Name: Atharva Patil Div: D15C Roll No: 39

Advanced DevOps Lab
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 profoundly changed the way development and operations teams test and deploy modern software. Containers help companies modernize by making it easier to scale and deploy applications, but containers have also introduced new challenges and more complexity by creating an entirely new infrastructure ecosystem.

Large and small software companies alike are now deploying thousands of container instances daily, and that's a complexity of scale they have to manage. So how do they do it?

Enter the age of Kubernetes.

Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. In fact, Kubernetes has established itself as the defacto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation (CNCF), backed by key players like Google, AWS, Microsoft, IBM, Intel, Cisco, and Red Hat.

Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage the following activities:

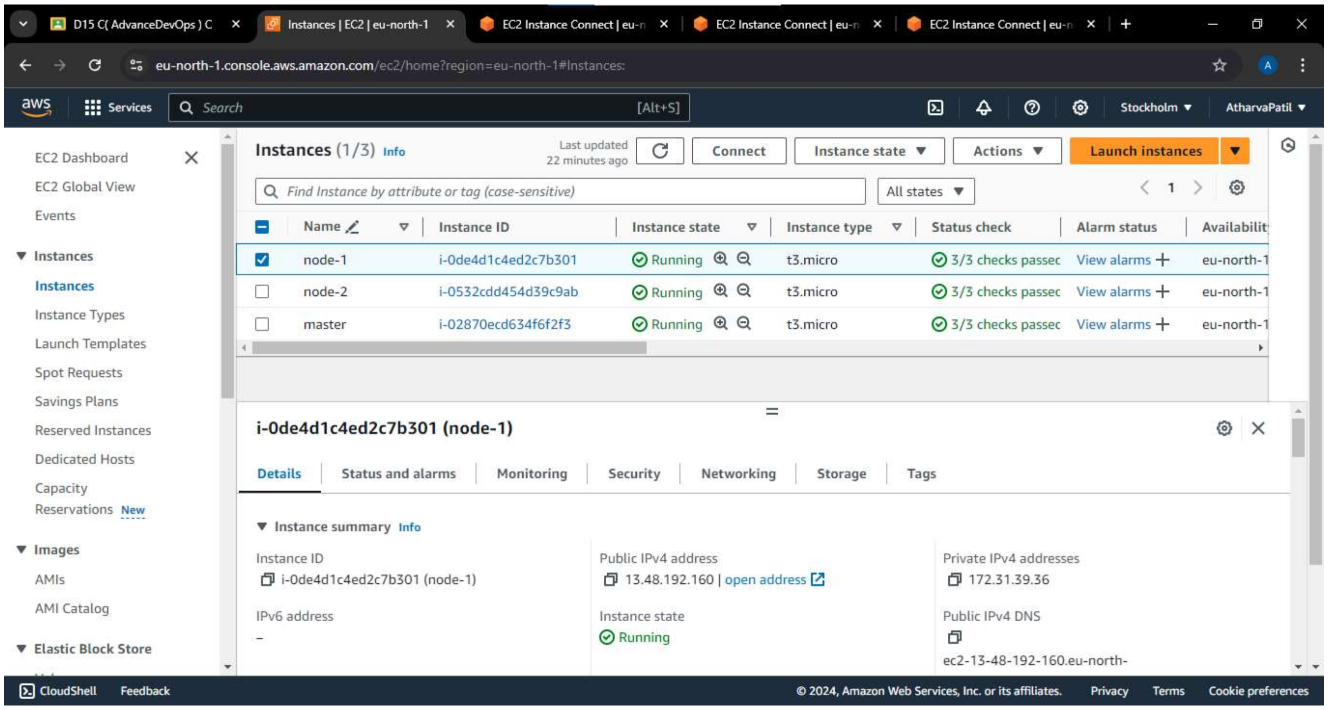
- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again
- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies
- Automatically leveraging additional resources made available when a new host is added to the cluster
- Easily performing canary deployments and rollbacks

Name: Atharva Patil Div: D15C Roll No: 39

Steps:

1. Create 3 EC2 Ubuntu Instances on AWS.

   (Name 1 as Master, the other 2 as worker-1 and worker-2)



