

Case Study 02

Kubernetes Application Deployment

Name: Naikwadi Yash Shivdas

Class: D15B

Roll no.: 42

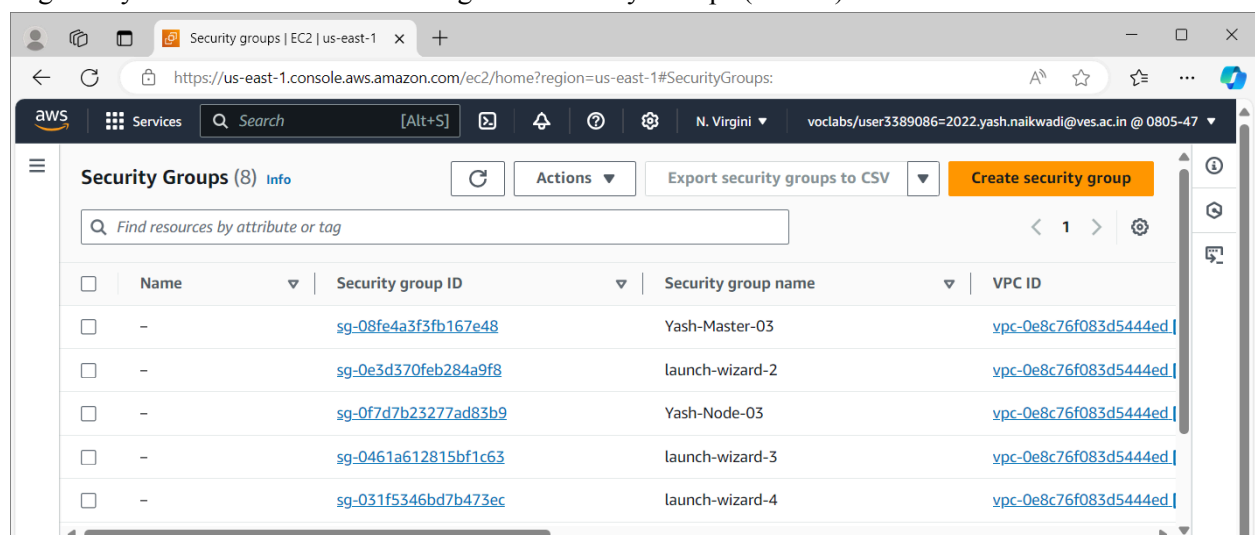
- **Concepts Used:** Kubernetes, AWS Cloud9 IDE, and Kubectl.
- **Problem Statement:** "Set up a Kubernetes cluster on AWS using the Cloud9 IDE. Deploy a sample application using **kubectl** and ensure it runs successfully."
- **Tasks:**
 - Install and configure **kubectl** using AWS Cloud9 IDE.
 - Deploy 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.

Create security group [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name [Info](#)
naikwadi-case-study
Name cannot be edited after creation.

Description [Info](#)
This is advance devops case study

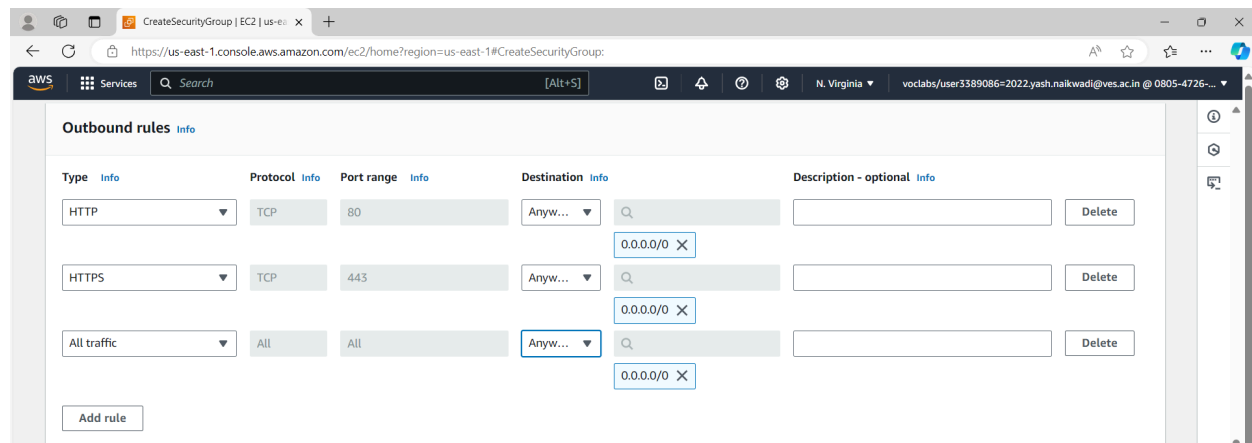
VPC [Info](#)
vpc-0e8c76f083d5444ed

Add the following inbound rules to the security group.

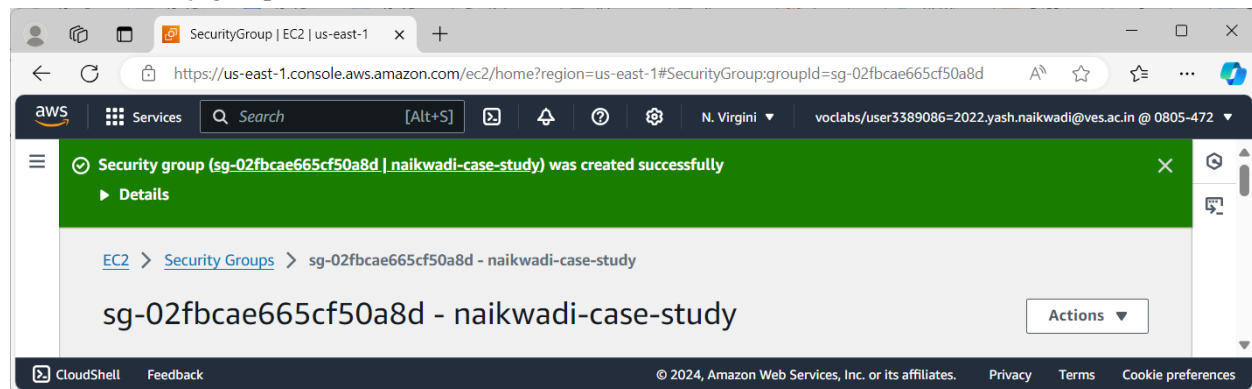
Inbound rules [Info](#)

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	Anywhere...	
All traffic	All	All	Anywhere...	
Custom TCP	TCP	6443	Anywhere...	
Custom TCP	TCP	10251	Anywhere...	
Custom TCP	TCP	10250	Anywhere...	
All TCP	TCP	0 - 65535	Anywhere...	
Custom TCP	TCP	10252	Anywhere...	
SSH	TCP	22	Anywhere...	

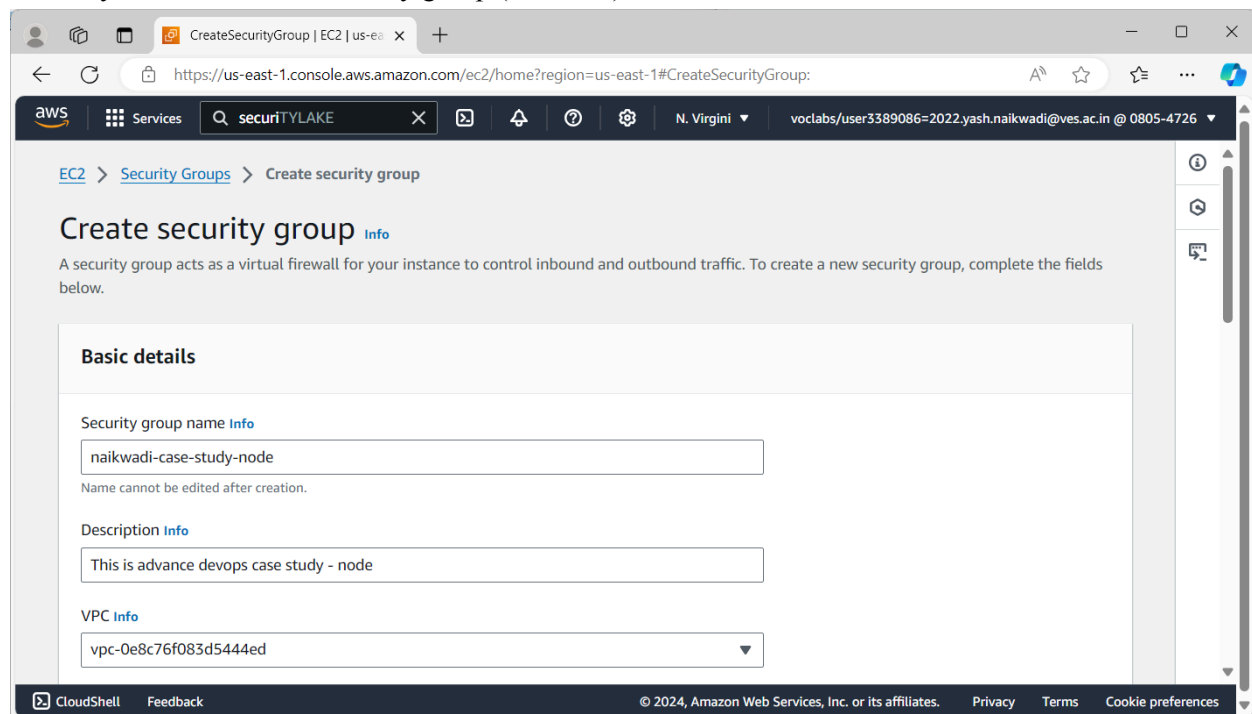
[Add rule](#)



