Assignment–7

Module-10: Container Orchestration Tool - Kubernetes

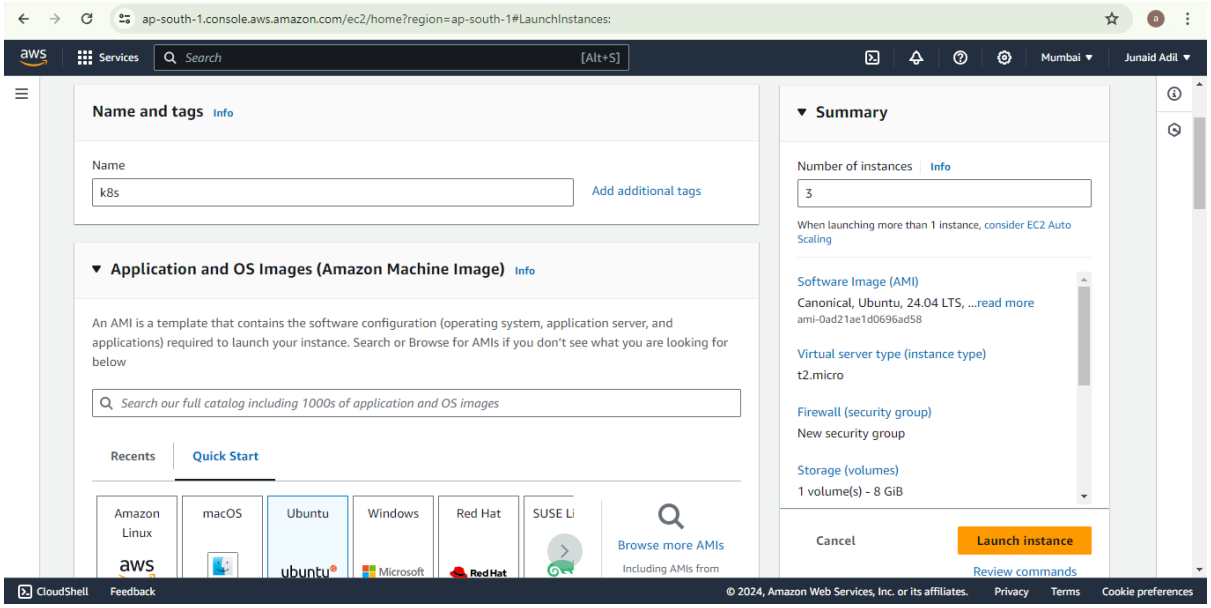
Submitted by : Shaik Junaid Adil

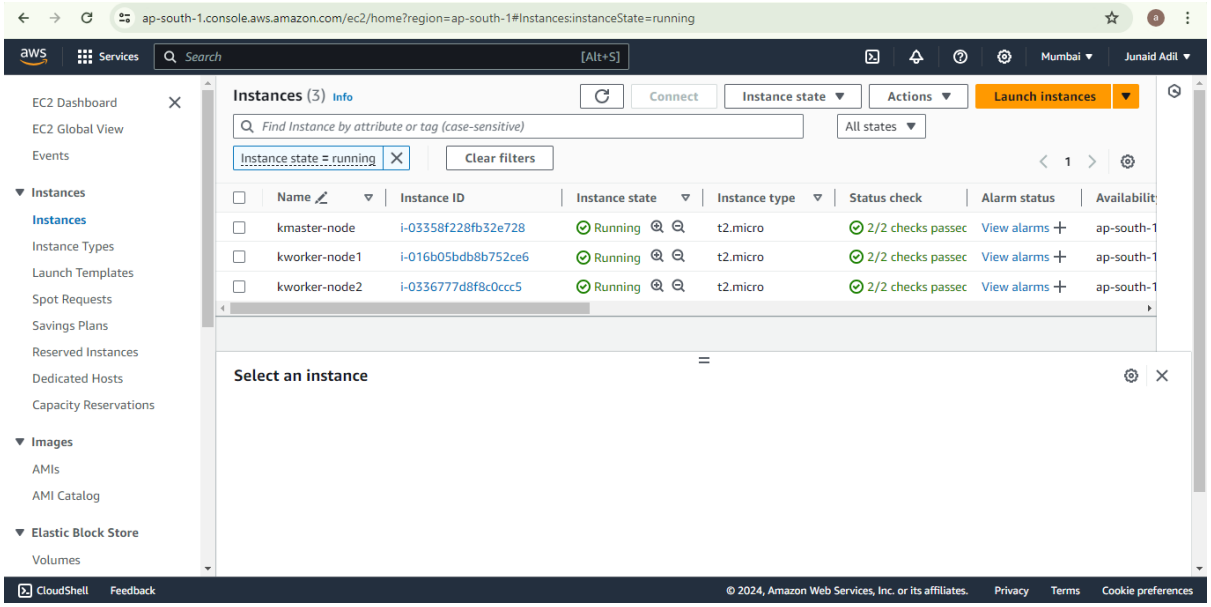
Date of Submission: 11-07-2024

Submitted to: Vikul

L1 - Create Deployment Controller Object to Deploy the Application Image Created in Docker Module and Expose it to the Internet

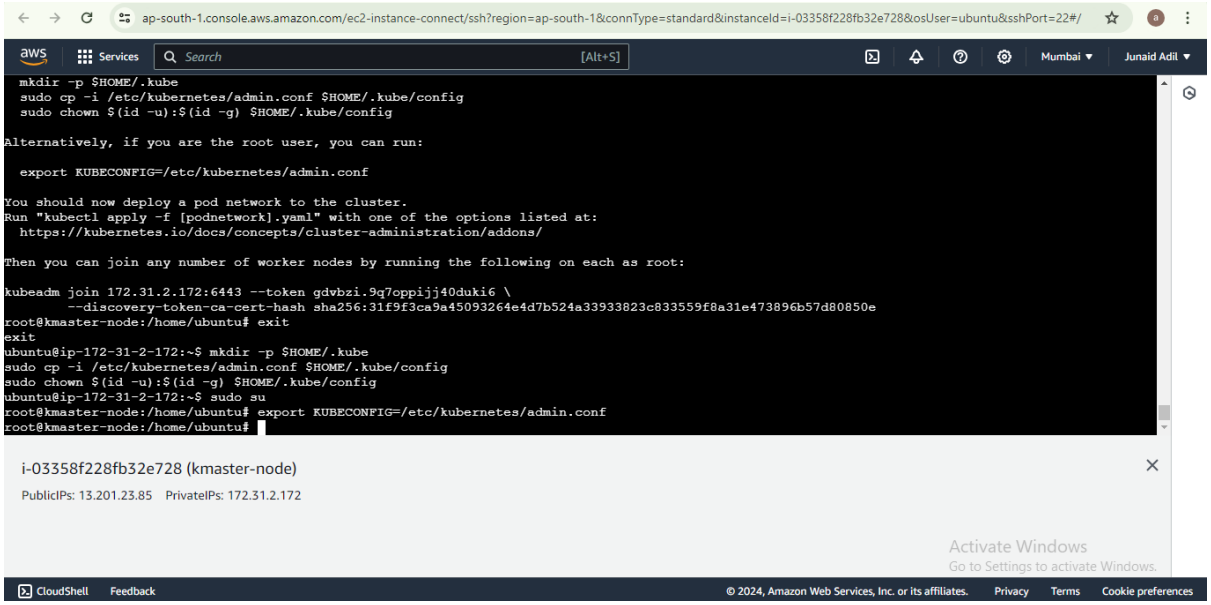
Step 1: Create 3 Instances, 1 for Master node, and 2 for worker nodes.



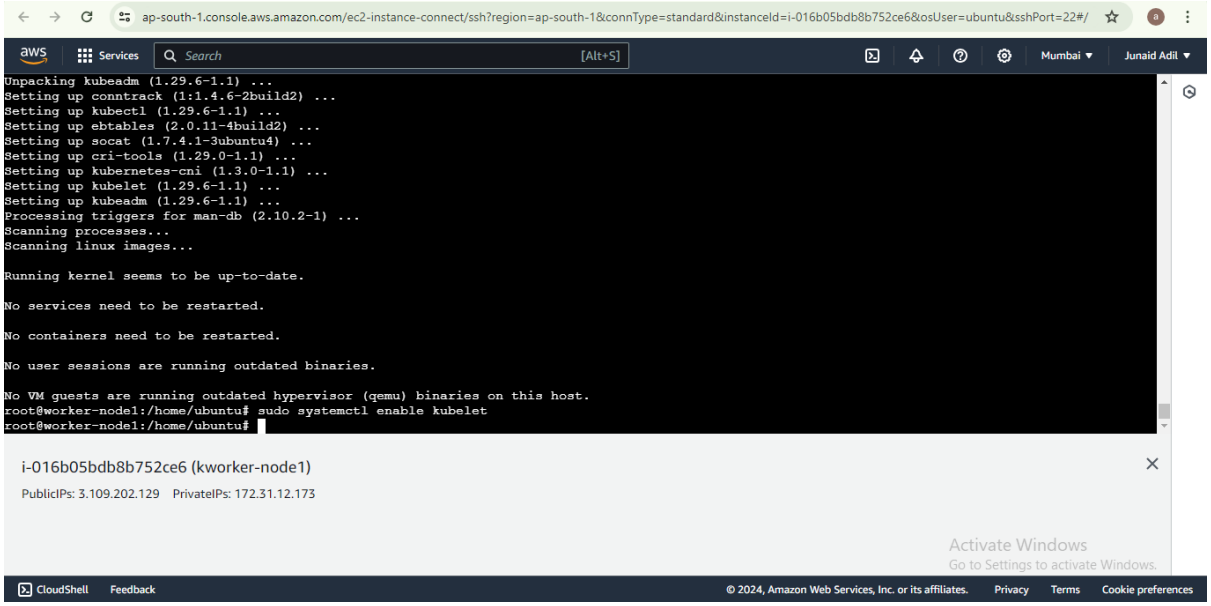


Step 2: Install Kubernetes and all the pre requisites in master node and worker nodes.

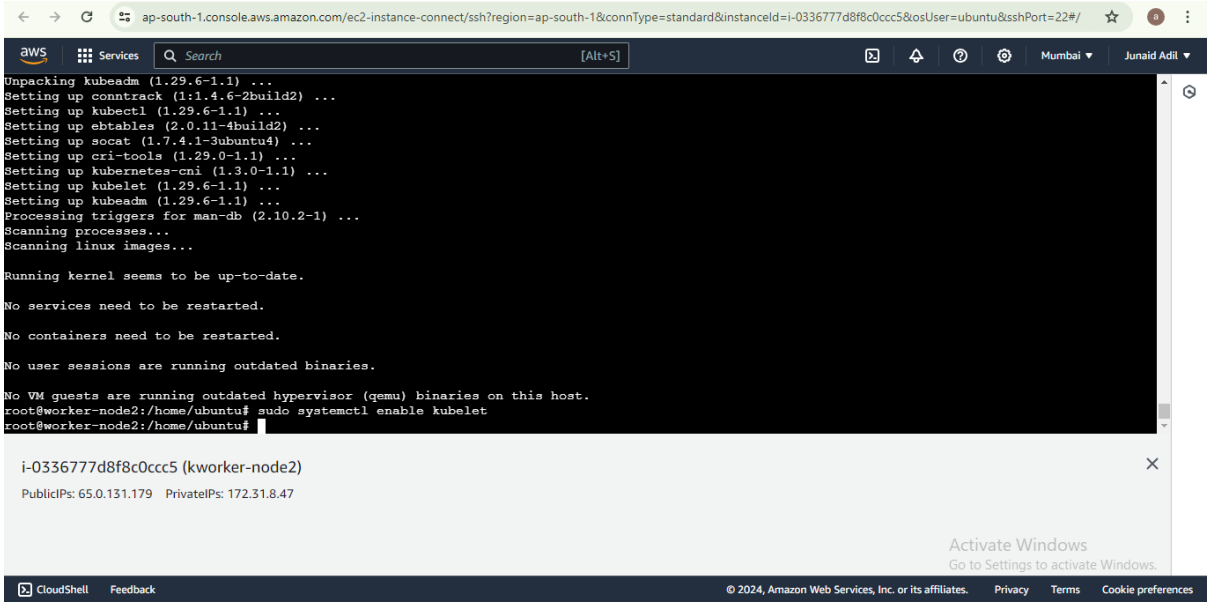
Master node



Worker node1

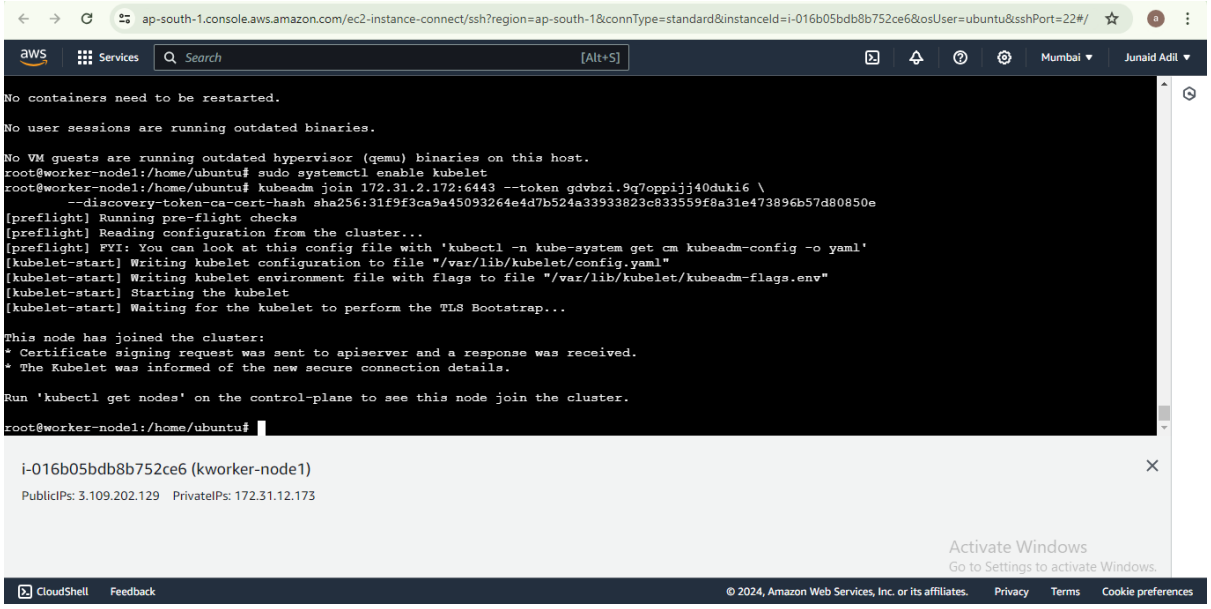


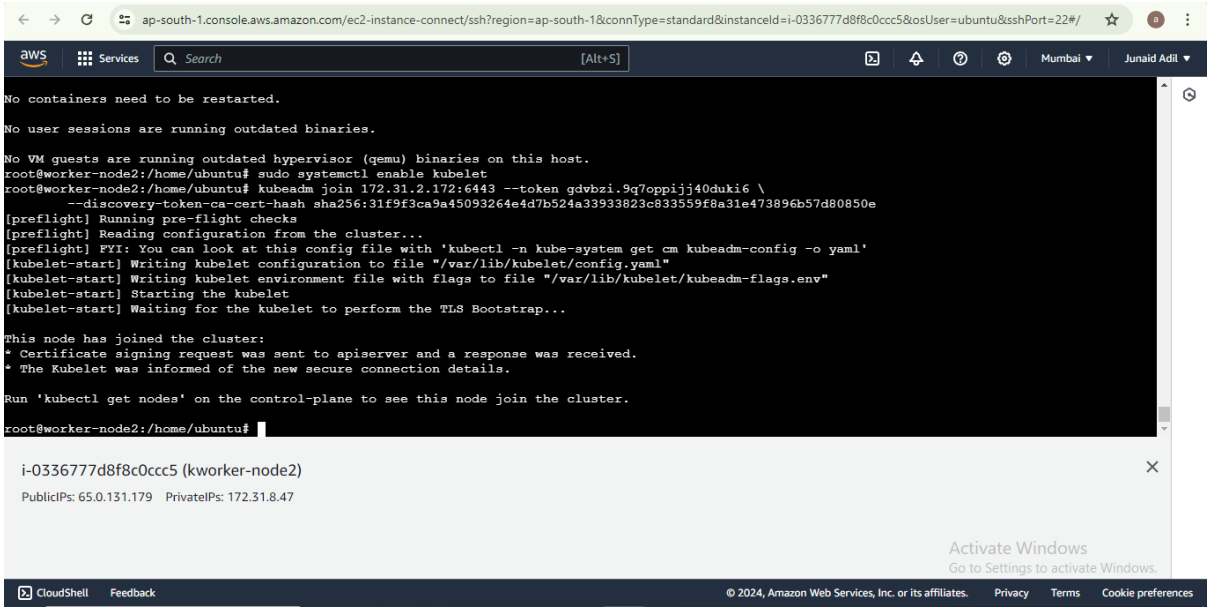
Worker node2



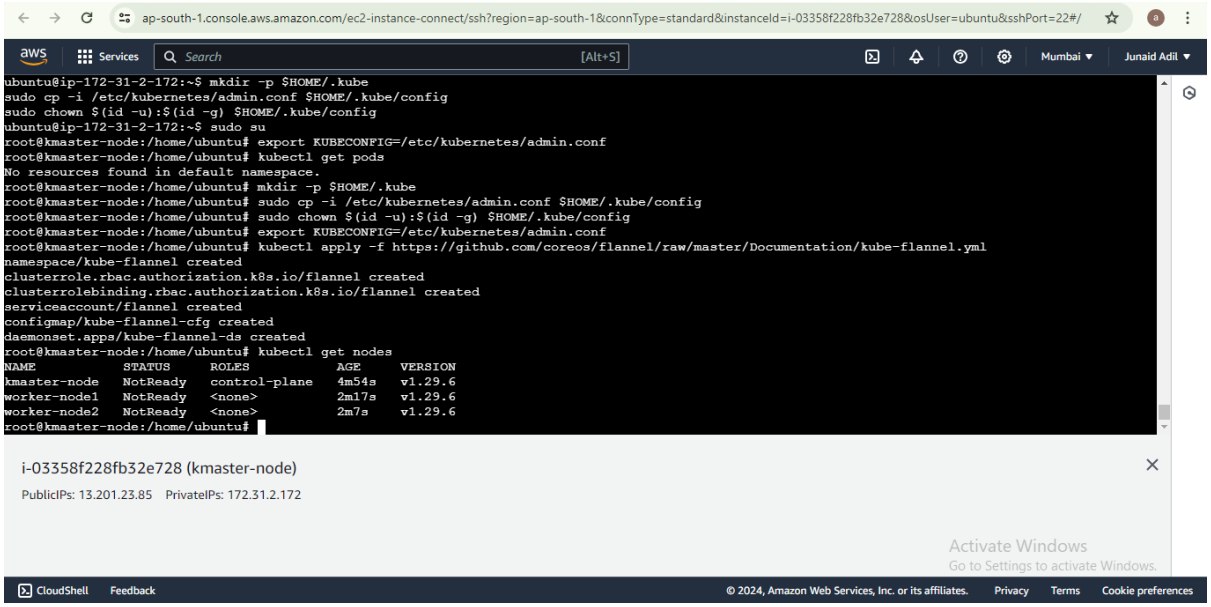
Step 3: Use the command to connect worker node to master node. This command has the Ip address along with the Port number and Token.

“ kubeadm join 172.31.2.172:6443 --token gdvbzi.9q7oppijj40duki6 \
--discovery-token-ca-cert-hash sha256:31f9f3ca9a45093264e4d7b524a33933823c833559f8a31e473896b57d80850e “





Step 4: Run “kubectl get nodes” to see the details of nodes



Step 5: Create a deployment file “deploy.yaml” and enter the data to deploy and create replicas

```
root@kmaster-node:/home/ubuntu/deployment-files# cat nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: junaidadil/junaidnginx:latest
          ports:
            - containerPort: 80

root@kmaster-node:/home/ubuntu/deployment-files#
```

i-03358f228fb32e728 (kmaster-node)

PublicIPs: 13.201.23.85 PrivateIPs: 172.31.2.172

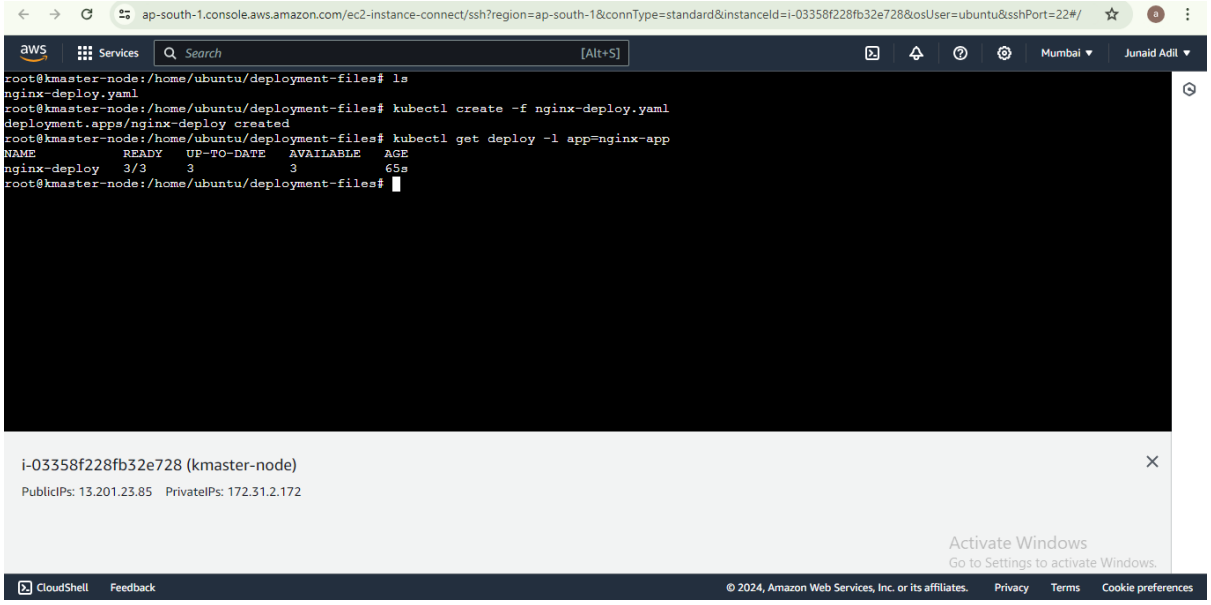
Step 6: Run “kubectl create –f nginx-deploy.yaml” to create a deployment controller object as per the definition.

```
root@kmaster-node:/home/ubuntu/deployment-files# ls
nginx-deploy.yaml
root@kmaster-node:/home/ubuntu/deployment-files# kubectl create -f nginx-deploy.yaml
deployment.apps/nginx-deploy created
root@kmaster-node:/home/ubuntu/deployment-files#
```

i-03358f228fb32e728 (kmaster-node)

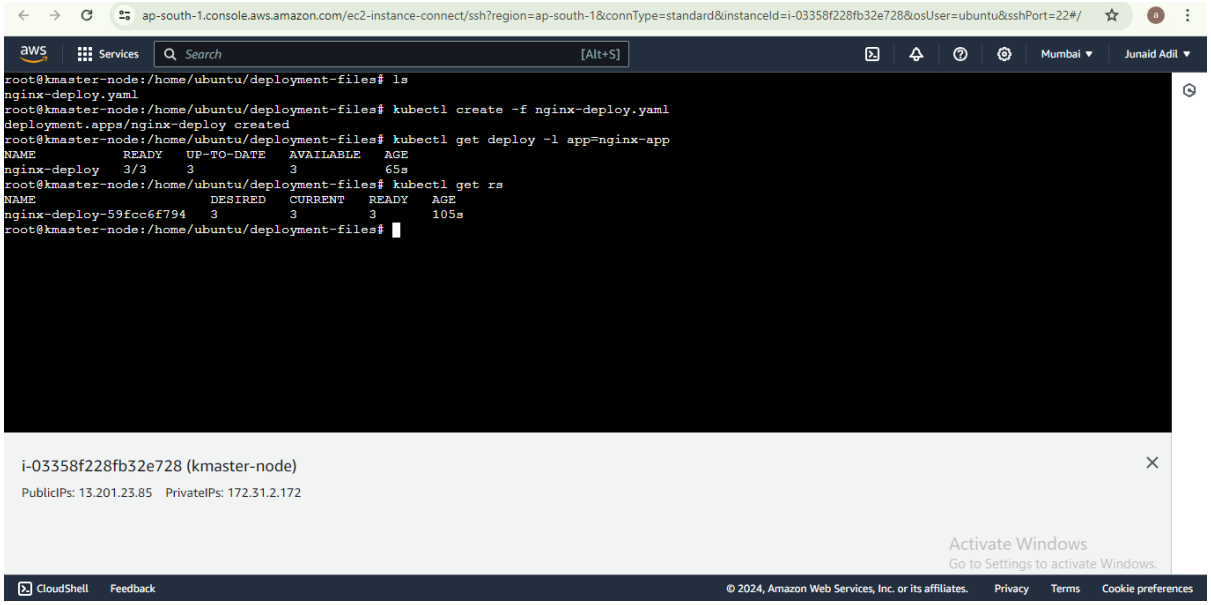
PublicIPs: 13.201.23.85 PrivateIPs: 172.31.2.172

Step 7: To check the deployed objects run command “kubectl get deploy -l app=nginx-app”

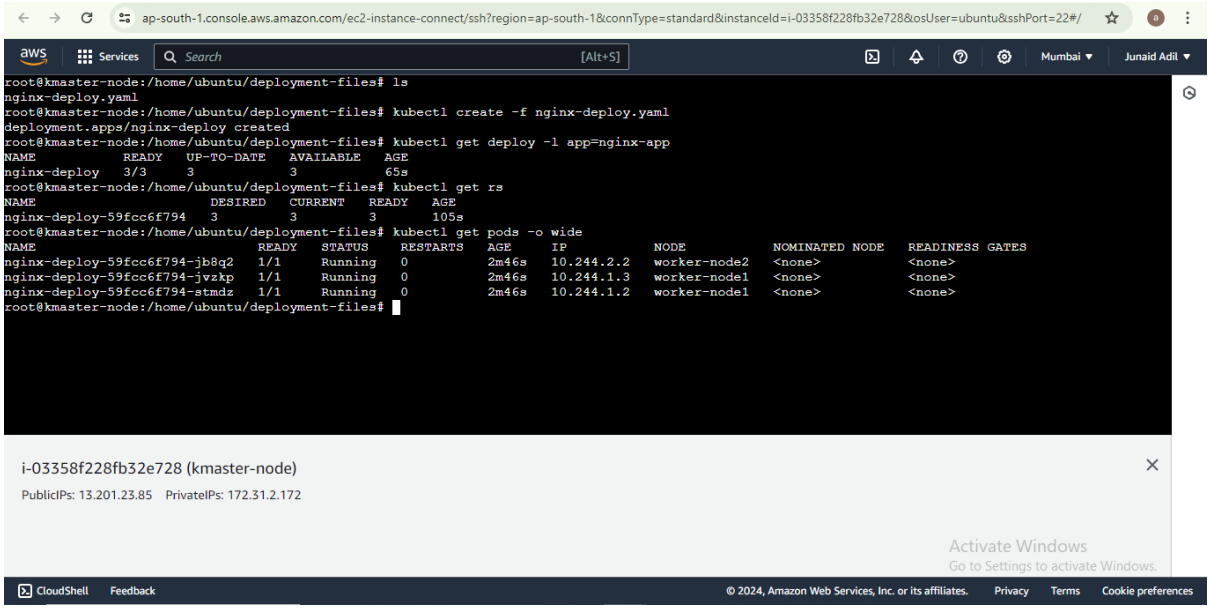


We can see it is upon running

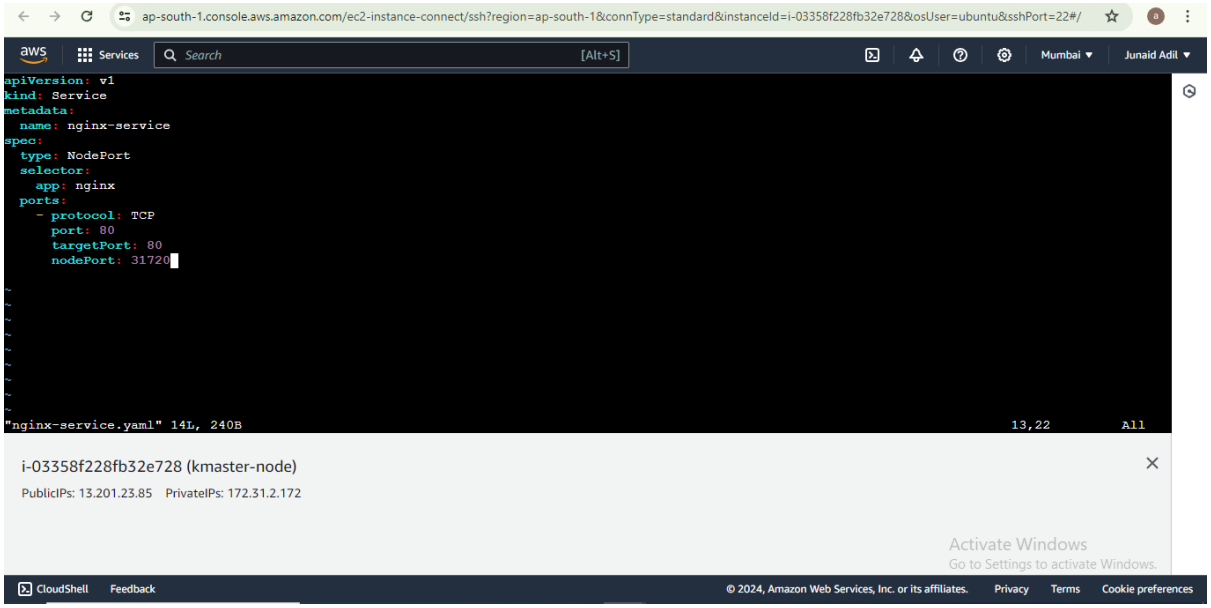
Step 8: To check the replica set, Run command “kubectl get rs”



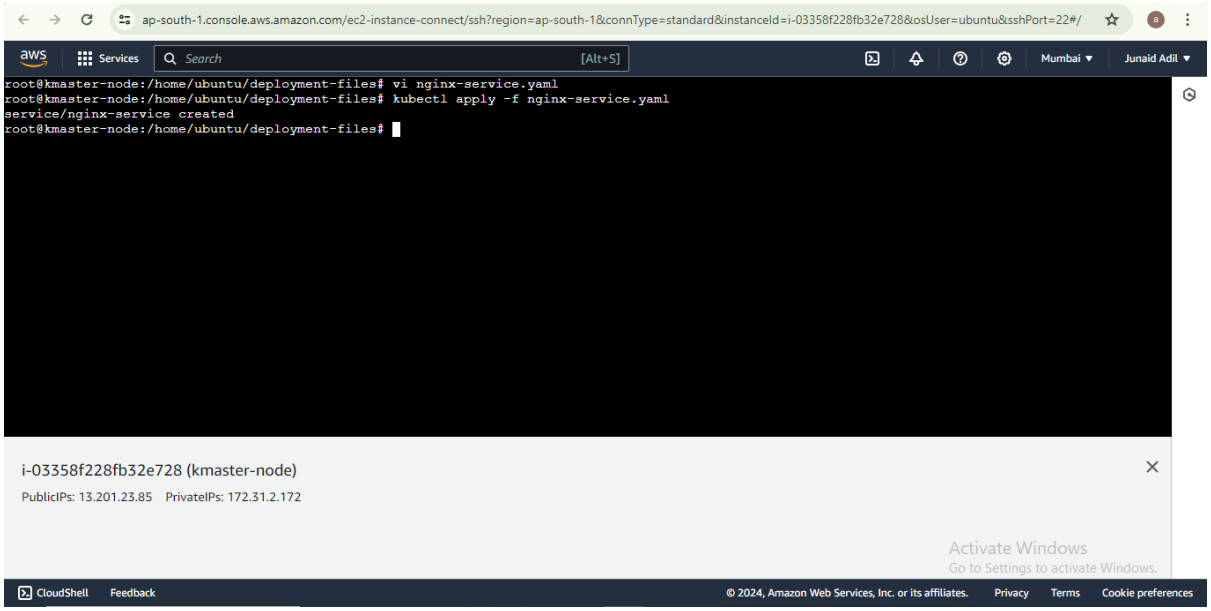
Step 9: To check in which nodes the 3 replicas are running, execute command “kubectl get pods –o wide”



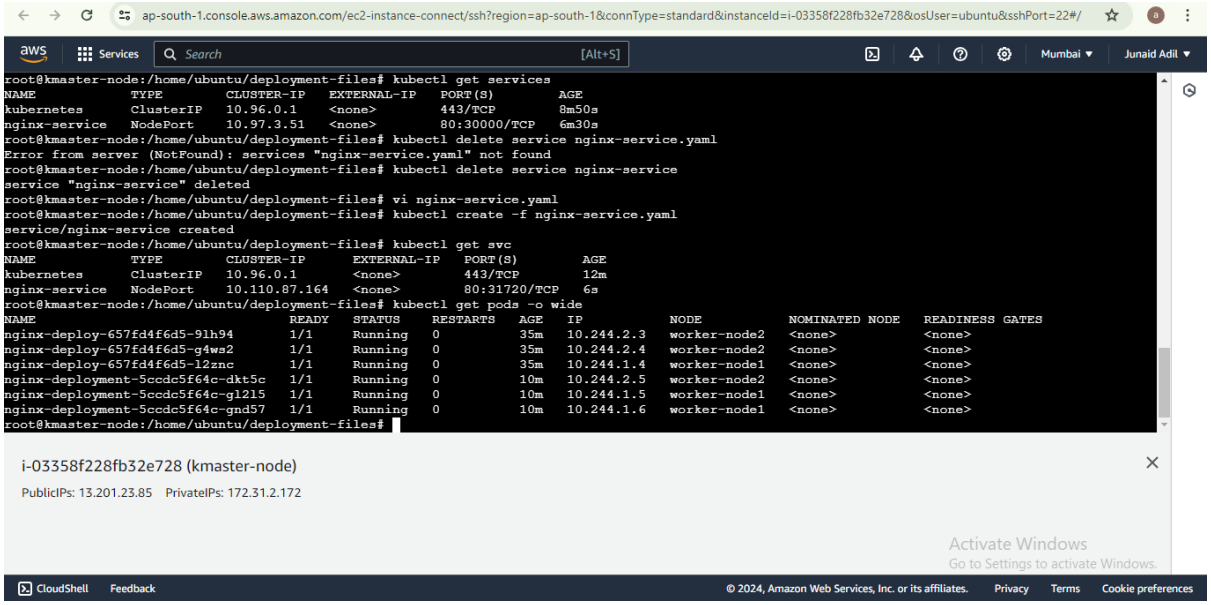
Step 10: Now to expose it to the internet, we need to create a “Node-port” service. For that create a file “nginx-service.yaml”



Step 11: Execute command “kubectl apply -f nginx-service.yaml” to create/apply the service.

