Case Study 02

Kubernetes Application Deployment

Name: Naikwadi Yash Shivdas
Class: D15B
Roll no.: 42

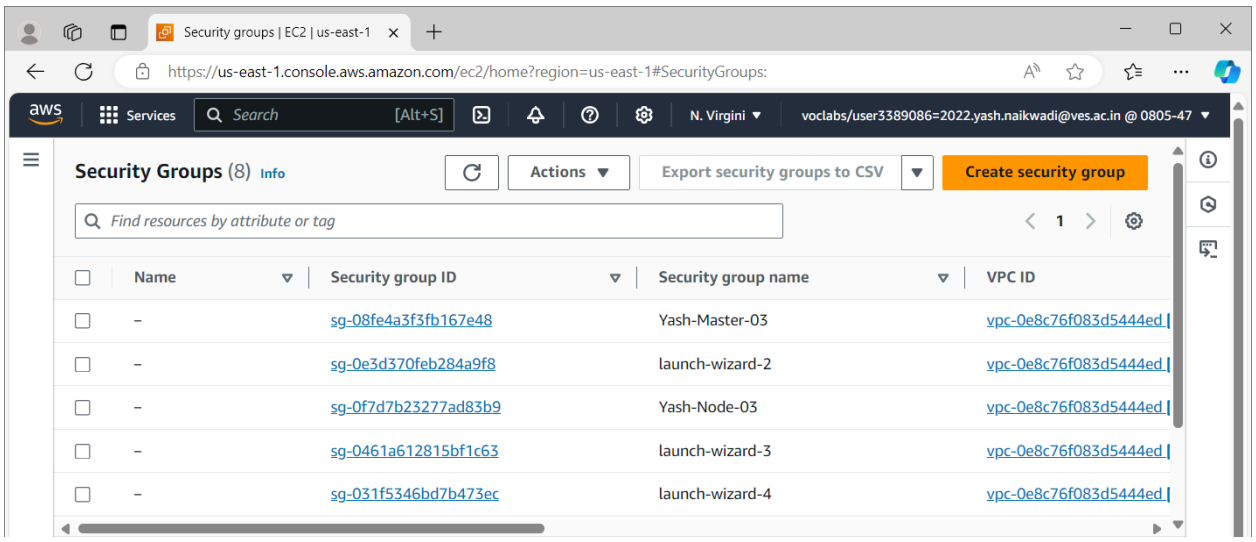
- **Concepts Used**: Kubernetes, AWS Cloud9 IDE, and Kubectl.
- **Problem Statement**: "Set up a Kubernetes cluster on AWS using the Cloud9 IDE. Deploy a sample application using kubectl and ensure it runs successfully."
- **Tasks**:
    - Install and configure kubectl using AWS Cloud9 IDE.
    - Deploy a sample application (like a simple Nginx server) on the Kubernetes cluster.
    - Verify the application deployment by accessing it through a NodePort or LoadBalancer.

SOLUTION

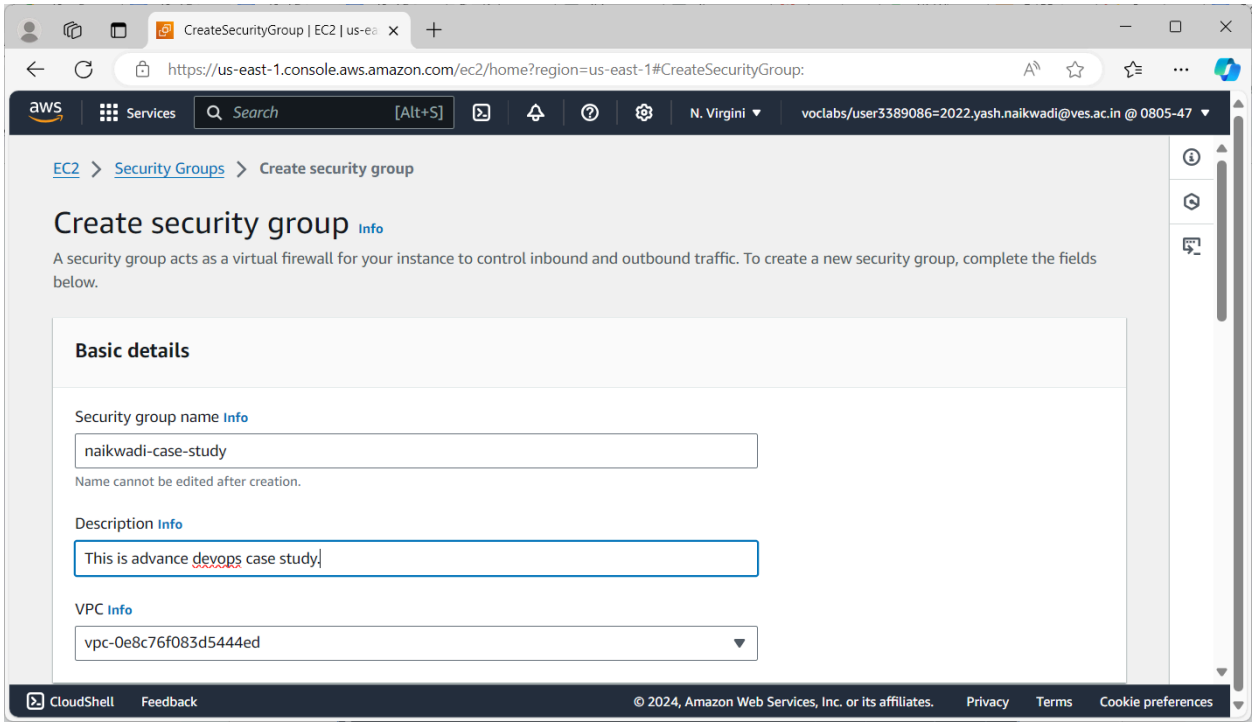
Ensure instances have the appropriate security groups allowing traffic on ports 6443 (Kubernetes API), 10250 (kubelet), and other necessary ports like 80 (HTTP) and 443 (HTTPS) for LoadBalancer.

Create Security Groups for Instances

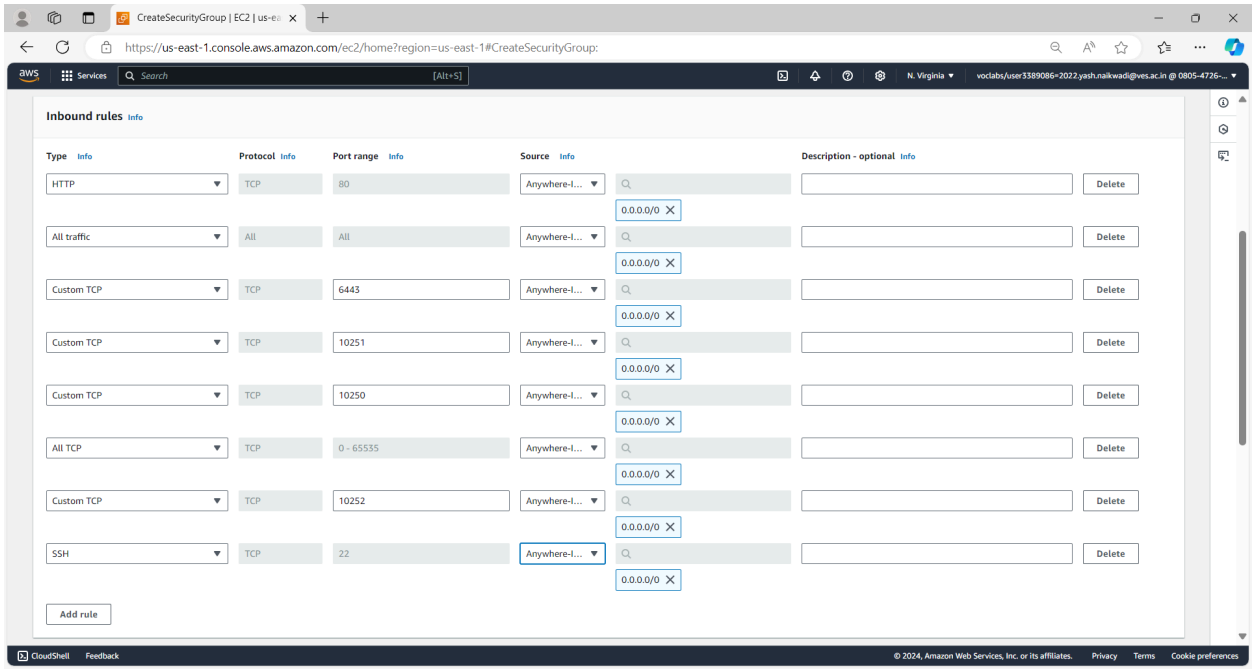
Log in to your **AWS account** and navigate to Security Groups (in EC2).

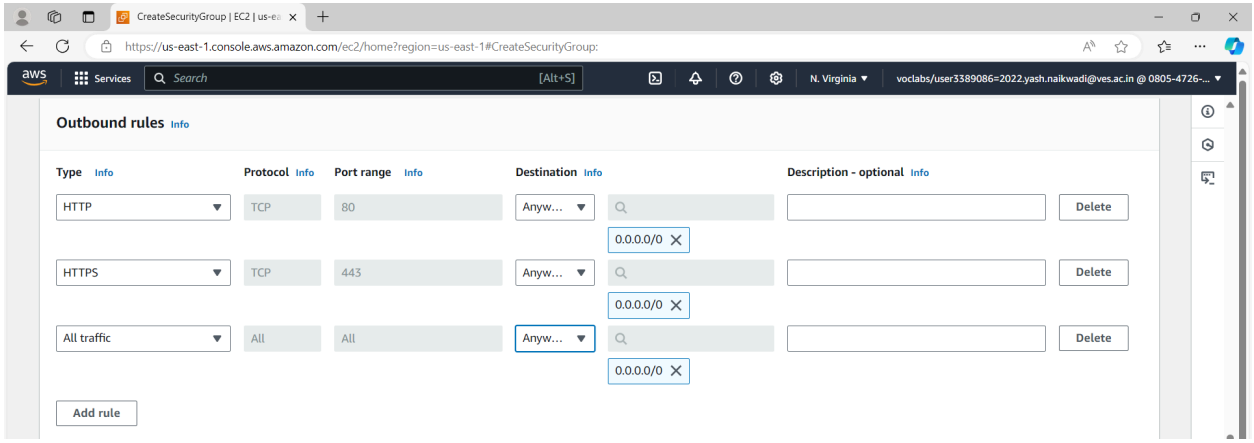


Click on create security group and give the name and description to the security group.

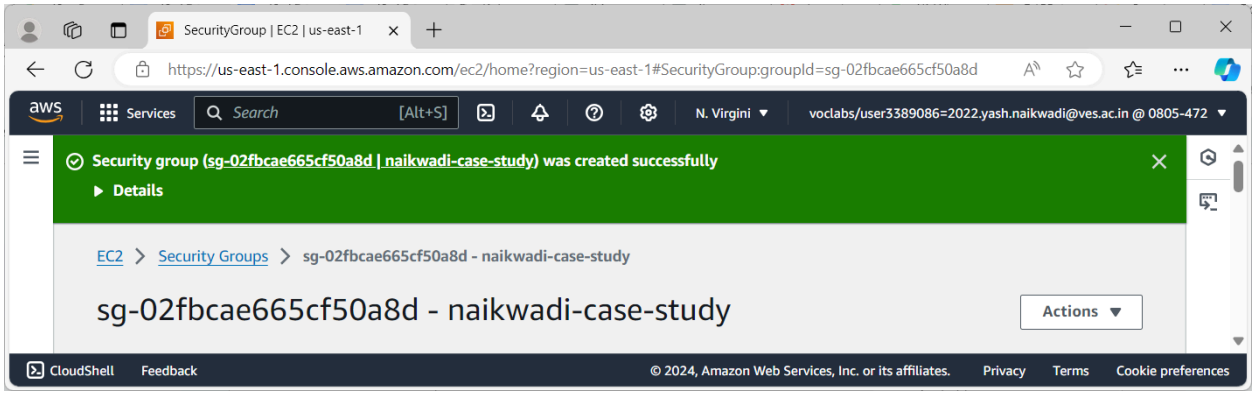


Add the following inbound rules to the security group.

