafana.github.io/helm-charts>

Then create grafana.yaml as below.



Then execute install command:

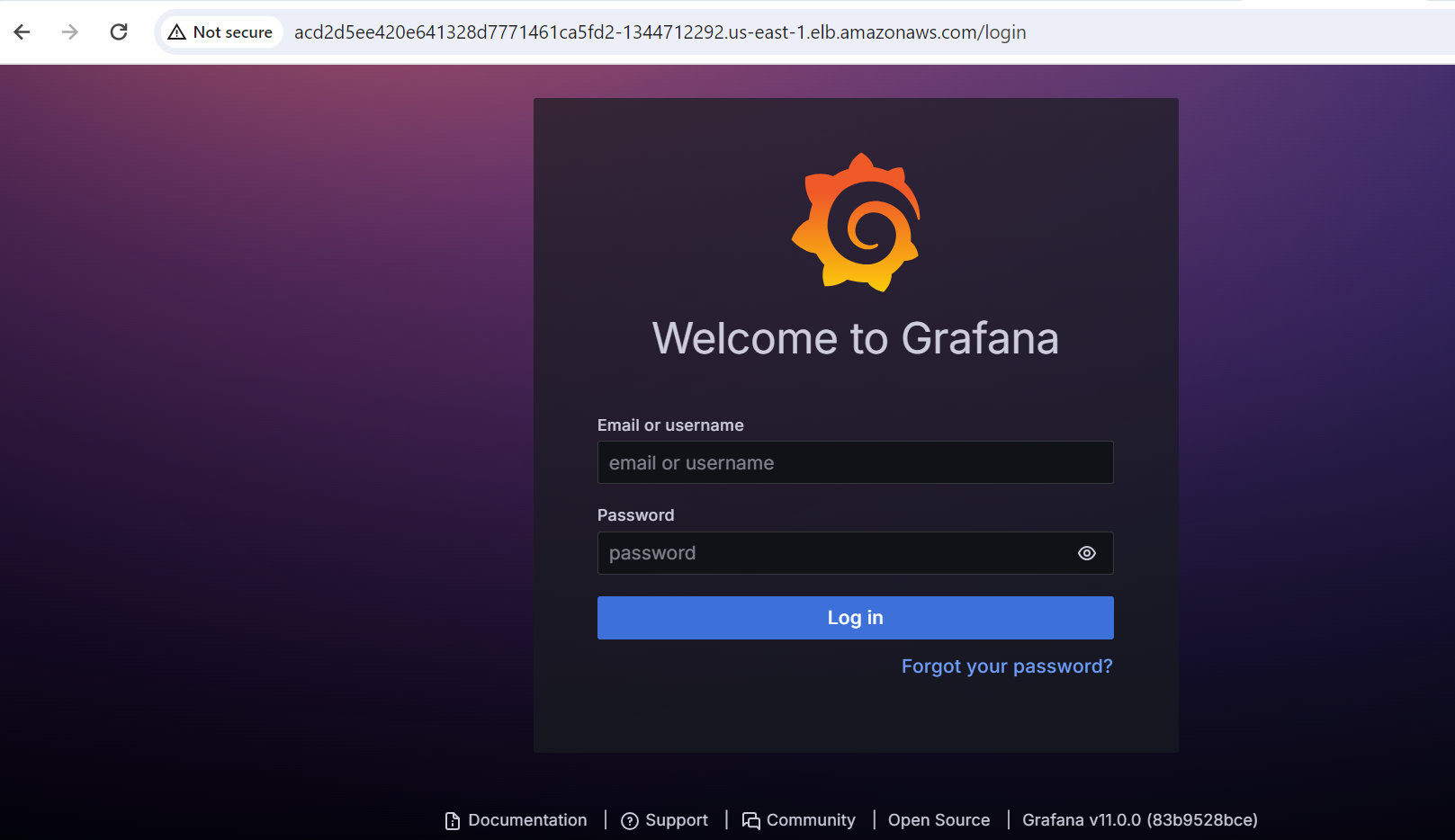
helm install grafana grafana/grafana -f grafana.yaml -n staging



