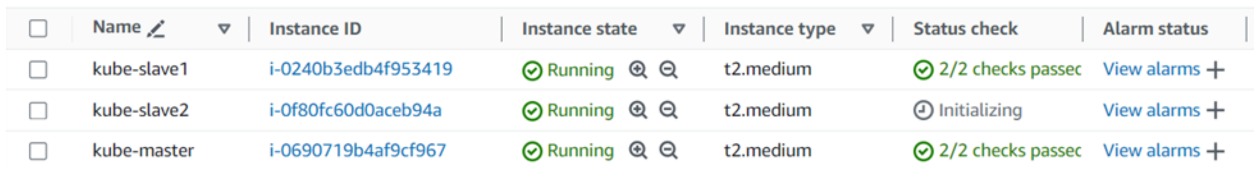
Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 3

<u>AIM:</u> To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

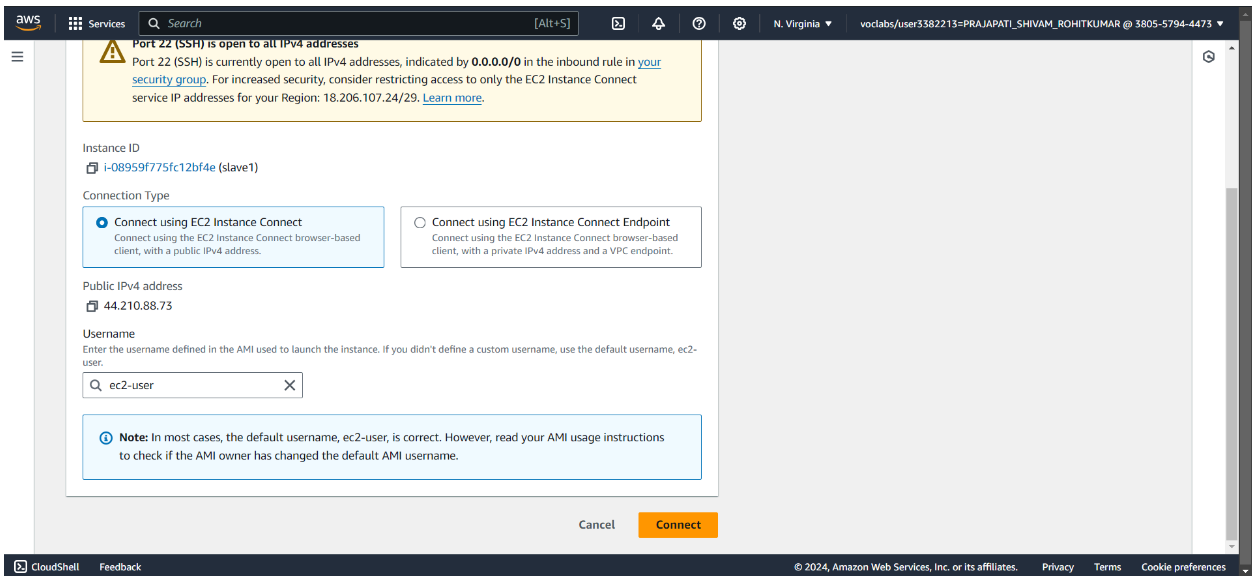
<u>STEPS</u>:

Step 1: Launch three EC2 instances with Amazon Linux on AWS: one as the master node and two as additional slave nodes. Ensure you choose the t2.medium instance type rather than the default t2.micro. This is because t2.medium provides more CPU, memory and consistent performance which are crucial for effectively running Kubernetes components and managing cluster workloads.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status
kube-slave1	i-0240b3edb4f953419	Running	t2.medium	2/2 checks passec	View alarms +
kube-slave2	i-0f80fc60d0aceb94a	Running	t2.medium	Initializing	View alarms +
kube-master	i-0690719b4af9cf967	Running	t2.medium	2/2 checks passec	View alarms +

Step 2: Select each instance and click on connect button at the top in order to connect it to SSH terminal



aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3382213=PRAJAPATI_SHIVAM_ROHITKUMAR @ 3805-5794-4473

Port 22 (SSH) is open to all IPv4 addresses

Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 18.206.107.24/29. Learn more.

Instance ID

i-08959f775fc12bf4e (slave1)

Connection Type

- Connect using EC2 Instance Connect — Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.
- Connect using EC2 Instance Connect Endpoint — Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address

44.210.88.73

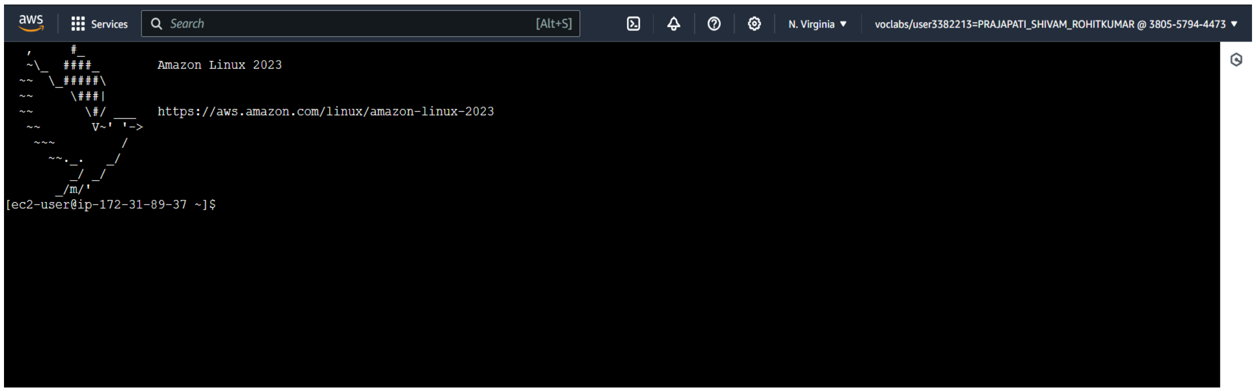
Username

Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

ec2-user

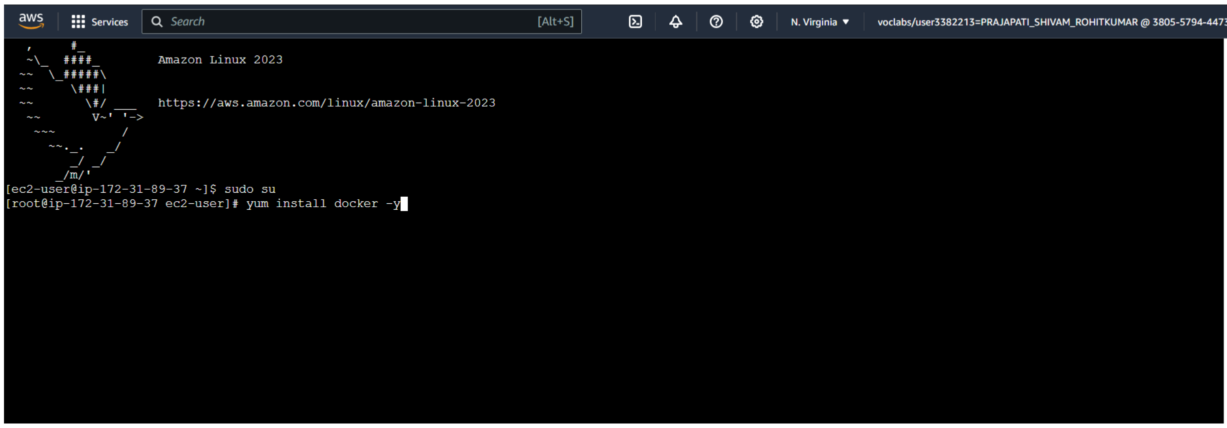
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

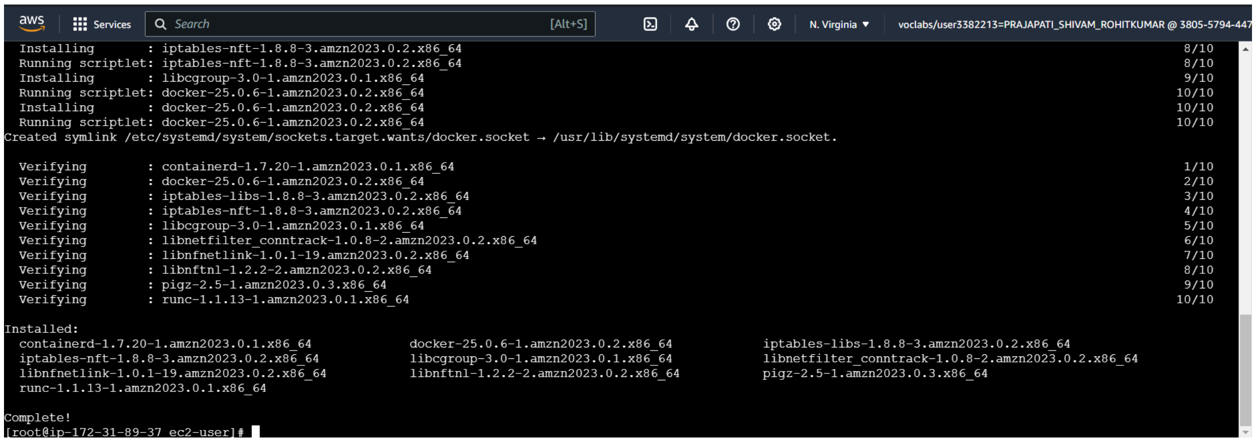

aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3382213=PRAJAPATI_SHIVAM_ROHITKUMAR @ 3805-5794-4473

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-89-37 ~]$
```

Step 3: Install docker on all 3 machines using the "**yum install docker -y**" command.


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3382213=PRAJAPATI_SHIVAM_ROHITKUMAR @ 3805-5794-4473

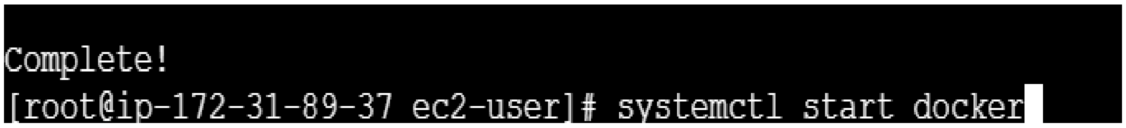
```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-89-37 ~]$ sudo su
[root@ip-172-31-89-37 ec2-user]# yum install docker -y
```


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3382213=PRAJAPATI_SHIVAM_ROHITKUMAR @ 3805-5794-447

```
  Installing       : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                      8/10
  Running scriptlet: iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                      8/10
  Installing       : libcgroup-3.0-1.amzn2023.0.1.x86_64                                                           9/10
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64                                                          10/10
  Installing       : docker-25.0.6-1.amzn2023.0.2.x86_64                                                          10/10
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64                                                          10/10
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.

  Verifying        : containerd-1.7.20-1.amzn2023.0.1.x86_64                                                       1/10
  Verifying        : docker-25.0.6-1.amzn2023.0.2.x86_64                                                           2/10
  Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                                                     3/10
  Verifying        : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                      4/10
  Verifying        : libcgroup-3.0-1.amzn2023.0.1.x86_64                                                           5/10
  Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64                                            6/10
  Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                                                     7/10
  Verifying        : libnftnl-1.2.2-2.amzn2023.0.2.x86_64                                                          8/10
  Verifying        : pigz-2.5-1.amzn2023.0.3.x86_64                                                                9/10
  Verifying        : runc-1.1.13-1.amzn2023.0.1.x86_64                                                            10/10

Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64           docker-25.0.6-1.amzn2023.0.2.x86_64           iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  iptables-nft-1.8.8-3.amzn2023.0.2.x86_64          libcgroup-3.0-1.amzn2023.0.1.x86_64           libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64         libnftnl-1.2.2-2.amzn2023.0.2.x86_64          pigz-2.5-1.amzn2023.0.3.x86_64
  runc-1.1.13-1.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-89-37 ec2-user]#
```

Similarly do for all remaining instances.

Step 4: Execute the command **systemctl start docker** on all three machines to initiate the Docker service, enabling Docker container management and operation on each system.



```
Complete!
[root@ip-172-31-89-37 ec2-user]# systemctl start docker
```

Step 5: Set up Docker on all three machines to use systemd for handling cgroups by adjusting its settings. Make sure Docker starts up automatically when the machine boots. Follow these steps:

1. Update Docker's configuration file to use systemd.
2. Reload the systemd configuration.
3. Restart Docker to apply the new settings.

cat <<EOF | sudo tee /etc/docker/daemon.json

{

"exec-opts": ["native.cgroupdriver=systemd"]

}

EOFcd /etc/docker

sudo systemctl enable docker

sudo systemctl daemon-reload

sudo systemctl restart docker

Step6: Install kubernetes on all instances for this visit the website https://kubernetes.io/docs/reference/setup-tools/kubeadm/ or type **kubeadm** on the browser. Scroll down and select red hat-based distributions and follow the given scripts sequentially from the website

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

1)<u>Set SELinux to permissive mode:</u>

```
# Set SELinux in permissive mode (effectively disabling it)

sudo setenforce 0

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

2)<u>To create a repository for kubernetes:</u>

```
# This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/

enabled=1

gpgcheck=1

gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key

exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni

EOF
```

3)<u> Install kubelet, kubeadm and kubectl:</u>

```
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

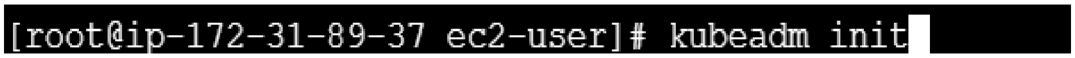
4) <u>Enable the kubelet service before running kubeadm:</u>

```
sudo systemctl enable --now kubelet
```

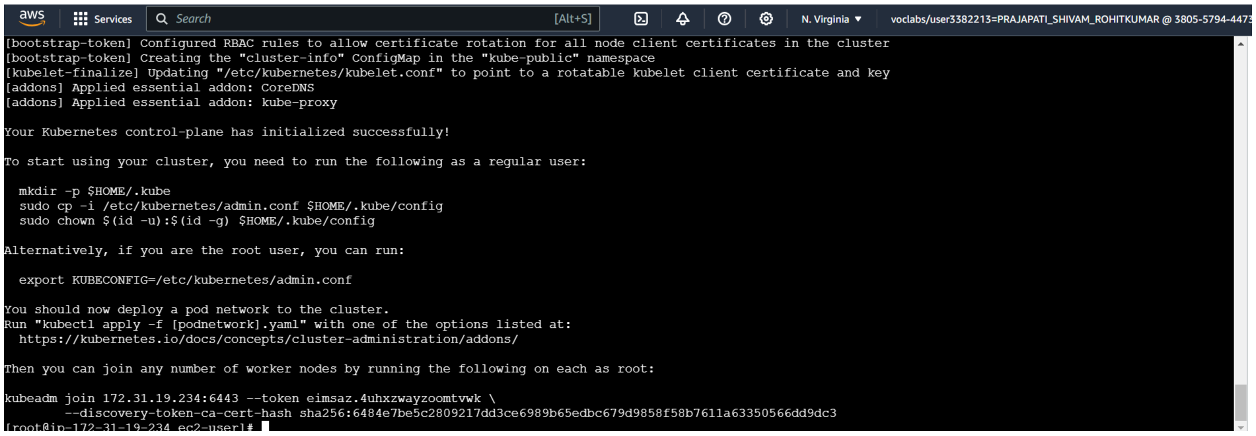
So this all steps will ultimately create a repository for kubernetes and will install necessary packages for the same.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Step 7: To initialize the Kubernetes cluster, run the **kubeadm init** command only on the master machine. If you're not logged in as the root user, make sure to use sudo with the command.This will set up the master node and generate the necessary configurations for the cluster.



```
[root@ip-172-31-89-37 ec2-user]# kubeadm init
```

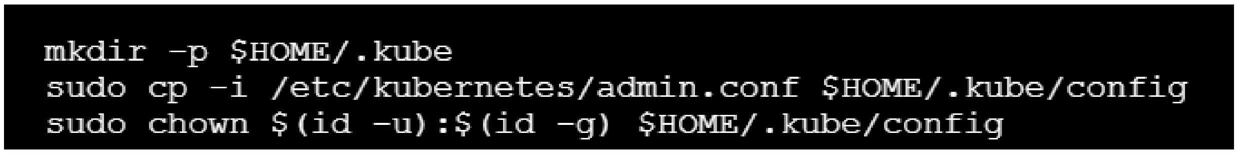


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3382213=PRAJAPATI_SHIVAM_ROHITKUMAR @ 3805-5794-4473

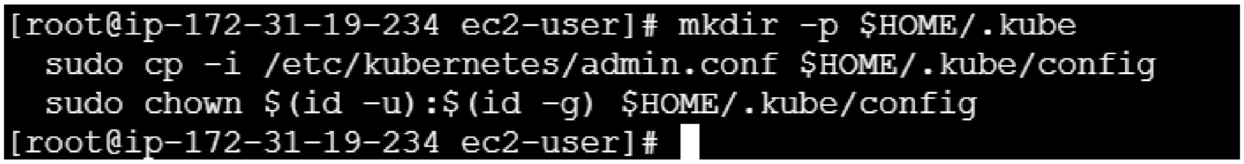
```
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

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

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

kubeadm join 172.31.19.234:6443 --token eimsaz.4uhxzwayzoomtvwk \
        --discovery-token-ca-cert-hash sha256:6484e7be5c2809217dd3ce6989b65edbc679d9858f58b7611a63350566dd9dc3
[root@ip-172-31-19-234 ec2-user]#
```

Step 8: After running the kubeadm init command on the master machine, you'll see instructions in the output, including commands for setting up the necessary directories and permissions. Look for commands related to mkdir and chown in the output.

Copy and execute these commands to properly configure the Kubernetes setup. They typically look something like this:



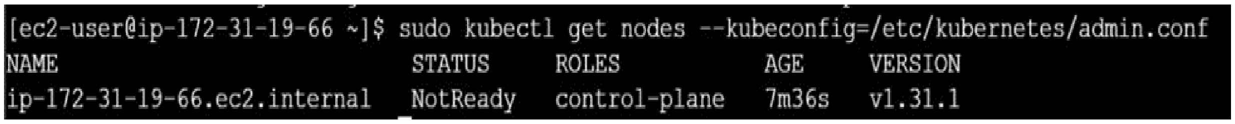
```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```



```
[root@ip-172-31-19-234 ec2-user]# mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@ip-172-31-19-234 ec2-user]#
```

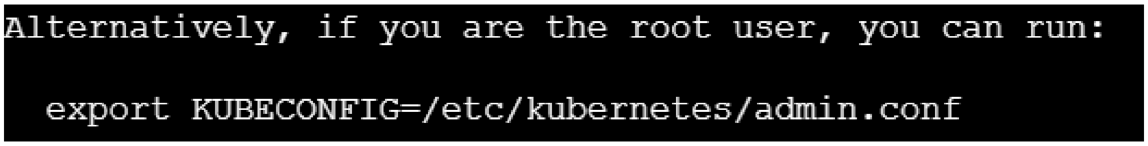
Step9: Run the command **sudo kubectl get nodes --kubeconfig=/etc/kubernetes/admin.conf** to list the nodes in the Kubernetes cluster; initially, you will see only the master machine listed.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**



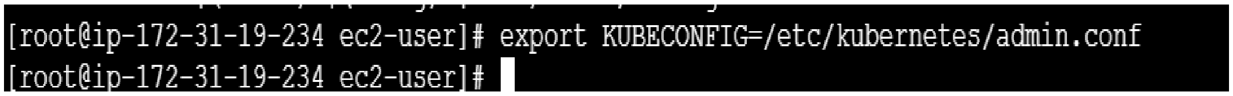
```
[ec2-user@ip-172-31-19-66 ~]$ sudo kubectl get nodes --kubeconfig=/etc/kubernetes/admin.conf
NAME                          STATUS     ROLES           AGE     VERSION
ip-172-31-19-66.ec2.internal  NotReady   control-plane   7m36s   v1.31.1
```

Step 10: Copy the sudo kubeadm join command snippet from the output of the sudo kubeadm init command, which looks like:

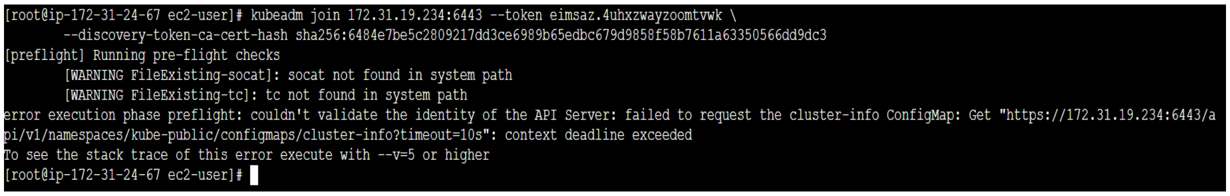


```
Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf
```

Execute this command on each worker machine to add them to the existing Kubernetes cluster. This process connects the worker nodes to the master node specified by <ip>, using the provided token and certificate hash for secure authentication.



```
[root@ip-172-31-19-234 ec2-user]# export KUBECONFIG=/etc/kubernetes/admin.conf
[root@ip-172-31-19-234 ec2-user]#
```



```
[root@ip-172-31-24-67 ec2-user]# kubeadm join 172.31.19.234:6443 --token eimsaz.4uhxzwayzoomtvwk \
        --discovery-token-ca-cert-hash sha256:6484e7be5c2809217dd3ce6989b65edbc679d9858f58b7611a63350566dd9dc3
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
error execution phase preflight: couldn't validate the identity of the API Server: failed to request the cluster-info ConfigMap: Get "https://172.31.19.234:6443/a
pi/v1/namespaces/kube-public/configmaps/cluster-info?timeout=10s": context deadline exceeded
To see the stack trace of this error execute with --v=5 or higher
[root@ip-172-31-24-67 ec2-user]#
```

On executing the previously mentioned commands, it is observed that the above error occurs on both the worker machines. This is due to the inability of the worker machines to get added to the kubernetes cluster within a deadline (a certain fixed amount of time)

Conclusion:

In the experiment, we learned how to install and set up a Kubernetes cluster on Linux machines or cloud platforms. Initially, we launched three EC2 Amazon Linux instances on AWS—one as the master node and two as worker nodes—and connected them to a remote server via SSH. We then installed, configured, and started Docker on all three machines. Following that, Kubernetes was installed on each machine. We proceeded by initializing Kubernetes only on the master machine, successfully adding it to the cluster. However, when we attempted to add the worker machines to the cluster using the join command, we encountered an error because the worker nodes failed to join the cluster within the specified deadline.