**DevOps module 5 Capstone Project**

Git Repo:

Task 0: Installation of required software

#aws-cli:

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

unzip awscliv2.zip

sudo ./aws/install

aws --version

#terraform:

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

sudo apt-get update && sudo apt-get install terraform -y

terraform -help

#kubectl:

curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl

curl -o kubectl.sha256 https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl.sha256

openssl sha1 -sha256 kubectl

chmod +x ./kubectl

mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc

kubectl version --short --client

#eksctl:

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

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

eksctl version

#helm:

curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -

sudo apt-get install apt-transport-https --yes

echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update

sudo apt-get install helm -y

#ab:

apt-get update

apt-get install apache2-utils -y

ab -V

#Additional AWS-IAM-Authenticator

curl -o aws-iam-authenticator https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/aws-iam-authenticator

curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/aws-iam-authenticator.sha256

openssl sha1 -sha256 aws-iam-authenticator

chmod +x ./aws-iam-authenticator

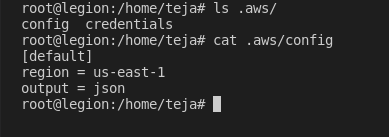
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$PATH:$HOME/bin

echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc

aws-iam-authenticator help

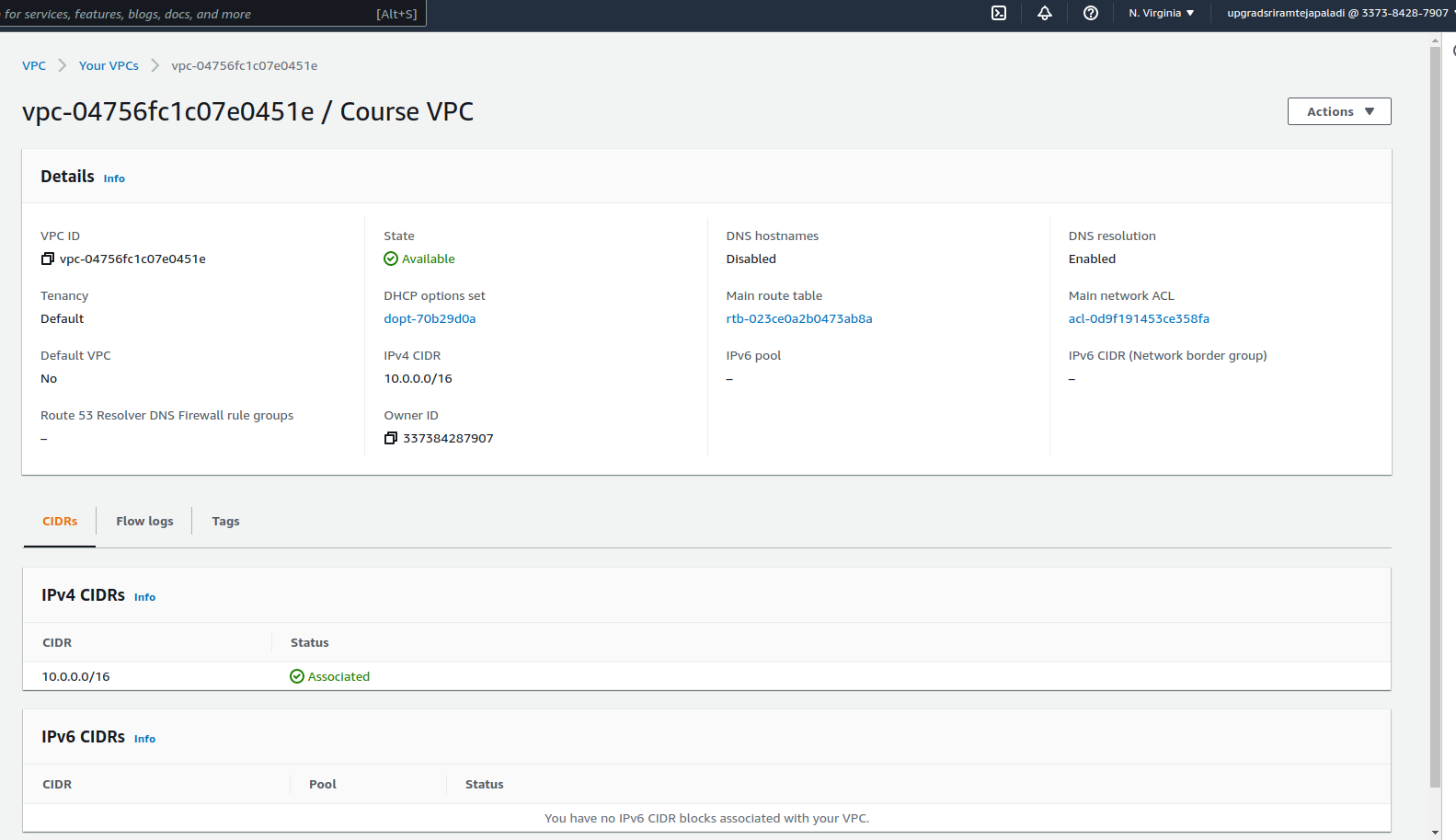
Task 1: Setup EKS Cluster

Before proceeding configure aws using the command “aws configure” and provide the necessary credentials details and verify the same in the .aws folder:

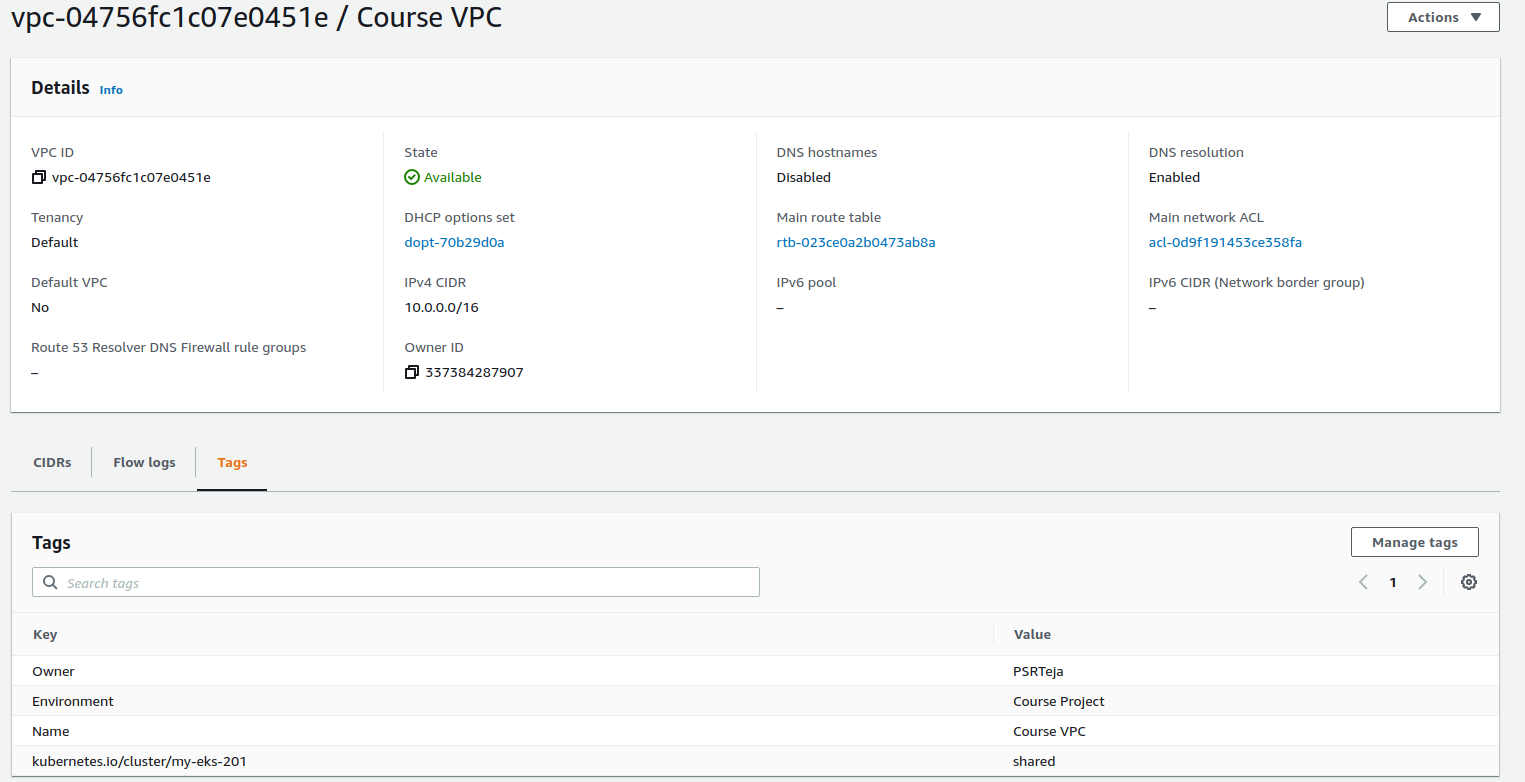


Now use terraform init, plan and apply workflow to create the vpc, s3 bucket, dynamodb, subnets (2 private and 2 public), internet gateway, route tables for the subnets, nat gateway using the terraform .tf files.

