Case Study 02
Kubernetes Application Deployment

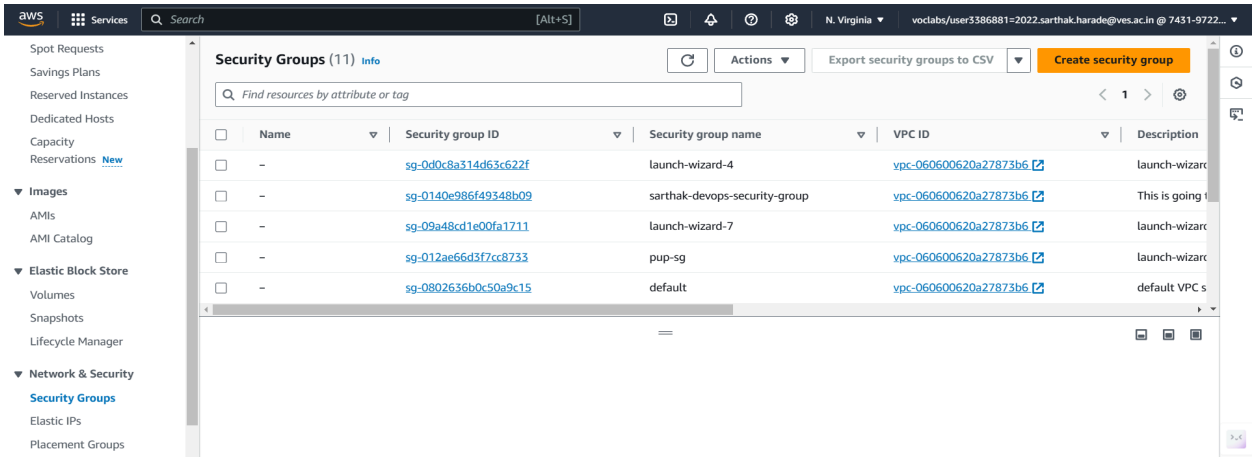
Name:Sarthak Harade
Class: D15B
Roll no.: 22

- 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.

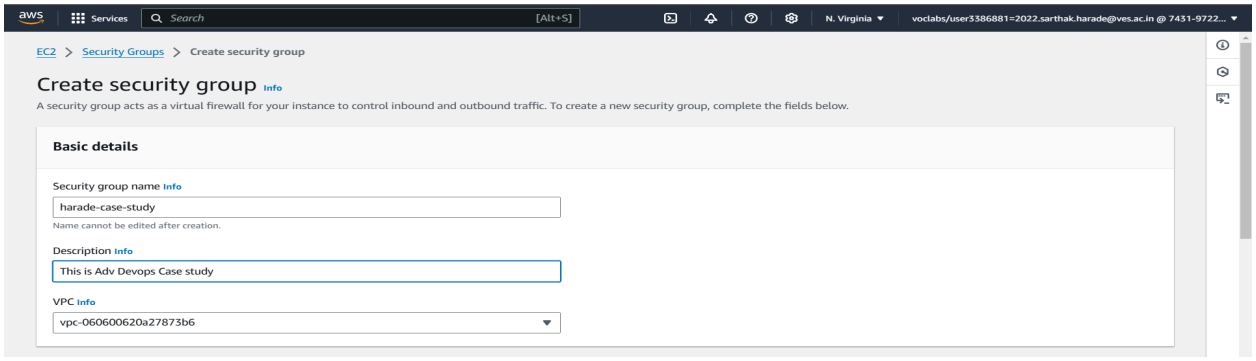
SOLUTION

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.

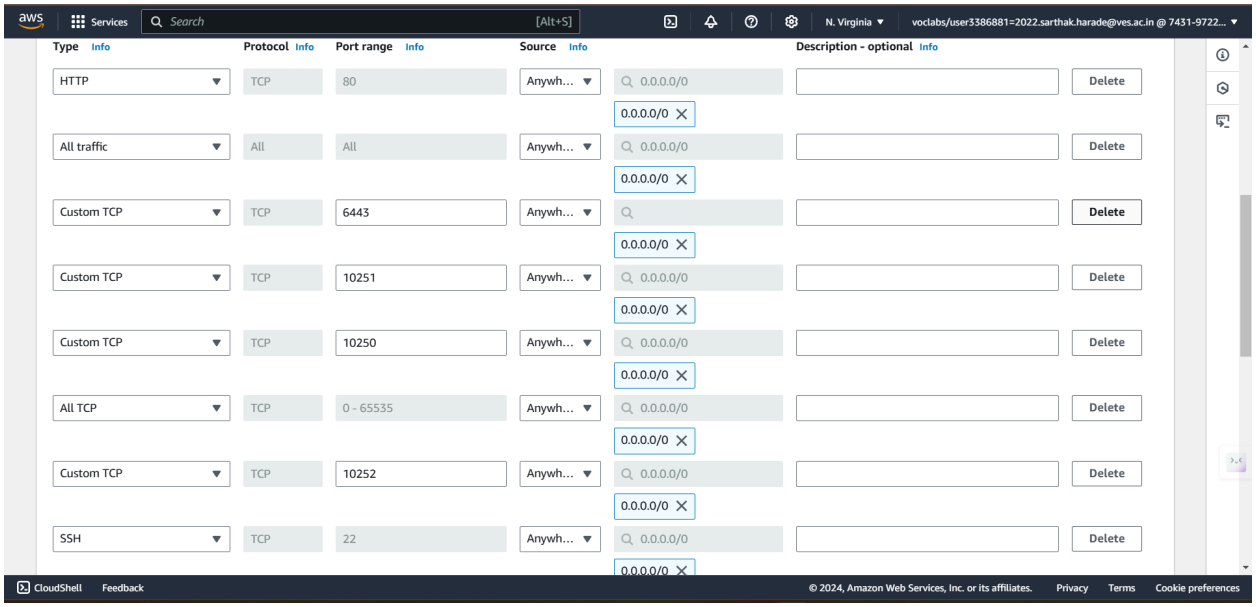
Create Security Groups for Instances Log in to your AWS account and navigate to Security Groups (in EC2).

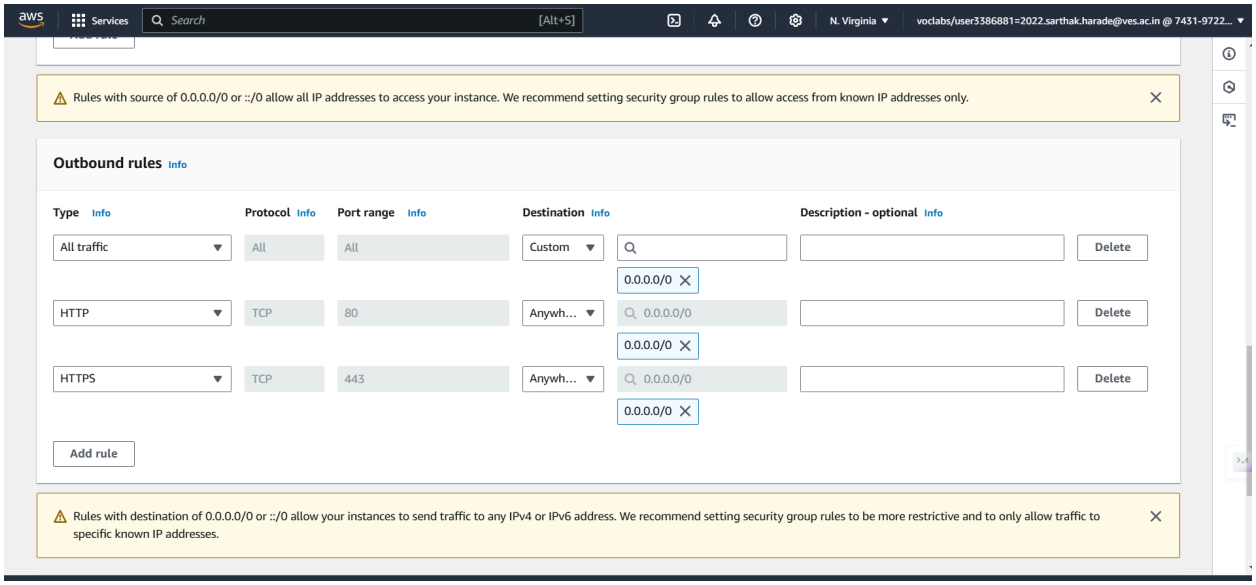


Click on create security group and give the name and description to the security group.