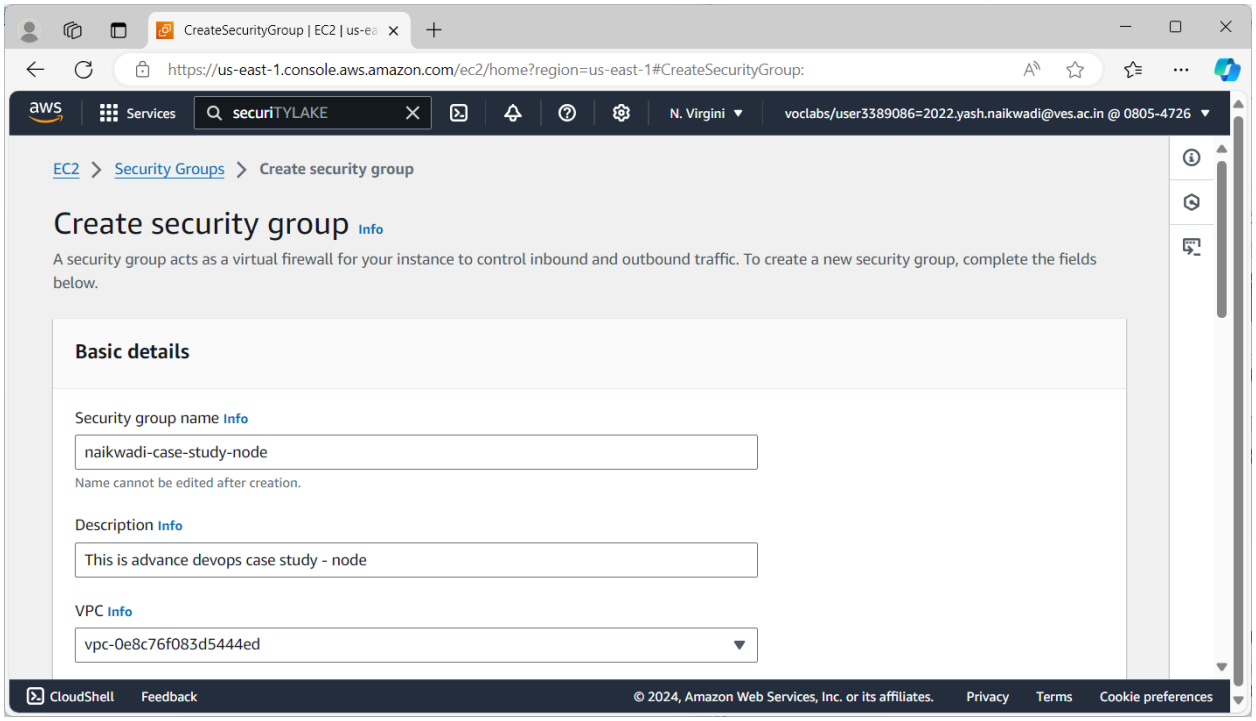




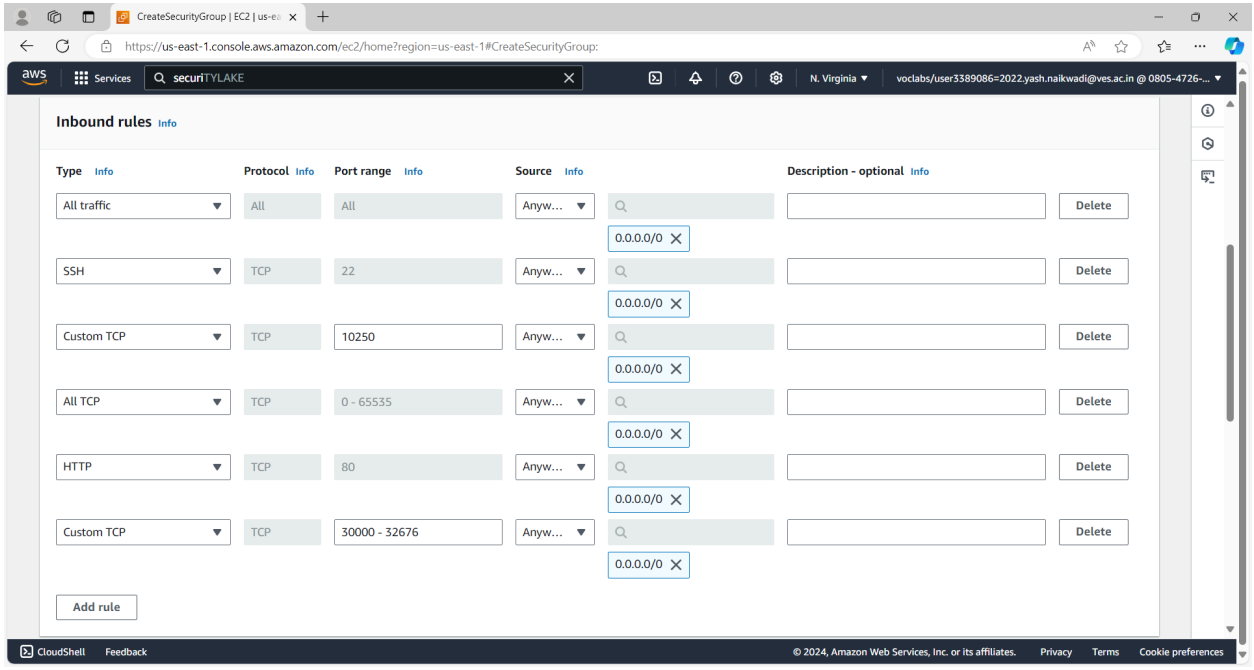
Create a security group.

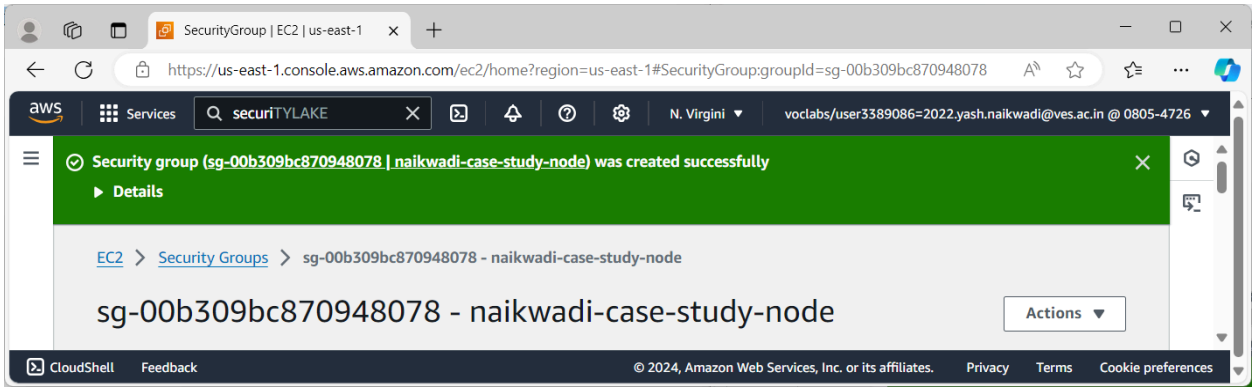


Similarly, create one more security group (for Node).



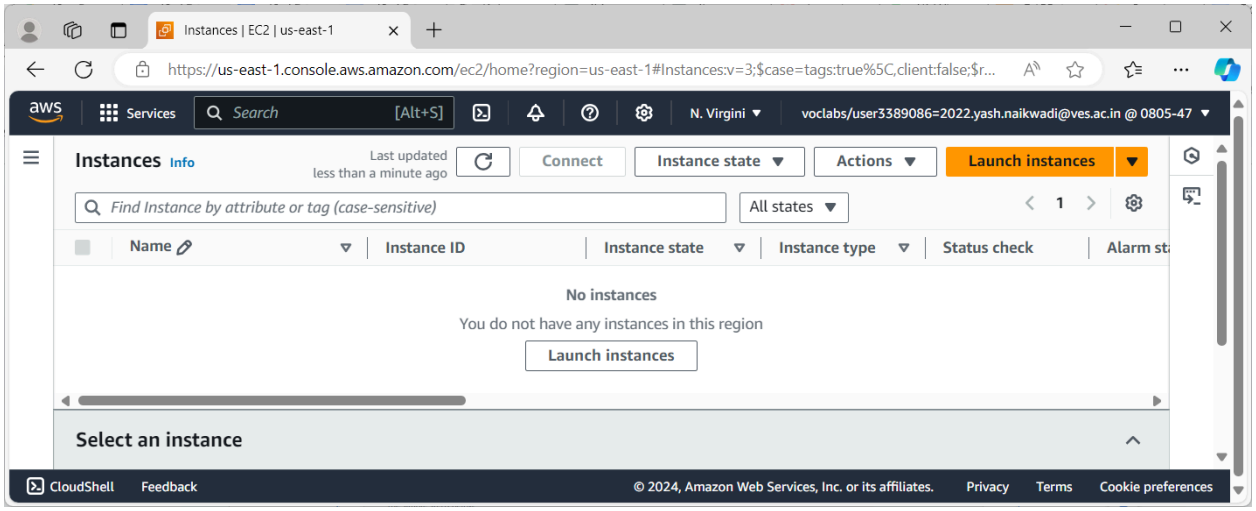
Add the inbound rules for Node security group.





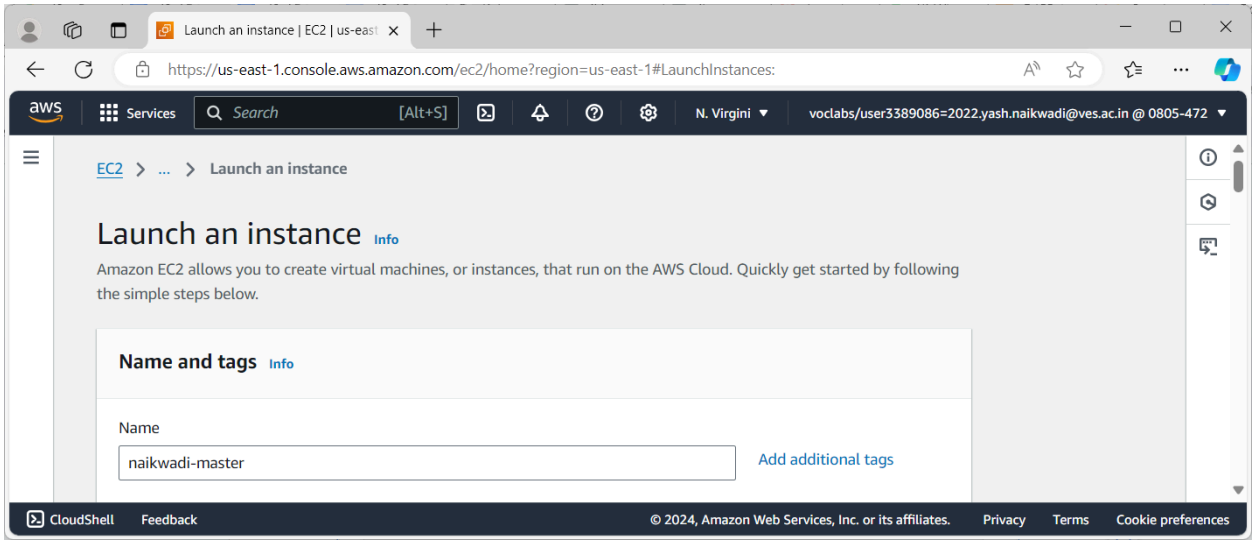
Launch EC2 Instances

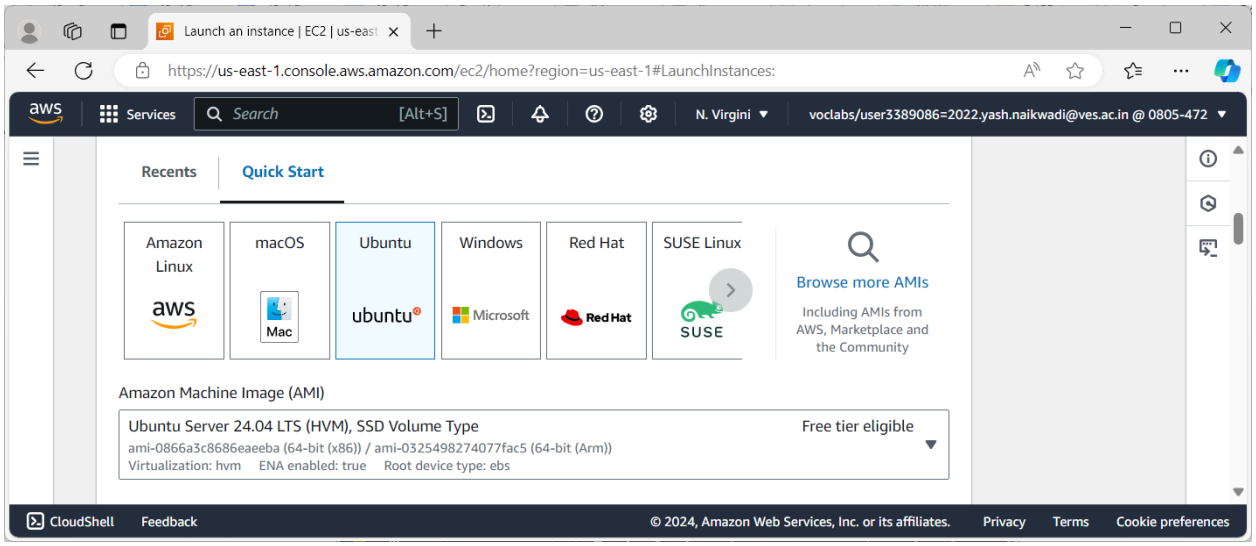
Navigate to **EC2**.



