

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

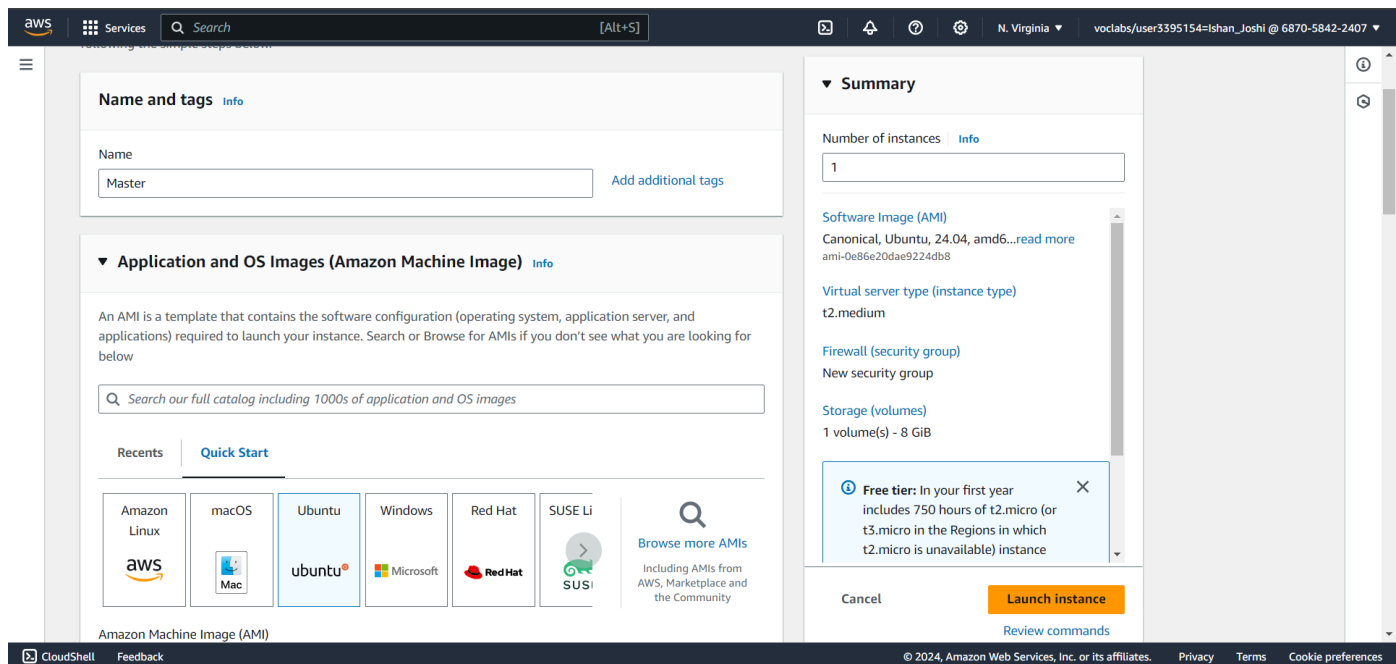
Create 2 Security Groups for Master and Nodes and add the following rules inbound rules in those Groups.