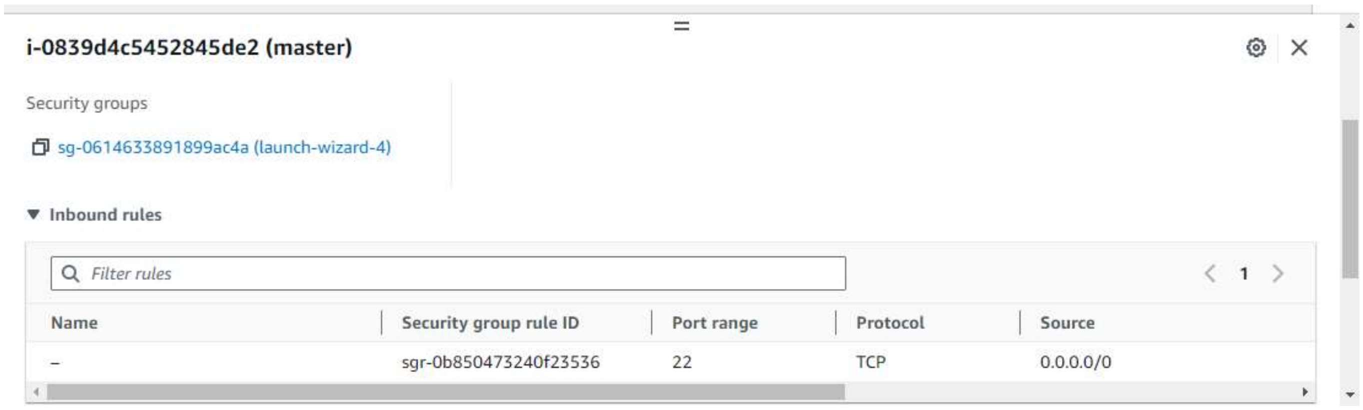
Instances (1/3)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availabilit
node-1	i-0de4d1c4ed2c7b301	Running	t3.micro	3/3 checks passec	View alarms	eu-north-1
node-2	i-0532cdd454d39c9ab	Running	t3.micro	3/3 checks passec	View alarms	eu-north-1
master	i-02870ecd634f6f2f3	Running	t3.micro	3/3 checks passec	View alarms	eu-north-1

i-0de4d1c4ed2c7b301 (node-1)

Instance ID: i-0de4d1c4ed2c7b301 (node-1)
Public IPv4 address: 13.48.192.160 | open address
Private IPv4 addresses: 172.31.39.36
IPv6 address: –
Instance state: Running
Public IPv4 DNS: ec2-13-48-192-160.eu-north-

2. Edit the Security Group Inbound Rules to allow SSH



i-0839d4c5452845de2 (master)

Security groups: sg-0614633891899ac4a (launch-wizard-4)

Inbound rules

Name	Security group rule ID	Port range	Protocol	Source
–	sgr-0b850473240f23536	22	TCP	0.0.0.0/0