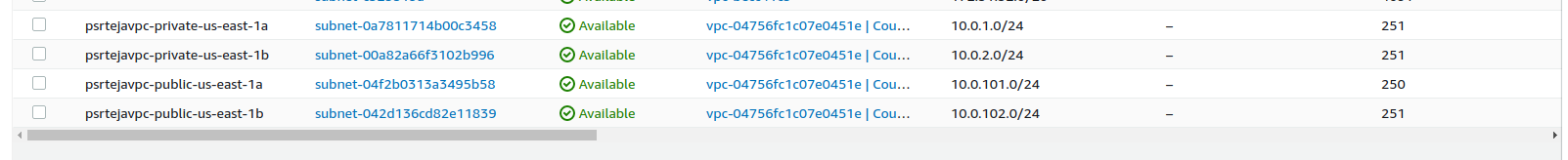
VPC :



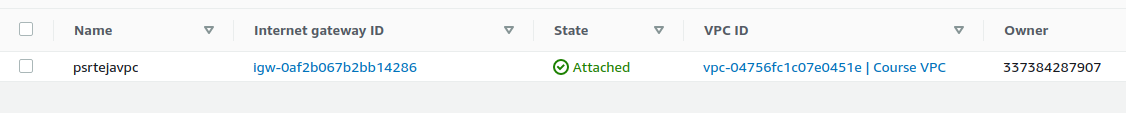
VPC tags:



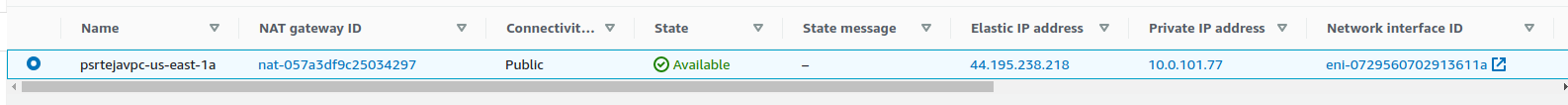
Subnets:



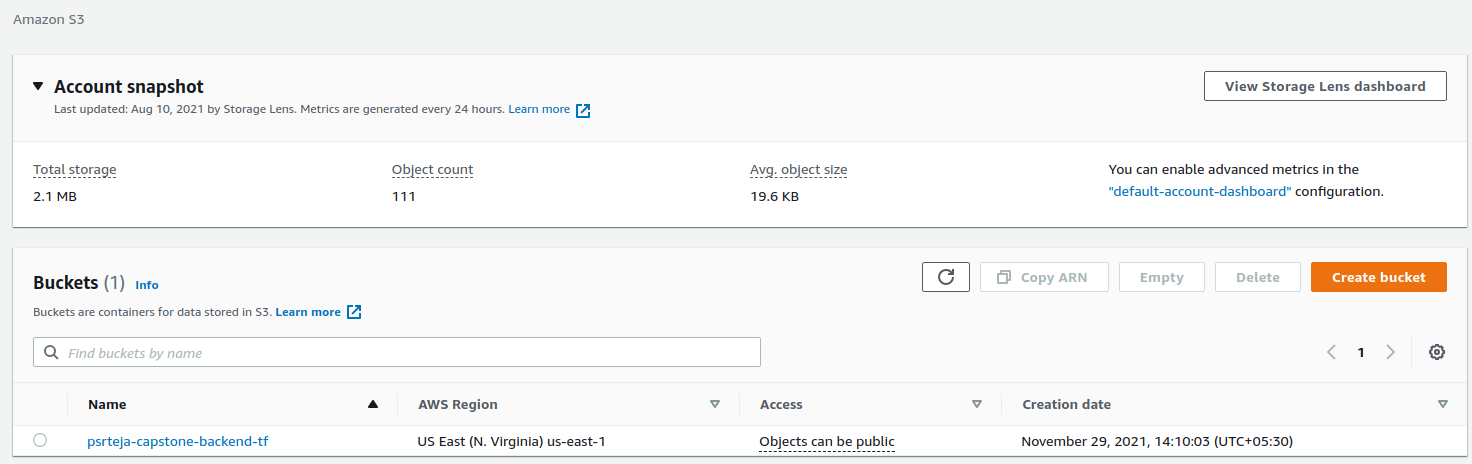
Internet Gateway:



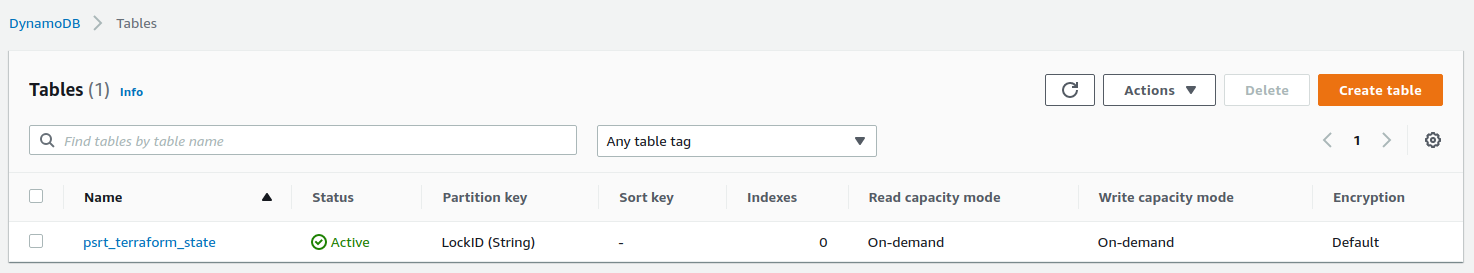
NAT Gateway:



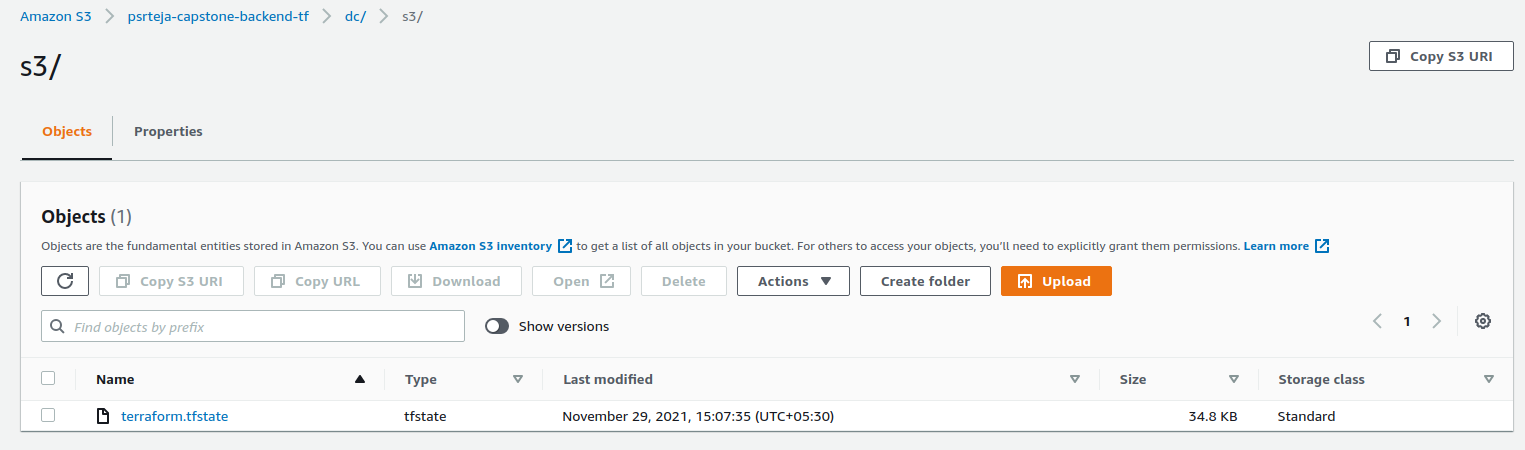
S3 bucket:



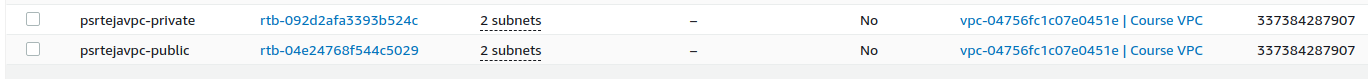
Dynamodb:



tfstate backup in s3 bucket:

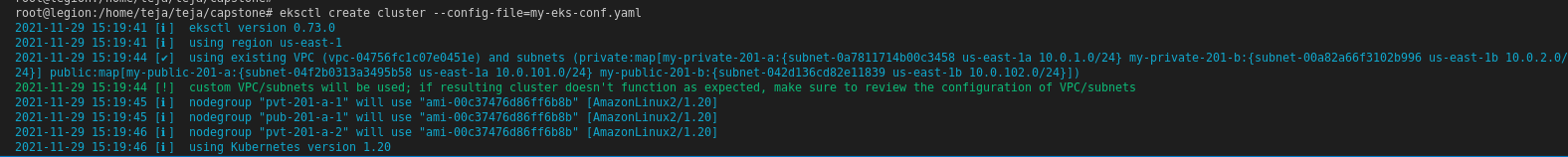


Routing tables:

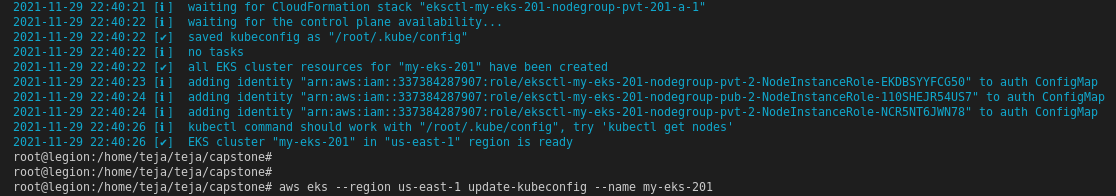


Create EKS Cluster with the help of cluster configuration yaml file.

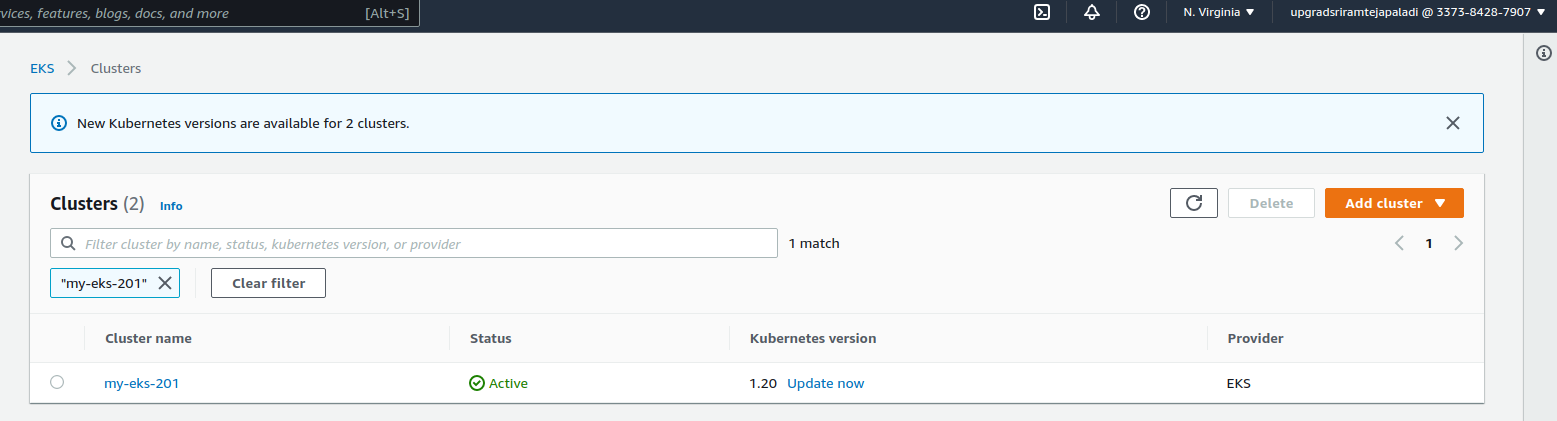
Cluster creation;



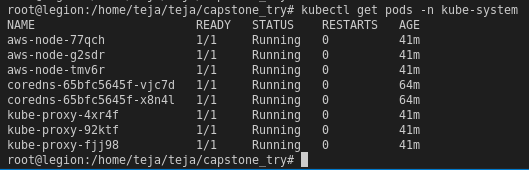
Wait for about 20 minutes for the cluster to be ready:



Check in EKS AWS Console:



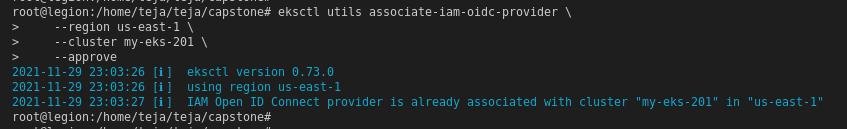
Pods available after running the command kubectl get pods –n Kube-System:



Also, important to update kubeconfig file for the cluster like below so there are no issues with connecting to the amazon EKS cluster.



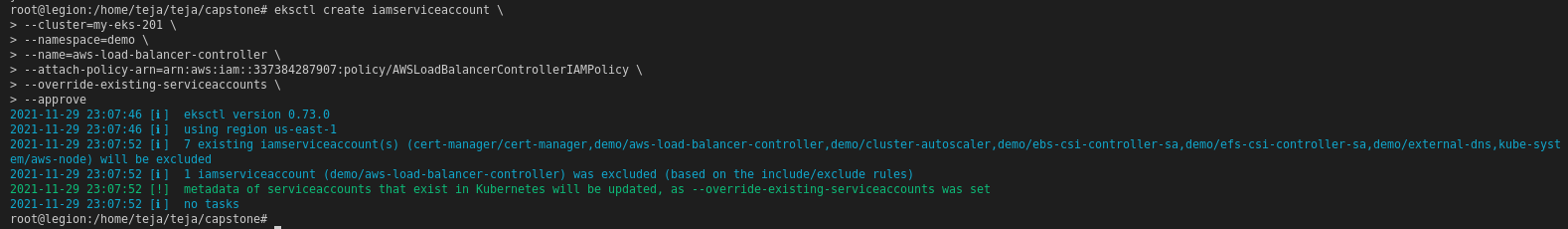
Associate OIDC provider:



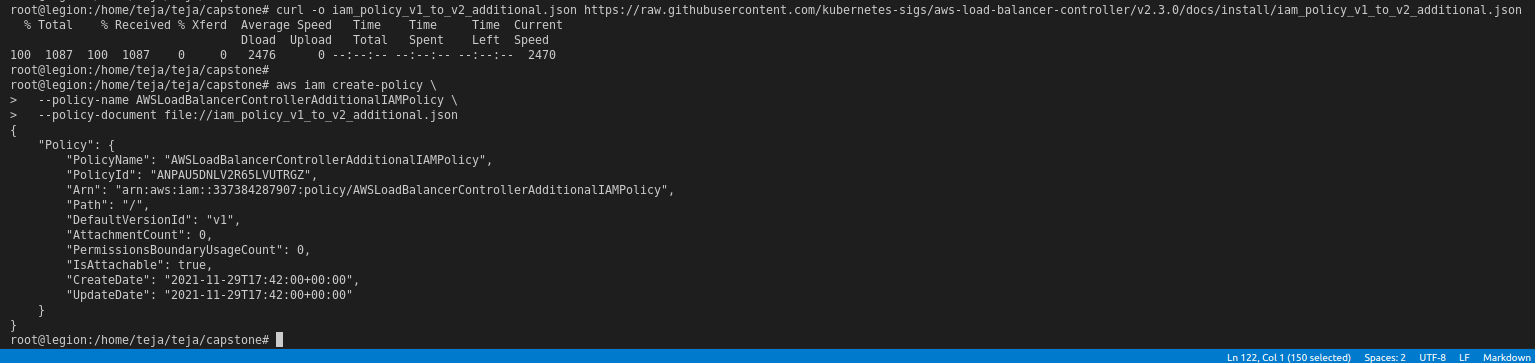
Create ALB policy:



Create service account for ALB:

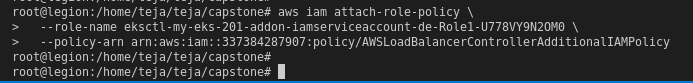


Now create ALB Additional controller policy:

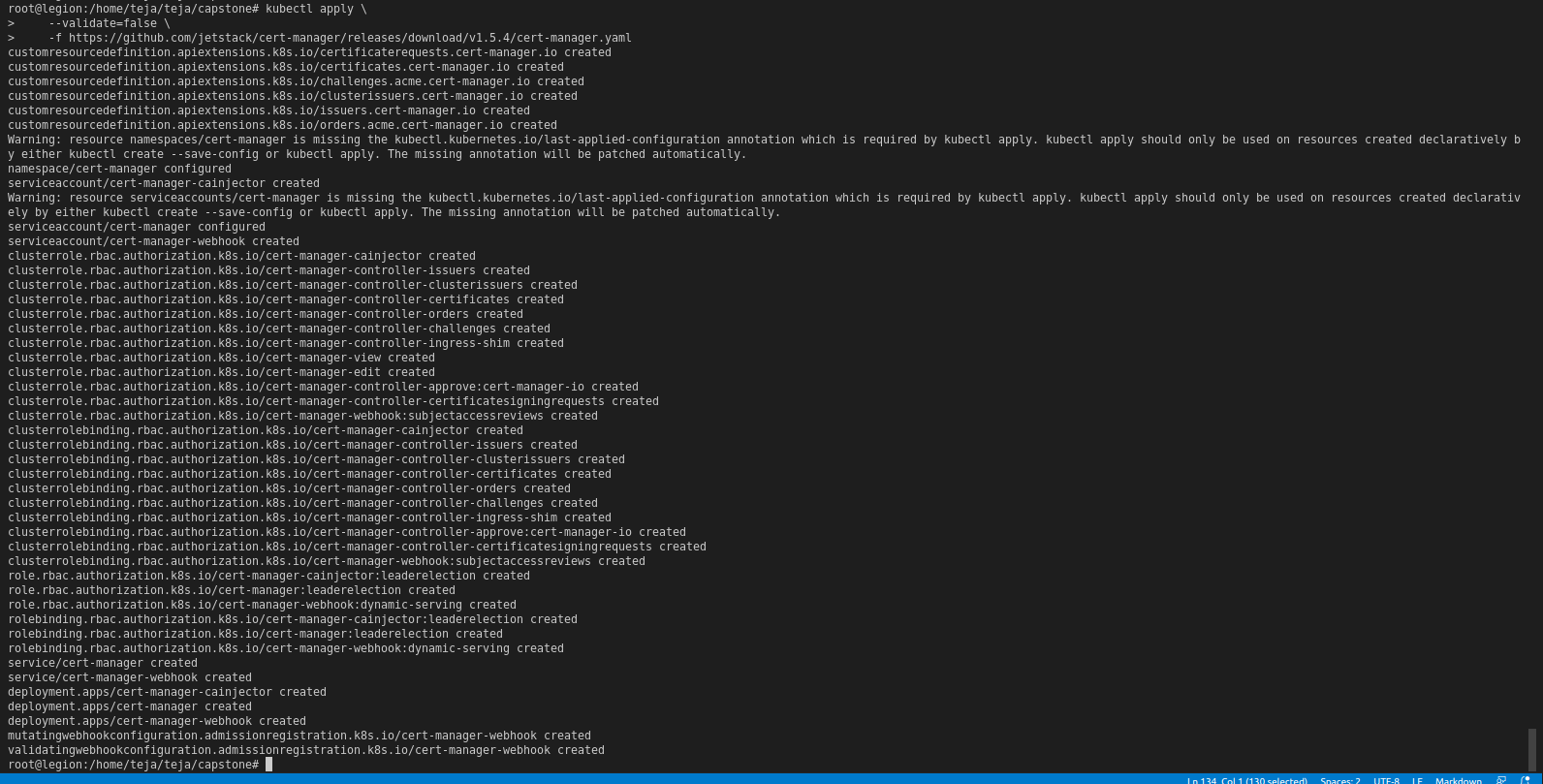


Now attach AWSLoadBalancerControllerAdditionalIAMPolicy by providing the role created for ALB controller.

Attach the IAM policy to the IAM role that you created in a previous step. Replace your-role-name with the name of the role. If you created the role using eksctl, then to find the role name that was created, open the AWS CloudFormation console and select the eksctl-your-cluster-name-addon-iamserviceaccount-kube-system-aws-load-balancer-controller stack. Select the Resources tab. The role name is in the Physical ID column. If you used the AWS Management Console to create the role, then the role name is whatever you named it, such as AmazonEKSLoadBalancerControllerRole.

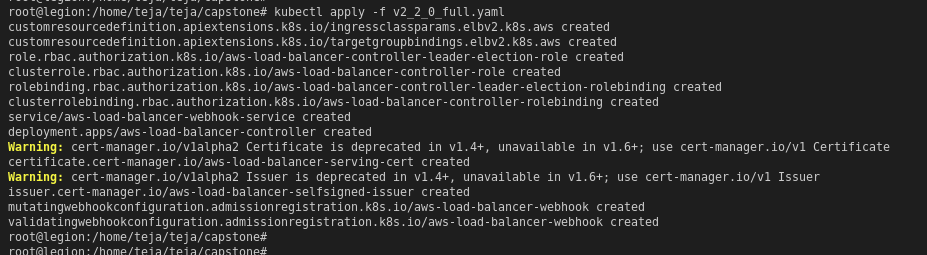


Now install the certificate Manager:

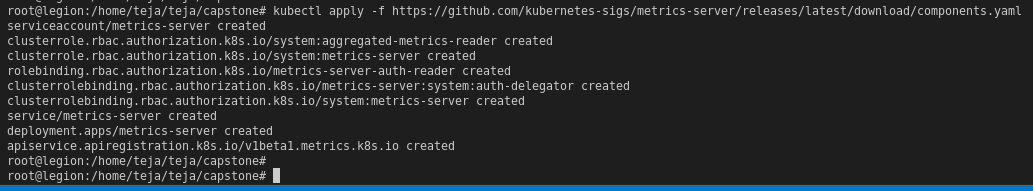


Download controller specification file as guided in <https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html>

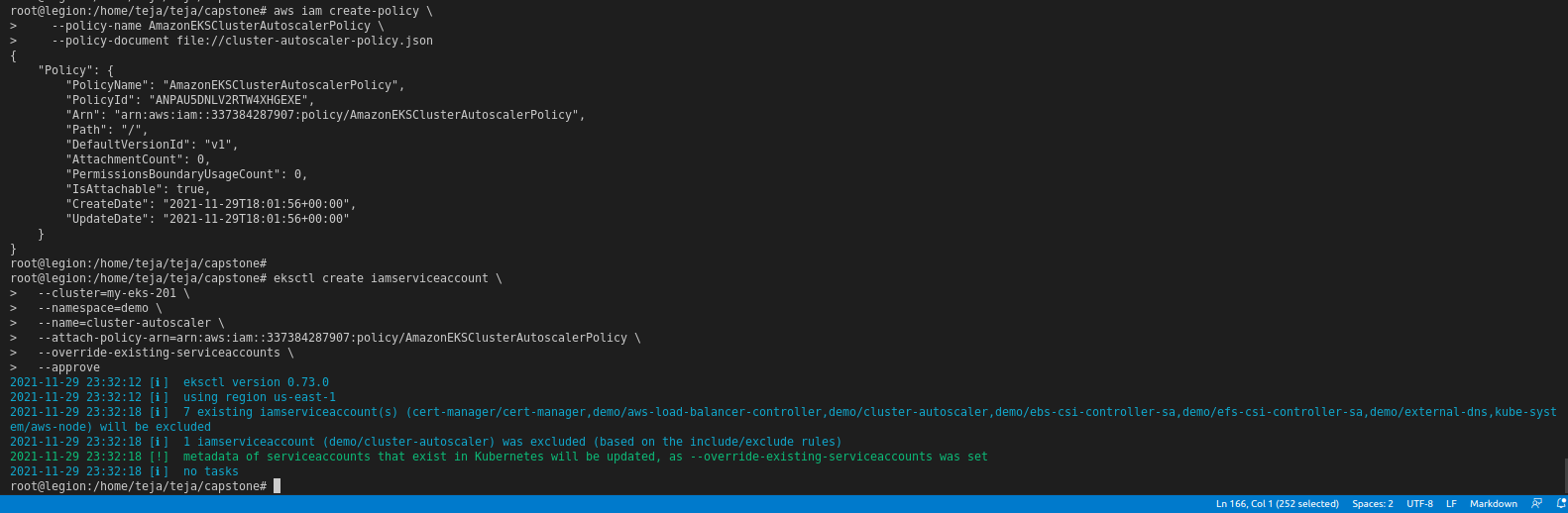
and then apply it after making necessary file modifications given in the guide:



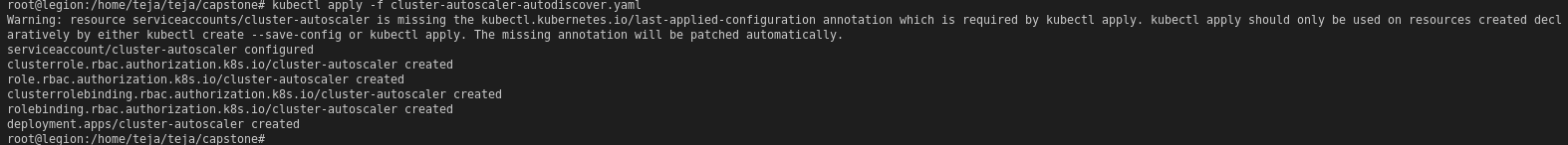
Install the metrics server now:

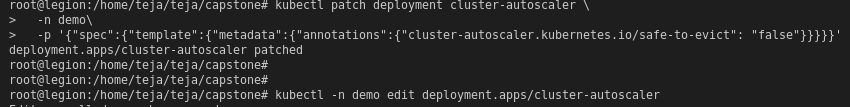


Create autoscalar policy and service account:

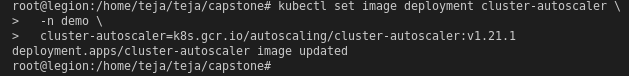


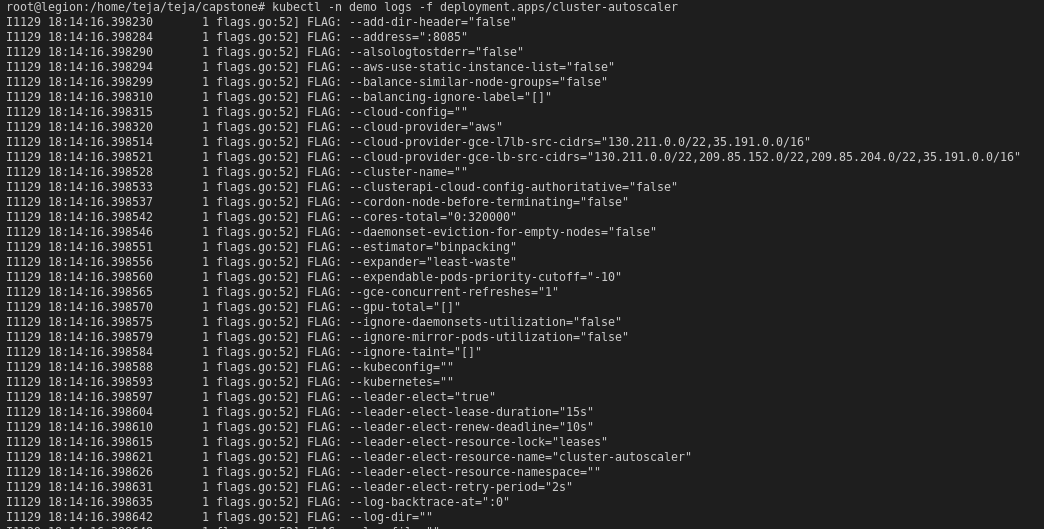
Follow further autoscalar creation steps from https://docs.aws.amazon.com/eks/latest/userguide/cluster-autoscaler.html



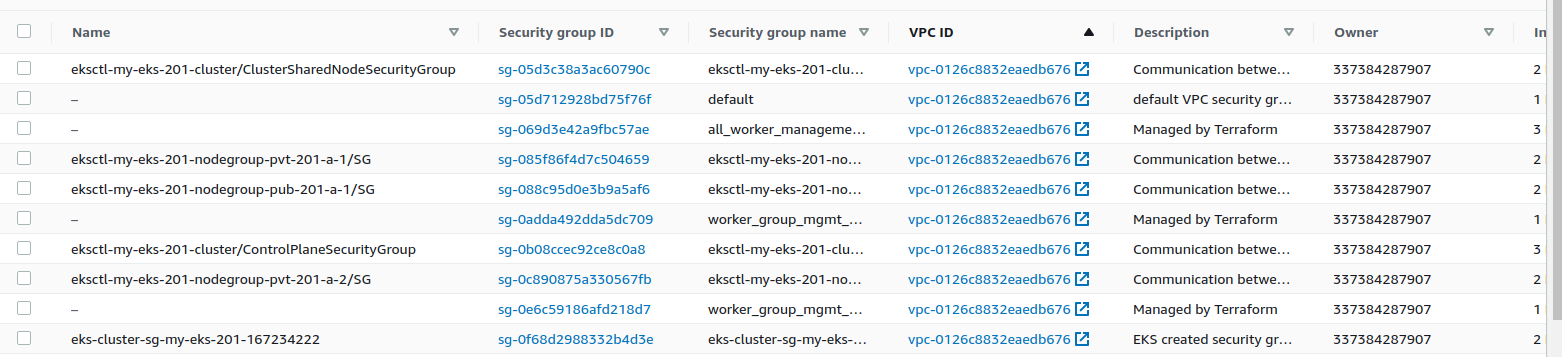






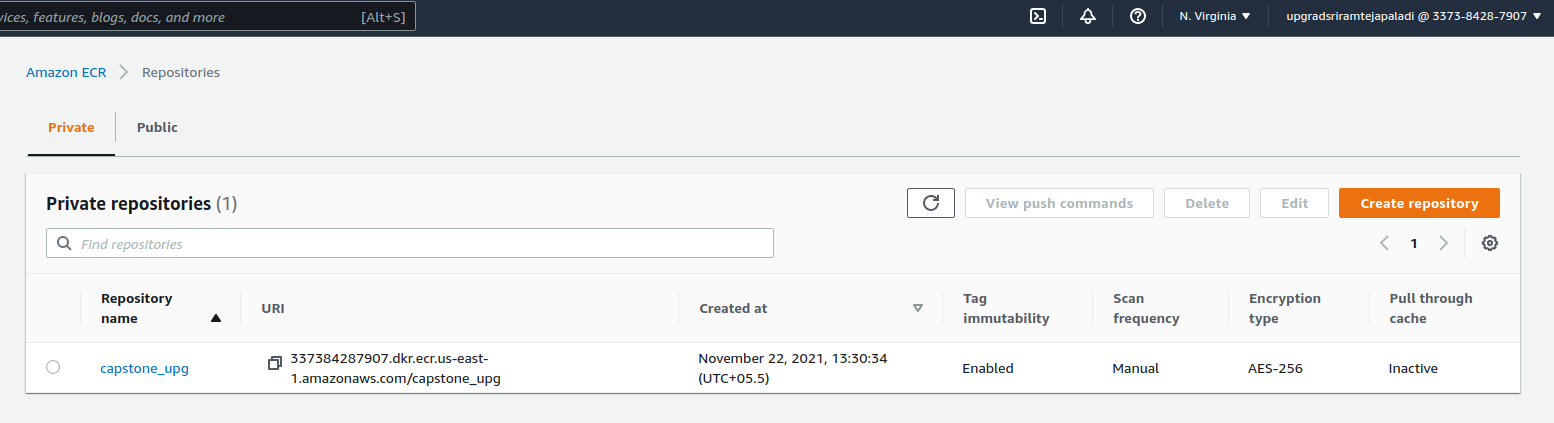


All the relevant security groups created as part of cluster creation, node creation and rest of deployments so far are shown below:

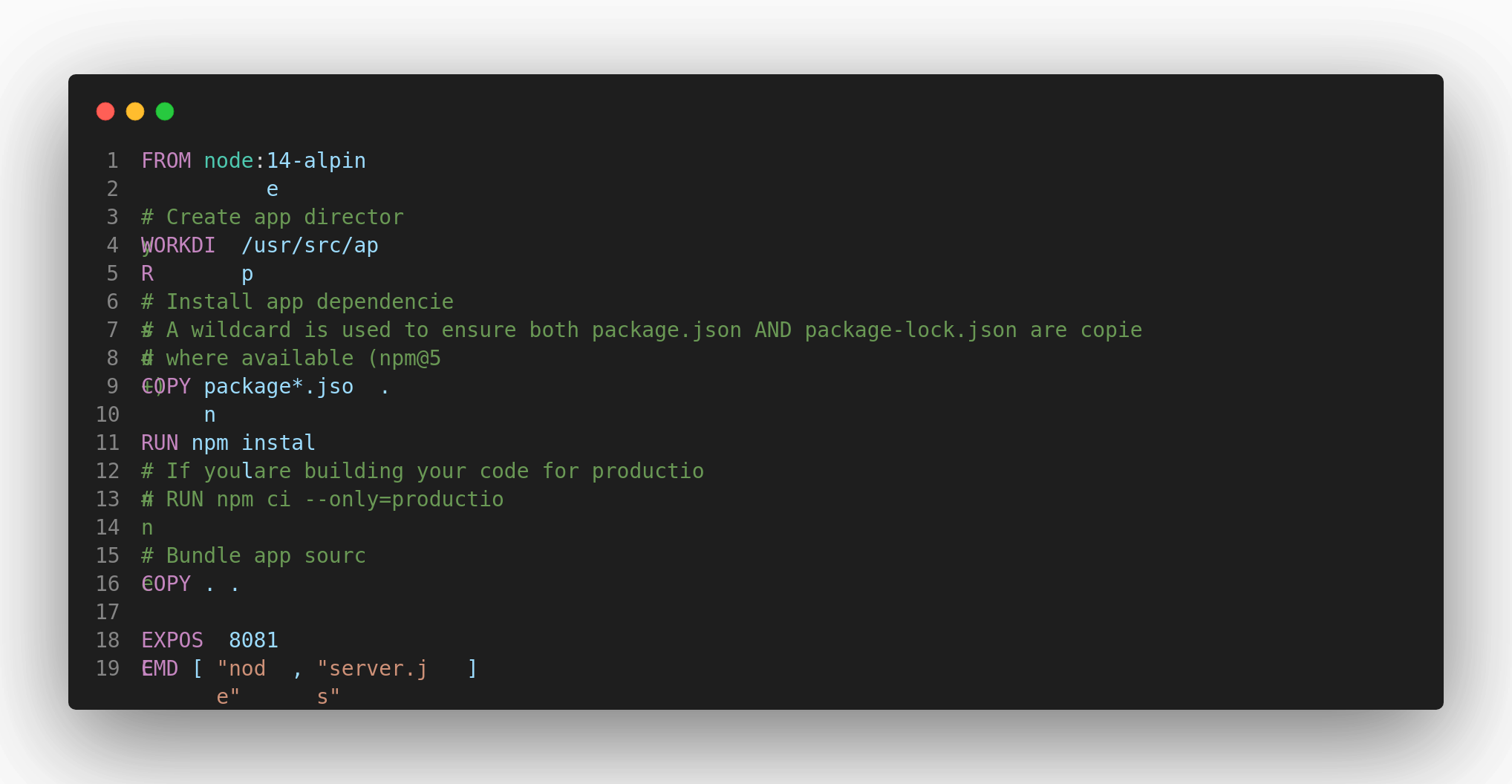


Task 2: Deployment of sample application

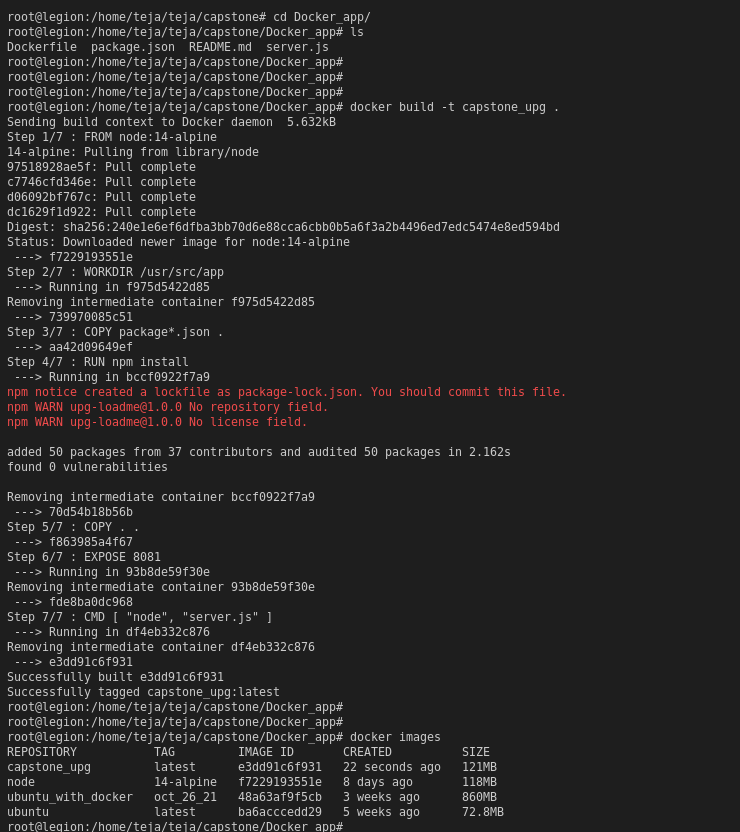
Create ECR repository in AWS;



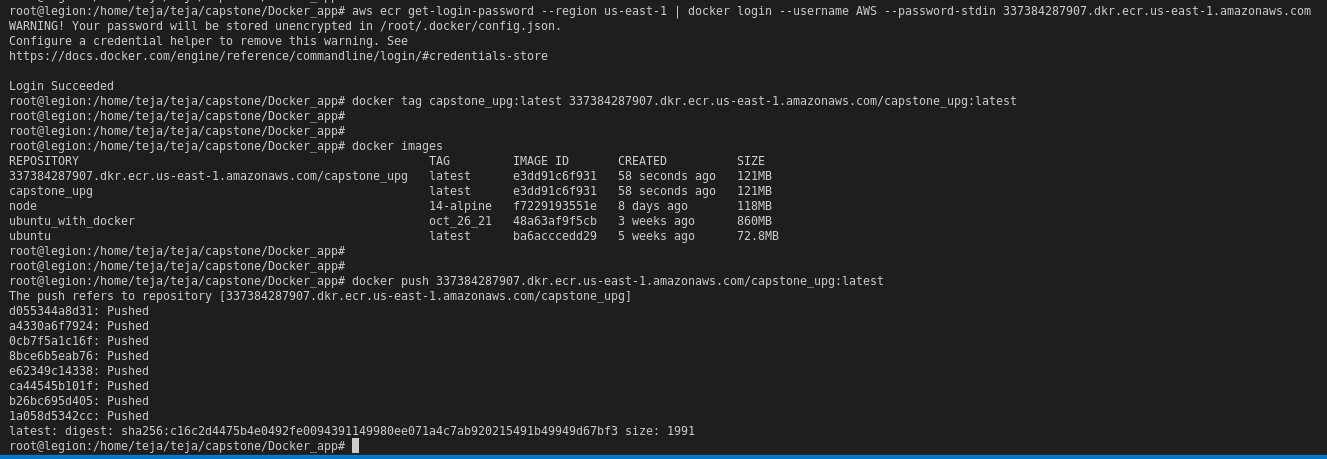
Create Dockerfile:



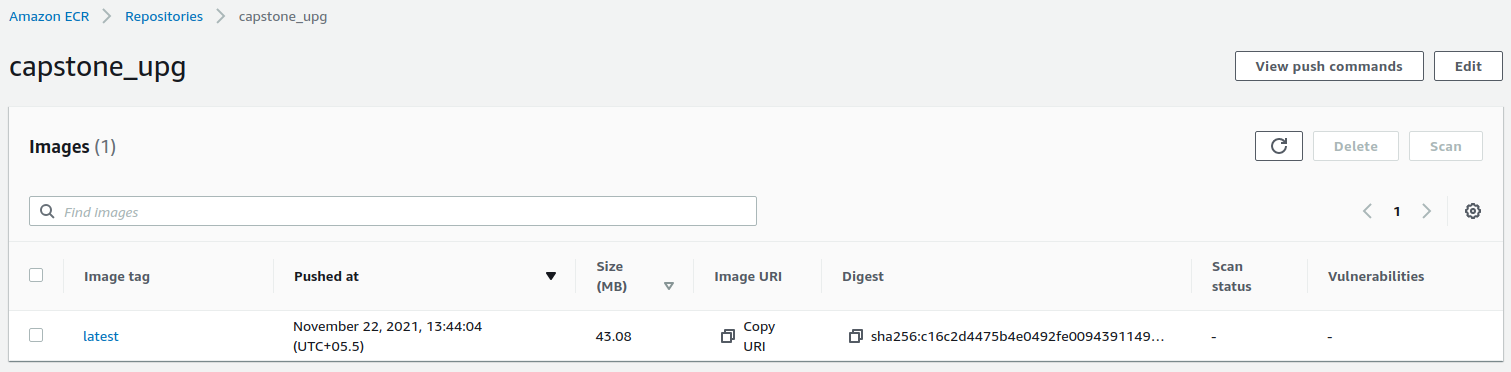
Build it:



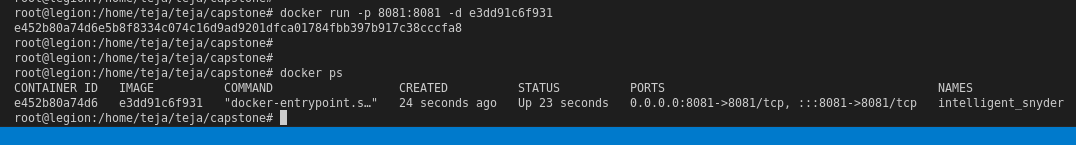
Authenticate ECR repo through aws\_cli and tag and push the docker image:

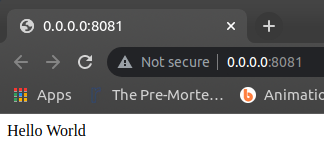


Check pushed ECR Image:



Run docker image to check the application:





Now the application is working!