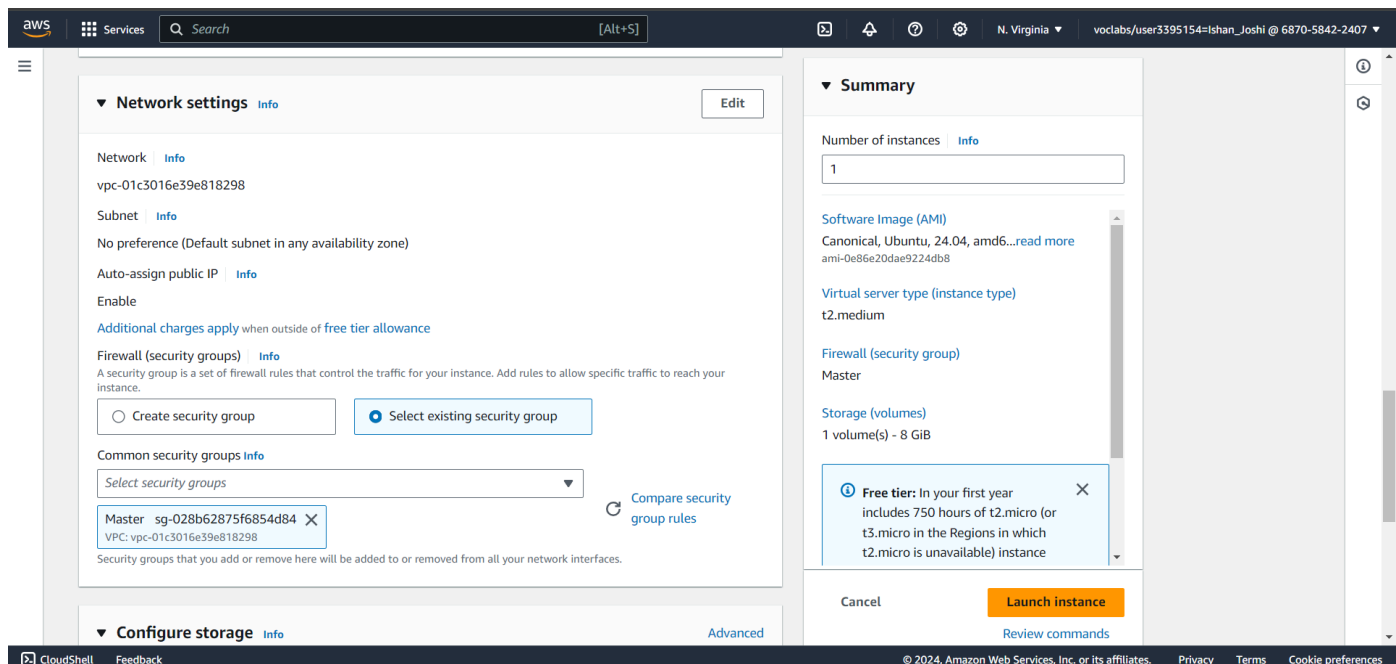
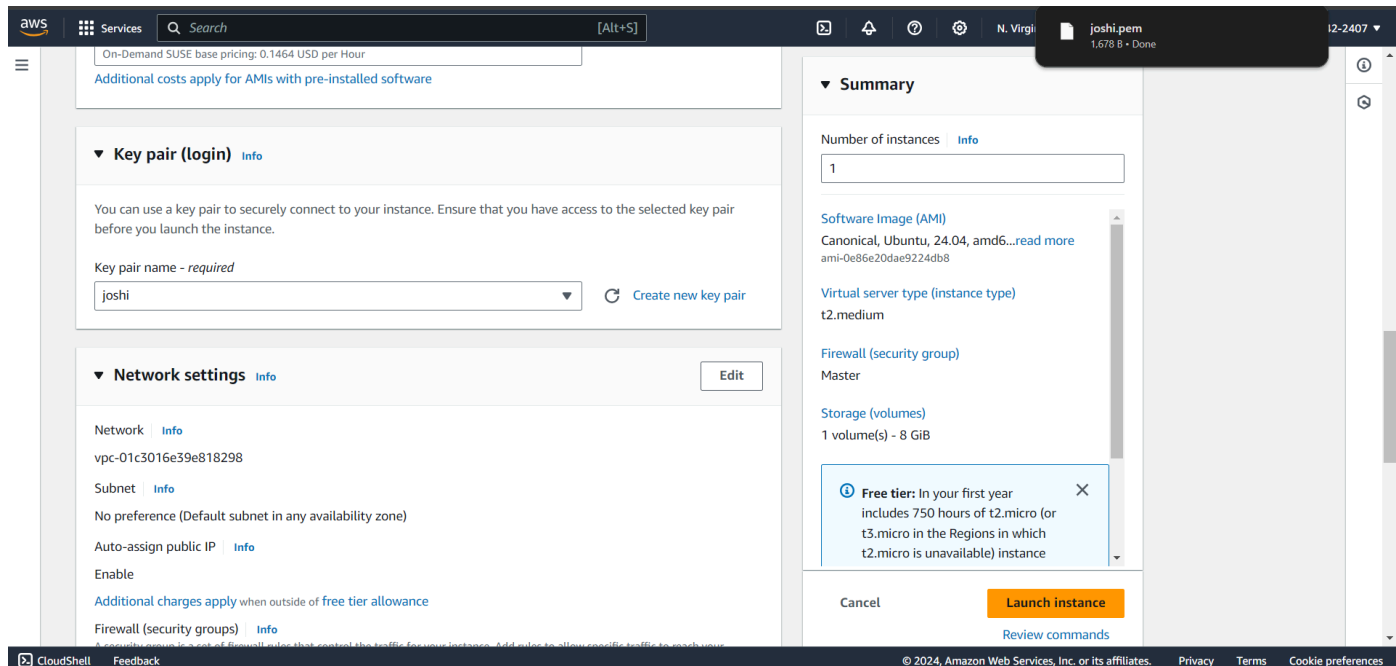
Master:

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. 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 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.

