Name:Chinmay Chaudhari D15C Roll. No 6

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. Like 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.

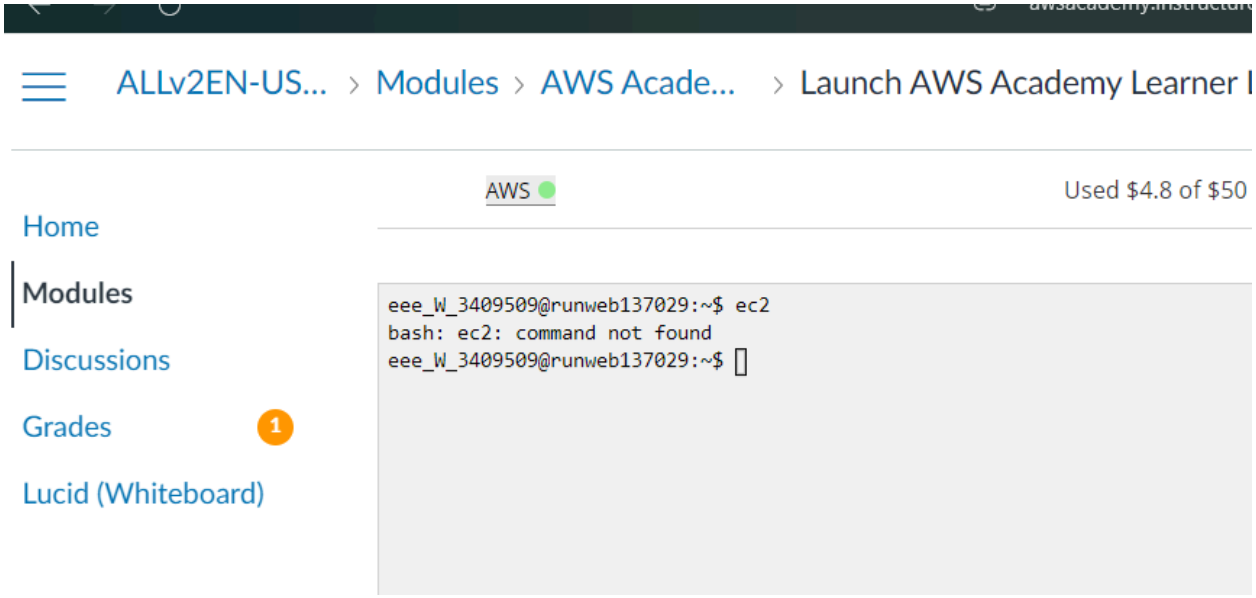
Kubernetes simplifies the deployment and operation of applications in a microservice architecture by providing an abstraction layer over a group of hosts. This allows development teams to deploy their applications while Kubernetes takes care of key tasks, including:

- 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.

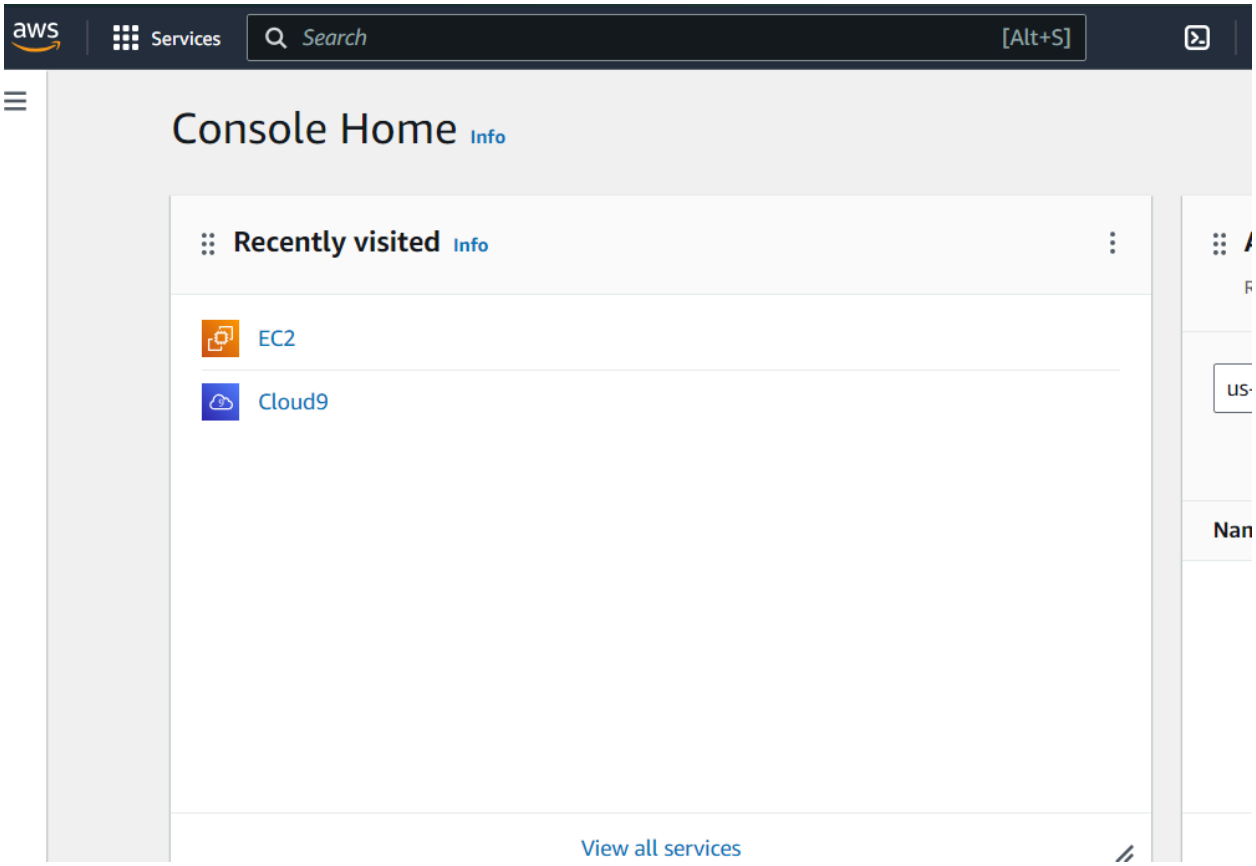
Steps:

Set Up the instances of each machine

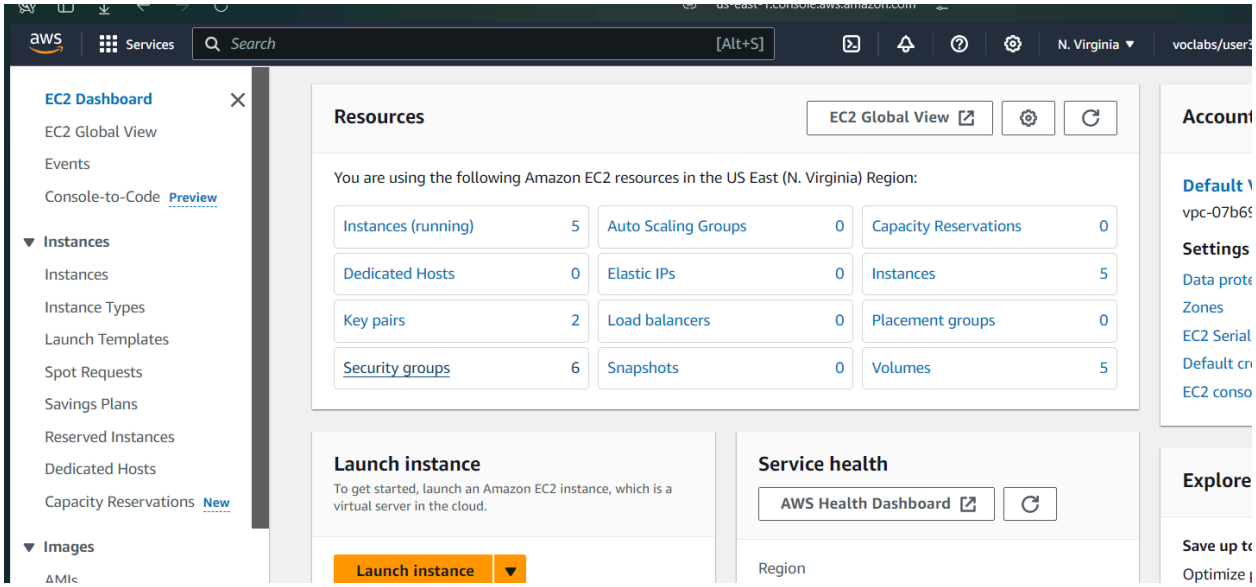
1. open the aws academy.