Create the nodegroup by adding below code into the config file:



Use eks ctl command to create the node by using command “eksctl create cluster -f my-eks-conf.yaml”

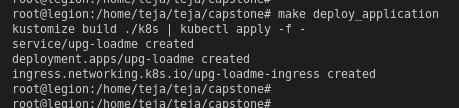
Create the demo namespace;



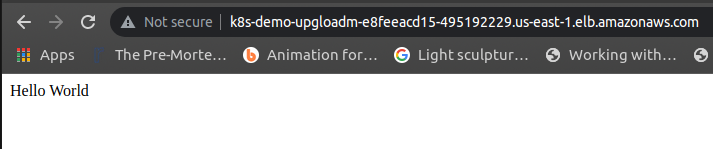
Create deployment, service and ingress yaml files for the app image uploaded into ECR repo before.

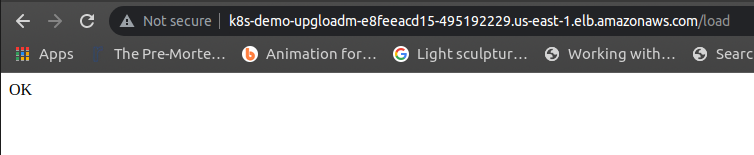
Deploy the files using “kubectl apply -f <file-name.yaml>”

Or if you are making use of the Makefile, use the appropriate command name:



Now copy the load balancer DNS name and check the output in the browser:



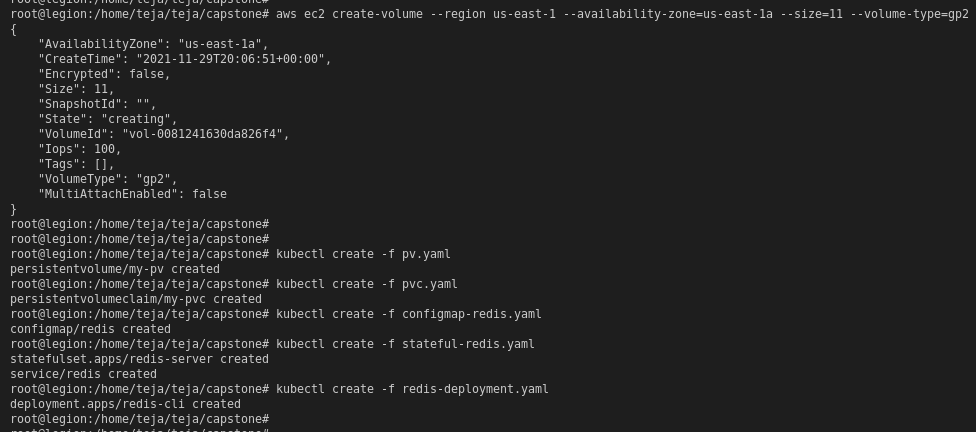


Task 3: Deploy Redis server on Kubernetes

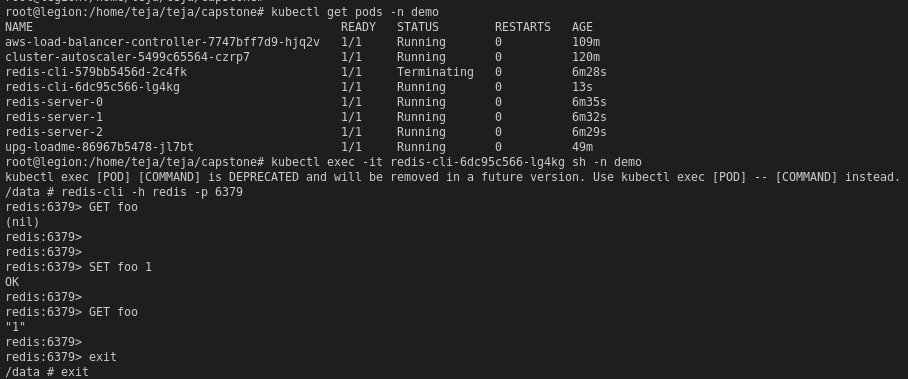
First create an EBS volume of required/desired memory using aws\_cli:

aws ec2 create-volume --region us-east-1 --availability-zone=us-east-1a --size=11 --volume-type=gp2

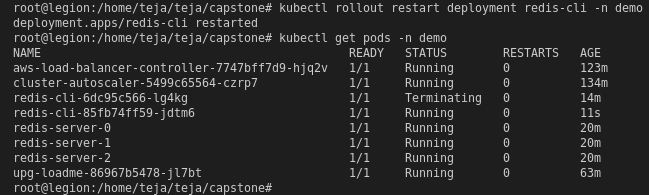
Then create and deploy PersistentVolume, PersistentVolumeClaim, config-map for redis, statefulset redis server and a redis deployment:



Now list all the pods and get into the redis-cli pod:



Now restart the deployment:

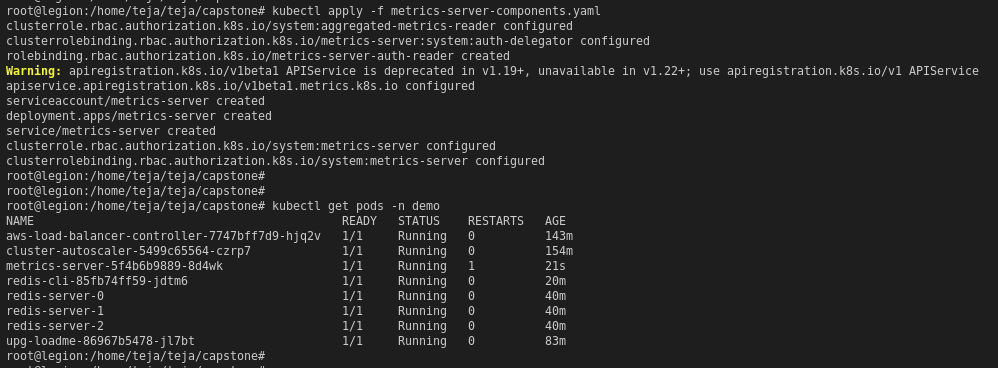


Check for the new redis-cli pod created and login to it to check that the previously set foo key value is still 1 showing that the data is persisted:



Task 4: Test auto scaling of the application

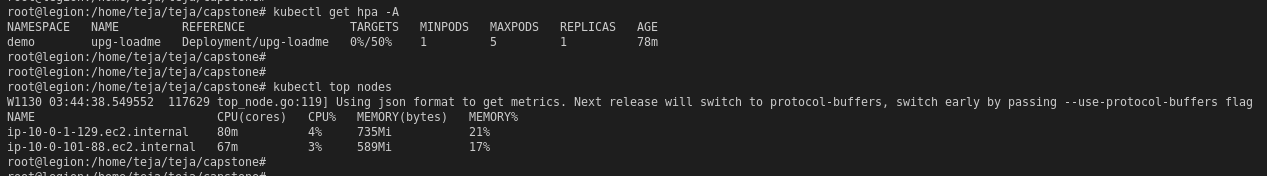
Deploying metrics server components and checking the metrics pod:



Deploy HPA file for the upg-loadme-app:

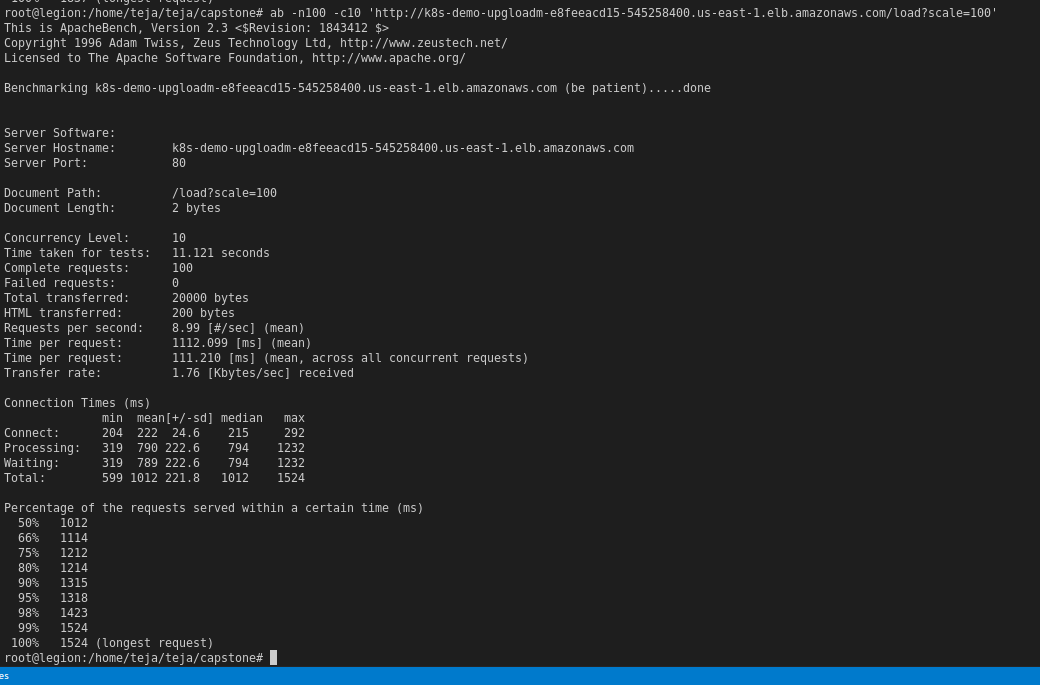


Check HPA and after some time the metrics will load and can be checked with the top command:



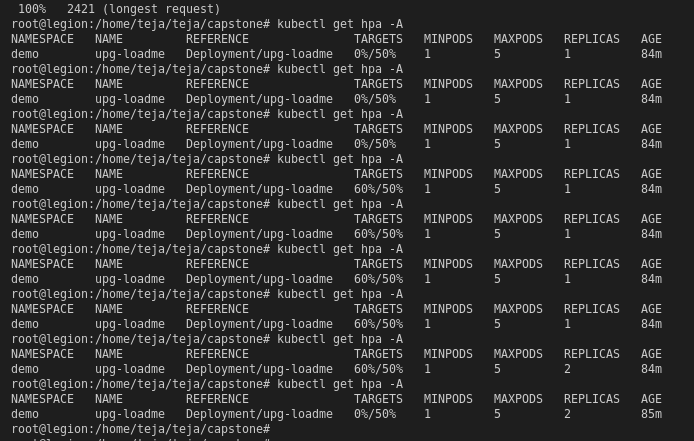
Now using the apache benchmark, scale up the nodes by increasing the load:

ab -n100 -c10 'http://<INSERT-LB-DNS>/load?scale=100'

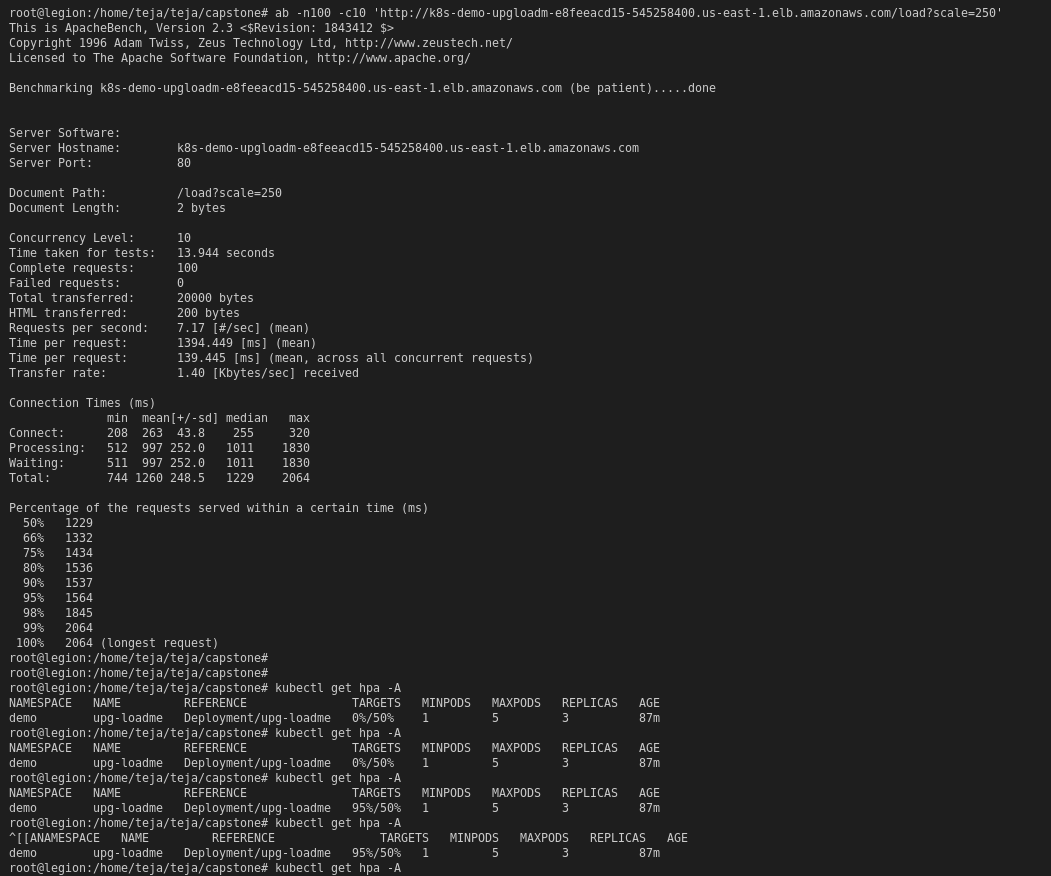


Increase the load in steps and notice the autoscaling work:

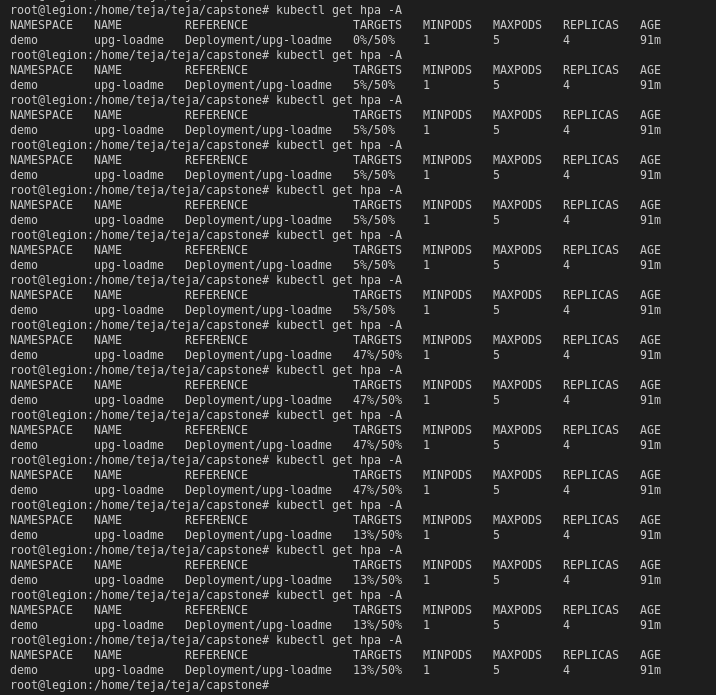
Load = 150



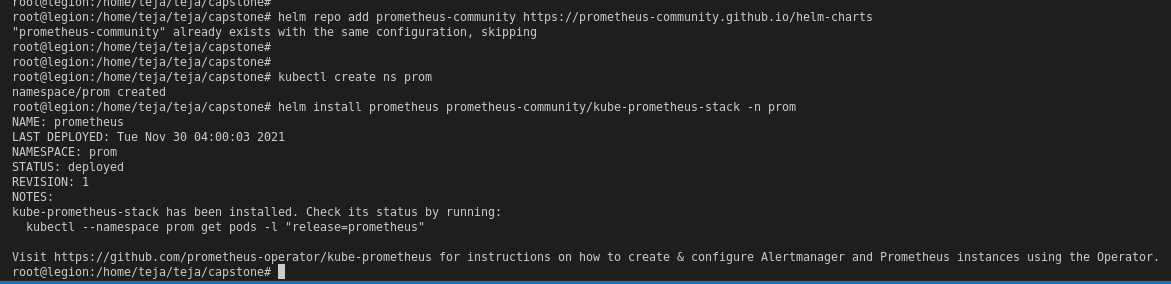
Load = 250



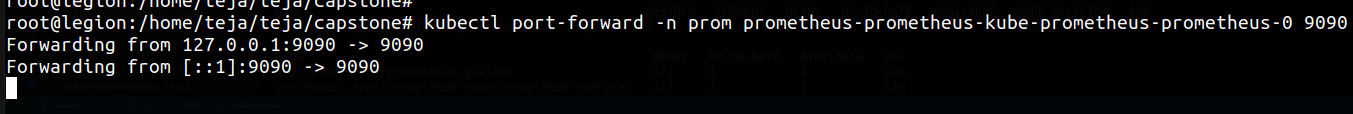
Load = 300



Now Install Prometheus to visualize and monitor the load on the cluster:



Port forward Prometheus pod and if needed Grafana pod as well:





Access the Prometheus application on the local host port 9090 and execute the prom commands:

sum(kube\_pod\_container\_status\_ready{namespace="demo",pod=~"upg-loadme-.\*"})

This shows the load on the application and as done before keep increasing the load using the apache benchmark and the prometheus will reflect the change in the load:

