ADITYA LALWANI D15C Roll No. 30

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:

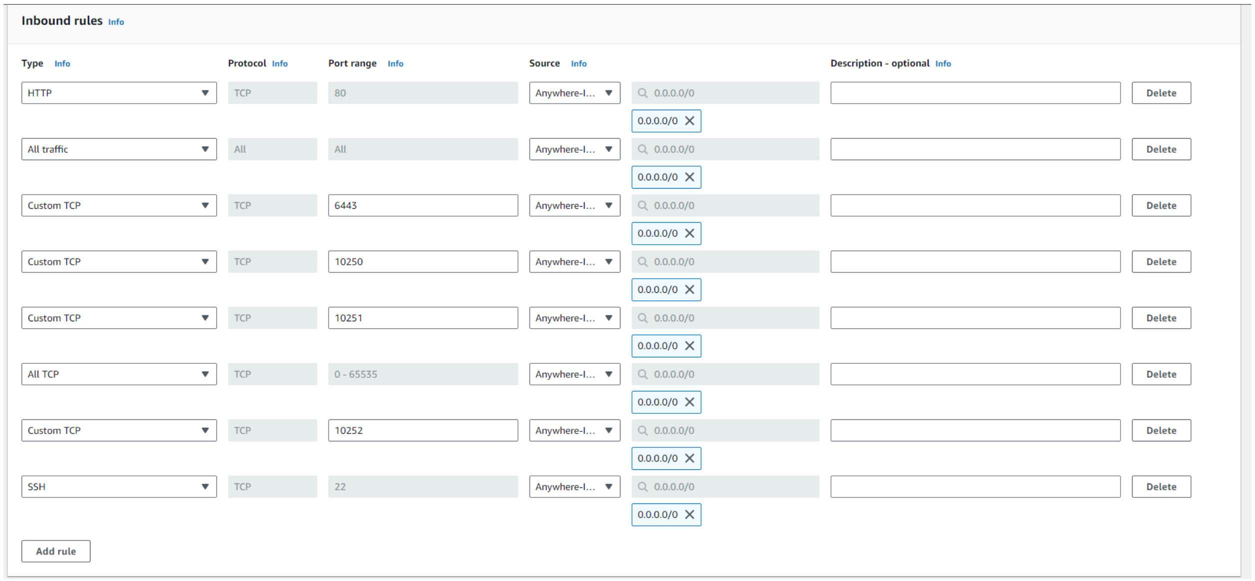
- Managing resource consumption by applications or teams
- Distributing application load evenly across the infrastructure
- Automatically load balancing requests across multiple instances of an application
- Monitoring resource usage to prevent applications from exceeding resource limits and automatically 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.

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

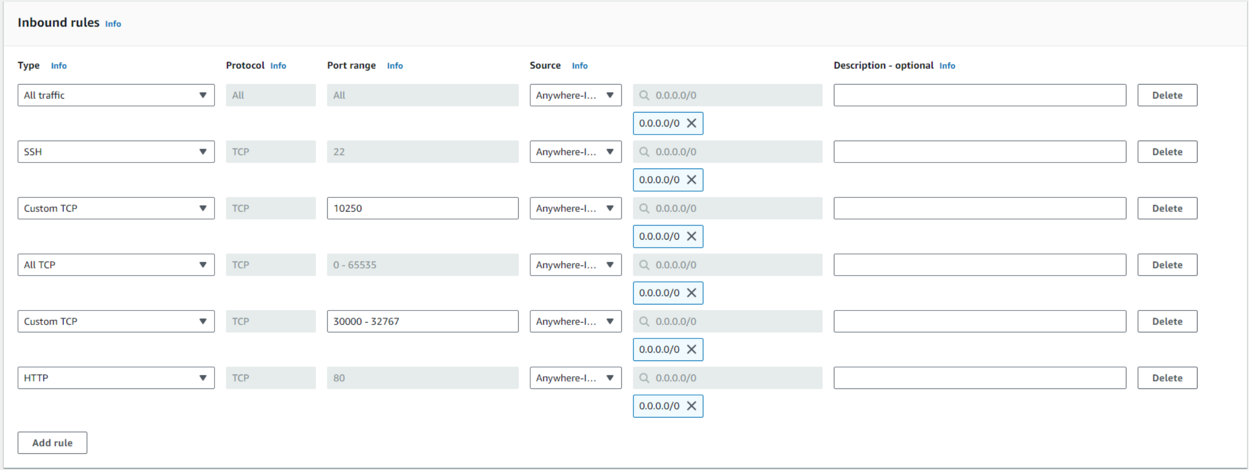
Master:



Inbound rules Info

Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
HTTP	TCP	80	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
All traffic	All	All	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	6443	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10250	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10251	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
All TCP	TCP	0 - 65535	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10252	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
SSH	TCP	22	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete

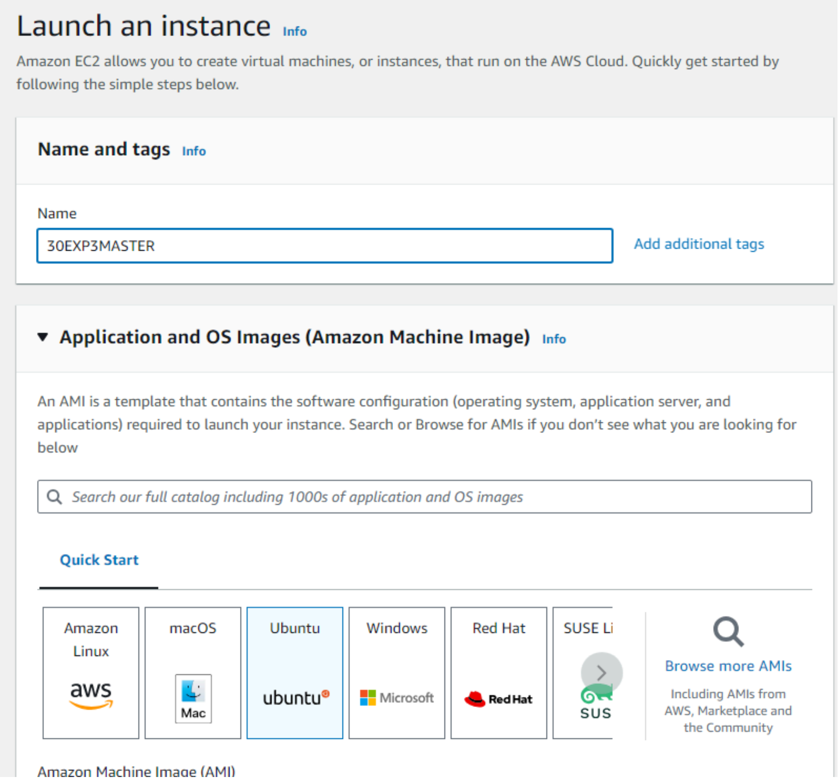
Node :



Inbound rules Info

Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
All traffic	All	All	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
SSH	TCP	22	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10250	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
All TCP	TCP	0 - 65535	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	30000 - 32767	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
HTTP	TCP	80	Anywhere-I...	0.0.0.0/0 0.0.0.0/0 ✕		Delete

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.



Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: 30EXP3MASTER

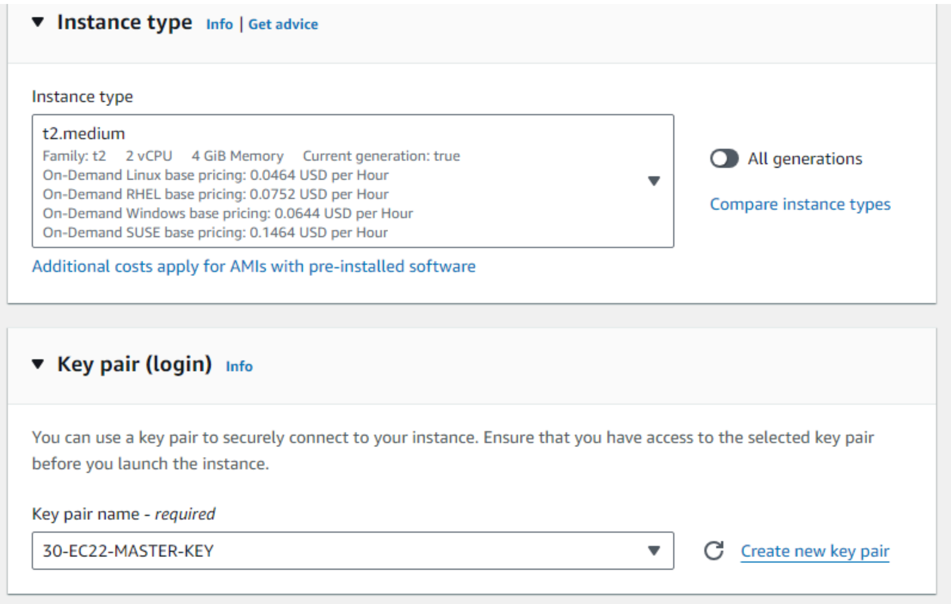
Add additional tags

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)