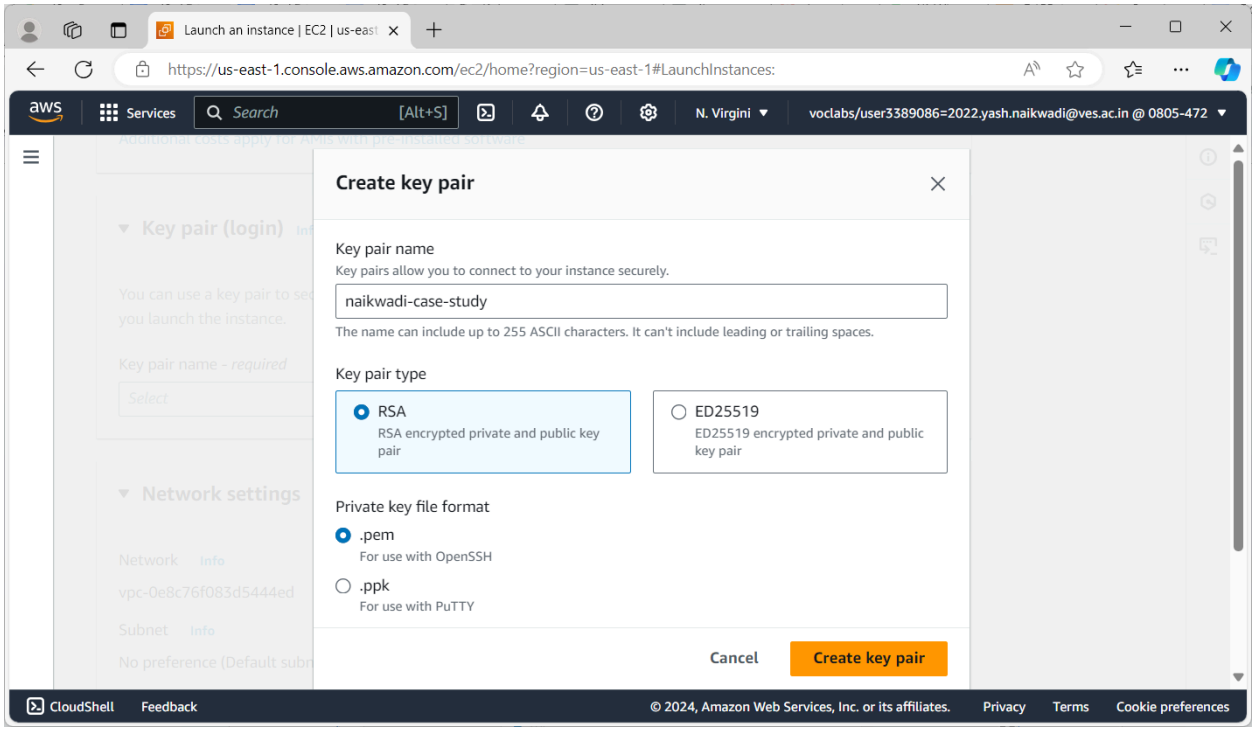
MASTER

Launch **a new EC2 instance for master** with the following settings.

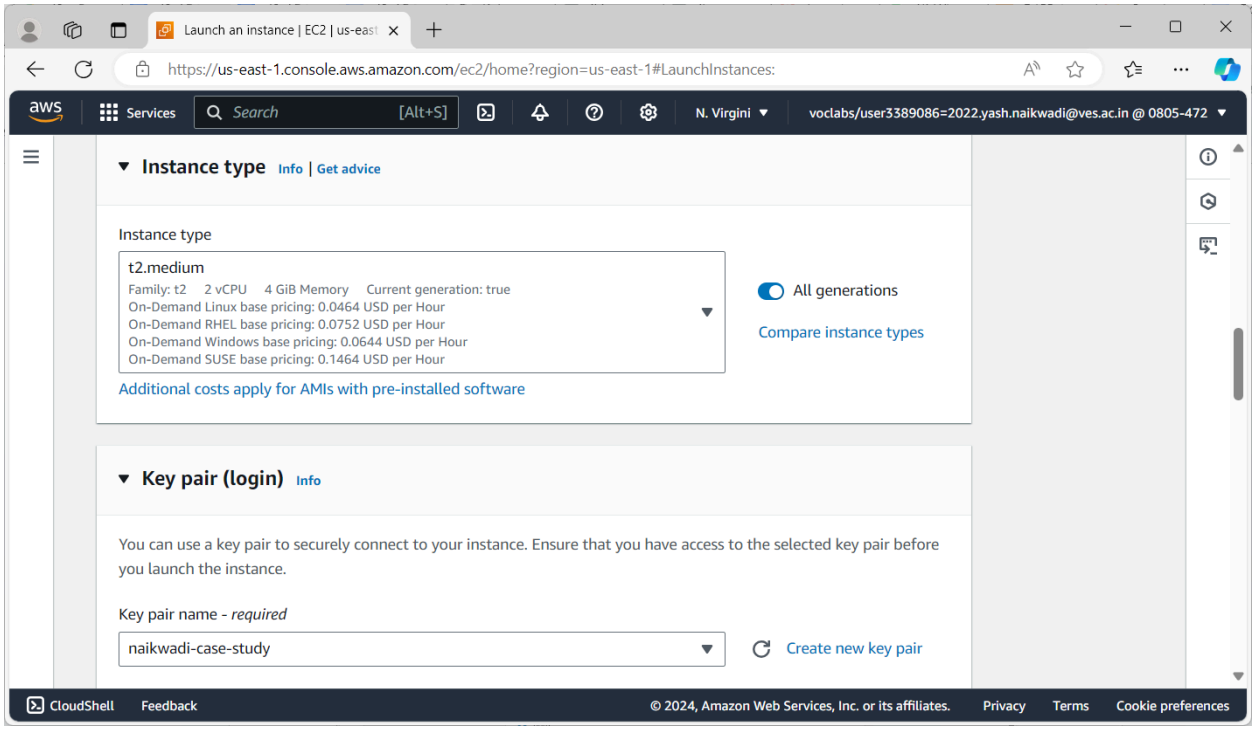




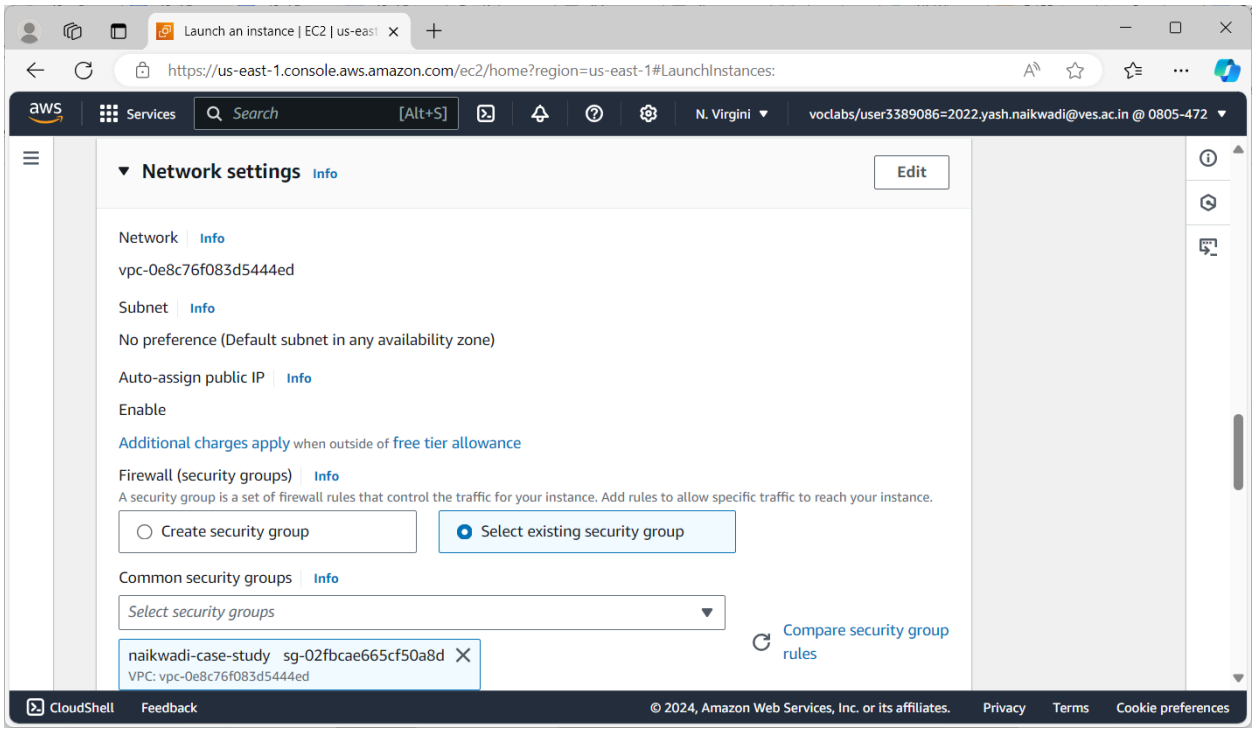
Generate the .pem file (or key-value pair) once while configuring the master ec2.



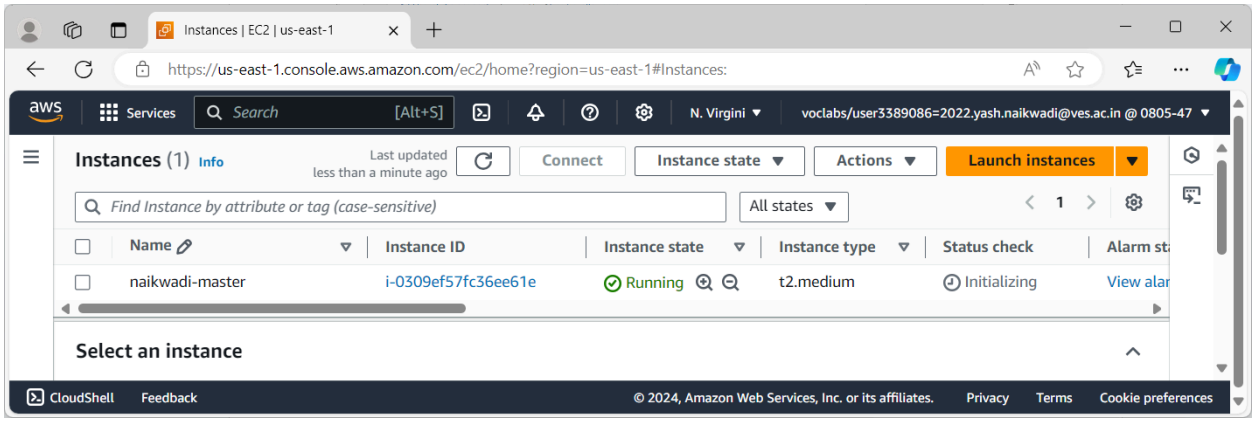
Keep the instance type as t2.medium



Select the existing security group under the network setting tab in which you have to select the group of master which was made in the beginning of the experiment.