Instances (4) Info

Last updated less than a minute ago

Connect

Instance state

Actions

Launch instances

Find Instance by attribute or tag (case-sensitive)

Running

Instance state = running

Clear filters

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input type="checkbox"/>	Node2	i-00603aab9f9d4cd5b	Running	t2.medium	Initializing	View alarms +	us-east-1d	ec2-44-203-98-216.compute-1.amazonaws.com
<input type="checkbox"/>	Master	i-0fbb153c0c173229d	Running	t2.medium	Initializing	View alarms +	us-east-1d	ec2-54-231-100-100.compute-1.amazonaws.com
<input type="checkbox"/>	Node1	i-0ae444332be12b196	Running	t2.medium	Initializing	View alarms +	us-east-1d	ec2-54-211-100-100.compute-1.amazonaws.com
<input type="checkbox"/>	Master	i-05136994e47f297db	Running	t2.medium	Initializing	View alarms +	us-east-1d	ec2-3-95-100-100.compute-1.amazonaws.com

Select an instance

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](https://ec2-44-203-98-216.compute-1.amazonaws.com)) Master:

aws

Services

Search

[Alt+S]

N. Virginia

voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407

EC2

Instances

i-05136994e47f297db

Connect to instance

Connect to instance

Info

Connect to your instance i-05136994e47f297db (Master) using any of these options

EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

Instance ID

i-05136994e47f297db (Master)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is joshi.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "joshi.pem"
4. Connect to your instance using its Public DNS:
ec2-44-203-98-216.compute-1.amazonaws.com

Example:

ssh -i "joshi.pem" ubuntu@ec2-44-203-98-216.compute-1.amazonaws.com

Note:

In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel

CloudShell

Feedback

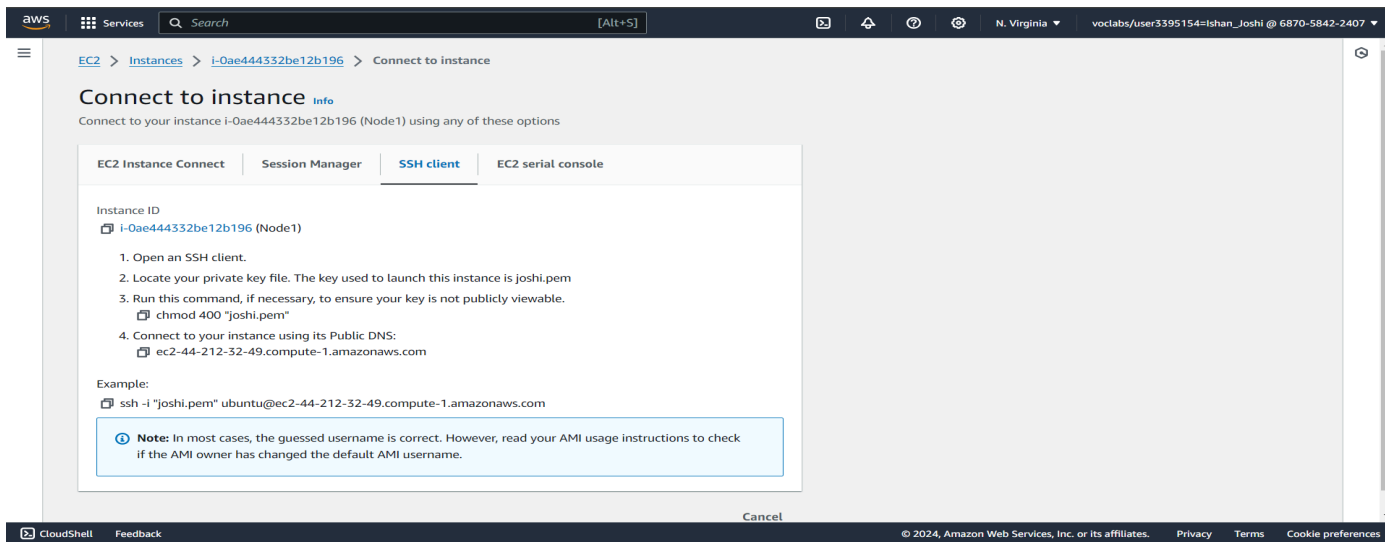
© 2024, Amazon Web Services, Inc. or its affiliates.