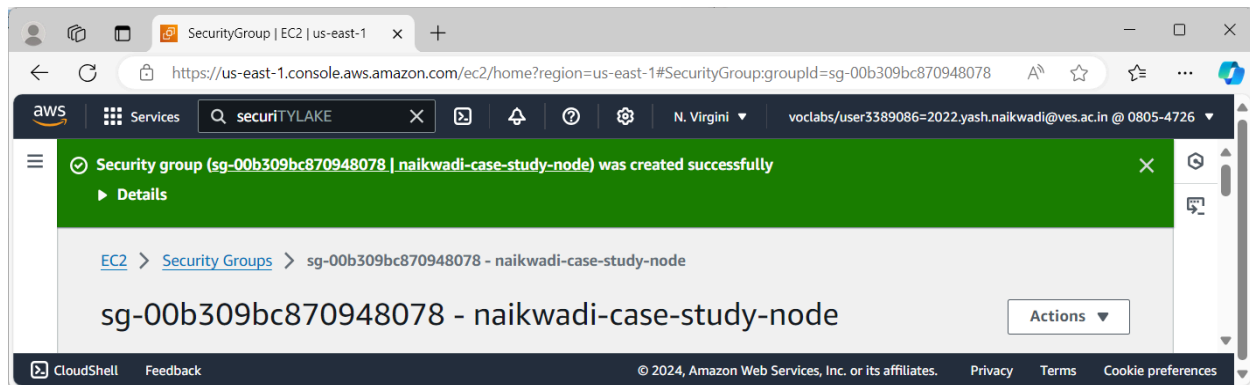
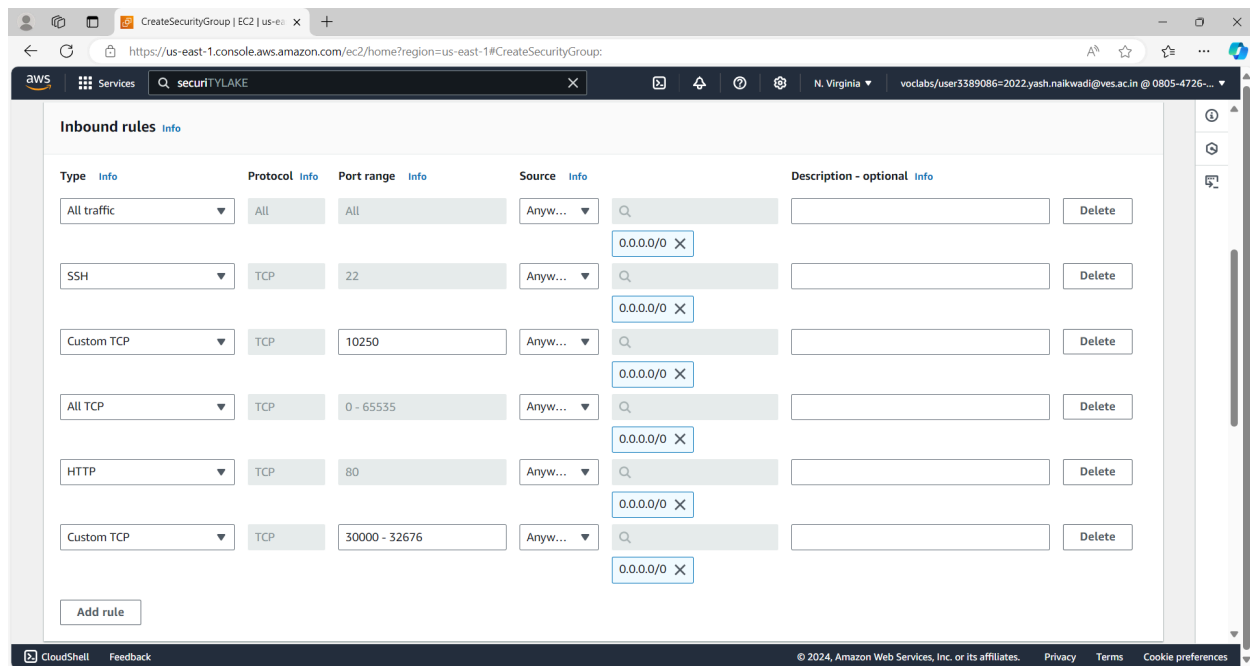
Create a security group.



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

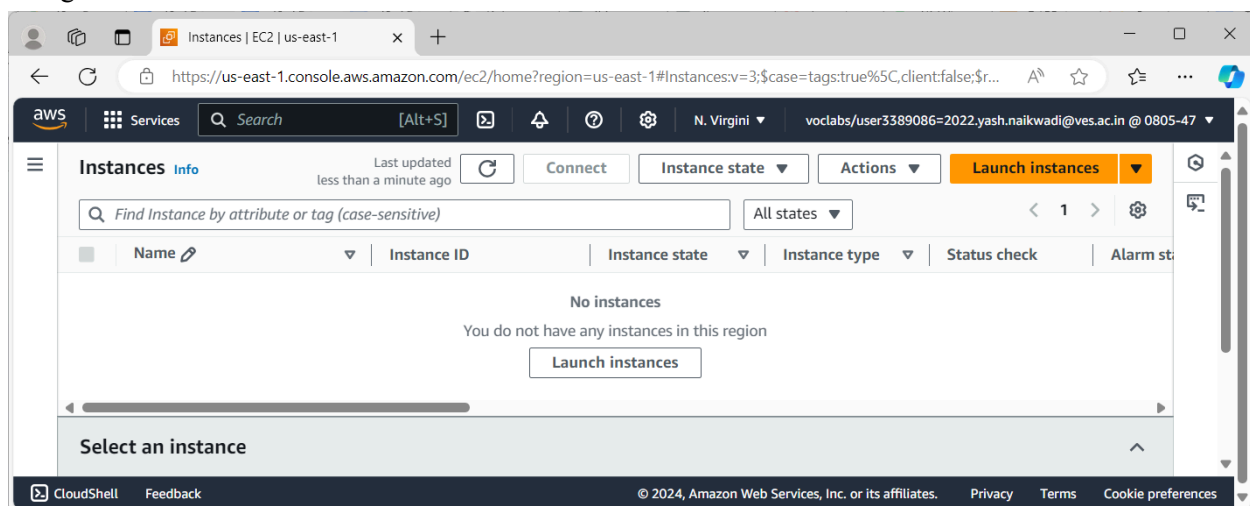


Add the inbound rules for Node security group.



