Name:Krushikesh Shelar 2024-2025 D15C Roll No:51

Experiment 3:

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.

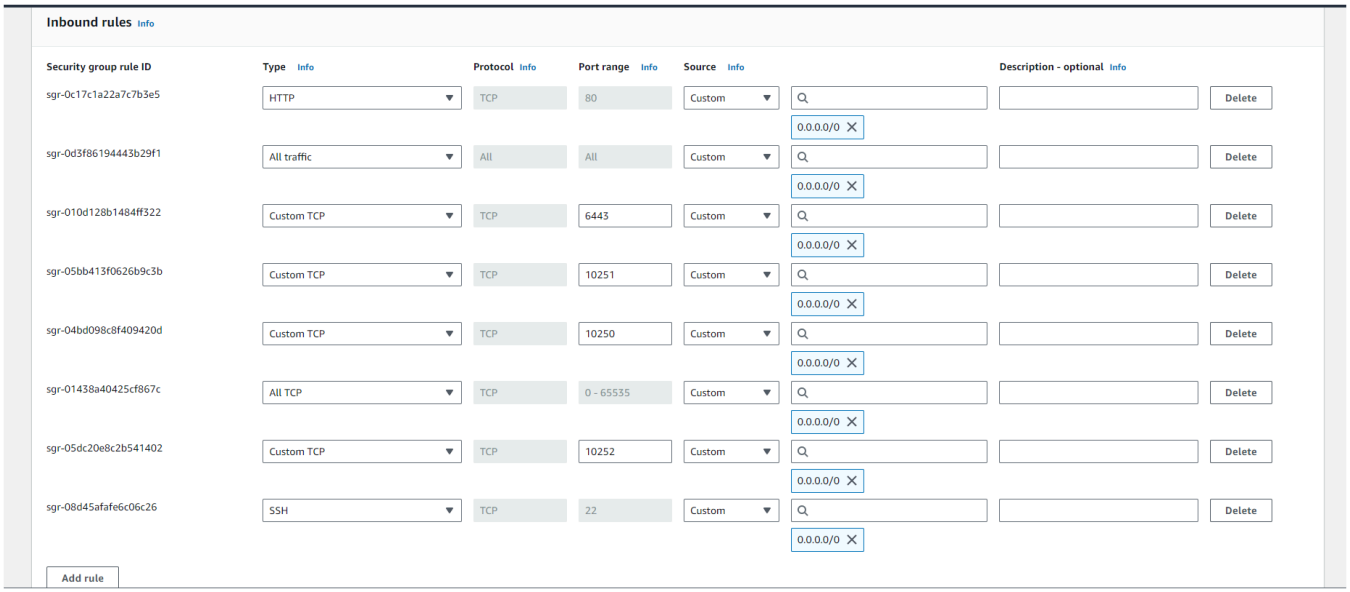
This ensured that the Kubernetes cluster had the necessary resources to function smoothly

Prerequisites :

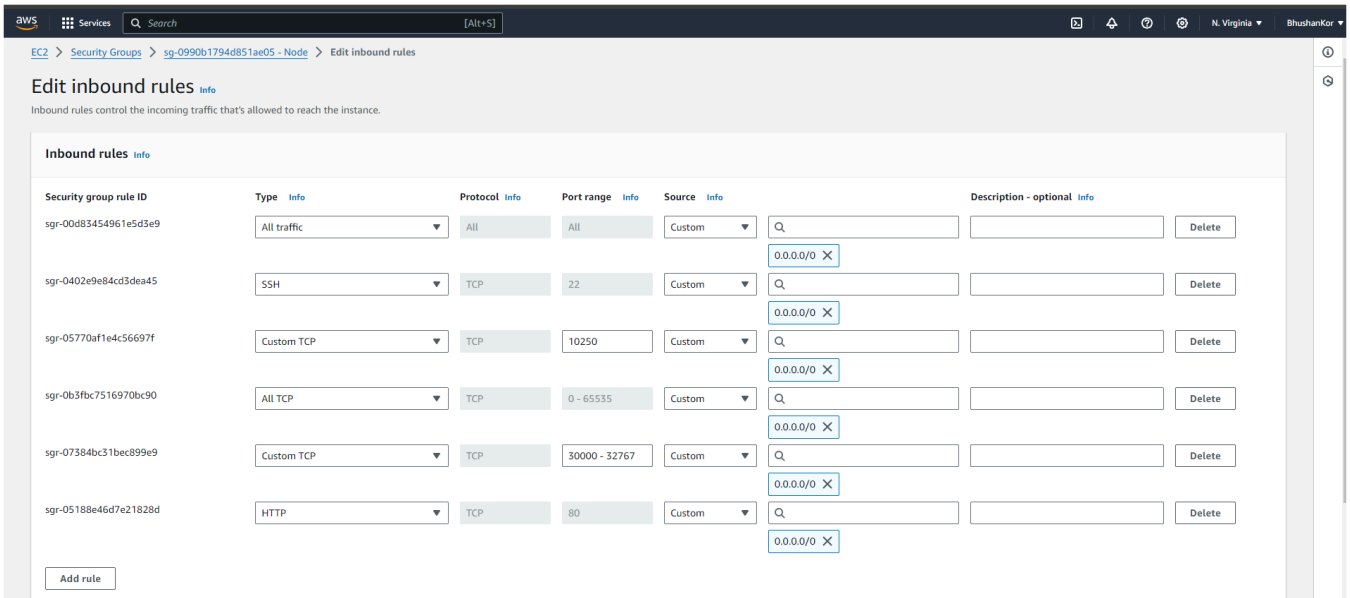
Create 2 Security Groups for Master and Nodes and add the following rules inbound rules in those Groups.

Name:Krushikesh Shelar **2024-2025** **D15C** **Roll No:51**

Master:



Node :

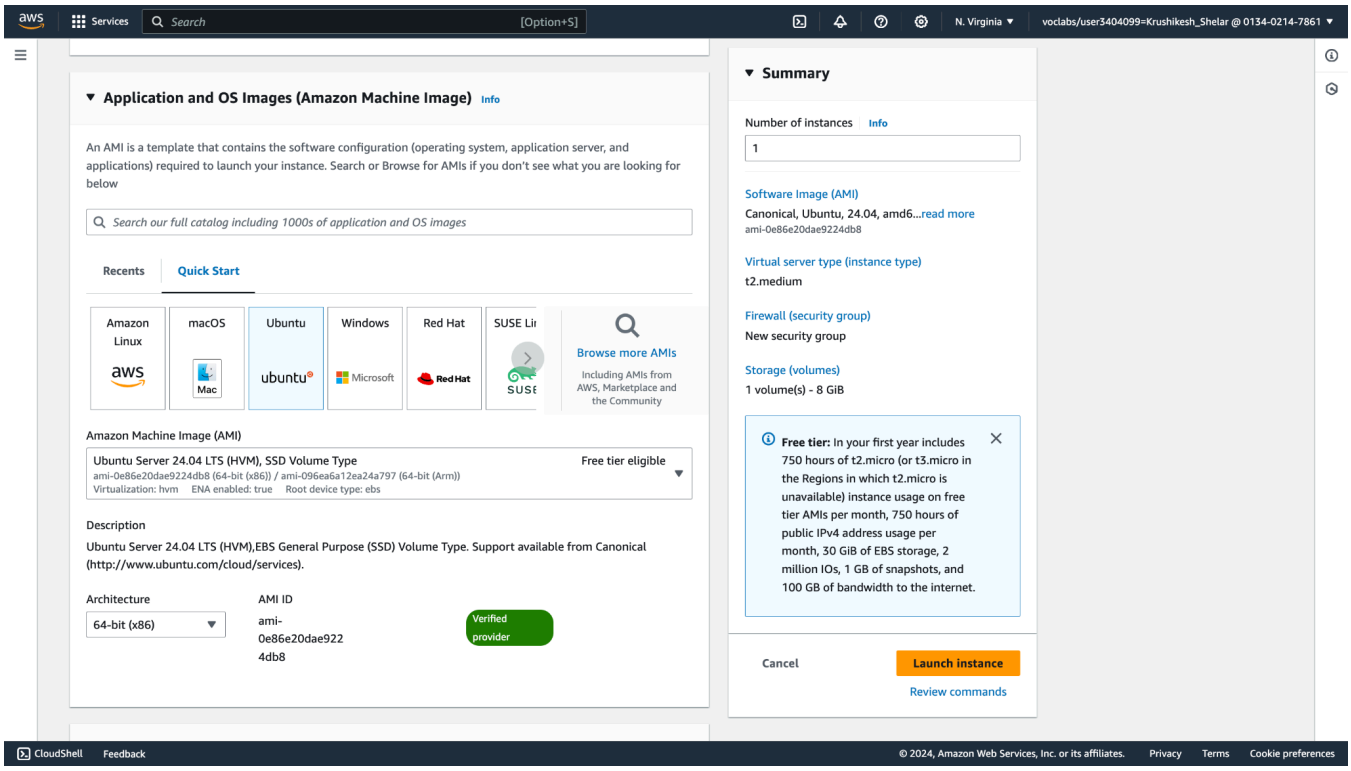


Step 1: Log in to your AWS Academy/personal account and launch 3 new Ec2 Instances. Select Ubuntu as AMI and t2.medium as Instance Type and create a key of type RSA with .pem extension and move the downloaded key to the new folder.We can use 3 Different keys or 1 common key also.

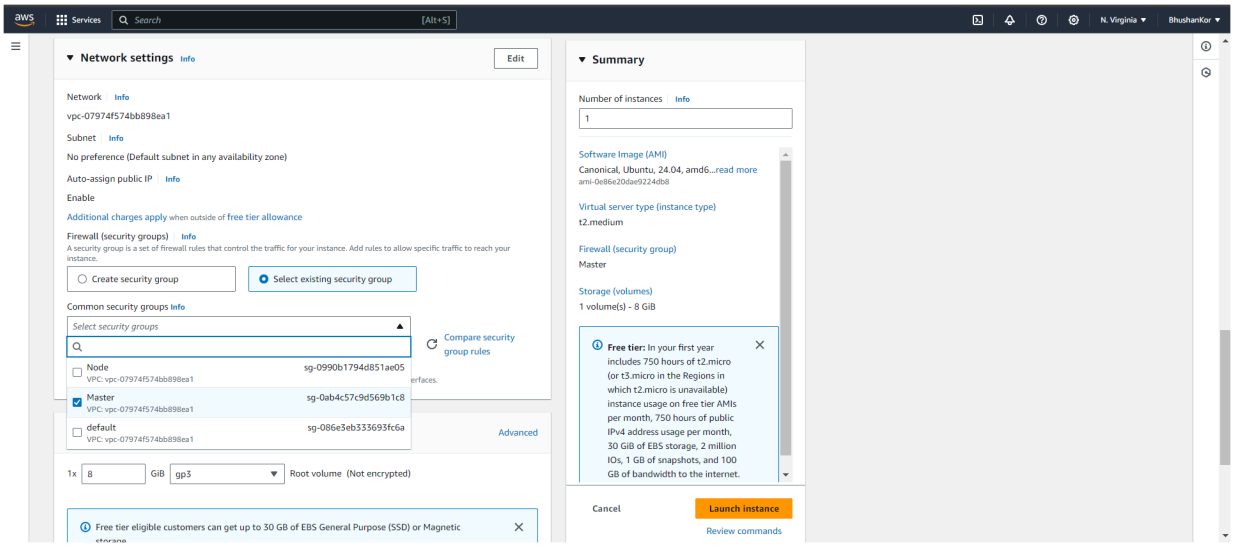
Name:Krushikesh Shelar **2024-2025** **D15C** **Roll No:51**

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:

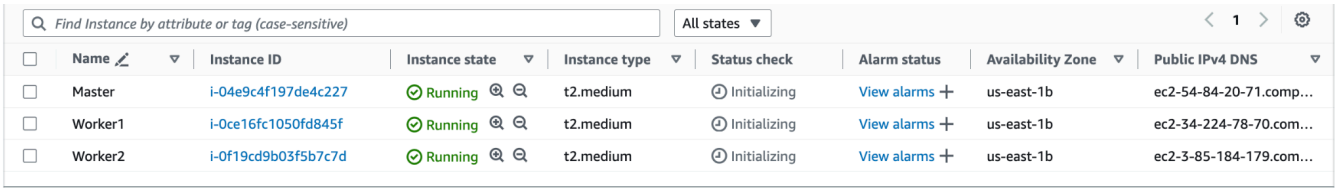


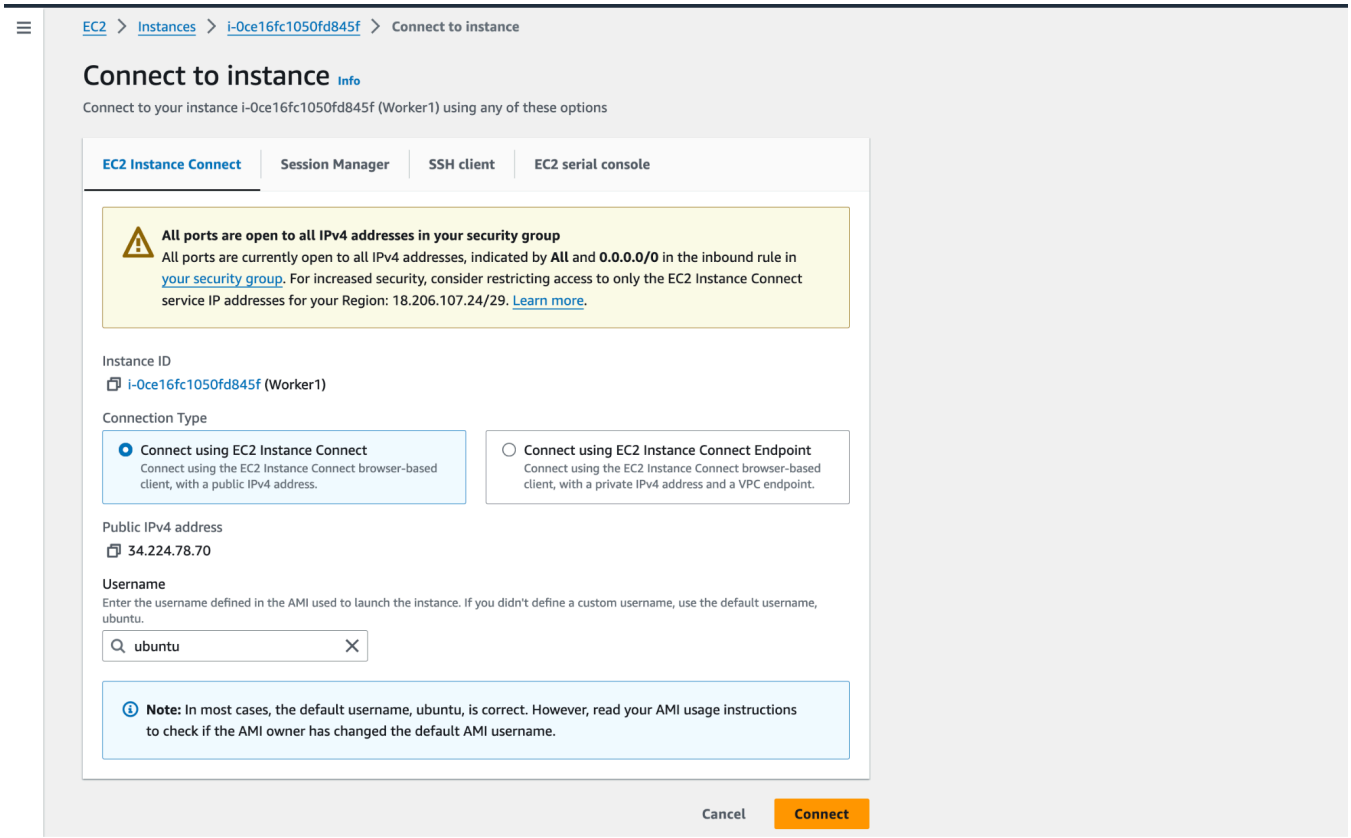
Do Same for 2 Workers and use security groups of Worker for that.



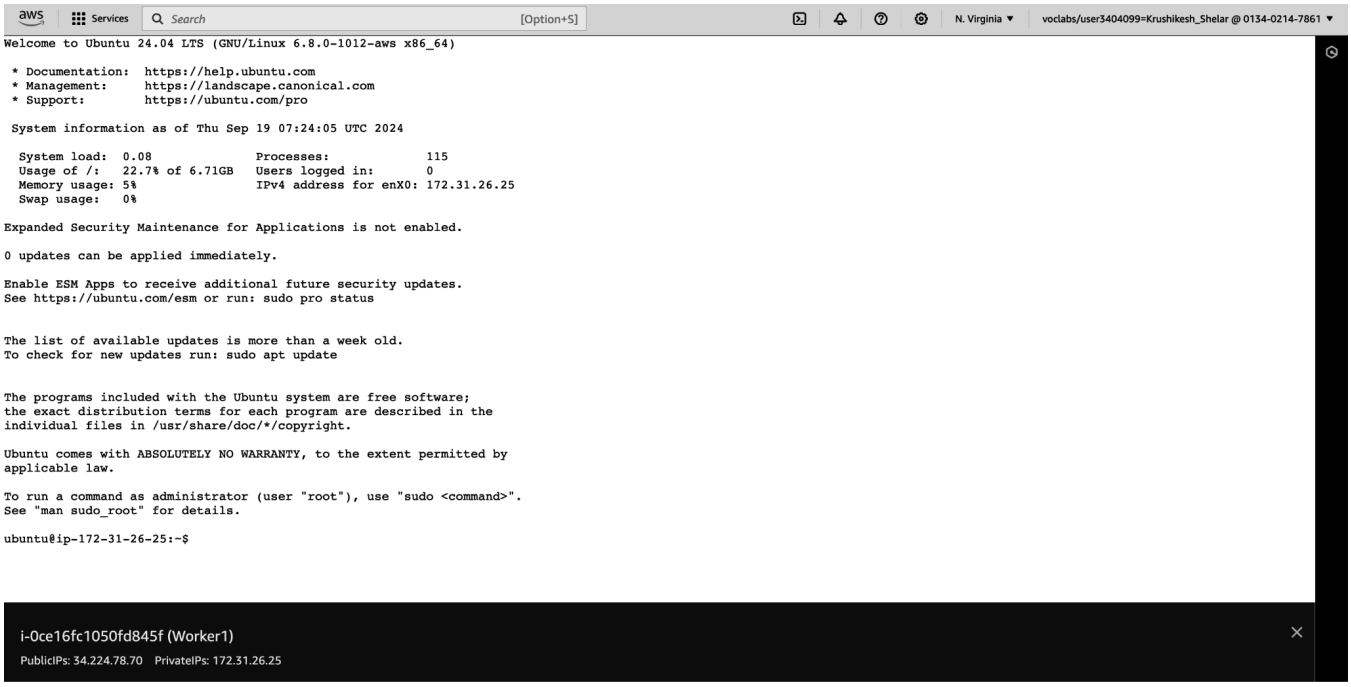
Step 2: After creating the instances click on Connect & connect all 3 instances and navigate to EC2 Instance Connect.

Name:Krushikesh Shelar **2024-2025** **D15C** **Roll No:51**





Successful Connection:



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-172-31-17-92:~$ 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 instead (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_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
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:7 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [13.8 kB]
```

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

```
Setting up pigz (2.8-1) ...
Setting up docker-ce-rootless-extras (5:27.2.1-1~ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.2.1-1~ubuntu.24.04~noble) ...
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 → /usr/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-172-31-17-92:~$ docker --version
Docker version 27.2.1, build 9e34c9b
```

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

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

```
ubuntu@ip-172-31-17-92:~$ docker --version
Docker version 27.2.1, build 9e34c9b
ubuntu@ip-172-31-17-92:~$ 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-172-31-17-92:~$ 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-172-31-17-92:~$
```

Step 5: Run the below command to install Kubernetes.
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-172-31-20-171:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyri
ng.gpg
ubuntu@ip-172-31-20-171:~$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /et
c/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-172-31-27-176:~$ 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

```
Setting up kubernetes-cni (1.5.1-1.1) ...
Setting up kubeadm (1.31.1-1.1) ...
Setting up kubelet (1.31.1-1.1) ...
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.
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
ubuntu@ip-172-31-17-92:~$
```

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

```
Unpacking runc (1.1.12-0ubuntu3.1) ...
Selecting previously unselected package containerd.
Preparing to unpack .../containerd_1.7.12-0ubuntu4.1_amd64.deb ...
Unpacking containerd (1.7.12-0ubuntu4.1) ...
Setting up runc (1.1.12-0ubuntu3.1) ...
Setting up containerd (1.7.12-0ubuntu4.1) ...
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-172-31-17-92:~$
```

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-172-31-17-92:~$ 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; preset: enabled)
     Active: active (running) since Thu 2024-09-19 07:47:29 UTC; 269ms ago
       Docs: https://containerd.io
   Main PID: 5079 (containerd)
      Tasks: 8
     Memory: 13.3M (peak: 14.0M)
        CPU: 69ms
     CGroup: /system.slice/containerd.service
             └─5079 /usr/bin/containerd

Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274330566Z" level=info msg="Start subscribing containerd event"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274370852Z" level=info msg="Start recovering state"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274417909Z" level=info msg="Start event monitor"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274428071Z" level=info msg=serving... address=/run/containerd/containerd.sock.ttrpc
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274430045Z" level=info msg="Start snapshots syncer"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274458201Z" level=info msg="Start cni network conf syncer for default"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274461808Z" level=info msg=serving... address=/run/containerd/containerd.sock
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274465481Z" level=info msg="Start streaming server"
Sep 19 07:47:29 ip-172-31-17-92 containerd[5079]: time="2024-09-19T07:47:29.274564095Z" level=info msg="containerd successfully booted in 0.028249s"
Sep 19 07:47:29 ip-172-31-17-92 systemd[1]: Started containerd.service - containerd container runtime.
```

sudo apt-get install -y socat

```
ubuntu@ip-172-31-17-92:~$  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 required:
  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 (10.8 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.
```

Step 6: Initialize the Kubecluster .Now Perform this Command only for Master.

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

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

```
[mark-control-plane] Marking the node ip-172-31-17-92 as control-plane by adding the taints [node-role.kubernetes.io/control-pla
[bootstrap-token] Using token: ikfl3a.g5zi9o3q6ac1cw2a
[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 certific
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap To
[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 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a \
        --discovery-token-ca-cert-hash sha256:43d27d0955ab782c8877f38d04e1146411c2c510ff75b82f8212bbf7f50db3dd
```

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

```
kubeadm join 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a \
        --discovery-token-ca-cert-hash sha256:43d27d0955ab782c8877f38d04e1146411c2c510ff75b82f8212bbf7f50db3dd
ubuntu@ip-172-31-17-92:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
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.

```
ubuntu@ip-172-31-17-92:~$ kubectl get nodes
NAME              STATUS     ROLES           AGE     VERSION
ip-172-31-17-92   NotReady   control-plane   2m56s   v1.31.1
```

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

sudo kubeadm join 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a
--discovery-token-ca-cert-hash
sha256:43d27d0955ab782c8877f38d04e1146411c2c510ff75b82f8212bbf7f50db3dd

Worker 1:

```
ubuntu@ip-172-31-26-25:~$ kubeadm join 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a \
        --discovery-token-ca-cert-hash sha256:43d27d0955ab782c8877f38d04e1146411c2c510ff75b82f8212bbf7f50db3dd
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR IsPrivilegedUser]: user is not running as root
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
ubuntu@ip-172-31-26-25:~$ sudo kubeadm join 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a \
        --discovery-token-ca-cert-hash sha256:43d27d0955ab782c8877f38d04e1146411c2c510ff75b82f8212bbf7f50db3dd
[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 501.106777ms
[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.
```

Worker 2:

```
adm join 172.31.17.92:6443 --token ikfl3a.g5zi9o3q6ac1cw2a        --discovery-token-ca-cert-hash sha256:43d27d0955ab
[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 1.000966554s
[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.
```

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