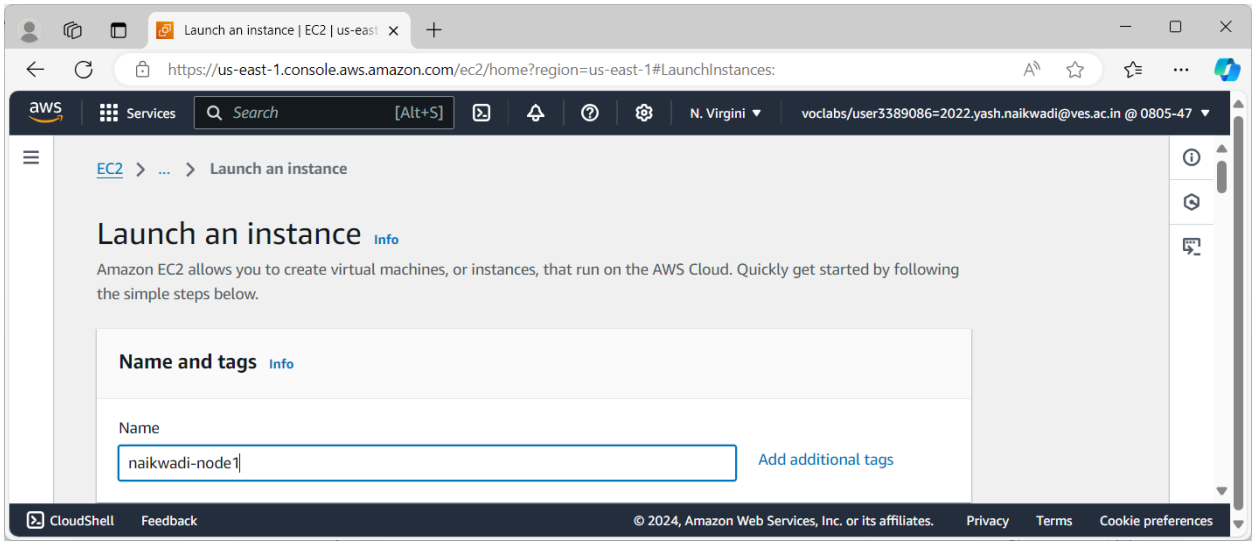


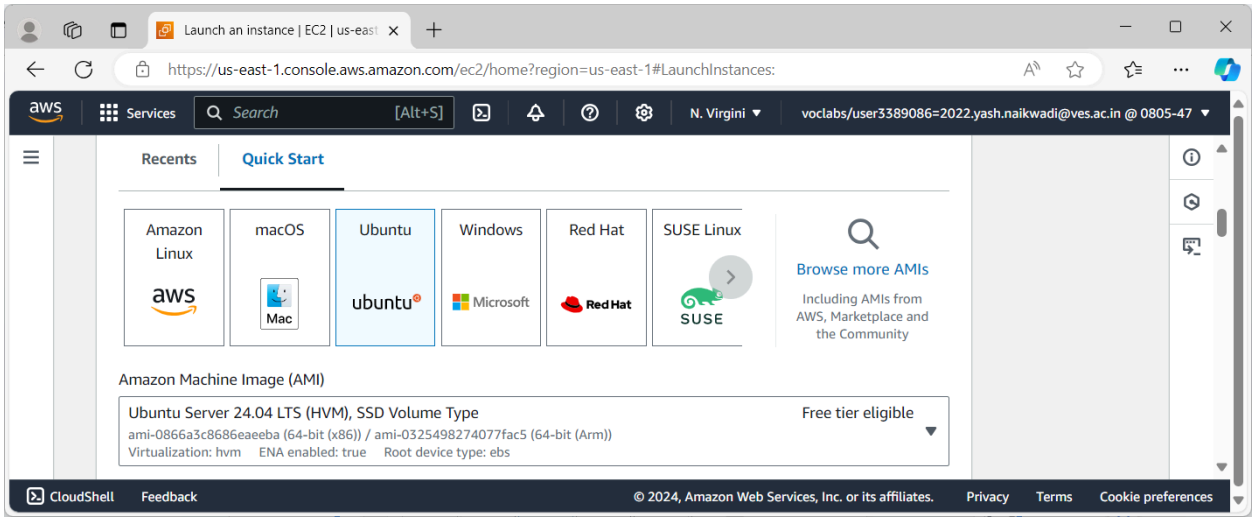
Successfully launched the ec2 instance.



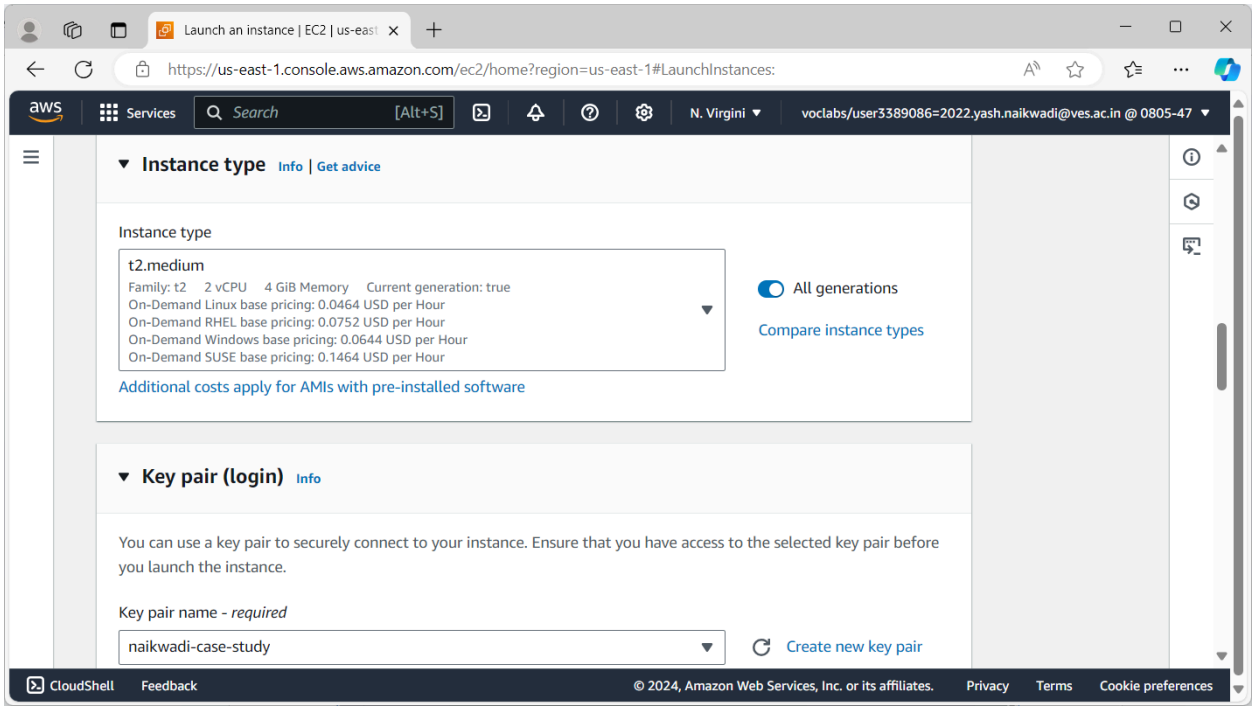
NODE

Similarly, launch 2 new ec2 instances for nodes.

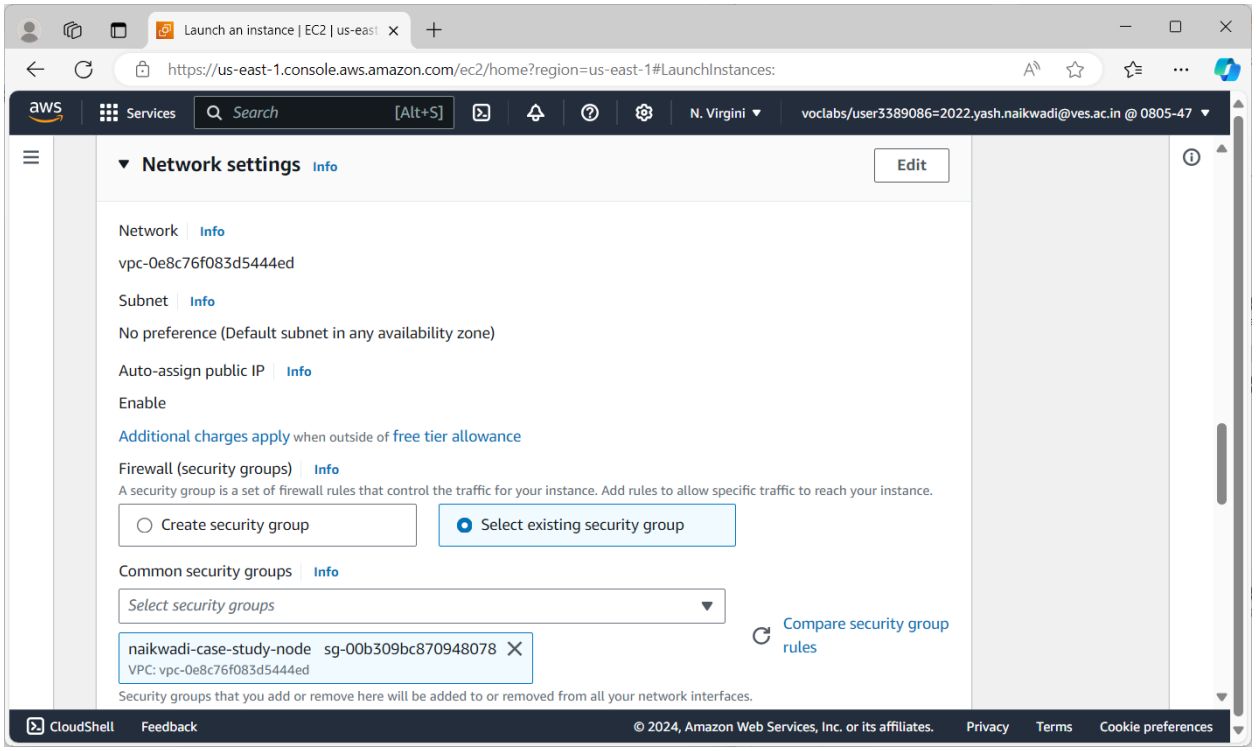




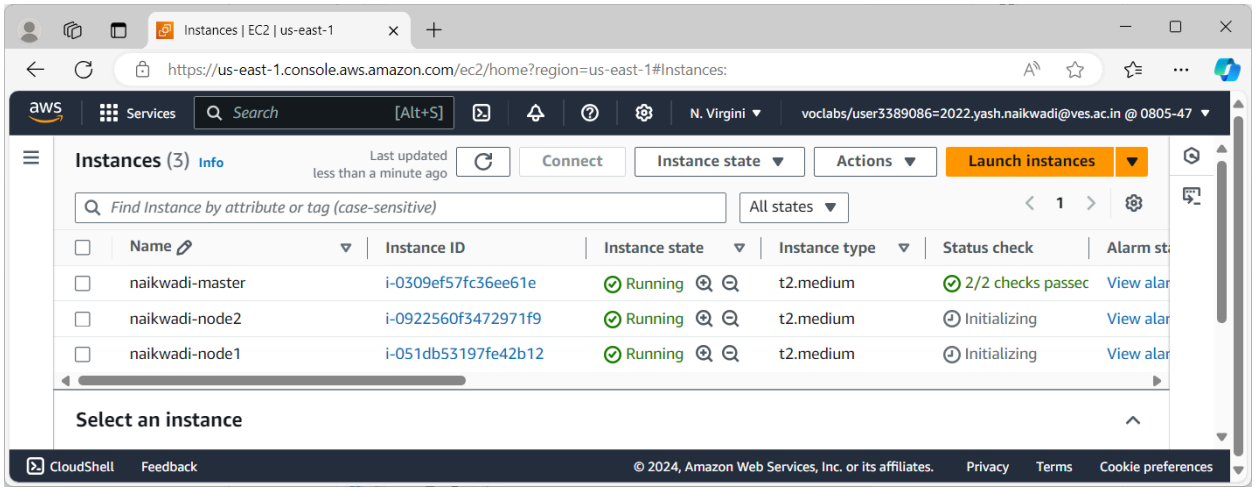
Keep the instance type of node as t2.medium and select the same key-value pair which was created during the master configuration.



Select the existing security group which was made for Node in the beginning of the experiment.