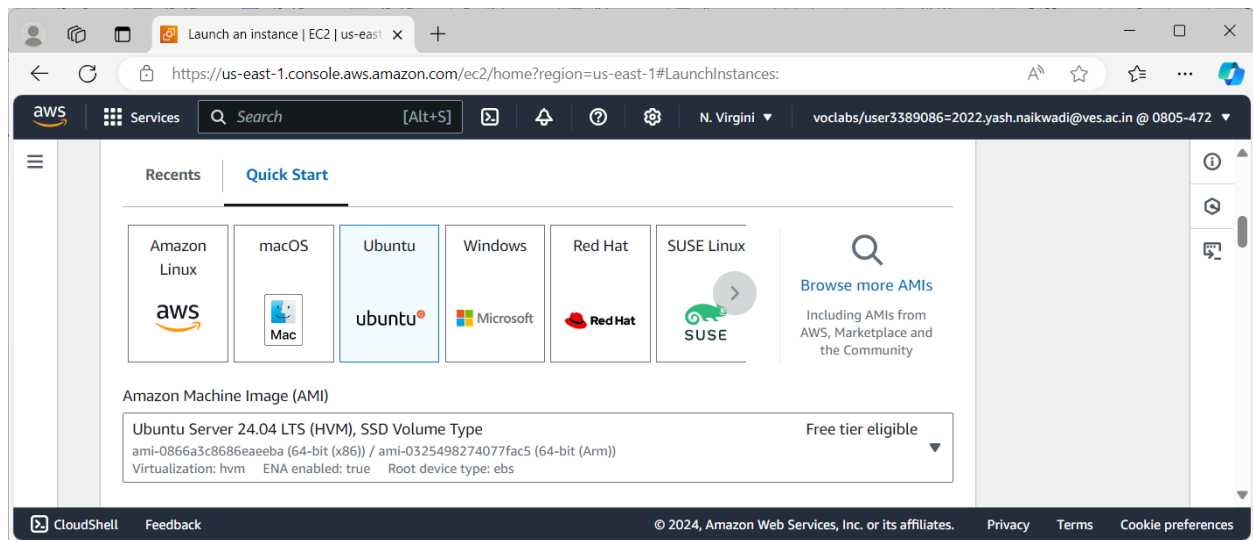
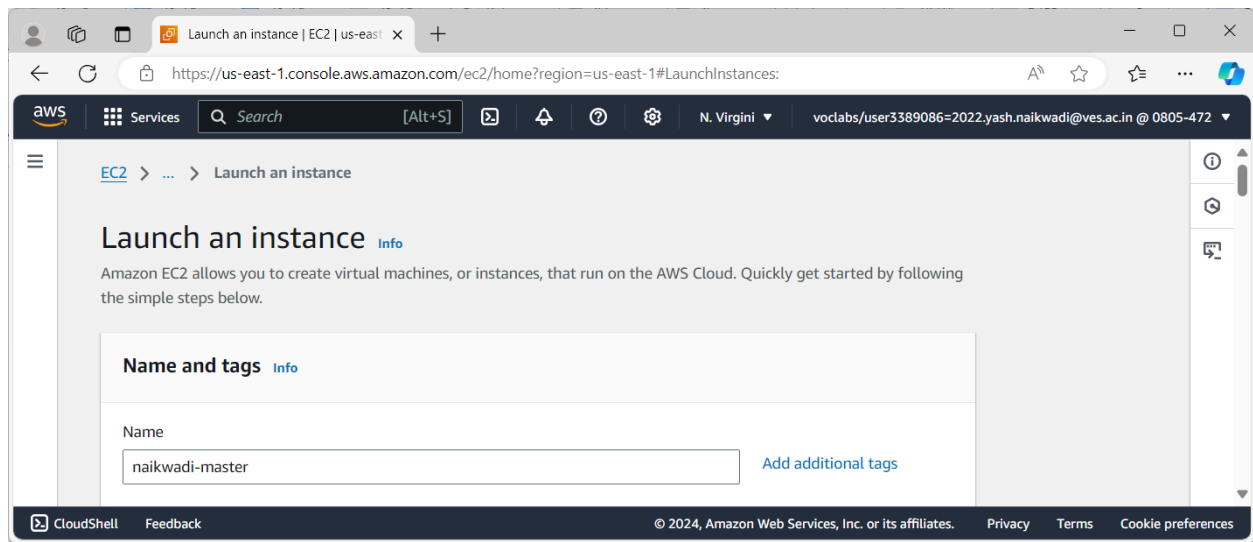
Launch EC2 Instances

Navigate to EC2.

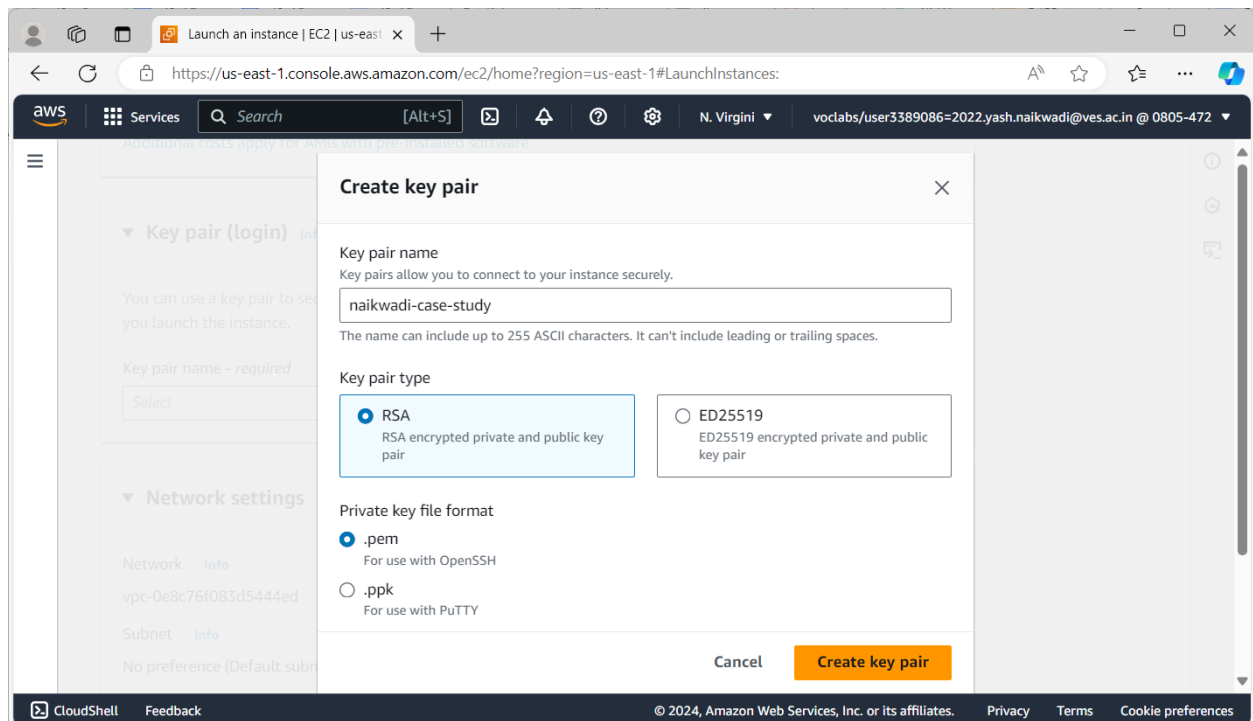


MASTER

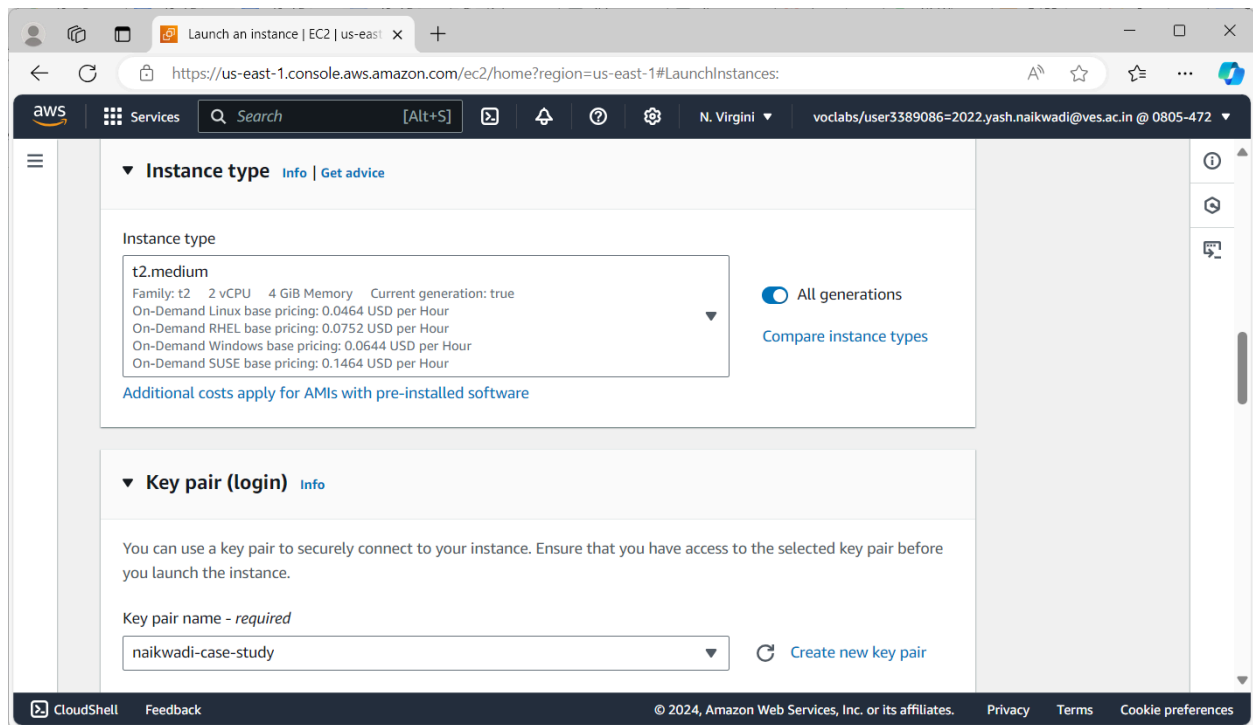
Launch a new EC2 instance for master with the following settings.