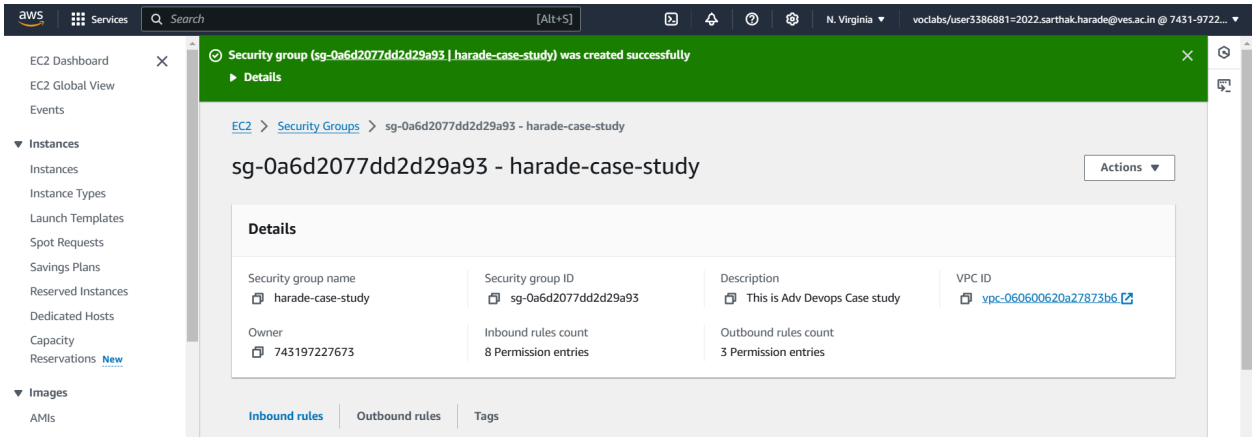


Add the following inbound rules to the security group

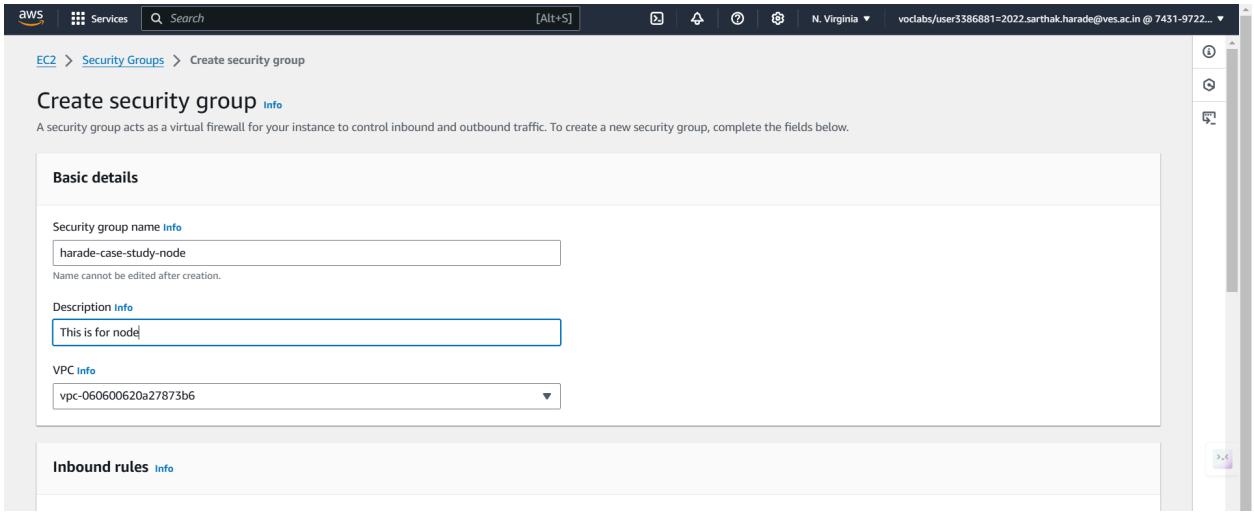




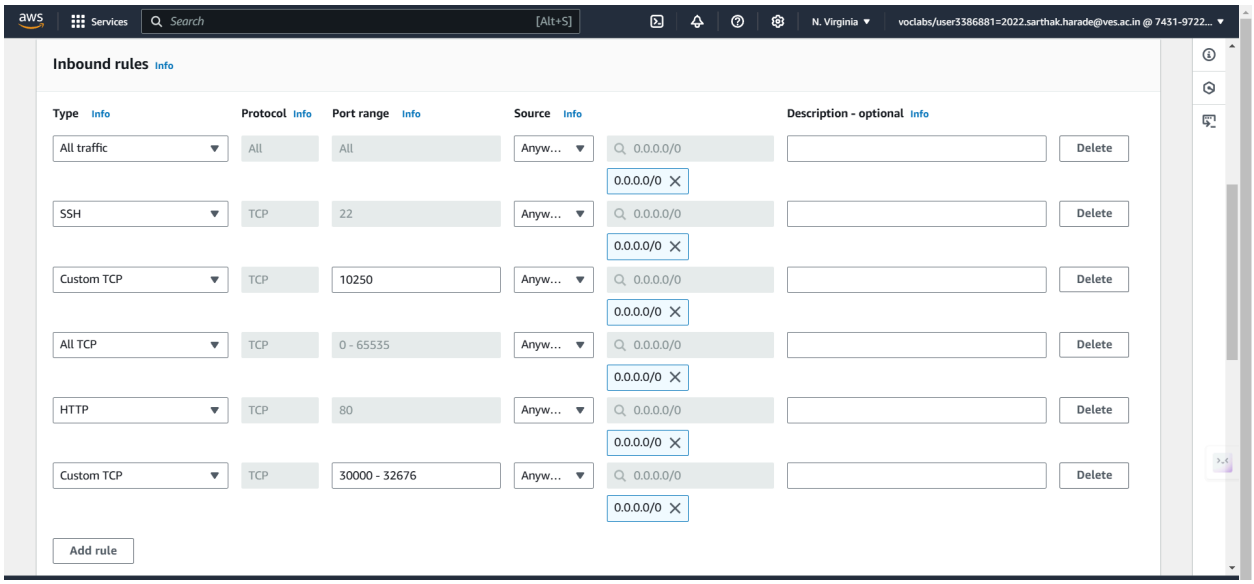
Create a security group.

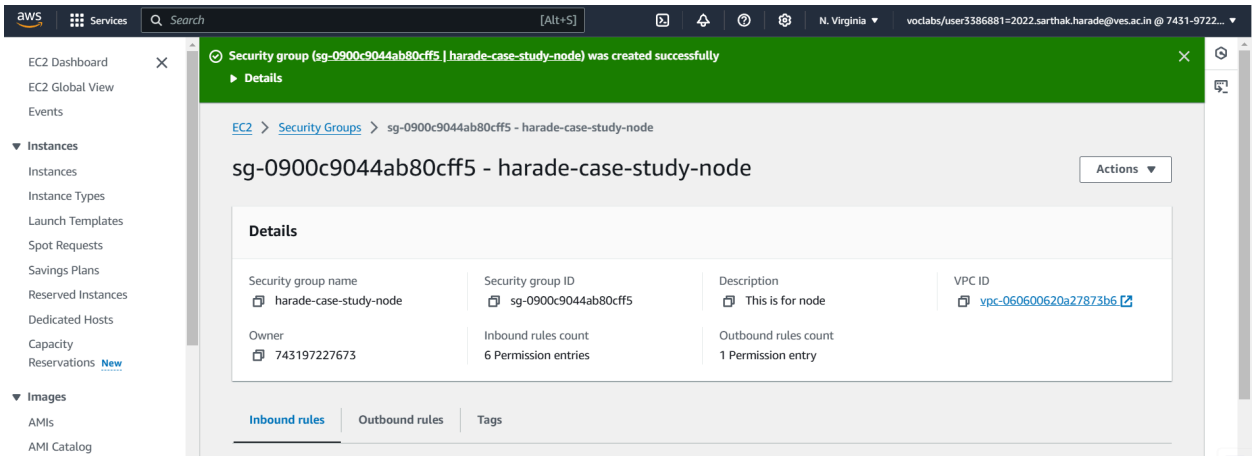


Similarly, create one more security group (for Node).



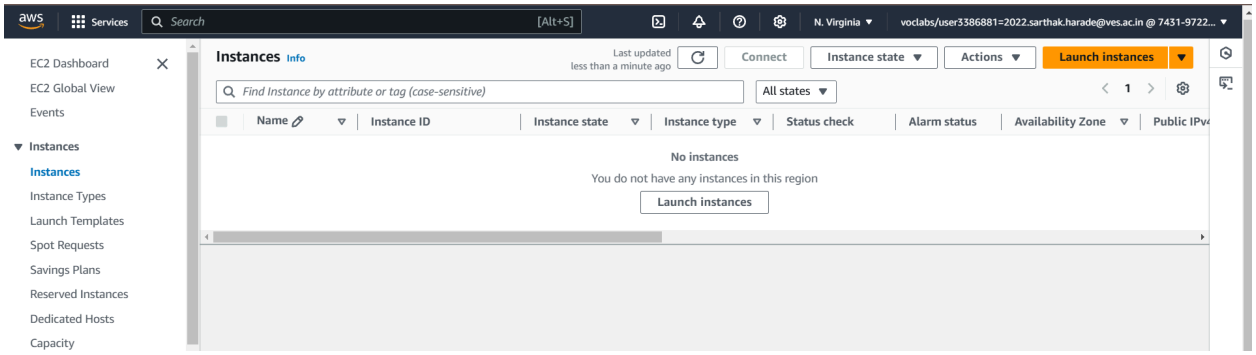
Add the inbound rules for Node security group.





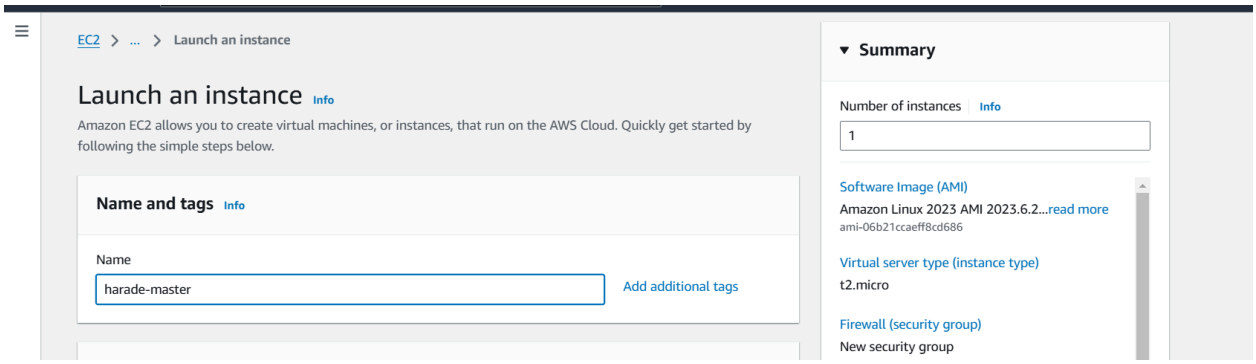
Launch EC2 Instances

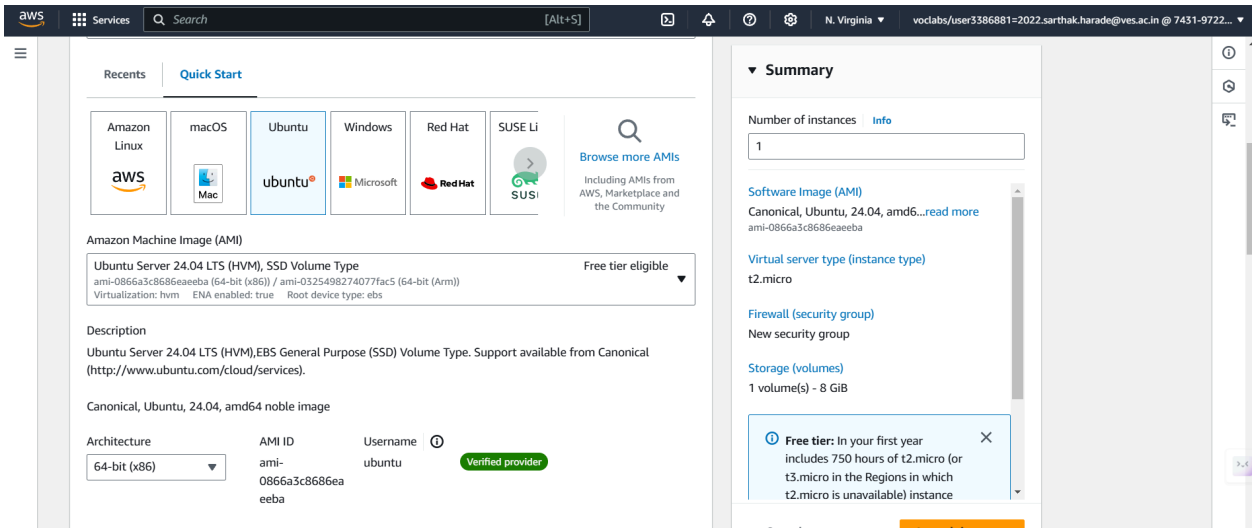
Navigate to EC2.