▼ Instance type Info | Get advice

Instance type

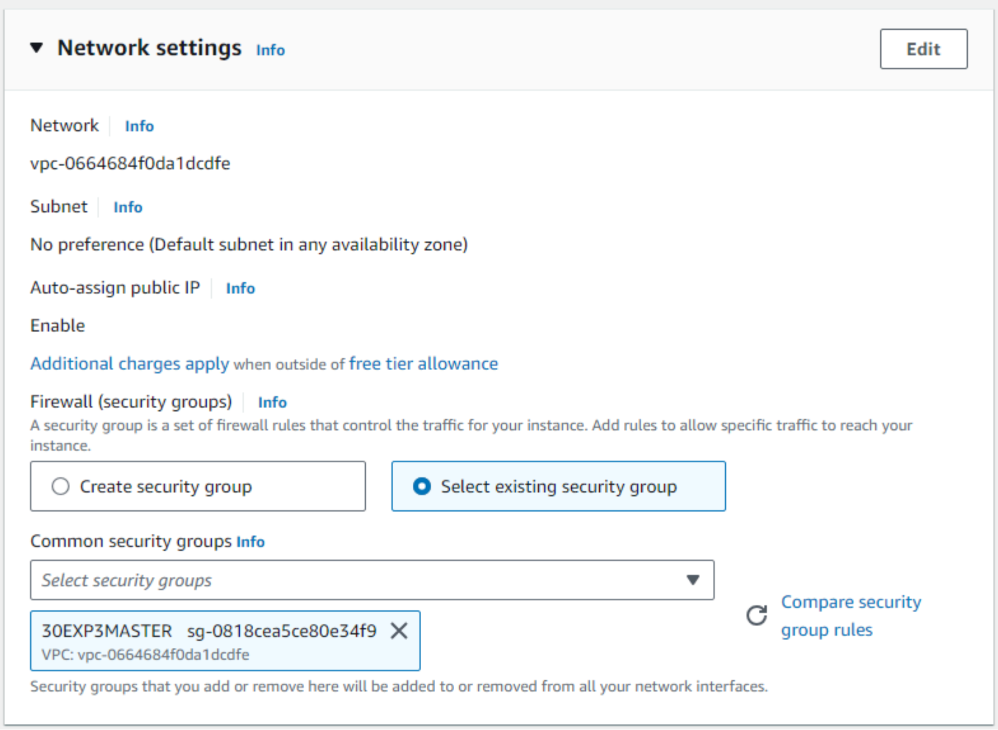
t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0464 USD per Hour
On-Demand RHEL base pricing: 0.0752 USD per Hour
On-Demand Windows base pricing: 0.0644 USD per Hour
On-Demand SUSE base pricing: 0.1464 USD per Hour

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*: 30-EC22-MASTER-KEY



Network Info

vpc-0664684f0da1dcdfe

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

○ Create security group ● Select existing security group

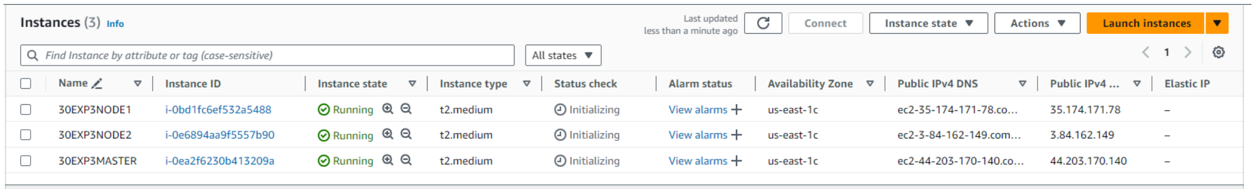
Common security groups Info

Select security groups ▼

30EXP3MASTER sg-0818cea5ce80e34f9 ✕
VPC: vpc-0664684f0da1dcdfe

Security groups that you add or remove here will be added to or removed from all your network interfaces.

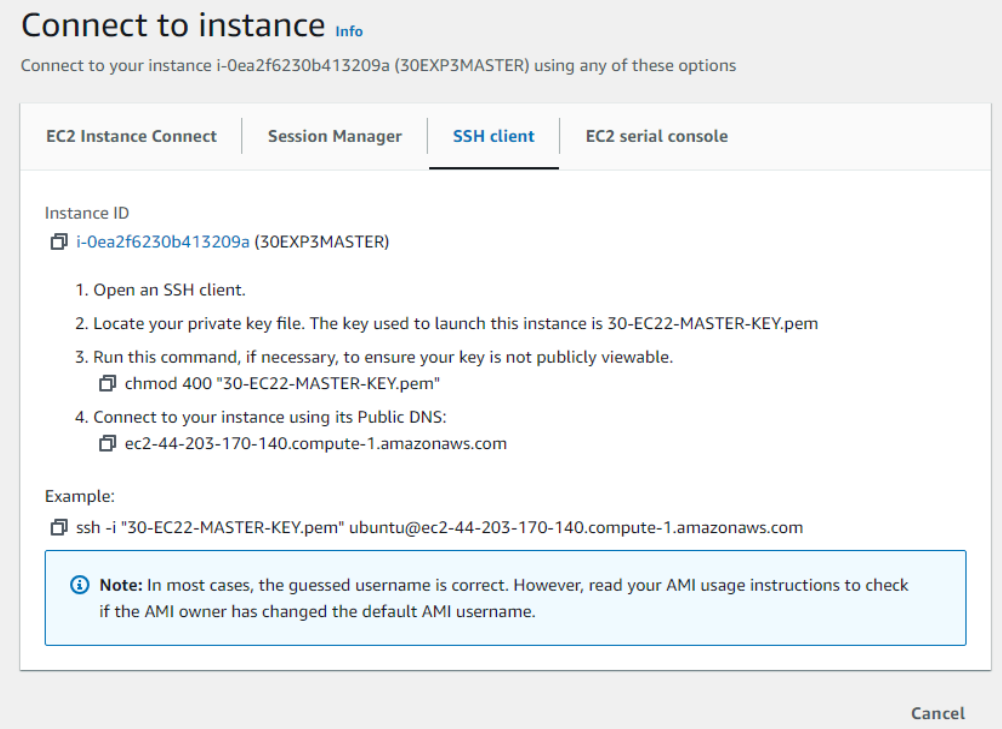
All instances



Instances (3) Info
Last updated less than a minute ago
Connect
Instance state ▼
Actions ▼
Launch instances ▼
Find Instance by attribute or tag (case-sensitive)
All states ▼
1

□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
□	30EXP3NODE1	i-0bd1fc6ef532a5488	✓ Running	t2.medium	Initializing	View alarms +	us-east-1c	ec2-35-174-171-78.co...	35.174.171.78	–
□	30EXP3NODE2	i-0e6894aa9f5557b90	✓ Running	t2.medium	Initializing	View alarms +	us-east-1c	ec2-3-84-162-149.com...	3.84.162.149	–
□	30EXP3MASTER	i-0ea2f6230b413209a	✓ Running	t2.medium	Initializing	View alarms +	us-east-1c	ec2-44-203-170-140.co...	44.203.170.140	–

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.(



Connect to instance

Connect to your instance i-0ea2f6230b413209a (30EXP3MASTER) using any of these options

Instance ID

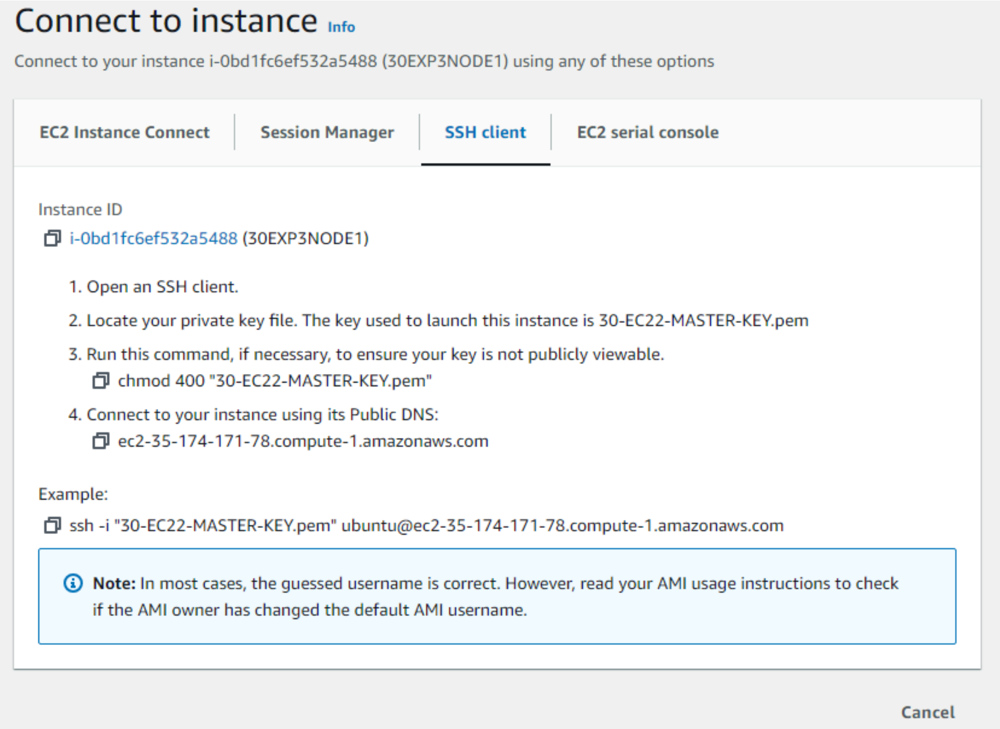
i-0ea2f6230b413209a (30EXP3MASTER)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is 30-EC22-MASTER-KEY.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   `chmod 400 "30-EC22-MASTER-KEY.pem"`
4. Connect to your instance using its Public DNS:
   ec2-44-203-170-140.compute-1.amazonaws.com

Example:

```
ssh -i "30-EC22-MASTER-KEY.pem" ubuntu@ec2-44-203-170-140.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.



Connect to instance

Connect to your instance i-0bd1fc6ef532a5488 (30EXP3NODE1) using any of these options

Instance ID

i-0bd1fc6ef532a5488 (30EXP3NODE1)

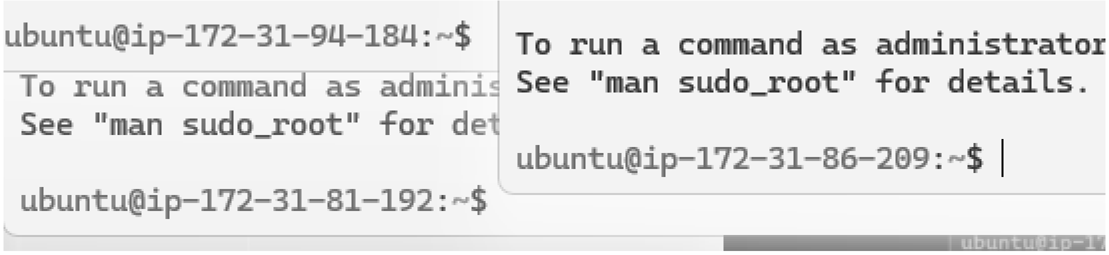
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is 30-EC22-MASTER-KEY.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   `chmod 400 "30-EC22-MASTER-KEY.pem"`
4. Connect to your instance using its Public DNS:
   ec2-35-174-171-78.compute-1.amazonaws.com

Example:

```
ssh -i "30-EC22-MASTER-KEY.pem" ubuntu@ec2-35-174-171-78.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.