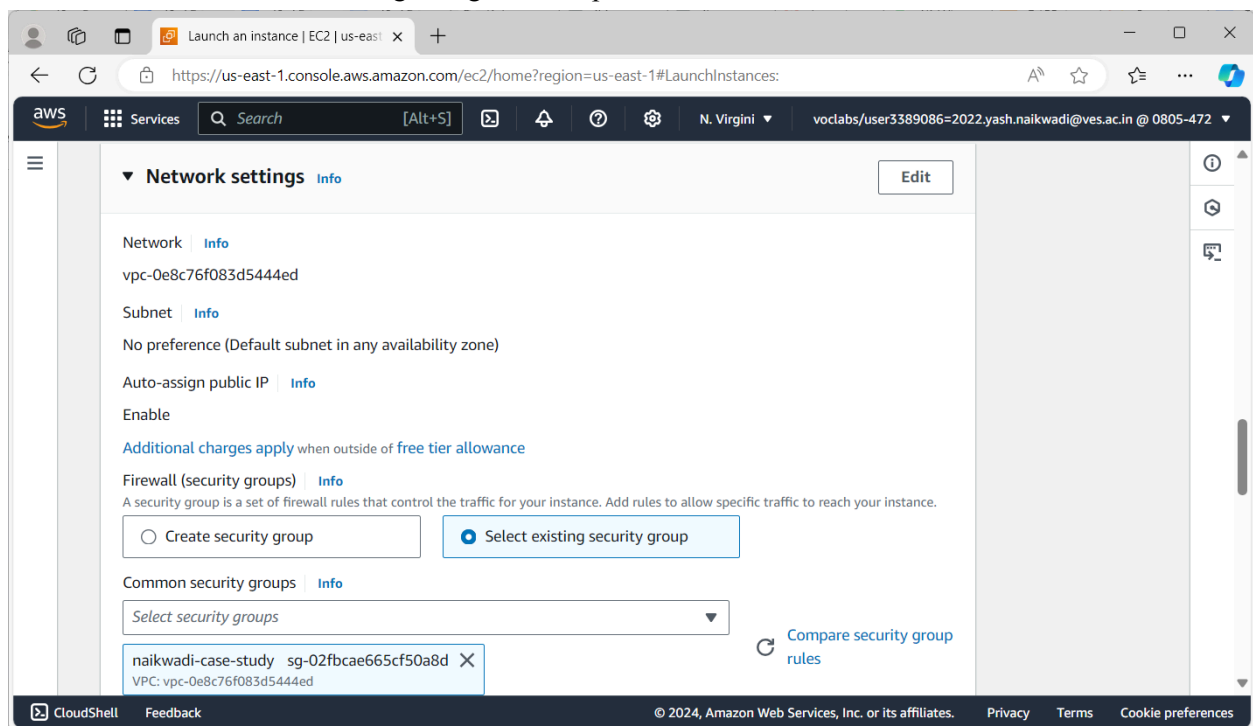
Generate the .pem file (or key-value pair) once while configuring the master ec2.



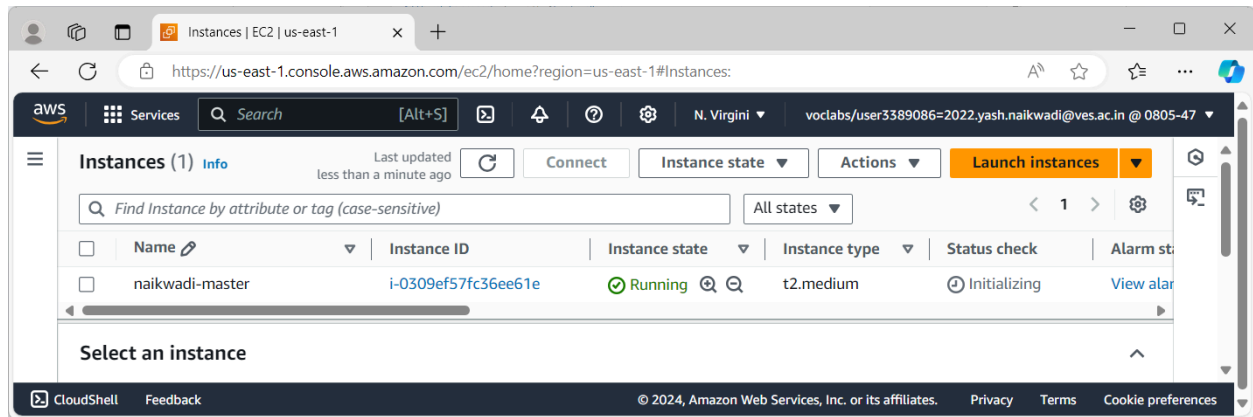
Keep the instance type as t2.medium



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.

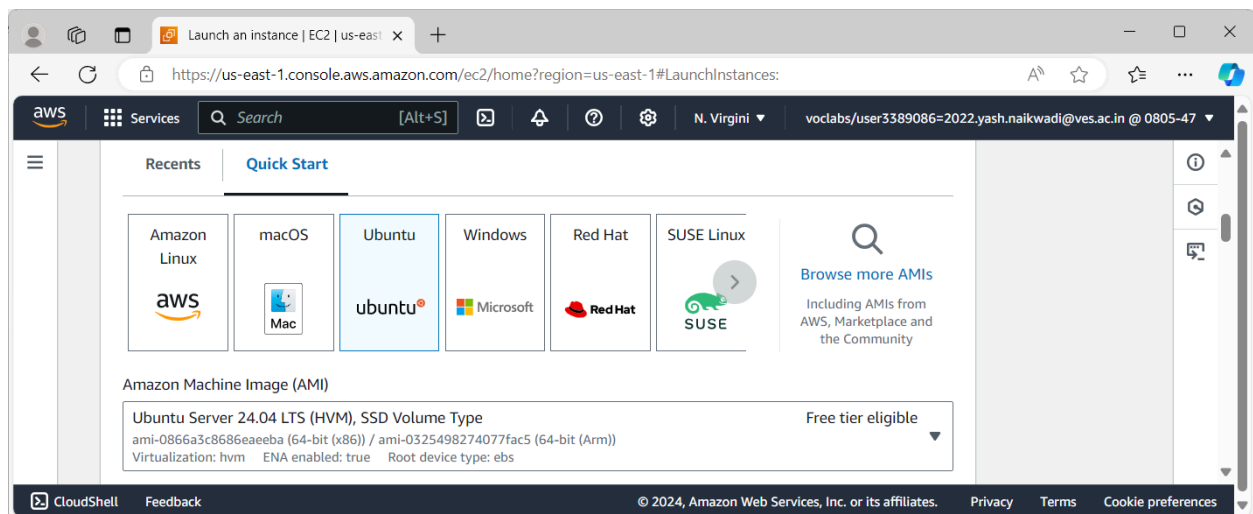
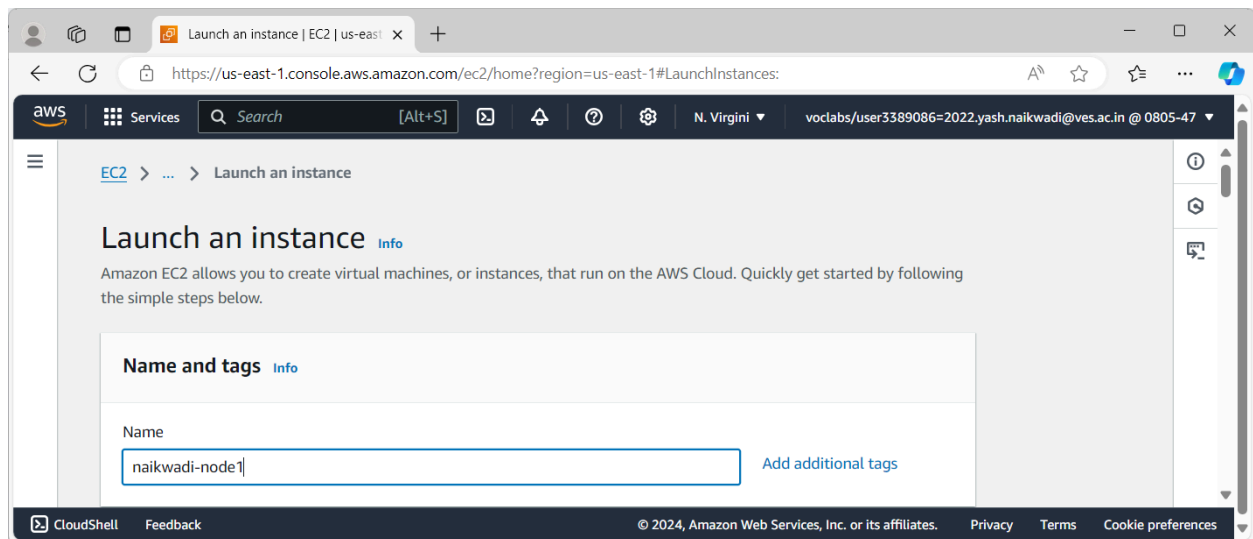


