Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

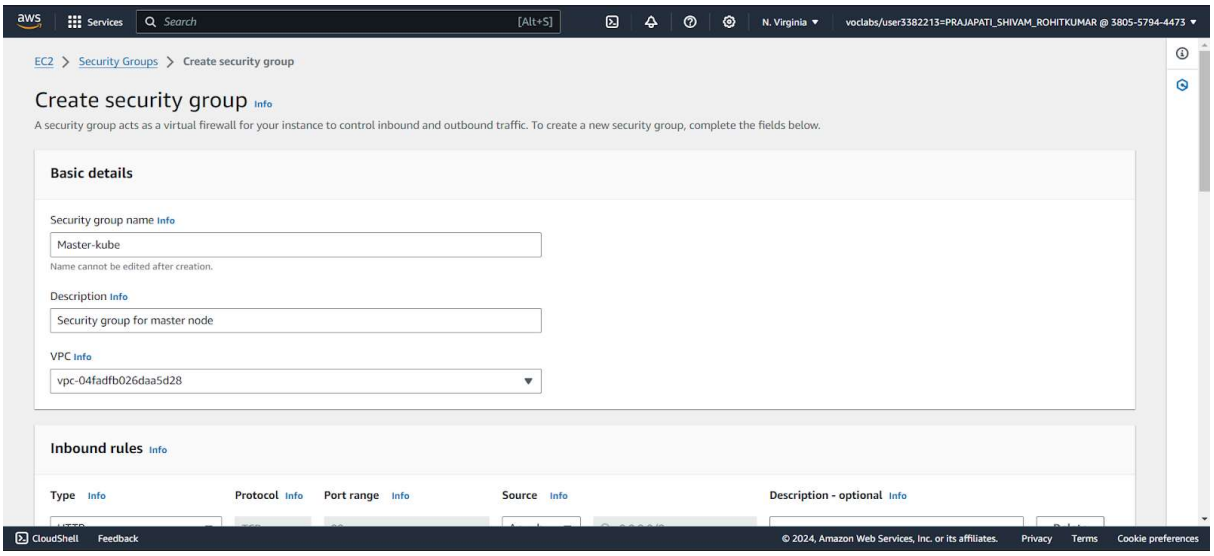
Experiment No : 3

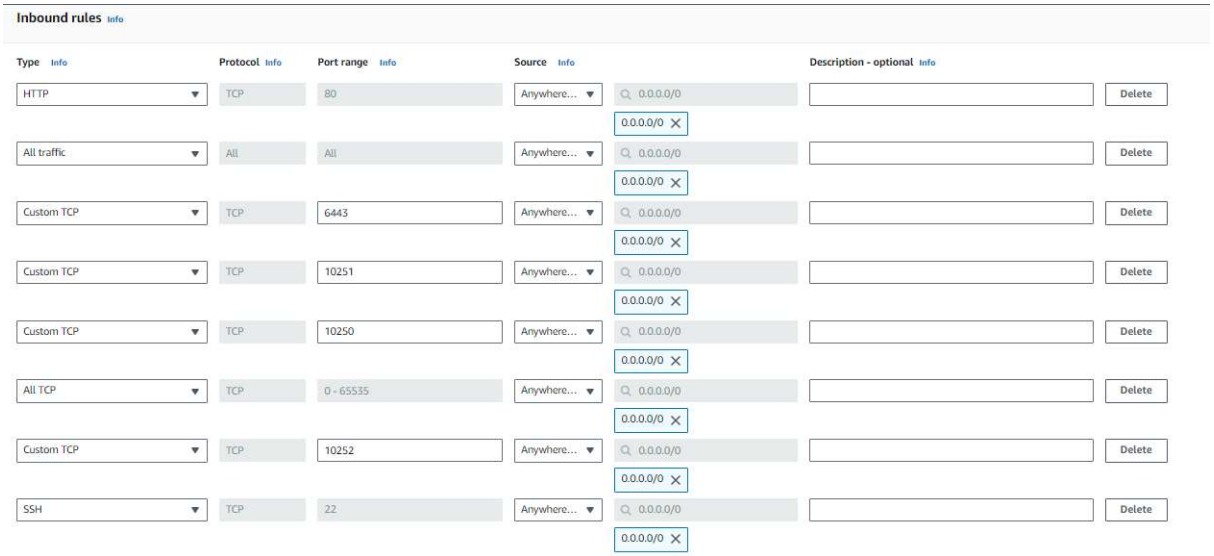
AIM: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernete Cluster on Linux Machines/Cloud Platforms.

PREREQUISITES:

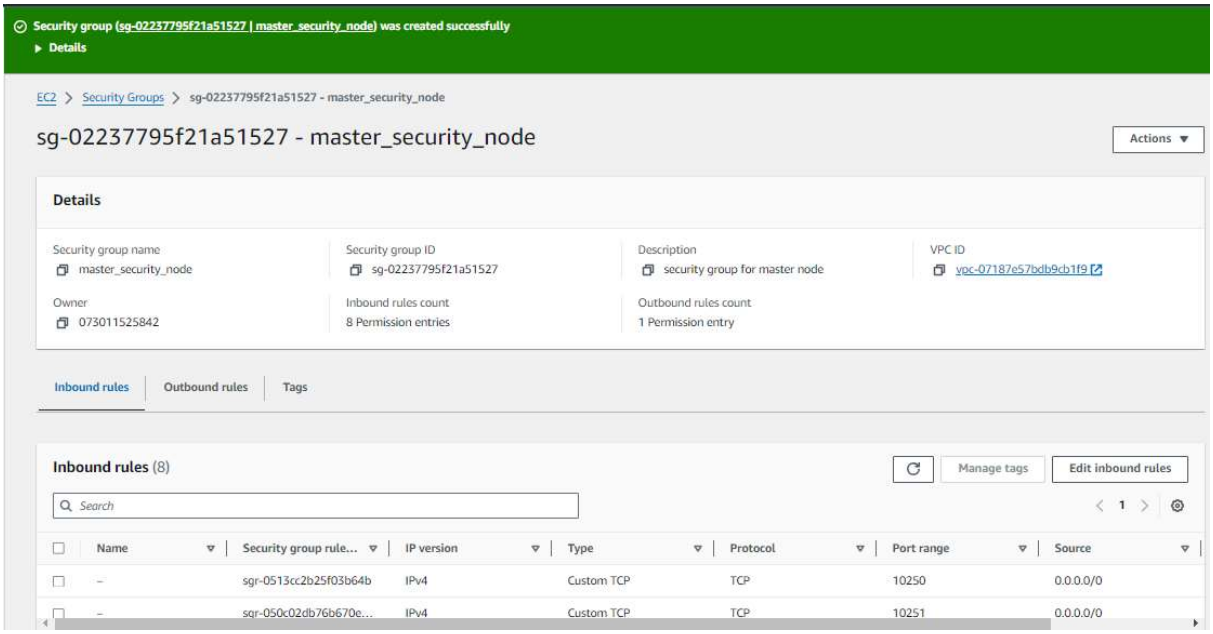
Create 2 Security Groups for Master Node and Worker Nodes and add the following inbound rules in those Groups.

Master Node Security Group:

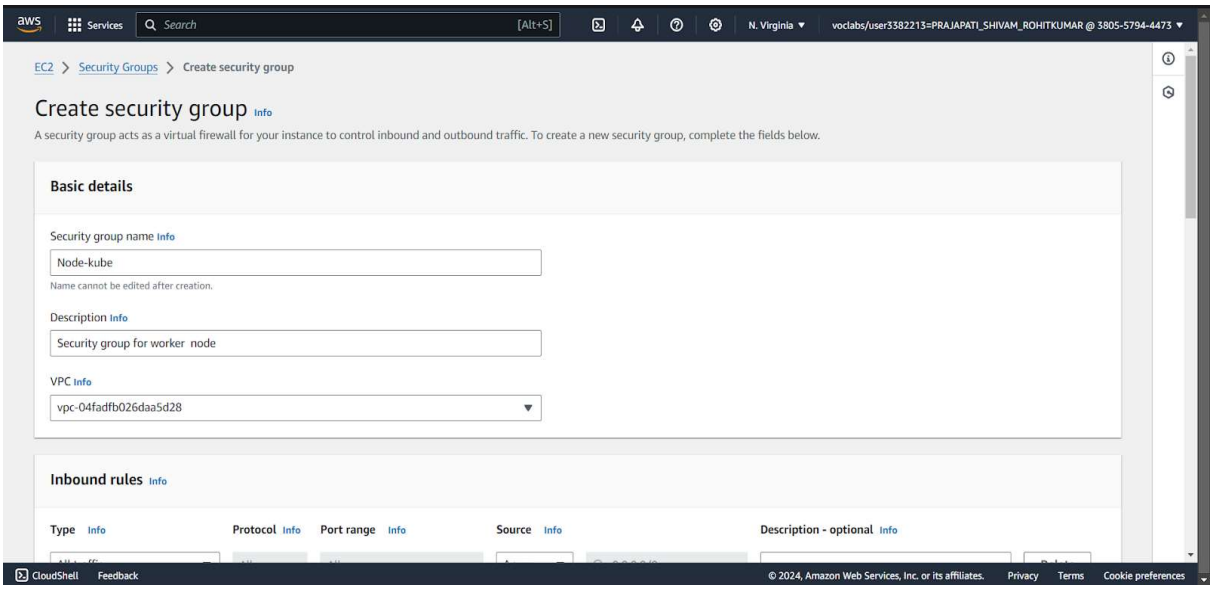


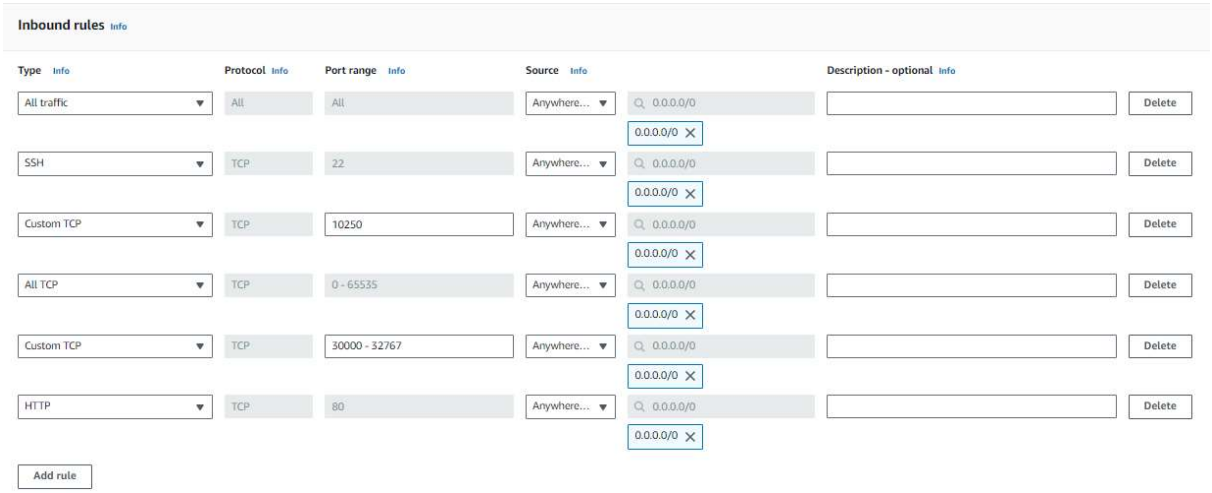


Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

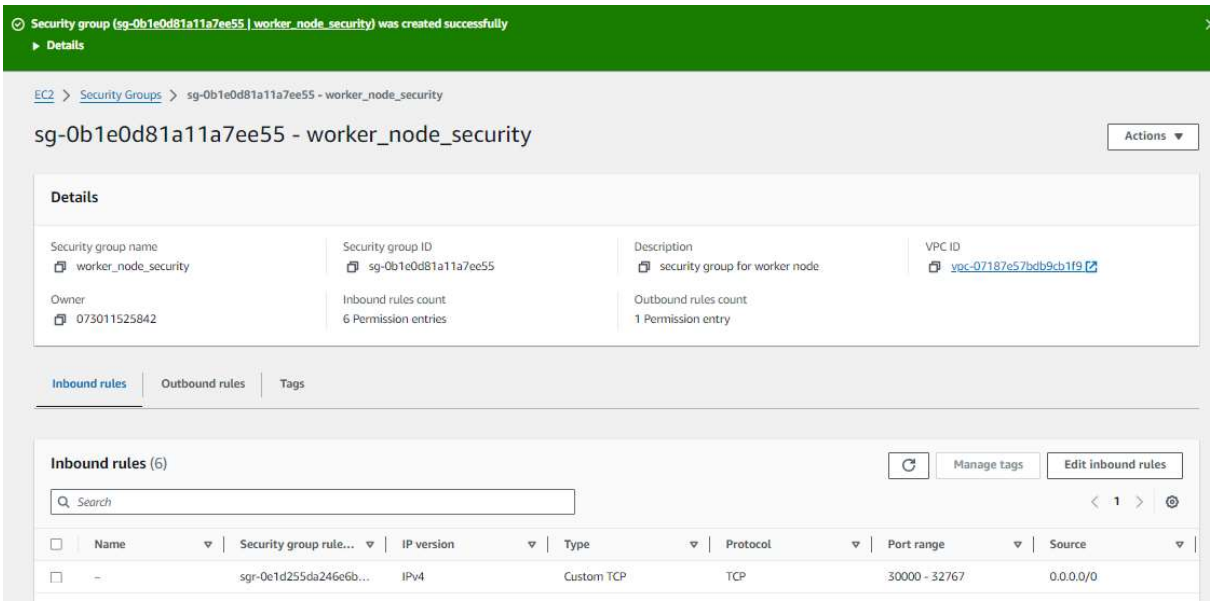


Worker Node Security Group :





Added Required Inbound rules for worker nodes

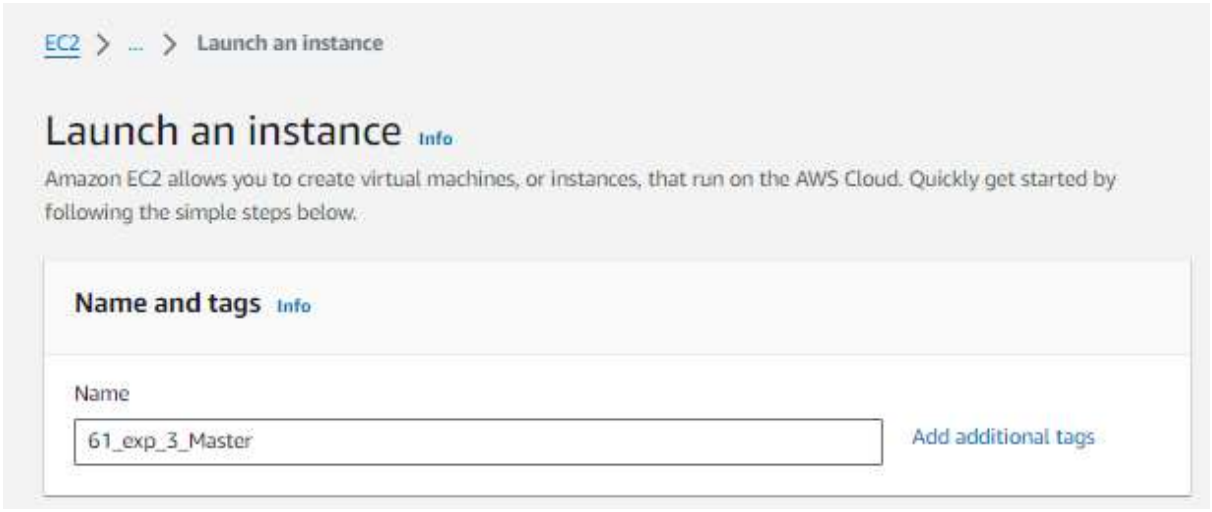


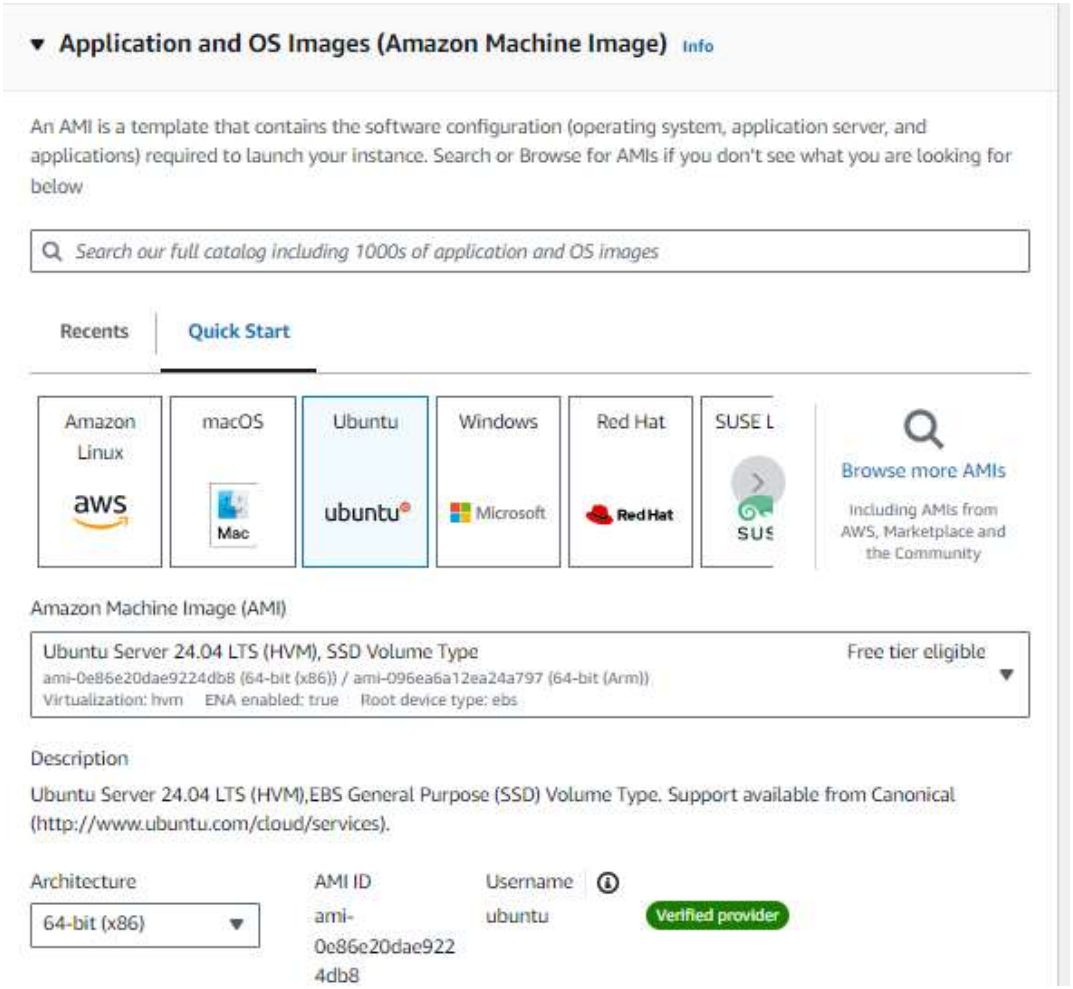
Step 1: Access your AWS Academy or personal account and create **three new EC2 instances**. For the AMI, choose **Ubuntu** and set the **instance type to t2.medium**. Generate an RSA key in .pem format and move the downloaded key to a new folder for **eg: Newfolder**; you can either use three separate keys or one shared key for all instances ,in my case I have made three different keys .

It's essential to select t2.medium, as it includes at least 2 CPUs, which are required for this setup.. Additionally, make sure to select security groups from the ones already

available i.e master node and worker node will select their own security groups respectively

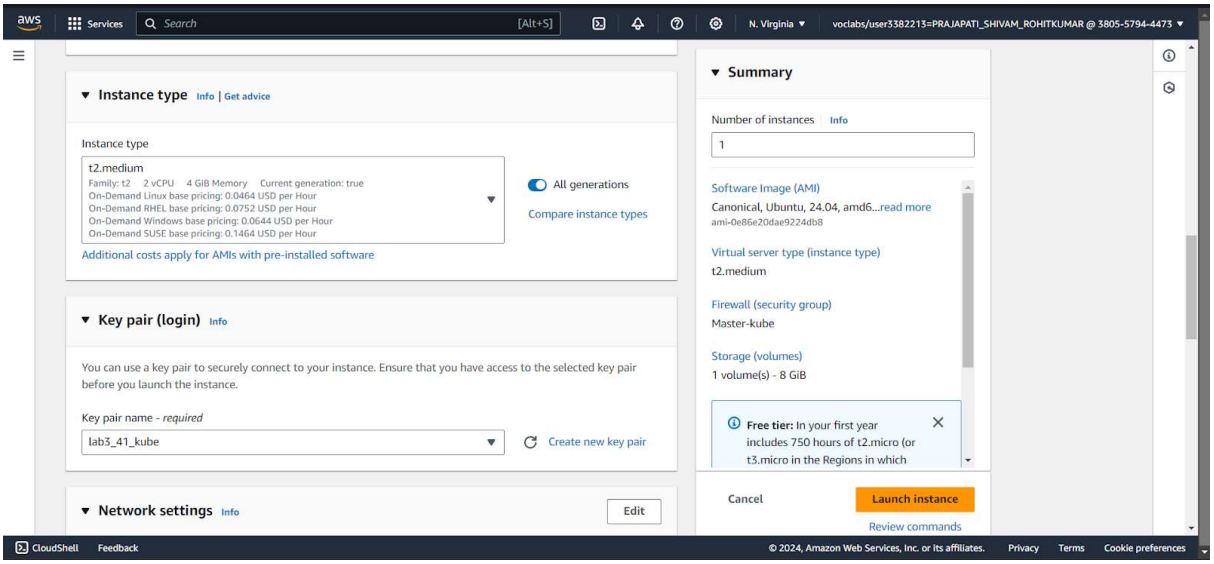
Master Node:



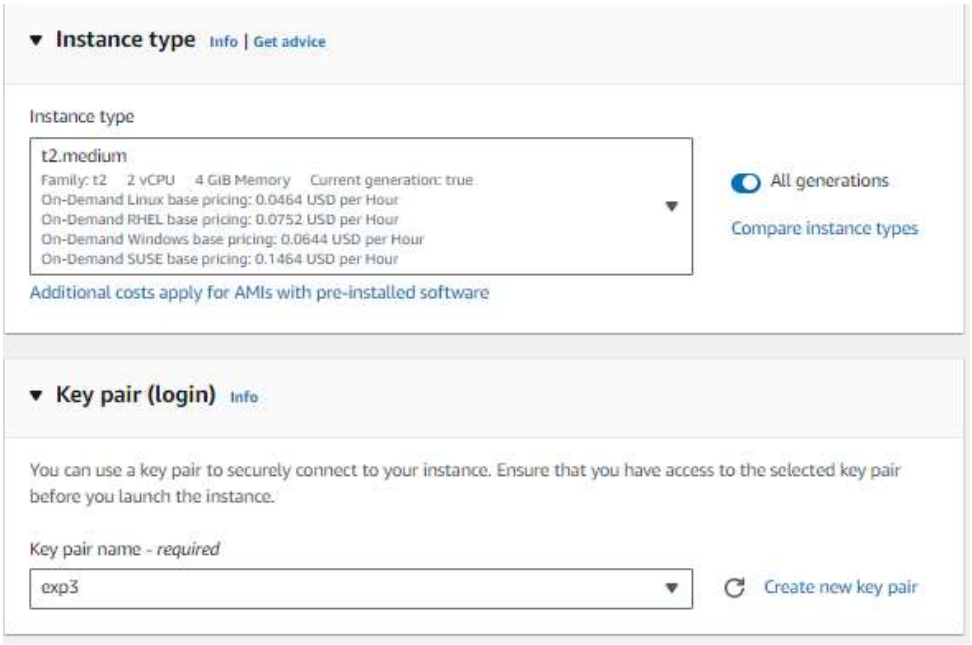


AMI as Ubuntu

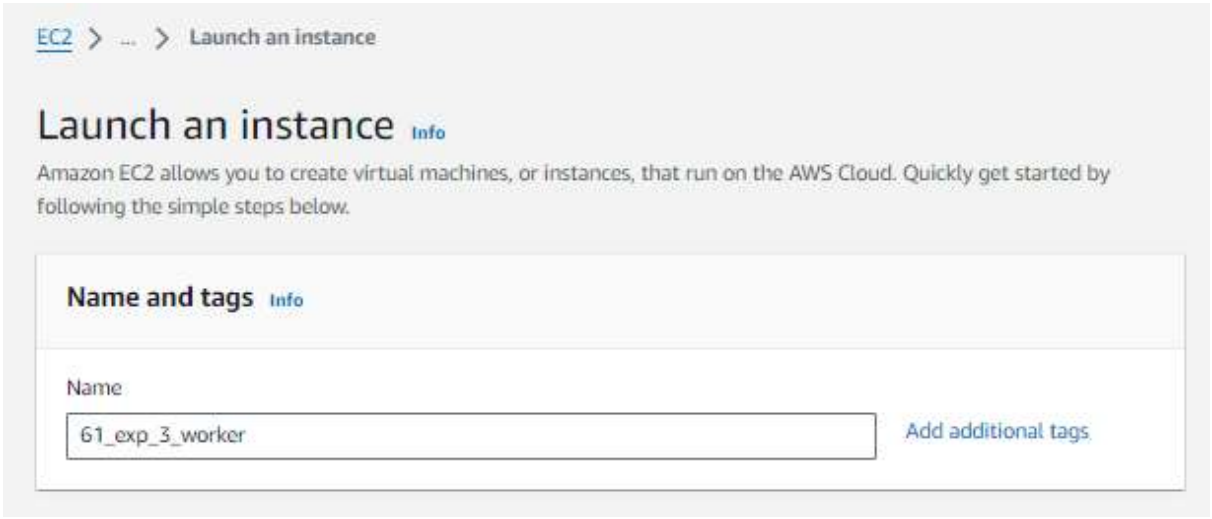
Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**



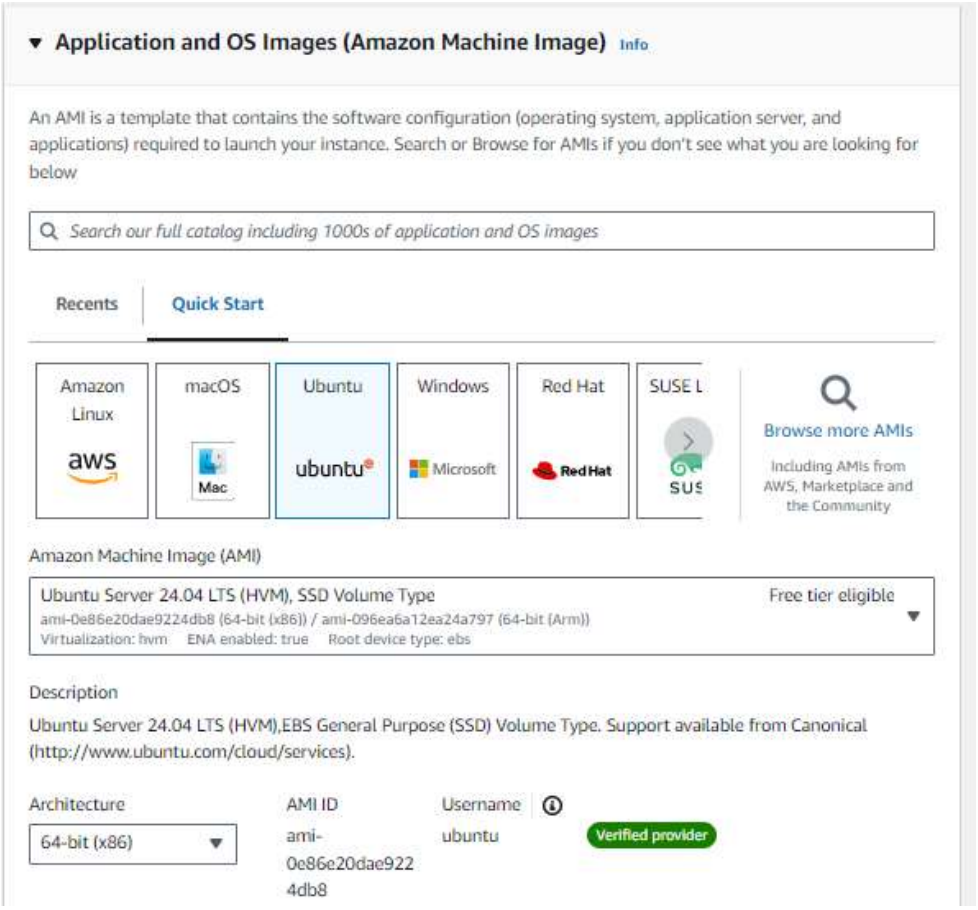
Generate Instance type and create key pair login



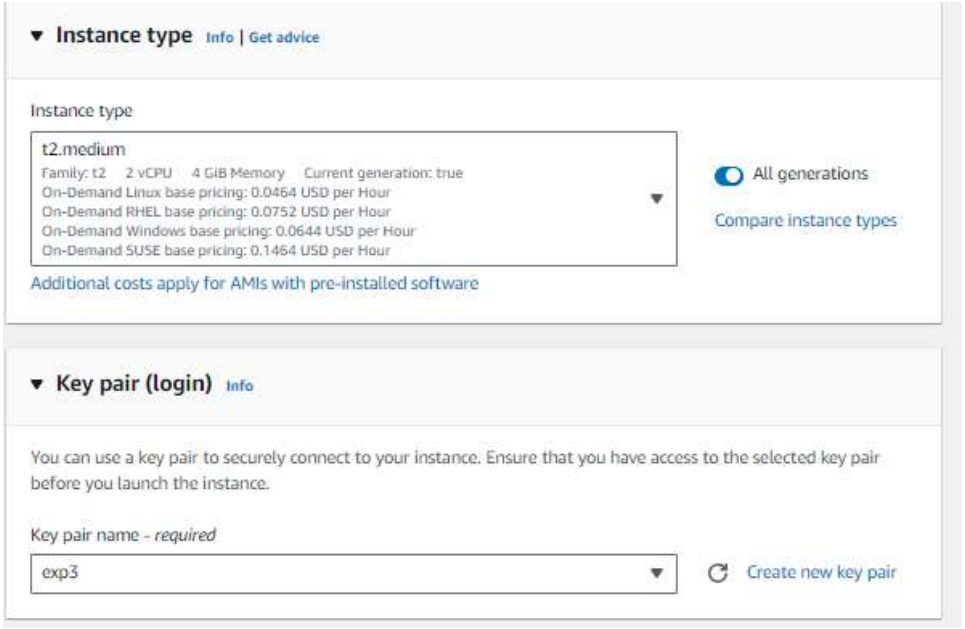
Worker Node 1:



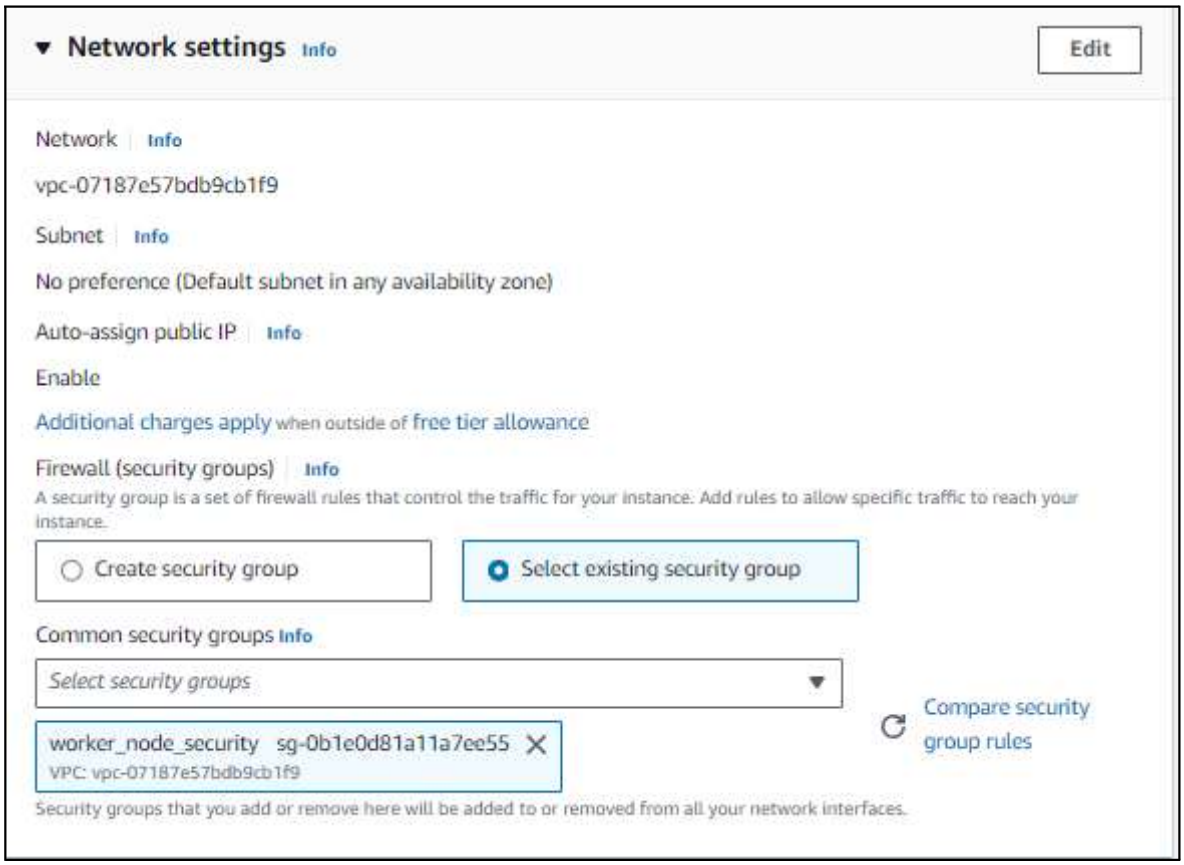
Give a name to your instance .



AMI as Ubuntu



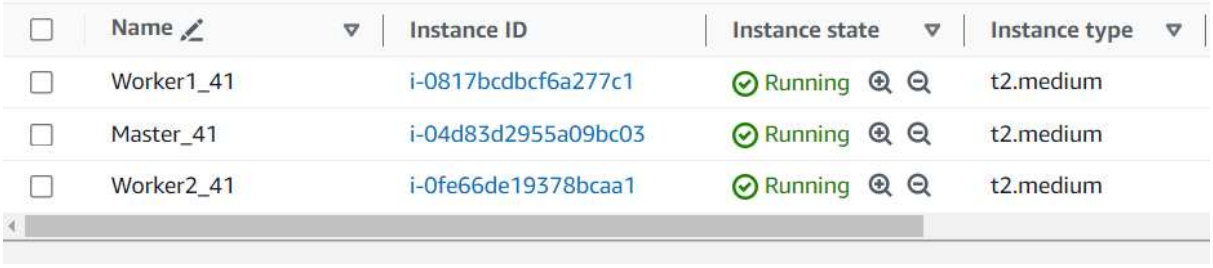
Created the Key pair login



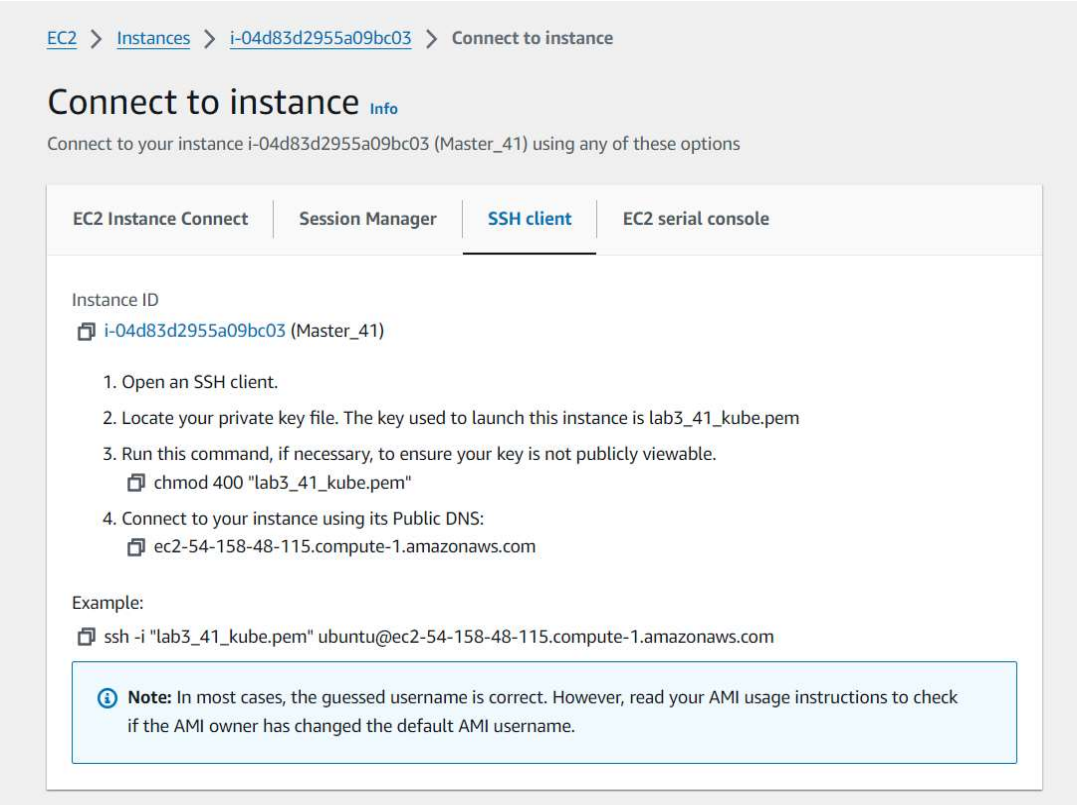
Select required Security Group

Instances are :

Step 2: After creating the instances click on Connect for all the instances one by one and navigate to the SSH Client section .

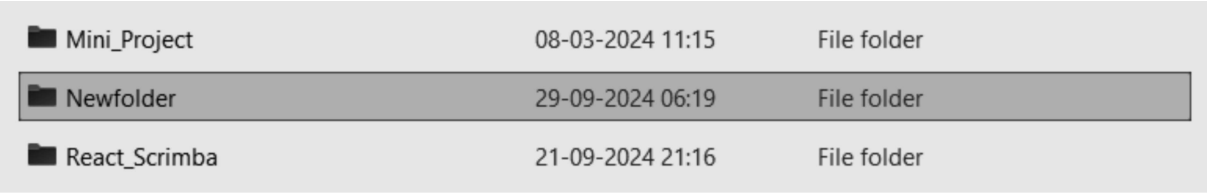


Master Node:



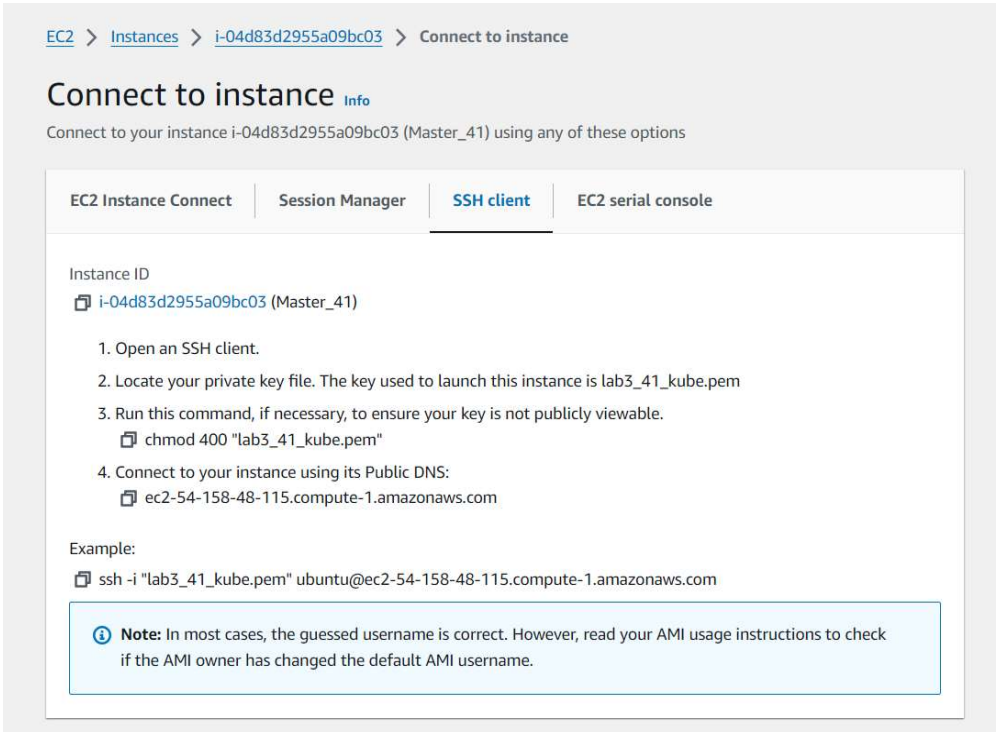
Step 3: Now open the folder in the terminal 3 times by right clicking on it for Master, Node1 & Node 2 where our .pem key is stored and paste the Example command (starting with ssh -i …..) from the ssh client section for instance in the terminal as shown above.

This will basically make our terminal to do remote login on our ec2 instance via SSH

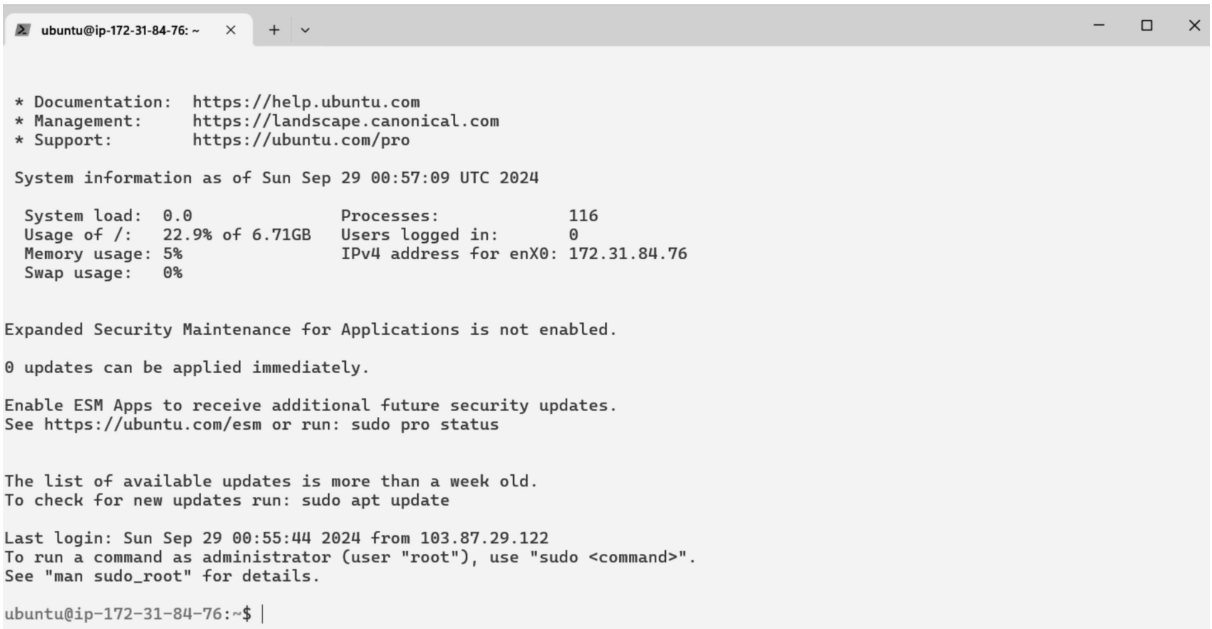


Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

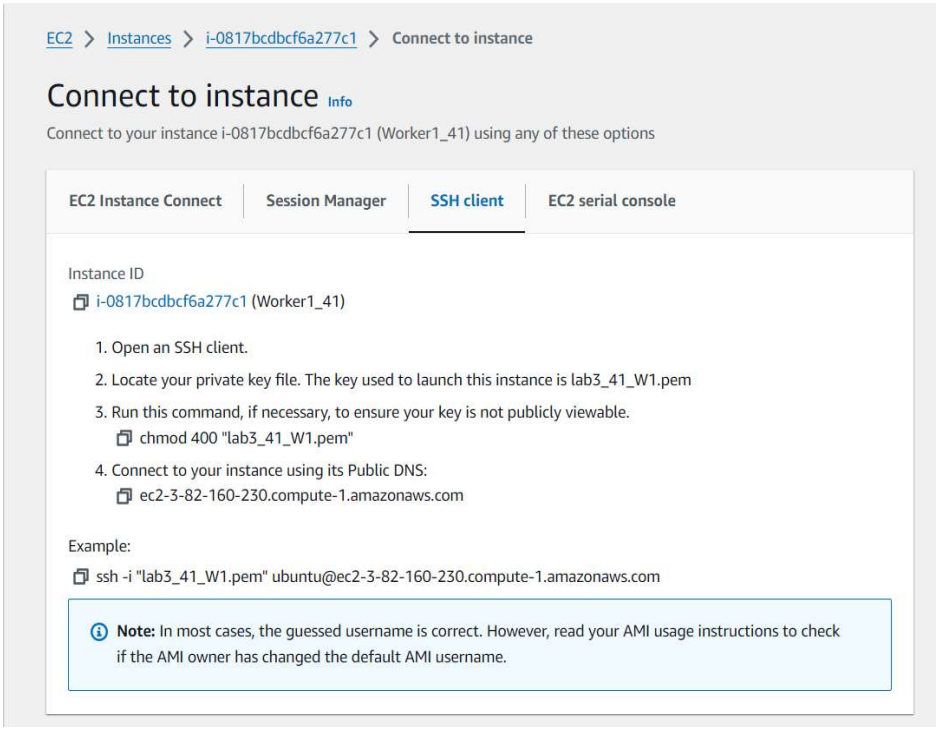
MasterNode:



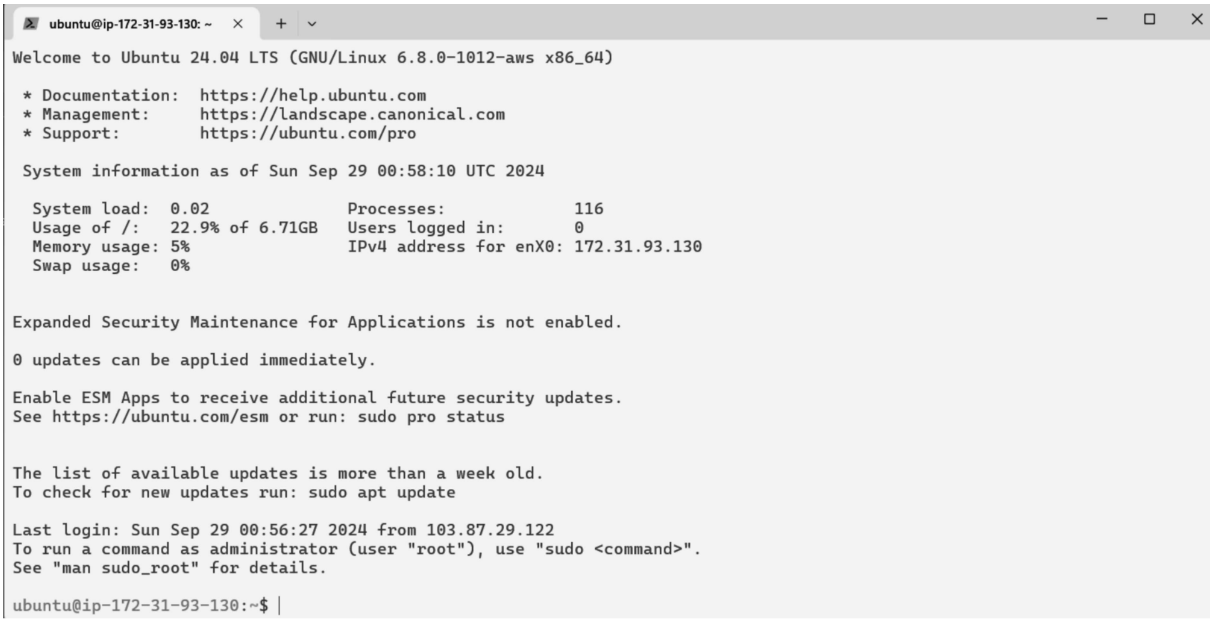
Successful Connection:



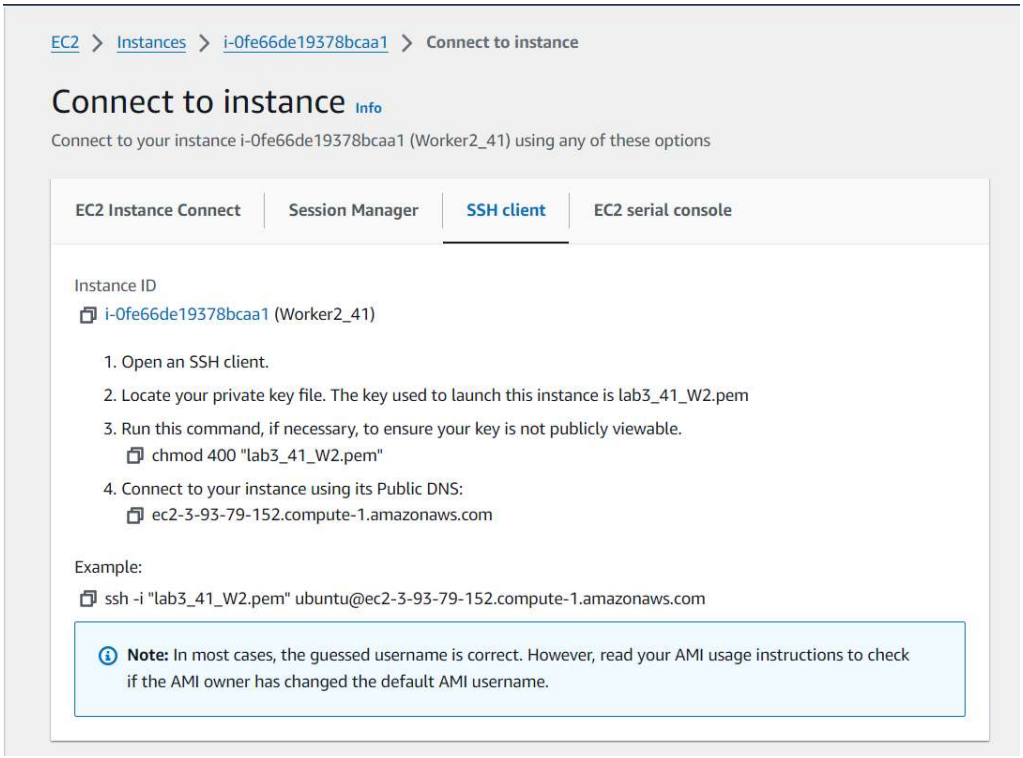
Worker Node 1:



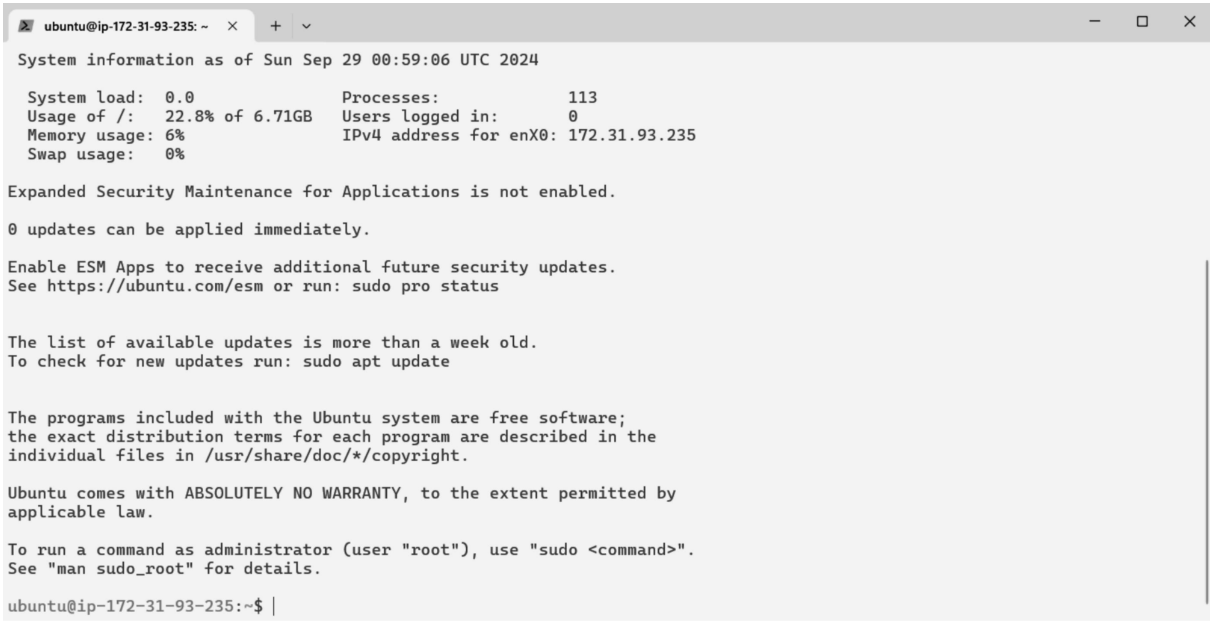
Successful Connection:



Worker Node 2:

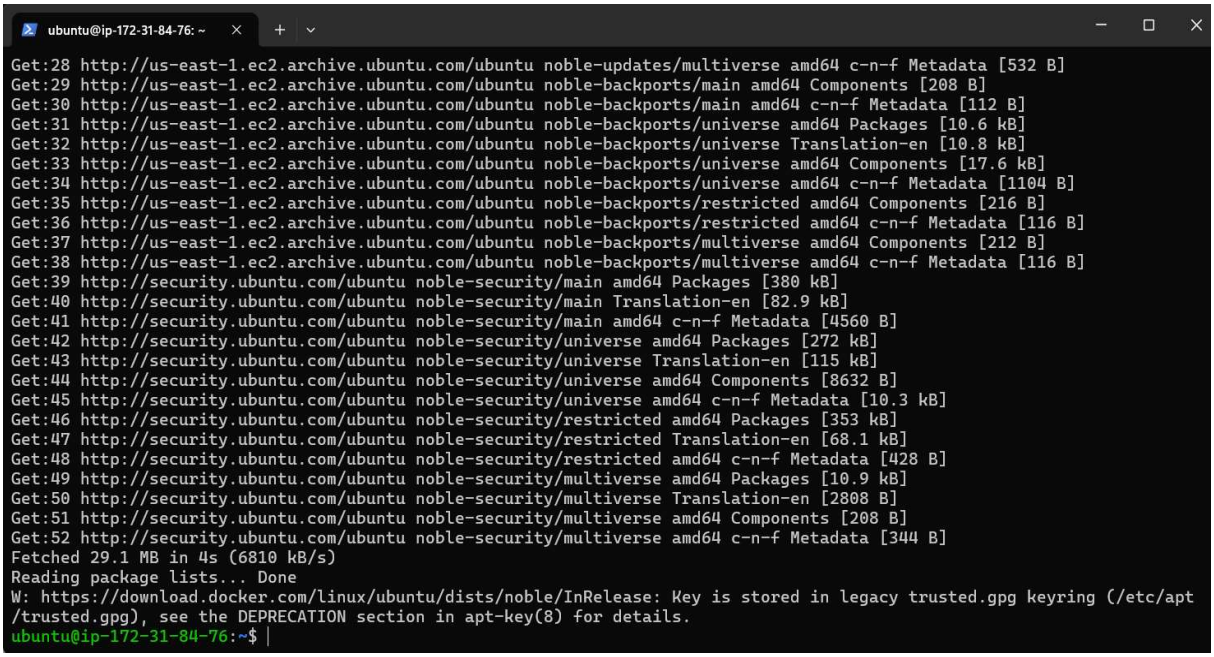


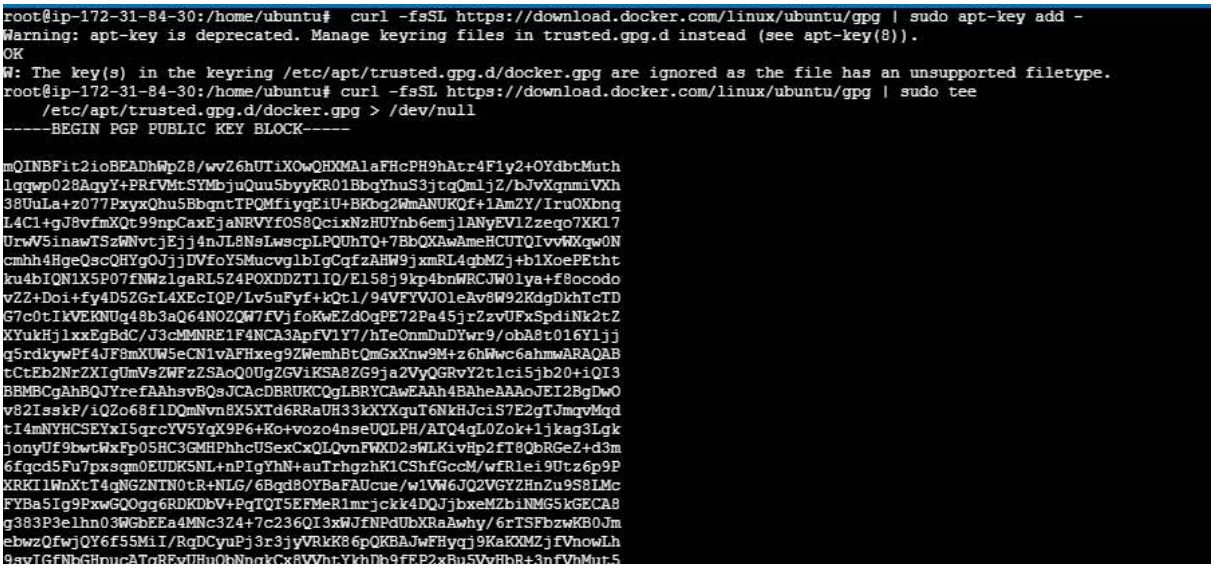
Successful Connection:



Step 4: Run on Master,Worker Node 1,and Worker Node 2 the below commands to install and setup Docker in Master,Worker Node 1,and Worker Node 2.

a) curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"





b) sudo apt-get update
sudo apt-get install -y docker-ce

```
root@ip-172-31-88-203:/home/ubuntu#  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 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.gpg.d/docker.gpg are ignc
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg),
details.
root@ip-172-31-88-203:/home/ubuntu#  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 sl
0 upgraded, 10 newly installed, 0 to remove and 143 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]
```

```
root@ip-172-31-84-30:/home/ubuntu#  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: The key(s) in the keyring /etc/apt/trusted.gpg.d/docker.gpg are
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gp
details.
root@ip-172-31-84-30:/home/ubuntu#    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 slirp4ne
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
```

i-04343d785c0fca4d4 (61_exp_3_worker)

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

```
root@ip-172-31-88-203:/home/ubuntu# mkdir -p /etc/docker
root@ip-172-31-88-203:/home/ubuntu#  cat <<EOF | sudo tee /etc/docker/daemon.json
> {
  "exec-opts": ["native.cgroupdriver=systemd"]
 }
> EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
 }
```

d) sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker

```
root@ip-172-31-88-203:/home/ubuntu# sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/sy
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
root@ip-172-31-88-203:/home/ubuntu#  sudo systemctl daemon-reload
root@ip-172-31-88-203:/home/ubuntu#    sudo systemctl restart docker
root@ip-172-31-88-203:/home/ubuntu# docker -v
Docker version 27.3.1, build ce12230
```

Step 5: Run the below command to **install Kubernetes** on all the three terminals one by one

a) sudo rm -f /etc/apt/sources.list.d/kubernetes.list

```
root@ip-172-31-88-203:/home/ubuntu# rm -f /etc/apt/sources.list.d/kubernetes.list
```

b) sudo apt-get update

```
root@ip-172-31-88-203:/home/ubuntu# 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 InRelea
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: The key(s
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is st
details.
```

```
i-05d1cf82d5660367e (61_exp_3_Master)
```

c) sudo apt-get install -y apt-transport-https ca-certificates curl gpg

```
root@ip-172-31-88-203:/home/ubuntu#  sudo apt-get install -y apt-transport-https ca-certificates curl gpg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
gpg is already the newest version (2.4.4-2ubuntu17).
gpg set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
The following packages will be upgraded:
  curl libcurl3t64-gnutls libcurl4t64
3 upgraded, 1 newly installed, 0 to remove and 140 not upgraded.
Need to get 904 kB of archives.
After this operation, 38.9 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 apt-transport-https all 2.7.14bu
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 curl amd64 8.5.0-2ubuntu10.4
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libcurl4t64 amd64 8.5.0-2ubu
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libcurl3t64-gnutls amd64 8.5
Fetched 904 kB in 0s (16.1 MB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 68007 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
```

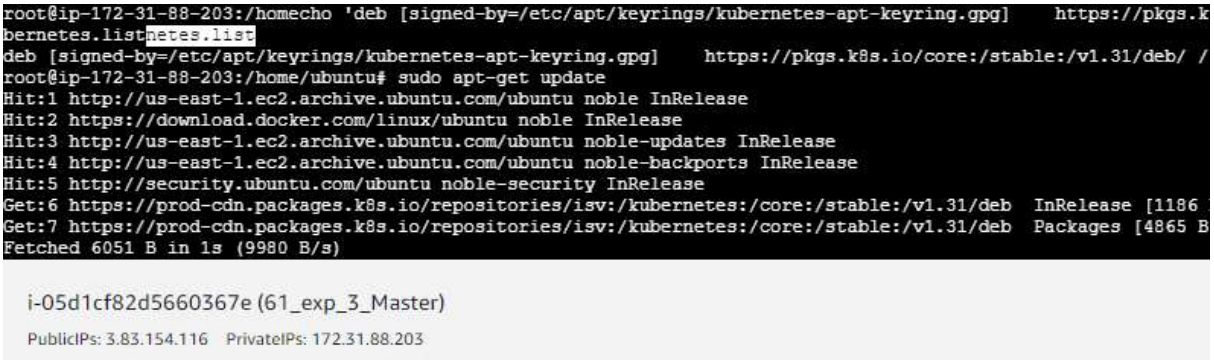
d) sudo mkdir -p -m 755 /etc/apt/keyrings

```
root@ip-172-31-88-203:/home/ubuntu#  mkdir -p -m 755 /etc/apt/keyrings
```

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

```
root@ip-172-31-88-203:/home/ubuntu# curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg  --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpgeyring
pg
```

g) 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



f) sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

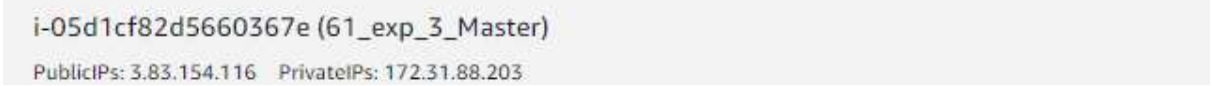
```
root@ip-172-31-88-203:/home/ubuntu# sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 https://download.docker.com/linux/ubuntu noble InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.
Fetched 6051 B in 1s (9980 B/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the k
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in le
details.
root@ip-172-31-88-203:/home/ubuntu#  sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
```

```
Running kernel seems to be up-to-date.

No services need to be restarted.

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.
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

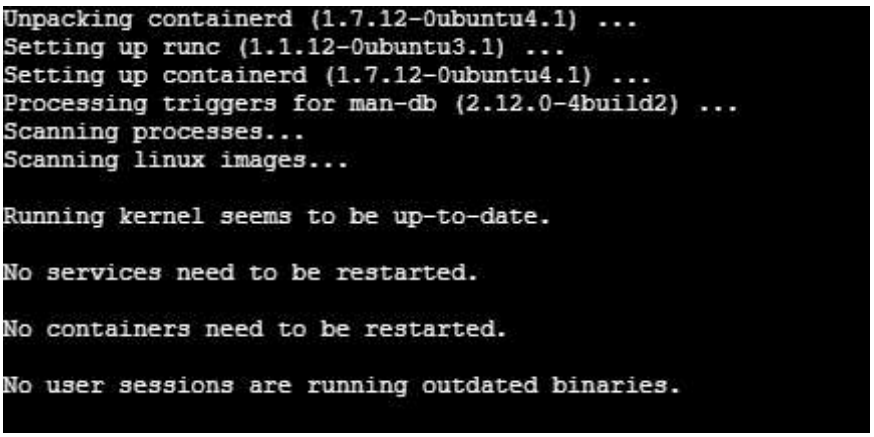
```
root@ip-172-31-88-203:/home/ubuntu#   sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

g) sudo systemctl enable --now kubelet

```
root@ip-172-31-88-203:/home/ubuntu#  systemctl enable --now kubelet
root@ip-172-31-88-203:/home/ubuntu#  sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libl
Use 'sudo apt autoremove' to remove them.
```



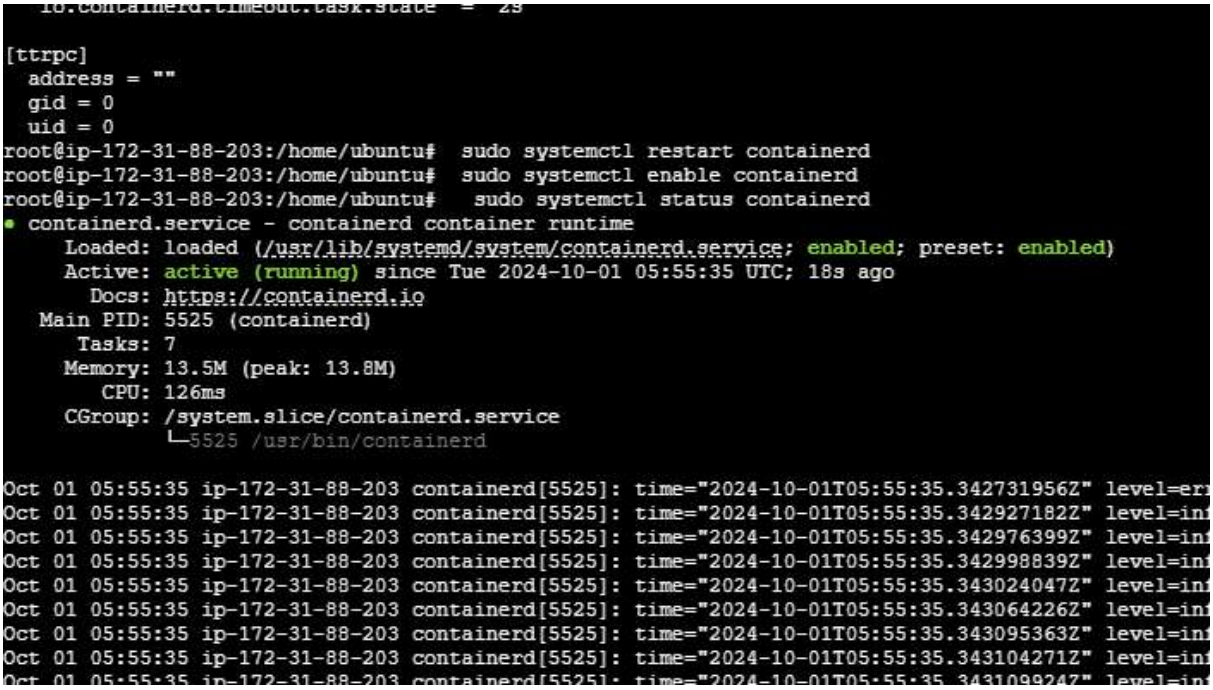
h) sudo apt-get install -y containerd



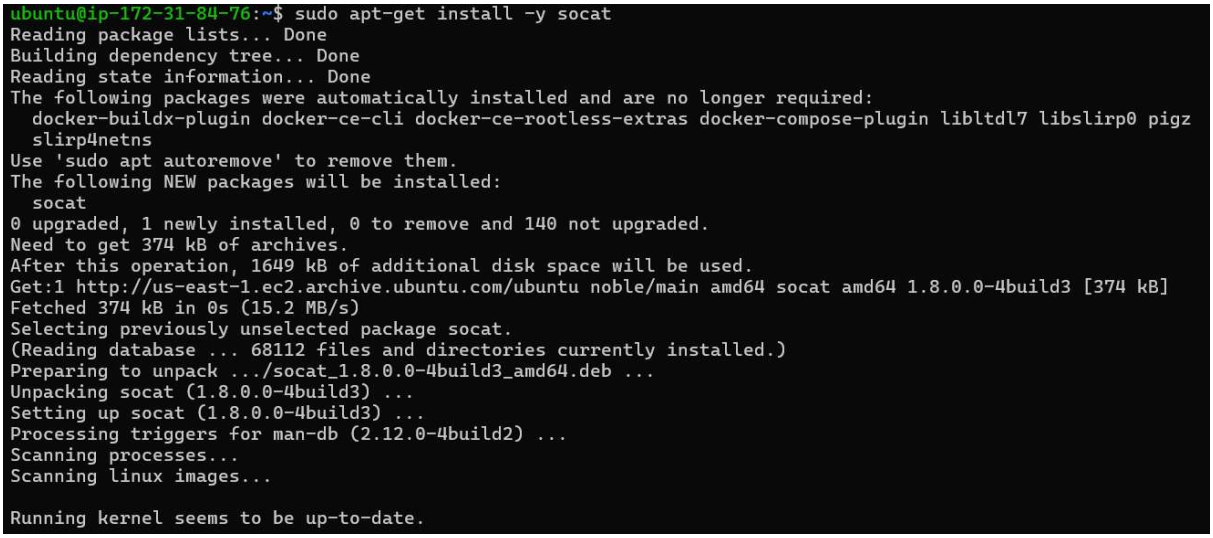
i) sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml

```
root@ip-172-31-88-203:/home/ubuntu#  sudo mkdir -p /etc/containerd
root@ip-172-31-88-203:/home/ubuntu# 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
```

j) sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd



k) sudo apt-get install -y socat



```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.
```

Step 6: Set up the Kubernetes cluster by running the following command exclusively on the master node:
sudo kubeadm init --pod-network-cidr=10.244.0.0/16. This command initializes the Kubernetes control plane and specifies the pod network range to be used within the cluster i.e kubeadm will initialize the kubernetes cluster and cidr is Classless Inter Domain Range which decides the range of network range of the pods

```
root@ip-172-31-88-203:/home/ubuntu# sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.31.0
[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 conn
[preflight] You can also perform this action beforehand using 'kubeadm config images pull
W1001 05:56:40.054582    5726 checks.go:846] detected that the sandbox image "registry.k8
ommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-88-203 kubernetes kuber
1 172.31.88.203]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
```

i-05d1cf82d5660367e (61_exp_3_Master)

PublicIPs: 3.83.154.116 PrivateIPs: 172.31.88.203

Run this command on master:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

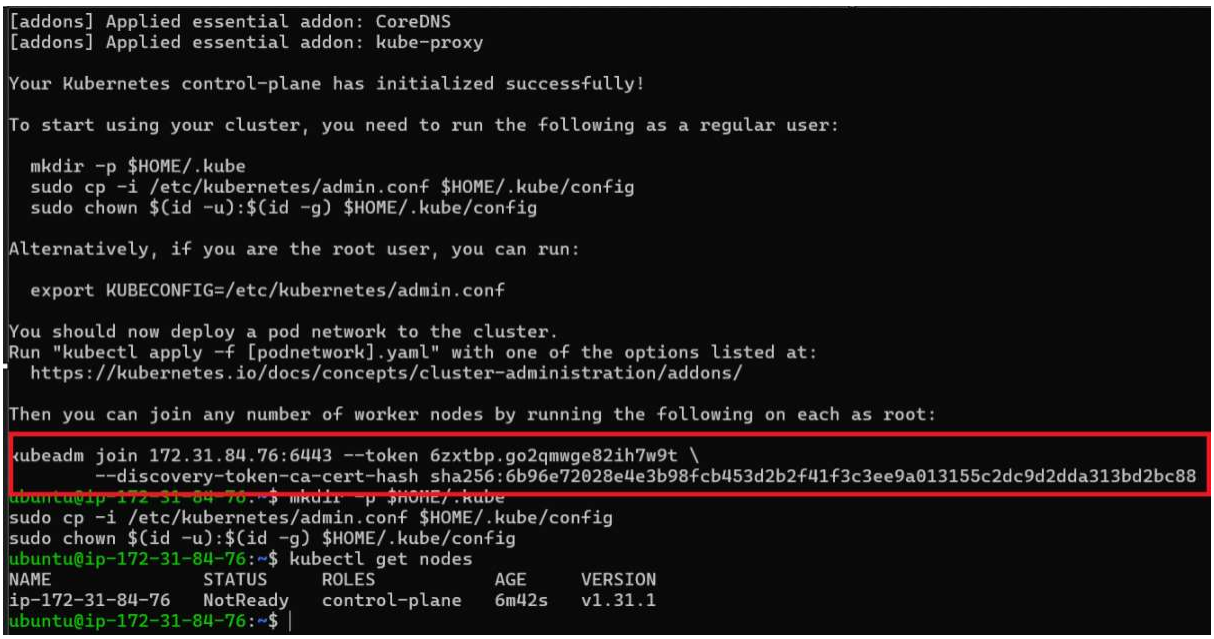
These commands create a .kube directory in your home folder, copy the Kubernetes admin configuration file into it, and change the ownership of that configuration file to the current user to ensure proper access.

```
To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.88.203:6443 --token ef5xtx.okbe4aouwj2i4fdf \
        --discovery-token-ca-cert-hash sha256:71074273c6192db966a142598ef7d65305691636883a1cf66233df241d4c7c76
root@ip-172-31-88-203:/home/ubuntu#  mkdir -p $HOME/.kube
root@ip-172-31-88-203:/home/ubuntu# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@ip-172-31-88-203:/home/ubuntu# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 7: Now Run the command **kubectl get nodes** to see the nodes before executing Join
command on nodes. kubectl is a command-line tool used to interact with and manage Kubernetes clusters. It allows users to deploy applications, inspect and manage cluster resources, and view logs, among other tasks, by sending commands to the Kubernetes API server.

```
root@ip-172-31-88-203:/home/ubuntu# kubectl get nodes
NAME               STATUS     ROLES           AGE   VERSION
ip-172-31-88-203   NotReady   control-plane   57s   v1.31.1
```

Step 8: Now Run the following command on Node 1 and Node 2 to Join to master. For this from the kubeadm init command output for the master node copy and paste this on both the node



Copy and paste: ***kubeadm join 172.31.88.203:6443 --token ef5xtx.okbe4aouwj2i4fdf ***

--discovery-token-ca-cert-hash sha256:71074273c6192db966a142598ef7d65305691636883a1cf66233df241d4c7c76

Worker Node 1:

```
root@ip-172-31-84-30:/home/ubuntu# kubeadm join 172.31.88.203:6443 --token ef5xtx.okbe4aouwj2i4fdf \
        --discovery-token-ca-cert-hash sha256:71074273c6192db966a142598ef7d65305691636883a1cf66233df241d4c7c76
[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 1.001611728s
[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.

root@ip-172-31-84-30:/home/ubuntu#
```

i-04343d785c0fca4d4 (61_exp_3_worker)

Step 9: Run the command **kubectl get nodes** to see the nodes after executing Join command on nodes.

```
root@ip-172-31-88-203:/home/ubuntu# kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTE
ip-172-31-84-30   Ready    Worker_Node1_61   2m43s   v1.31.1   172.3
ip-172-31-88-203  Ready    control-plane     13m     v1.31.1   172.3
```

Step 10: 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 .This command applies the Calico network plugin configuration, which enables networking capabilities in the Kubernetes cluster .It enhances the effective communication between pods in network

```
root@ip-172-31-88-203:/home/ubuntu# 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
```

i-05d1cf82d5660367e (61_exp_3_Master)
PublicIPs: 3.83.154.116 PrivateIPs: 172.31.88.203

Now perform **sudo systemctl status kubelet**

```
root@ip-172-31-88-203:/home/ubuntu# 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 Tue 2024-10-01 05:56:58 UTC; 11min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 6396 (kubelet)
      Tasks: 10 (limit: 4676)
     Memory: 32.5M (peak: 33.0M)
```

i-05d1cf82d5660367e (61_exp_3_Master)
PublicIPs: 3.83.154.116 PrivateIPs: 172.31.88.203

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

```
ubuntu@ip-172-31-84-76:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-84-76   Ready    control-plane   39m     v1.31.1   172.31.84.76    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-93-130  Ready    <none>          12m     v1.31.1   172.31.93.130   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-93-235  Ready    <none>          8m45s   v1.31.1   172.31.93.235   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-84-76:~$
```

Now to Rename run this command
Rename to Worker_Node1_61: kubectl label node ip-172-31-84-30 kubernetes.io/role=Worker_Node_61

Make Sure to give Correct corresponding IP while renaming which can be seen from previous output also.

```
root@ip-172-31-88-203:/home/ubuntu# kubectl get nodes -o wide
NAME              STATUS   ROLES                          AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
CONTAINER-RUNTIME
ip-172-31-84-30   Ready    Worker_Node1_61                3m47s   v1.31.1   172.31.84.30    <none>        Ubuntu 24.04 LTS
containerd://1.7.12
ip-172-31-88-203  Ready    Master_Node_61,control-plane   14m     v1.31.1   172.31.88.203   <none>        Ubuntu 24.04 LTS
containerd://1.7.12
root@ip-172-31-88-203:/home/ubuntu#
```

i-05d1cf82d5660367e (61_exp_3_Master)
PublicIPs: 3.83.154.116 PrivateIPs: 172.31.88.203

Step 11: Run command **kubectl get nodes -o wide** . And Hence we can see we have Successfully connected both the Worker Nodes to the Master Node.


```
root@ip-172-31-88-203:/home/ubuntu# kubectl get nodes -o wide
NAME              STATUS   ROLES                          AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
CONTAINER-RUNTIME
ip-172-31-84-30   Ready    Worker_Node1_61                3m47s   v1.31.1   172.31.84.30    <none>        Ubuntu 24.04 LTS
containerd://1.7.12
ip-172-31-88-203  Ready    Master_Node_61,control-plane   14m     v1.31.1   172.31.88.203   <none>        Ubuntu 24.04 LTS
containerd://1.7.12
root@ip-172-31-88-203:/home/ubuntu#
```

i-05d1cf82d5660367e (61_exp_3_Master)
PublicIPs: 3.83.154.116 PrivateIPs: 172.31.88.203

CONCLUSION:

In this experiment, we established a Kubernetes cluster on AWS EC2 instances, consisting of one master node and two worker nodes. After installing Docker and the necessary Kubernetes tools (kubelet, kubeadm, kubectl, and containerd) on all nodes, we initialized the master and added the worker nodes to the cluster. Initially, the nodes showed a NotReady status, which we resolved by installing the Calico network plugin. We also tagged the nodes with their respective roles (control-plane and worker). Ultimately, all nodes transitioned to the Ready state, demonstrating our successful setup and management of the Kubernetes cluster.