All 3 instances connected successfully



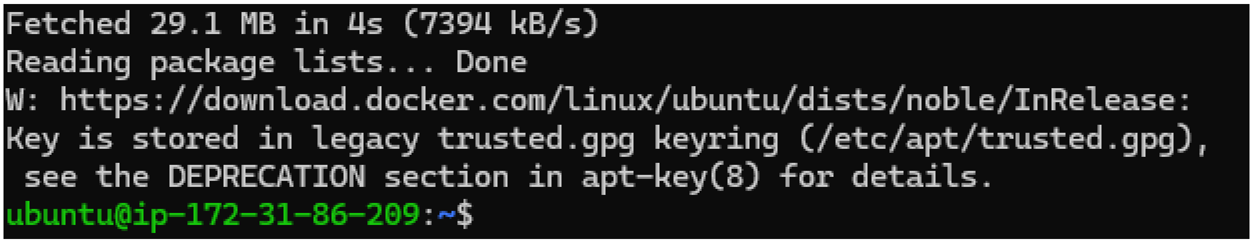
```
ubuntu@ip-172-31-94-184:~$
 To run a command as adminis
 See "man sudo_root" for det

 ubuntu@ip-172-31-81-192:~$
```

```
To run a command as administrator
See "man sudo_root" for details.

ubuntu@ip-172-31-86-209:~$
```

```
ubuntu@ip-17
```

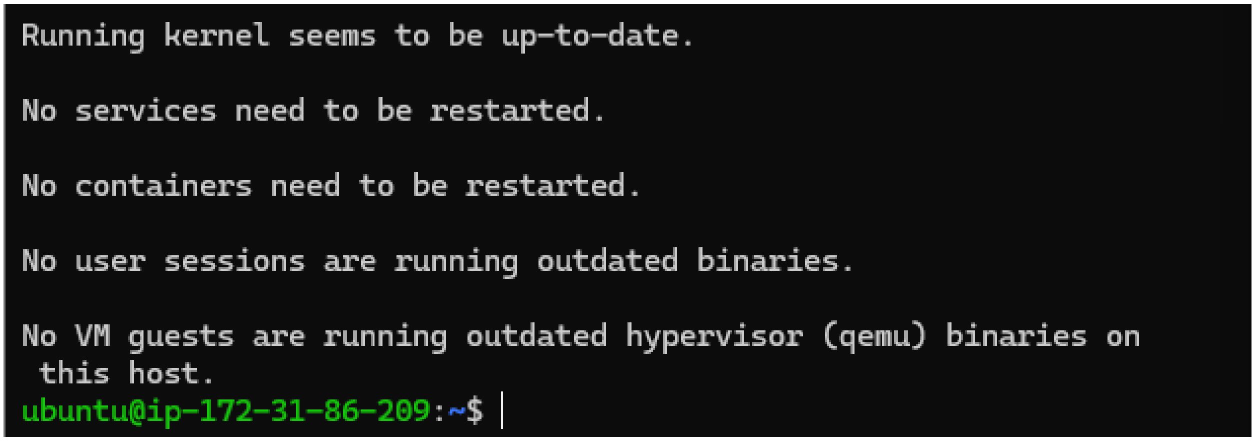
Run on Master,Node 1,and Node 2 the below commands to install and setup Docker in Master, Node1, and Node2.

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"



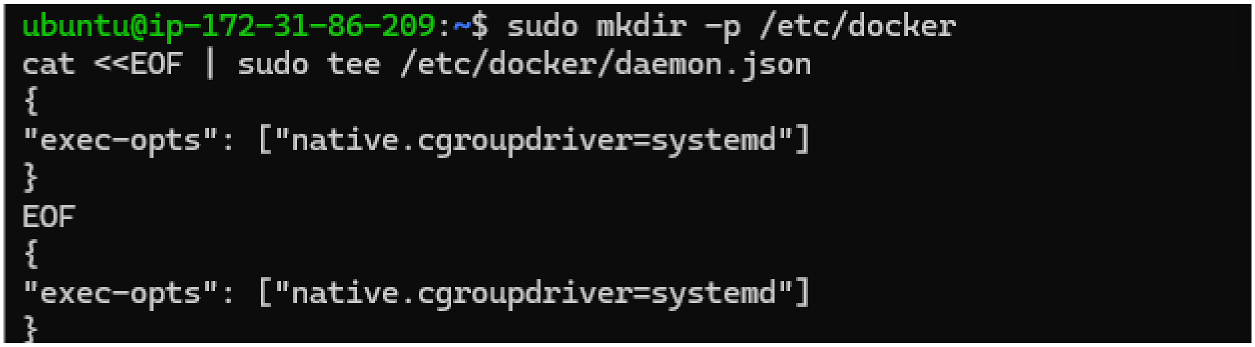
```
Fetched 29.1 MB in 4s (7394 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease:
Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg),
 see the DEPRECATION section in apt-key(8) for details.
ubuntu@ip-172-31-86-209:~$
```

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



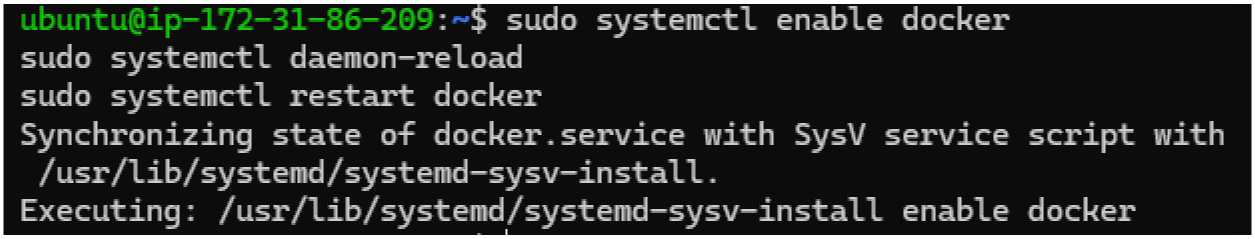
```
Running kernel seems to be up-to-date.

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-172-31-86-209:~$
```

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



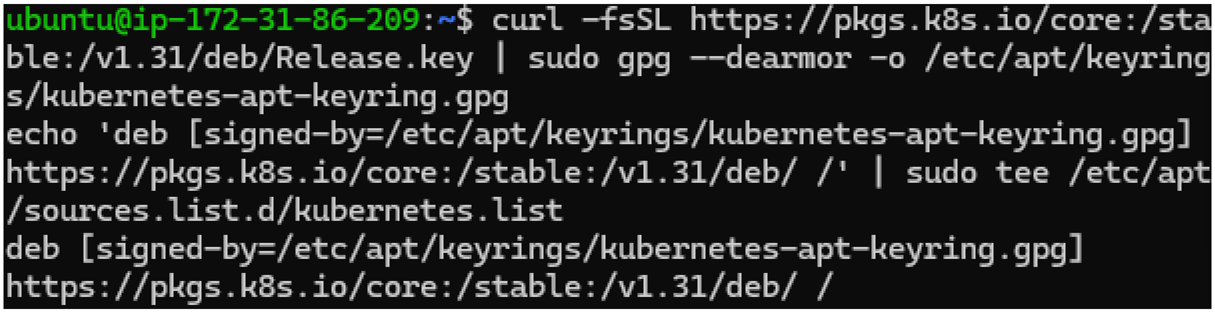
```
ubuntu@ip-172-31-86-209:~$ sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
```

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



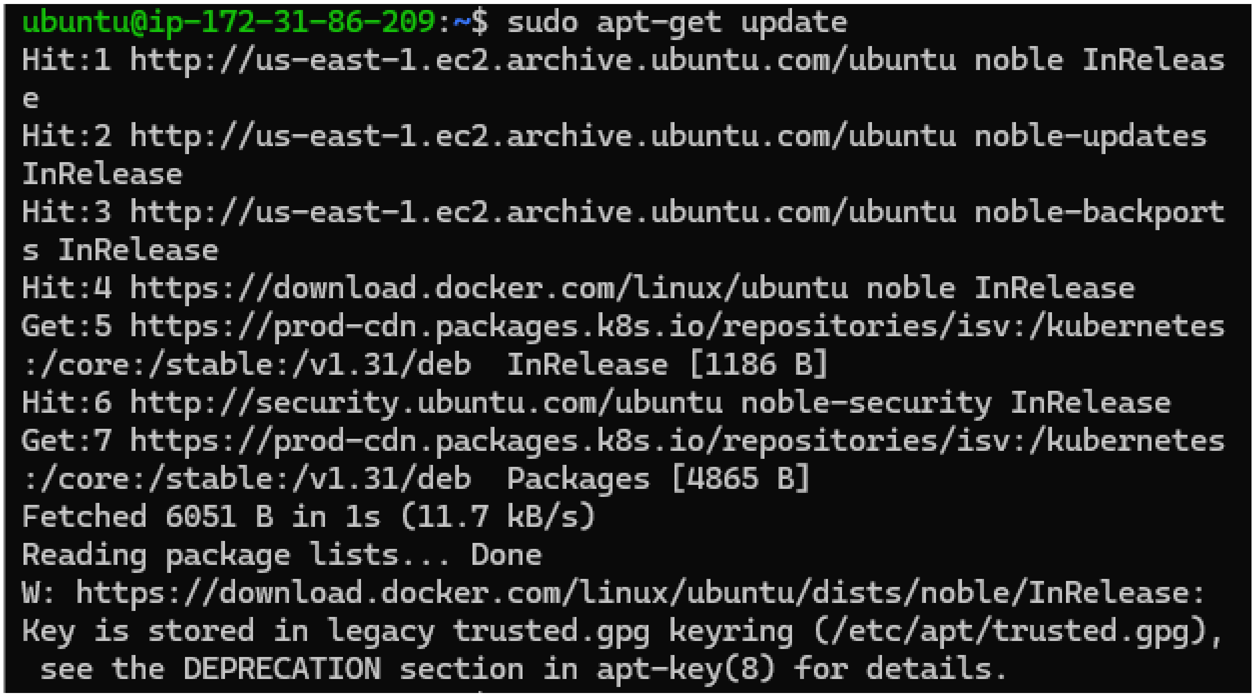
```
ubuntu@ip-172-31-86-209:~$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Synchronizing state of docker.service with SysV service script with
 /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
```