Privacy

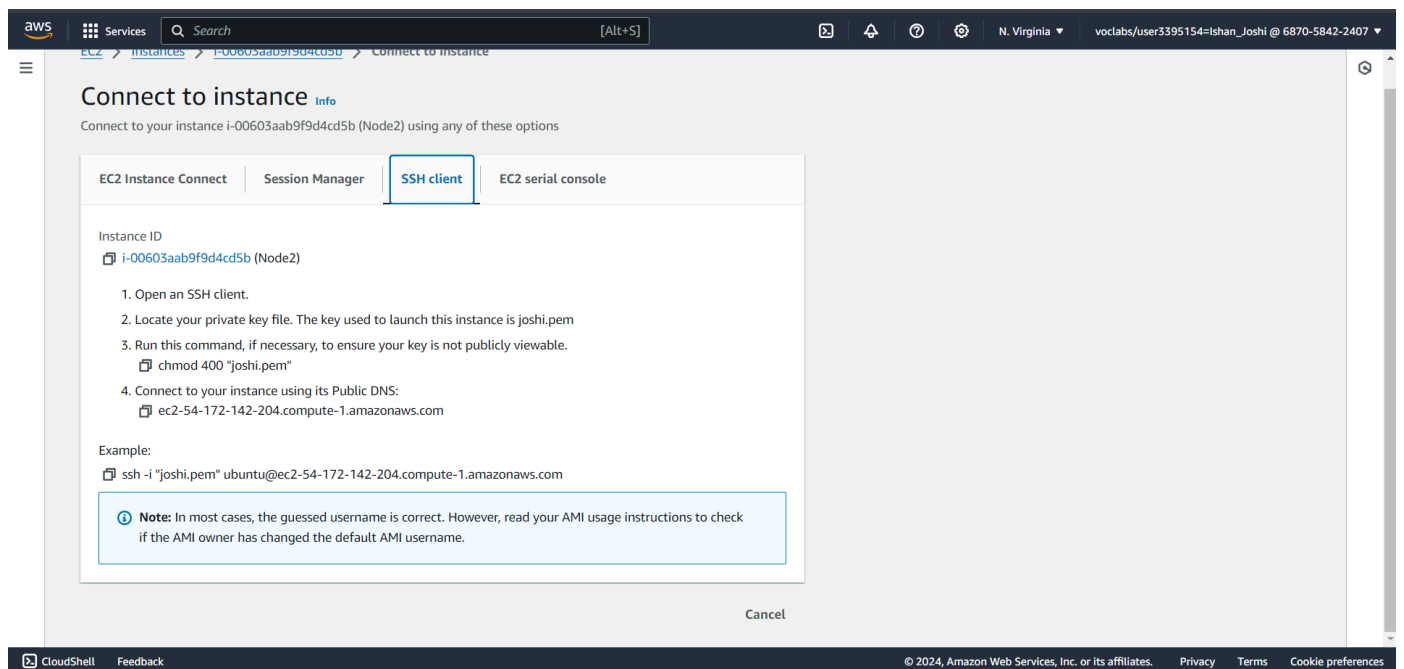
Terms

Cookie preferences

Node 1:



Node 2:



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"
```

```
aws Services Search [Alt+S] N. Virginia voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407
Get:30 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [128 kB]
Get:31 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [8564 B]
Get:32 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [374 kB]
Get:33 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [154 kB]
Get:34 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [45.0 kB]
Get:35 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [14.6 kB]
Get:36 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [353 kB]
Get:37 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted Translation-en [68.1 kB]
Get:38 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 c-n-f Metadata [424 B]
Get:39 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [14.4 kB]
Get:40 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [3608 B]
Get:41 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [212 B]
Get:42 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 c-n-f Metadata [532 B]
Get:43 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:44 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 c-n-f Metadata [112 B]
Get:45 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [10.6 kB]
Get:46 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe Translation-en [10.8 kB]
Get:47 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [17.6 kB]
Get:48 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 c-n-f Metadata [1104 B]
Get:49 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:50 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 c-n-f Metadata [116 B]
Get:51 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:52 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 c-n-f Metadata [116 B]
Fetched 29.1 MB in 4s (7226 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section
in apt-key(8) for details.
ubuntu@ip-172-31-92-242:~$

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
```

sudo apt-get update

sudo apt-get install -y docker-ce

```
aws Services Search [Alt+S] N. Virginia voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407
Setting up docker-buildx-plugin (0.17.1-1-ubuntu.24.04~noble) ...
Setting up containerd.io (1.7.22-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service -> /usr/lib/systemd/system/containerd.service.
Setting up docker-compose-plugin (2.29.7-1-ubuntu.24.04~noble) ...
Setting up libtdl7:amd64 (2.4.7-7build1) ...
Setting up docker-ce-cli (5:27.3.1-1-ubuntu.24.04~noble) ...
Setting up libslirp0:amd64 (4.7.0-1ubuntu3) ...
Setting up pigz (2.8-1) ...
Setting up docker-ce-rootless-extras (5:27.3.1-1-ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.3.1-1-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-172-31-92-242:~$

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242
```

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

```
aws Services Search [Alt+S]
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.3.1-1~ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
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-172-31-92-242:~$ 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-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

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

```
aws Services Search [Alt+S]
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-172-31-92-242:~$ 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-172-31-92-242:~$ 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-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

CloudShell Feedback © 2024 Amazon Web Services, Inc. or its affiliates Privacy Terms Cookie preferences

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

```
aws Services Search [Alt+S]
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-172-31-92-242:~$ 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-172-31-92-242:~$ 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-172-31-92-242:~$ 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
gpg: missing argument for option '-o'
-bash: /etc/apt/keyrings/kubernetes-apt-keyring.gpg: No such file or directory
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

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

```
aws Services Search [Alt+S]
Preparing to unpack .../4-kubernetes-cni_1.5.1-1.1_amd64.deb ...
Unpacking kubernetes-cni (1.5.1-1.1) ...
Selecting previously unselected package kubelet.
Preparing to unpack .../5-kubelet_1.31.1-1.1_amd64.deb ...
Unpacking kubelet (1.31.1-1.1) ...
Setting up conntrack (1:1.4.8-1ubuntu1) ...
Setting up kubectl (1.31.1-1.1) ...
Setting up cri-tools (1.31.1-1.1) ...
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-172-31-92-242:~$
```

i-05136994e47f297db (Master)

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


```
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.12-0ubuntu4.1 [38.6 MB]
Fetched 47.2 MB in 1s (53.3 MB/s)
(Reading database ... 68064 files and directories currently installed.)
Removing docker-ce (5:27.3.1-1-ubuntu.24.04-noble) ...
Removing containerd.io (1.7.22-1) ...
Selecting previously unselected package runc.
(Reading database ... 68044 files and directories currently installed.)
Preparing to unpack ../runc_1.1.12-0ubuntu3.1_amd64.deb ...
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-172-31-92-242:~$
```

i-05136994e47f297db (Master)

PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

sudo mkdir -p /etc/containerd

sudo containerd config default | sudo tee /etc/containerd/config.toml

```
[stream_processors."io.containerd.ocicrypt.decoder.v1.tar"]
accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]
args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
path = "ctd-decoder"
returns = "application/vnd.oci.image.layer.v1.tar"

[stream_processors."io.containerd.ocicrypt.decoder.v1.tar+gzip"]
accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
path = "ctd-decoder"
returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[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"

[tttrpc]
address = ""
gid = 0
uid = 0
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)

PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

sudo systemctl restart containerd

sudo systemctl enable containerd

sudo systemctl status containerd

```
aws Services Search [Alt+S] N. Virginia voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407

address = ""
gid = 0
uid = 0
ubuntu@ip-172-31-87-251:~$ 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 Wed 2024-09-25 04:16:33 UTC; 229ms ago
     Docs: https://containerd.io
    Main PID: 5408 (containerd)
      Tasks: 7
   Memory: 13.2M (peak: 14.2M)
      CPU: 62ms
   CGroup: /system.slice/containerd.service
           └─5408 /usr/bin/containerd

Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.500535912Z" level=info msg=serving... address=/run/containerd/containerd.sock.ttrpc
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.500568795Z" level=info msg=serving... address=/run/containerd/containerd.sock
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.500636299Z" level=info msg="Start subscribing containerd event"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.500661162Z" level=info msg="Start recovering state"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.502149073Z" level=info msg="Start event monitor"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.502184930Z" level=info msg="Start snapshots syncer"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.502196121Z" level=info msg="Start cni network conf syncer for default"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.502202781Z" level=info msg="Start streaming server"
Sep 25 04:16:33 ip-172-31-87-251 containerd[5408]: time="2024-09-25T04:16:33.502254362Z" level=info msg="containerd successfully booted in 0.029882s"
Sep 25 04:16:33 ip-172-31-87-251 systemd[1]: Started containerd.service - containerd container runtime.
ubuntu@ip-172-31-87-251:~$
```

i-00603aab9f9d4cd5b (Node2)

PublicIPs: 54.172.142.204 PrivateIPs: 172.31.87.251

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

sudo apt-get install -y socat

```
aws Services Search [Alt+S] N. Virginia voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407

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 139 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 (17.4 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.

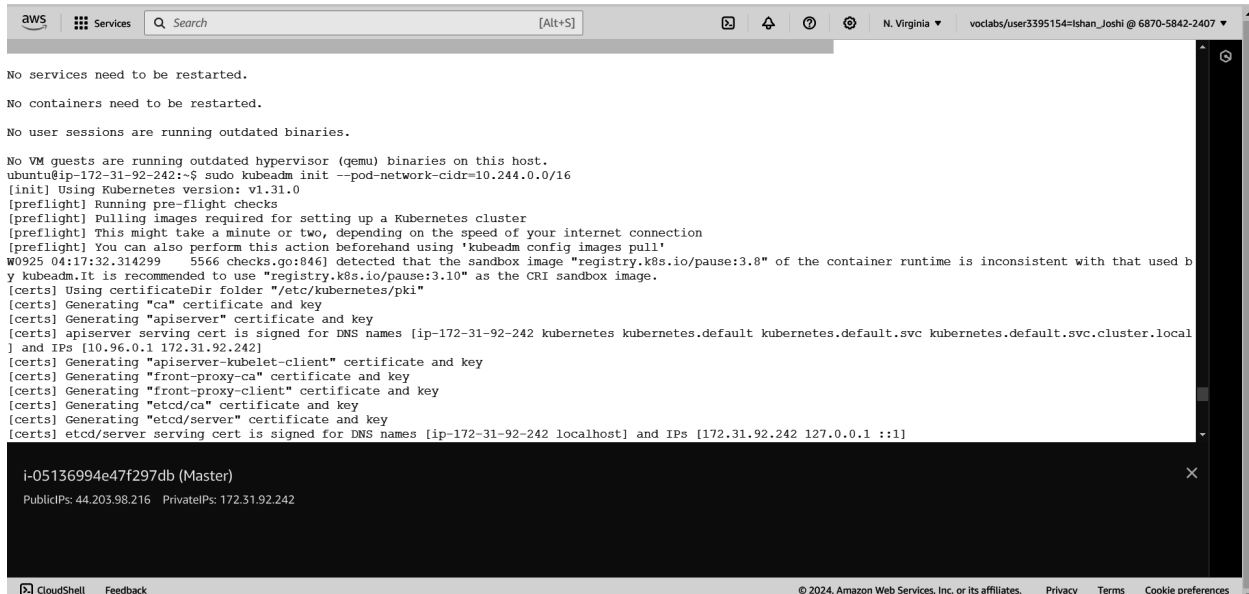
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)

PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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



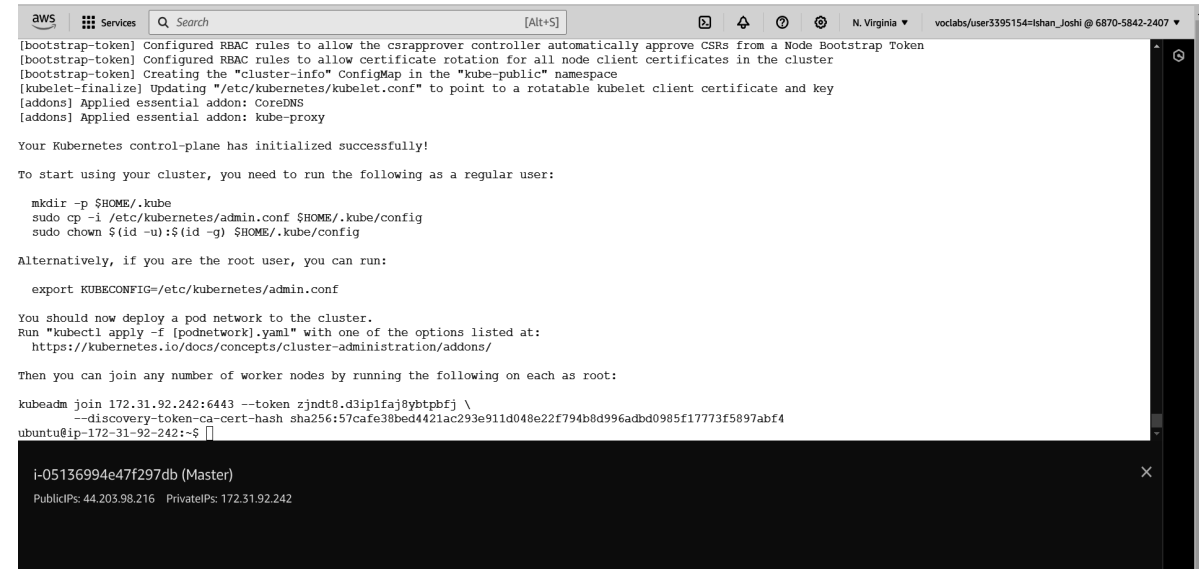
```
No services need to be restarted.

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-172-31-92-242:~$ 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'
W0925 04:17:32.314299 5566 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-172-31-92-242 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.92.242]
[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-172-31-92-242 localhost] and IPs [172.31.92.242 127.0.0.1 ::1]

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242
```



```
[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 172.31.92.242:6443 --token zjndt8.d3lplfaj8ybtgbfj \
--discovery-token-ca-cert-hash sha256:57cafe38bed4421ac293e911d048e22f794b8d996adb0905f17773f5897abf4
ubuntu@ip-172-31-92-242:~$


i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242
```

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



```
ubuntu@ip-172-31-92-242:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-92-242:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@ip-172-31-92-242:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-92-242:~$

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242
```

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

on nodes.

```
ubuntu@ip-172-31-92-242:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-92-242    NotReady control-plane 2m43s  v1.31.1
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)

PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

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:

```
[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-172-31-85-255:~$ kubeadm join 172.31.92.242:6443 --token zjndt8.d3iplfaj8ybtbpfj \
--discovery-token-ca-cert-hash sha256:57cafe38bed4421ac293e911d048e22f794b8d996adb0985f1773f5897abf4
[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-172-31-85-255:~$ sudo kubeadm join 172.31.92.242:6443 --token zjndt8.d3iplfaj8ybtbpfj \
--discovery-token-ca-cert-hash sha256:57cafe38bed4421ac293e911d048e22f794b8d996adb0985f1773f5897abf4
[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 1.000847427s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