Name:Chinmay Chaudhari D15C Roll. No 6

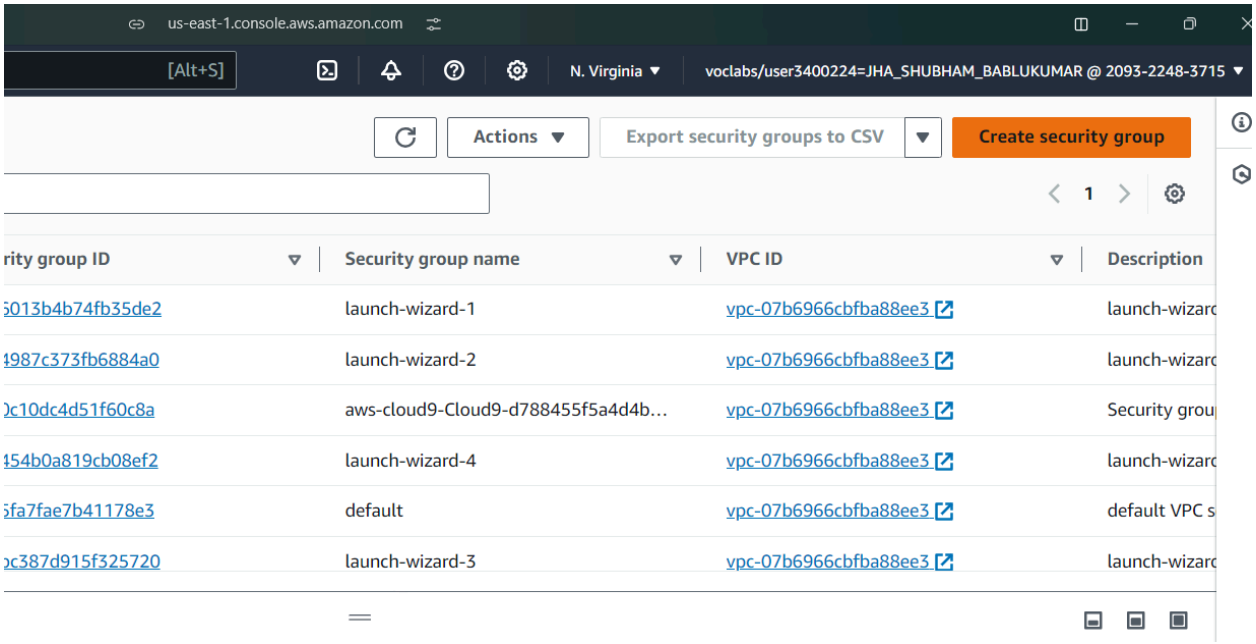


2. Click on security groups

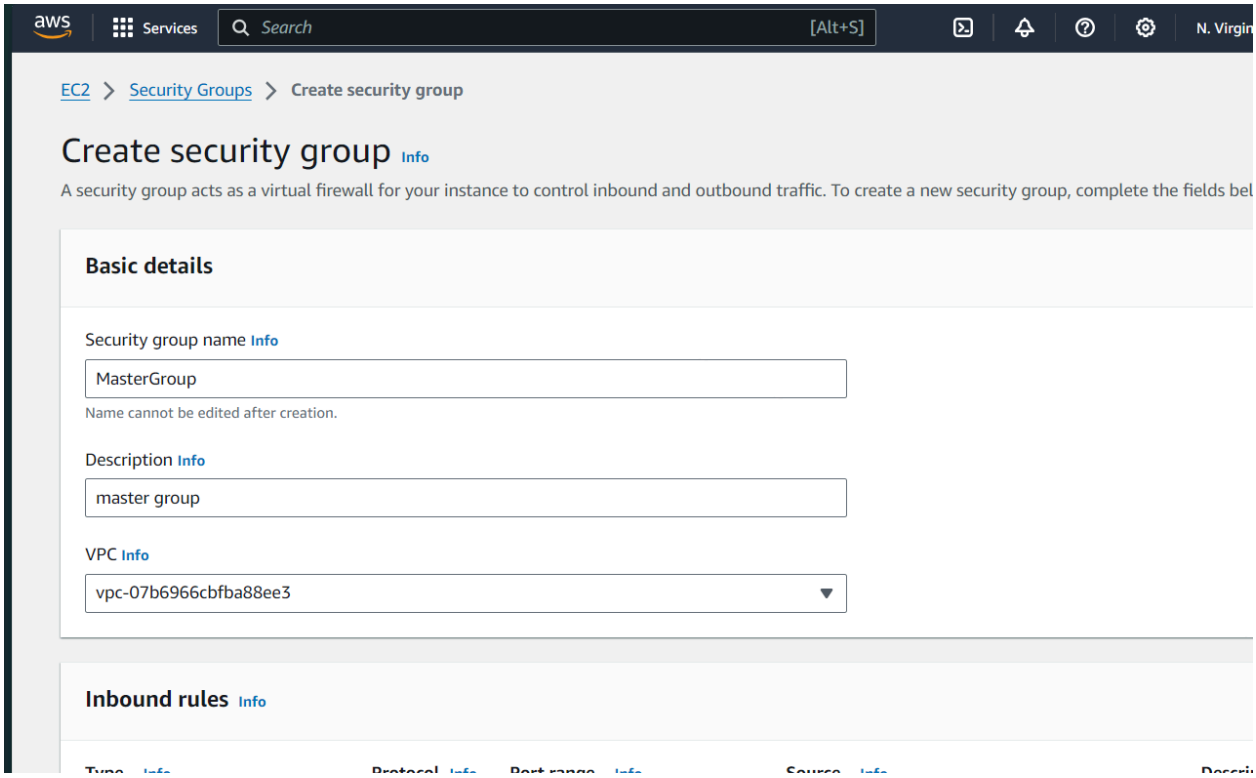


3. create two secure groups one for master and other for the two nodes.

Name:Chinmay Chaudhari D15C Roll. No 6

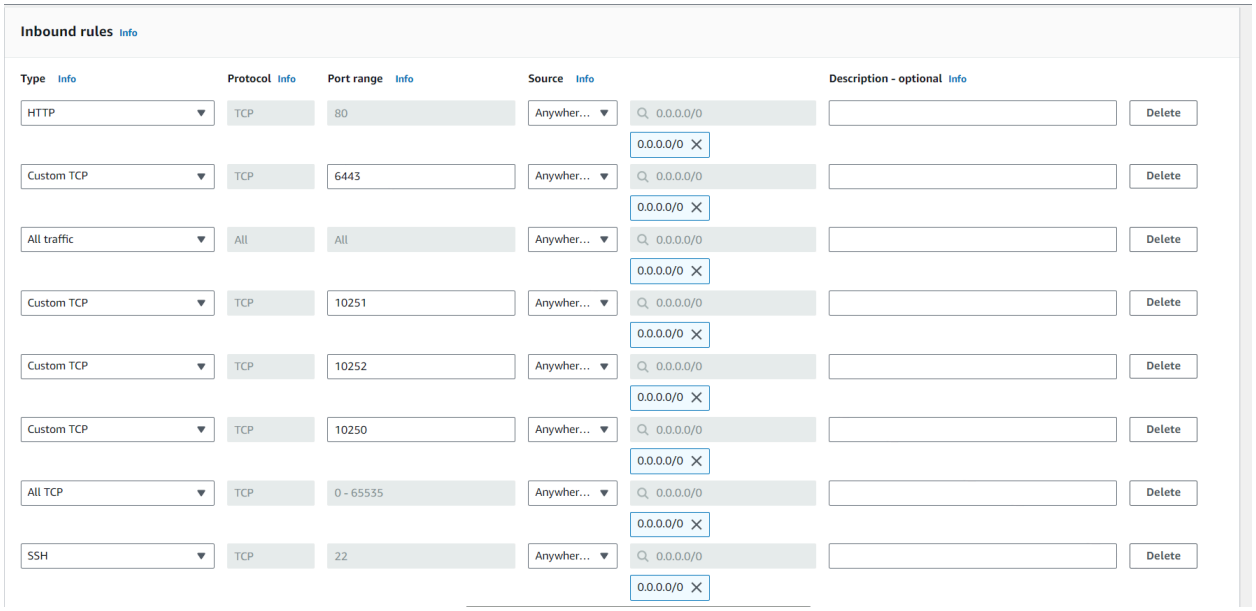


4. enter details and add inbound rules (I have given MasterGroup for the master group)