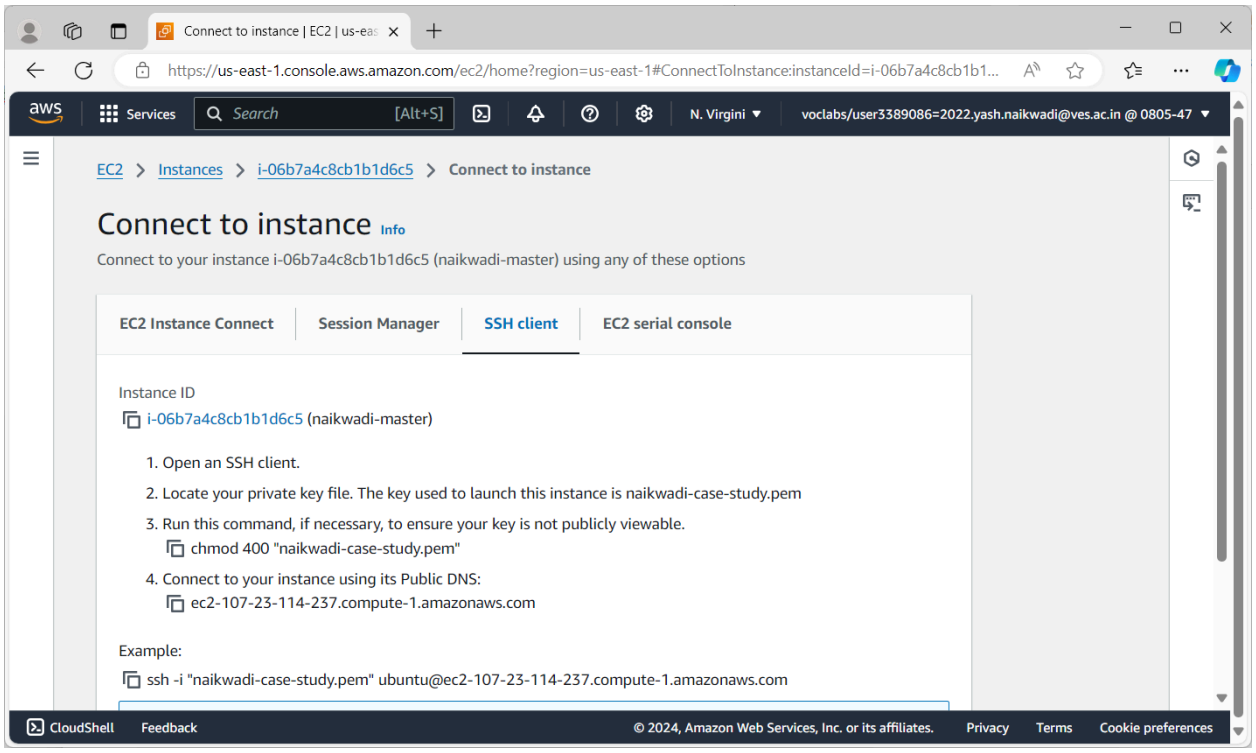


Successfully, created the 3 ec2 instances (1 for master & 2 for nodes).

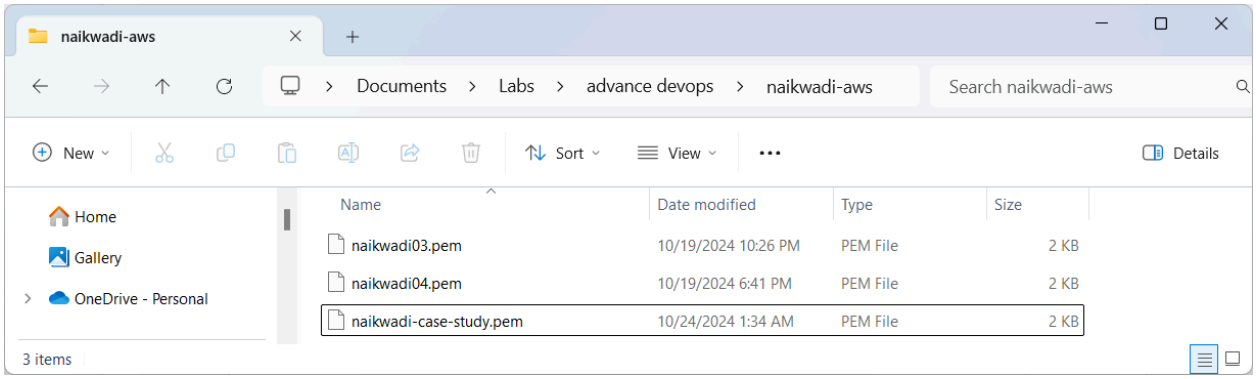


Connect to the Instances

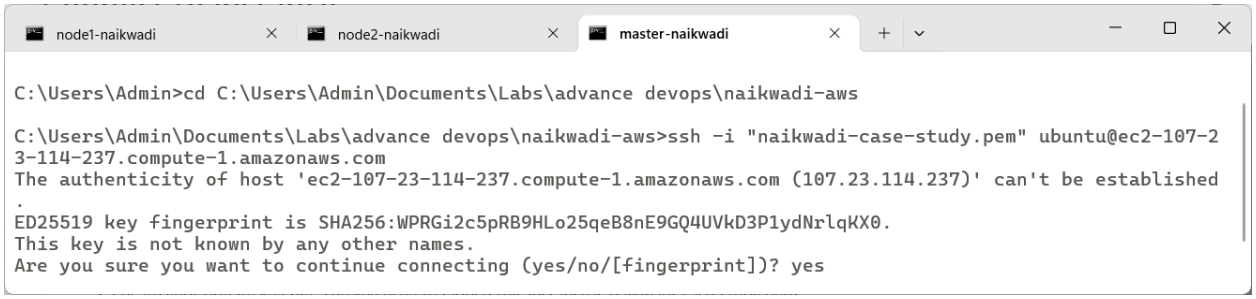
After launching the instances, go to the **EC2 dashboard**, select each instance, and click **Connect** to get the SSH command.

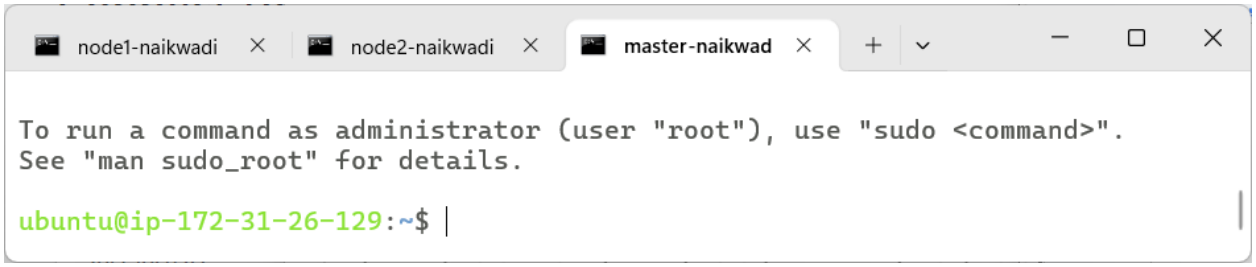


Locate the folder in which the .pem file is placed.

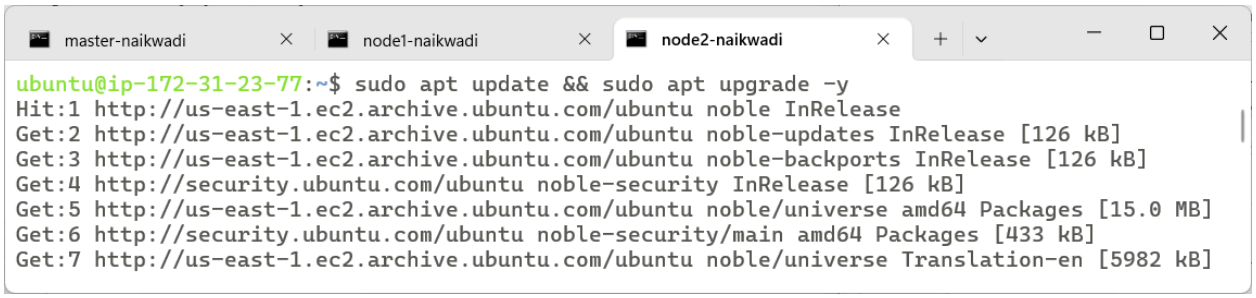


Open your terminal, navigate to the folder where your **.pem key** is stored, and connect to each instance using:
ssh -i "your-key.pem" ubuntu@your-ec2-public-ip

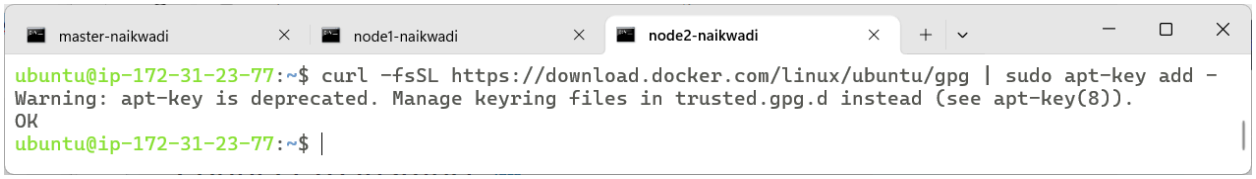




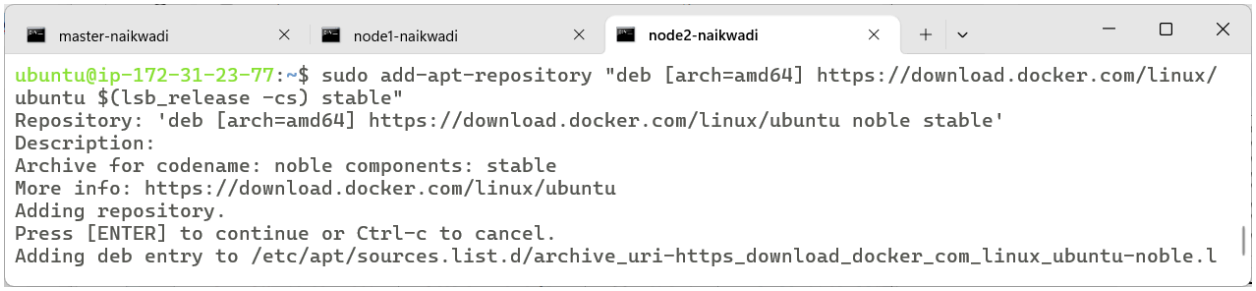
Update the Instances:
On all nodes (master and workers), update the system:
sudo apt update && sudo apt upgrade -y



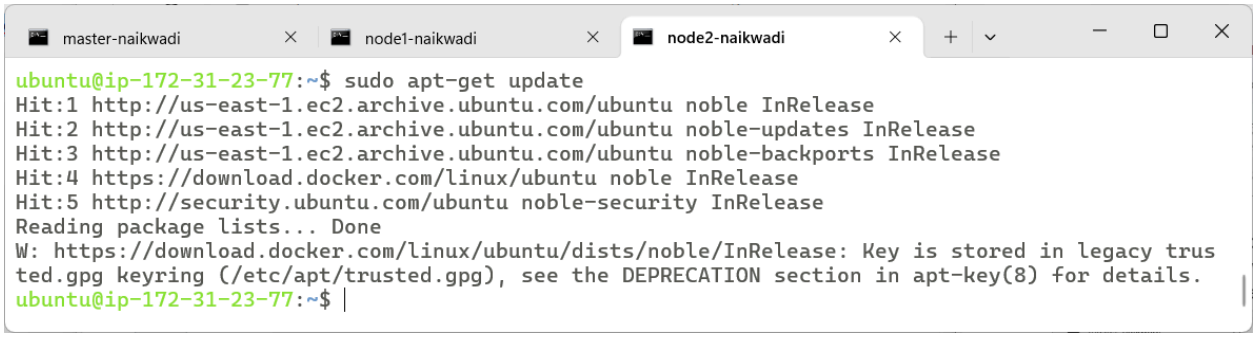
<u>Install Docker on All Instances (Master and Nodes)</u>
On each instance (Master, Node 1, Node 2), run the following commands to install Docker:
Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -



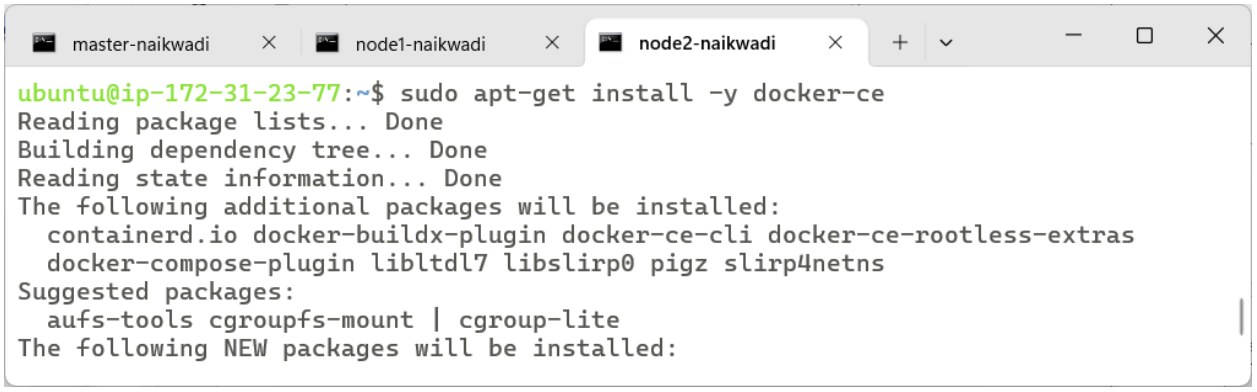
# Add Docker's APT repository
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"



# Update the package index
sudo apt-get update

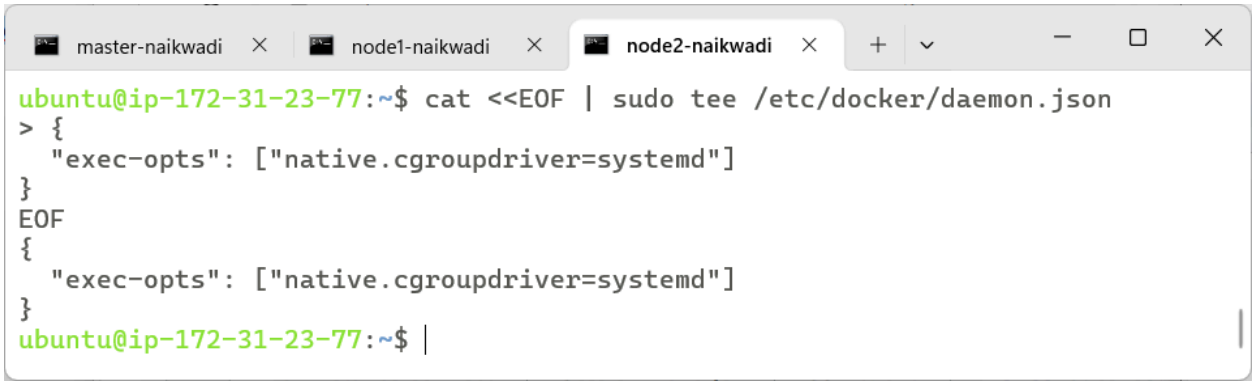


# Install Docker CE (Community Edition)
sudo apt-get install -y docker-ce

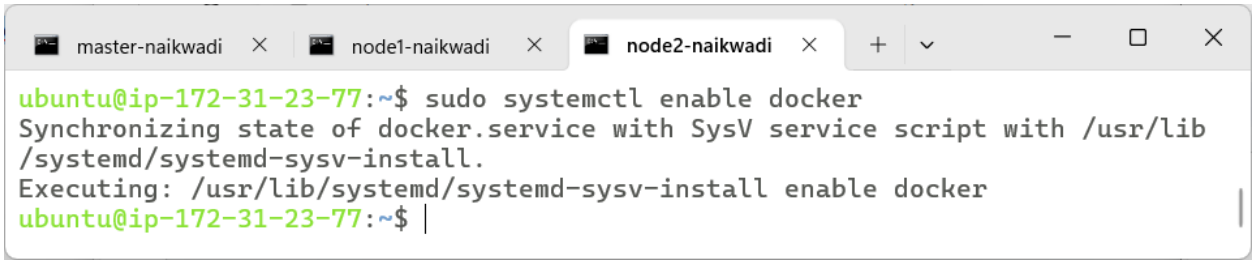


# Configure Docker to use systemd as the cgroup driver
sudo mkdir -p /etc/docker

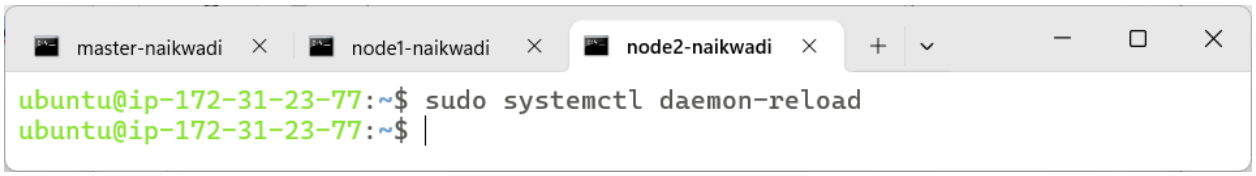
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF



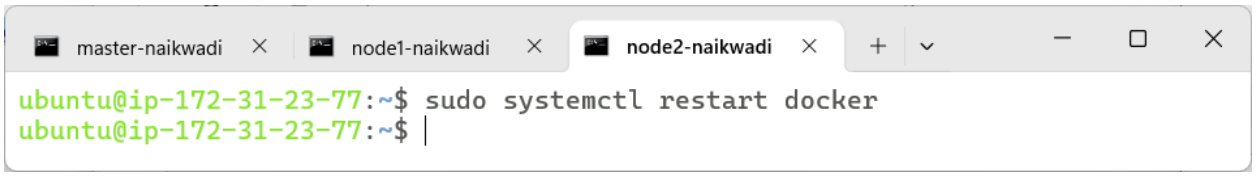
# Enable and start Docker
sudo systemctl enable docker



sudo systemctl daemon-reload



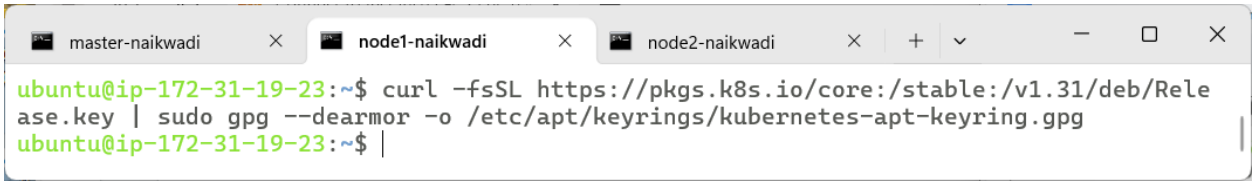
sudo systemctl restart docker



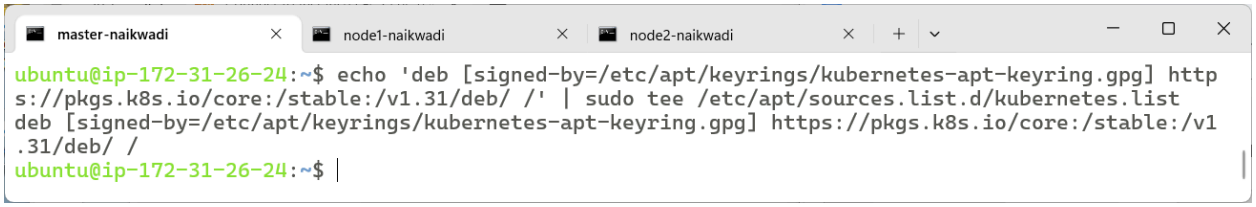
Install Kubernetes on All Instances

Run the below command to install Kubernets.

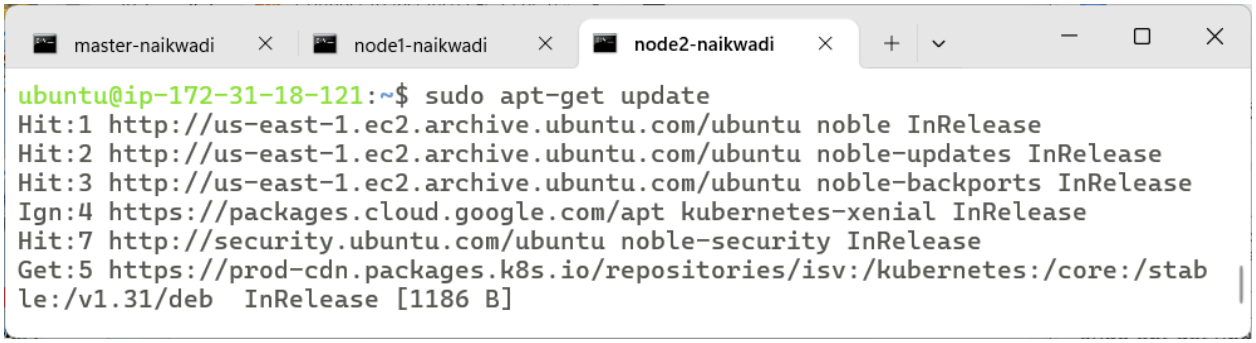
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg



echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list



sudo apt-get update



sudo apt-get install -y kubelet kubeadm kubectl

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-26-24:~$ sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
The following NEW packages will be installed:
```

sudo apt-mark hold kubelet kubeadm kubectl

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-18-121:~$ sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
ubuntu@ip-172-31-18-121:~$
```

sudo systemctl enable --now kubelet

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-26-24:~$ sudo systemctl enable --now kubelet
ubuntu@ip-172-31-26-24:~$
```

sudo apt-get install -y containerd

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-18-121:~$ sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
containerd is already the newest version (1.7.12-0ubuntu4.1).
containerd set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ip-172-31-18-121:~$
```

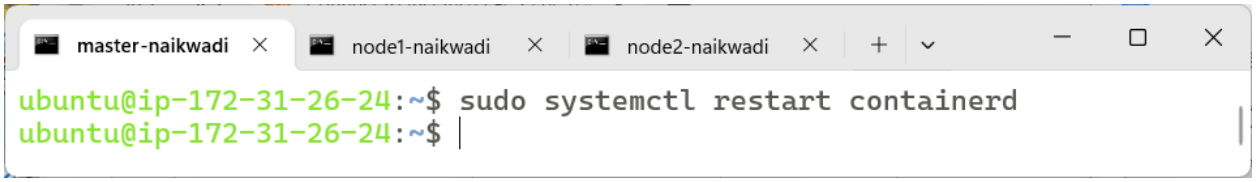
sudo mkdir -p /etc/containerd

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-26-24:~$ sudo mkdir -p /etc/containerd
ubuntu@ip-172-31-26-24:~$
```

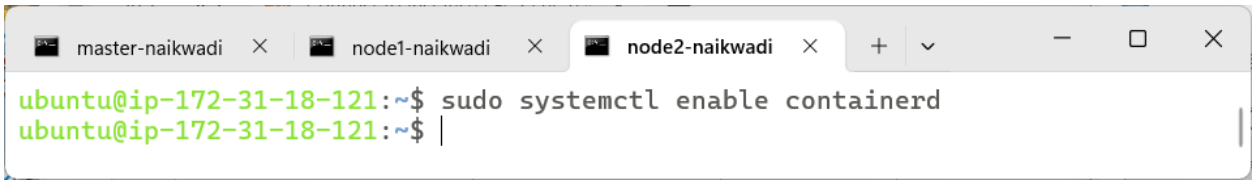
sudo containerd config default | sudo tee /etc/containerd/config.toml