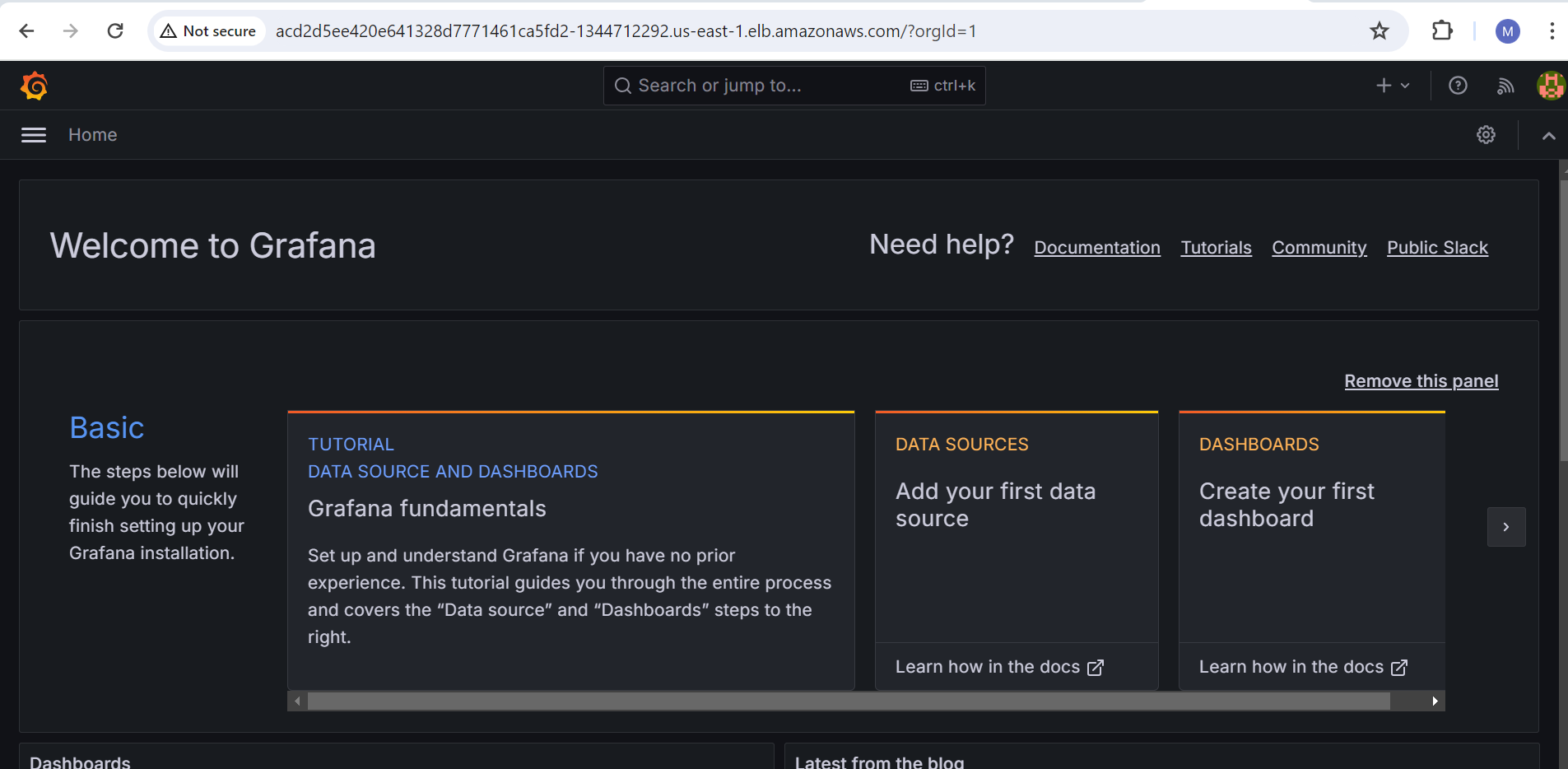


Let’s check if Grafana is accessible through loadbalancer:

acd2d5ee420e641328d7771461ca5fd2-1344712292.us-east-1.elb.amazonaws.com

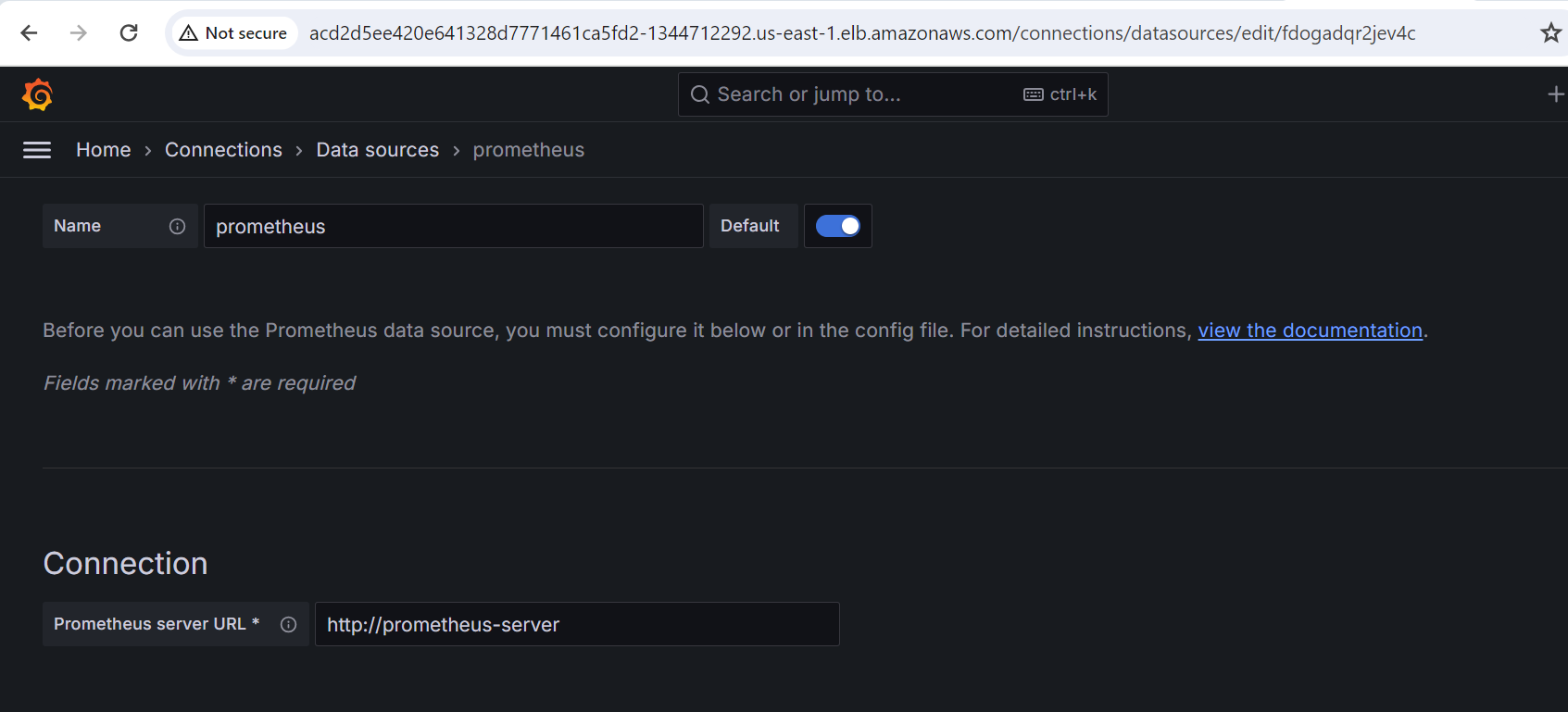


Give the specified username and password as mentioned in the code and we will be able to login successfully.