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

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

ubuntu@ip-172-31-85-255:~$
```

i-0ae444332be12b196 (Node1)

PublicIPs: 44.212.32.49 PrivateIPs: 172.31.85.255

Node 2:

```
ubuntu@ip-172-31-87-251:~$ kubeadm join 172.31.92.242:6443 --token zjndt8.d3iplfaj8ybtbpfj \
--discovery-token-ca-cert-hash sha256:57cafe38bed4421ac293e911d048e22f794b8d996adb0985f1773f5897abf4
[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-172-31-87-251:~$ sudo kubeadm join 172.31.92.242:6443 --token zjndt8.d3iplfaj8ybtbpfj \
--discovery-token-ca-cert-hash sha256:57cafe38bed4421ac293e911d048e22f794b8d996adb0985f1773f5897abf4
[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 1.001745521s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

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

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

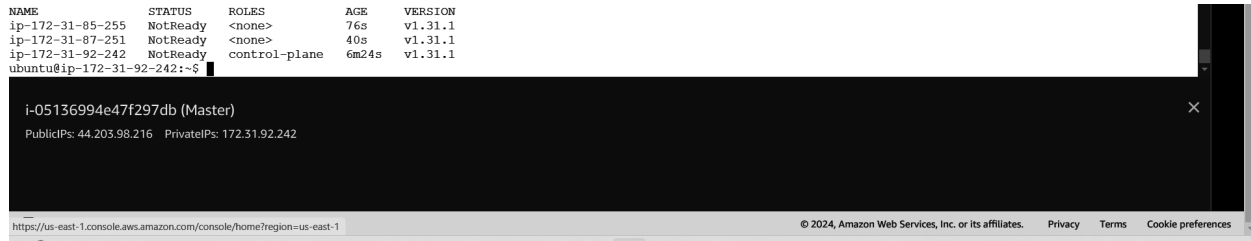
ubuntu@ip-172-31-87-251:~$
```

i-00603aab9f9d4cd5b (Node2)

PublicIPs: 54.172.142.204 PrivateIPs: 172.31.87.251

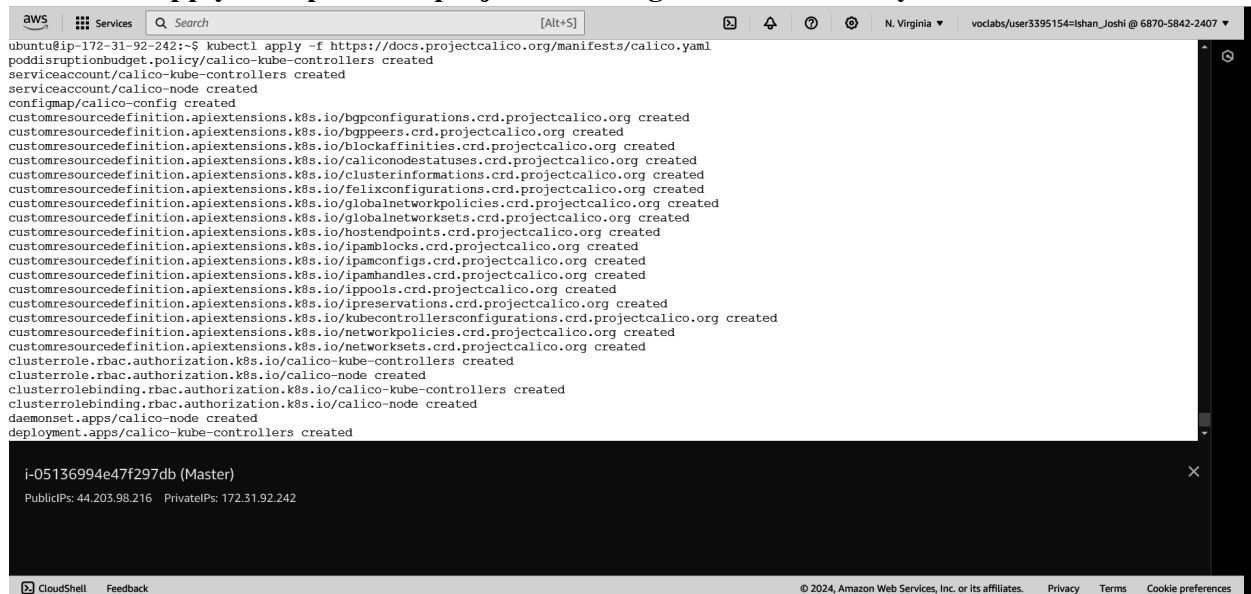
Step 9: Now Run the command `kubectl get nodes` to see the nodes after executing `Join` command on nodes.

