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

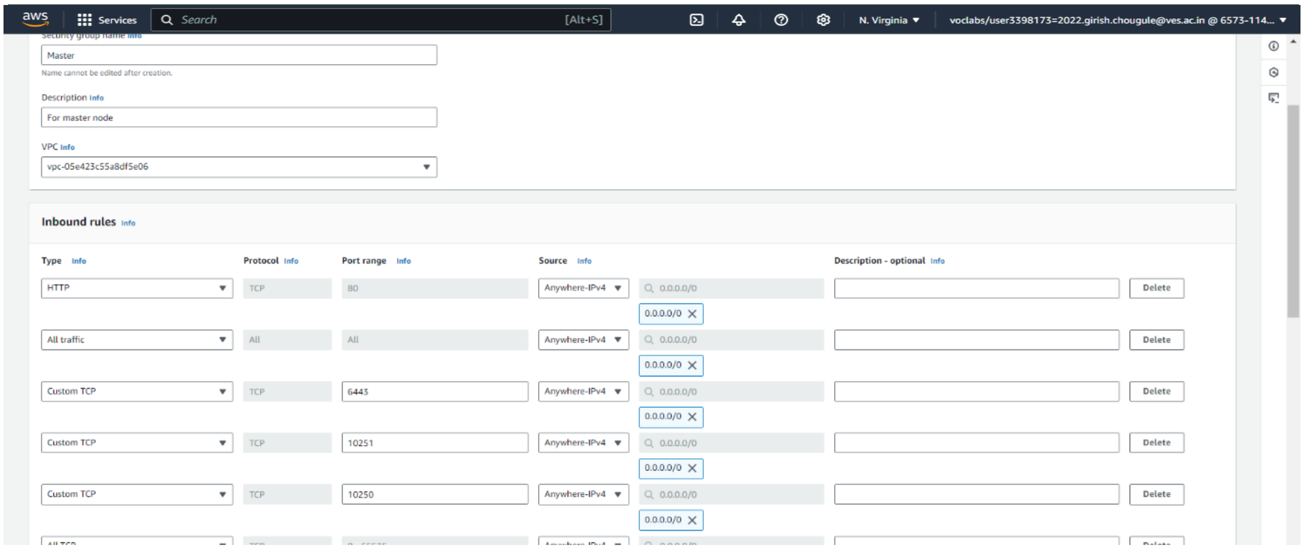
Note:

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

Prerequisites :

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

Master:


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3398173=2022.girish.chougule@ves.ac.in @ 6573-114...

Security group name Info

Master

Name cannot be edited after creation.

Description Info

For master node

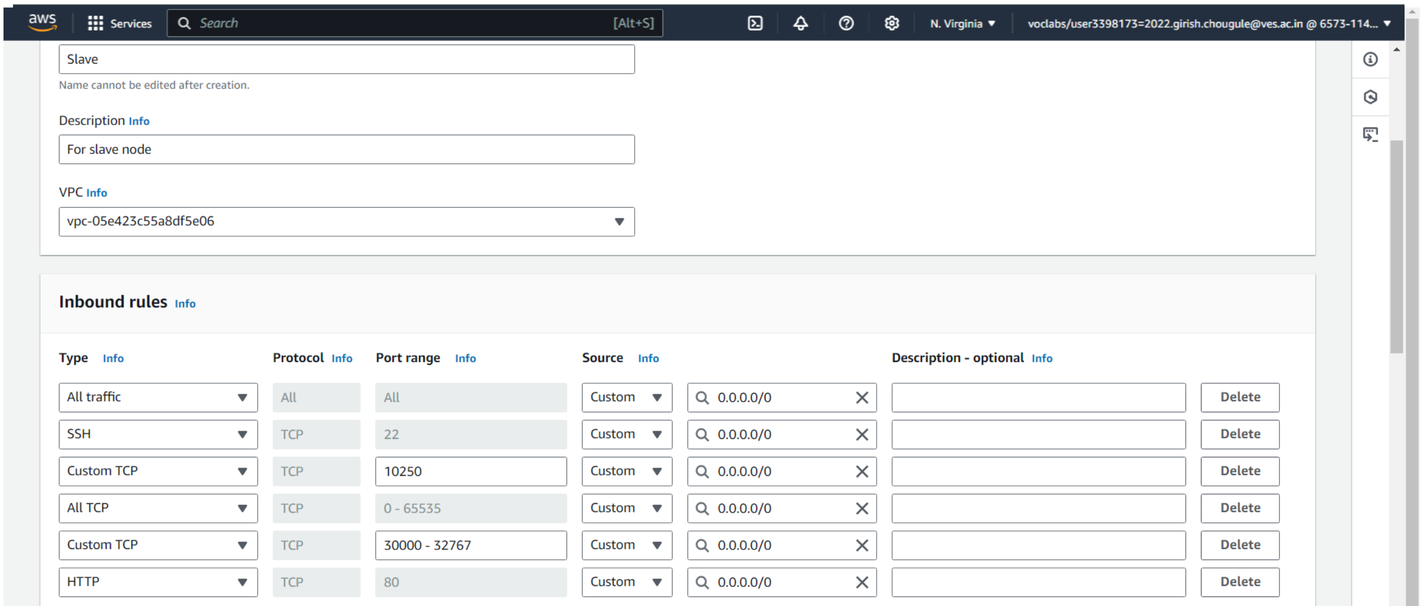
VPC Info

vpc-05e423c55a8df5e06

Inbound rules Info

Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
HTTP	TCP	80	Anywhere-IPv4	0.0.0.0/0 0.0.0.0/0		Delete
All traffic	All	All	Anywhere-IPv4	0.0.0.0/0 0.0.0.0/0		Delete
Custom TCP	TCP	6443	Anywhere-IPv4	0.0.0.0/0 0.0.0.0/0		Delete
Custom TCP	TCP	10251	Anywhere-IPv4	0.0.0.0/0 0.0.0.0/0		Delete
Custom TCP	TCP	10250	Anywhere-IPv4	0.0.0.0/0 0.0.0.0/0		Delete
All TCP	TCP	0 - 65535	Anywhere-IPv4	0.0.0.0/0		Delete

Node :


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3398173=2022.girish.chougule@ves.ac.in @ 6573-114...

Slave

Name cannot be edited after creation.

Description Info

For slave node

VPC Info

vpc-05e423c55a8df5e06

Inbound rules Info

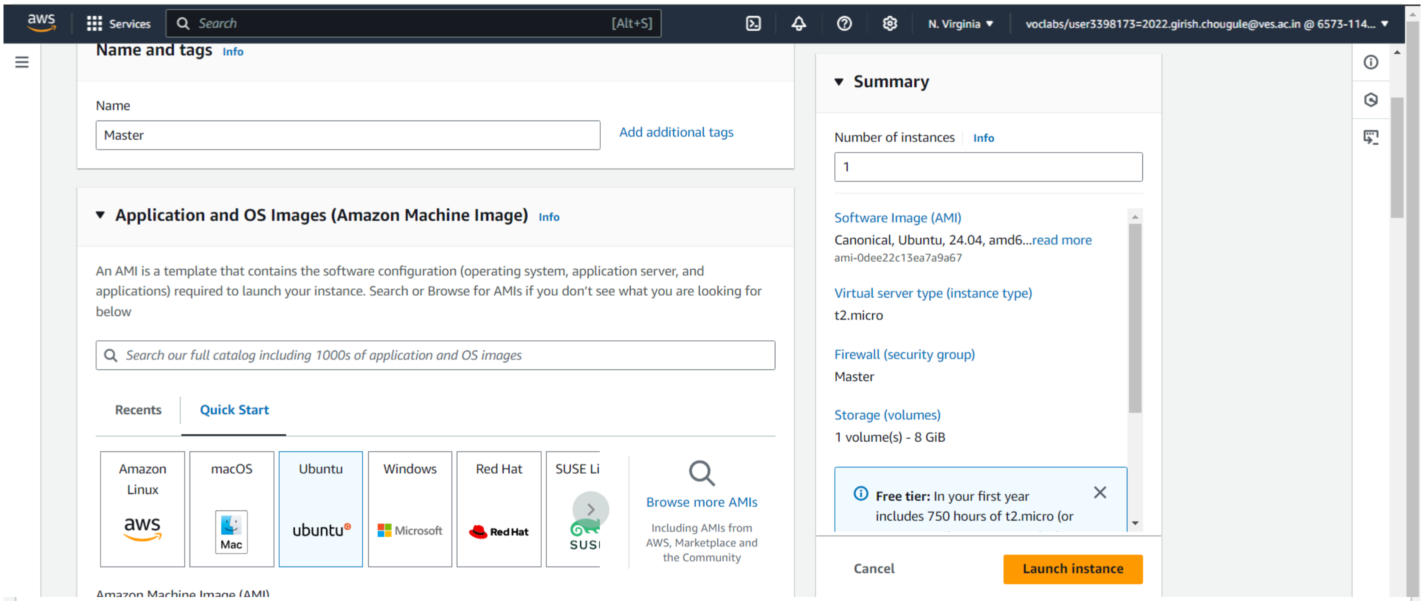
Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
All traffic	All	All	Custom	0.0.0.0/0		Delete
SSH	TCP	22	Custom	0.0.0.0/0		Delete
Custom TCP	TCP	10250	Custom	0.0.0.0/0		Delete
All TCP	TCP	0 - 65535	Custom	0.0.0.0/0		Delete
Custom TCP	TCP	30000 - 32767	Custom	0.0.0.0/0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

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:


aws
Services
Search
[Alt+S]
N. Virginia ▾
voclabs/user3398173=2022.girish.chougule@ves.ac.in @ 6573-114... ▾

Name and tags Info

Name

Master

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

Recents | Quick Start

Amazon Machine Image (AMI)

▾ Summary

Number of instances Info

1

Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-0dee22c13ea7a9a67

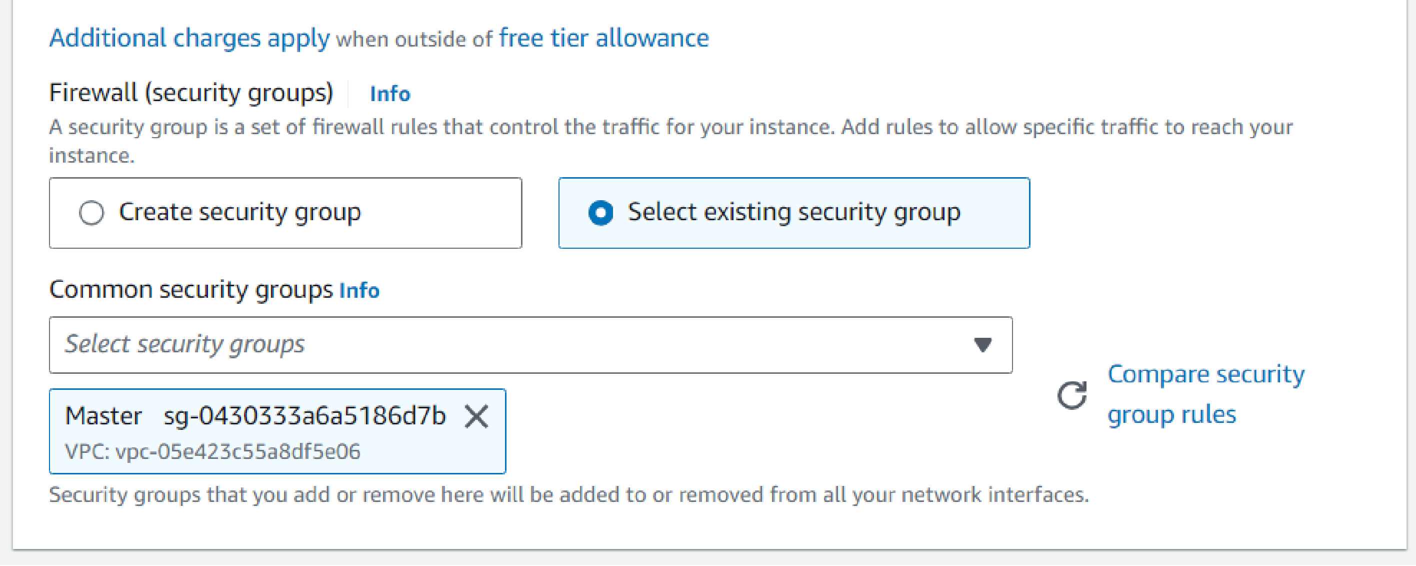
Virtual server type (instance type)
t2.micro

Firewall (security group)
Master

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or

Cancel | Launch instance



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 ▾

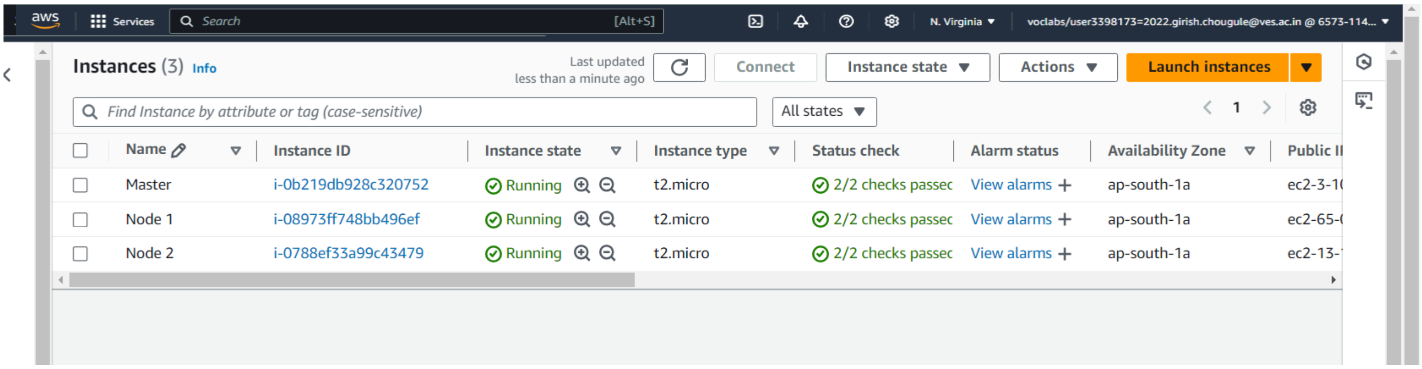
Master sg-0430333a6a5186d7b ✕
VPC: vpc-05e423c55a8df5e06

Compare security group rules

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

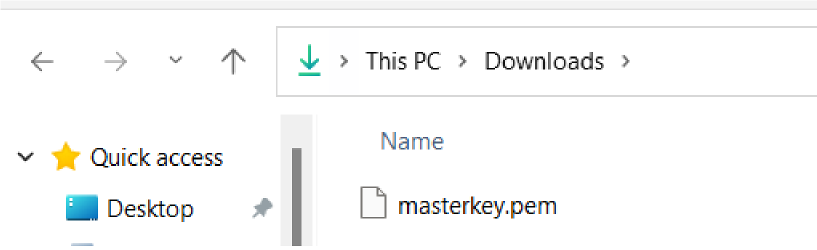
Do Same for 2 Nodes and use security groups of Node for that.