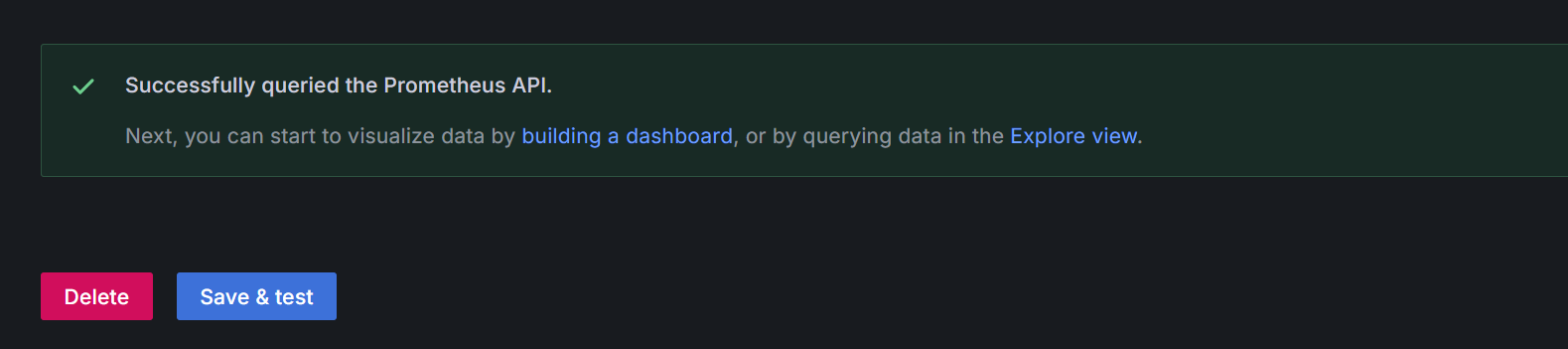


Integrating Grafana with Prometheus:

Then go to Data sources and select Prometheus and configure it with using Prometheus server details which we just created.



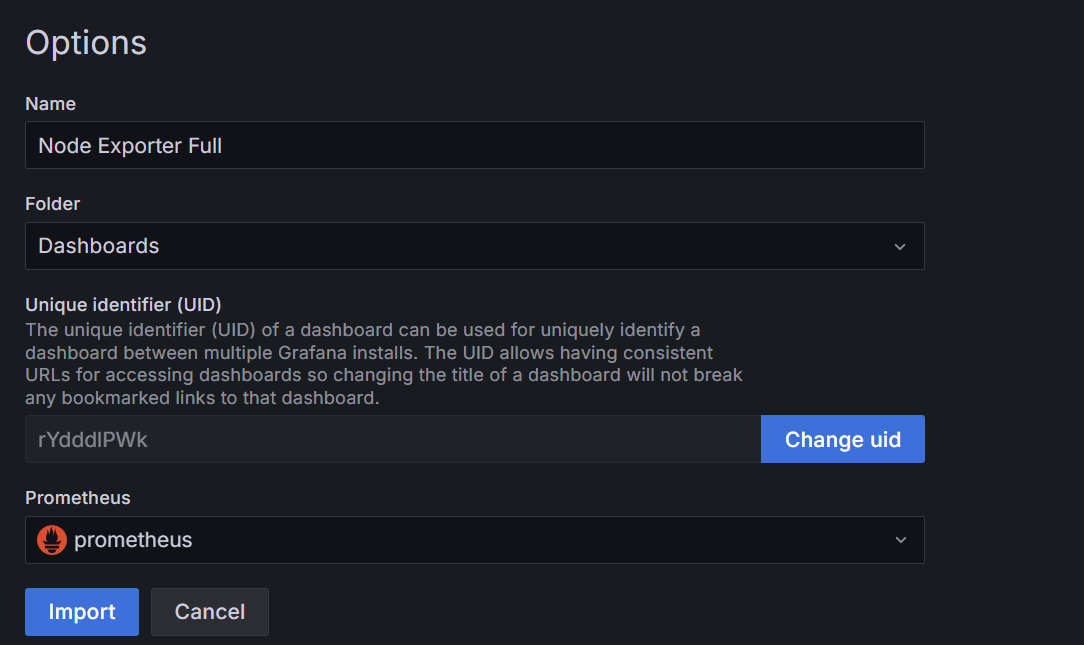
Then click save and test.



Then go to Grafana.com and navigate to dashboards (At the end) where we can fetch code for readymade dashboards >>>> then copy the code

Then come back to our Grafana Dashboards and select import dashboard and paste the code and load it.