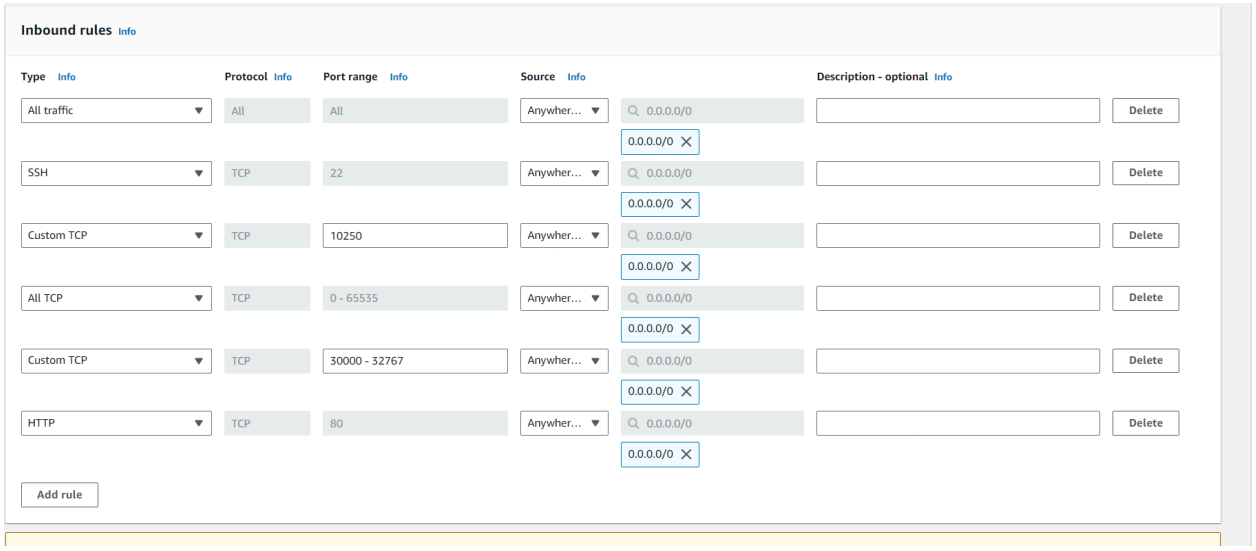
Name:Chinmay Chaudhari D15C Roll. No 6

You have to look for the particular configuration which I did (in the image below)



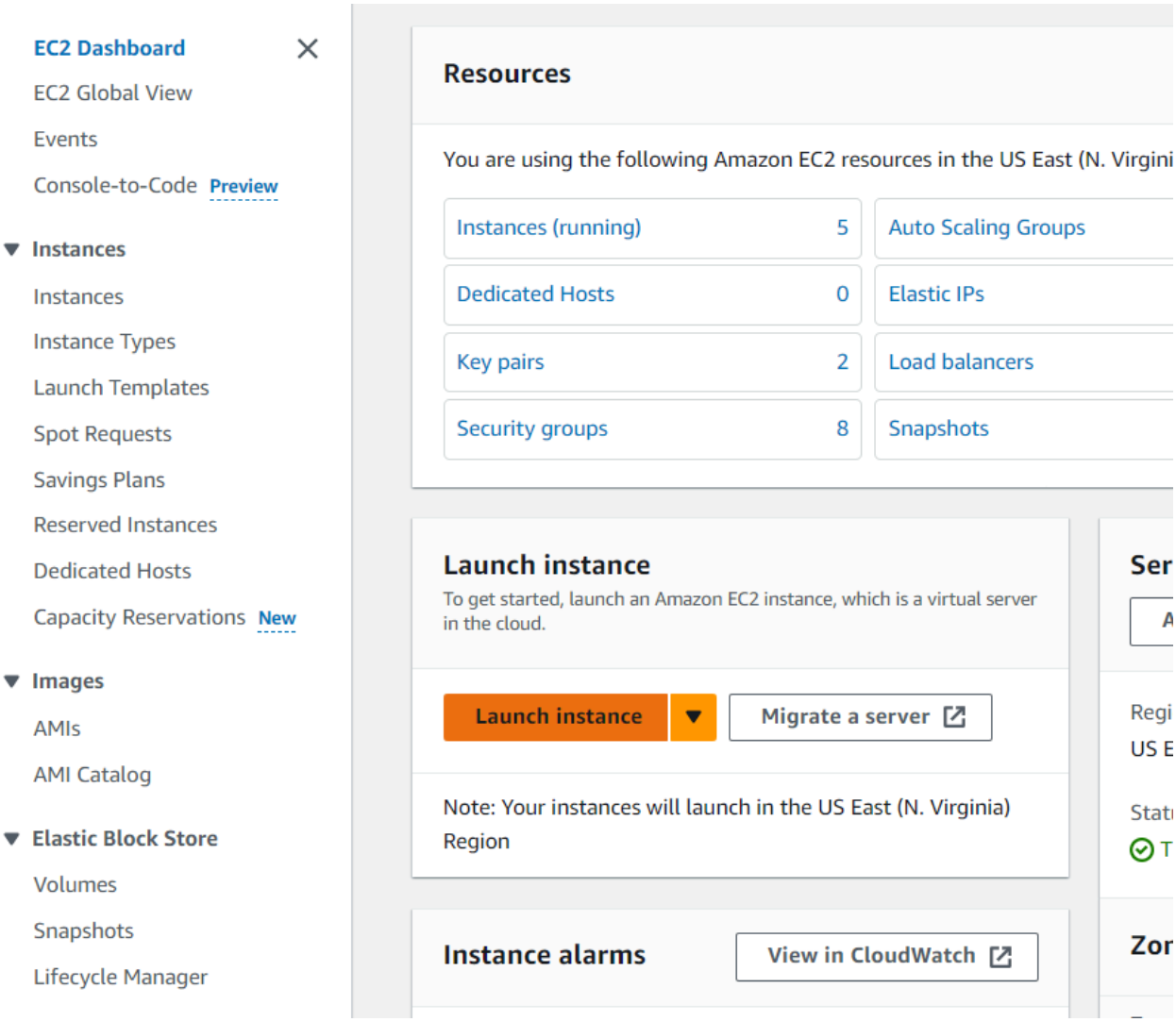
click on create security group below.

now do the same for a node group.



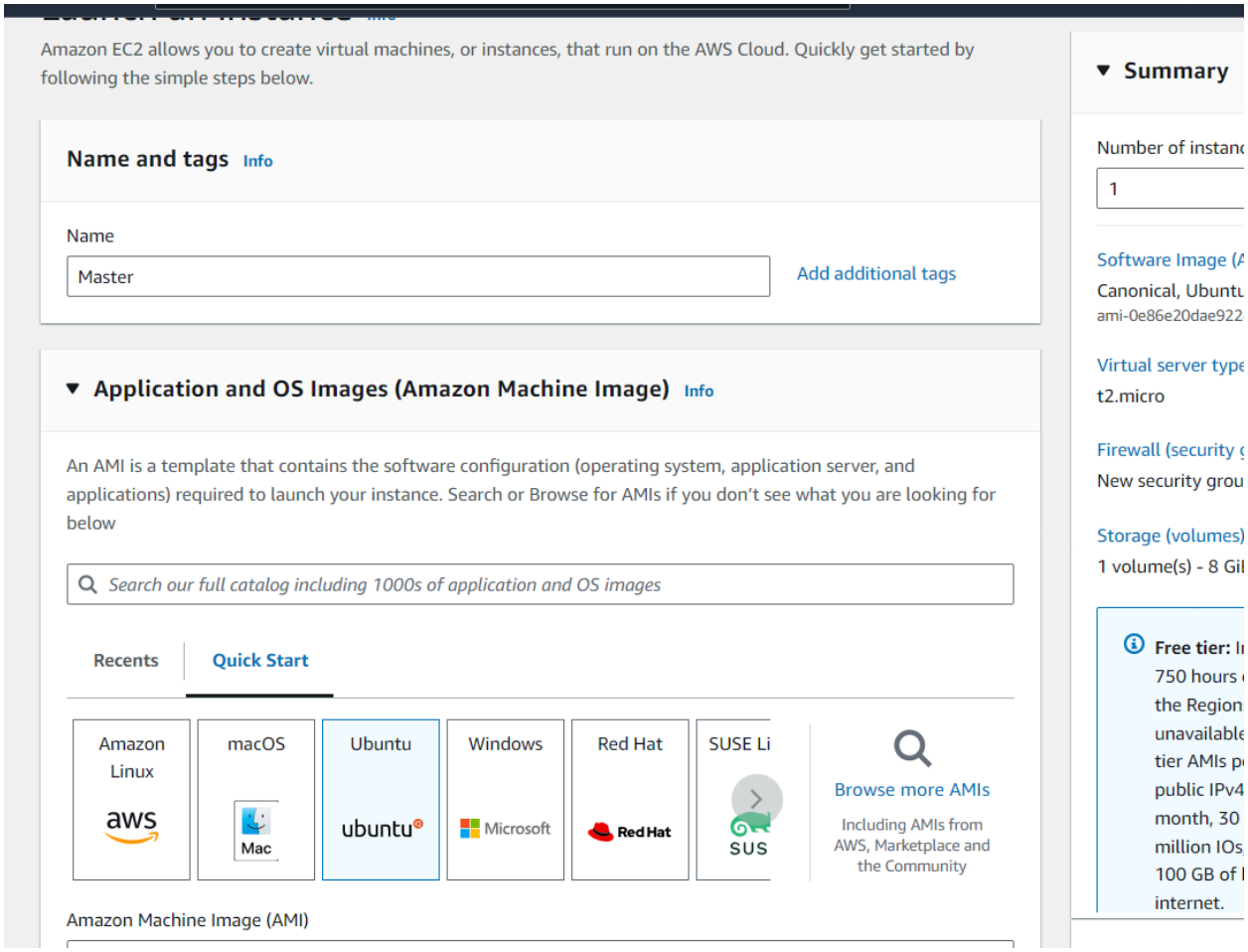
Name:Chinmay Chaudhari D15C Roll. No 6

5. now go to ec2 and launch an instance

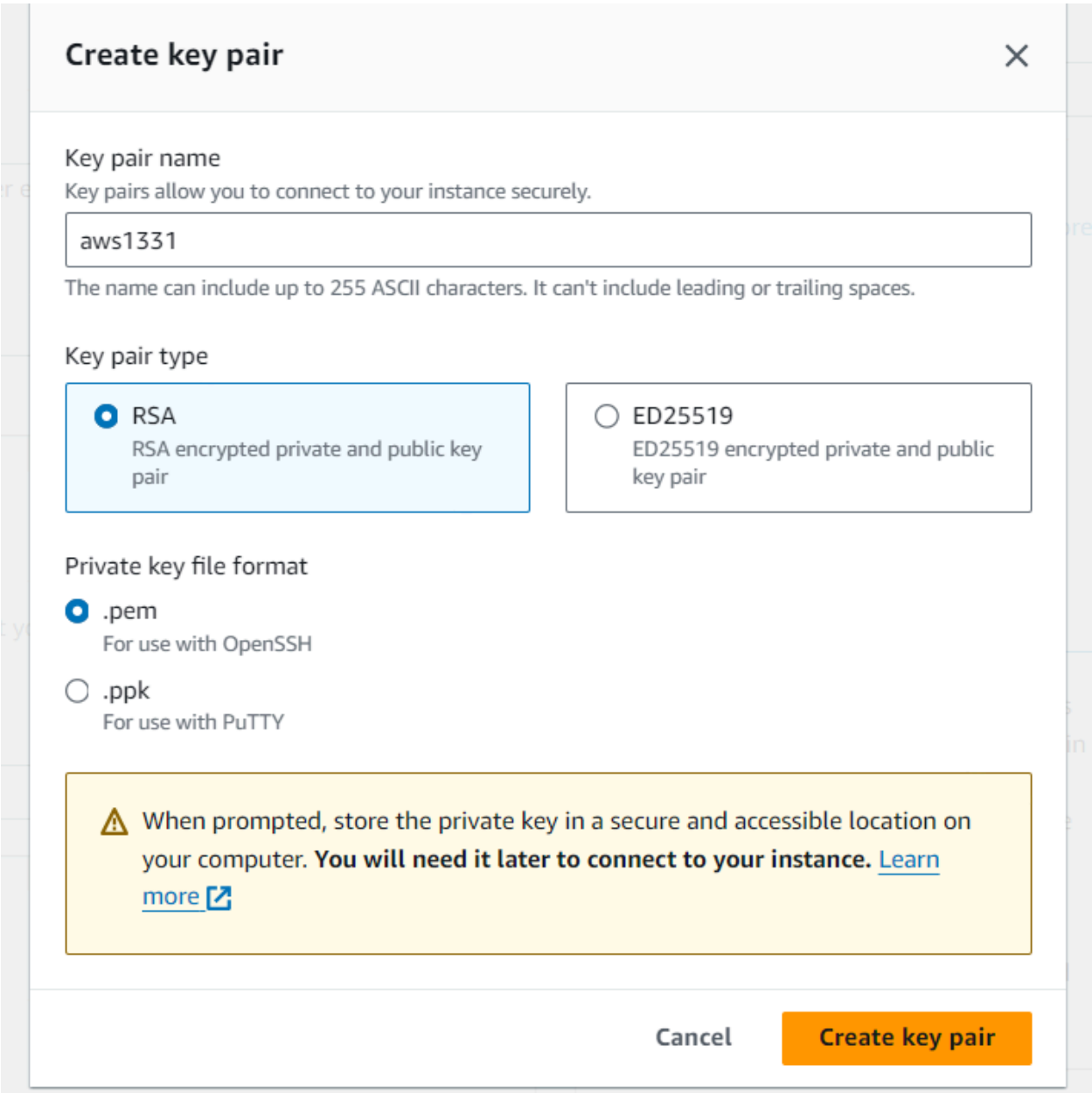


add name and set ubuntu:

Name:Chinmay Chaudhari D15C Roll. No 6



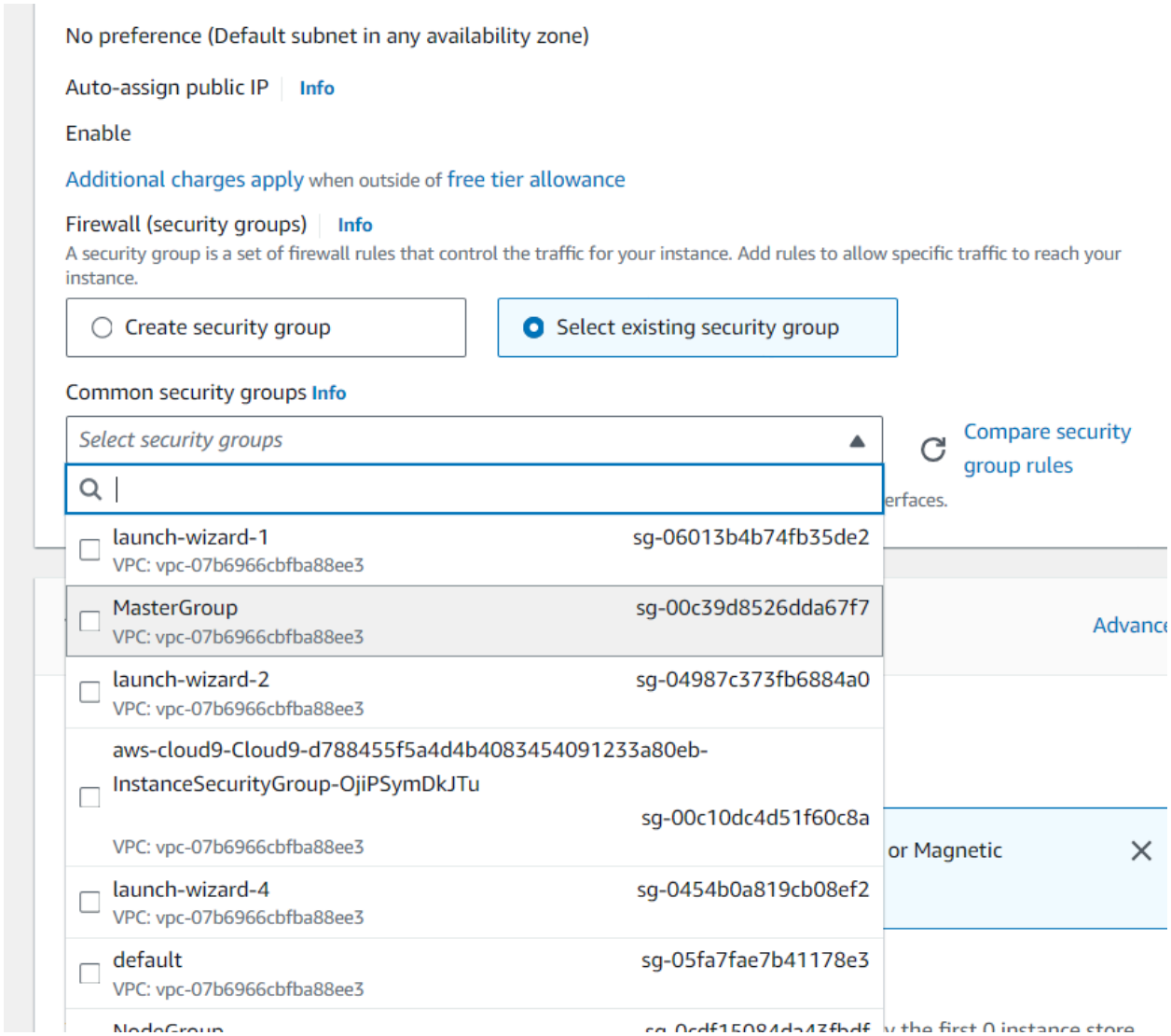
create a key if you want



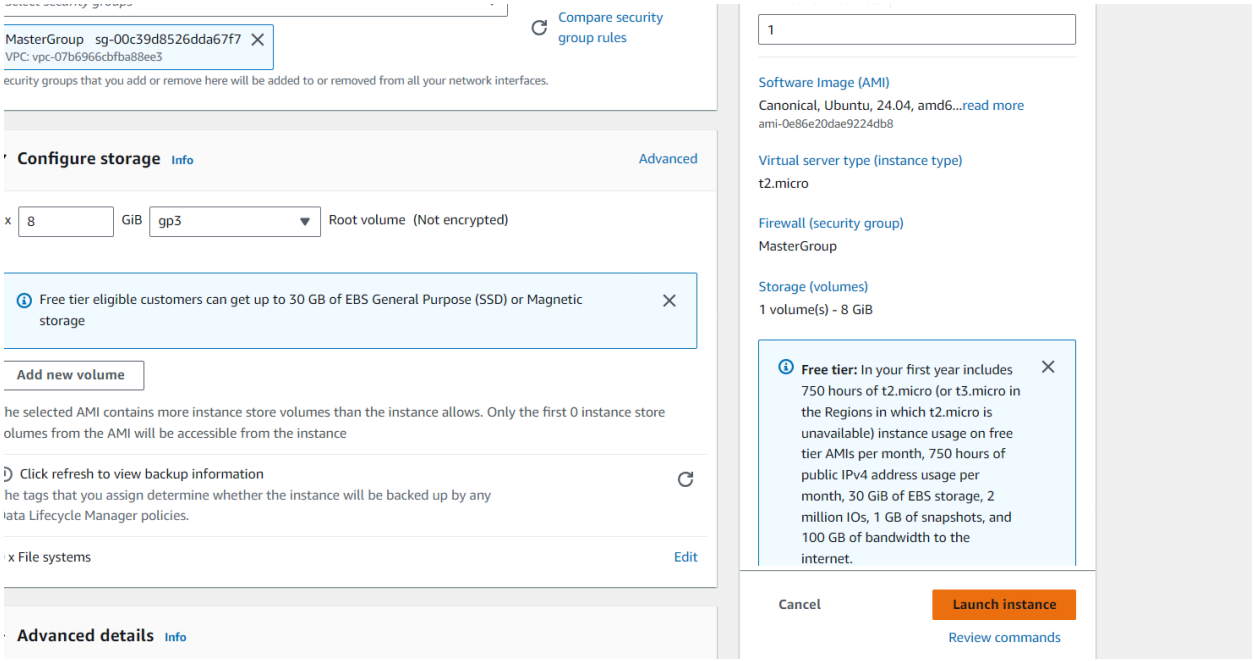
If you want you can reuse the key pair generated earlier.

Name:Chinmay Chaudhari D15C Roll. No 6

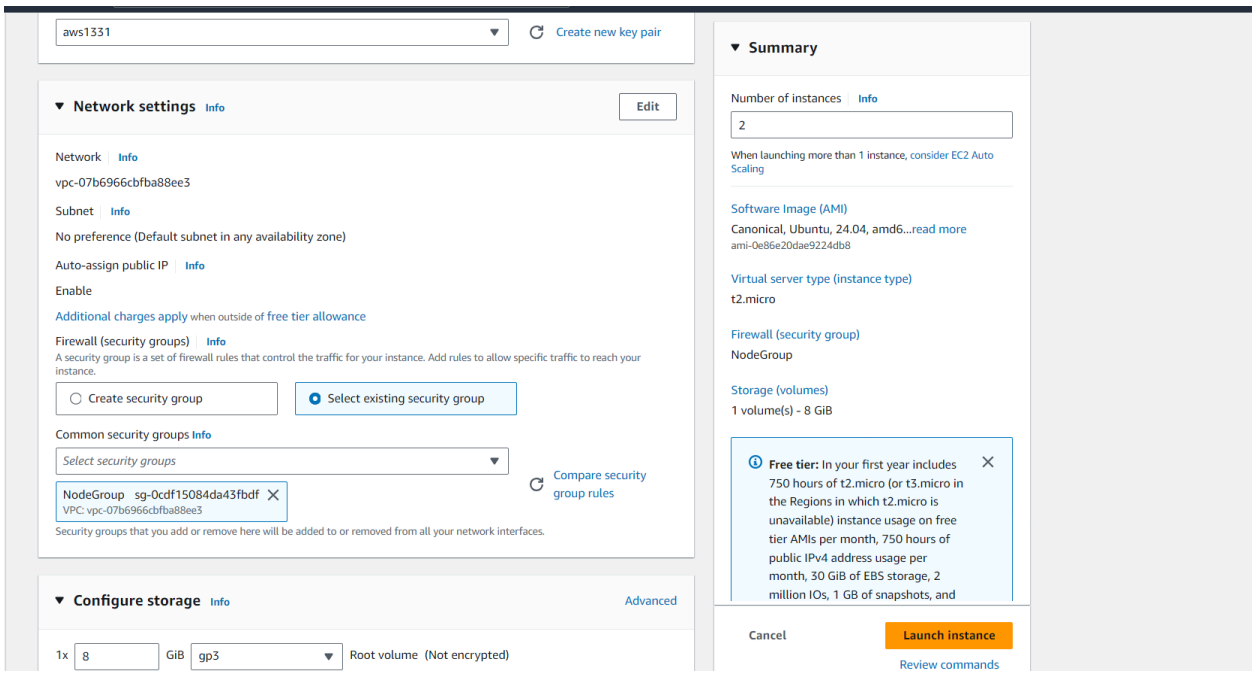
Select the security group for master.



then launch:



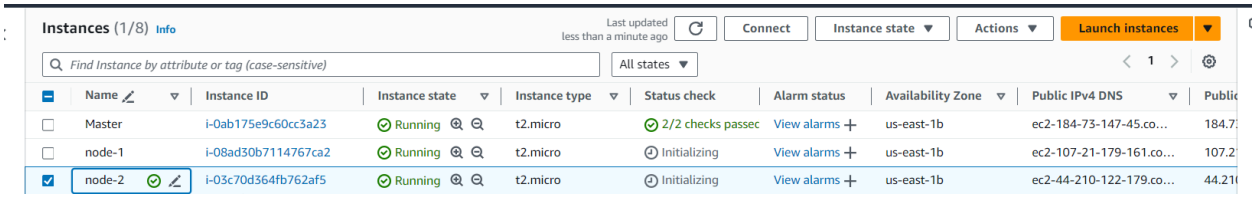
do the same for node instance just select the number of instance as 2.
and select custom security group as node group.



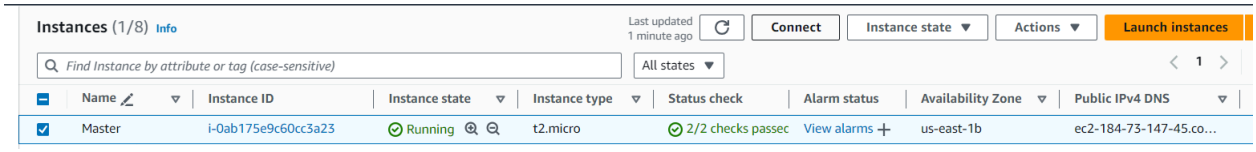
dont give name now. and launch instance.

now go to instances and give name to the blanck ones:

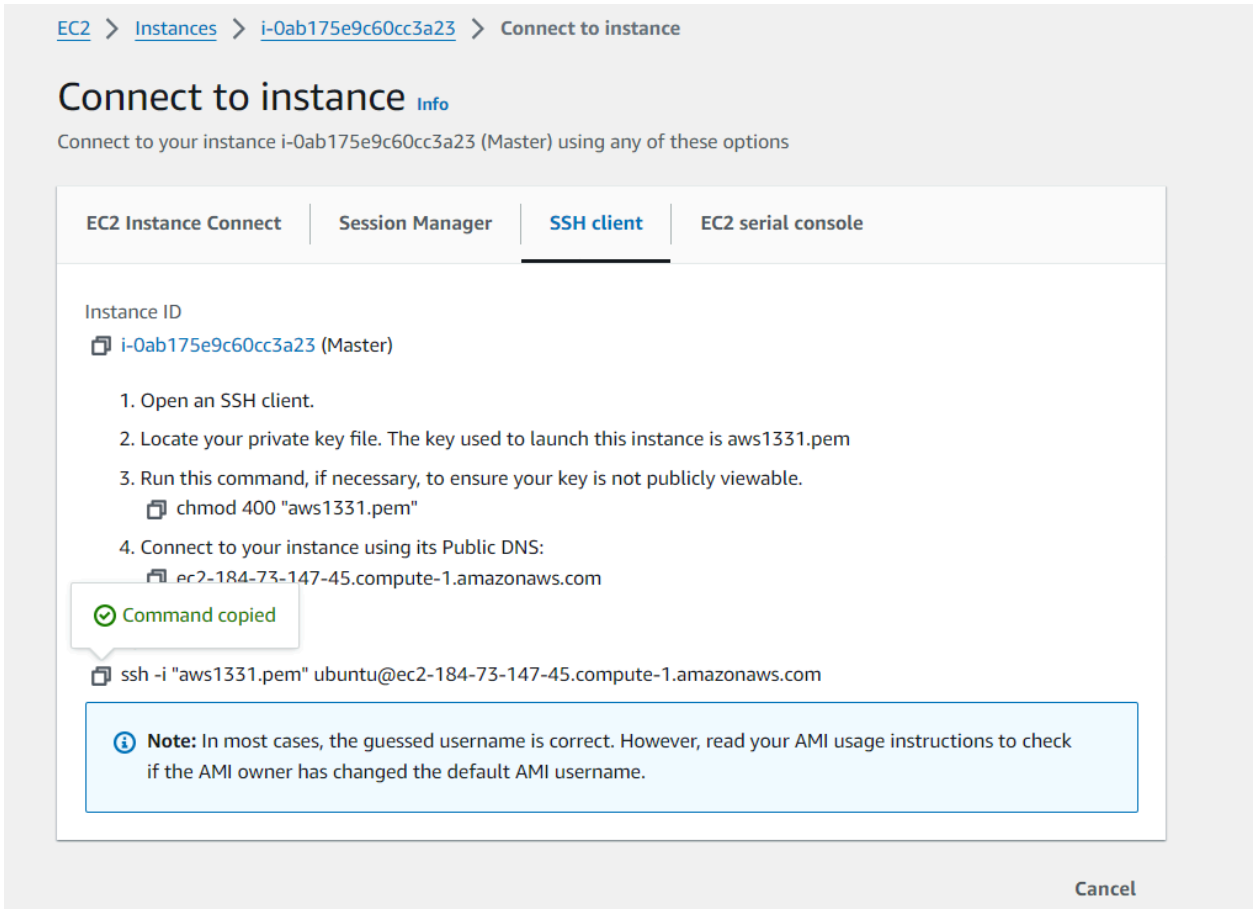
Name:Chinmay Chaudhari D15C Roll. No 6



6. select master and connect:

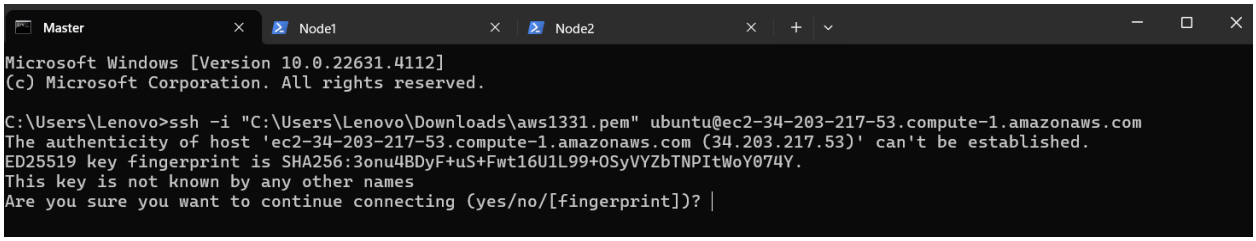


click on ssh client:
copy the command below the SSH client session

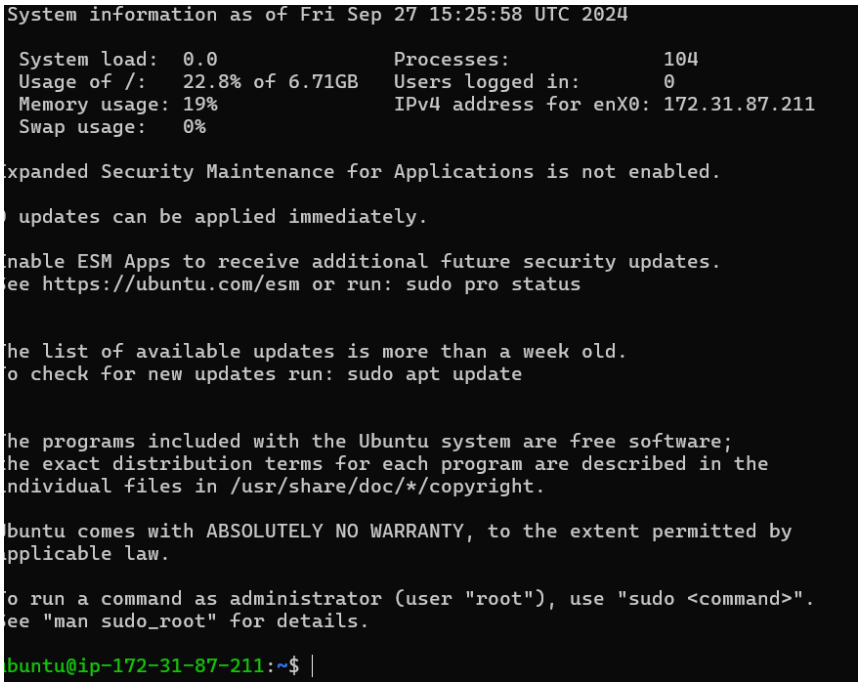


Name:Chinmay Chaudhari D15C Roll. No 6

7. Enter the copied command to a terminal window.

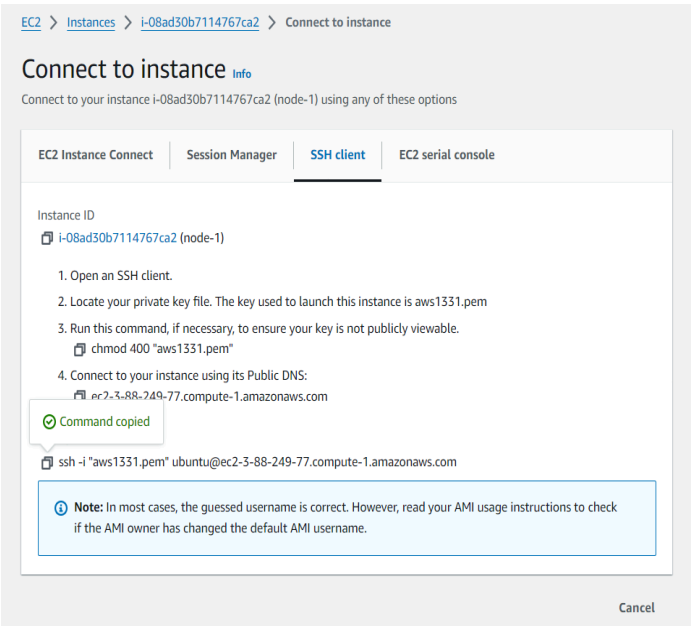


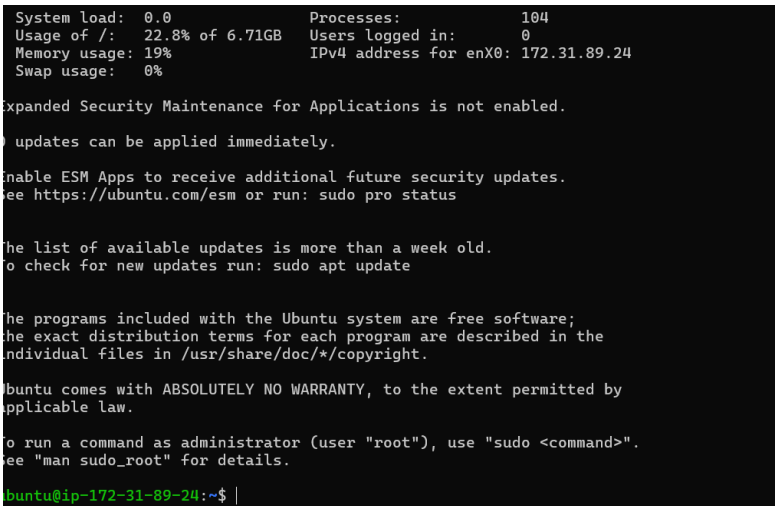
It would prompt whether we want to continue connecting. type yes.



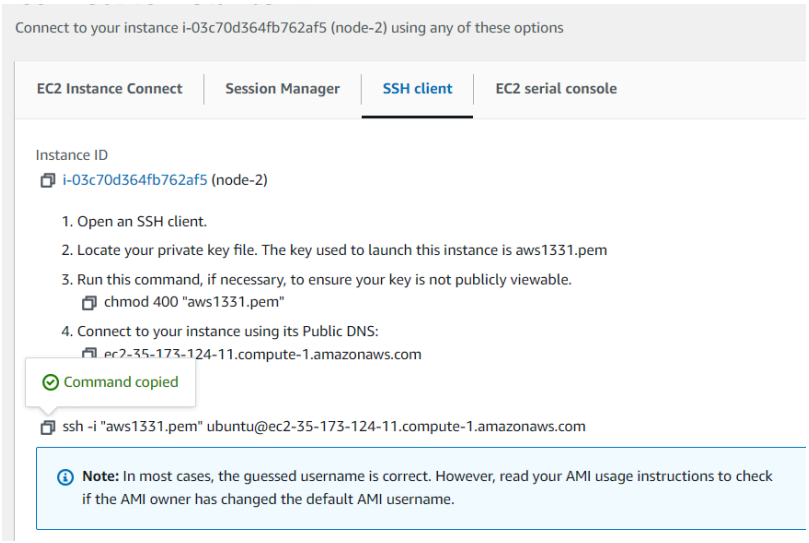
The step is similar for node 1 and node 2 instances too. Just use different terminal windows.

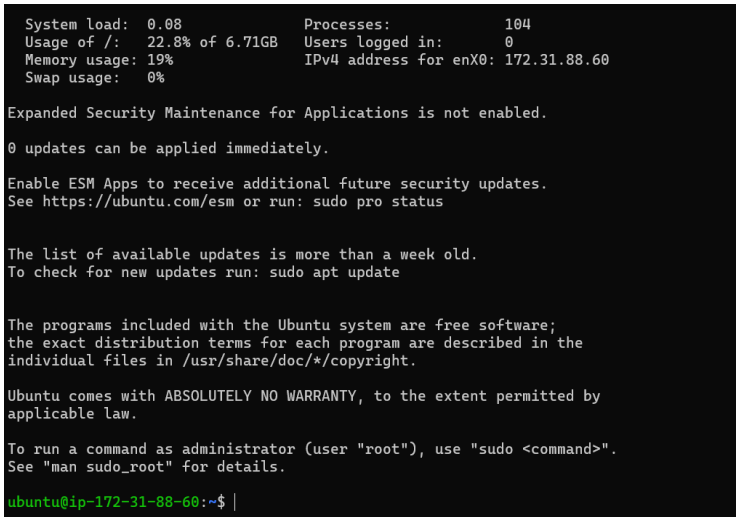
node 1:





node 2:





8. From now on run the commands on all the 3 terminals unless instructed otherwise.

and the images (screen shots) will only be of master unless stated otherwise.

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"

Name:Chinmay Chaudhari D15C Roll. No 6

```
ubuntu@ip-172-31-87-211:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
```

now for node1 and node2:

Master | Node1 | Node2

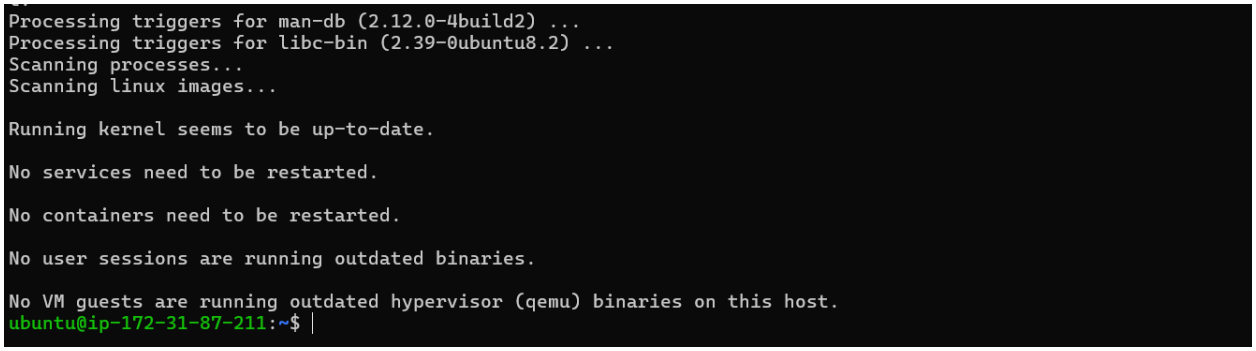
```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-89-24:~$ 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/n
ull
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
```

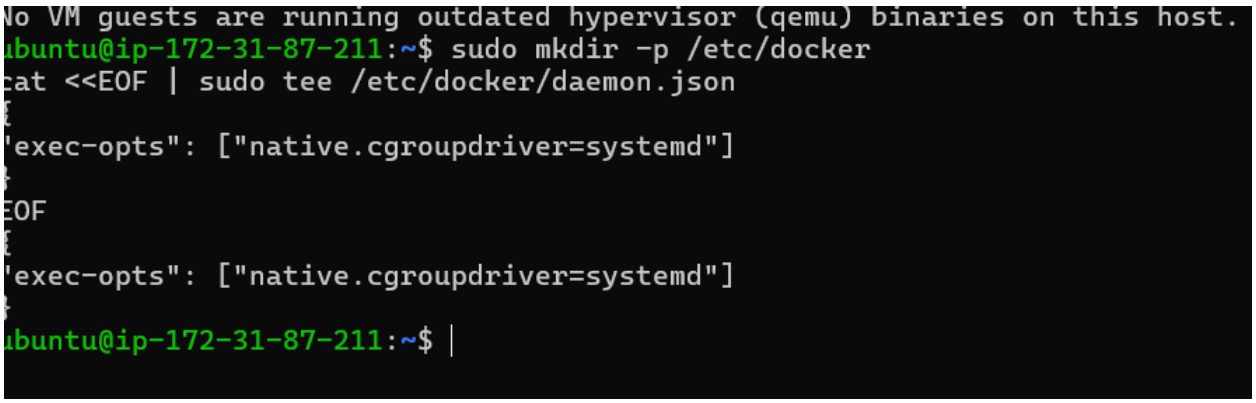
Master | Node1 | Node2

```
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-88-60:~$ 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/n
ull
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
```

sudo apt-get update

sudo apt-get install -y docker-ce

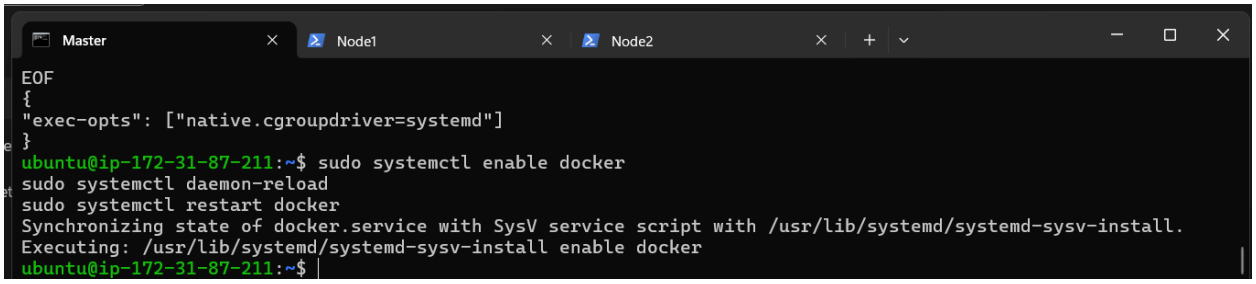




sudo systemctl enable docker

sudo systemctl daemon-reload

sudo systemctl restart docker



9. 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

```
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-172-31-87-211:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --de
armor -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/de
b/ /' | 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/ /
ubuntu@ip-172-31-87-211:~$
```

Run the commands:

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

```
ubuntu@ip-172-31-87-211:~$ sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  InRelease [1186
 B]
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  Packages [4865
B]
```

```
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.
```

sudo systemctl enable --now kubelet

sudo apt-get install -y containerd

```
ubuntu@ip-172-31-87-211:~$ 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 longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7
  pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
```

```
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-172-31-87-211:~$
```

sudo mkdir -p /etc/containerd

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

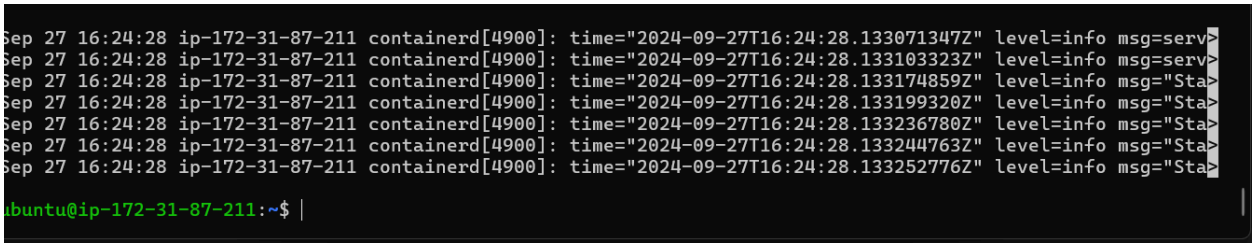
```
Master  Node1  Node2
No services need to be restarted.

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-172-31-87-211:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
```

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

[ttrpc]
  address = ""
  gid = 0
  uid = 0
ubuntu@ip-172-31-87-211:~$
```

sudo systemctl restart containerd

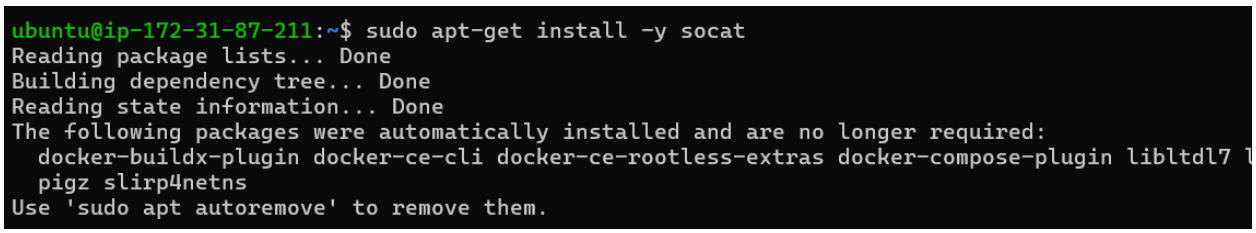
sudo systemctl enable containerd

sudo systemctl status containerd



exit with ctrl+c.

sudo apt-get install -y socat



Run the following command in master only:

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

if it gives error use:

sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU,Mem

```
To see the stack trace of this error execute with --v=5 or higher
ubuntu@ip-172-31-87-211:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCP
U,Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (957 MB) is less than the minimum 1700 MB
[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'
W0927 16:47:12.068193    6025 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of t
he container runtime is inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/paus
e:3.10" as the CRI sandbox image.
```

```
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 172.31.87.211:6443 --token lhbvbx.kzlxu87vmv8hvx4o \
        --discovery-token-ca-cert-hash sha256:1cef7709c45a42691a2ff0e44e3acf7f0e214fec7f4f822bb6818f3cfd24ea4
3
ubuntu@ip-172-31-87-211:~$
```

Token and ca

Note: copy the text after kubeadm that you see at the later part like below:

**kubeadm join 172.31.87.211:6443 --token lhbvbx.kzlxu87vmv8hvx4o **

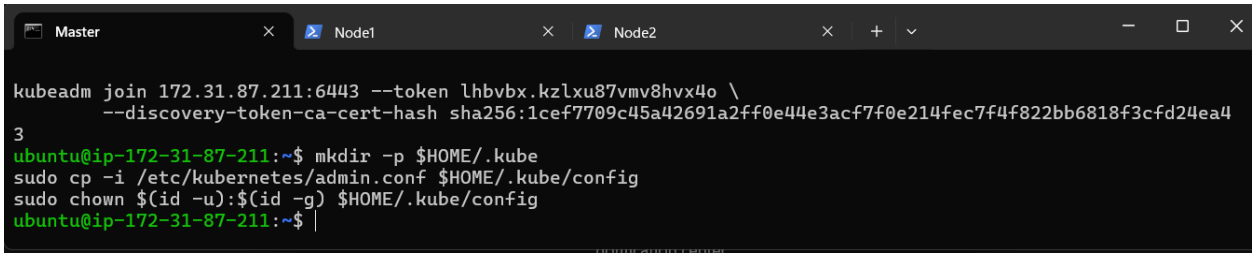
--discovery-token-ca-cert-hash sha256:1cef7709c45a42691a2ff0e44e3acf7f0e214fec7f4f822bb6818f3cfd24ea43

Run this command on master

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config



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

```
ubuntu@ip-172-31-87-211:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-87-211   NotReady   control-plane   3m48s   v1.31.1
```

Now paste the token and ca that I asked to copy earlier, on both the nodes.
use sudo before them.

it would be something like:
sudo kubeadm join <your-master-node-ip>:6443 --token <your-token> --discovery-token-ca-cert-hash sha256:<your-ca-cert-hash>
(it has placeholders)

Node1:

```
ubuntu@ip-172-31-89-24:~$ sudo kubeadm join 172.31.87.211:6443 --token lhbvbx.kzlxu87vmv8hvx4o \
        --discovery-token-ca-cert-hash sha256:1cef7709c45a42691a2ff0e44e3acf7f0e214fec7f4f822bb6818f3cfd24ea4
3
[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
```

Node2:

```
ubuntu@ip-172-31-88-60:~$ sudo kubeadm join 172.31.87.211:6443 --token lhbvbx.kzlxu87vmv8hvx4o \
        --discovery-token-ca-cert-hash sha256:1cef7709c45a42691a2ff0e44e3acf7f0e214fec7f4f822bb6818f3cfd24ea4
3
[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 1.00236733s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap
```

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

```
ubuntu@ip-172-31-87-211:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-87-211   NotReady   control-plane   7m35s   v1.31.1
ip-172-31-88-60    NotReady   <none>          12s     v1.31.1
ip-172-31-89-24    NotReady   <none>          32s     v1.31.1
ubuntu@ip-172-31-87-211:~$
```

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

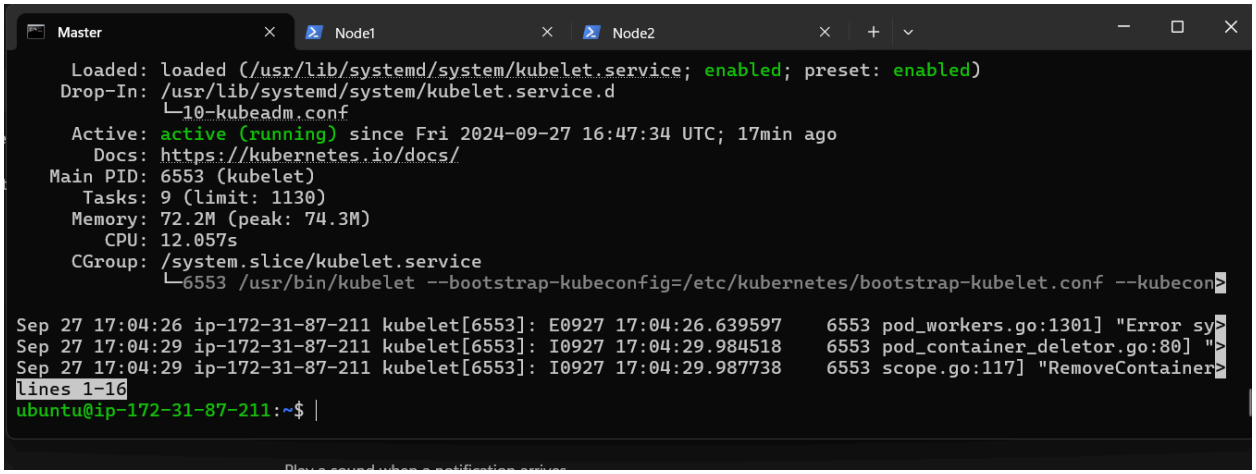
the name to the nodes.

paste this command on master terminal. (and the following commands will be based on master unless states otherwise.

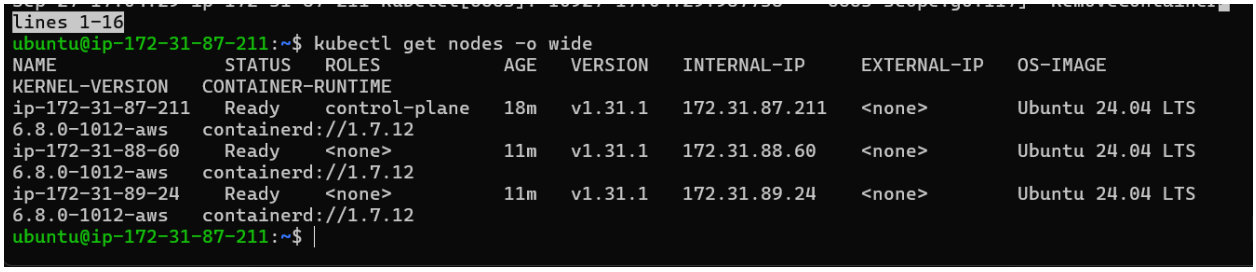
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

sudo systemctl status kubelet

again use ctrl+c to exit.



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



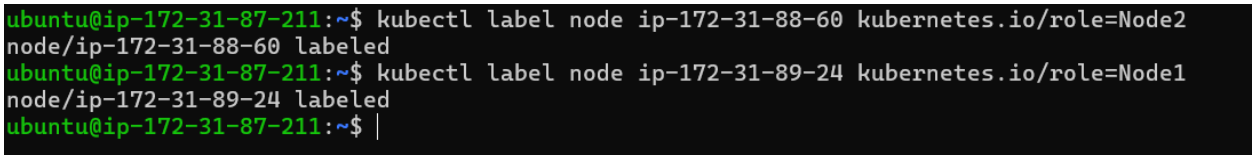
Now to Rename run this command

Syntax: kubectl label node <node-ip> kubernetes.io/role=worker

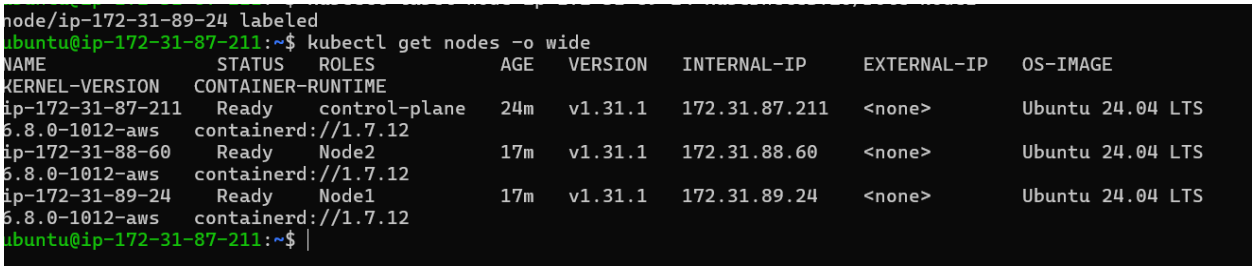
examples:

Rename to Node 1:kubectl label node ip-<node1ip> kubernetes.io/role=Node1

Rename to Node 2:kubectl label node ip-<node2ip> kubernetes.io/role=Node2



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.



Conclusion:

In this Advanced DevOps Lab experiment, we began by setting up three EC2 Ubuntu instances on AWS, designating one as the Master node and the others as Worker nodes.
We then installed Docker and Kubernetes on all instances, ensuring Docker was properly configured.
The Kubernetes cluster was initialized on the Master node, and the Flannel networking plugin was applied to facilitate communication between nodes.
Finally, we joined the Worker nodes to the cluster using the provided token and hash, resulting in a fully operational Kubernetes cluster ready for managing and scaling containerized applications.