```
master-naikwadi | node1-naikwadi | node2-naikwadi
ubuntu@ip-172-31-18-121:~$ sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2
```

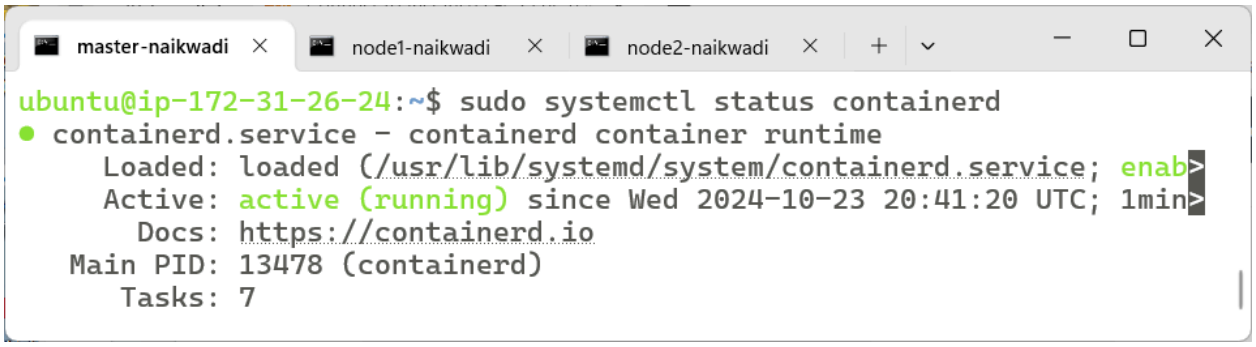
sudo systemctl restart containerd



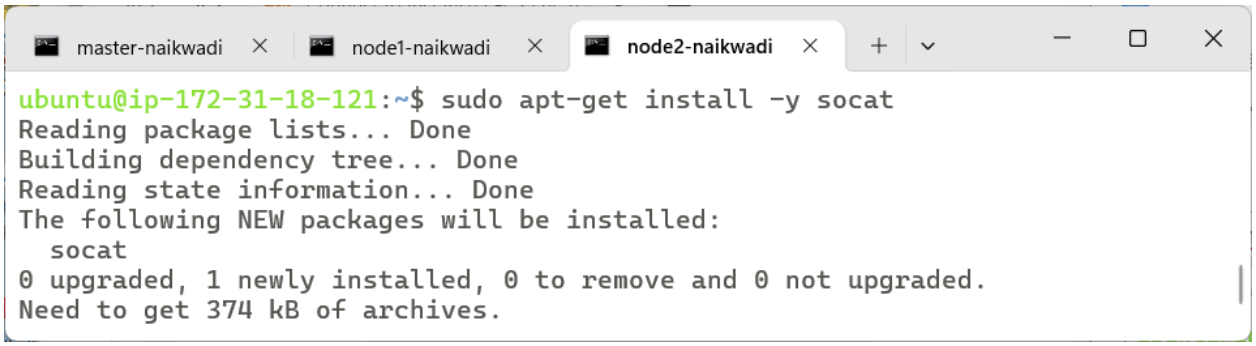
sudo systemctl enable containerd



sudo systemctl status containerd



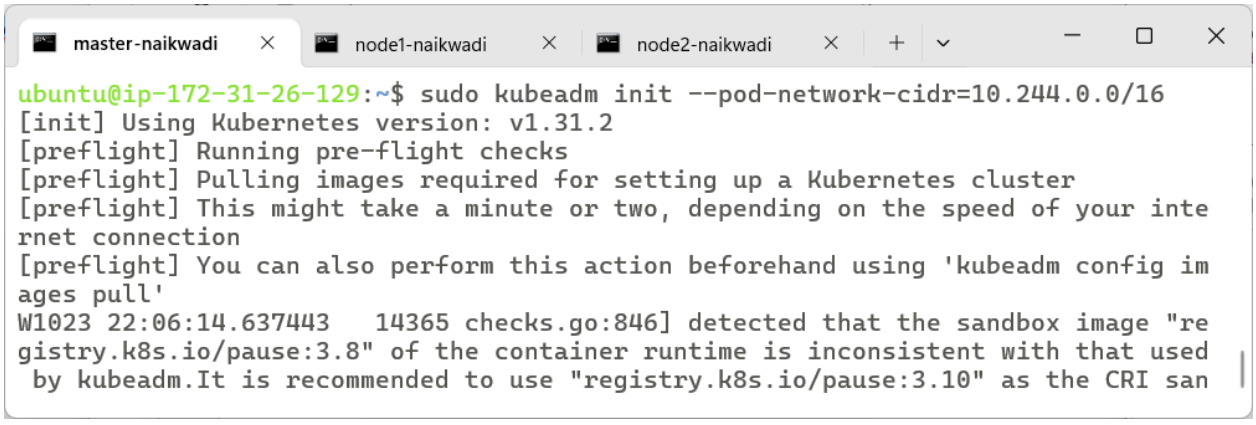
sudo apt-get install -y socat



Initialize the Kubernetes Master Node

Initialize the Kubecluster .Now Perform this Command only for Master.

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

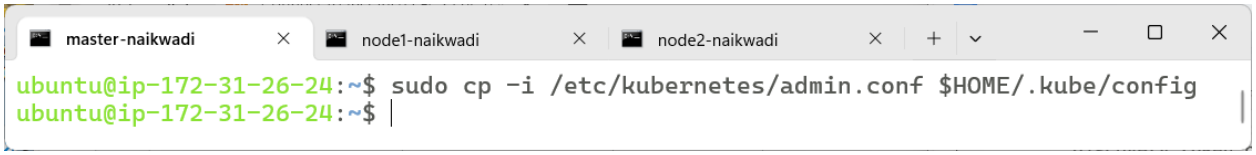


Once the initialization is complete, you will see a command that looks like this:

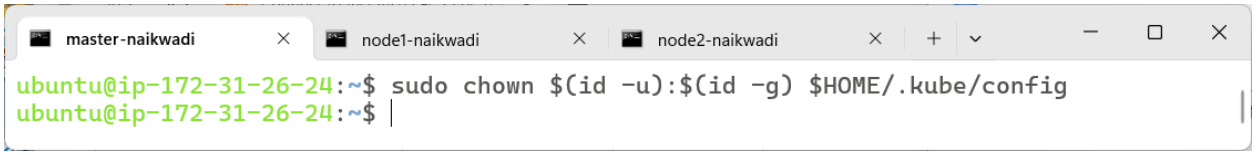
(**Copy and save** this command for later; you will need it to join the worker nodes to the master.)

Next, configure kubectl to interact with the cluster:
mkdir -p $HOME/.kube

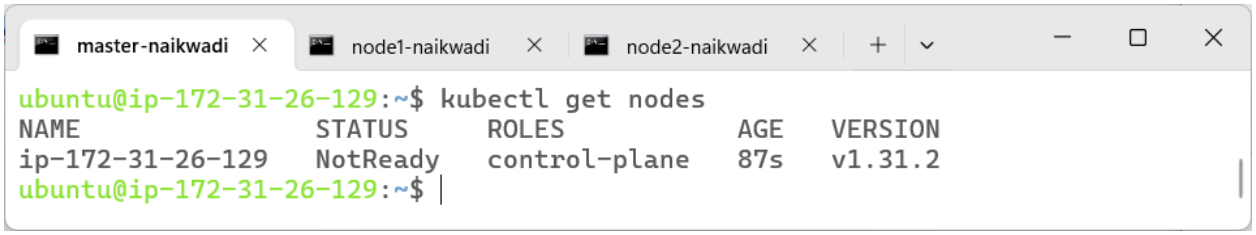
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config



sudo chown $(id -u):$(id -g) $HOME/.kube/config

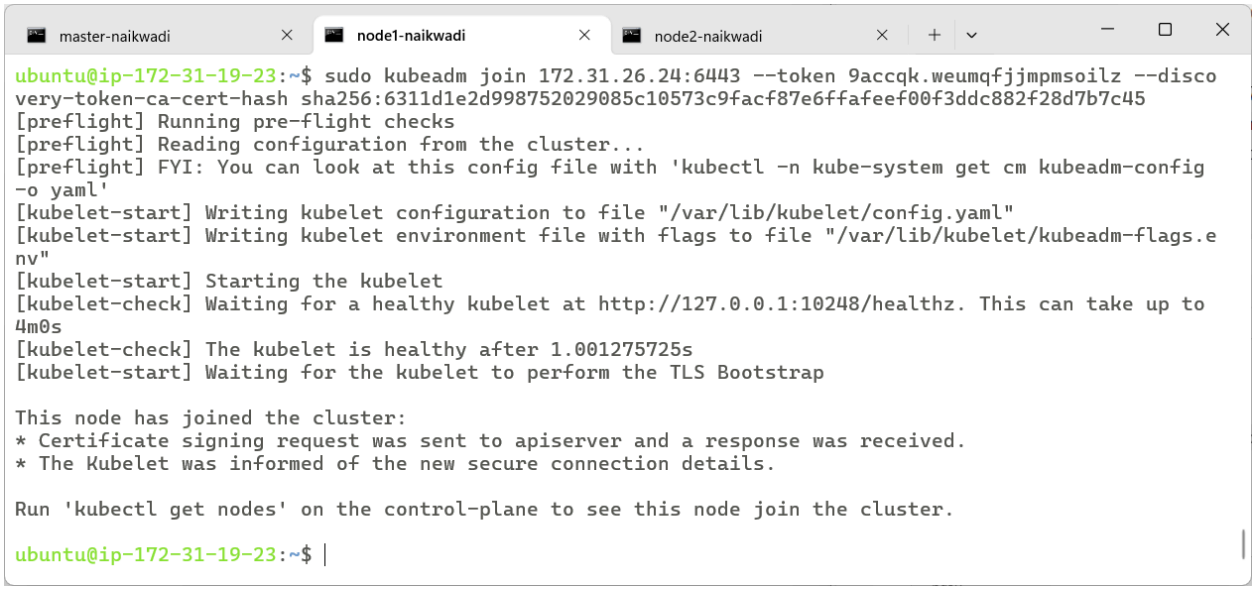


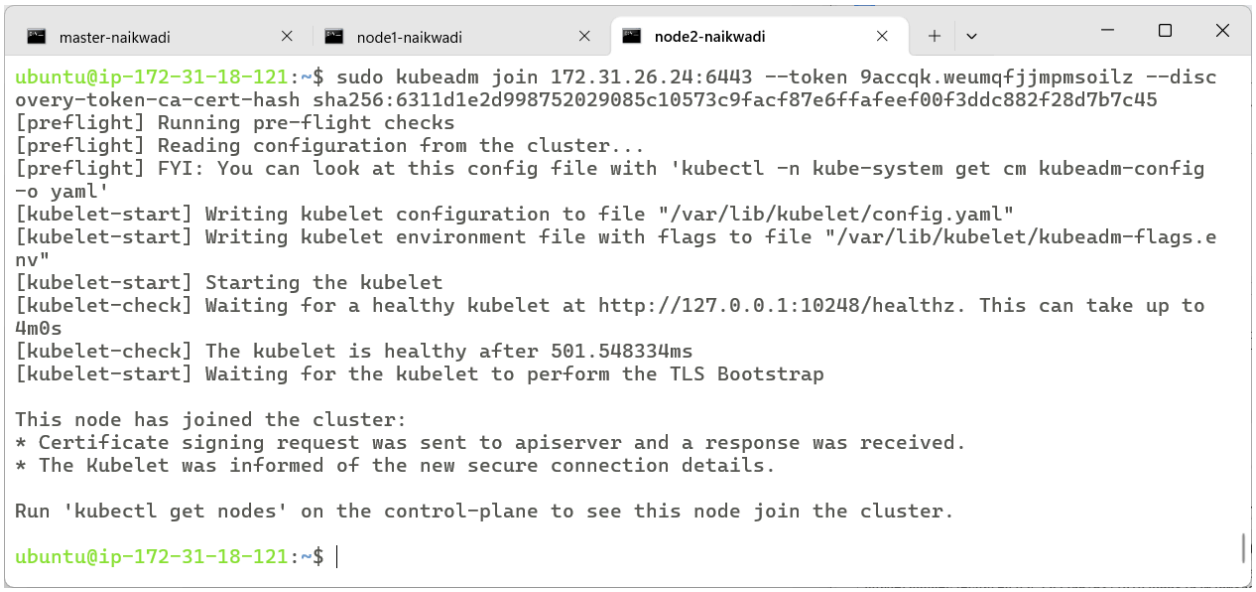
Now Run the command
kubectl get nodes
to see the nodes before executing the Join command on nodes.



Join Worker Nodes to the Cluster

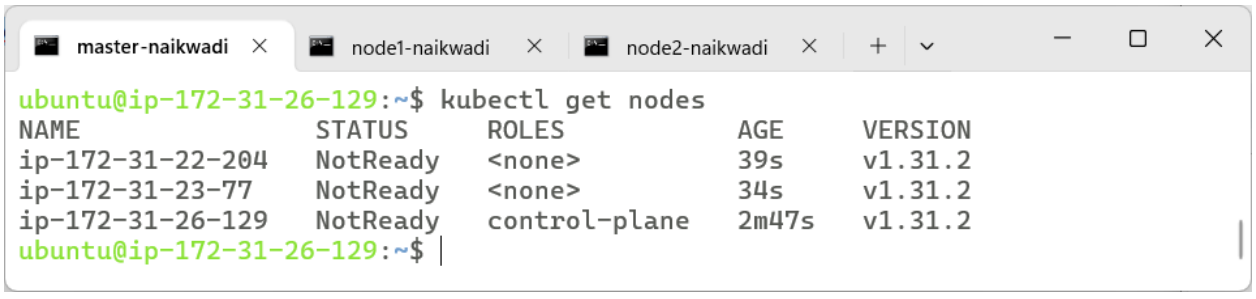
On **Node 1** and **Node 2**, run the kubeadm join command you saved from the master node initialization. It should look like this:
sudo kubeadm join <Master-IP>:6443 --token <token> --discovery-token-ca-cert-hash <hash>
(In my case, command is as follows:
sudo kubeadm join 172.31.26.24:6443 --token 9accqk.weumqfjjmpmsoilz --discovery-token-ca-cert-hash sha256:6311d1e2d998752029085c10573c9facf87e6ffafeef00f3ddc882f28d7b7c45)





Verify the Cluster

On the master node, verify that the nodes have joined the cluster:
kubectl get nodes



Install a Network Plugin (Calico)

Since Status is NotReady we have to add a network plugin. And also we have to give the name to the nodes.
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

```
ubuntu@ip-172-31-26-24:~$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
```

sudo systemctl status kubelet