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


Services
Search [Alt+S]
N. Virginia
voclabs/user3398173=2022.girish.chougule@ves.ac.in @ 6573-114...
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

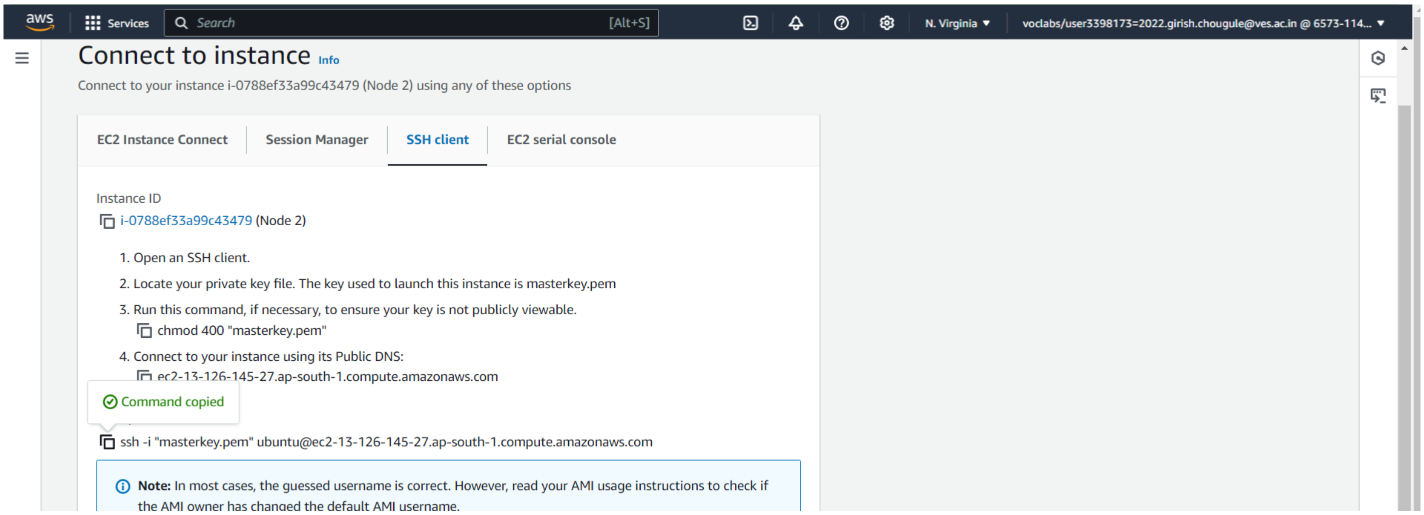
☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public I
☐	Master	i-0b219db928c320752	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	ap-south-1a	ec2-3-1(
☐	Node 1	i-08973ff748bb496ef	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	ap-south-1a	ec2-65-(
☐	Node 2	i-0788ef33a99c43479	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	ap-south-1a	ec2-13-

(Downloaded Key)



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

Master:



Connect to instance Info

Connect to your instance i-0788ef33a99c43479 (Node 2) using any of these options

Instance ID

i-0788ef33a99c43479 (Node 2)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is masterkey.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.

```
chmod 400 "masterkey.pem"
```

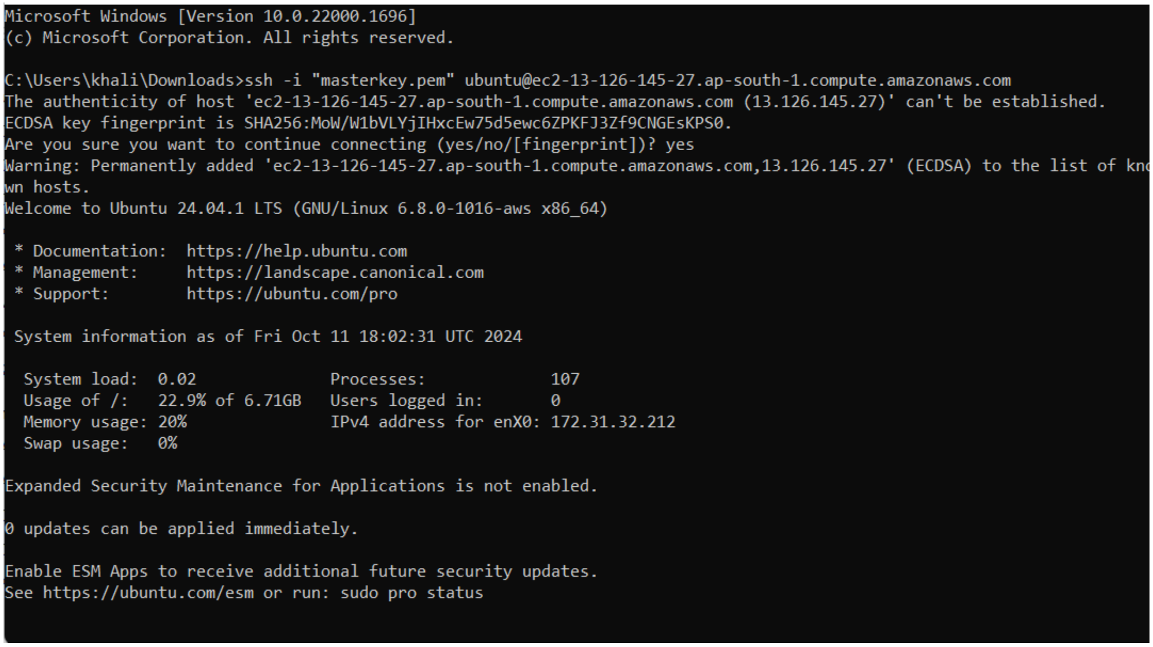
4. Connect to your instance using its Public DNS:

```
ec2-13-126-145-27.ap-south-1.compute.amazonaws.com
```

```
ssh -i "masterkey.pem" ubuntu@ec2-13-126-145-27.ap-south-1.compute.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.

Successful Connection:



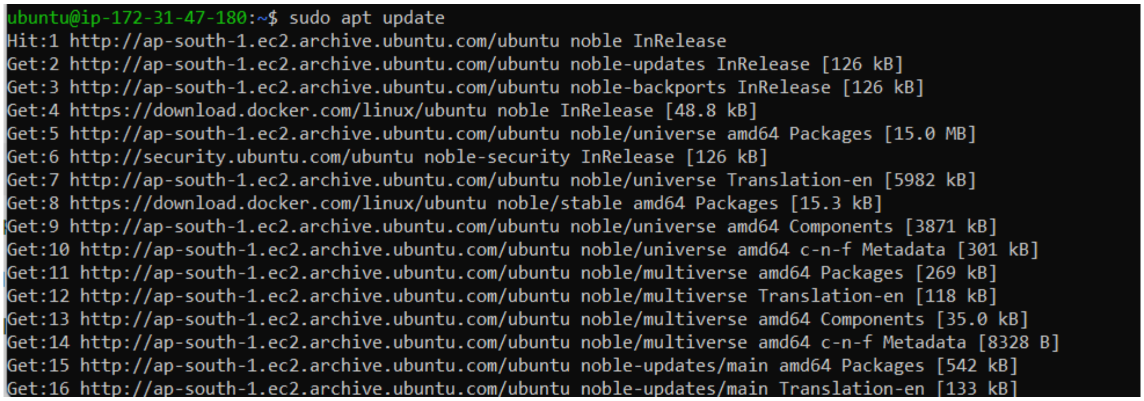
```
Microsoft Windows [Version 10.0.22000.1696]
(c) Microsoft Corporation. All rights reserved.

C:\Users\khali\Downloads>ssh -i "masterkey.pem" ubuntu@ec2-13-126-145-27.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-126-145-27.ap-south-1.compute.amazonaws.com (13.126.145.27)' can't be established.
ECDSA key fingerprint is SHA256:MoW/W1bVLYjIHxcEw75d5ewc6ZPKFJ3Zf9CNGEsKPS0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-126-145-27.ap-south-1.compute.amazonaws.com,13.126.145.27' (ECDSA) to the list of kno
wn hosts.
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1016-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Oct 11 18:02:31 UTC 2024

  System load:  0.02              Processes:             107
  Usage of /:   22.9% of 6.71GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for enX0: 172.31.32.212
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
```

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

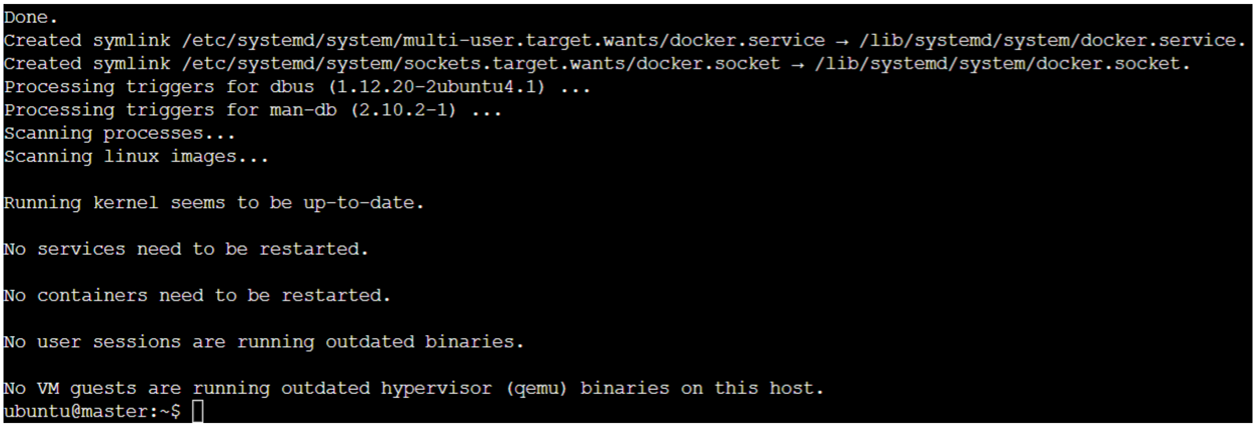
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt-get update



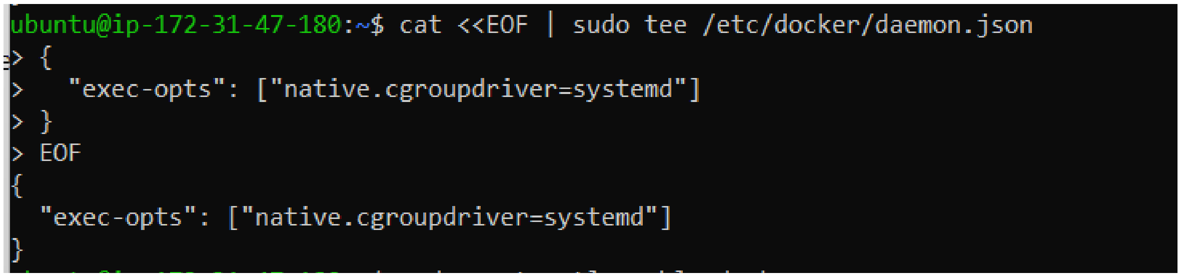
```
ubuntu@ip-172-31-47-180:~$ sudo apt update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [15.3 kB]
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:10 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:11 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:12 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:13 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:14 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:15 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [542 kB]
Get:16 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [133 kB]
```

sudo apt-get install -y docker-c



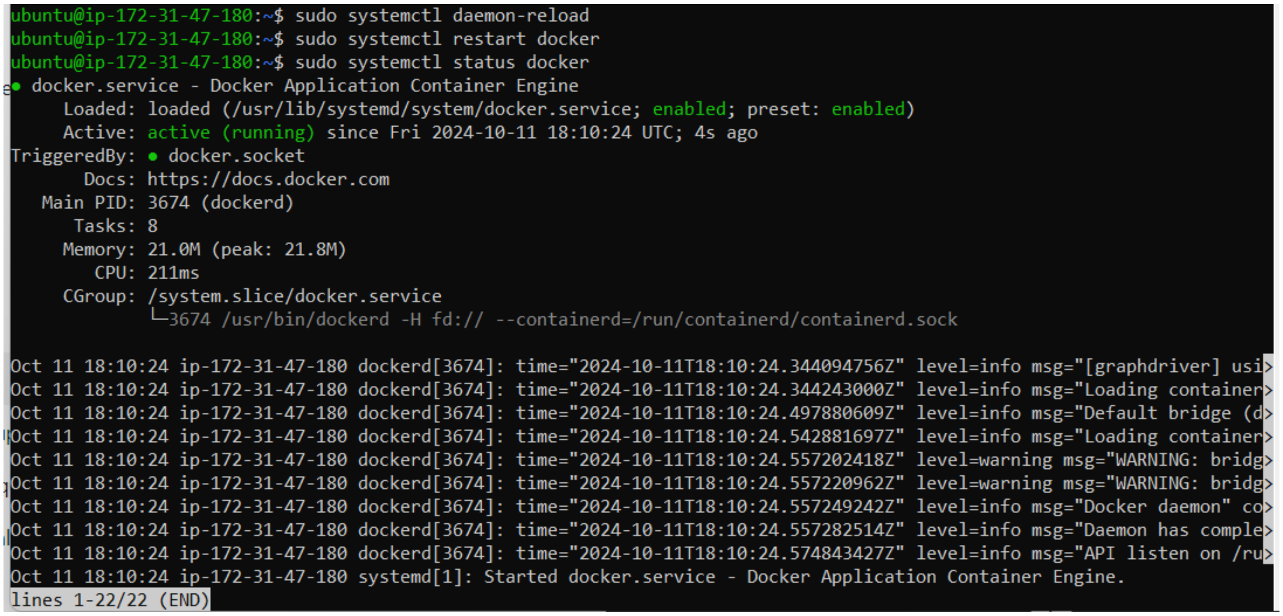
```
Done.
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
Processing triggers for man-db (2.10.2-1) ...
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@master:~$ 
```

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



```
ubuntu@ip-172-31-47-180:~$ 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
```

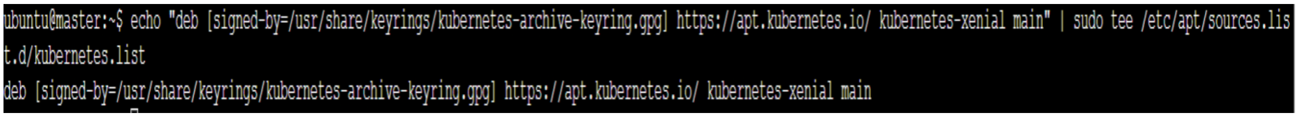


```
ubuntu@ip-172-31-47-180:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-47-180:~$ sudo systemctl restart docker
ubuntu@ip-172-31-47-180:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Fri 2024-10-11 18:10:24 UTC; 4s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 3674 (dockerd)
      Tasks: 8
     Memory: 21.0M (peak: 21.8M)
        CPU: 211ms
     CGroup: /system.slice/docker.service
             └─3674 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.344094756Z" level=info msg="[graphdriver] usi>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.344243000Z" level=info msg="Loading container>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.497880609Z" level=info msg="Default bridge (d>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.542881697Z" level=info msg="Loading container>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.557202418Z" level=warning msg="WARNING: bridg>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.557220962Z" level=warning msg="WARNING: bridg>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.557249242Z" level=info msg="Docker daemon" co>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.557282514Z" level=info msg="Daemon has comple>
Oct 11 18:10:24 ip-172-31-47-180 dockerd[3674]: time="2024-10-11T18:10:24.574843427Z" level=info msg="API listen on /ru>
Oct 11 18:10:24 ip-172-31-47-180 systemd[1]: Started docker.service - Docker Application Container Engine.
lines 1-22/22 (END)
```

Step 5: Run the below command to install Kubernets.

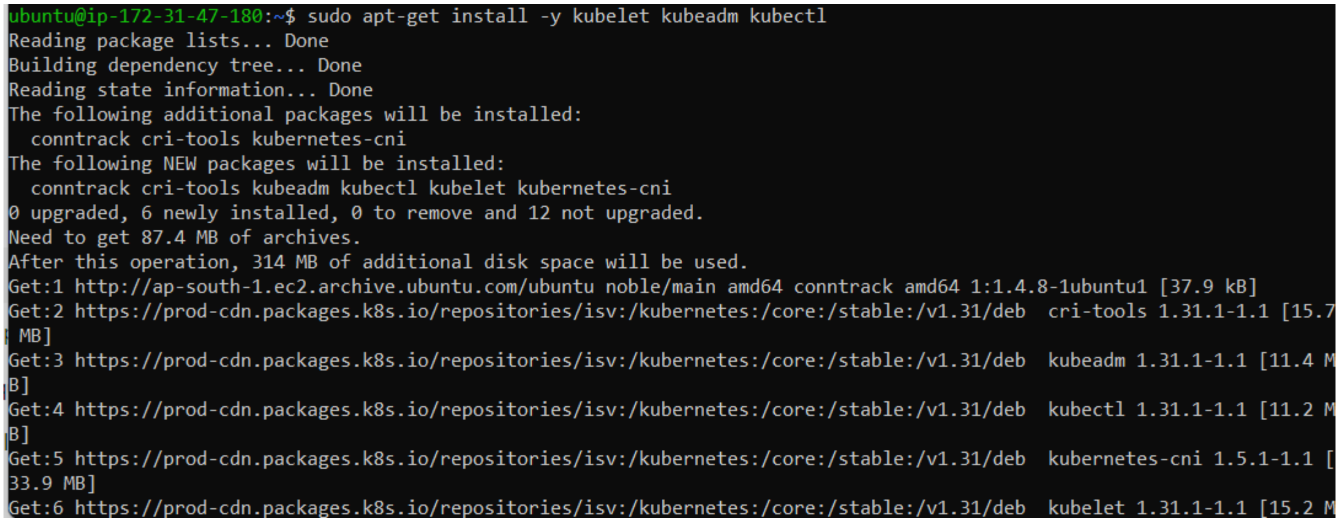
```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

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



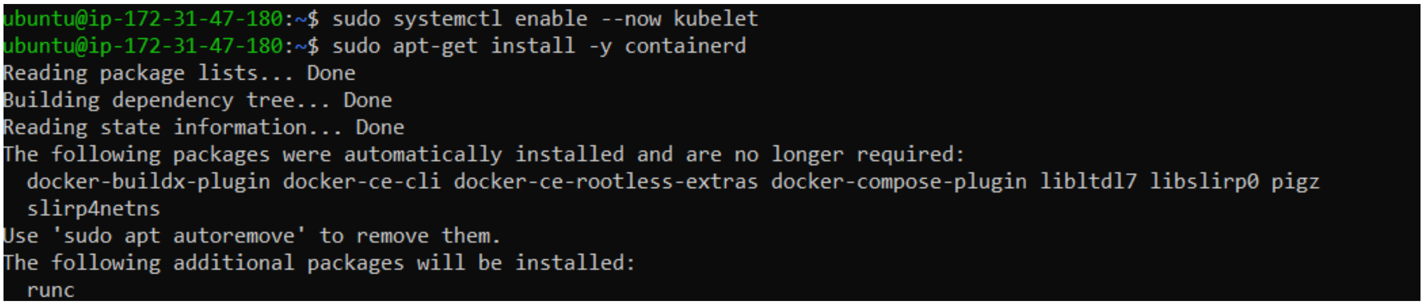
```
ubuntu@master:~$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.lis
t.d/kubernetes.list
deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main
```

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



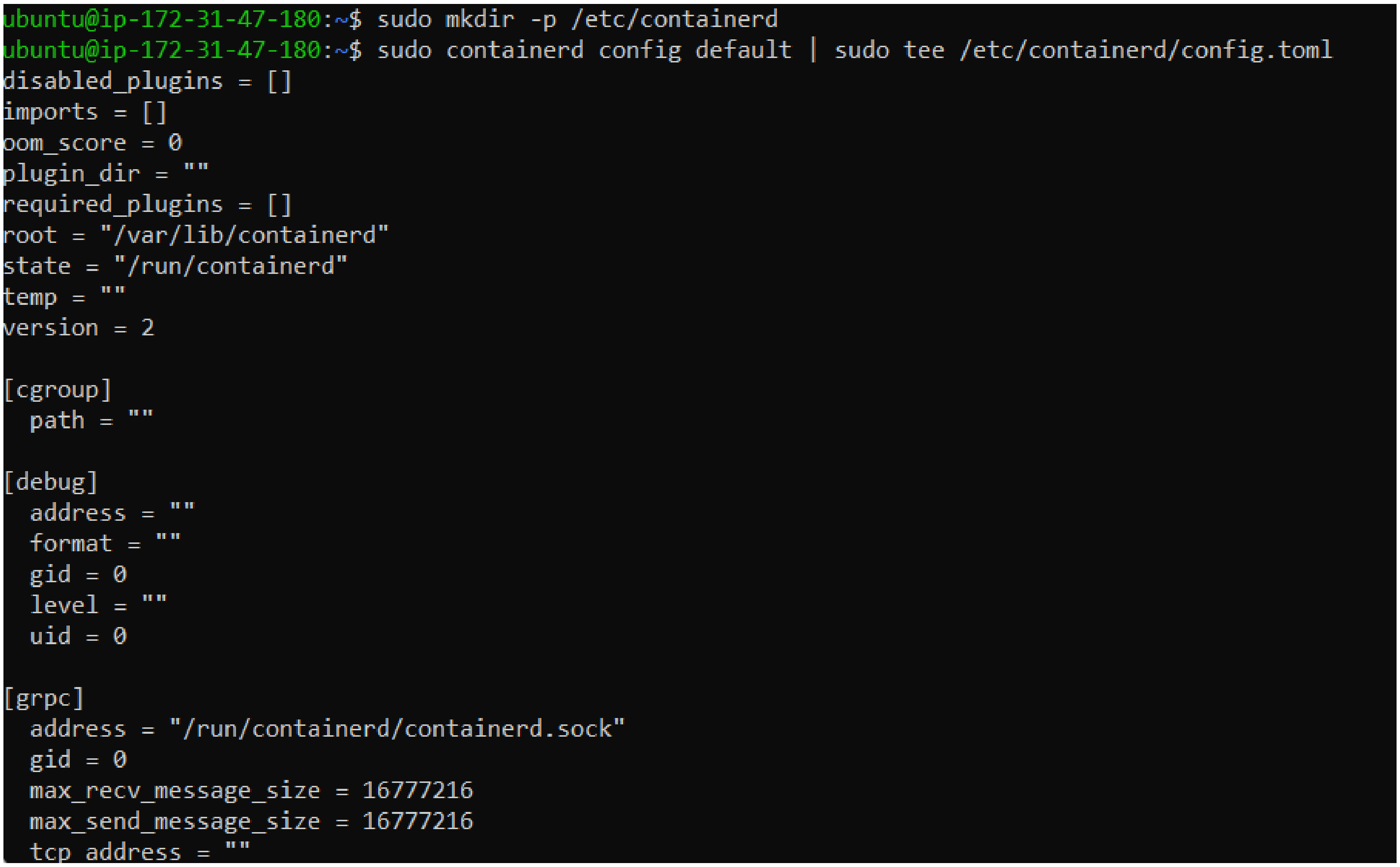
```
ubuntu@ip-172-31-47-180:~$ sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
The following NEW packages will be installed:
  conntrack cri-tools kubeadm kubectl kubelet kubernetes-cni
0 upgraded, 6 newly installed, 0 to remove and 12 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://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 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 M
B]
Get:4 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  kubectl 1.31.1-1.1 [11.2 M
B]
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 M
```

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



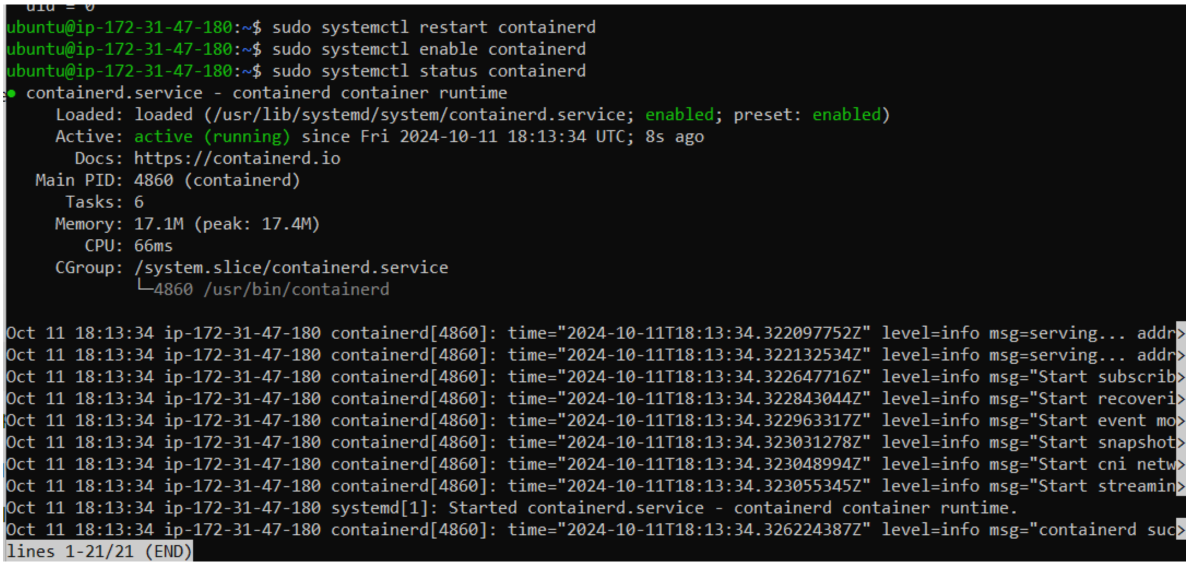
```
ubuntu@ip-172-31-47-180:~$ sudo systemctl enable --now kubelet
ubuntu@ip-172-31-47-180:~$ sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz
  slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  runc
```

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



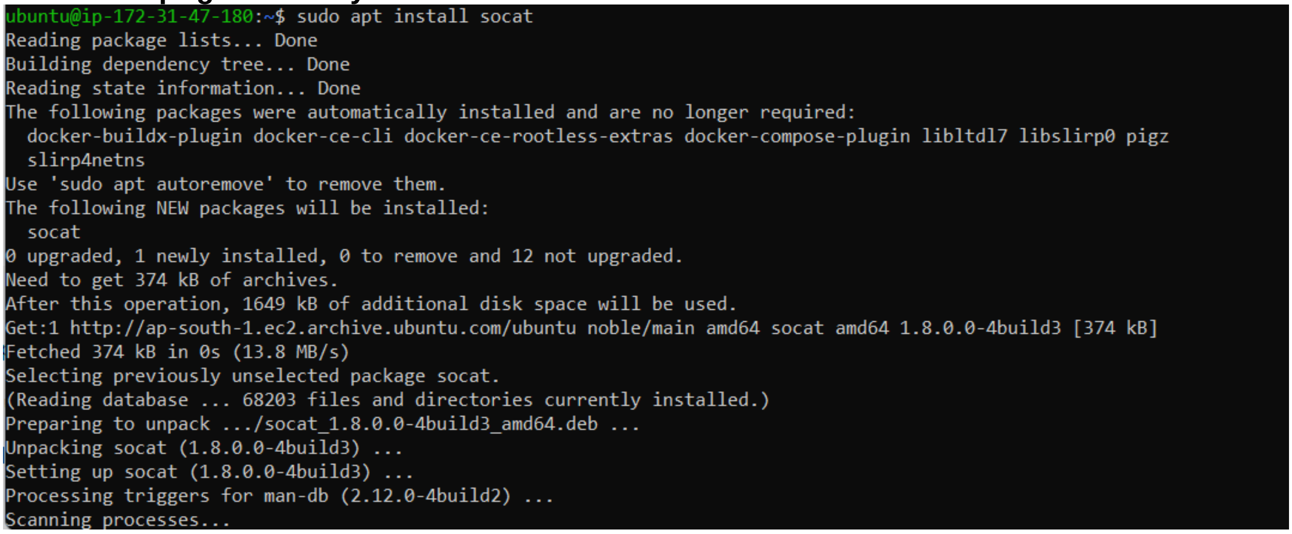
```
ubuntu@ip-172-31-47-180:~$ sudo mkdir -p /etc/containerd
ubuntu@ip-172-31-47-180:~$ sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

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

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
```

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



```
  uid = 0
ubuntu@ip-172-31-47-180:~$ sudo systemctl restart containerd
ubuntu@ip-172-31-47-180:~$ sudo systemctl enable containerd
ubuntu@ip-172-31-47-180:~$ sudo systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; preset: enabled)
     Active: active (running) since Fri 2024-10-11 18:13:34 UTC; 8s ago
       Docs: https://containerd.io
   Main PID: 4860 (containerd)
      Tasks: 6
     Memory: 17.1M (peak: 17.4M)
        CPU: 66ms
     CGroup: /system.slice/containerd.service
             └─4860 /usr/bin/containerd

Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.322097752Z" level=info msg=serving... addr>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.322132534Z" level=info msg=serving... addr>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.322647716Z" level=info msg="Start subscrib>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.322843044Z" level=info msg="Start recoveri>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.322963317Z" level=info msg="Start event mo>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.323031278Z" level=info msg="Start snapshot>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.323048994Z" level=info msg="Start cni netw>
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.323055345Z" level=info msg="Start streamin>
Oct 11 18:13:34 ip-172-31-47-180 systemd[1]: Started containerd.service - containerd container runtime.
Oct 11 18:13:34 ip-172-31-47-180 containerd[4860]: time="2024-10-11T18:13:34.326224387Z" level=info msg="containerd suc>
lines 1-21/21 (END)
```

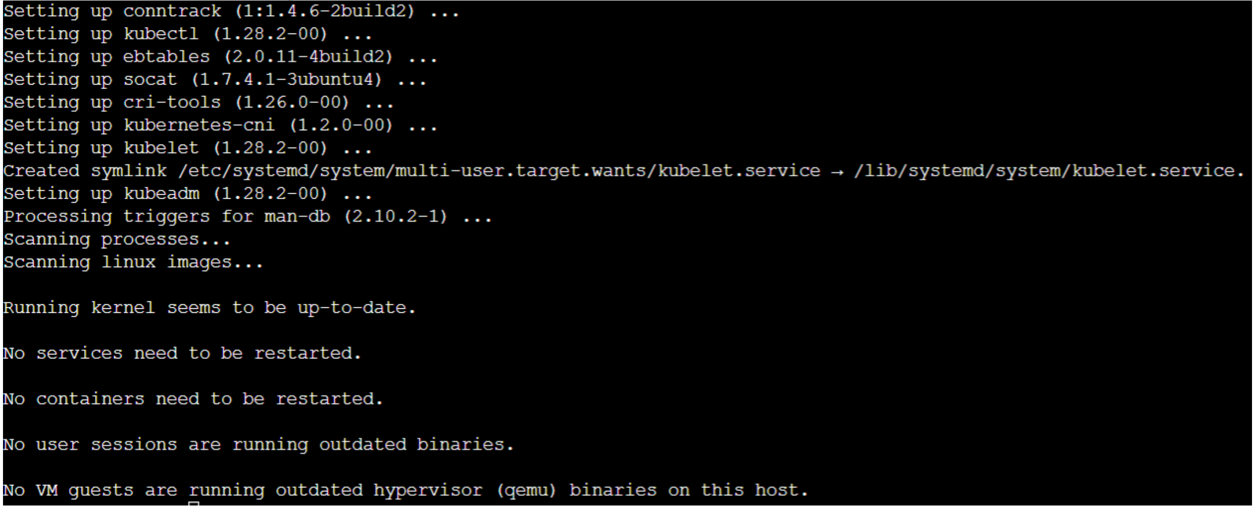
sudo apt-get install -y socat



```
ubuntu@ip-172-31-47-180:~$ sudo apt install 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 12 not upgraded.
Need to get 374 kB of archives.
After this operation, 1649 kB of additional disk space will be used.
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 socat amd64 1.8.0.0-4build3 [374 kB]
Fetched 374 kB in 0s (13.8 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68203 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...
```

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

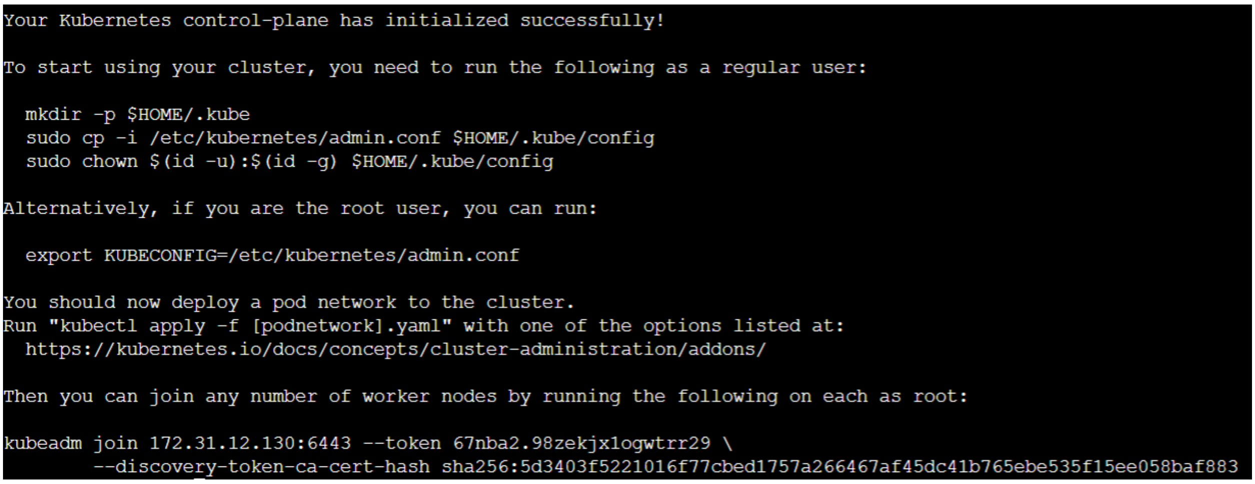
sudo kubeadm init --pod-network-cidr=10.244.0.0/16



```
Setting up conntrack (1:1.4.6-2build2) ...
Setting up kubectl (1.28.2-00) ...
Setting up ebtables (2.0.11-4build2) ...
Setting up socat (1.7.4.1-3ubuntu4) ...
Setting up cri-tools (1.26.0-00) ...
Setting up kubernetes-cni (1.2.0-00) ...
Setting up kubelet (1.28.2-00) ...
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /lib/systemd/system/kubelet.service.
Setting up kubeadm (1.28.2-00) ...
Processing triggers for man-db (2.10.2-1) ...
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.
```

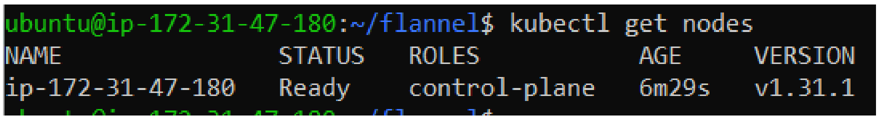
Run this command on master and also copy and save the Join command

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



```
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 172.31.12.130:6443 --token 67nba2.98zekjx1ogwtrr29 \
        --discovery-token-ca-cert-hash sha256:5d3403f5221016f77cbed1757a266467af45dc41b765ebe535f15ee058baf883
```

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

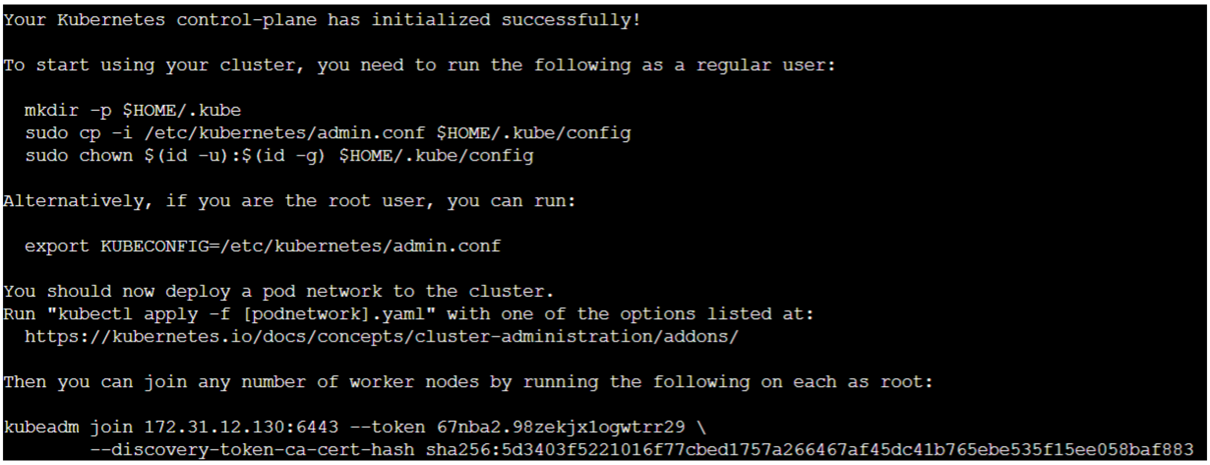


```
ubuntu@ip-172-31-47-180:~/flannel$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-47-180   Ready    control-plane   6m29s   v1.31.1
```

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

master.sudo kubeadm join 172.31.27.176:6443 --token

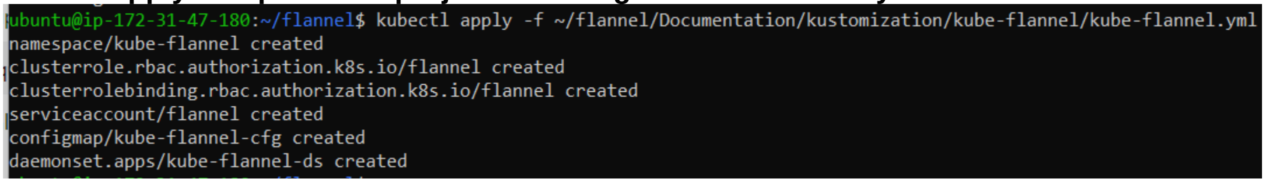
**ttay2x.n0sqeukjai8sgfg3 **
--discovery-token-ca-cert-hash
sha256:d6fc5fb7e984c83e2807780047fec6c4f2acfe9da9184ecc028d77157608fb
b6



```
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 172.31.12.130:6443 --token 67nba2.98zekjx1ogwtrr29 \
        --discovery-token-ca-cert-hash sha256:5d3403f5221016f77cbed1757a266467af45dc41b765ebe535f15ee058baf883
```

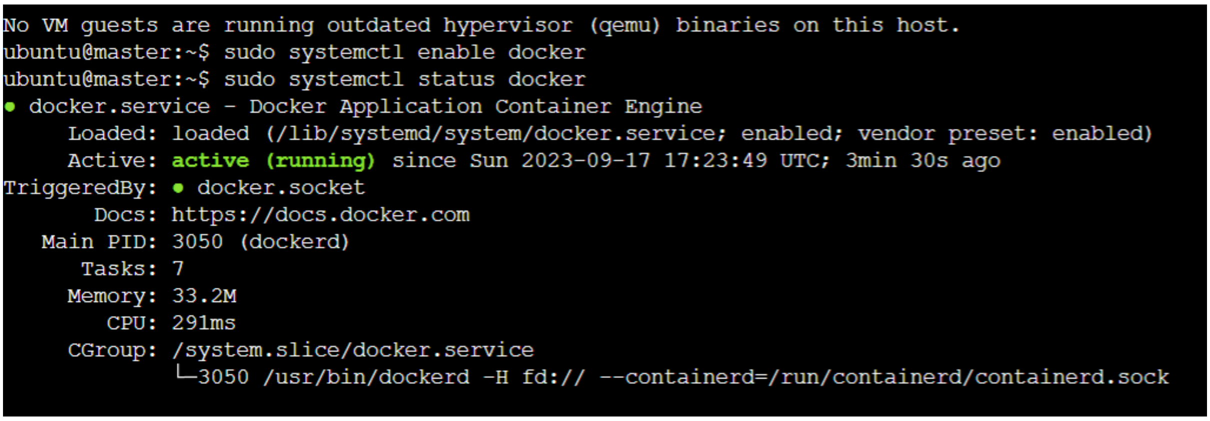
Step 9: Now Run the command kubectl get nodes to see the nodes after executing Joincommand on nodes. Since Status is NotReady we have to add a network plugin. And also we have to give the name to the nodes.

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



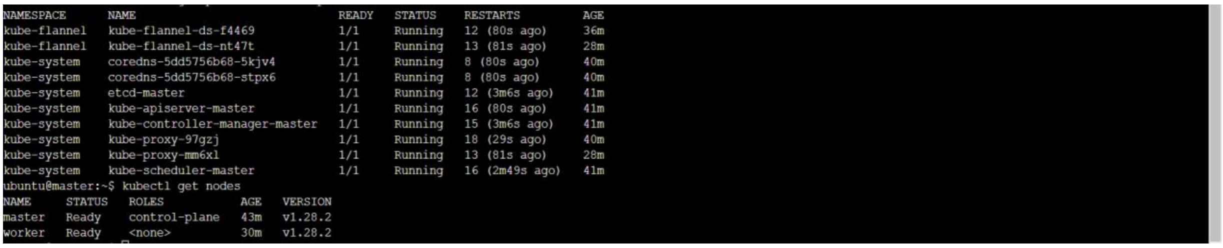
```
ubuntu@ip-172-31-47-180:~/flannel$ kubectl apply -f ~/flannel/Documentation/kustomization/kube-flannel/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

sudo systemctl status kubelet



```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@master:~$ sudo systemctl enable docker
ubuntu@master:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-09-17 17:23:49 UTC; 3min 30s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 3050 (dockerd)
      Tasks: 7
     Memory: 33.2M
        CPU: 291ms
     CGroup: /system.slice/docker.service
             └─3050 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Step 10: Run command kubectl get nodes -o wide . And Hence we can see we have Successfully connected Node 1 and Node 2 to the Master.



```
NAMESPACE     NAME                               READY   STATUS    RESTARTS        AGE
kube-flannel  kube-flannel-ds-f4469              1/1     Running   12 (80s ago)    36m
kube-flannel  kube-flannel-ds-nt47t              1/1     Running   13 (81s ago)    28m
kube-system   coredns-5dd5756b68-5kjv4           1/1     Running   8 (80s ago)     40m
kube-system   coredns-5dd5756b68-stpx6           1/1     Running   8 (80s ago)     40m
kube-system   etcd-master                        1/1     Running   12 (3m6s ago)   41m
kube-system   kube-apiserver-master              1/1     Running   16 (80s ago)    41m
kube-system   kube-controller-manager-master     1/1     Running   15 (3m6s ago)   41m
kube-system   kube-proxy-97gzj                   1/1     Running   18 (29s ago)    40m
kube-system   kube-proxy-mm6xl                   1/1     Running   13 (81s ago)    28m
kube-system   kube-scheduler-master              1/1     Running   16 (2m49s ago)  41m
ubuntu@master:~$ kubectl get nodes
NAME     STATUS   ROLES           AGE   VERSION
master   Ready    control-plane   43m   v1.28.2
worker   Ready    <none>          30m   v1.28.2
```

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.