```
ubuntu@ip-172-31-17-92:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-17-92    NotReady   control-plane   2m56s   v1.31.1
ubuntu@ip-172-31-17-92:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-17-92    NotReady   control-plane   8m13s   v1.31.1
ip-172-31-19-166   NotReady   <none>          73s     v1.31.1
ip-172-31-26-25    NotReady   <none>          2m9s    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-172-31-17-92:~$ 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-172-31-17-92:~$ 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 2024-09-19 07:52:00 UTC; 11min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 6095 (kubelet)
      Tasks: 11 (limit: 4676)
     Memory: 33.4M (peak: 33.9M)
        CPU: 10.601s
     CGroup: /system.slice/kubelet.service
             └─6095 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.>

Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]:         : unknown
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]:  > pod="kube-system/kube-controller-manager-ip-172-31-17-92" podUID="6b8789cacbbef1a009680071caa0155b" containerName="kube-cont>
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]: E0919 08:03:09.952283    6095 log.go:32] "StopPodSandbox from runtime service failed" err=<
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]:         rpc error: code = Unknown desc = failed to stop container "6e2944b387635ee7a4b18076822bbec229e1c514e6c5ca4aca41090e1ec>
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]:         : unknown
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]:  > podSandboxID="2d281910fe2ca3000c55d391cb651d4941eb3d1b239d3658e4396afd0b3755b4"
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]: E0919 08:03:09.952334    6095 kuberuntime_manager.go:1479] "Failed to stop sandbox" podSandboxID={"Type":"containerd","ID":"2d2>
Sep 19 08:03:09 ip-172-31-17-92 kubelet[6095]: E0919 08:03:09.952373    6095 kubelet.go:1865] "KillPod failed" err="[failed to \"KillContainer\" for \"kube-controller-manager>
Sep 19 08:03:11 ip-172-31-17-92 kubelet[6095]: E0919 08:03:11.145213    6095 kubelet.go:2902] "Container runtime network not ready" networkReady="NetworkReady=false reason:Ne>
Sep 19 08:03:18 ip-172-31-17-92 kubelet[6095]: I0919 08:03:18.207896    6095 pod_startup_latency_tracker.go:104] "Observed pod startup duration" pod="kube-system/calico-node->
...skipping...
● 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 2024-09-19 07:52:00 UTC; 11min ago
```

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

```
ubuntu@ip-172-31-17-92:~$ kubectl get nodes -o wide
NAME              STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-17-92   Ready    control-plane   14m     v1.31.1   172.31.17.92    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-19-166  Ready    <none>          7m11s   v1.31.1   172.31.19.166   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-26-25   Ready    <none>          8m7s    v1.31.1   172.31.26.25    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
```

Now to Rename run this command

```
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>
```

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=worker1
Rename to Node 2:kubectl label node ip-172-31-18-135 kubernetes.io/role=worker2

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.

run kubectl get nodes

```
ubuntu@ip-172-31-17-92:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-17-92    Ready    control-plane   14m     v1.31.1   172.31.17.92    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-19-166   Ready    <none>          7m11s   v1.31.1   172.31.19.166   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-26-25    Ready    <none>          8m7s    v1.31.1   172.31.26.25    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-17-92:~$ kubectl label node ip-172-31-26-25  kubernetes.io/role=worker1
node/ip-172-31-26-25 labeled
ubuntu@ip-172-31-17-92:~$ kubectl label node ip-172-31-19-166  kubernetes.io/role=worker2
node/ip-172-31-19-166 labeled
ubuntu@ip-172-31-17-92:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE   VERSION
ip-172-31-17-92    Ready    control-plane   19m   v1.31.1
ip-172-31-19-166   Ready    worker2         12m   v1.31.1
ip-172-31-26-25    Ready    worker1         13m   v1.31.1
```

Conclusion: In this experiment, we successfully established a Kubernetes cluster consisting of one master node and two worker nodes on AWS EC2 instances. After setting up Docker and installing essential Kubernetes tools (kubelet, kubeadm, kubectl) as well as containerd on all nodes, we proceeded to initialize the master node. The worker nodes were then joined to the cluster. Initially, we encountered a NotReady state for the nodes, but this issue was promptly resolved by installing the Calico network plugin. Additionally, we labeled the nodes to designate their roles (control-plane and worker). As a result, the cluster became fully operational, with all nodes transitioning to the Ready state, effectively demonstrating the successful configuration and orchestration of Kubernetes.