Select Prometheus as source data and then click on import.

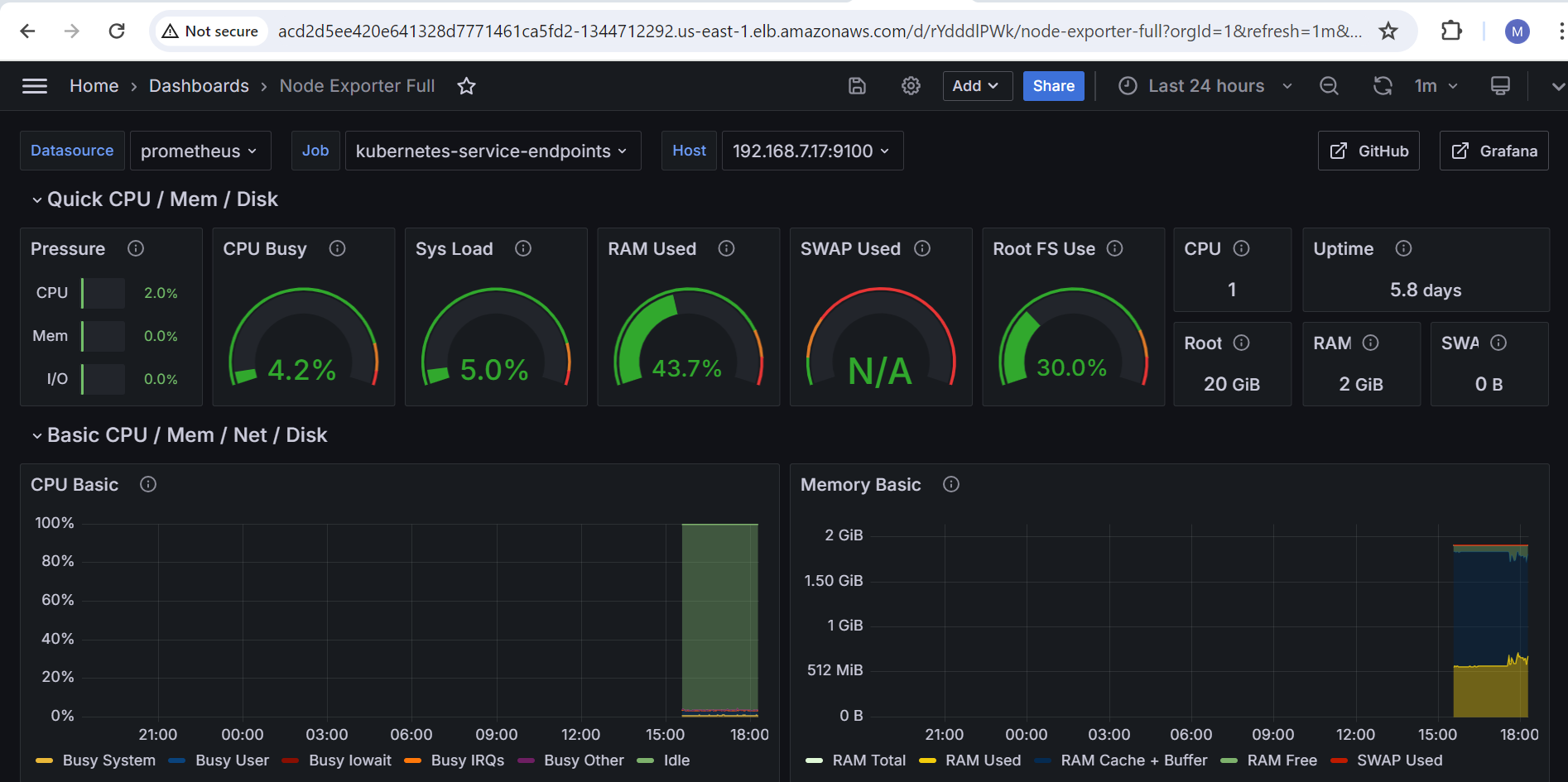


Then we will be able to see the metrics that are fetched from grafana.