```
NAME                 STATUS    ROLES    AGE   VERSION
ip-172-31-85-255     NotReady <none>   76s   v1.31.1
ip-172-31-87-251     NotReady <none>   40s   v1.31.1
ip-172-31-92-242     NotReady control-plane 6m24s v1.31.1
ubuntu@ip-172-31-92-242:~$
```

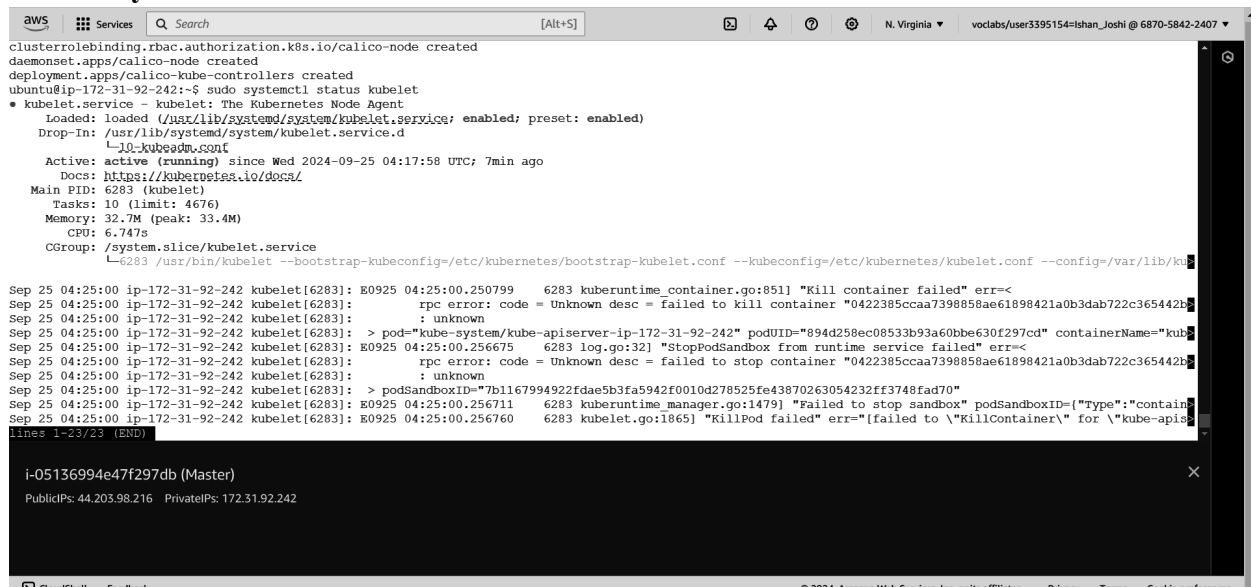


Step 10: Since Status is `NotReady` we have to add a network plugin. And also we have to give the name to the nodes.

`kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml`



`sudo systemctl status kubelet`



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

```
ubuntu@ip-172-31-92-242:~$ kubectl get nodes -o wide
NAME          STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-85-255    Ready    <none>    2m47s   v1.31.1   172.31.85.255   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-87-251    Ready    <none>    2m11s   v1.31.1   172.31.87.251   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-92-242    Ready    control-plane 7m55s   v1.31.1   172.31.92.242   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-92-242:~$
```

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-172-31-92-242:~$ kubectl label node ip-172-31-85-255 kubernetes.io/role=Node1
node/ip-172-31-85-255 labeled
ubuntu@ip-172-31-92-242:~$ kubectl label node ip-172-31-87-251 kubernetes.io/role=Node2
node/ip-172-31-87-251 labeled
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

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-172-31-92-242:~$ kubectl get nodes -o wide
NAME          STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-85-255    Ready    Node1     8m34s   v1.31.1   172.31.85.255   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-87-251    Ready    Node2     7m58s   v1.31.1   172.31.87.251   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ip-172-31-92-242    Ready    control-plane 13m   v1.31.1   172.31.92.242   <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

Or run kubectl get nodes

```
ubuntu@ip-172-31-92-242:~$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
ip-172-31-85-255    Ready    Node1     9m   v1.31.1
ip-172-31-87-251    Ready    Node2     8m24s v1.31.1
ip-172-31-92-242    Ready    control-plane 14m   v1.31.1
ubuntu@ip-172-31-92-242:~$
```

i-05136994e47f297db (Master)
PublicIPs: 44.203.98.216 PrivateIPs: 172.31.92.242

Conclusion: In this experiment, we effectively established a Kubernetes cluster comprising a single master and two worker nodes using AWS EC2 instances. We began by installing Docker, Kubernetes components (kubelet, kubeadm, and kubectl), and containerd on each node. Following this, the master node was initialized, and the worker nodes were joined to the cluster. Although the nodes initially appeared in the NotReady state, installing the Calico network plugin resolved this issue. We also assigned appropriate labels to the nodes, designating their roles as control-plane and worker. Ultimately,

the cluster became fully operational with all nodes in the Ready state, confirming the successful setup and orchestration of Kubernetes.