```
ubuntu@ip-172-31-26-24:~$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Wed 2024-10-23 20:47:48 UTC; 8min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 14570 (kubelet)
lines 1-7...skipping...
```

Now Run command (we can see Status is ready).
kubectl get nodes -o wide

```
ubuntu@ip-172-31-26-129:~$ kubectl get nodes -o wide
NAME              STATUS     ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-22-204  Ready      <none>          76s     v1.31.2   172.31.22.204   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-23-77   Ready      <none>          71s     v1.31.2   172.31.23.77    <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-26-129  NotReady   control-plane   3m24s   v1.31.2   172.31.26.129   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
```

Rename/Label the Nodes

kubectl label node ip-172-31-22-204 kubernetes.io/role=Node1
kubectl label node ip-172-31-23-77 kubernetes.io/role=Node2

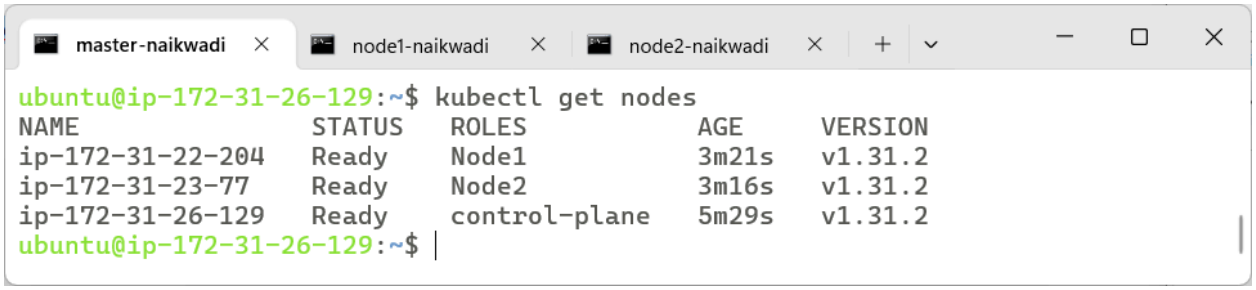
```
ubuntu@ip-172-31-26-24:~$ kubectl label node ip-172-31-18-121 kubernetes.io/role=Node1
node/ip-172-31-18-121 labeled
ubuntu@ip-172-31-26-24:~$ kubectl label node ip-172-31-19-23 kubernetes.io/role=Node2
node/ip-172-31-19-23 labeled
ubuntu@ip-172-31-26-24:~$
```

Check Again

Run (again to confirm all nodes are in the Ready state and properly labeled).
kubectl get nodes -o wide

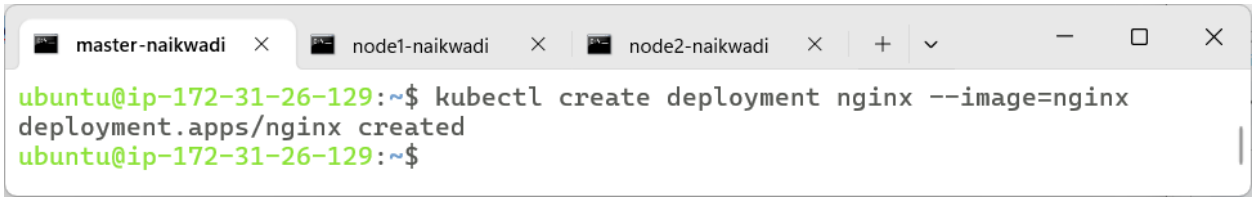
```
ubuntu@ip-172-31-26-129:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-22-204  Ready    Node1           2m37s   v1.31.2   172.31.22.204   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-23-77   Ready    Node2           2m32s   v1.31.2   172.31.23.77    <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-26-129  Ready    control-plane   4m45s   v1.31.2   172.31.26.129   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ubuntu@ip-172-31-26-129:~$
```

Or run
kubectl get nodes



Deploy the Sample Nginx Application

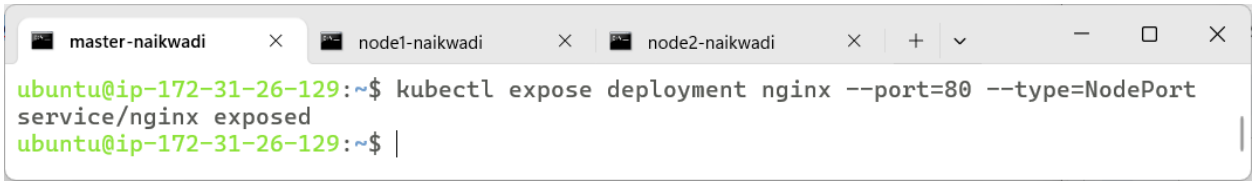
1. **Deploy Nginx**:
   - Create a deployment:
     bash
     Copy code
     kubectl create deployment nginx --image=nginx



Expose the Nginx Deployment:

Expose the deployment via NodePort (or LoadBalancer if using cloud load balancing):

kubectl expose deployment nginx --port=80 --type=NodePort

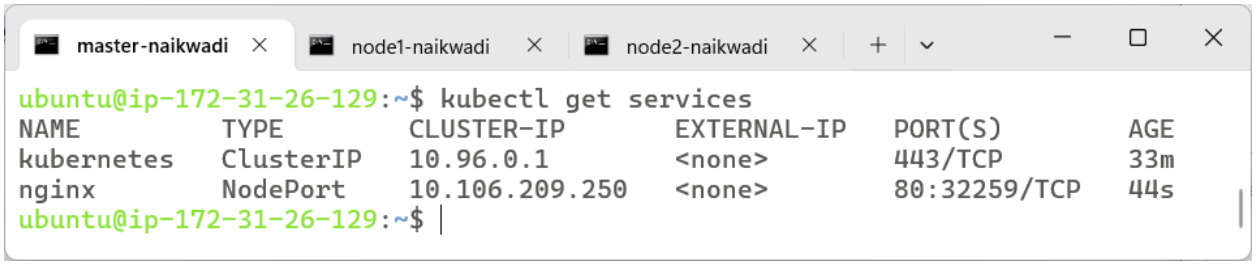


Get the NodePort:

To get the NodePort:

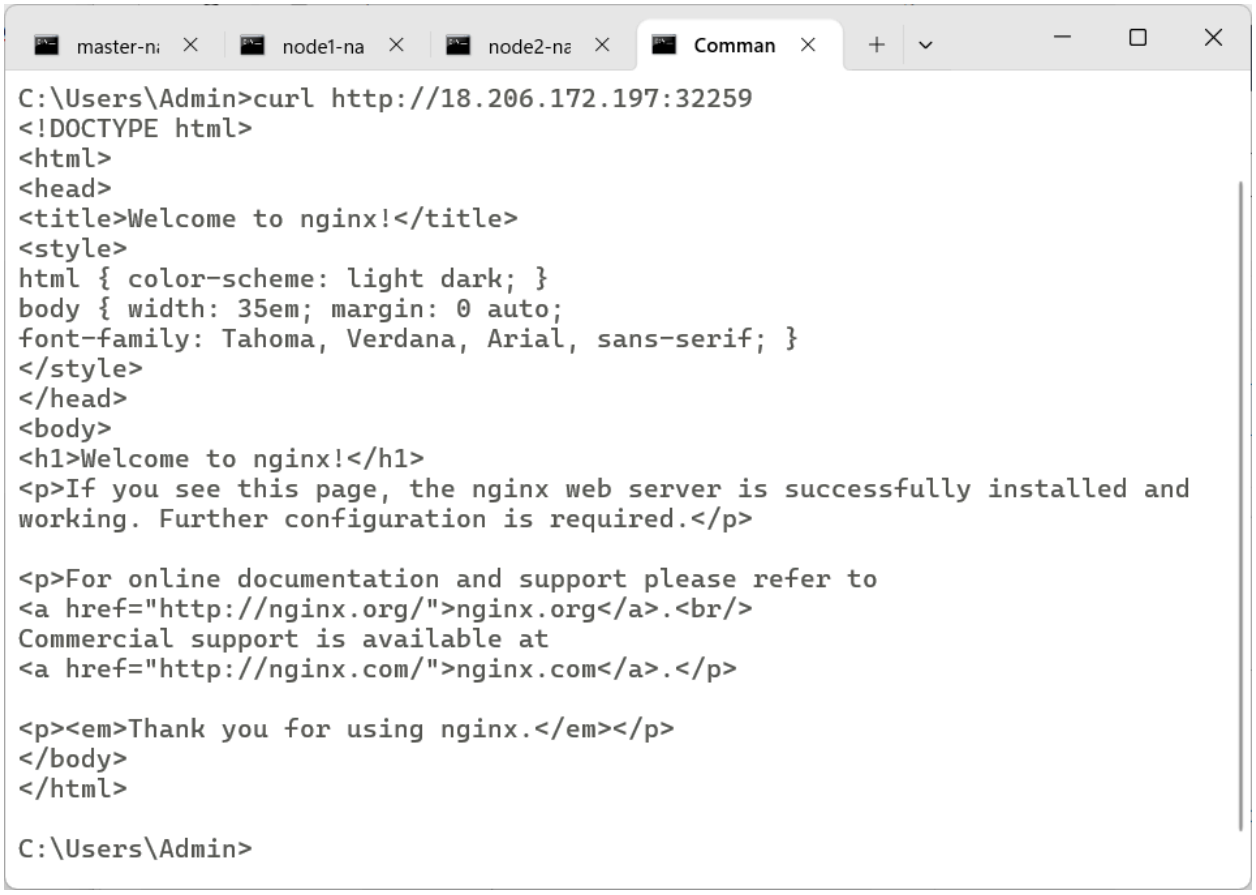
kubectl get services

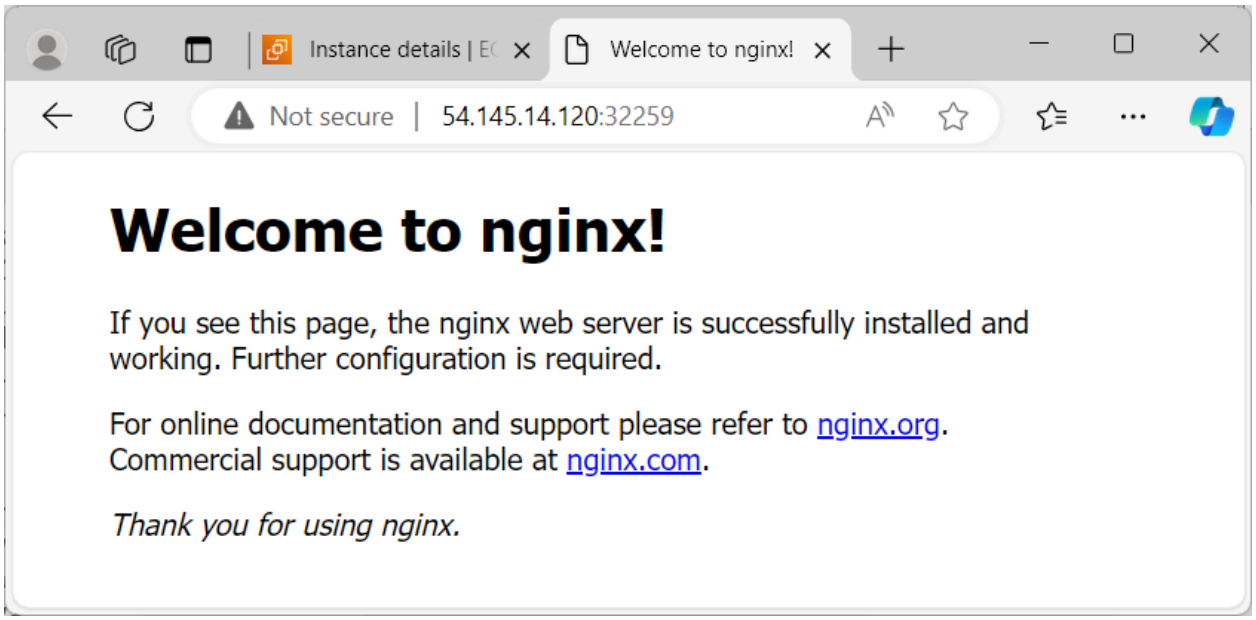
- Note the NodePort value (it will be something like 30000-32767).



Access the Application:

- Open the browser and visit http://<ec2-instance-public-ip>:<NodePort> to verify that the Nginx server is running.





Cleanup (Optional)

When done, clean up the resources:

kubectl delete service nginx

kubectl delete deployment nginx

Conclusion

The case study successfully demonstrated the setup of a Kubernetes cluster on AWS and the deployment of a sample Nginx application. The steps outlined provide a clear pathway for future deployments and emphasize the importance of understanding each component's role within a Kubernetes architecture.