Nodejs application is running on 2 node 192.168.74.224 and 192.168.7.17

A picture containing table

Description automatically generated

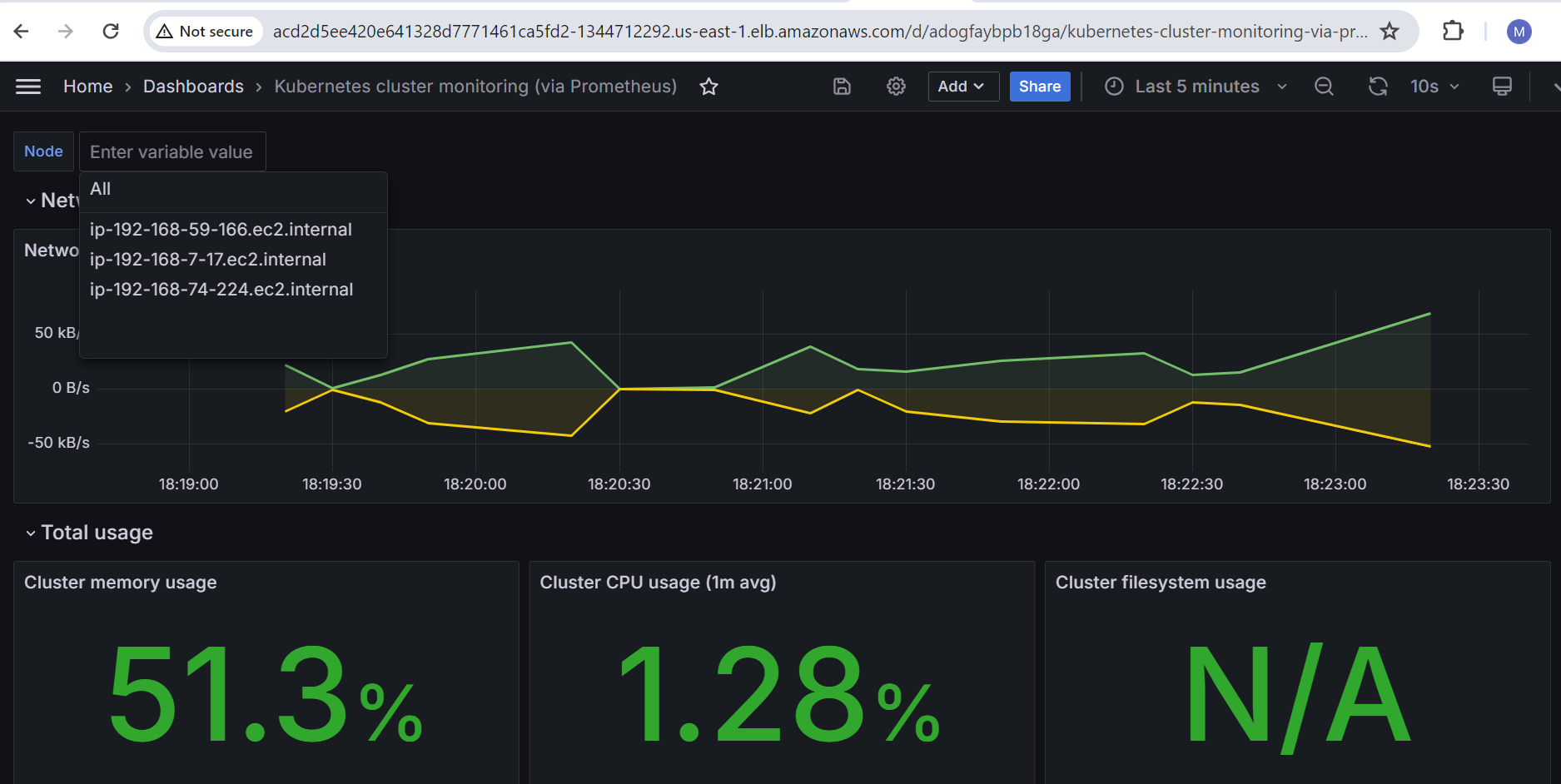


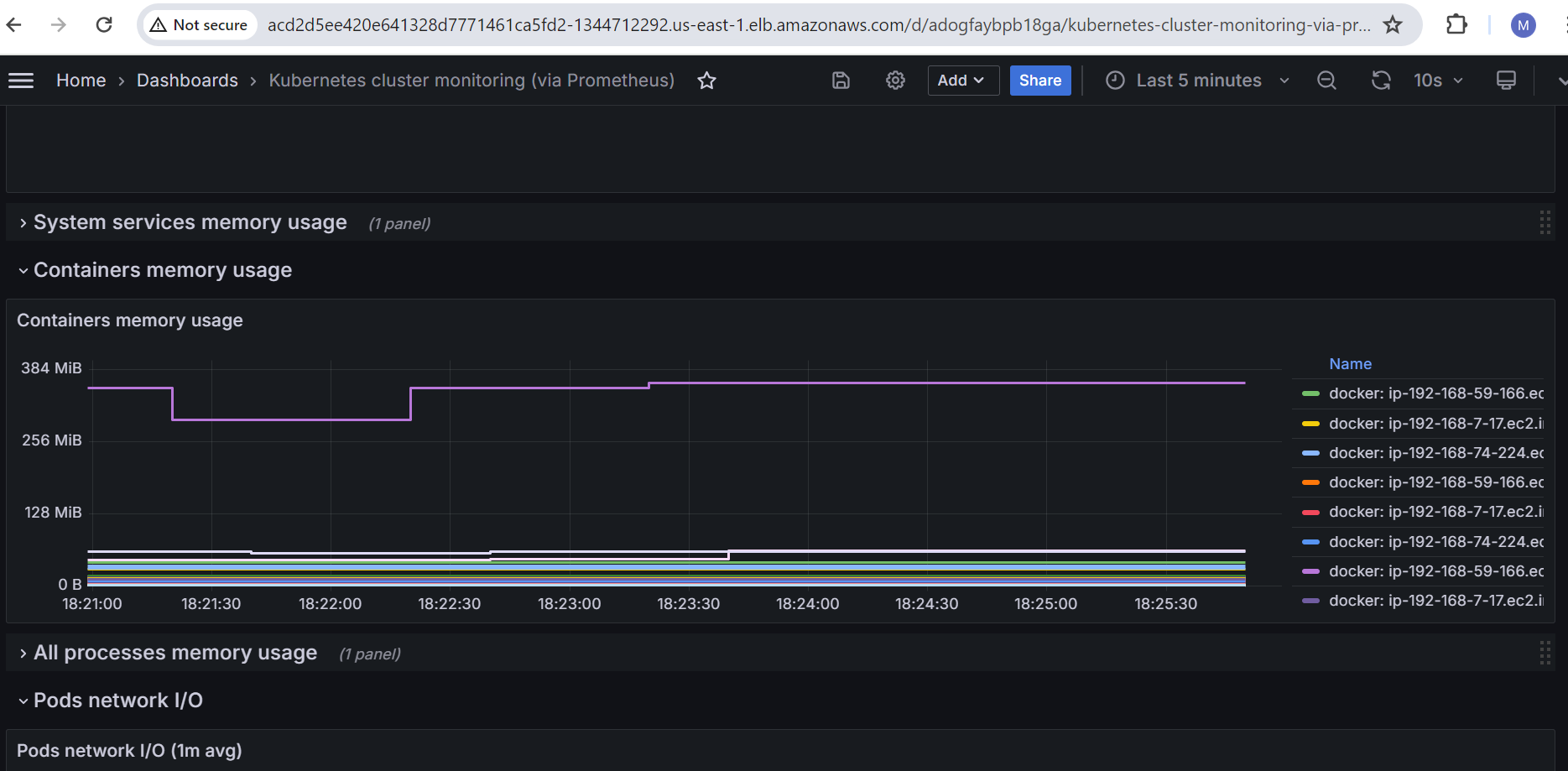
A screenshot of a computer

Description automatically generated with medium confidence

We can also use different dashboards such as Kubernetes cluster monitoring.

Where we can view all nodes metrics in cluster level





A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence