Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

Theory:

Container-based microservices architectures have revolutionized how development and operations teams test and deploy modern software. Containers allow companies to scale and deploy applications more efficiently, but they also introduce new challenges, adding complexity by creating a whole new infrastructure ecosystem.

Today, both large and small software companies are deploying thousands of container instances daily. Managing this level of complexity at scale requires advanced tools. Enter Kubernetes.

Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. Kubernetes has quickly become the de facto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation (CNCF), supported by major players like Google, AWS, Microsoft, IBM, Intel, Cisco, and Red Hat.

Kubernetes simplifies the deployment and operation of applications in a microservice architecture by providing an abstraction layer over a group of hosts. This allows development teams to deploy their applications while Kubernetes takes care of key tasks, including:

- Managing resource consumption by applications or teams
- Distributing application load evenly across the infrastructure
- Automatically load balancing requests across multiple instances of an application
- Monitoring resource usage to prevent applications from exceeding resource limits and automatically restarting them if needed
- Moving application instances between hosts when resources are low or if a host fails
- Automatically utilizing additional resources when new hosts are added to the cluster
- Facilitating canary deployments and rollbacks with ease

Necessary Requirements:

● **EC2 Instance:** The experiment required launching a t2.medium EC2 instance with 2 CPUs, as Kubernetes demands sufficient resources for effective functioning.

● **Minimum Requirements:**

  ○ **Instance Type:** t2.medium
  ○ **CPUs:** 2
  ○ **Memory:** Adequate for container orchestration.

This ensured that the Kubernetes cluster had the necessary resources to function smoothly

Note:

AWS Personal Account is preferred but we can also perform it on AWS Academy(adding some ignores in the command if any error occurs in below as the below experiment is performed on Personal Account .).If You are using AWS Academy Account Errors you will face in kubeadm init command so you have to add some ignores with this command.

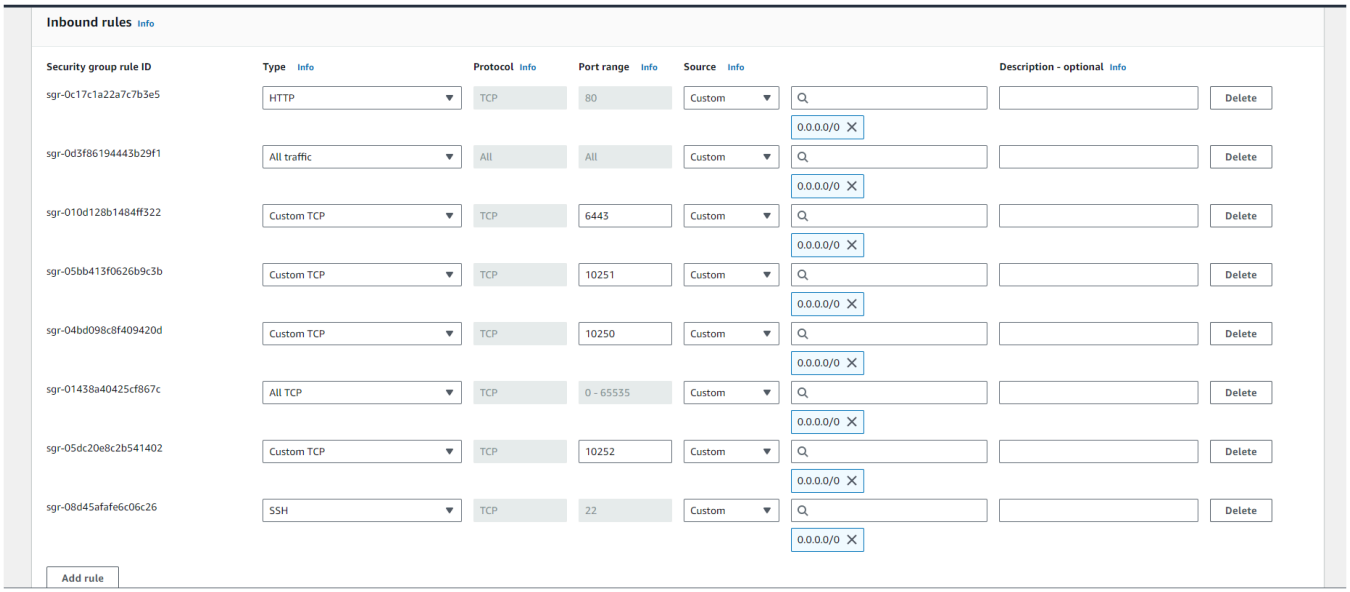
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

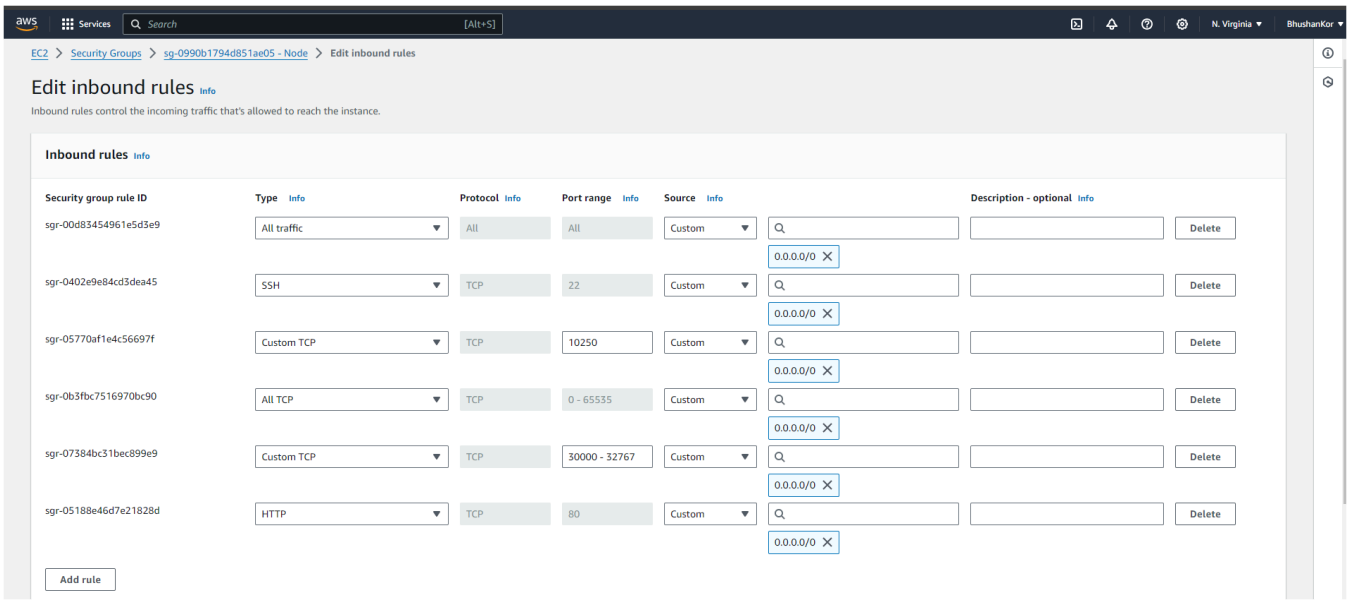
Prerequisites :

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

Master:



Node :

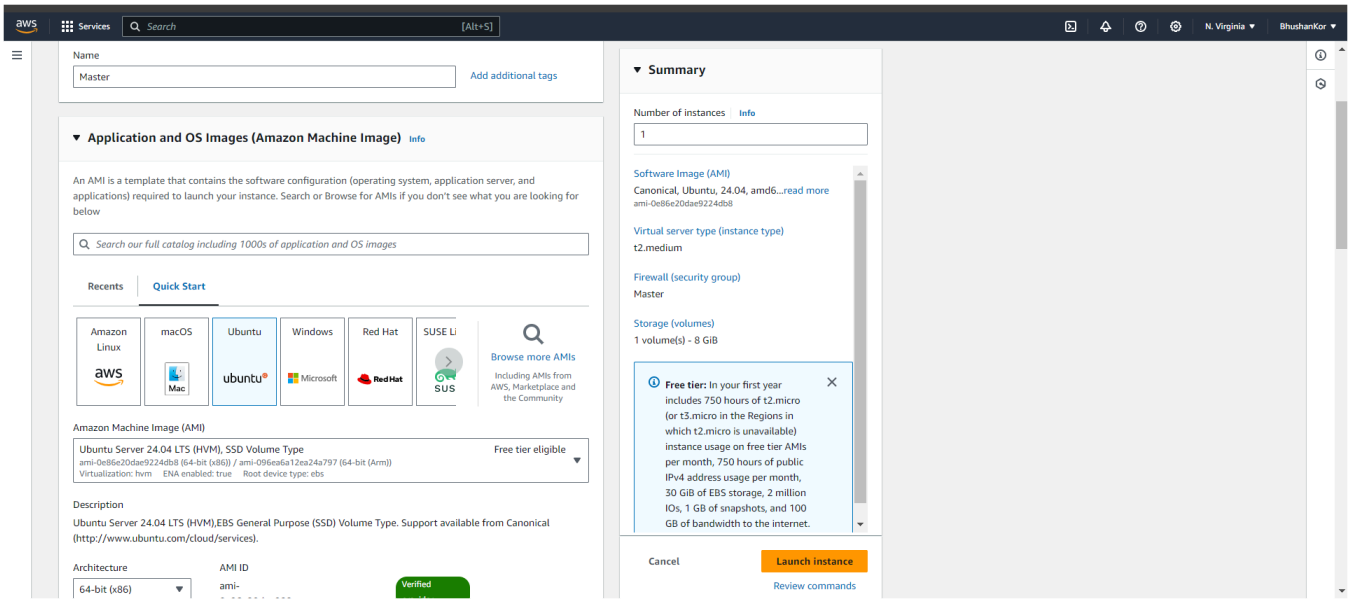


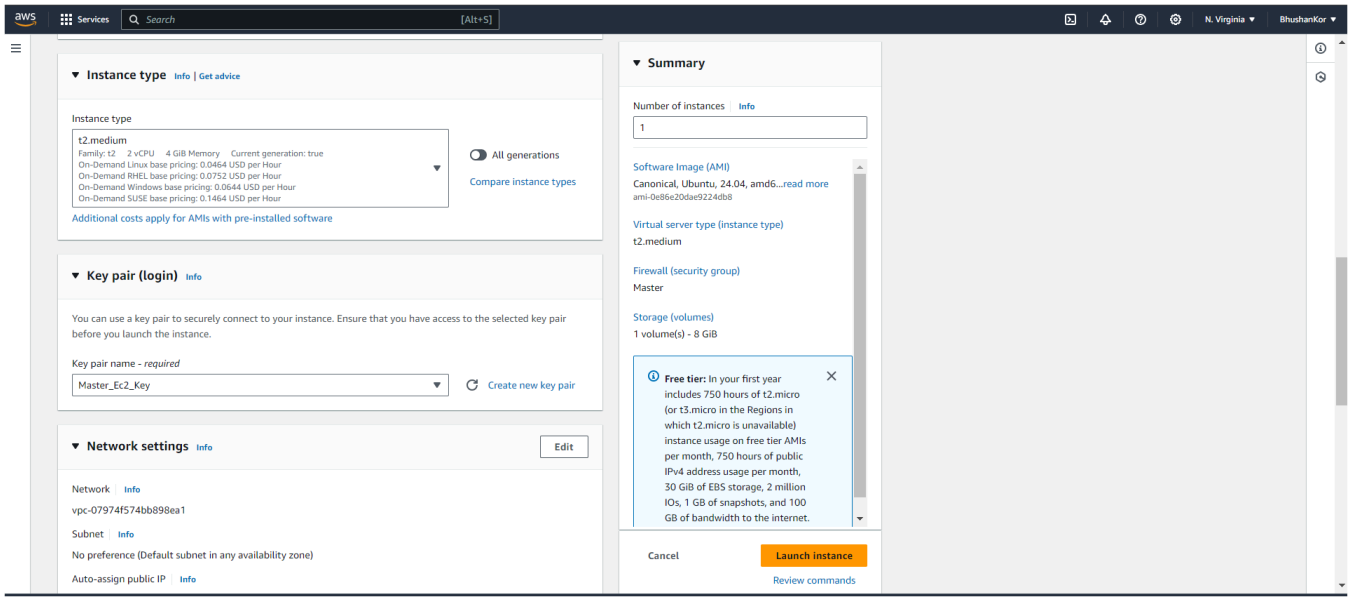
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 1: Log in to your AWS Academy/personal account and launch 3 new Ec2 Instances.
Select Ubuntu as AMI and t2.medium as Instance Type and create a key of type RSA with .pem extension and move the downloaded key to the new folder.We can use 3 Different keys or 1 common key also.
Note: A minimum of 2 CPUs are required so Please select t2.medium and do not forget to stop the instance after the experiment because it is not available in the free tier.
Also Select Security groups from existing.

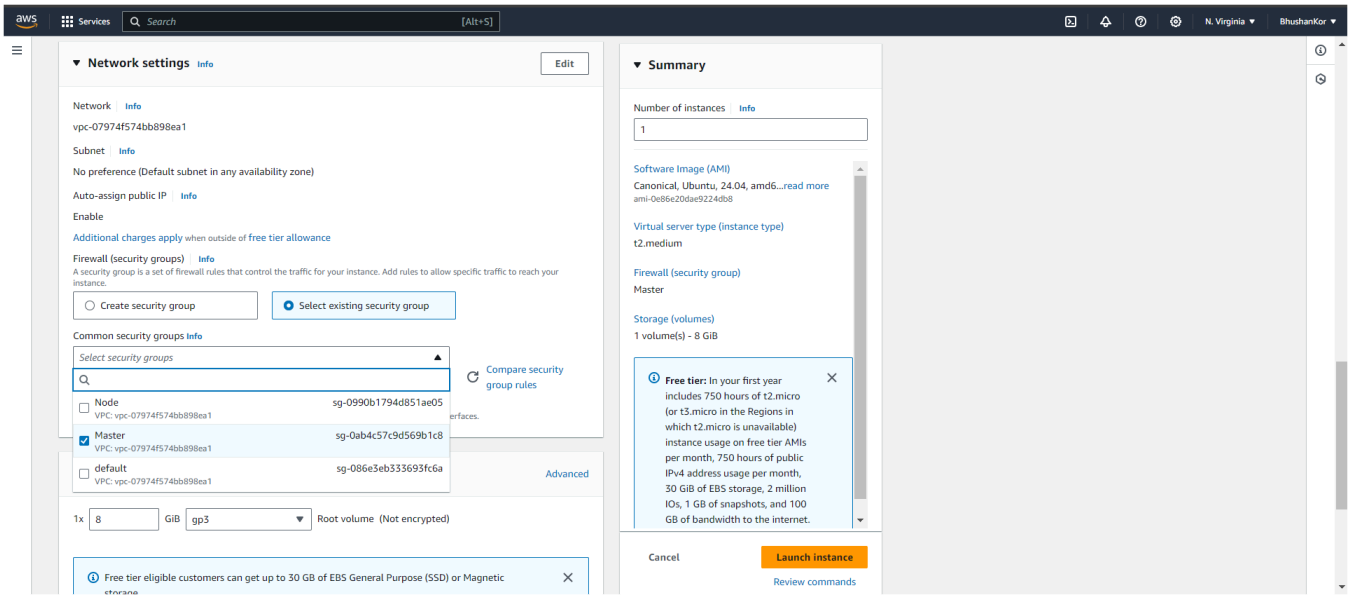
Master:





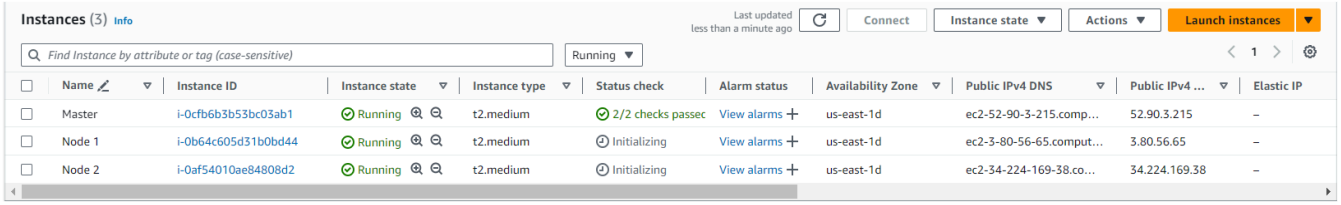
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

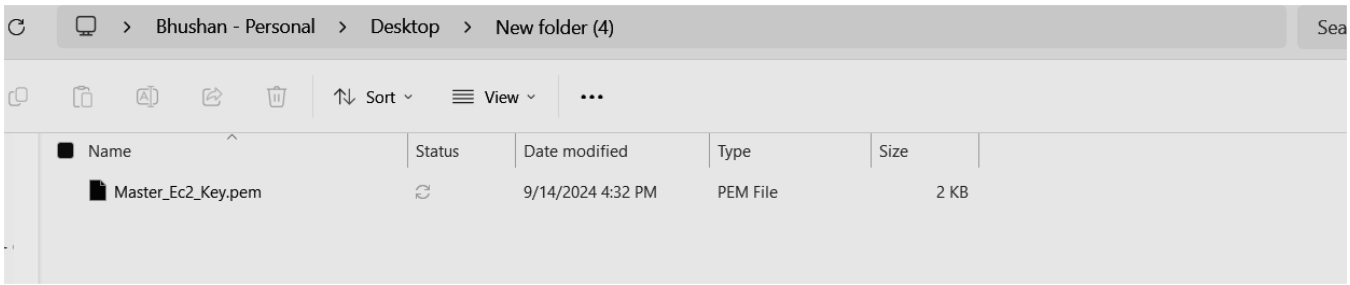


Do Same for 2 Nodes and use security groups of Node for that.

Step 2: After creating the instances click on Connect & connect all 3 instances and navigate to SSH Client.



(Downloded Key

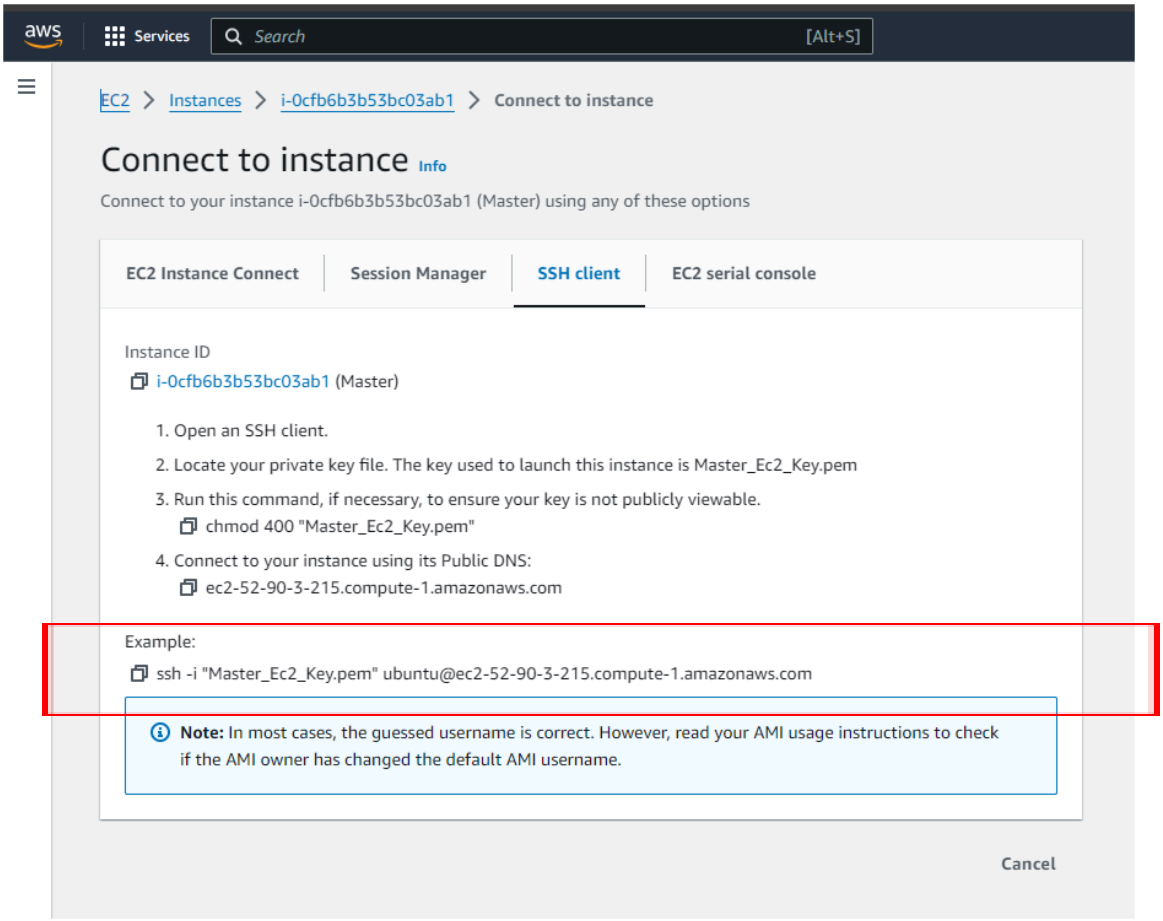


)

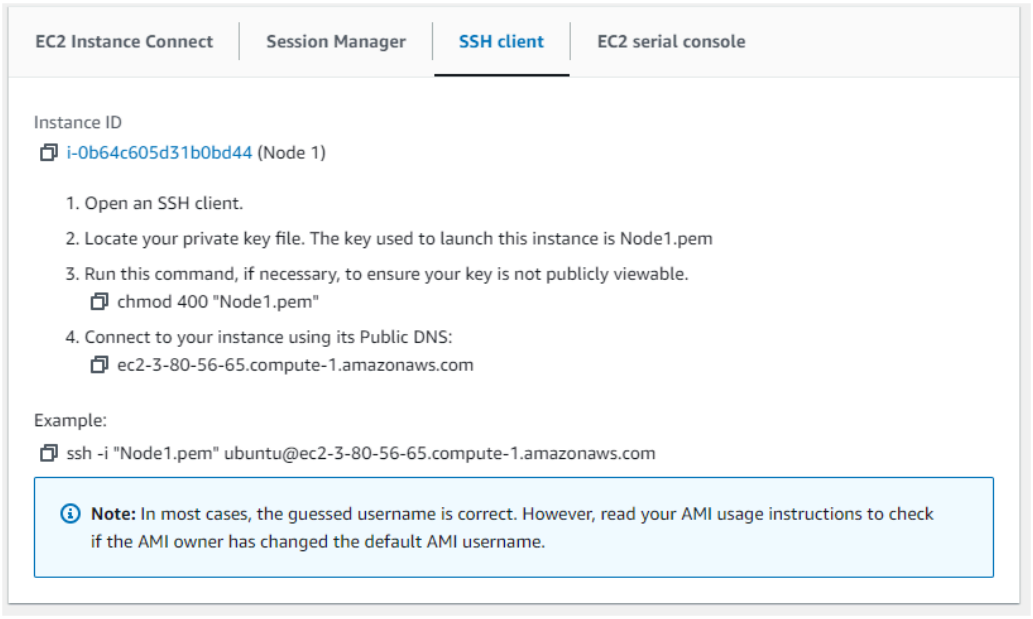
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 3: Now open the folder in the terminal 3 times for Master, Node1& Node 2 where our .pem key is stored and paste the Example command (starting with ssh -i …..) in the terminal.(ssh -i "Master_Ec2_Key.pem" ubuntu@ec2-54-196-129-215.compute-1.amazonaws.com)
Master:



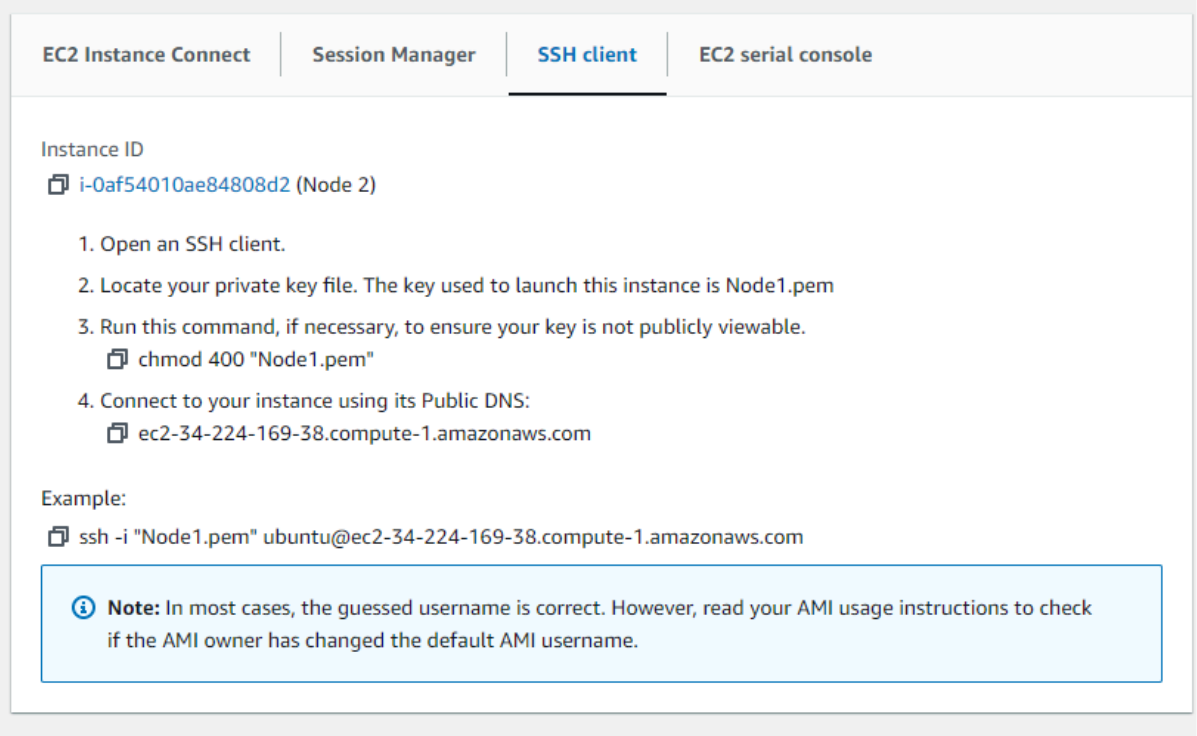
Node 1:



Name:Bhushan Mukund Kor **Academic Year:2024-2025**

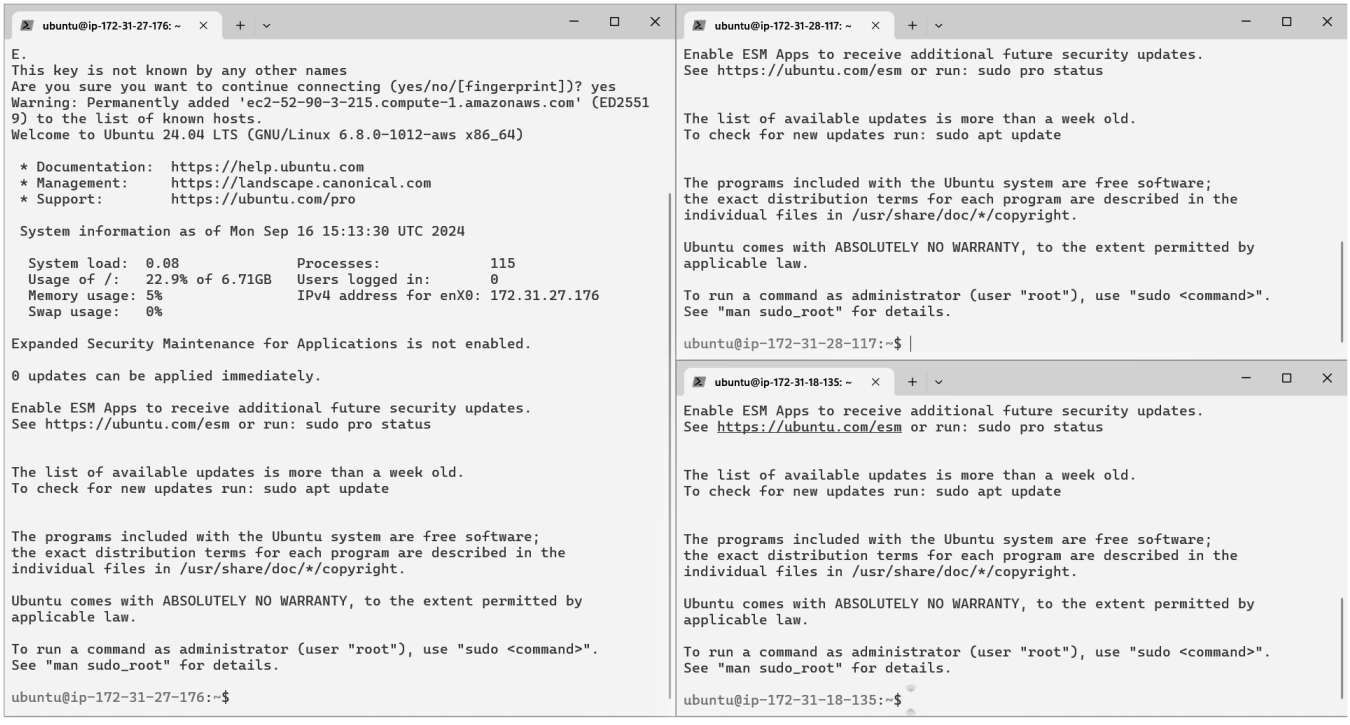
Division: D15C **Roll No: 28**

Node 2:



Here I have use 2 keys 1 for master and 1 for 2 node so I have to run open 3 terminals.In master key folder 1 terminal and 2 terminals in node 1 key folder.
If you use 1 Key only, you can open 3 terminal in one folder only.

Successful Connection:



Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 4: Run on Master,Node 1,and Node 2 the below commands to install and setup Docker in Master, Node1, and Node2.

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"

```
ubuntu@ip-172-31-27-176:~$
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"
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instea
d (see apt-key(8)).
OK
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_docke
r_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_d
ownload_docker_com_linux_ubuntu-noble.list
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
 [126 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelea
se [126 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Pa
ckages [15.0 MB]
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [
13.8 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [
354 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [
79.4 kB]
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
cted amd64 c-n-f Metadata [116 B]
Get:51 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiv
erse amd64 Components [212 B]
Get:52 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiv
erse amd64 c-n-f Metadata [116 B]
Fetched 28.9 MB in 4s (6976 kB/s)
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 ignored as the file h
as an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is st
ored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATI
ON section in apt-key(8) for details.
ubuntu@ip-172-31-27-176:~$
```

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

```
ubuntu@ip-172-31-27-176:~$ sudo apt-get update
sudo apt-get install -y docker-ce
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
se
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 ignored as the file h
as an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is st
ored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATI
ON section in apt-key(8) for details.
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 133 not upgraded.
Need to get 122 MB of archives.
After this operation, 440 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pi
gz amd64 2.8-1 [65.6 kB]
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
 /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /us
r/lib/systemd/system/docker.socket.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.2) ...
Scanning processes...
Scanning linux images...

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.
ubuntu@ip-172-31-27-176:~$
```

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

```
ubuntu@ip-172-31-27-176:~$ sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
    "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
    "exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-27-176:~$
```

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

```
ubuntu@ip-172-31-27-176:~$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart 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-172-31-27-176:~$
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 5: Run the below command to install Kubernets.
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

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

```
ubuntu@ip-172-31-27-176:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.3
1/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-k
eyring.gpg

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

deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8
s.io/core:/stable:/v1.31/deb/ /
ubuntu@ip-172-31-27-176:~$
```

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

```
ubuntu@ip-172-31-27-176:~$ sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
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
se
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/s
table:/v1.31/deb  InRelease [1186 B]
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/s
table:/v1.31/deb  Packages [4865 B]
Fetched 6051 B in 1s (11.1 kB/s)
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 ignored as the file h
as an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is st
ored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATI
ON section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
Scanning processes...
Scanning linux images...

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.
ubuntu@ip-172-31-27-176:~$
```

sudo systemctl enable --now kubelet
sudo apt-get install -y containerd

```
ubuntu@ip-172-31-27-176:~$ sudo systemctl enable --now kubelet
ubuntu@ip-172-31-27-176:~$ 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 requir
ed:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  runc
The following packages will be REMOVED:
  containerd.io docker-ce
The following NEW packages will be installed:
  containerd runc
0 upgraded, 2 newly installed, 2 to remove and 133 not upgraded.
Need to get 47.2 MB of archives.
After this operation, 53.1 MB disk space will be freed.
```

```
Scanning linux images...

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.
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

```
ubuntu@ip-172-31-27-176:~$ sudo mkdir -p /etc/containerd
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

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
```

```
[timeouts]
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
```

sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd

```
ubuntu@ip-172-31-27-176:~$ sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; p>
     Active: active (running) since Mon 2024-09-16 15:31:58 UTC; 210ms ago
       Docs: https://containerd.io
   Main PID: 4763 (containerd)
      Tasks: 7
     Memory: 13.9M (peak: 14.4M)
        CPU: 50ms
     CGroup: /system.slice/containerd.service
             └─4763 /usr/bin/containerd

Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 containerd[4763]: time="2024-09-16T15:31:5>
Sep 16 15:31:58 ip-172-31-27-176 systemd[1]: Started containerd.service - c>
```

sudo apt-get install -y socat

```
ubuntu@ip-172-31-27-176:~$  sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer requir
ed:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 133 not upgraded.
Need to get 374 kB of archives.
After this operation, 1649 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 socat
amd64 1.8.0.0-4build3 [374 kB]
Fetched 374 kB in 0s (11.2 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68108 files and directories currently installed.)
Preparing to unpack .../socat_1.8.0.0-4build3_amd64.deb ...
Unpacking socat (1.8.0.0-4build3) ...
Setting up socat (1.8.0.0-4build3) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

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.
ubuntu@ip-172-31-27-176:~$ |
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 6: Initialize the Kubecluster .Now Perform this Command only for Master.
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```
ubuntu@ip-172-31-27-176:~$ 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 connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0916 15:39:33.685919    5313 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 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-172-31-27-176 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster
.local] and IPs [10.96.0.1 172.31.27.176]
[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
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-27-176 localhost] and IPs [172.31.27.176 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-27-176 localhost] and IPs [172.31.27.176 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests"
[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 1.001059178s
```

```
[api-check] Waiting for a healthy API server. This can take up to 4m0s
[api-check] The API server is healthy after 6.500965245s
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node ip-172-31-27-176 as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/excl
ude-from-external-load-balancers]
[mark-control-plane] Marking the node ip-172-31-27-176 as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: ttay2x.n0sqeukjai8sgfg3
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

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 172.31.27.176:6443 --token ttay2x.n0sqeukjai8sgfg3 \
        --discovery-token-ca-cert-hash sha256:d6fc5fb7e984c83e2807780047fec6c4f2acfe9da9184ecc028d77157608fbb6
ubuntu@ip-172-31-27-176:~$
```

Run this command on master and also copy and save the Join command from above.

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

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
ubuntu@ip-172-31-27-176:~$ mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-27-176:~$
```

Step 7: Now Run the command kubectl get nodes to see the nodes before executing Join command on nodes.