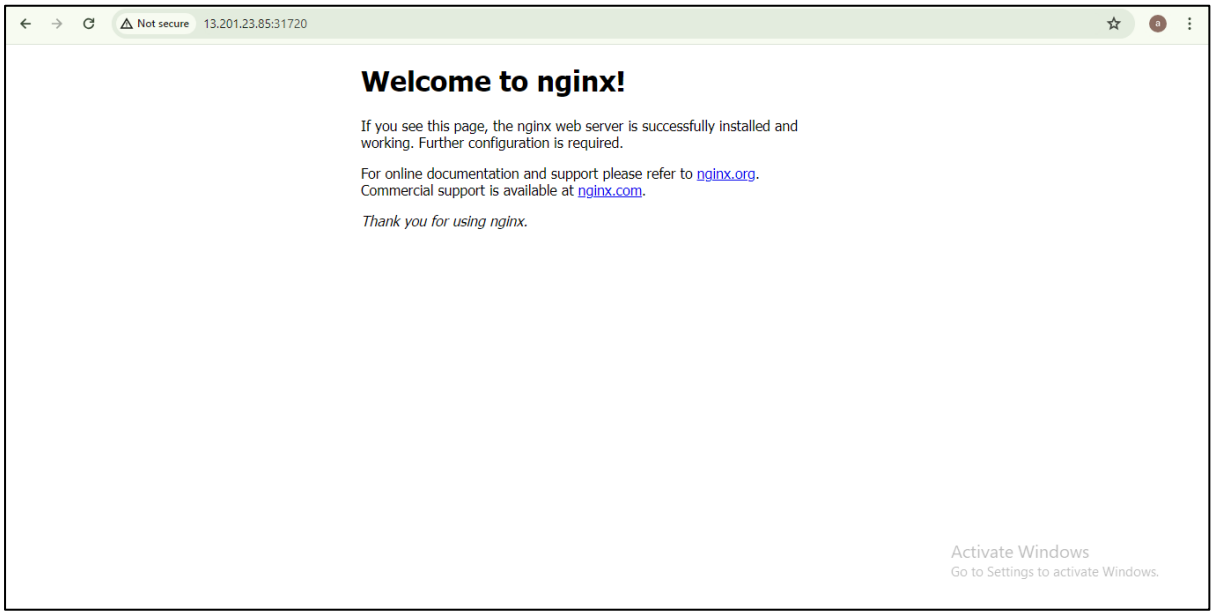


Step 12: Execute the command to get the IP address, Node and status “kubectl get nodes -o wide”



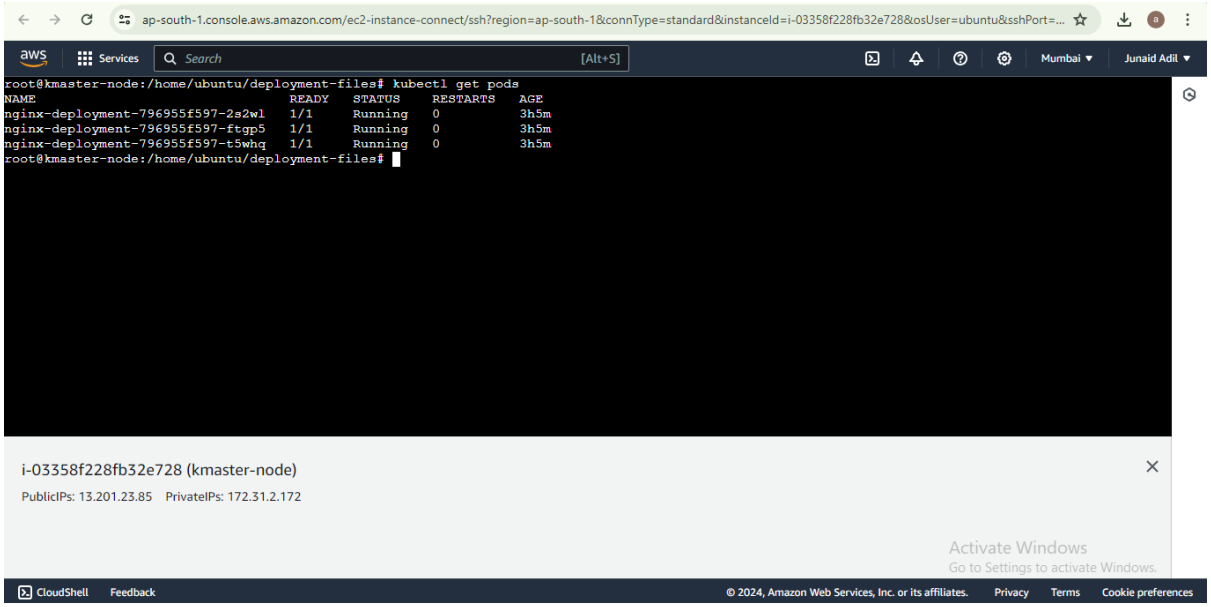
Step 13: Using public IP and the port number 31720 we can access the application through browser

<External IP>:<Port Number>



L2 - Scale-up and Scale-Down the Pods Deployed.

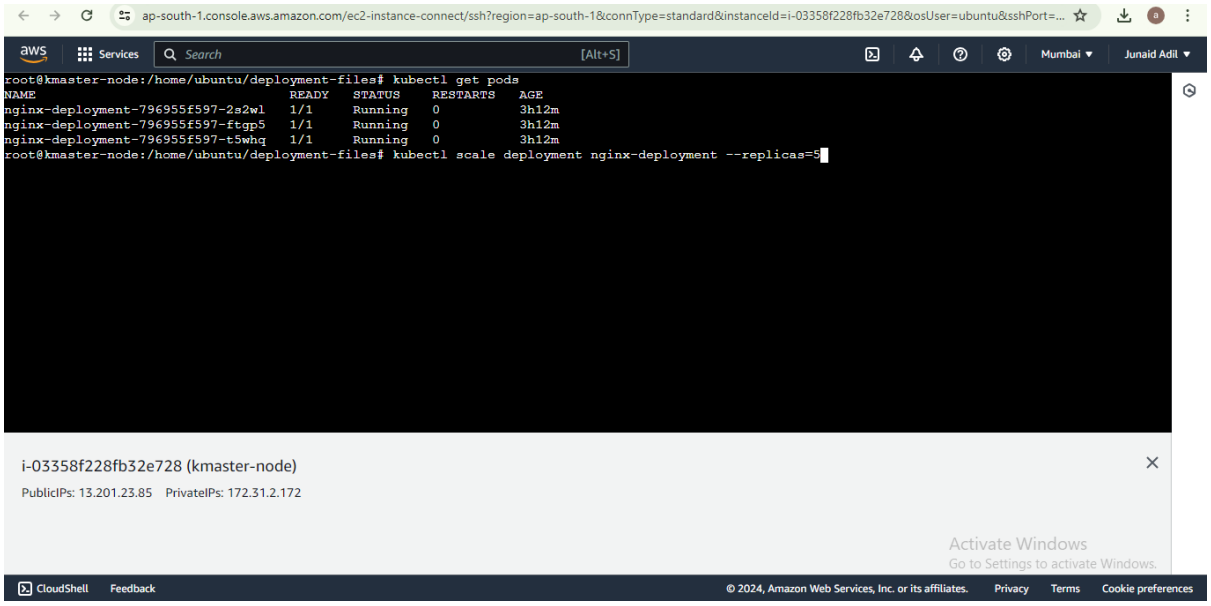
Step 1: To scale up and scale down the Pods, Deploy the Pods.

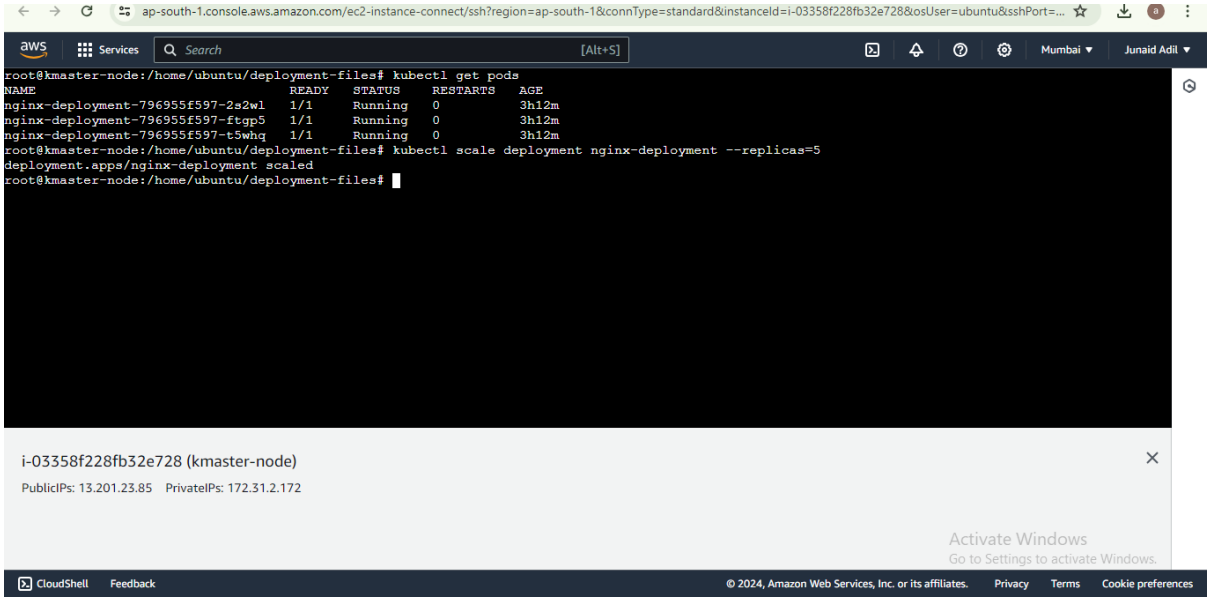


Deployed 3 Pods.

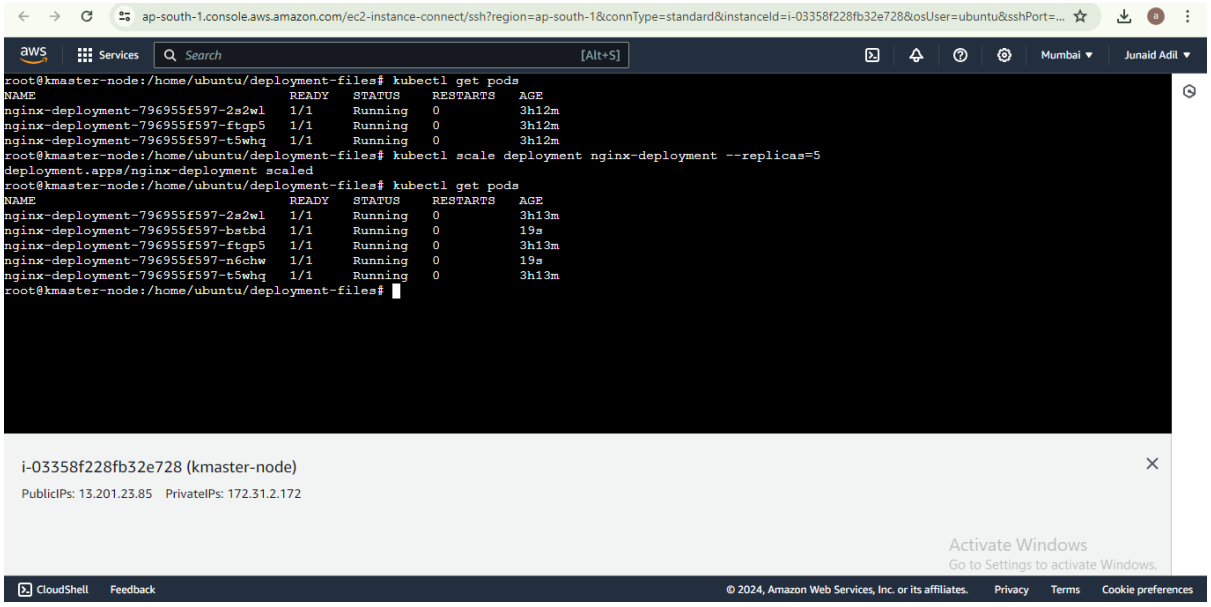
Step 2: Syntax " kubectl scale deployment <deployment-name> --replicas=<desired-replica-count> "

Now to scale up we can use command "kubectl scale deployment nginx-deploy --replicas=5" here 5 is the number of Pods we want to scale up with.





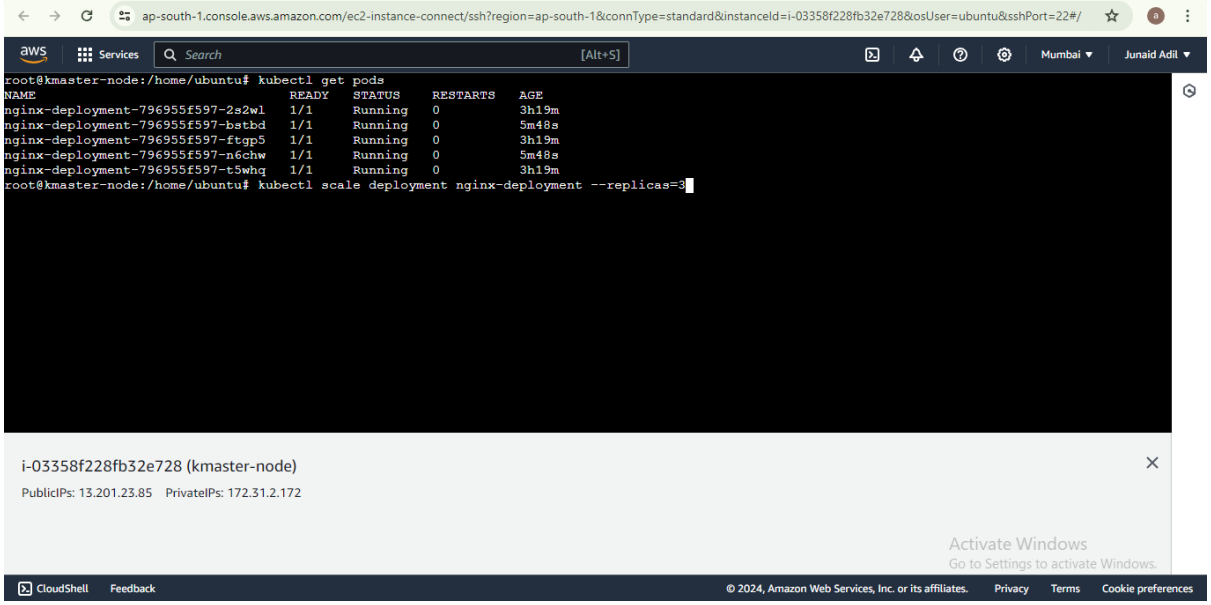
We can see the pods are scaled up to 5 pods.

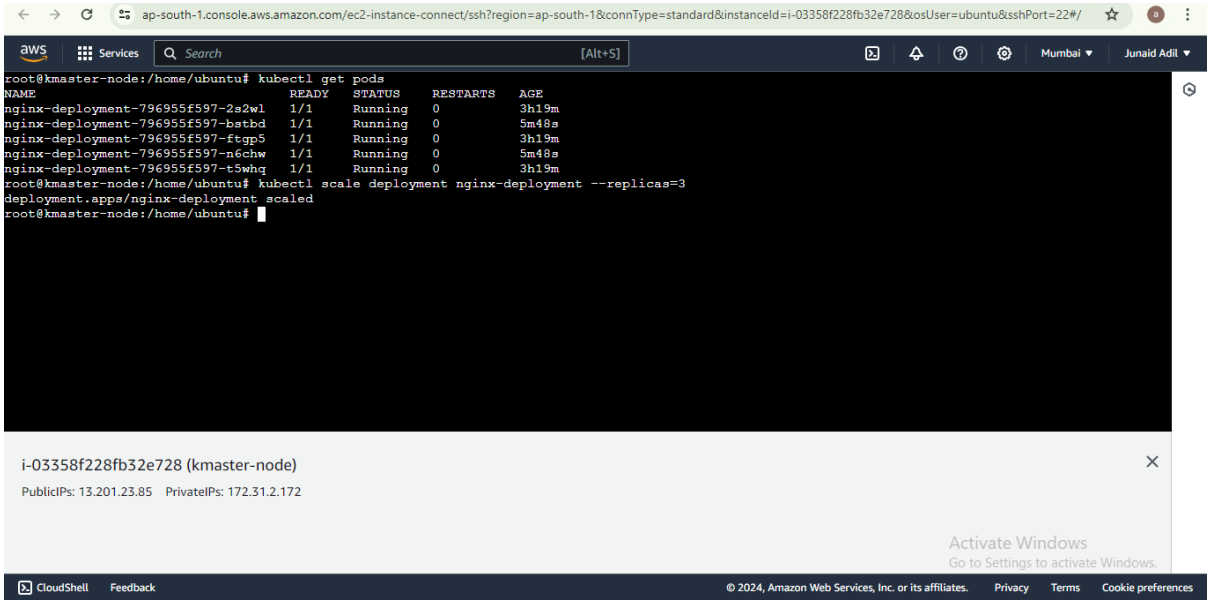


Step 3: We can use the command to scale down:

Syntax “ kubectl scale deployment <deployment-name> --replicas=<desired-replica-count> ”

Use command “ kubectl scale deployment nignx-deploy --replicas=3 ”





We can see the pods are scaled down to 3 pods.

```
root@kmaster-node:/home/ubuntu# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-796955f597-2s2wl   1/1     Running   0          3h19m
nginx-deployment-796955f597-bstbd   1/1     Running   0          5m48s
nginx-deployment-796955f597-ftgp5   1/1     Running   0          3h19m
nginx-deployment-796955f597-n6chw   1/1     Running   0          5m48s
nginx-deployment-796955f597-t5whq   1/1     Running   0          3h19m
root@kmaster-node:/home/ubuntu# kubectl scale deployment nginx-deployment --replicas=3
deployment.apps/nginx-deployment scaled
root@kmaster-node:/home/ubuntu# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-796955f597-bstbd   1/1     Running   0          6m46s
nginx-deployment-796955f597-ftgp5   1/1     Running   0          3h19m
nginx-deployment-796955f597-t5whq   1/1     Running   0          3h19m
root@kmaster-node:/home/ubuntu#
```

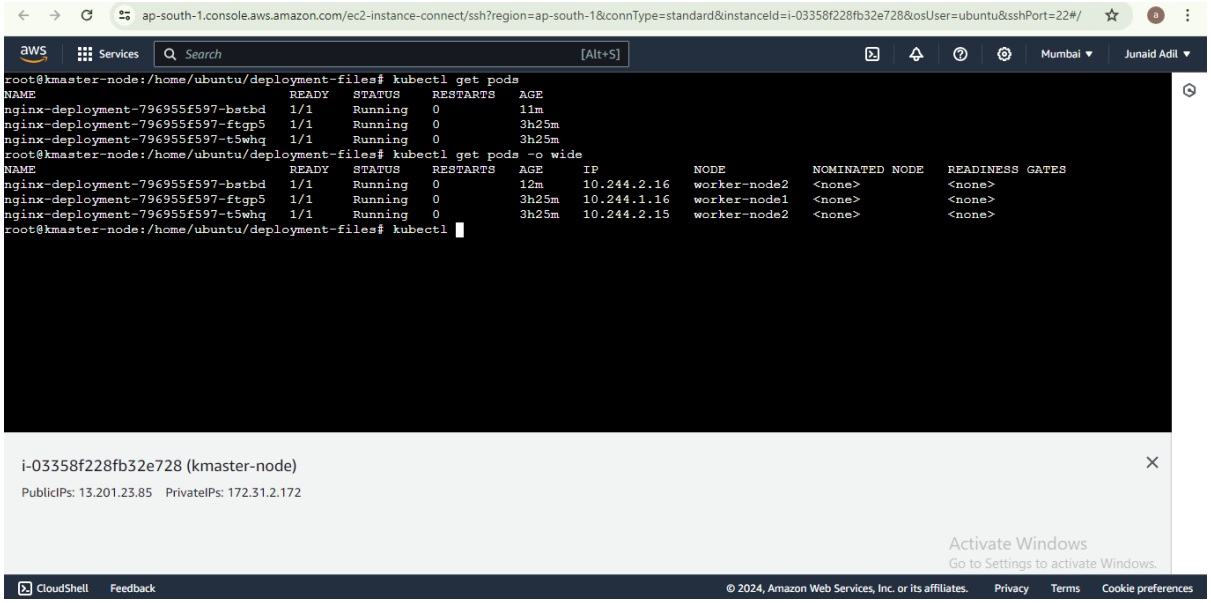
Step 4: We can also modify the number of replicas directly in the Deployment YAML file and then apply the updated configuration.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: junaidadil/junaidnginx:latest
          ports:
            - containerPort: 80
~
"nginx-deploy.yaml" 22L, 379B                                   8,13          All
```

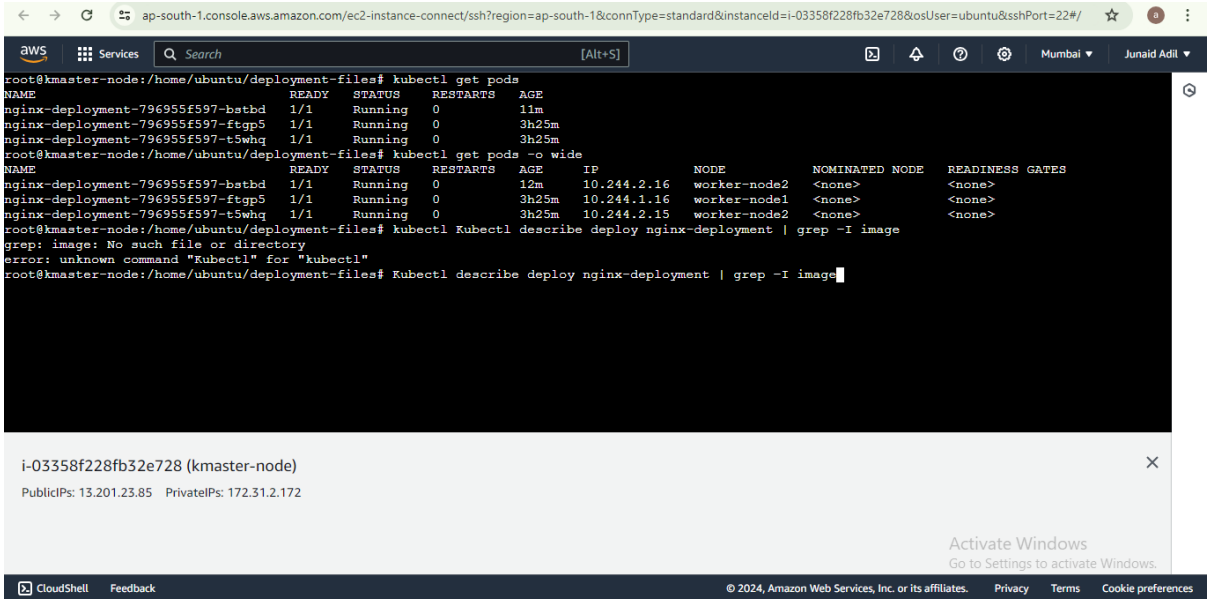
We can update the replicas required in the Deployment.yaml file & save the file. Then apply the deployment configuration using command “kubectl apply -f nginx-deployment.yaml”.

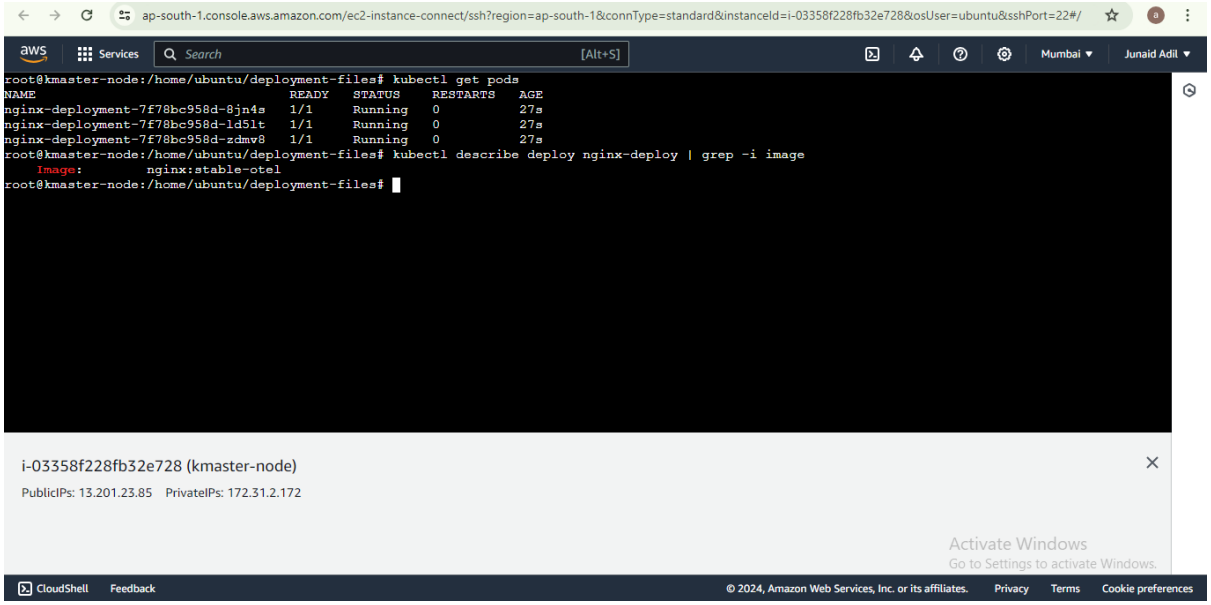
L3 - Implement Rolling-Update Strategy to Upgrade the Application Image from V1.0 to V1.1

Step 1: For Rolling-Update Strategy we need to identify the version of Image. Depending on that we can update the version of Image.



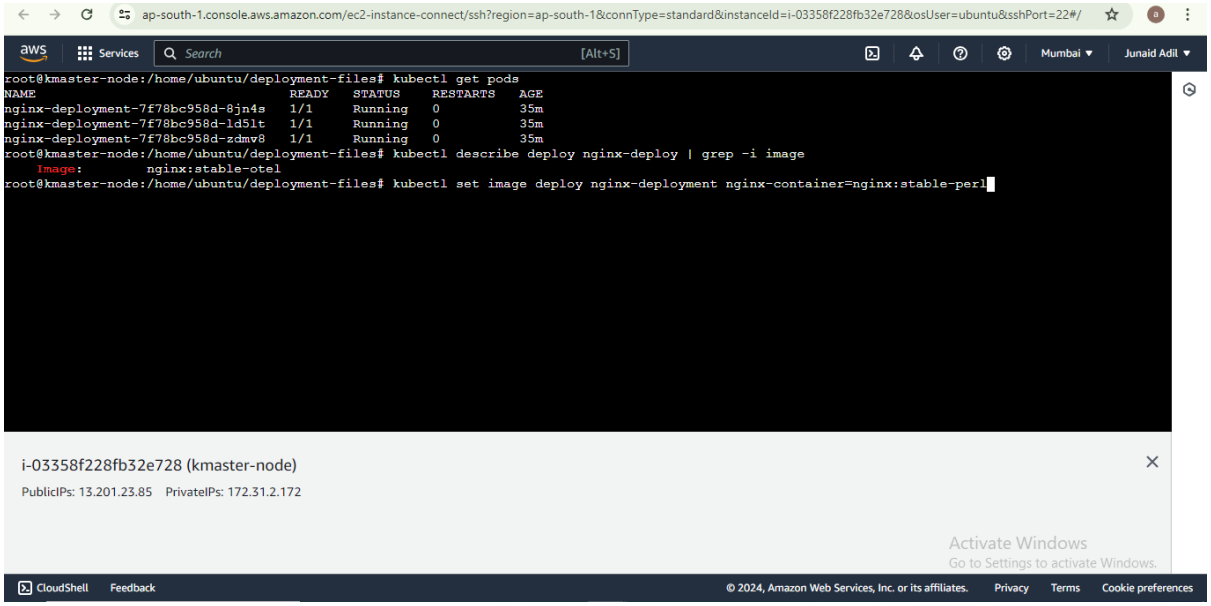
Step 2: Execute command “Kubectl describe deploy nginx-deployment | grep –I image “ to check the version of Image.

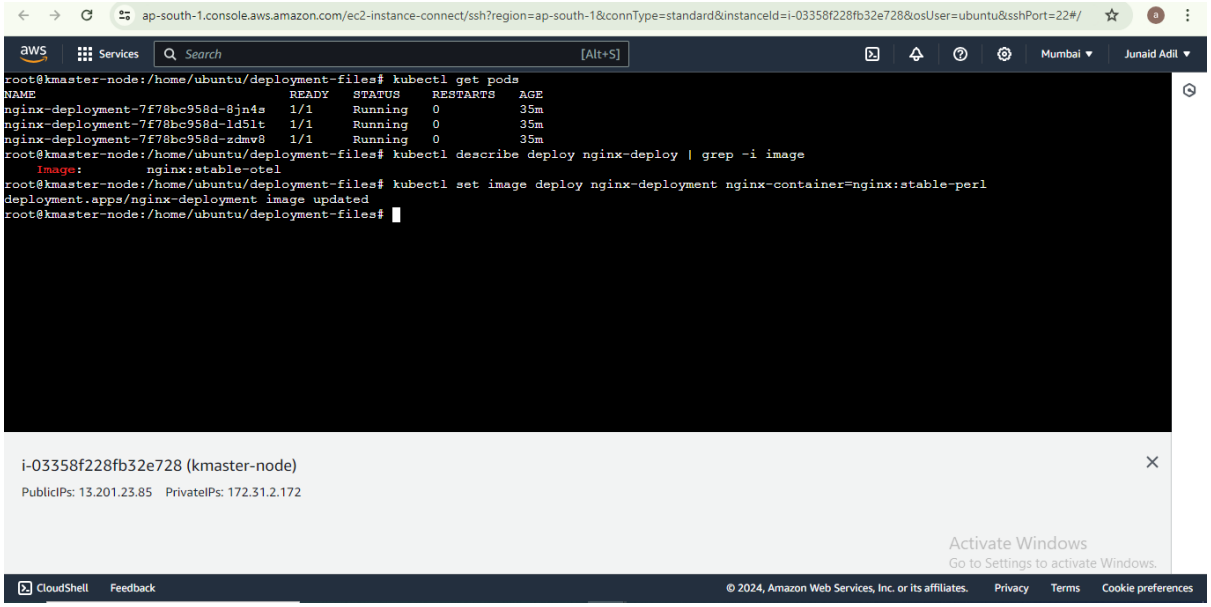




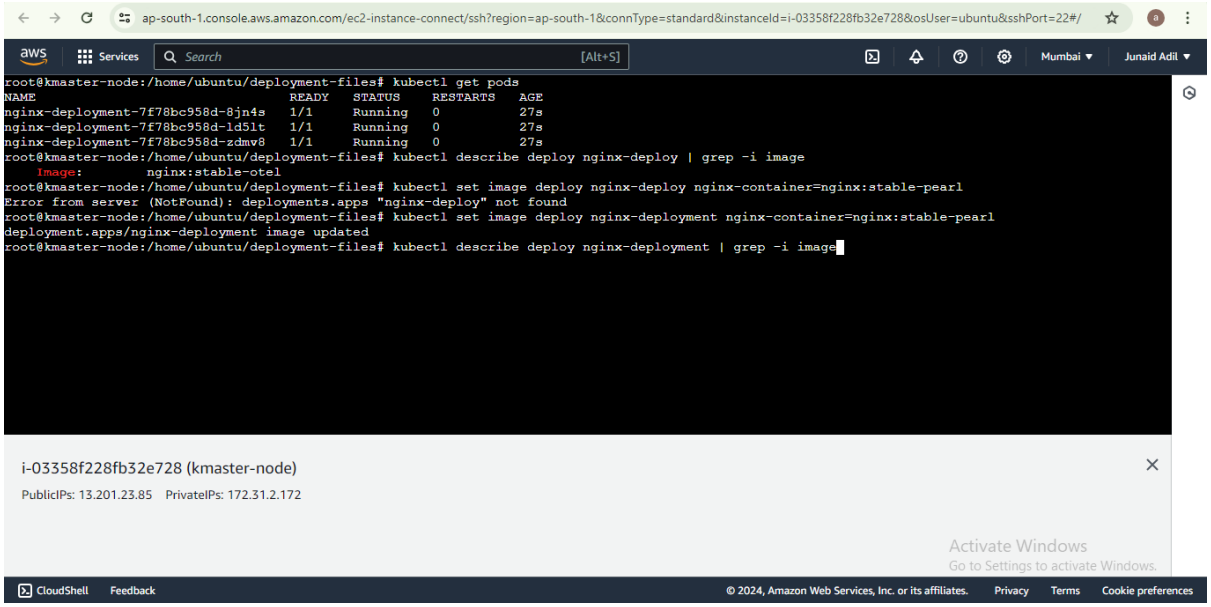
We an see It is working on version “nginx:stable-otel”.

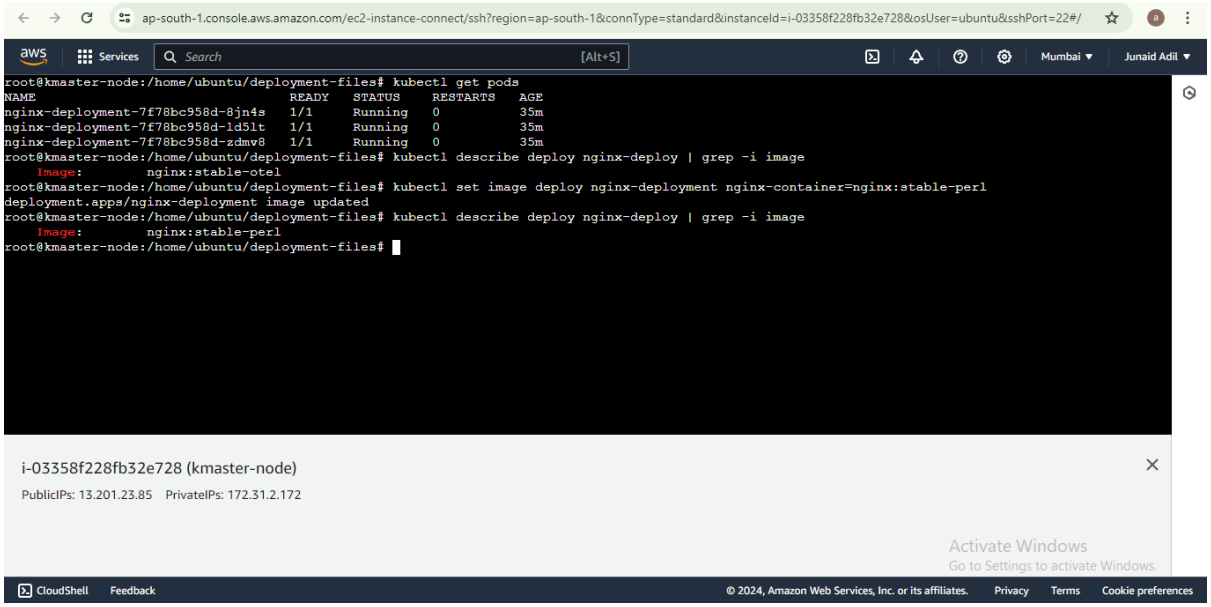
<u>Step 3:</u> To Upgrade the Image, Execute command “kubectl set image deploy nginx-deployment nginx-container=nginx:stable-perl “





Step 4: Now execute command “kubectl describe deploy nginx-deploy | grep -i image “ to check if the image has been upgraded





We can see the Image version has been upgraded from “nginx:stable-otel" to “nginx:stable-perl”