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

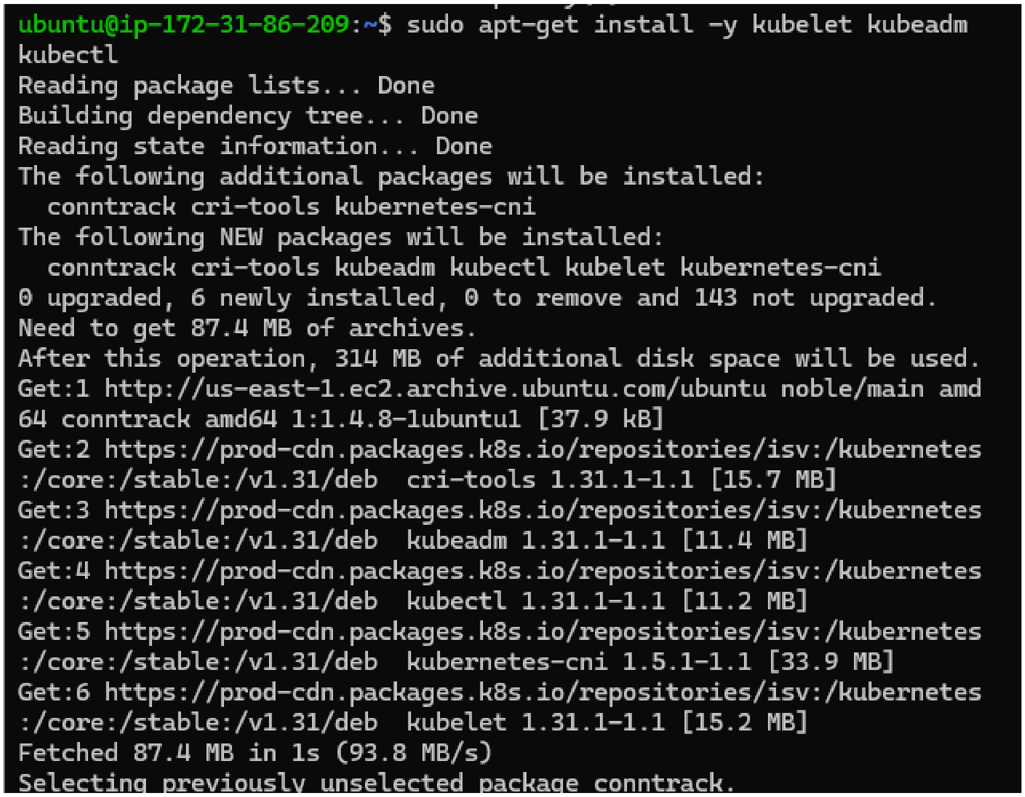


```
ubuntu@ip-172-31-86-209:~$ curl -fsSL https://pkgs.k8s.io/core:/sta
ble:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyring
s/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt
/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
```

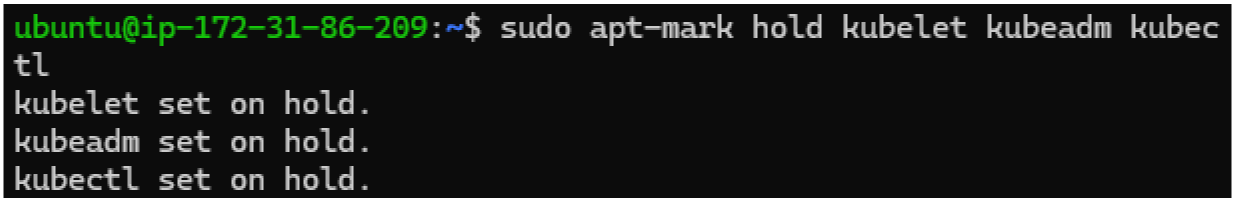
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl



```
ubuntu@ip-172-31-86-209:~$ sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InReleas
e
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-backport
s InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Get:5 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  InRelease [1186 B]
Hit:6 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  Packages [4865 B]
Fetched 6051 B in 1s (11.7 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.
```

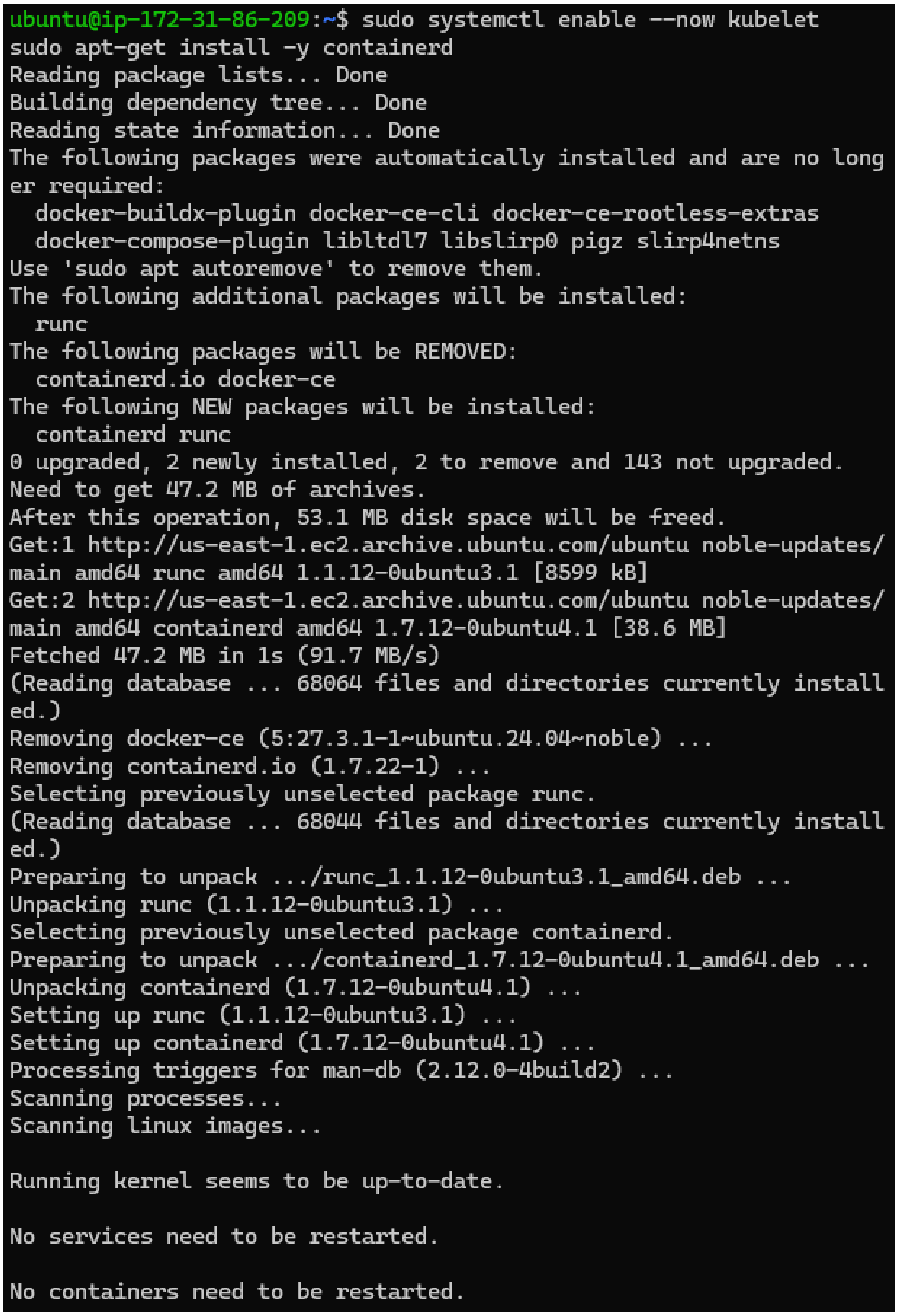


```
ubuntu@ip-172-31-86-209:~$ sudo apt-get install -y kubelet kubeadm
kubectl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
The following NEW packages will be installed:
  conntrack cri-tools kubeadm kubectl kubelet kubernetes-cni
0 upgraded, 6 newly installed, 0 to remove and 143 not upgraded.
Need to get 87.4 MB of archives.
After this operation, 314 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd
64 conntrack amd64 1:1.4.8-1ubuntu1 [37.9 kB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  cri-tools 1.31.1-1.1 [15.7 MB]
Get:3 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  kubeadm 1.31.1-1.1 [11.4 MB]
Get:4 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  kubectl 1.31.1-1.1 [11.2 MB]
Get:5 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  kubernetes-cni 1.5.1-1.1 [33.9 MB]
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes
:/core:/stable:/v1.31/deb  kubelet 1.31.1-1.1 [15.2 MB]
Fetched 87.4 MB in 1s (93.8 MB/s)
Selecting previously unselected package conntrack.
```



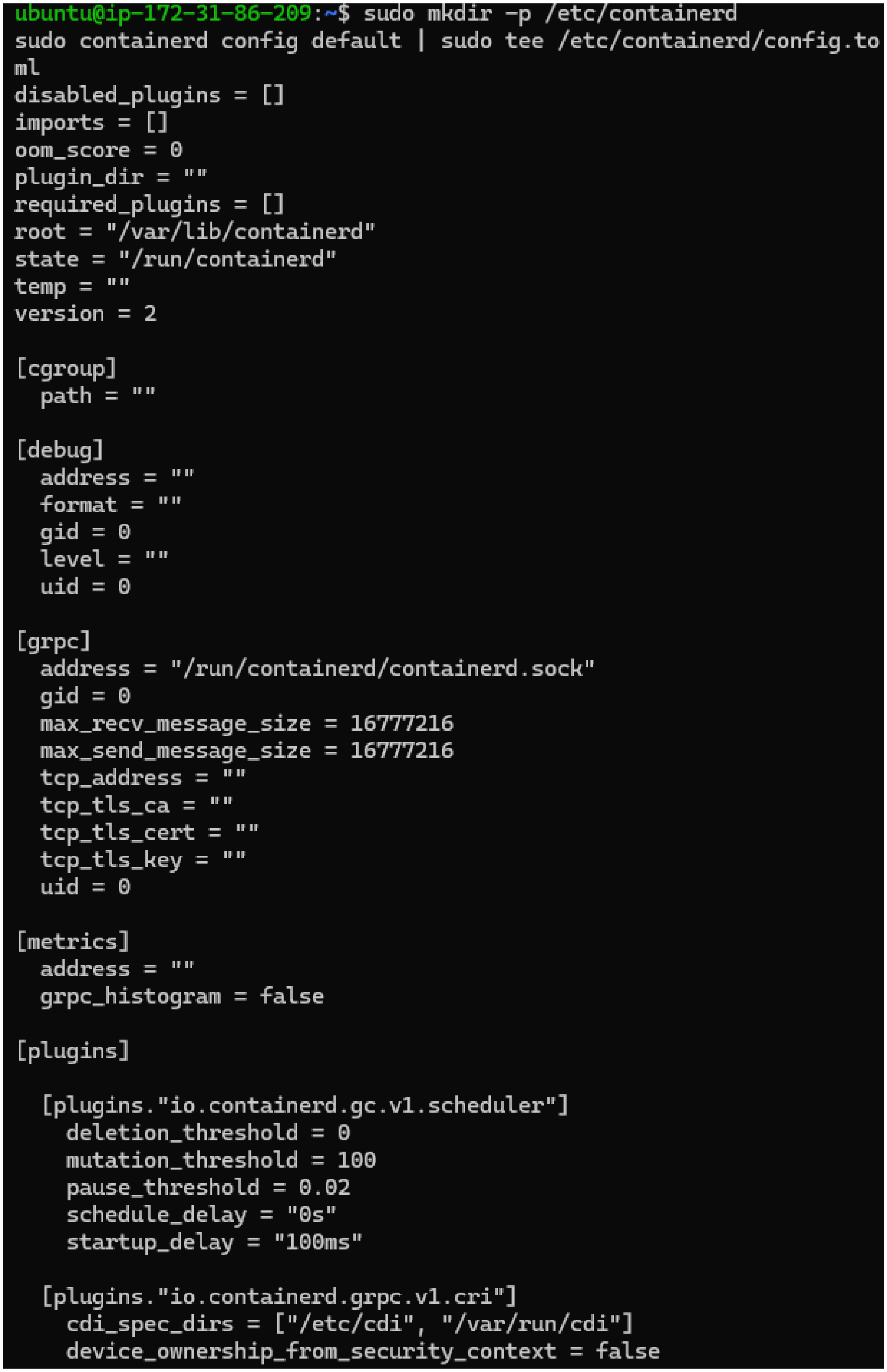
```
ubuntu@ip-172-31-86-209:~$ sudo apt-mark hold kubelet kubeadm kubec
tl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

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



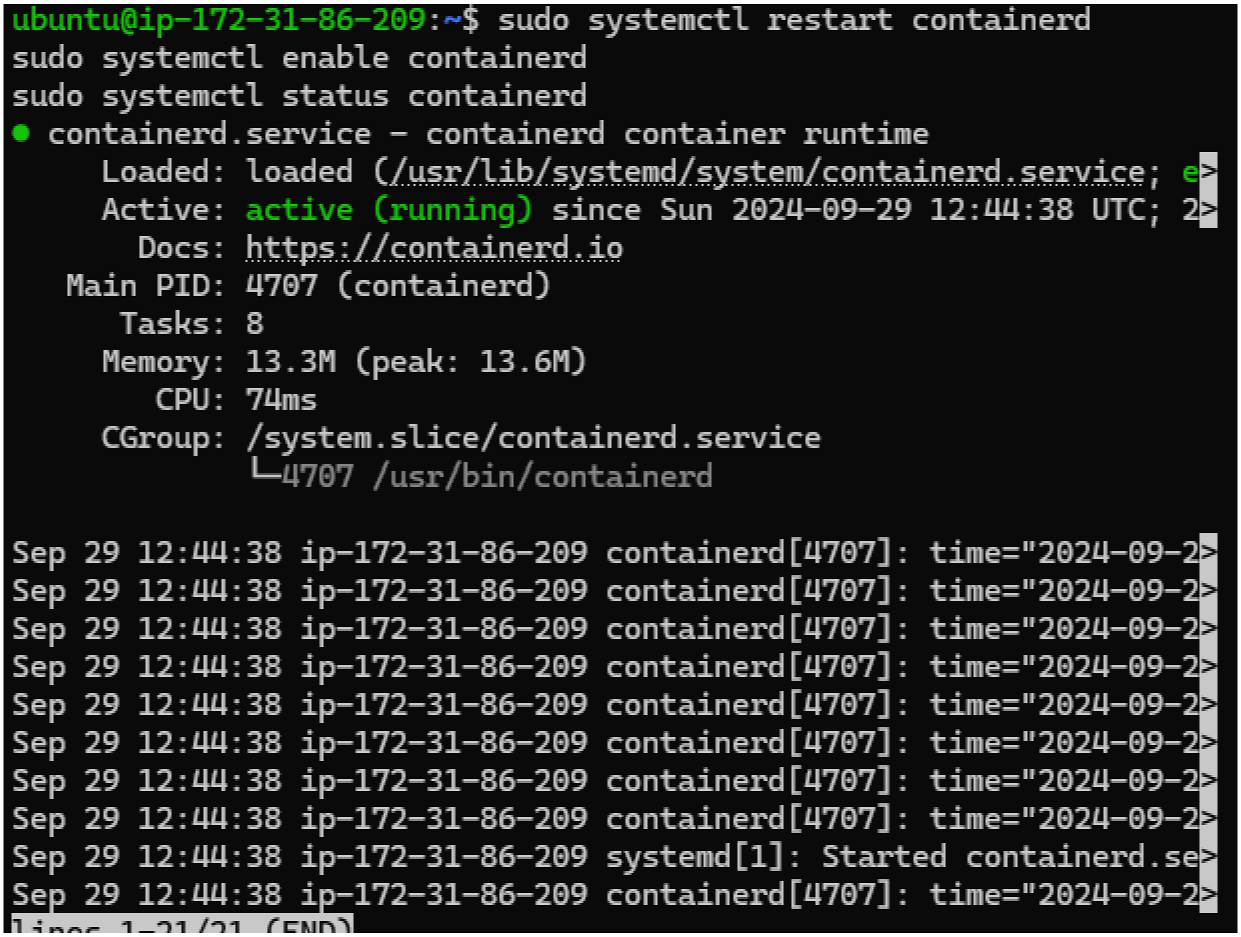
```
ubuntu@ip-172-31-86-209:~$ sudo systemctl enable --now kubelet
sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no long
er required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following 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 143 not upgraded.
Need to get 47.2 MB of archives.
After this operation, 53.1 MB disk space will be freed.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/
main amd64 runc amd64 1.1.12-0ubuntu3.1 [8599 kB]
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 (91.7 MB/s)
(Reading database ... 68064 files and directories currently install
ed.)
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 install
ed.)
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.
```

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



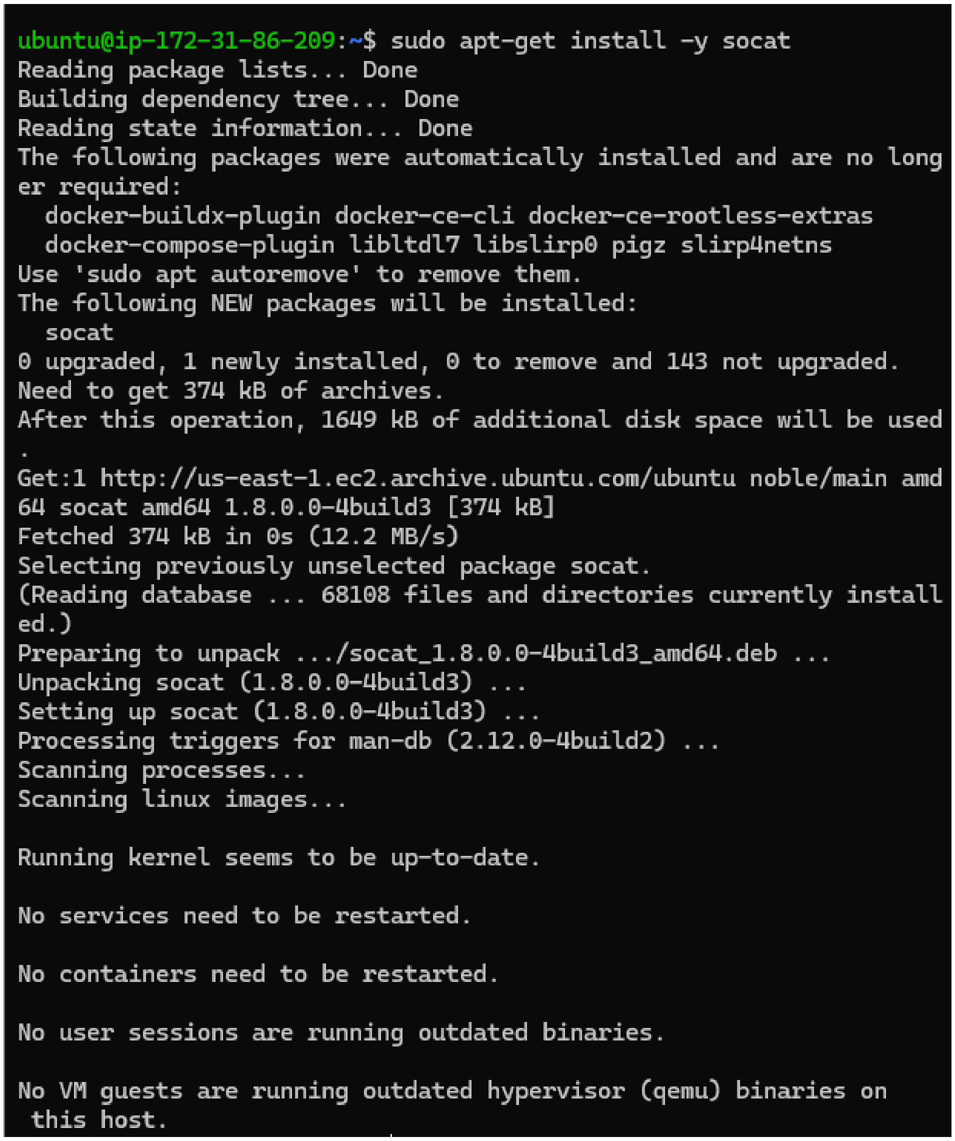
```
ubuntu@ip-172-31-86-209:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.to
ml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