Successfully launched the ec2 instance.

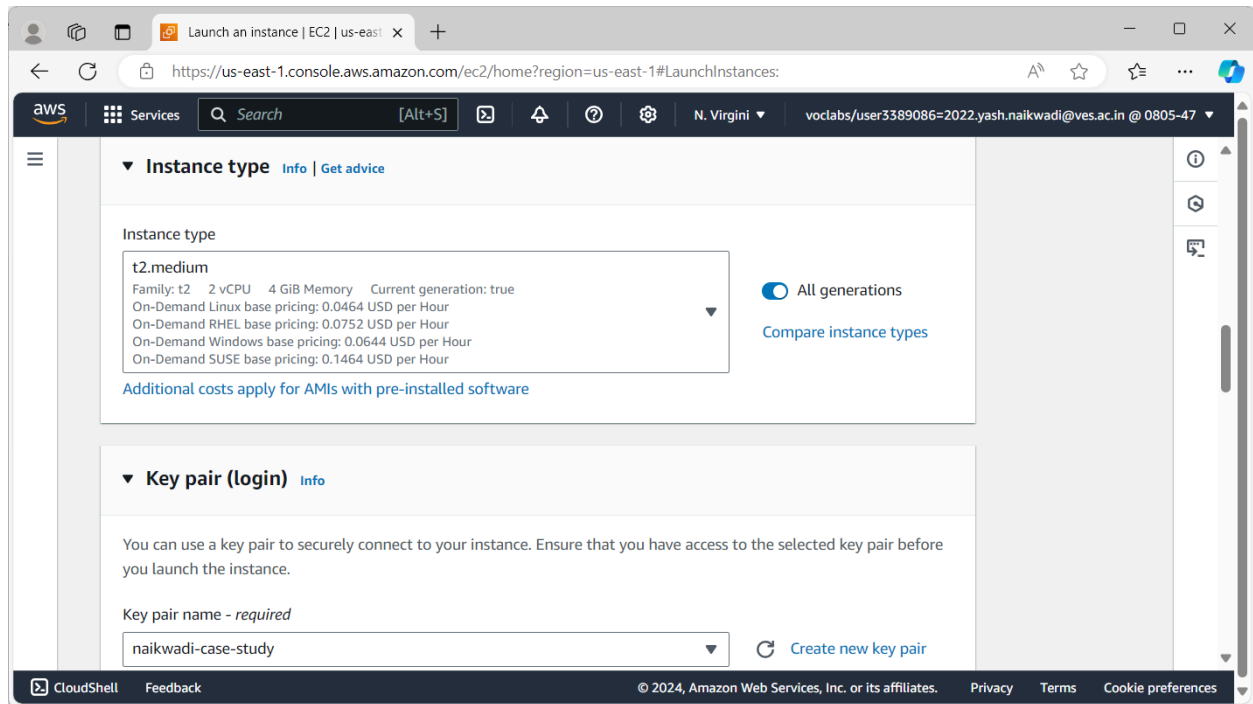


NODE

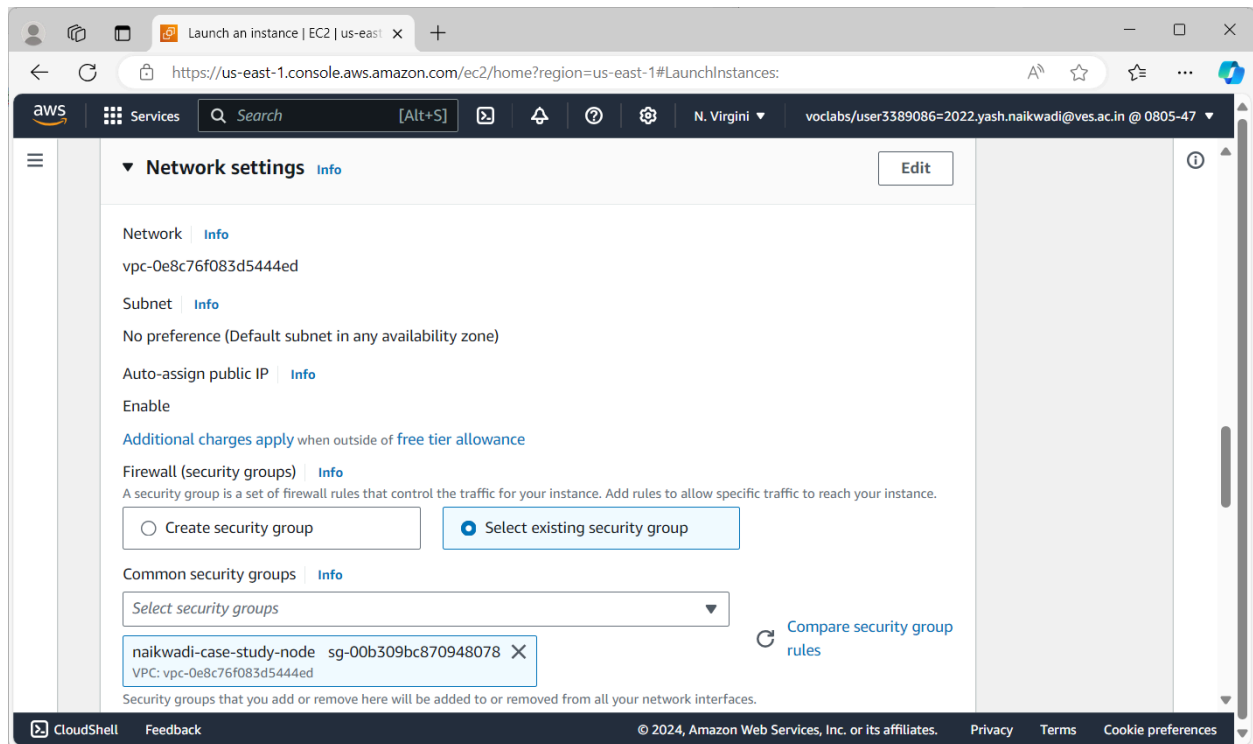
Similarly, launch 2 new ec2 instances for nodes.



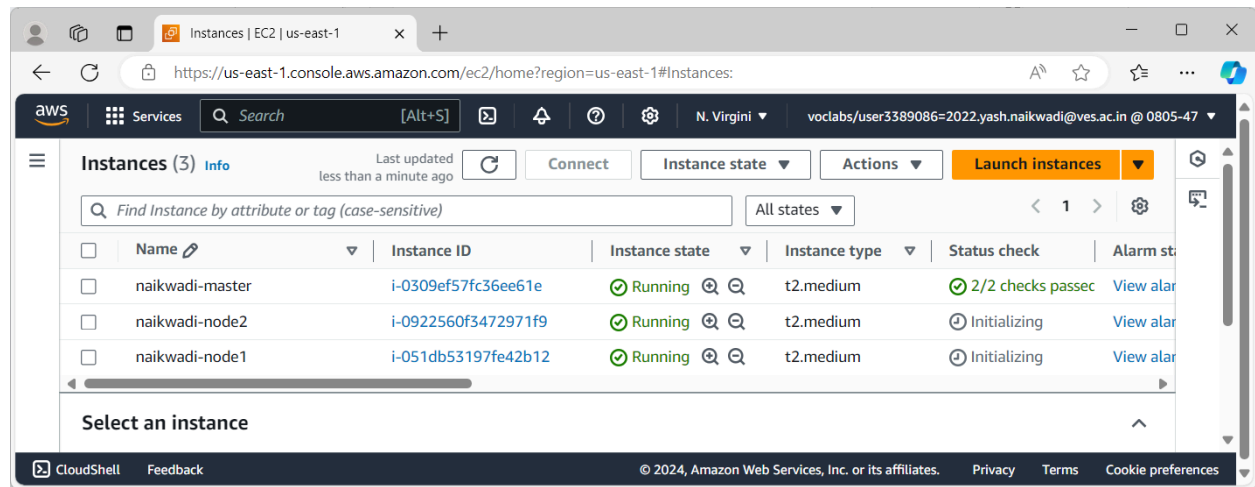
Keep the instance type of node as t2.medium and select the same key-value pair which was created during the master configuration.



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

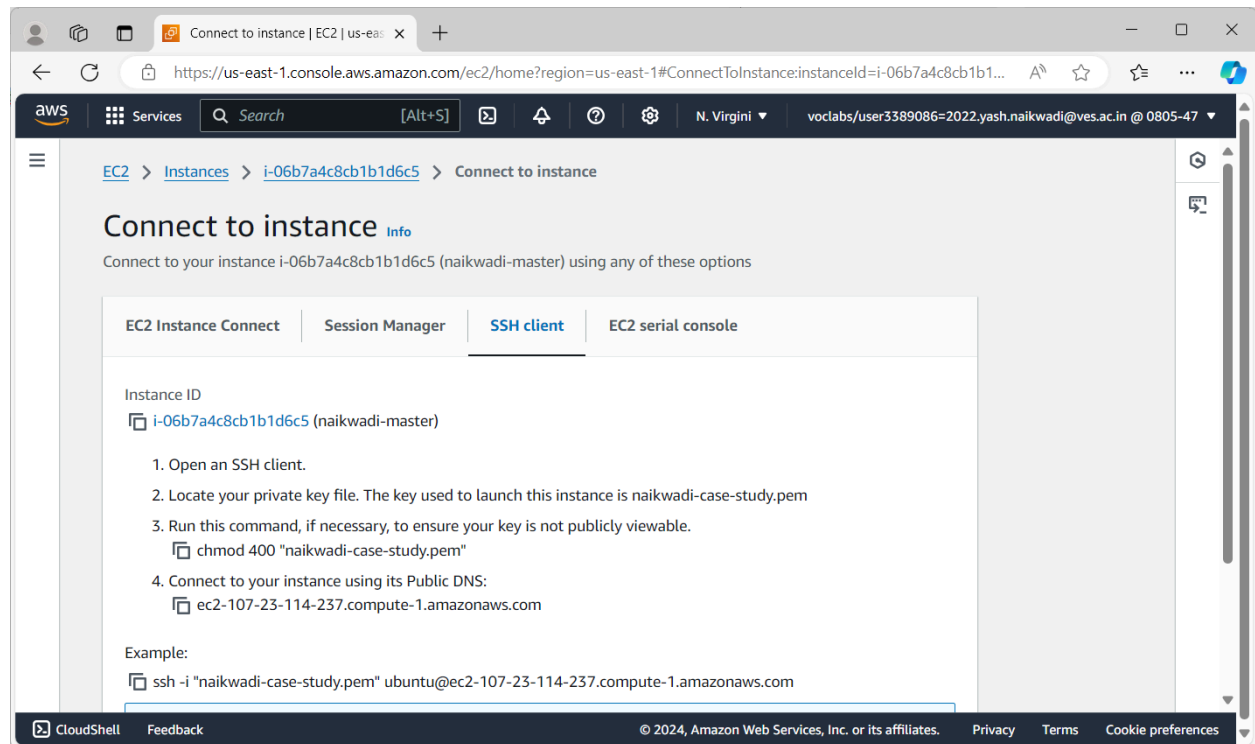


Successfully, created the 3 ec2 instances (1 for master & 2 for nodes).

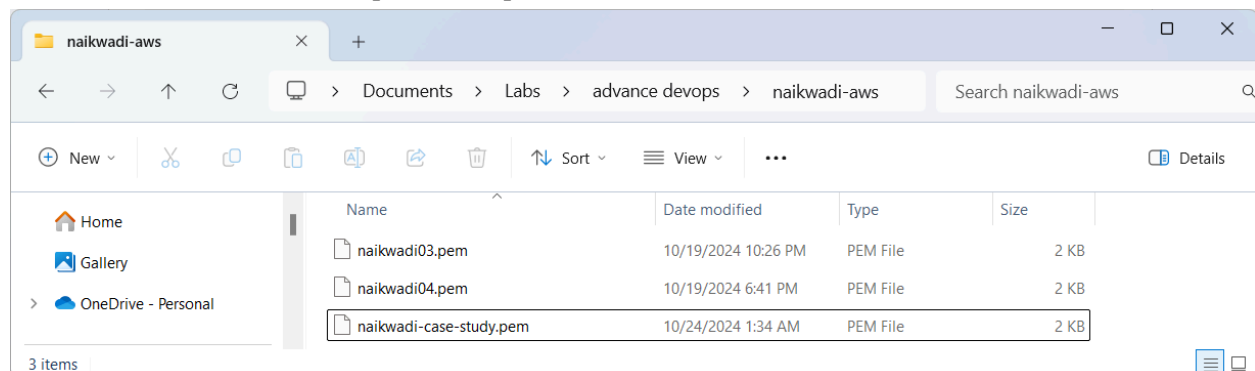


Connect to the Instances

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

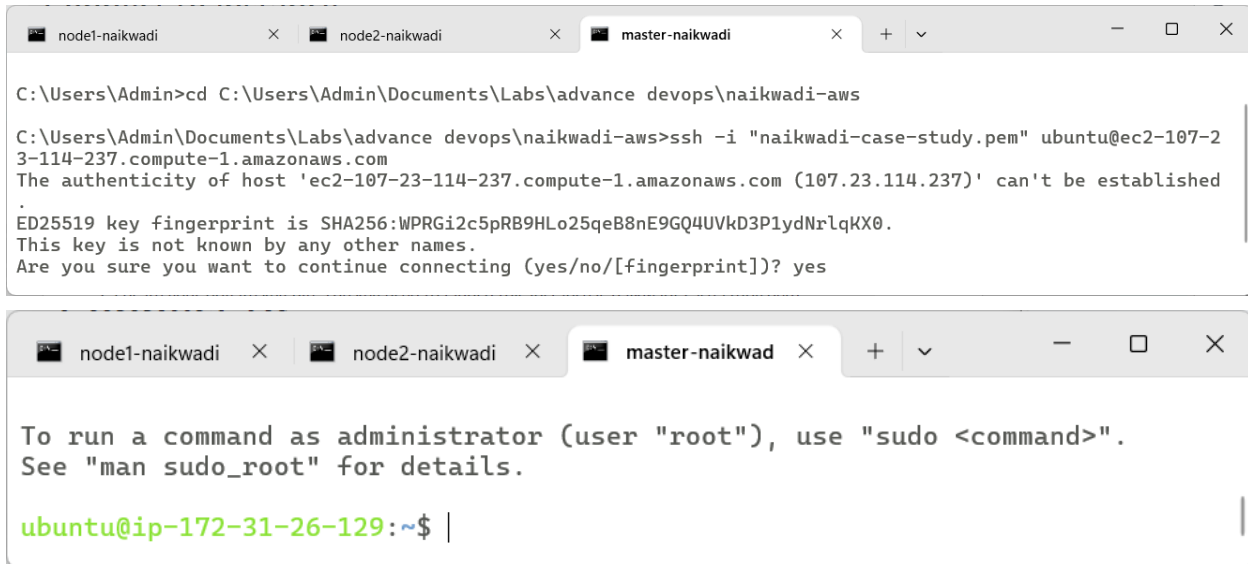


Locate the folder in which the .pem file is placed.



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
```



```
node1-naikwadi x node2-naikwadi x master-naikwadi x + v - □ x
C:\Users\Admin>cd C:\Users\Admin\Documents\Labs\advance devops\naikwadi-aws
C:\Users\Admin\Documents\Labs\advance devops\naikwadi-aws>ssh -i "naikwadi-case-study.pem" ubuntu@ec2-107-23-114-237.compute-1.amazonaws.com
The authenticity of host 'ec2-107-23-114-237.compute-1.amazonaws.com (107.23.114.237)' can't be established
ED25519 key fingerprint is SHA256:WPRGi2c5pRB9HLo25qeB8nE9GQ4UVkD3P1ydNrLqKX0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

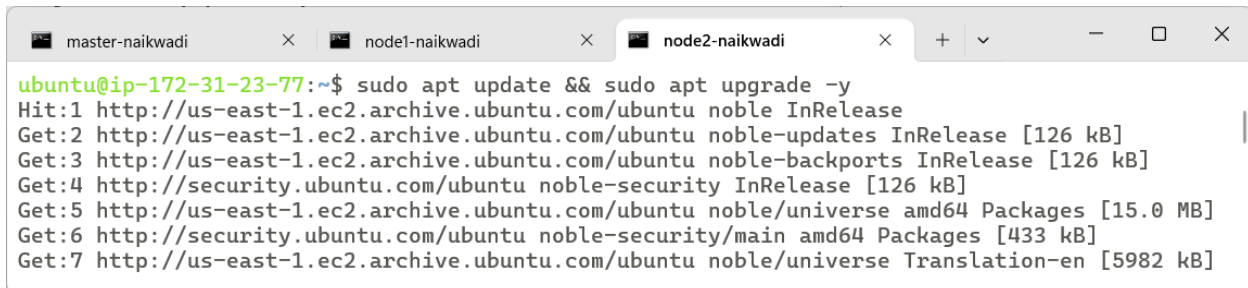
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-26-129:~$ |
```

Update the Instances:

On all nodes (master and workers), update the system:

```
sudo apt update && sudo apt upgrade -y
```



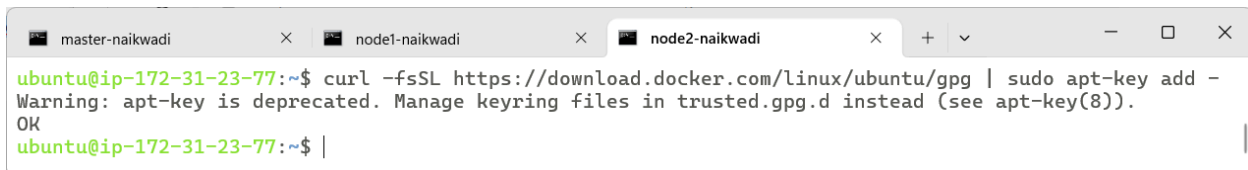
```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-23-77:~$ sudo apt update && sudo apt upgrade -y
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [433 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
```

Install Docker on All Instances (Master and Nodes)

On each instance (Master, Node 1, Node 2), run the following commands to install Docker:

Add Docker's official GPG key

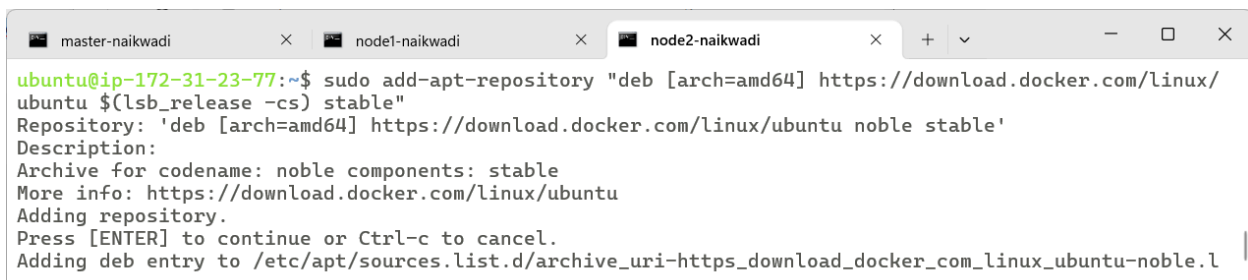
```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```



```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-23-77:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
ubuntu@ip-172-31-23-77:~$ |
```

Add Docker's APT repository

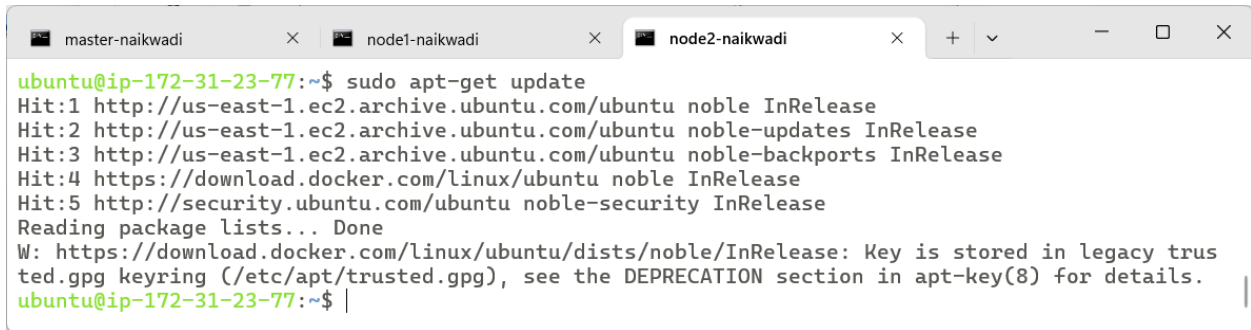
```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```



```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-23-77:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri=https_download_docker_com_linux_ubuntu-noble.l
```

Update the package index

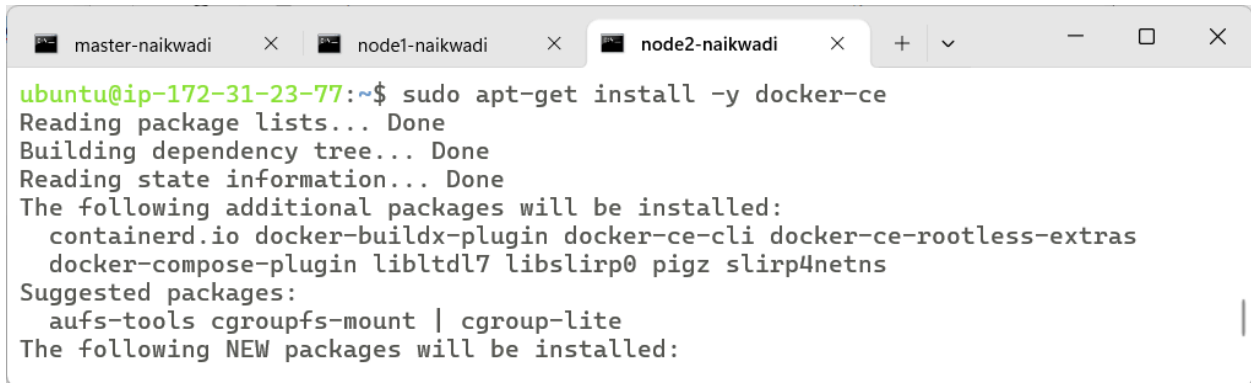
sudo apt-get update



```
ubuntu@ip-172-31-23-77:~$ sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
ubuntu@ip-172-31-23-77:~$
```

Install Docker CE (Community Edition)

sudo apt-get install -y docker-ce