```
Every 2.0s: kubectl get nodes                                                    ip-172-31-27-176: Mon Sep 16 15:45:34 2024

NAME              STATUS     ROLES           AGE     VERSION
ip-172-31-27-176   NotReady   control-plane   5m38s   v1.31.1
```

Step 8: Now Run the following command on Node 1 and Node 2 to Join to master.

**sudo kubeadm join 172.31.27.176:6443 --token ttay2x.n0sqeukjai8sgfg3 **
--discovery-token-ca-cert-hash
sha256:d6fc5fb7e984c83e2807780047fec6c4f2acfe9da9184ecc028d77157608fbb6

Node 1:

```
ubuntu@ip-172-31-28-117:~$ sudo kubeadm join 172.31.27.176:6443 --token ttay2x.n0sqeukjai8sgfg3 \
        --discovery-token-ca-cert-hash sha256:d6fc5fb7e984c83e2807780047fec6c4f2acfe9da9184ecc028d77157608fbb6

[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 501.396793ms
[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-172-31-28-117:~$
```

Node 2:

```
ubuntu@ip-172-31-18-135:~$ sudo kubeadm join 172.31.27.176:6443 --token ttay2x.n0sqeukjai8sgfg3 \
        --discovery-token-ca-cert-hash sha256:d6fc5fb7e984c83e2807780047fec6c4f2acfe9da9184ecc028d77157608fbb6
[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 1.001003808s
[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-172-31-18-135:~$
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

```
ubuntu@ip-172-31-27-176:~$ kubectl get nodes
NAME                STATUS     ROLES           AGE     VERSION
ip-172-31-18-135    NotReady   <none>          88s     v1.31.1
ip-172-31-27-176    NotReady   control-plane   10m     v1.31.1
ip-172-31-28-117    NotReady   <none>          2m58s   v1.31.1
```

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

```
ubuntu@ip-172-31-27-176:~$ 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-172-31-27-176:~$ 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 Mon 2024-09-16 15:40:01 UTC; 11min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 5989 (kubelet)
      Tasks: 10 (limit: 4676)
     Memory: 32.6M (peak: 33.2M)
        CPU: 10.705s
     CGroup: /system.slice/kubelet.service
             └─5989 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/>

Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497458    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497516    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497569    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497620    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497669    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:29 ip-172-31-27-176 kubelet[5989]: I0916 15:51:29.497719    5989 reconciler_common.go:245] "operationExecutor.VerifyControllerAttachedVolume s>
Sep 16 15:51:31 ip-172-31-27-176 kubelet[5989]: E0916 15:51:31.605091    5989 kubelet.go:2902] "Container runtime network not ready" networkReady="NetworkR>
Sep 16 15:51:32 ip-172-31-27-176 kubelet[5989]: I0916 15:51:32.366237    5989 scope.go:117] "RemoveContainer" containerID="f44f06967c5b3e567e07841a7b4352ae>
Sep 16 15:51:36 ip-172-31-27-176 kubelet[5989]: E0916 15:51:36.606675    5989 kubelet.go:2902] "Container runtime network not ready" networkReady="NetworkR>
Sep 16 15:51:41 ip-172-31-27-176 kubelet[5989]: E0916 15:51:41.608404    5989 kubelet.go:2902] "Container runtime network not ready" networkReady="NetworkR>
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

```
ubuntu@ip-172-31-27-176:~$ ubuntu@ip-172-31-27-176:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-18-135  Ready    <none>          6m19s   v1.31.1   172.31.18.135   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-27-176  Ready    control-plane   15m     v1.31.1   172.31.27.176   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-28-117  Ready    <none>          7m49s   v1.31.1   172.31.28.117   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
```

Now to Rename run this command

kubectl label node ip-172-31-18-135 kubernetes.io/role=worker

Rename to Node 1:kubectl label node ip-172-31-28-117 kubernetes.io/role=Node1
Rename to Node 2:kubectl label node ip-172-31-18-135 kubernetes.io/role=Node2

```
ubuntu@ip-172-31-27-176:~$ kubectl label node ip-172-31-28-117 kubernetes.io/role=Node1
node/ip-172-31-28-117 labeled
ubuntu@ip-172-31-27-176:~$ kubectl label node ip-172-31-18-135 kubernetes.io/role=Node2
node/ip-172-31-18-135 labeled
```

Step 11: Run command kubectl get nodes -o wide . And Hence we can see we have Successfully connected Node 1 and Node 2 to the Master.

```
ubuntu@ip-172-31-27-176:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-18-135  Ready    Node2           12m   v1.31.1   172.31.18.135   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-27-176  Ready    control-plane   21m   v1.31.1   172.31.27.176   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-28-117  Ready    Node1           13m   v1.31.1   172.31.28.117   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-27-176:~$
```

Or run kubectl get nodes

```
ubuntu@ip-172-31-27-176:~$ kubectl get nodes
NAME              STATUS   ROLES           AGE   VERSION
ip-172-31-18-135  Ready    Node2           24m   v1.31.1
ip-172-31-27-176  Ready    control-plane   33m   v1.31.1
ip-172-31-28-117  Ready    Node1           25m   v1.31.1
ubuntu@ip-172-31-27-176:~$
```

Conclusion: In this experiment, we successfully set up a Kubernetes cluster with one master and two worker nodes on AWS EC2 instances. After installing Docker, Kubernetes tools (kubelet, kubeadm, kubectl), and containerd on all nodes, the master node was initialized and the worker nodes were joined to the cluster. Initially, the nodes were in the `NotReady` state, which was resolved by installing the Calico network plugin. We also labeled the nodes with appropriate roles (control-plane and worker). The cluster became fully functional with all nodes in the `Ready` state, demonstrating the successful configuration and orchestration of Kubernetes.