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

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0

[metrics]
  address = ""
  grpc_histogram = false

[plugins]

  [plugins."io.containerd.gc.v1.scheduler"]
    deletion_threshold = 0
    mutation_threshold = 100
    pause_threshold = 0.02
    schedule_delay = "0s"
    startup_delay = "100ms"

  [plugins."io.containerd.grpc.v1.cri"]
    cdi_spec_dirs = ["/etc/cdi", "/var/run/cdi"]
    device_ownership_from_security_context = false
```

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



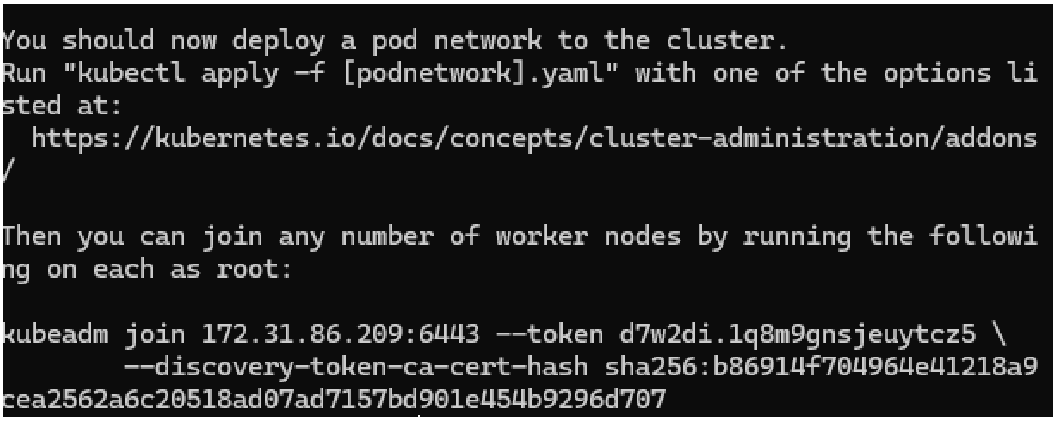
```
ubuntu@ip-172-31-86-209:~$ sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/usr/lib/systemd/system/containerd.service; e>
     Active: active (running) since Sun 2024-09-29 12:44:38 UTC; 2>
       Docs: https://containerd.io
   Main PID: 4707 (containerd)
      Tasks: 8
     Memory: 13.3M (peak: 13.6M)
        CPU: 74ms
     CGroup: /system.slice/containerd.service
             └─4707 /usr/bin/containerd

Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
Sep 29 12:44:38 ip-172-31-86-209 systemd[1]: Started containerd.se>
Sep 29 12:44:38 ip-172-31-86-209 containerd[4707]: time="2024-09-2>
lines 1-21/21 (END)
```

sudo apt-get install -y socat



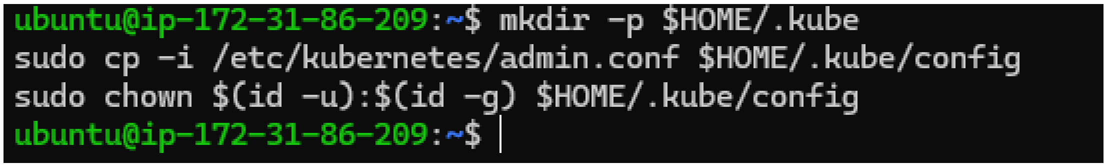
```
ubuntu@ip-172-31-86-209:~$ sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no long
er required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  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 143 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 amd
64 socat amd64 1.8.0.0-4build3 [374 kB]
Fetched 374 kB in 0s (12.2 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68108 files and directories currently install
ed.)
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.
```

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



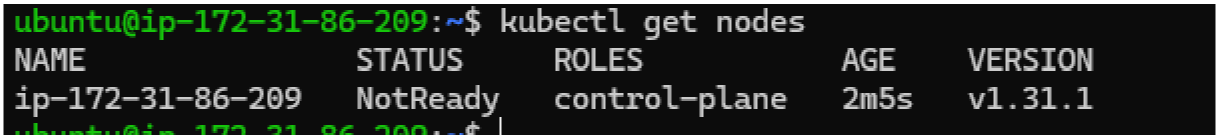
```
You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options li
sted at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons
/

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

kubeadm join 172.31.86.209:6443 --token d7w2di.1q8m9gnsjeuytcz5 \
        --discovery-token-ca-cert-hash sha256:b86914f704964e41218a9
cea2562a6c20518ad07ad7157bd901e454b9296d707
```

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-172-31-86-209:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-86-209:~$
```

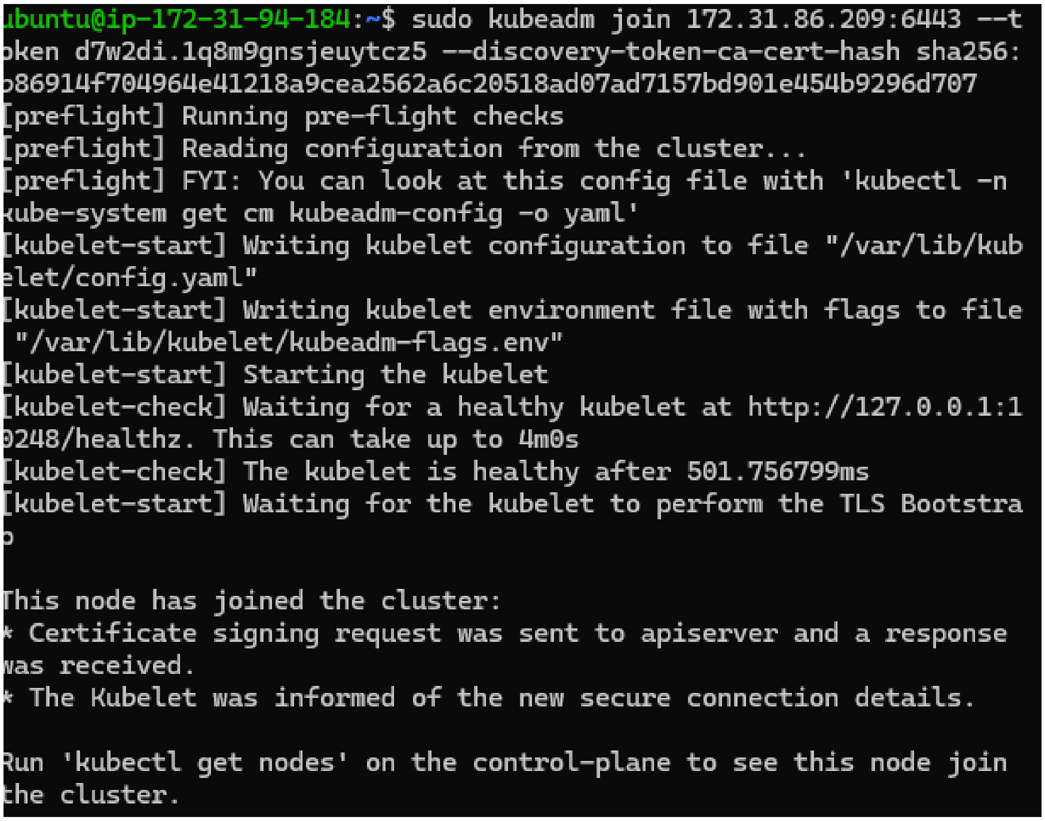
Run the command kubectl get nodes



```
ubuntu@ip-172-31-86-209:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE    VERSION
ip-172-31-86-209   NotReady   control-plane   2m5s   v1.31.1
ubuntu@ip-172-31-86-209:~$
```