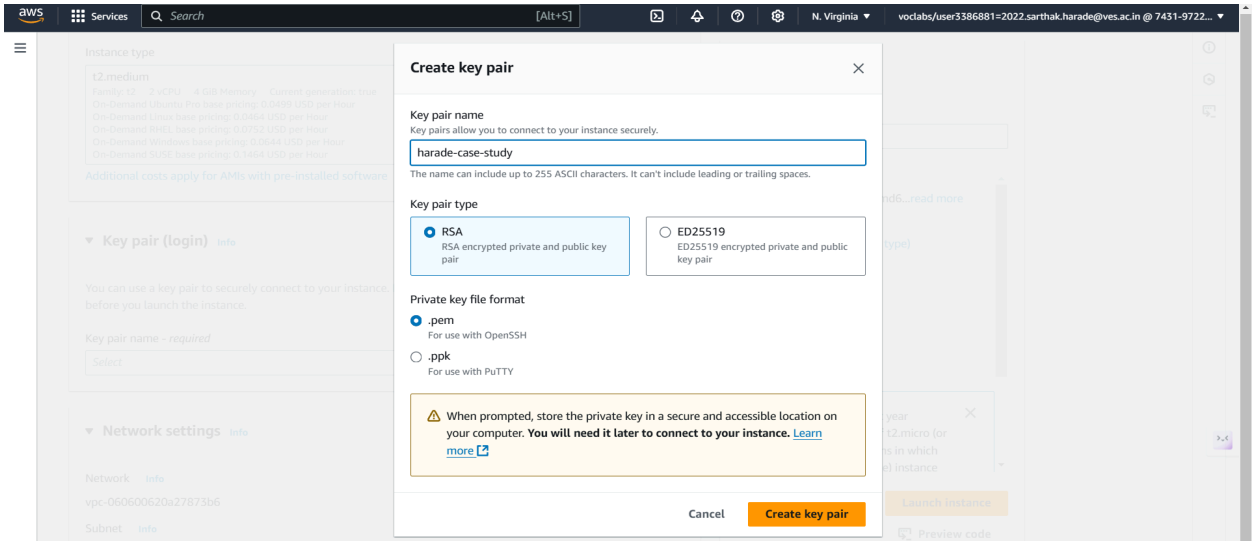
MASTER

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

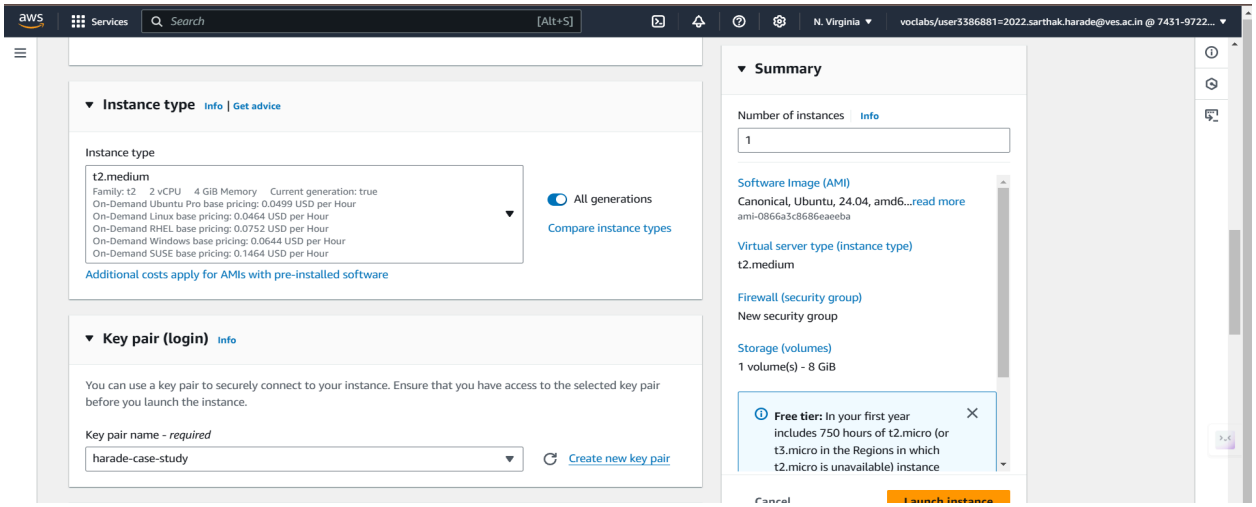




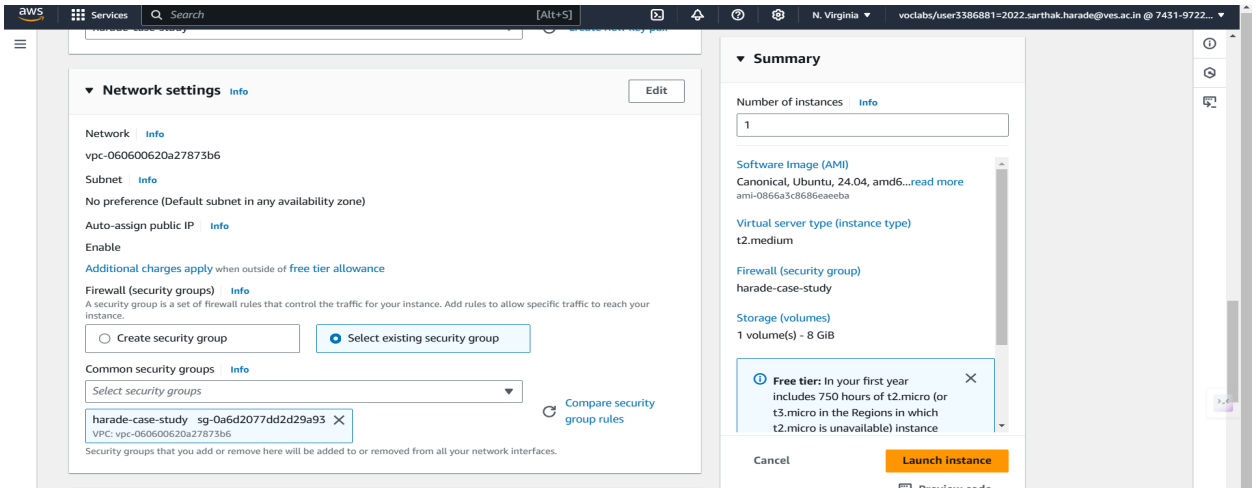
Generate the .pem file (or key-value pair) once while configuring the master ec2



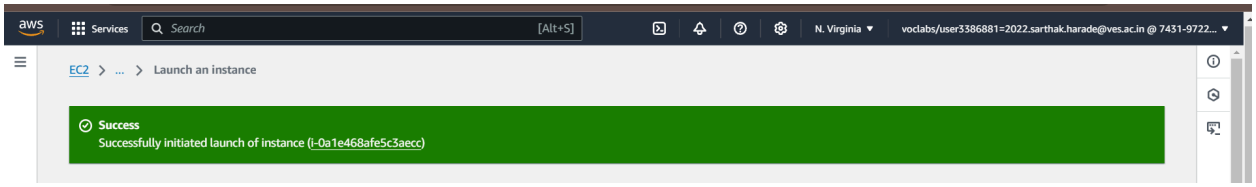
Keep the instance type as t2.medium

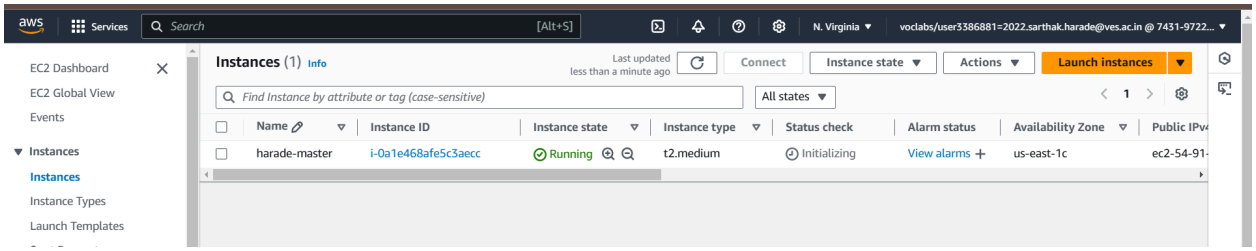


Select the existing security group under the network setting tab in which you have to select the group of master which was made in the beginning of the experiment.



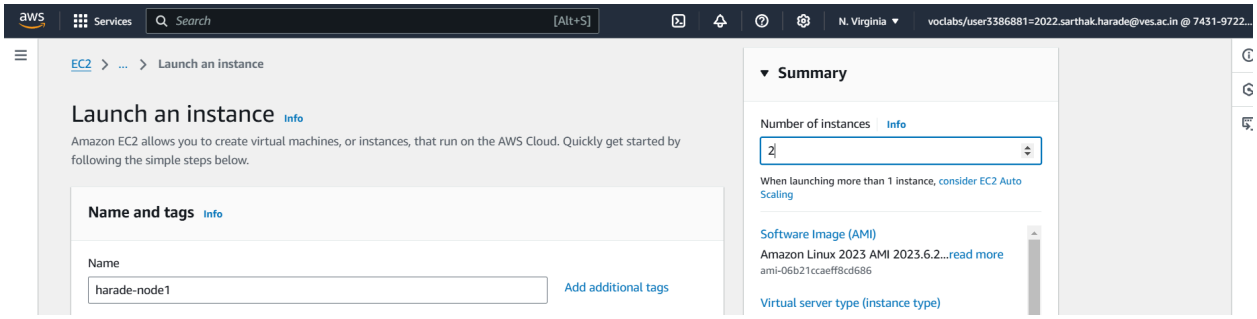
Successfully launched the ec2 instance.

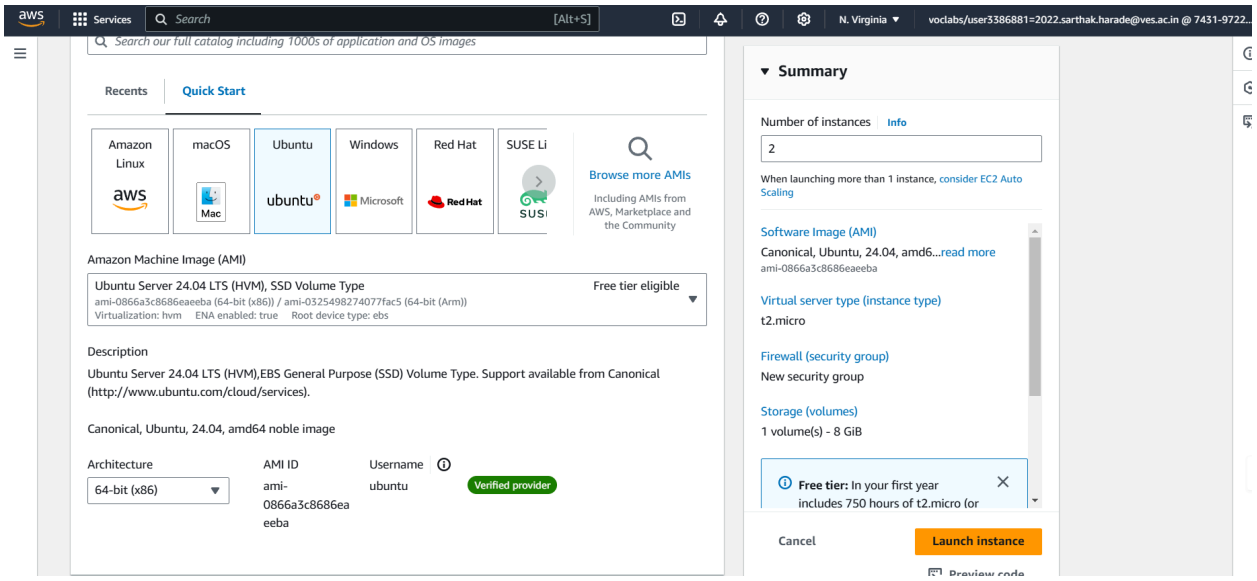




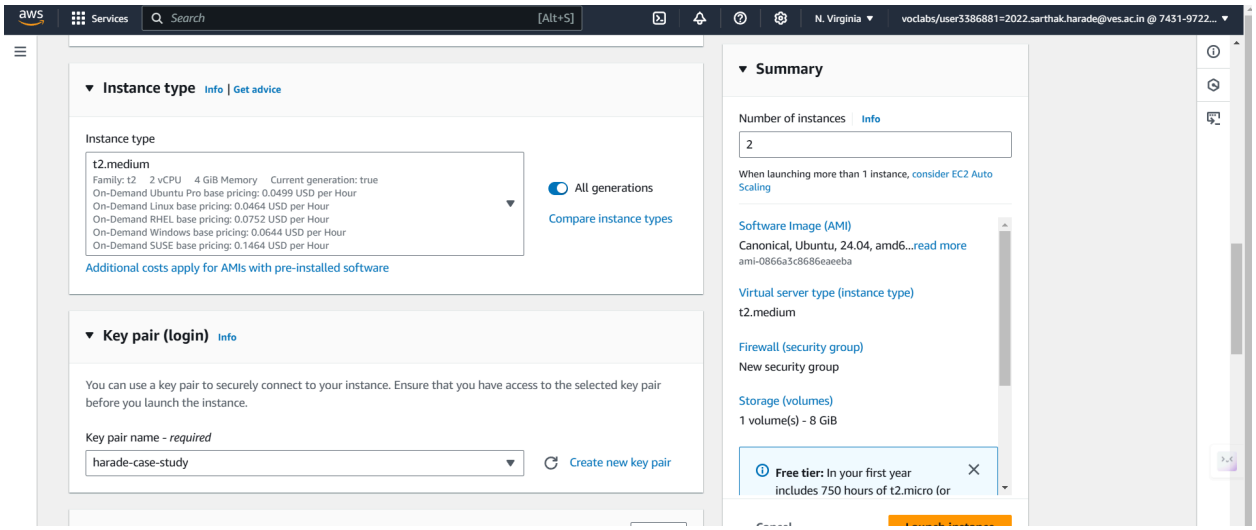
NODE

Similarly, launch 2 new ec2 instances for nodes.

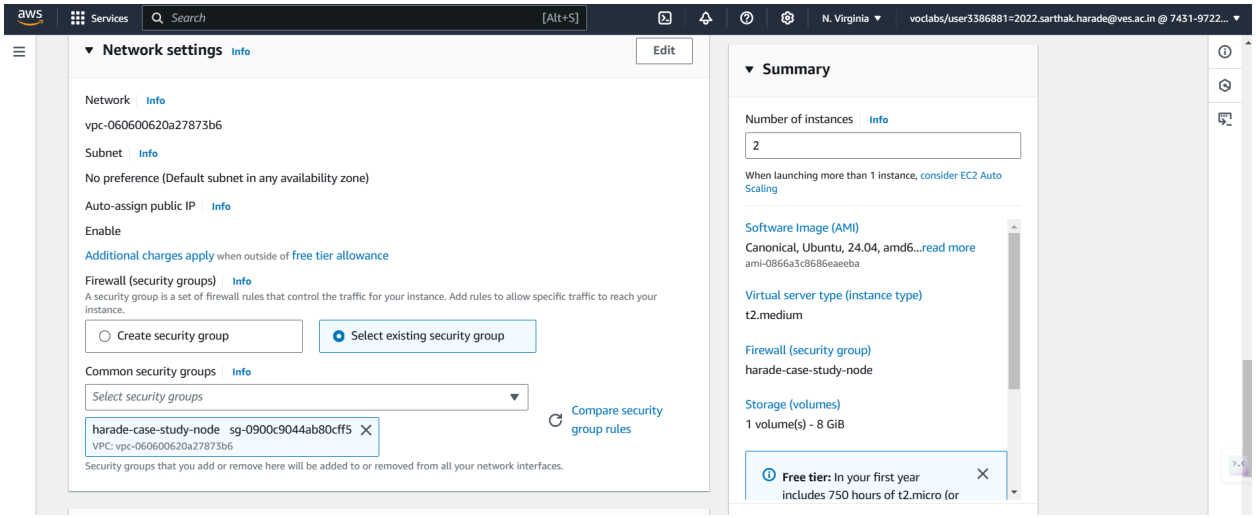




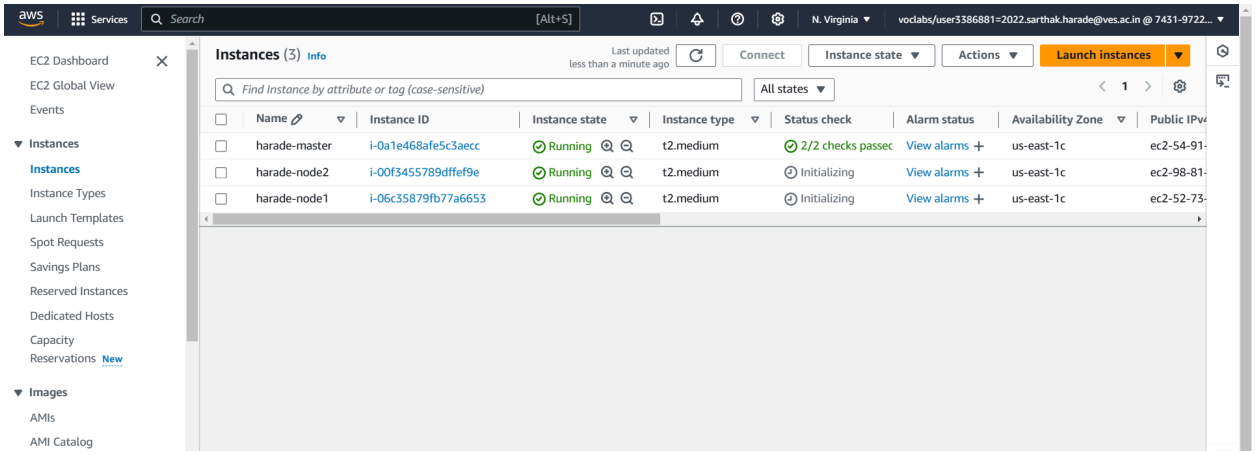
Keep the instance type of node as t2.medium and select the same key-value pair which was created during the master configuration.



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

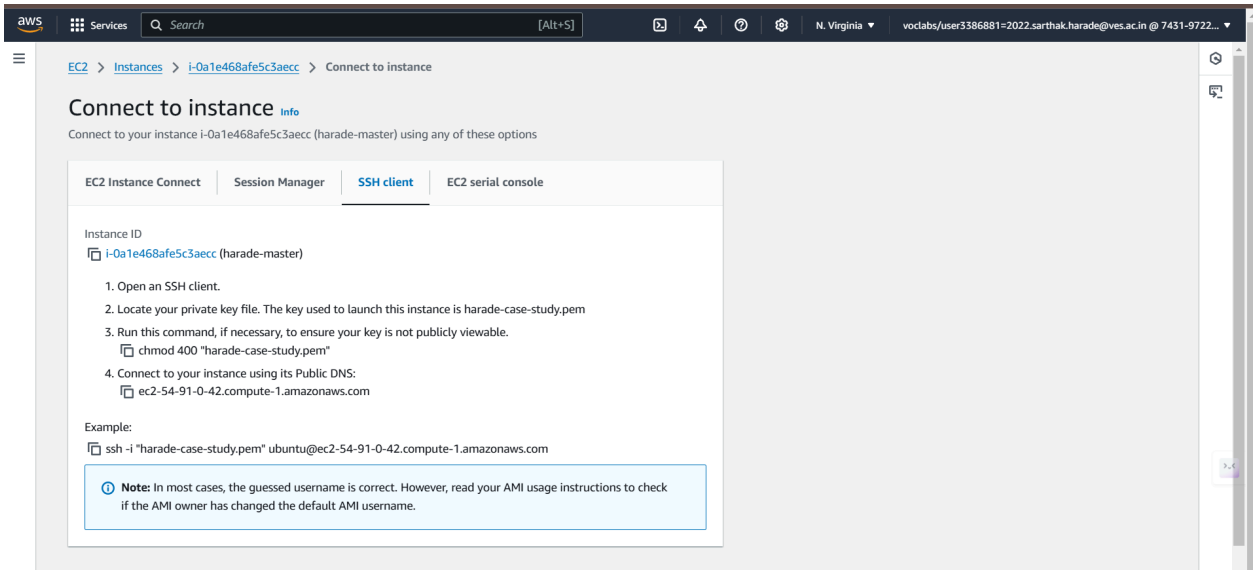


Successfully, created the 3 ec2 instances (1 for master & 2 for nodes).

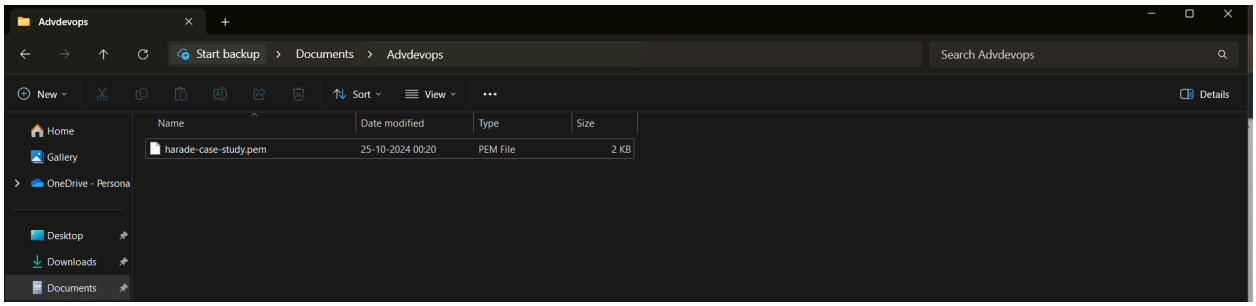


Connect to the Instances

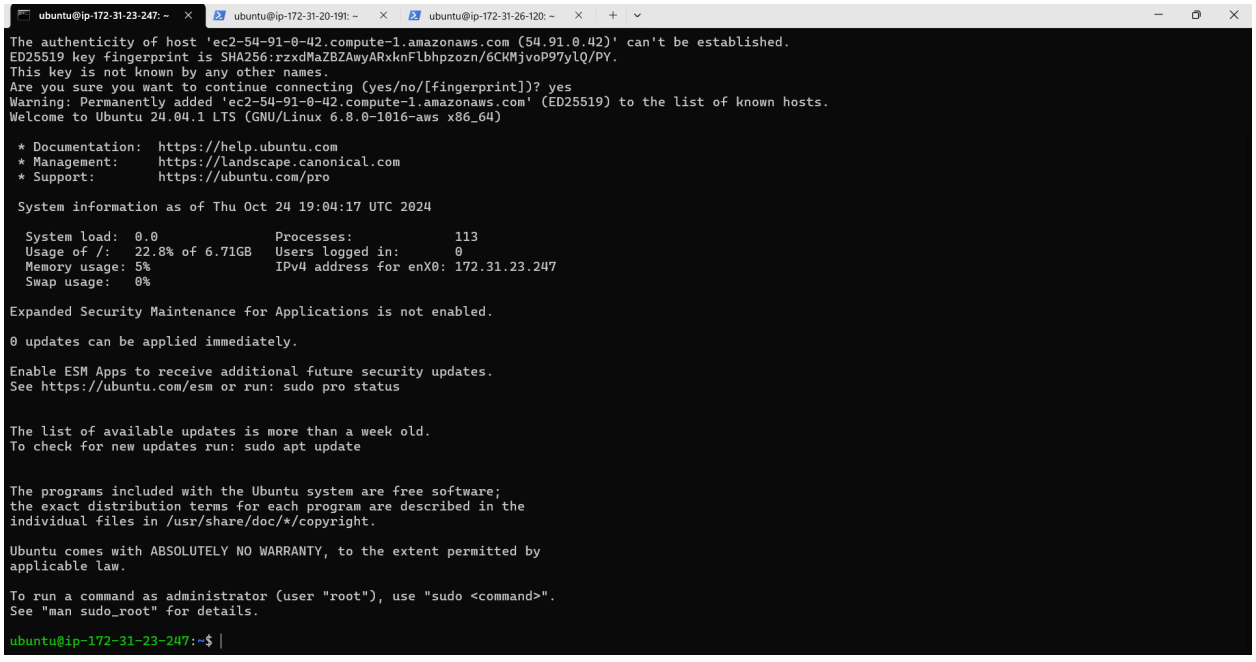
After launching the instances, go to the EC2 dashboard, select each instance, and click Connect to get the SSH command.



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



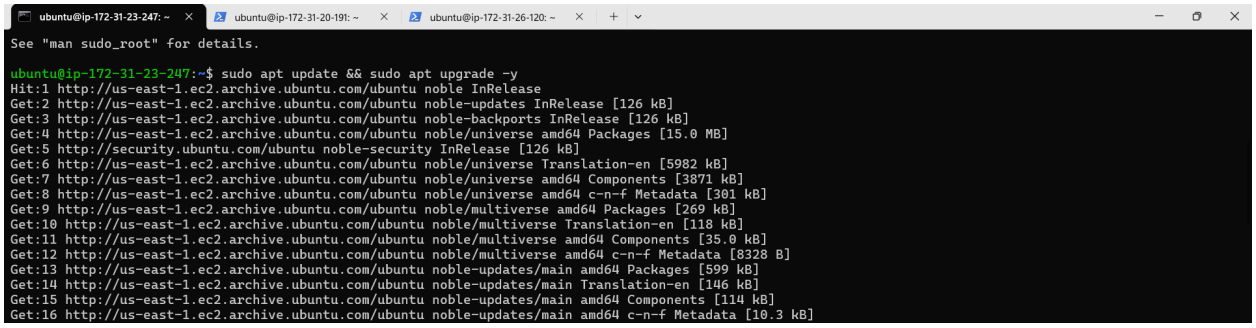
Open your terminal,
navigate to the folder where your .pem key is stored, and connect to each instance using:
ssh -i "your-key.pem" ubuntu@your-ec2-public-ip



Update the Instances:

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

sudo apt update && sudo apt upgrade -y



Install Docker on All Instances (Master and Nodes)

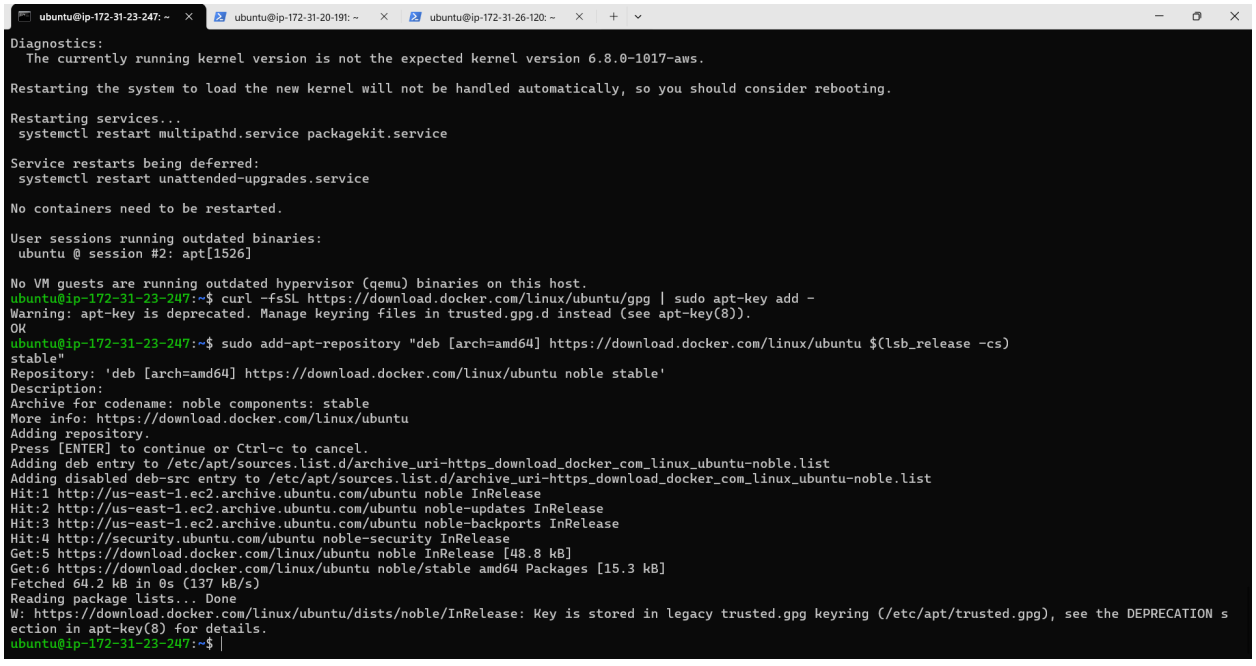
On each instance (Master, Node 1, Node 2), run the following commands to install Docker:

Add Docker's official GPG key

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

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

```
ubuntu@ip-172-31-26-120:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [15.3 kB]
Fetched 64.2 kB in 0s (148 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION s
ection in apt-key(8) for details.
ubuntu@ip-172-31-26-120:~$
```

```
ubuntu@ip-172-31-23-247: ~   ubuntu@ip-172-31-20-191: ~   ubuntu@ip-172-31-26-120: ~

 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #2: apt[1526]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-23-247:~$ 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-172-31-23-247:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:5 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [15.3 kB]
Fetched 64.2 kB in 0s (137 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION s
ection in apt-key(8) for details.
ubuntu@ip-172-31-23-247:~$ 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 s
ection in apt-key(8) for details.
ubuntu@ip-172-31-23-247:~$
```

Search

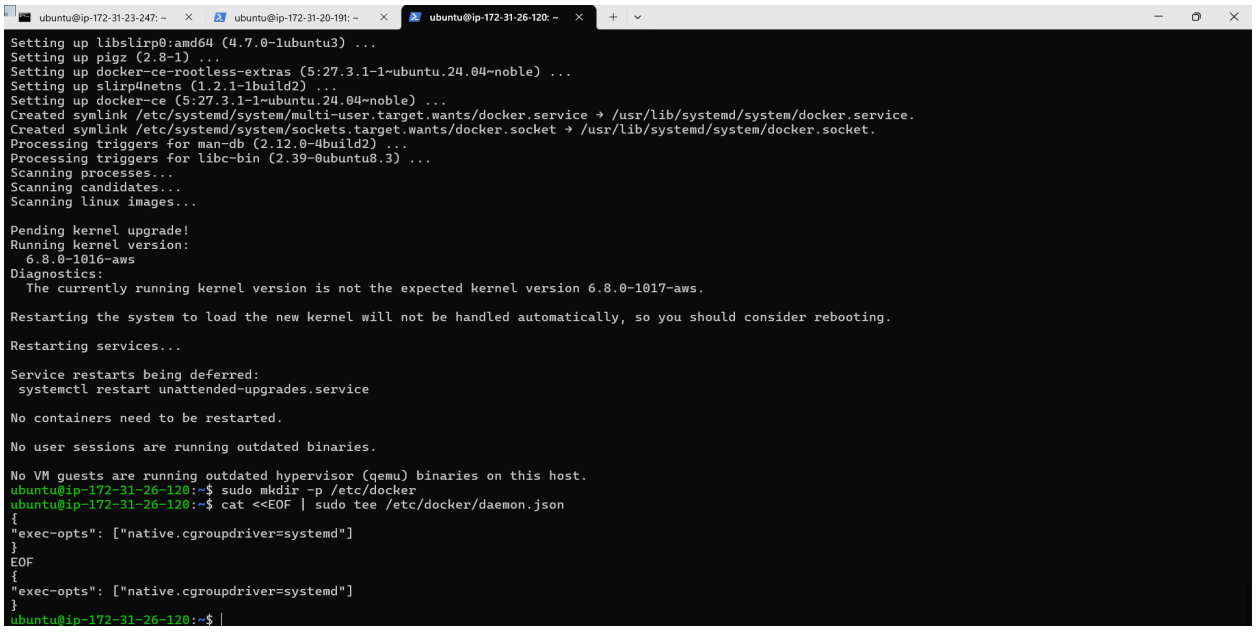
# Install Docker CE (Community Edition)
sudo apt-get install -y docker-ce

```
ubuntu@ip-172-31-23-247: ~   ubuntu@ip-172-31-20-191: ~   ubuntu@ip-172-31-26-120: ~

ubuntu@ip-172-31-26-120:~$ 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:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 123 MB of archives.
After this operation, 442 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libltdl7 amd64 2.4.7-7build1 [40.3 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 slirp4netns amd64 1.2.1-1build2 [34.9 kB]
Get:5 https://download.docker.com/linux/ubuntu noble/stable amd64 containerd.io amd64 1.7.22-1 [29.5 MB]
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-buildx-plugin amd64 0.17.1-1~ubuntu.24.04~noble [30.3 MB]
Get:7 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-cli amd64 5:27.3.1-1~ubuntu.24.04~noble [15.0 MB]
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce amd64 5:27.3.1-1~ubuntu.24.04~noble [25.6 MB]
Get:9 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-rootless-extras amd64 5:27.3.1-1~ubuntu.24.04~noble [9588 kB]
Get:10 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-compose-plugin amd64 2.29.7-1~ubuntu.24.04~noble [12.7 MB]
Fetched 123 MB in 2s (74.1 MB/s)
Selecting previously unselected package pigz.
(Reading database ... 98430 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.8-1_amd64.deb ...
Unpacking pigz (2.8-1) ...
Selecting previously unselected package containerd.io.
Preparing to unpack .../1-containerd.io_1.7.22-1_amd64.deb ...
```

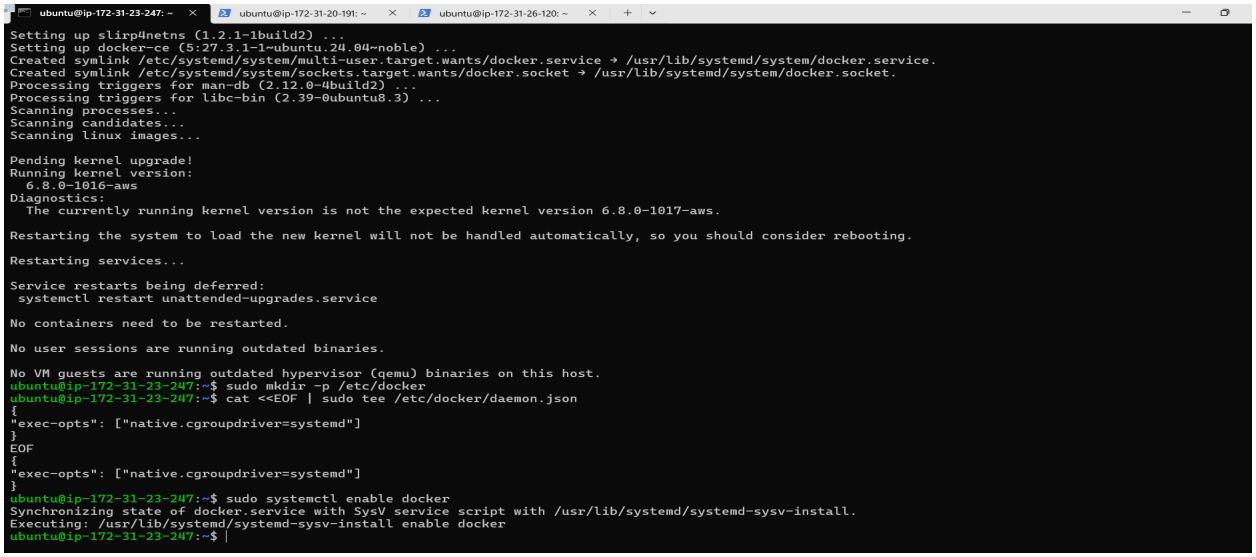
# Configure Docker to use systemd as the cgroup driver
sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{

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

```
ubuntu@ip-172-31-23-247:~$ sudo mkdir -p /etc/docker
ubuntu@ip-172-31-23-247:~$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-23-247:~$
```



Enable and start Docker sudo systemctl enable docker

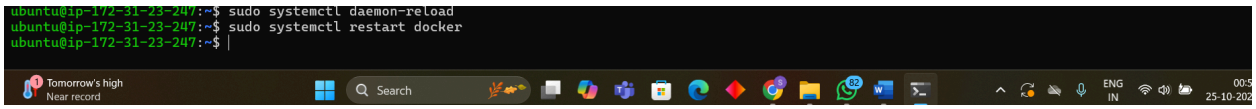


sudo systemctl daemon-reload

```
ubuntu@ip-172-31-23-247:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-23-247:~$
```

sudo systemctl restart docker

```
ubuntu@ip-172-31-23-247:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-23-247:~$ sudo systemctl restart docker
ubuntu@ip-172-31-23-247:~$
```



Install Kubernetes on All Instances

Run the below command to install Kubernets.

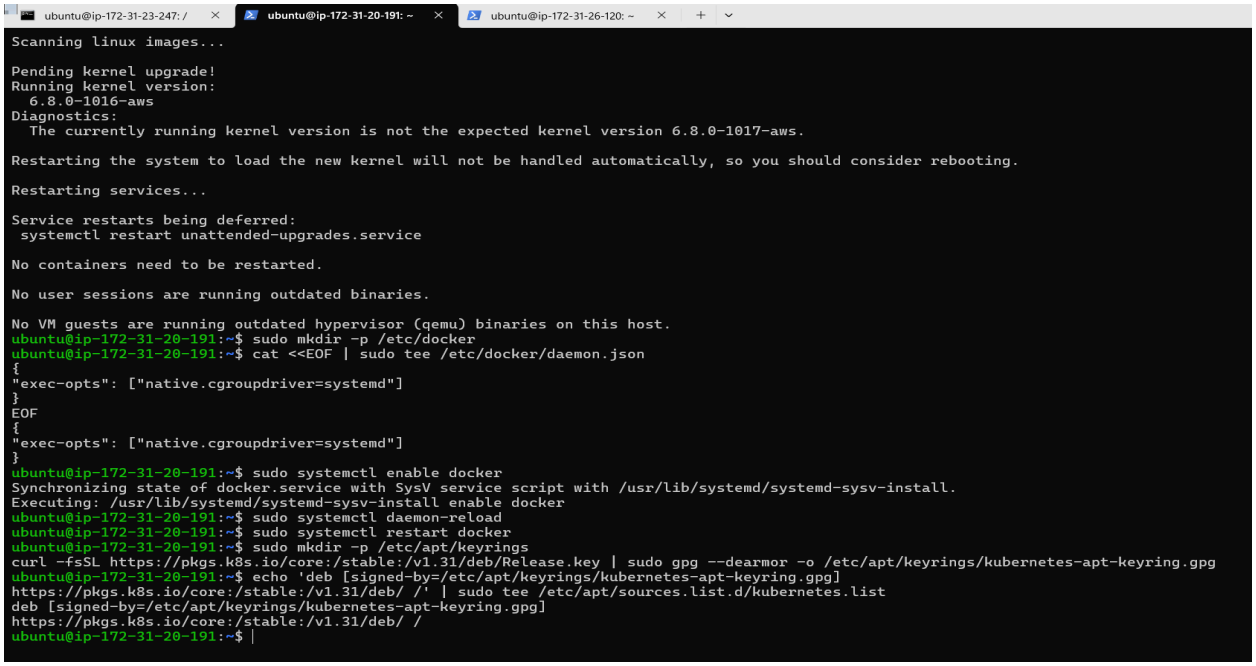
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

```
ubuntu@ip-172-31-23-247:/$ sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
File '/etc/apt/keyrings/kubernetes-apt-keyring.gpg' exists. Overwrite? (y/N) y
ubuntu@ip-172-31-23-247:/$
```