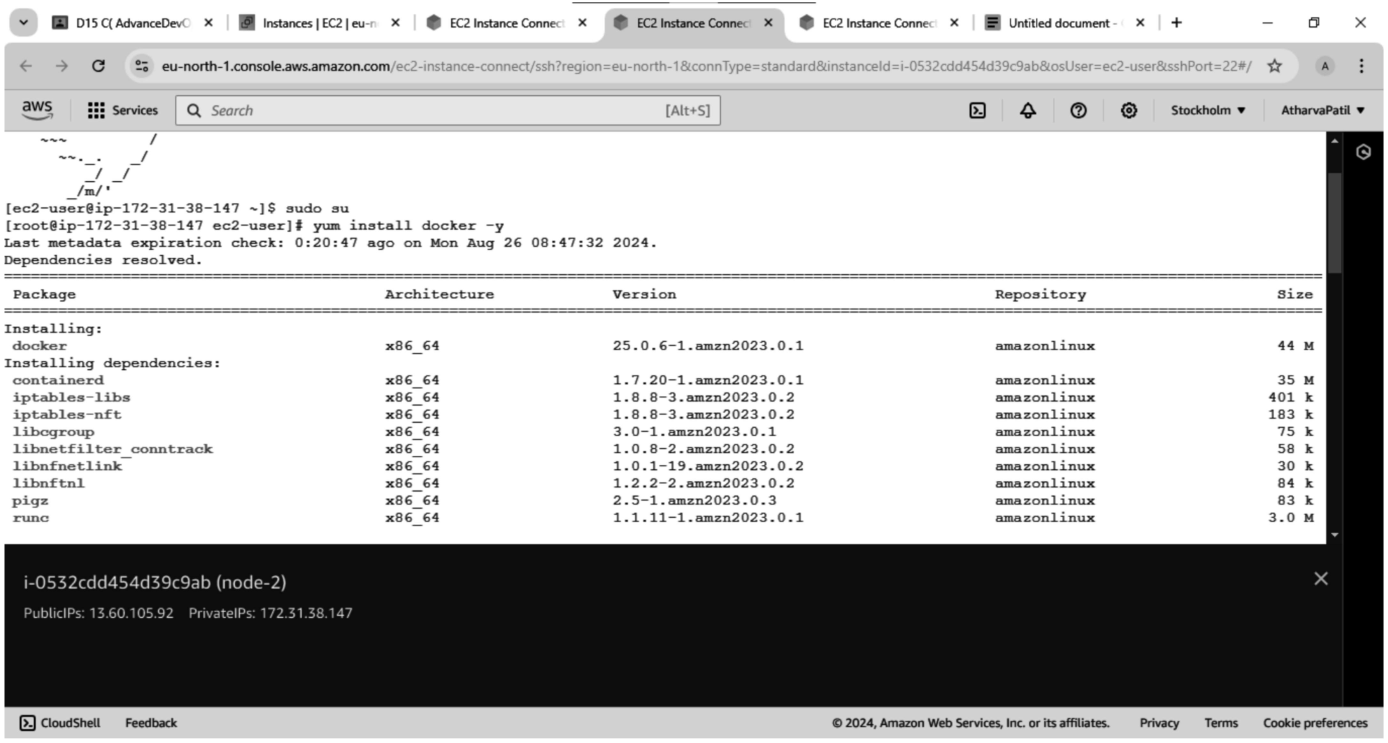
3. SSH into all 3 machines

   ssh -i <keyname>.pem ubuntu@<public_ip_address>

4. From now on, until mentioned, perform these steps on all 3 machines.

Install Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install -y docker-ce
```



```
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-38-147 ~]$ sudo su
[root@ip-172-31-38-147 ec2-user]# yum install docker -y
Last metadata expiration check: 0:20:47 ago on Mon Aug 26 08:47:32 2024.
Dependencies resolved.
================================================================================================================================
 Package                        Architecture          Version                             Repository                     Size
================================================================================================================================
Installing:
 docker                         x86_64                25.0.6-1.amzn2023.0.1               amazonlinux                    44 M
Installing dependencies:
 containerd                     x86_64                1.7.20-1.amzn2023.0.1               amazonlinux                    35 M
 iptables-libs                  x86_64                1.8.8-3.amzn2023.0.2                amazonlinux                   401 k
 iptables-nft                   x86_64                1.8.8-3.amzn2023.0.2                amazonlinux                   183 k
 libcgroup                      x86_64                3.0-1.amzn2023.0.1                  amazonlinux                    75 k
 libnetfilter_conntrack         x86_64                1.0.8-2.amzn2023.0.2                amazonlinux                    58 k
 libnfnetlink                   x86_64                1.0.1-19.amzn2023.0.2               amazonlinux                    30 k
 libnftnl                       x86_64                1.2.2-2.amzn2023.0.2                amazonlinux                    84 k
 pigz                           x86_64                2.5-1.amzn2023.0.3                  amazonlinux                    83 k
 runc                           x86_64                1.1.11-1.amzn2023.0.1               amazonlinux                   3.0 M
```

i-0532cdd454d39c9ab (node-2)

PublicIPs: 13.60.105.92 PrivateIPs: 172.31.38.147

eu-north-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=eu-north-1&connType=standard&instanceId=i-0532cdd454d39c9ab&osUser=ec2-user&sshPort=22#/
Services
Search [Alt+S]
Stockholm
AtharvaPatil
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

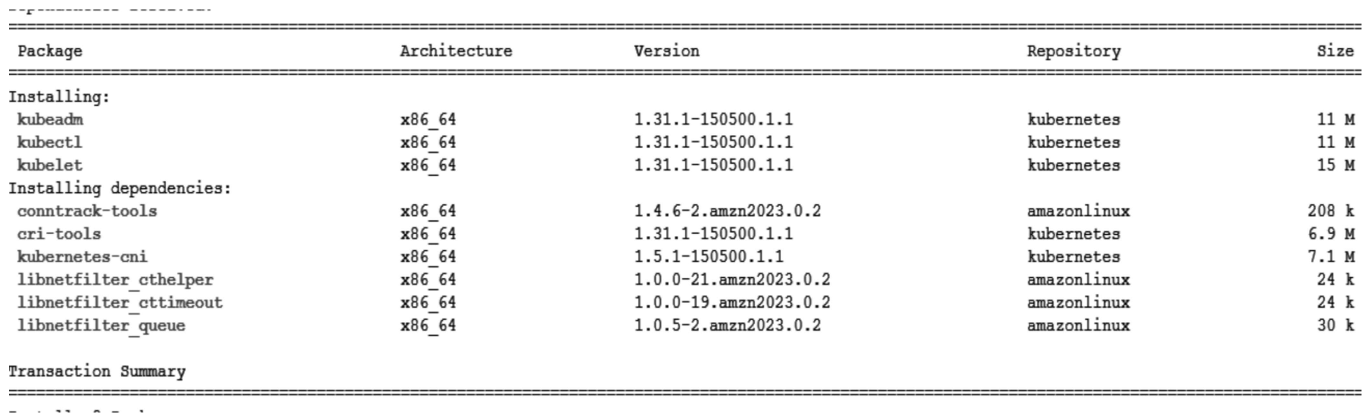
Then, configure cgroup in a daemon.json file.

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart
```

```
docker
```