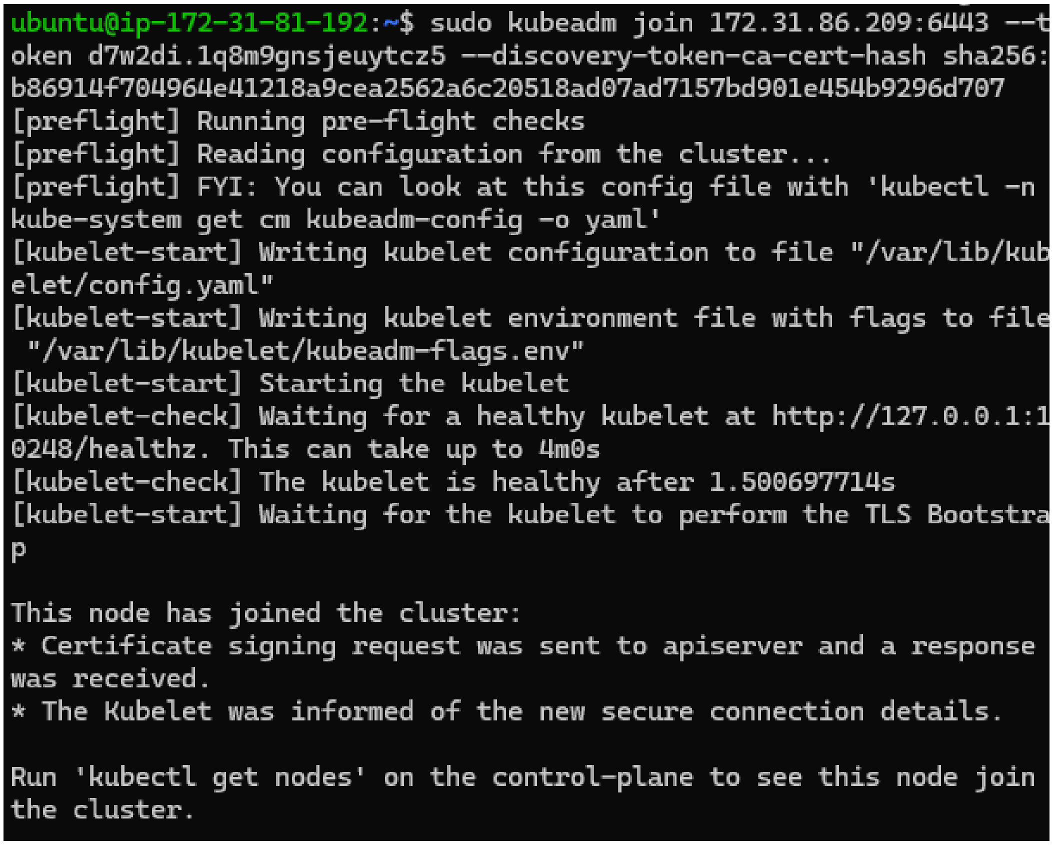
Now Run the following command on Node 1 and Node 2 to Join to master.
sudo kubeadm join 172.31.86.209:6443 --token d7w2di.1q8m9gnsjeuytcz5
--discovery-token-ca-cert-hash
sha256:b86914f704964e41218a9cea2562a6c20518ad07ad7157bd901e454b9296d707

Node 1



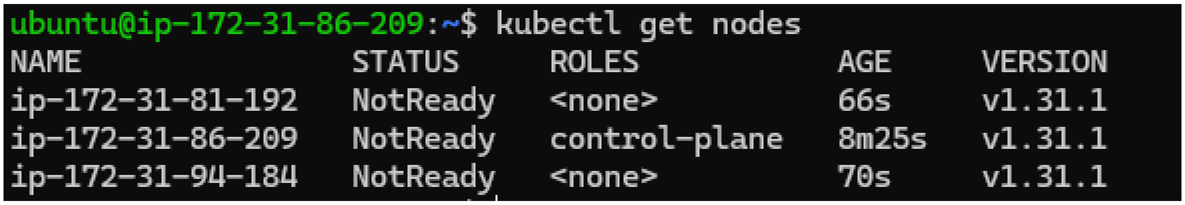
```
ubuntu@ip-172-31-94-184:~$ sudo kubeadm join 172.31.86.209:6443 --t
oken d7w2di.1q8m9gnsjeuytcz5 --discovery-token-ca-cert-hash sha256:
b86914f704964e41218a9cea2562a6c20518ad07ad7157bd901e454b9296d707
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] 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/kub
elet/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://127.0.0.1:1
0248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 501.756799ms
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstra
p

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.
```

Node 2



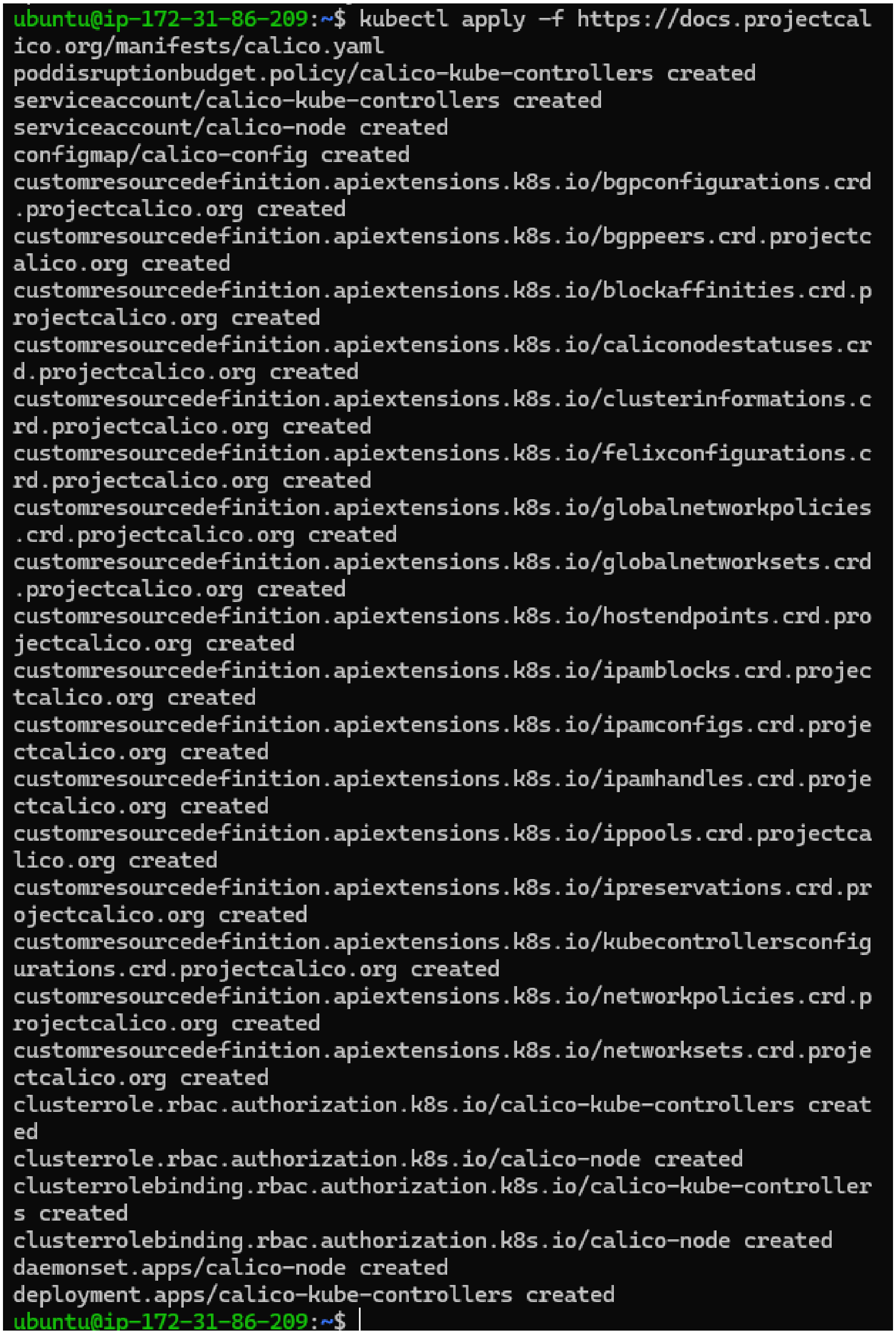
```
ubuntu@ip-172-31-81-192:~$ sudo kubeadm join 172.31.86.209:6443 --t
oken d7w2di.1q8m9gnsjeuytcz5 --discovery-token-ca-cert-hash sha256:
b86914f704964e41218a9cea2562a6c20518ad07ad7157bd901e454b9296d707
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] 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/kub
elet/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://127.0.0.1:1
0248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.500697714s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstra
p

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.
```