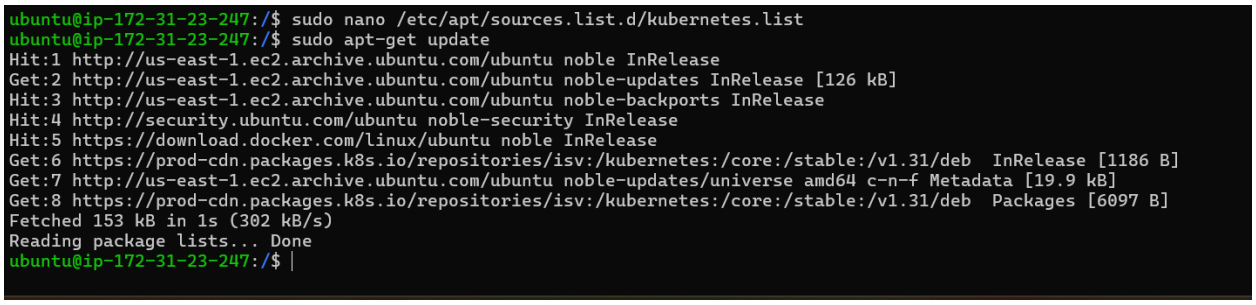
```
ubuntu@ip-172-31-23-247: /   ubuntu@ip-172-31-20-191: ~   ubuntu@ip-172-31-26-120: ~

Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-1016-aws
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-1017-aws.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-26-120:~$ sudo mkdir -p /etc/docker
ubuntu@ip-172-31-26-120:~$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-26-120:~$ 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-172-31-26-120:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-26-120:~$ sudo systemctl restart docker
ubuntu@ip-172-31-26-120:~$ sudo mkdir -p /etc/apt/keyrings
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-172-31-26-120:~$
```

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

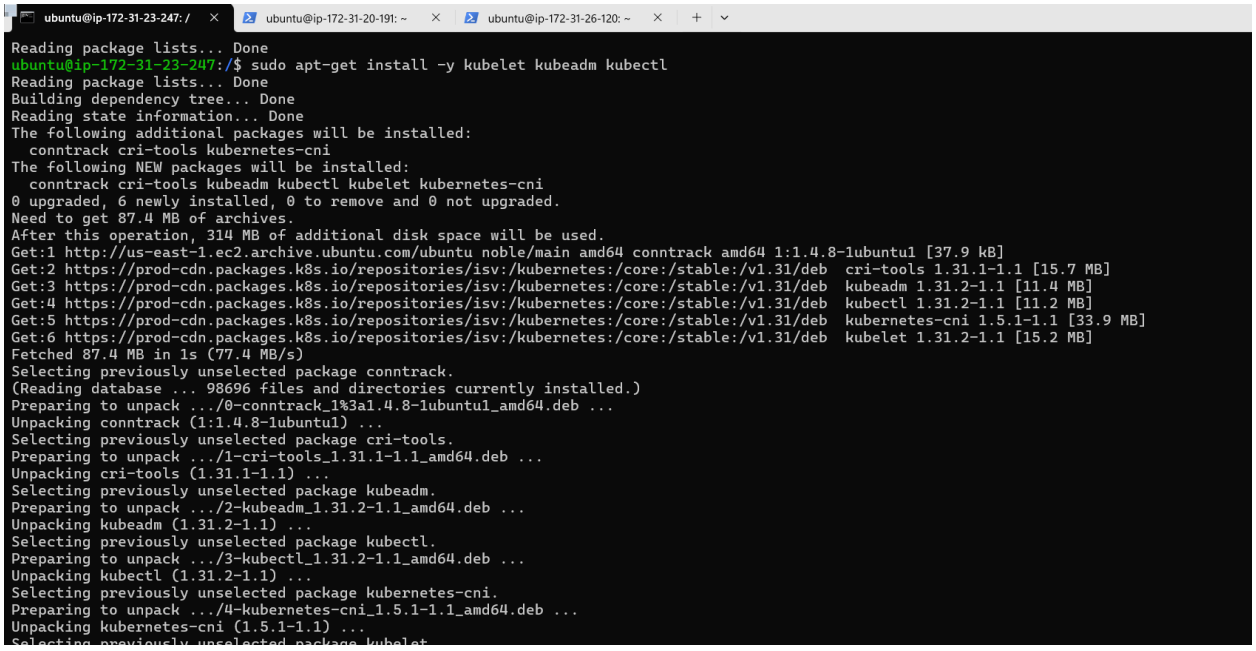
```
ubuntu@ip-172-31-23-247:/$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
ubuntu@ip-172-31-23-247:/$
```

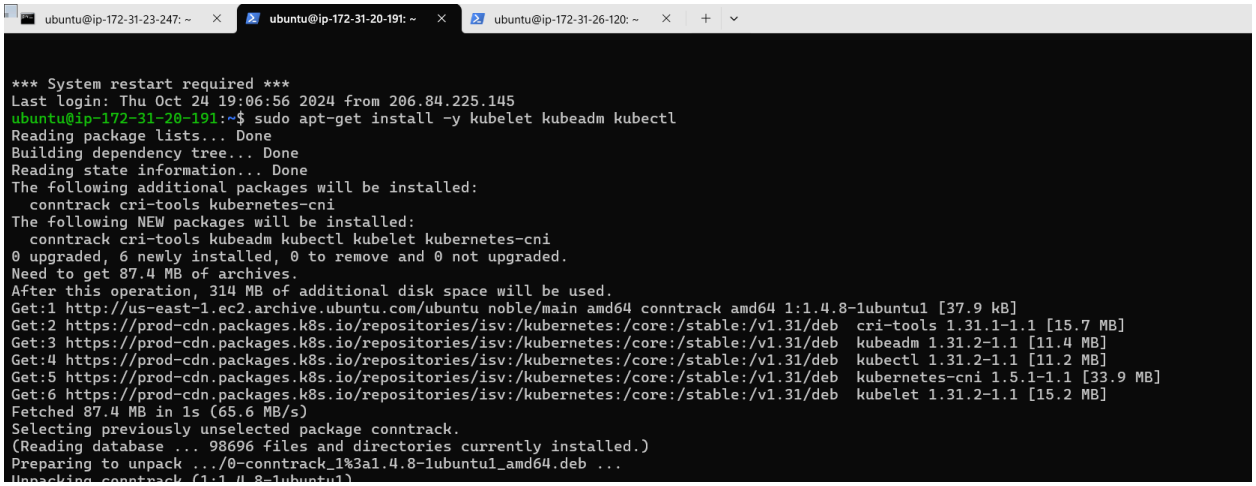


sudo apt-get update

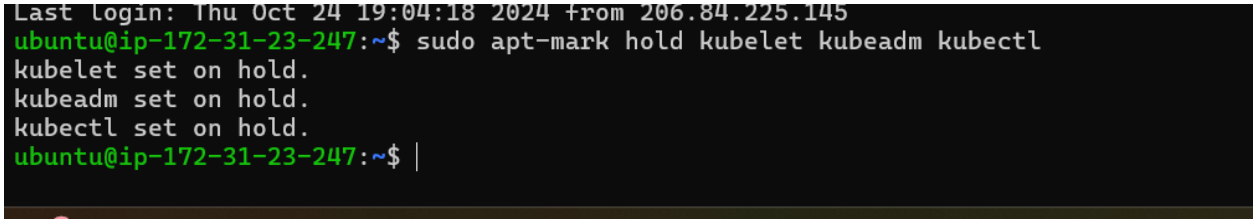


sudo apt-get install -y kubelet kubeadm kubectl

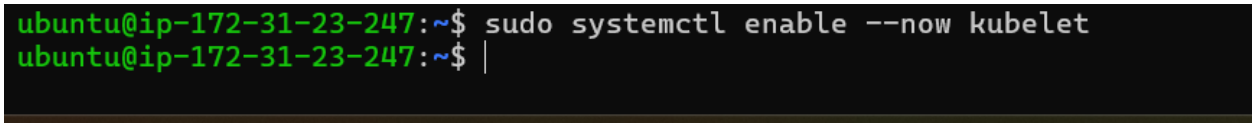




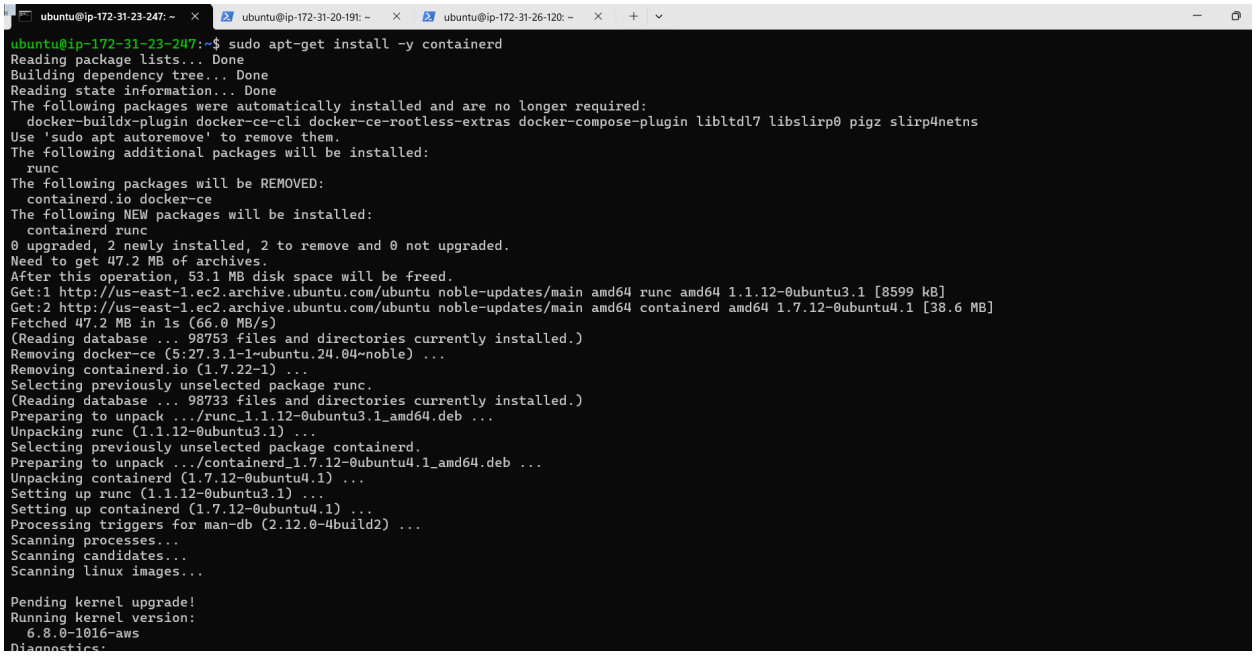
sudo apt-mark hold kubelet kubeadm kubectl



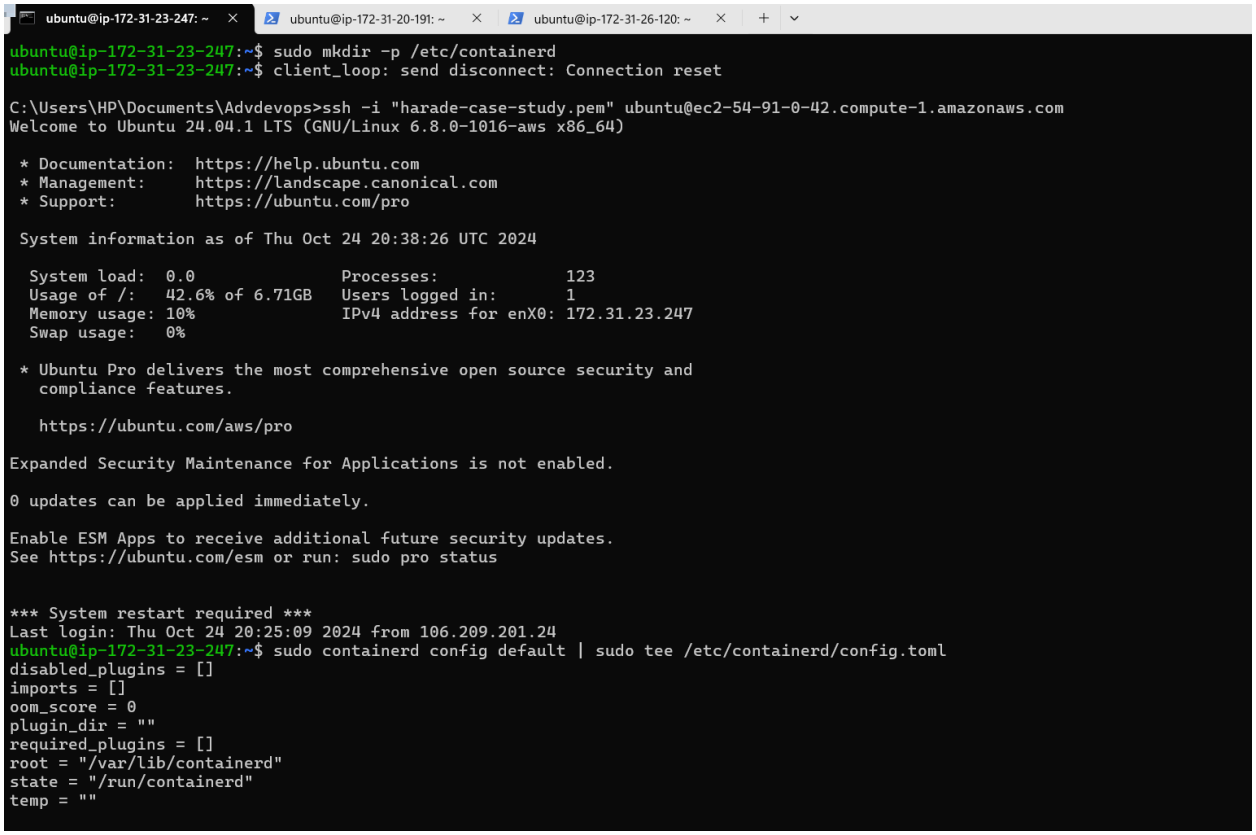
sudo systemctl enable --now kubelet



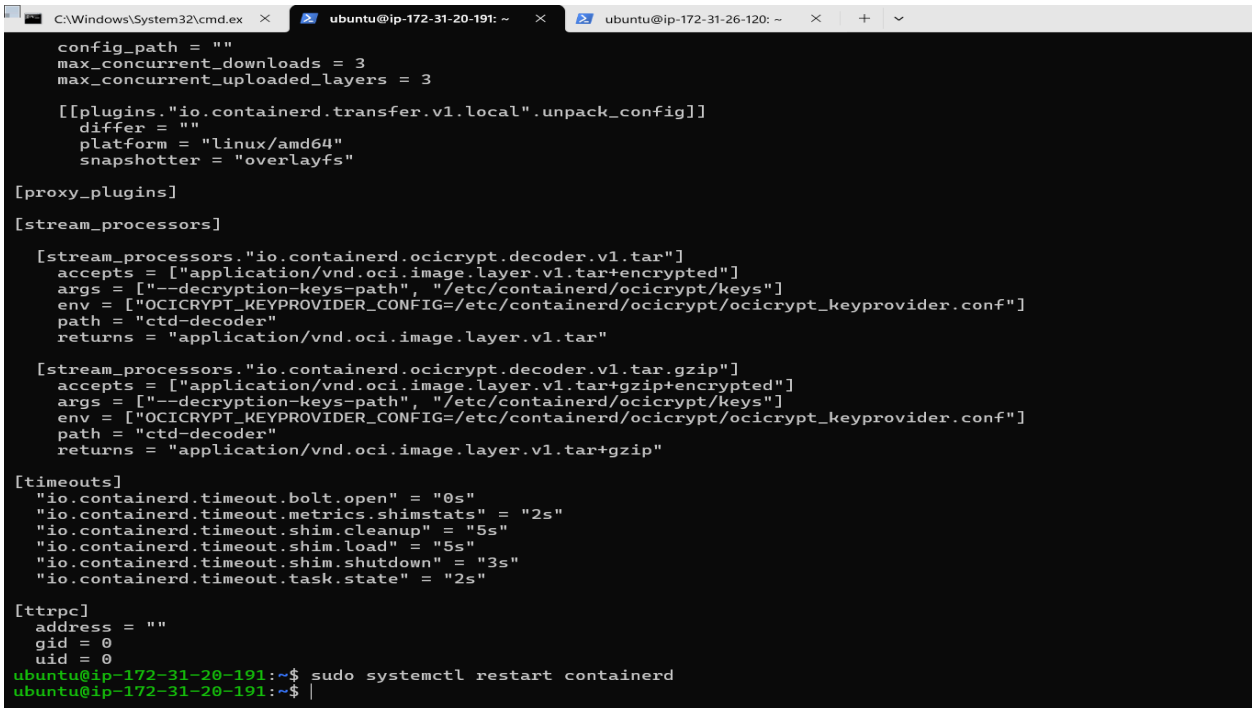
sudo apt-get install -y containerd



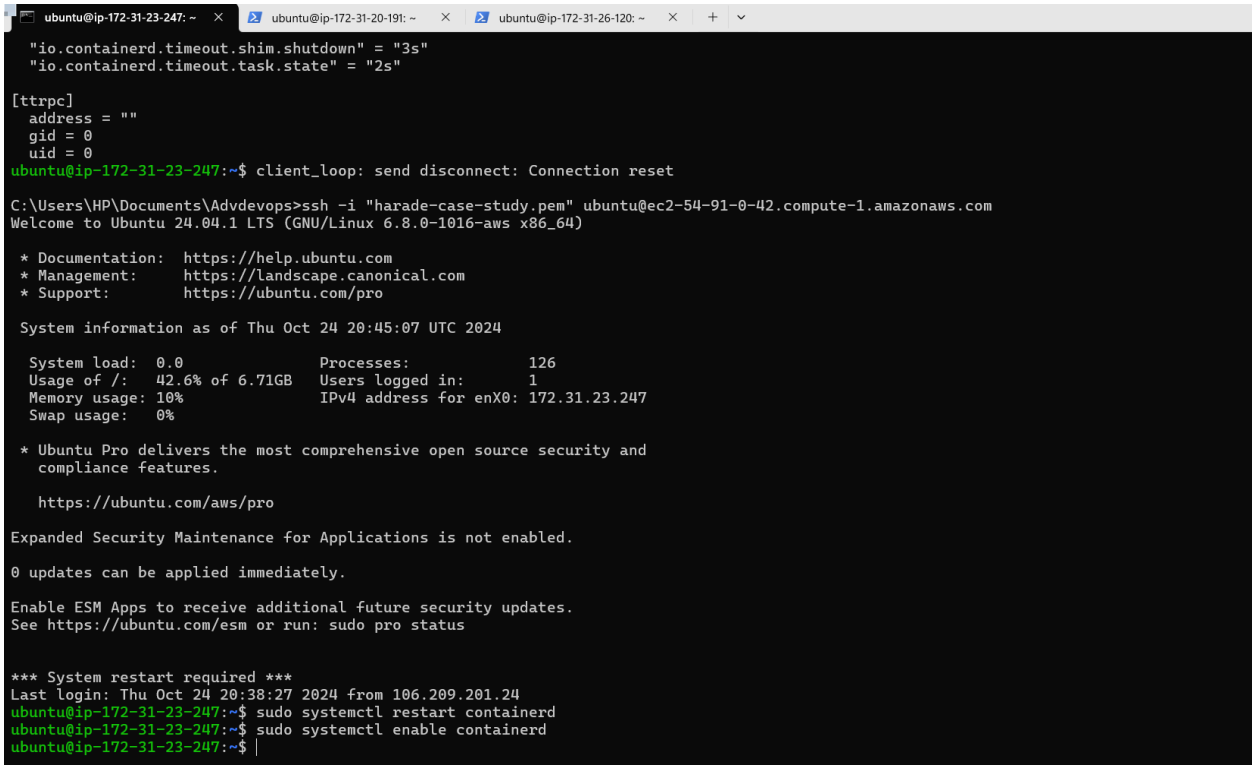
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml



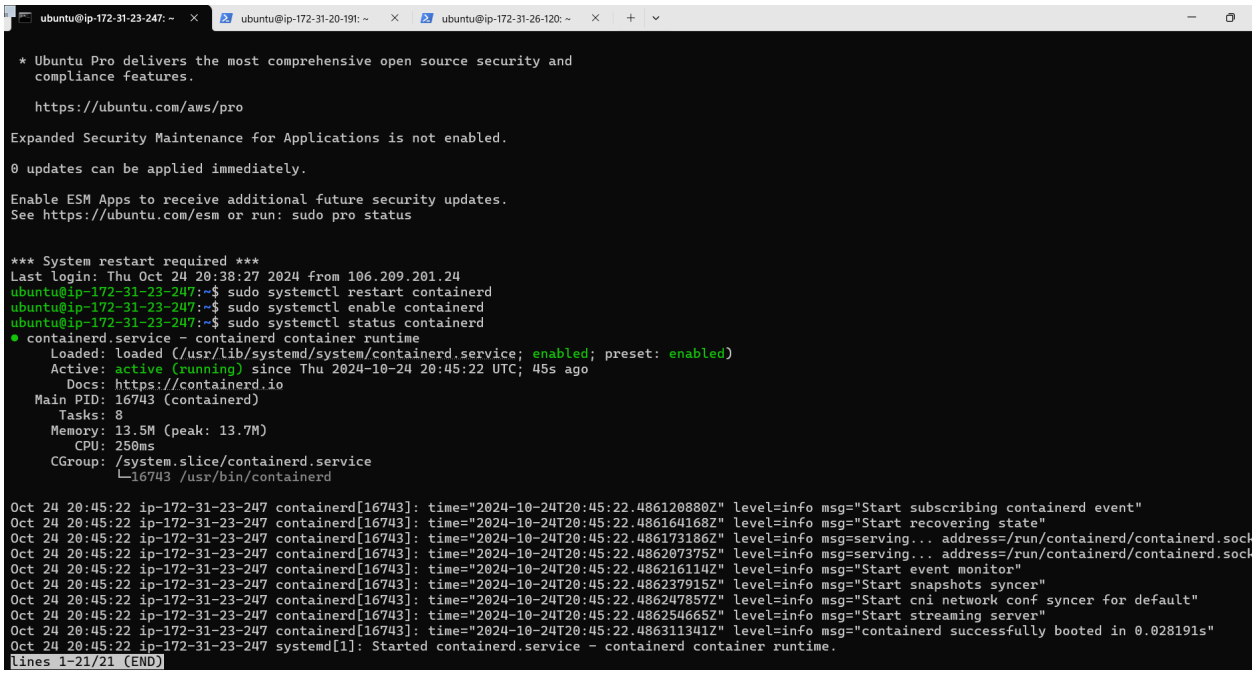
sudo systemctl restart containerd



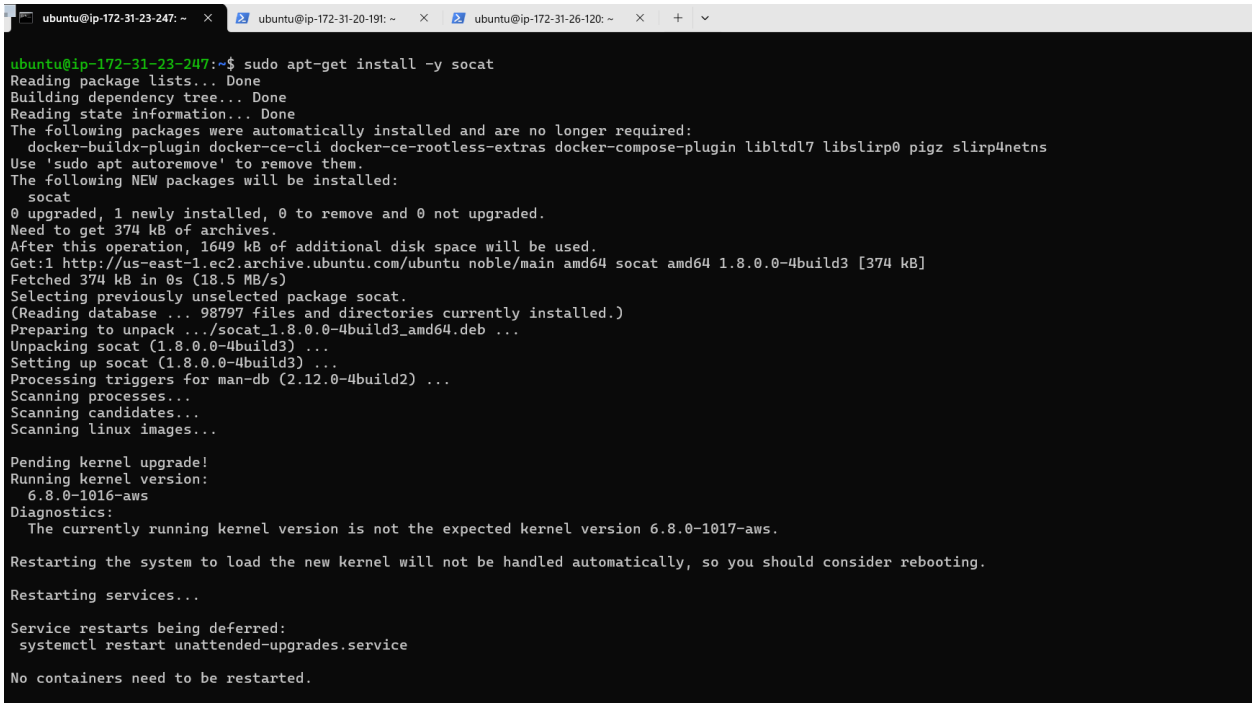
sudo systemctl enable containerd



sudo systemctl status containerd



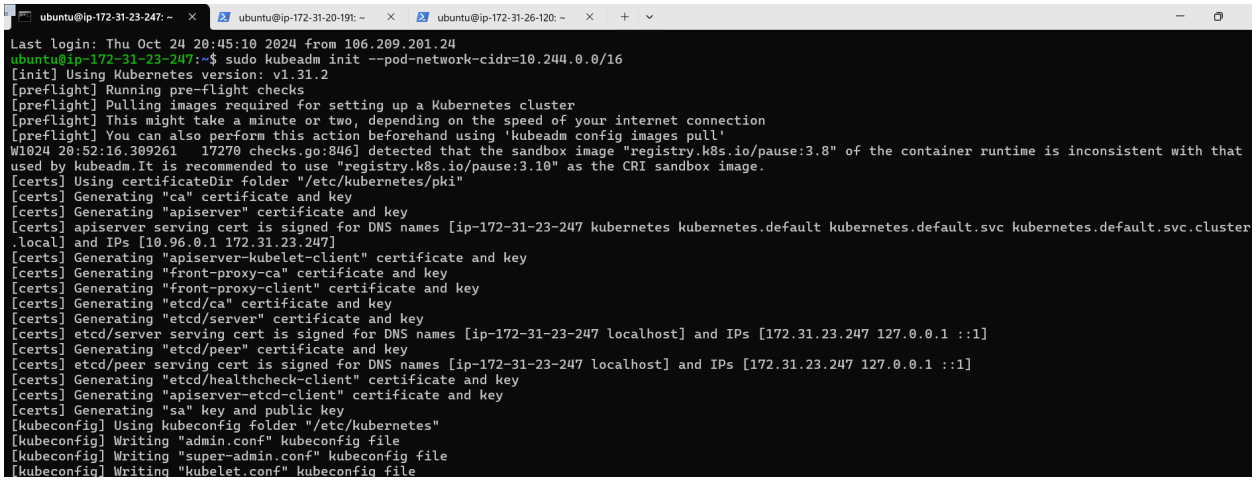
sudo apt-get install -y socat



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:

```
sudo kubeadm join 172.31.23.247:6443 --token lgl6o7.qmk75sn9uifh3131 \
        --discovery-token-ca-cert-hash sha256:372ecad85c51a779a6e7e587a28d7ac671709e434256bd83f5dbddd760f22a71
```

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

```
ubuntu@ip-172-31-23-247:~$  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-23-247:~$
```

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

```
ubuntu@ip-172-31-23-247:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE    VERSION
ip-172-31-23-247   NotReady   control-plane   3m2s   v1.31.2
ubuntu@ip-172-31-23-247:~$
```

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 :6443 --token --discovery-token-ca-cert-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)

```
ubuntu@ip-172-31-20-191:~$ sudo kubeadm join 172.31.23.247:6443 --token lgl6o7.qmk75sn9uifh3131 \
        --discovery-token-ca-cert-hash sha256:372ecad85c51a779a6e7e587a28d7ac671709e434256bd83f5dbddd760f22a71
[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 1.001799036s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver 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-172-31-20-191:~$
```

```
ubuntu@ip-172-31-23-247:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-20-191   NotReady   <none>          27s     v1.31.2
ip-172-31-23-247   NotReady   control-plane   5m14s   v1.31.2
ubuntu@ip-172-31-23-247:~$
```

```
ubuntu@ip-172-31-26-120:~$ sudo kubeadm join 172.31.23.247:6443 --token lgl6o7.qmk75sn9uifh3131 \
        --discovery-token-ca-cert-hash sha256:372ecad85c51a779a6e7e587a28d7ac671709e434256bd83f5dbddd760f22a71
[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 1.001705578s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver 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-172-31-26-120:~$
```

Verify the Cluster

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

```
ubuntu@ip-172-31-23-247:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-20-191   NotReady   <none>          72s     v1.31.2
ip-172-31-23-247   NotReady   control-plane   5m59s   v1.31.2
ip-172-31-26-120   NotReady   <none>          7s      v1.31.2
ubuntu@ip-172-31-23-247:~$
```

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-172-31-23-247:~$ 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
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
```

sudo systemctl status kubelet

```
ubuntu@ip-172-31-23-247:~$ 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 Thu 2024-10-24 20:52:42 UTC; 7min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 17937 (kubelet)
      Tasks: 10 (limit: 4676)
     Memory: 32.7M (peak: 33.2M)
        CPU: 7.880s
     CGroup: /system.slice/kubelet.service
             └─17937 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var>

Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]: E1024 21:00:12.943609   17937 kuberuntime_container.go:851] "Kill container failed" err=<
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:         rpc error: code = Unknown desc = failed to kill container "2ab7eb8c9ff9dca281d39a62d0693d2b9a4d164>
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:         : unknown
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:  > pod="kube-system/etcd-ip-172-31-23-247" podUID="fabe0b309d193997278b5c4e8d268e2f" containerName="etcd" >
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]: E1024 21:00:12.951620   17937 log.go:32] "StopPodSandbox from runtime service failed" err=<
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:         rpc error: code = Unknown desc = failed to stop container "2ab7eb8c9ff9dca281d39a62d0693d2b9a4d164>
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:         : unknown
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]:  > podSandboxID="838e9b8e4dc2f7fd92dd2659096bfbc354dff85b406a6fa71d9705bbc422f53d"
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]: E1024 21:00:12.951667   17937 kuberuntime_manager.go:1479] "Failed to stop sandbox" podSandboxID={"Type":">
Oct 24 21:00:12 ip-172-31-23-247 kubelet[17937]: E1024 21:00:12.951707   17937 kubelet.go:1865] "KillPod failed" err="[failed to \"KillContainer\" for \"et>
lines 1-23/23 (END)
```

Now Run command (we can see Status is ready).

kubectl get nodes -o wide

```
ubuntu@ip-172-31-23-247:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE     VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-20-191   Ready    <none>          3m28s   v1.31.2   172.31.20.191    <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-23-247   Ready    control-plane   8m15s   v1.31.2   172.31.23.247    <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-26-120   Ready    <none>          2m23s   v1.31.2   172.31.26.120    <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ubuntu@ip-172-31-23-247:~$
```

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-172-31-23-247:~$ kubectl label node ip-172-31-20-191 kubernetes.io/role=Node1
node/ip-172-31-20-191 labeled
ubuntu@ip-172-31-23-247:~$ kubectl label node ip-172-31-26-120 kubernetes.io/role=Node2
node/ip-172-31-26-120 labeled
ubuntu@ip-172-31-23-247:~$
```

Check Again

Run (again to confirm all nodes are in the Ready state and properly labeled).

kubectl get nodes -o wide

```
ubuntu@ip-172-31-23-247:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-20-191   Ready    Node1           8m32s   v1.31.2   172.31.20.191   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-23-247   Ready    control-plane   13m     v1.31.2   172.31.23.247   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ip-172-31-26-120   Ready    Node2           7m27s   v1.31.2   172.31.26.120   <none>        Ubuntu 24.04.1 LTS   6.8.0-1016-aws   containerd://1.7.12
ubuntu@ip-172-31-23-247:~$
```

Deploy the Sample Nginx Application

1. Deploy Nginx:

○ Create a deployment:

kubectl create deployment nginx --image=nginx

```
ubuntu@ip-172-31-23-247:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
ubuntu@ip-172-31-23-247:~$
```

Expose the Nginx Deployment:

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

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

```
ubuntu@ip-172-31-23-247:~$ kubectl expose deployment nginx --port=80 --type=NodePort
service/nginx exposed
ubuntu@ip-172-31-23-247:~$
```

Get the NodePort:

To get the NodePort: kubectl get services

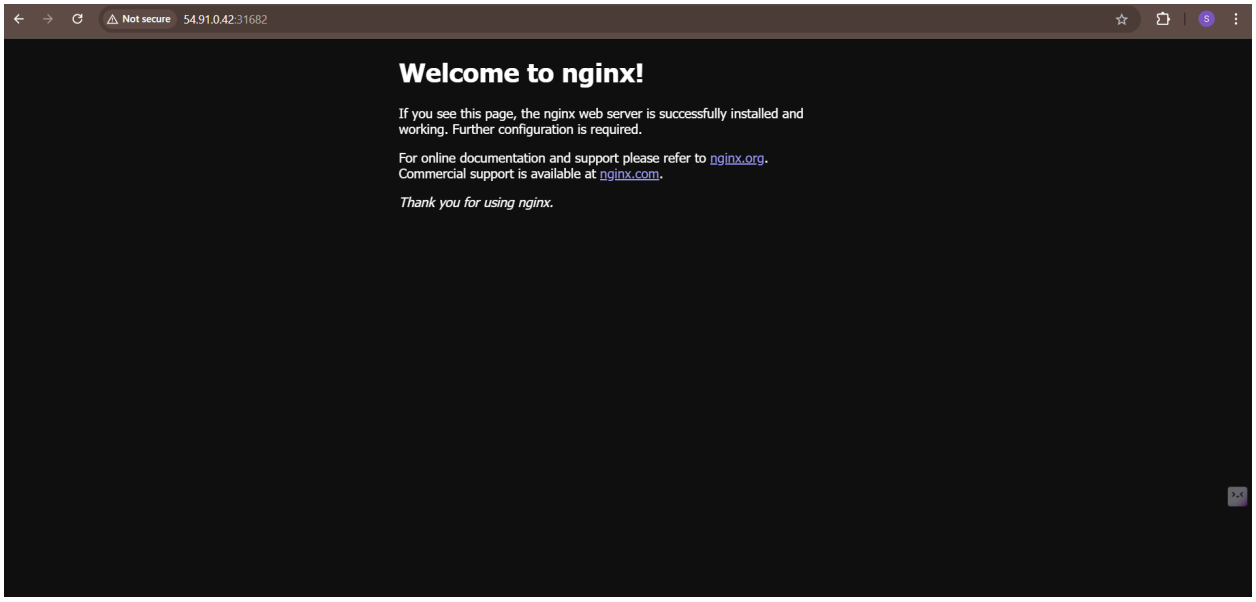
● Note the NodePort value (it will be something like 30000-32767).

```
service/nginx exposed
ubuntu@ip-172-31-23-247:~$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        16m
nginx        NodePort    10.96.126.244   <none>        80:31682/TCP   64s
ubuntu@ip-172-31-23-247:~$
```

Access the Application:

● Open the browser and visit http://public-ip of master instance:port to verify that the Nginx server is running.

```
PS C:\Users\HP> curl  http://54.91.0.42:31682

StatusCode        : 200
StatusDescription : OK
Content           : <!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...
RawContent        : HTTP/1.1 200 OK
                    Connection: keep-alive
                    Accept-Ranges: bytes
                    Content-Length: 615
                    Content-Type: text/html
                    Date: Thu, 24 Oct 2024 21:15:05 GMT
                    ETag: "66fd630f-267"
                    Last-Modified: Wed, 02 Oct 2024 ...
Forms             : {}
Headers           : {[Connection, keep-alive], [Accept-Ranges, bytes], [Content-Length, 615], [Content-Type, text/html]...}
Images            : {}
InputFields       : {}
Links             : {@{innerHTML=nginx.org; innerText=nginx.org; outerHTML=<A href="http://nginx.org/">nginx.org</A>; outerText=nginx.org; tagName=A;
                    href=http://nginx.org/}, @{innerHTML=nginx.com; innerText=nginx.com; outerHTML=<A href="http://nginx.com/">nginx.com</A>;
                    outerText=nginx.com; tagName=A; href=http://nginx.com/}}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 615



PS C:\Users\HP>
```



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.