Install Kubernetes on all 3 machines

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo apt-key add -
cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
```



```
================================================================================================================
 Package                    Architecture      Version                          Repository               Size
================================================================================================================
Installing:
 kubeadm                    x86_64            1.31.1-150500.1.1                kubernetes               11 M
 kubectl                    x86_64            1.31.1-150500.1.1                kubernetes               11 M
 kubelet                    x86_64            1.31.1-150500.1.1                kubernetes               15 M
Installing dependencies:
 conntrack-tools            x86_64            1.4.6-2.amzn2023.0.2             amazonlinux             208 k
 cri-tools                  x86_64            1.31.1-150500.1.1                kubernetes              6.9 M
 kubernetes-cni             x86_64            1.5.1-150500.1.1                 kubernetes              7.1 M
 libnetfilter_cthelper      x86_64            1.0.0-21.amzn2023.0.2            amazonlinux              24 k
 libnetfilter_cttimeout     x86_64            1.0.0-19.amzn2023.0.2            amazonlinux              24 k
 libnetfilter_queue         x86_64            1.0.5-2.amzn2023.0.2             amazonlinux              30 k

Transaction Summary
================================================================================================================
```

After installing Kubernetes, we need to configure internet options to allow bridging.

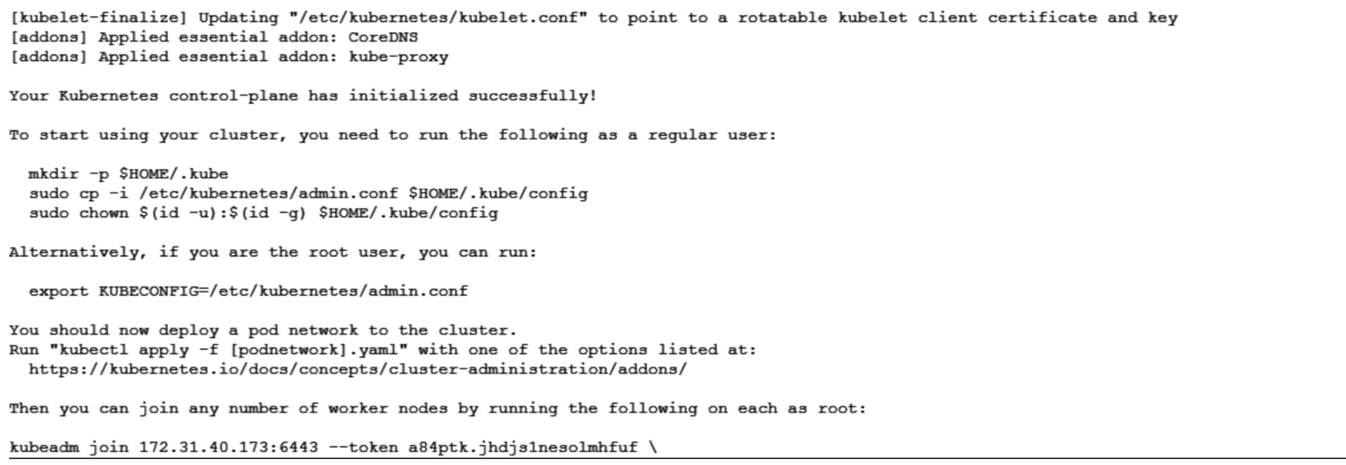
```
        sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
        sudo sysctl -p
```

5. Perform this **ONLY on the Master machine**

Initialize the Kubecluster

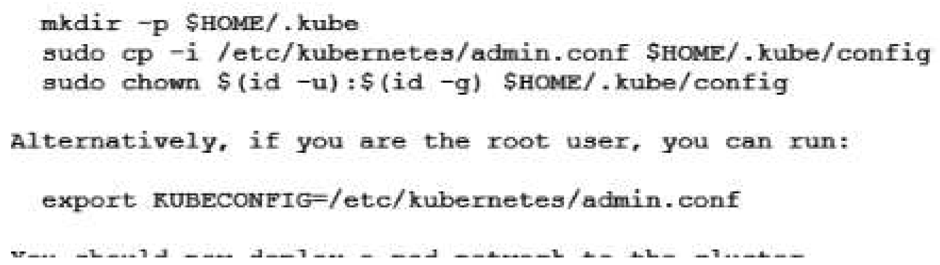
```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