Kubectl get nodes



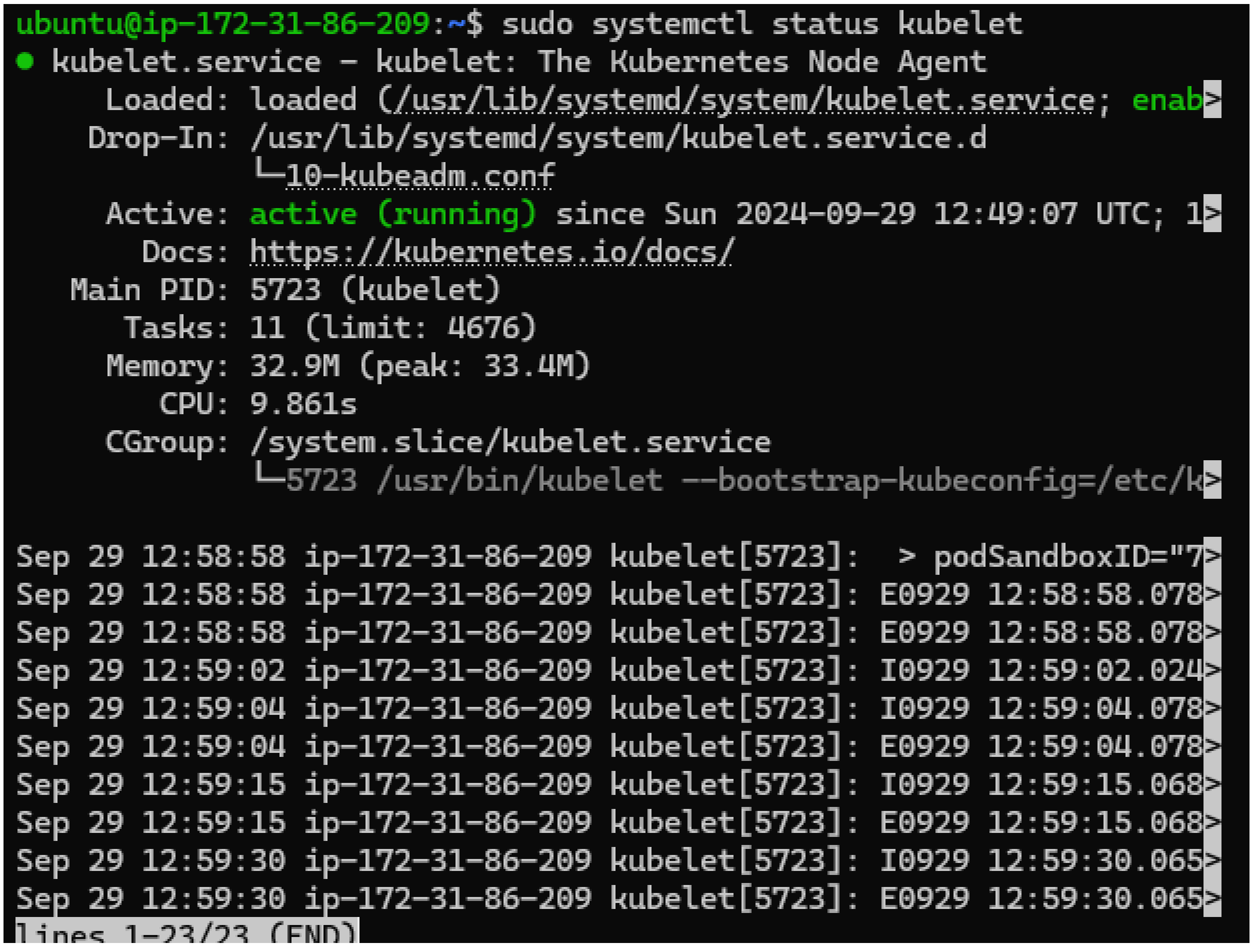
```
ubuntu@ip-172-31-86-209:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-81-192   NotReady   <none>          66s     v1.31.1
ip-172-31-86-209   NotReady   control-plane   8m25s   v1.31.1
ip-172-31-94-184   NotReady   <none>          70s     v1.31.1
```

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



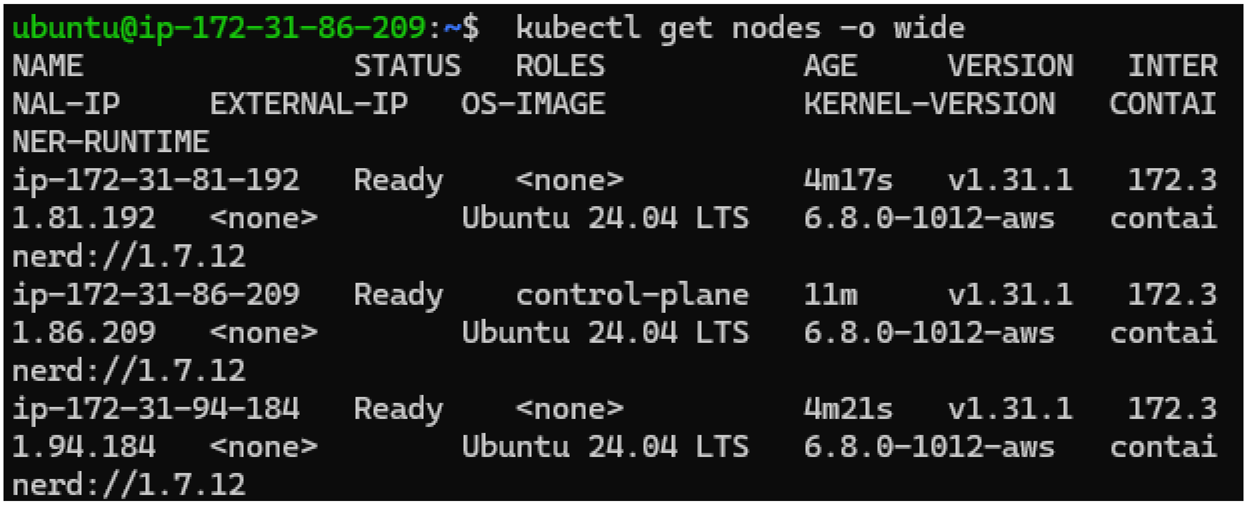
```
ubuntu@ip-172-31-86-209:~$ kubectl apply -f https://docs.projectcal
ico.org/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd
.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectc
alico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.p
rojectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.cr
d.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.c
rd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.c
rd.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.pro
jectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projec
tcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.proje
ctcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.proje
ctcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectca
lico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.pr
ojectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfig
urations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.p
rojectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.proje
ctcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers creat
ed
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controller
s created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
ubuntu@ip-172-31-86-209:~$
```

sudo systemctl status kubelet



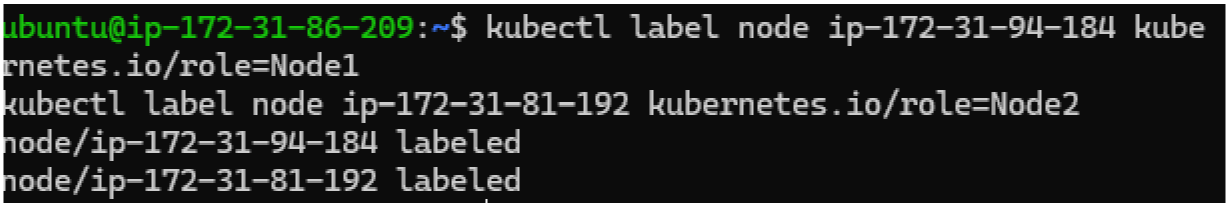
```
ubuntu@ip-172-31-86-209:~$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enab>
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Sun 2024-09-29 12:49:07 UTC; 1>
       Docs: https://kubernetes.io/docs/
   Main PID: 5723 (kubelet)
      Tasks: 11 (limit: 4676)
     Memory: 32.9M (peak: 33.4M)
        CPU: 9.861s
     CGroup: /system.slice/kubelet.service
             └─5723 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/k>

Sep 29 12:58:58 ip-172-31-86-209 kubelet[5723]:  > podSandboxID="7>
Sep 29 12:58:58 ip-172-31-86-209 kubelet[5723]: E0929 12:58:58.078>
Sep 29 12:58:58 ip-172-31-86-209 kubelet[5723]: E0929 12:58:58.078>
Sep 29 12:59:02 ip-172-31-86-209 kubelet[5723]: I0929 12:59:02.024>
Sep 29 12:59:04 ip-172-31-86-209 kubelet[5723]: I0929 12:59:04.078>
Sep 29 12:59:04 ip-172-31-86-209 kubelet[5723]: E0929 12:59:04.078>
Sep 29 12:59:15 ip-172-31-86-209 kubelet[5723]: I0929 12:59:15.068>
Sep 29 12:59:15 ip-172-31-86-209 kubelet[5723]: E0929 12:59:15.068>
Sep 29 12:59:30 ip-172-31-86-209 kubelet[5723]: I0929 12:59:30.065>
Sep 29 12:59:30 ip-172-31-86-209 kubelet[5723]: E0929 12:59:30.065>
lines 1-23/23 (END)
```

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



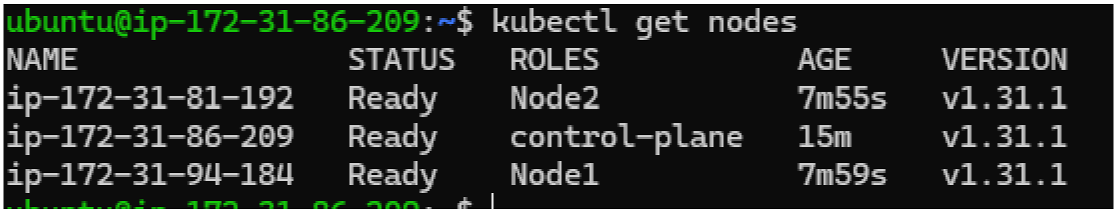
```
ubuntu@ip-172-31-86-209:~$  kubectl get nodes -o wide
NAME                STATUS   ROLES           AGE     VERSION   INTER
NAL-IP      EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAI
NER-RUNTIME
ip-172-31-81-192    Ready    <none>          4m17s   v1.31.1   172.3
1.81.192    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   contai
nerd://1.7.12
ip-172-31-86-209    Ready    control-plane   11m     v1.31.1   172.3
1.86.209    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   contai
nerd://1.7.12
ip-172-31-94-184    Ready    <none>          4m21s   v1.31.1   172.3
1.94.184    <none>        Ubuntu 24.04 LTS   6.8.0-1012-aws   contai
nerd://1.7.12
```

Now to Rename run this command
Rename to Node 1: kubectl label node ip-172-31-94-184 kubernetes.io/role=Node1
Rename to Node 2: kubectl label node ip-172-31-81-192 kubernetes.io/role=Node2



```
ubuntu@ip-172-31-86-209:~$ kubectl label node ip-172-31-94-184 kube
rnetes.io/role=Node1
kubectl label node ip-172-31-81-192 kubernetes.io/role=Node2
node/ip-172-31-94-184 labeled
node/ip-172-31-81-192 labeled
```

run kubectl get nodes



```
ubuntu@ip-172-31-86-209:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-81-192   Ready    Node2           7m55s   v1.31.1
ip-172-31-86-209   Ready    control-plane   15m     v1.31.1
ip-172-31-94-184   Ready    Node1           7m59s   v1.31.1
ubuntu@ip-172-31-86-209:~$
```

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.