```
ubuntu@ip-172-31-23-77:~$ sudo apt-get install -y docker-ce
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
```

Configure Docker to use systemd as the cgroup driver

sudo mkdir -p /etc/docker

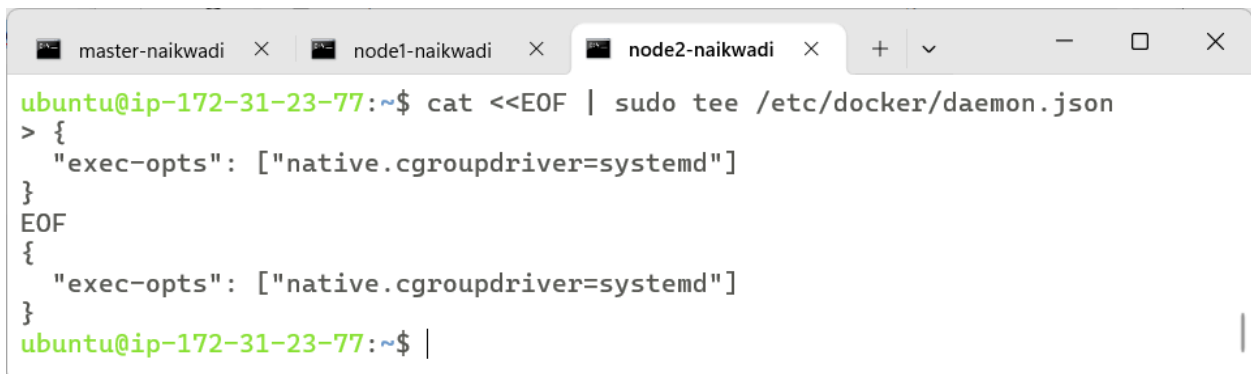


```
ubuntu@ip-172-31-23-77:~$ sudo mkdir -p /etc/docker
ubuntu@ip-172-31-23-77:~$
```

cat <<EOF | sudo tee /etc/docker/daemon.json

```
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
```

EOF



```
ubuntu@ip-172-31-23-77:~$ cat <<EOF | sudo tee /etc/docker/daemon.json
> {
  "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-23-77:~$
```

Enable and start Docker

sudo systemctl enable docker

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-23-77:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /usr/lib
/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
ubuntu@ip-172-31-23-77:~$
```

sudo systemctl daemon-reload

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-23-77:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-23-77:~$
```

sudo systemctl restart docker

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-23-77:~$ sudo systemctl restart docker
ubuntu@ip-172-31-23-77:~$
```

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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-19-23:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
ubuntu@ip-172-31-19-23:~$
```

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] http

https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-26-24:~$ 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
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
ubuntu@ip-172-31-26-24:~$
```

sudo apt-get update

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-18-121:~$ sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Ign:4 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:5 https://prod-cdn.packages.k8s.io/repositories/ipv:/kubernetes:/core:/stable:/v1.31/deb InRelease [1186 B]
```

sudo apt-get install -y kubelet kubeadm kubectl

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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:
  contrack cri-tools kubernetes-cni
The following NEW packages will be installed:
```

sudo apt-mark hold kubelet kubeadm kubectl

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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 x node1-naikwadi x node2-naikwadi x + v - □ x
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 x node1-naikwadi x node2-naikwadi x + v - □ x
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 x node1-naikwadi x node2-naikwadi x + v - □ x
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 x node1-naikwadi x node2-naikwadi x + v - □ x
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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-26-24:~$ sudo systemctl restart containerd
ubuntu@ip-172-31-26-24:~$
```

sudo systemctl enable containerd

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-18-121:~$ sudo systemctl enable containerd
ubuntu@ip-172-31-18-121:~$
```

sudo systemctl status containerd

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-26-24:~$ sudo systemctl status containerd
● containerd.service - containerd container runtime
   Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled)
   Active: active (running) since Wed 2024-10-23 20:41:20 UTC; 1min
     Docs: https://containerd.io
   Main PID: 13478 (containerd)
     Tasks: 7
```

sudo apt-get install -y socat

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-18-121:~$ sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 374 kB of archives.
```

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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-26-129:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.31.2
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W1023 22:06:14.637443 14365 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI san
```

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.)

```
naikwadi-master x  naikwadi-node1 x  naikwadi-node2 x  +  -  □  X
kubeadm join 172.31.88.198:6443 --token sfxllk.6ggcci840yv22h18 \
--discovery-token-ca-cert-hash sha256:1a08766f3ac51776ba839909a5e22b7
9a96f4b353260439519866eca40a50203
ubuntu@ip-172-31-88-198:~$ |
```

Next, configure kubectl to interact with the cluster:

mkdir -p \$HOME/.kube

```
master-naikwadi x  node1-naikwadi x  node2-naikwadi x  +  -  □  X
ubuntu@ip-172-31-26-24:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-26-24:~$ |
```

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

```
master-naikwadi x  node1-naikwadi x  node2-naikwadi x  +  -  □  X
ubuntu@ip-172-31-26-24:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@ip-172-31-26-24:~$ |
```

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

```
master-naikwadi x  node1-naikwadi x  node2-naikwadi x  +  -  □  X
ubuntu@ip-172-31-26-24:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-26-24:~$ |
```

Now Run the command

kubectl get nodes

to see the nodes before executing the Join command on nodes.

```
master-naikwadi x  node1-naikwadi x  node2-naikwadi x  +  -  □  X
ubuntu@ip-172-31-26-129:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-26-129    NotReady control-plane  87s   v1.31.2
ubuntu@ip-172-31-26-129:~$ |
```

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.weumqfjjpgmsoilz --discovery-token-ca-cert-hash sha256:6311d1e2d998752029085c10573c9facf87e6ffaef00f3ddc882f28d7b7c45)

```

master-naikwadi  node1-naikwadi  node2-naikwadi
ubuntu@ip-172-31-19-23:~$ sudo kubeadm join 172.31.26.24:6443 --token 9accqk.weumqfjjmpmsoilz --discovery-token-ca-cert-hash sha256:6311d1e2d998752029085c10573c9facf87e6ffaeeef00f3ddc882f28d7b7c45
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.001275725s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiservert and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

ubuntu@ip-172-31-19-23:~$ |

```

```

master-naikwadi  node1-naikwadi  node2-naikwadi
ubuntu@ip-172-31-18-121:~$ sudo kubeadm join 172.31.26.24:6443 --token 9accqk.weumqfjjmpmsoilz --discovery-token-ca-cert-hash sha256:6311d1e2d998752029085c10573c9facf87e6ffaeeef00f3ddc882f28d7b7c45
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 501.548334ms
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiservert and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

ubuntu@ip-172-31-18-121:~$ |

```

Verify the Cluster

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

kubectl get nodes

```

master-naikwadi  node1-naikwadi  node2-naikwadi
ubuntu@ip-172-31-26-129:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-22-204    NotReady <none>   39s   v1.31.2
ip-172-31-23-77     NotReady <none>   34s   v1.31.2
ip-172-31-26-129    NotReady control-plane 2m47s v1.31.2
ubuntu@ip-172-31-26-129:~$ |

```

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>


```

master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
ubuntu@ip-172-31-26-24:~$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
poddissruptionbudget.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

```

master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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

```

master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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

```

master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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

```

master-naikwadi x node1-naikwadi x node2-naikwadi x + v - □ x
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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-26-129:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE      VERSION
ip-172-31-22-204    Ready    Node1    3m21s    v1.31.2
ip-172-31-23-77     Ready    Node2    3m16s    v1.31.2
ip-172-31-26-129    Ready    control-plane 5m29s    v1.31.2
ubuntu@ip-172-31-26-129:~$ |
```

Deploy the Sample Nginx Application

1. Deploy Nginx:

- Create a deployment:

bash

Copy code

```
kubectl create deployment nginx --image=nginx
```

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-26-129:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
ubuntu@ip-172-31-26-129:~$
```

Expose the Nginx Deployment:

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

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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-26-129:~$ kubectl expose deployment nginx --port=80 --type=NodePort
service/nginx exposed
ubuntu@ip-172-31-26-129:~$ |
```

Get the NodePort:

To get the NodePort:

```
kubectl get services
```

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

```
master-naikwadi x node1-naikwadi x node2-naikwadi x + - □ ×
ubuntu@ip-172-31-26-129:~$ kubectl get services
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1     <none>         443/TCP          33m
nginx               NodePort    10.106.209.250 <none>         80:32259/TCP     44s
ubuntu@ip-172-31-26-129:~$ |
```

Access the Application:

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

```
master-ni X node1-na X node2-na X Comman X + - □ X
C:\Users\Admin>curl http://18.206.172.197:32259
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

C:\Users\Admin>
```



Cleanup (Optional)

When done, clean up the resources:

```
kubectrl delete service nginx
```

```
kubectrl 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.