--ignore-preflight-errors=all Copy the join command and keep it in a notepad, we'll need it later.



```
[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 172.31.40.173:6443 --token a84ptk.jhdjs1nesolmhfuf \
```

Copy the mkdir and chown commands from the top and execute them



```
  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
```

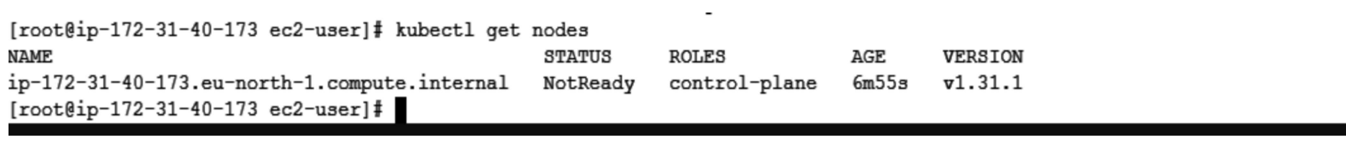
Then, add a common networking plugin called flammel file as mentioned in the code.

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
k ube-flannel.yml
```

Check the created pod using this command

Now, keep a watch on all nodes using the following command

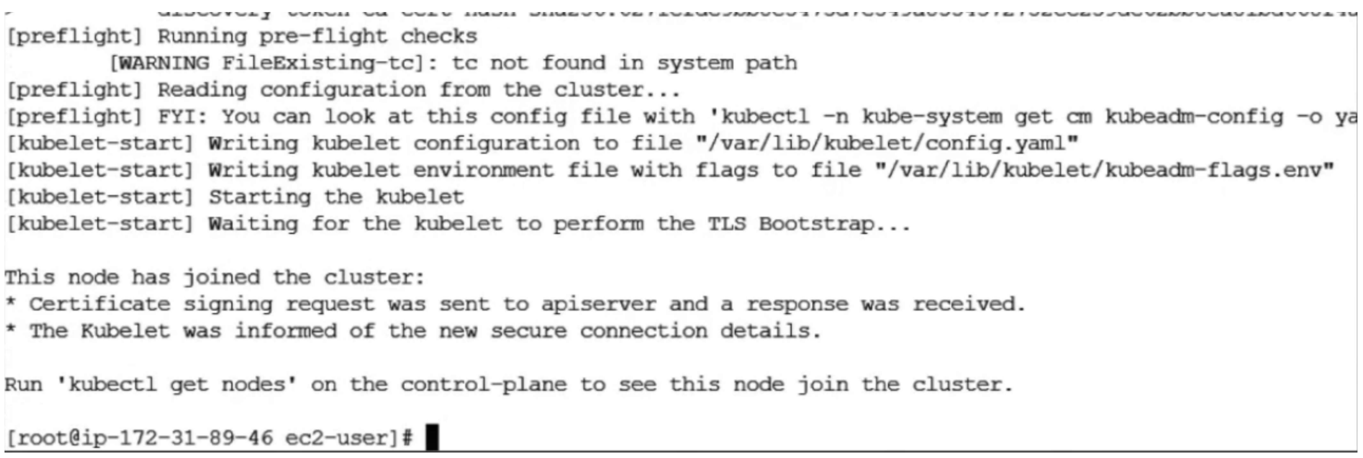
```
watch kubectl get nodes
```



```
[root@ip-172-31-40-173 ec2-user]# kubectl get nodes
NAME                                         STATUS     ROLES           AGE     VERSION
ip-172-31-40-173.eu-north-1.compute.internal NotReady   control-plane   6m55s   v1.31.1
[root@ip-172-31-40-173 ec2-user]#
```

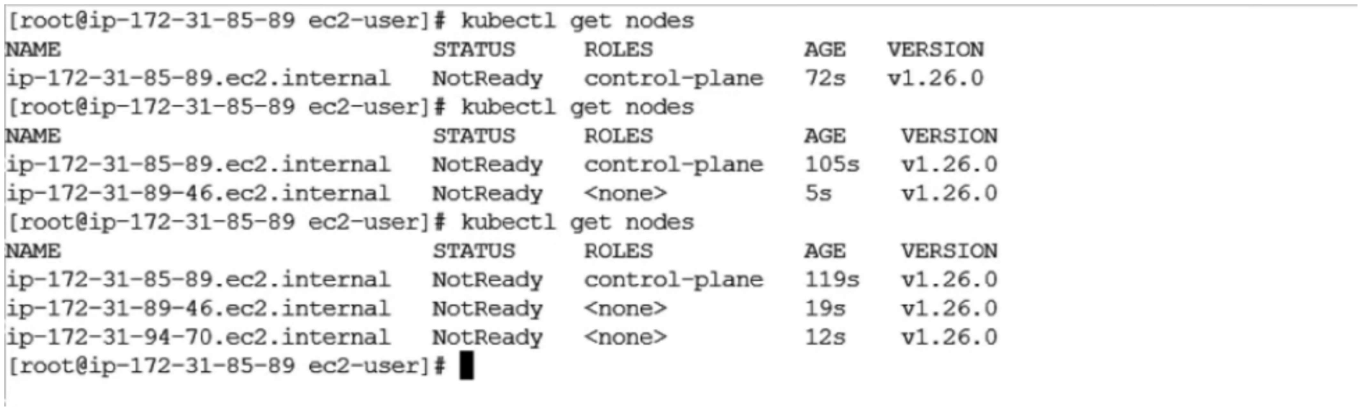
6. Perform this **ONLY on the worker machines**

```
sudo kubeadm join <ip> --token <token> \
        --discovery-token-ca-cert-hash <hash>
```



```
[preflight] Running pre-flight checks
        [WARNING FileExisting-tc]: tc not found in system path
[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 ya
[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-start] Waiting for the kubelet to perform the TLS Bootstrap...

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.

[root@ip-172-31-89-46 ec2-user]#
```

Now, notice the changes on the master terminal



```
[root@ip-172-31-85-89 ec2-user]# kubectl get nodes
NAME                           STATUS     ROLES           AGE   VERSION
ip-172-31-85-89.ec2.internal   NotReady   control-plane   72s   v1.26.0
[root@ip-172-31-85-89 ec2-user]# kubectl get nodes
NAME                           STATUS     ROLES           AGE    VERSION
ip-172-31-85-89.ec2.internal   NotReady   control-plane   105s   v1.26.0
ip-172-31-89-46.ec2.internal   NotReady   <none>          5s     v1.26.0
[root@ip-172-31-85-89 ec2-user]# kubectl get nodes
NAME                           STATUS     ROLES           AGE    VERSION
ip-172-31-85-89.ec2.internal   NotReady   control-plane   119s   v1.26.0
ip-172-31-89-46.ec2.internal   NotReady   <none>          19s    v1.26.0
ip-172-31-94-70.ec2.internal   NotReady   <none>          12s    v1.26.0
[root@ip-172-31-85-89 ec2-user]#
```

That's it, we now have a Kubernetes cluster running across 3 AWS EC2 Instances. This cluster can be used to further deploy applications and their loads being distributed across these machines.

Conclusion: Thus we have understood the Kubernetes cluster architecture and have successfully created Kubernetes cluster on a linux machine with the help of AWS by creating three EC2 instances one master and two nodes, installed docker and kubernetes on all three machines and then initialized a kubernetes control-plane node on the master and then added the worker nodes to the cluster to distribute the workload.