PRACTICAL NO: - 3

Case Study: Understanding Kubernetes Cluster Architecture

Introduction: -

Kubernetes is an open-source platform used to deploy and maintain a group of containers in a virtualized environment. In practice, Kubernetes is most commonly used alongside Docker for better control and implementation of containerized applications. Containerized applications "bundle" applications together with all its files, libraries, and packages required for it to run reliably and efficiently on different platforms. However, it operates at the container level rather than at the hardware level.

The name Kubernetes is derived from a Greek term meaning 'helmsman' or 'pilot.' True to this word, Kubernetes provides the guiding force for developer platforms to transition from virtual machines (VMs) to containers and the statically scheduled to the dynamically scheduled. This means no more manual integration and configuration when you move from a testing environment to an actual production environment or from on-premise to the cloud! The Kubernetes logical compute environment offers common services to all the applications in the cluster as part of the ecosystem for the software to run consistently.

Features of Kubernetes

- Automates various manual processes and controls server hosting and launching
- Manages containers offer security, and networking and storage services
- Monitors and continuously checks the health of nodes and containers
- Automates rollback for changes that go wrong
- Mounts and adds a storage system to run apps

Purpose of Kubernetes

The primary purpose of Kubernetes is to enable developers to write and deploy applications that can run seamlessly across multiple operating environments. Traditionally, application performance and deployment were tightly coupled to specific infrastructures, often requiring adherence to cloud provider-specific constructs and back-end storage systems. This dependency resulted in infrastructure lock-in, limiting flexibility and scalability

Kubernetes addresses this challenge by abstracting the underlying infrastructure, allowing developers to deploy cloud-native applications in containers without restrictions. This means applications can be managed and scaled consistently across different environments—be it in the cloud, on-premises, or in hybrid setups—providing true infrastructure independence and operational flexibility.

Working of Kubernetes

Before exploring how Kubernetes operates, it’s essential to grasp the concept of containers and their significance in modern application development. A container is a small, lightweight virtual machine (VM) that does not have device drivers and shares its operating system among the applications. It is a good way to bundle and run applications in a production environment. However, you need to manage these containers in a proper way so that there is no downtime. This is where Kubernetes comes to the rescue.

Kubernetes works as a “container orchestration system” that manages the lifecycle of containerized applications and automates the deployment of several containers. Containers running the same applications are usually grouped together into Pods. There can be one or multiple containers in a single Pod and each of them shares the same IP address and resources such as memory and storage. By grouping the containers in this manner, Kubernetes eliminates the need to cram multiple functionalities in one single container. There is a dedicated container orchestrator which supervises these groups and ensures that they operate correctly.

Kubernetes in DevOps

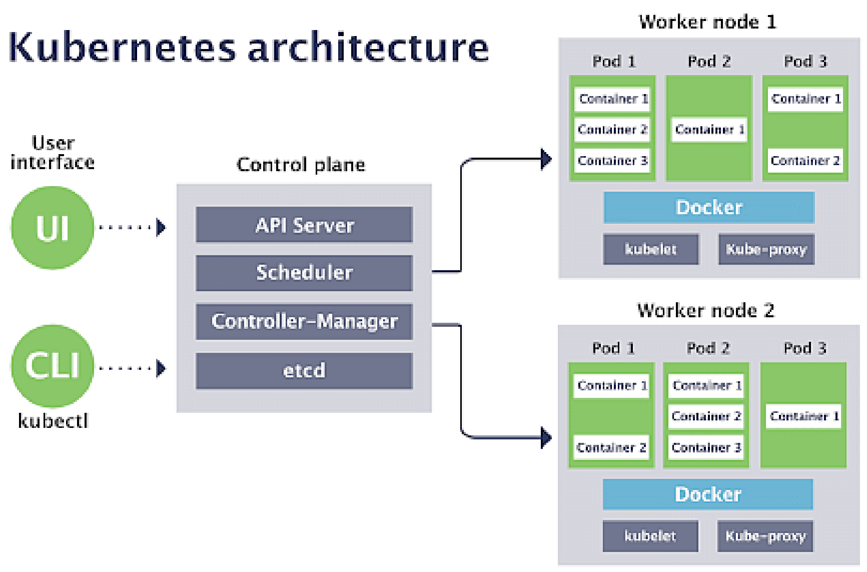
Kubernetes is more than just a container orchestration tool; it's a powerful enabler of DevOps practices. By bridging the gap between IT operations and development, Kubernetes fosters a collaborative DevOps environment, ensuring that software and its dependencies are shared seamlessly across different environments.

Kubernetes facilitates various stages of the software lifecycle, enhancing the build-test-deploy timeline:

- **Developer Environment:** Helps run software consistently in any setting, ensuring that applications behave the same across different environments.
- **QA/Testing Process:** Coordinates pipelines between test and production environments, streamlining the testing process.
- **Sys-Admin:** Once configured, Kubernetes runs anything, simplifying system administration tasks.
- **Operations:** Provides a comprehensive solution for building, shipping, and scaling software, making operations smoother and more efficient.

Kubernetes Cluster Architecture

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. The cluster is managed by a control plane, which is responsible for maintaining the desired state of the cluster, such as which applications are running and where they are running.



Control Plane: The control plane is the brain of the Kubernetes cluster, responsible for managing the desired state of the cluster, making decisions on scheduling, and responding to cluster events.

- **API Server:** The API server acts as the front-end for the Kubernetes control plane. It exposes the Kubernetes API, which is used by both internal components and external users (via CLI or UI) to communicate with the cluster.
- **Scheduler:** The scheduler is responsible for assigning newly created pods to nodes in the cluster. It evaluates the resource requirements of the pods against the available resources on the nodes, ensuring optimal placement.
- **Controller Manager:** This component runs various controllers that manage the state of the cluster. For example, it ensures that the number of pod replicas matches the desired configuration and handles node failures.
- **etcd:** A key-value store that holds all the configuration data for the Kubernetes cluster, including the current state and the desired state of the objects in the cluster. It is essential for maintaining cluster consistency and recovery.

Worker Nodes: Worker nodes are the machines where the application workloads run. Each worker node contains the services necessary to run pods and communicate with the control plane.

- **Kubelet:** The kubelet is an agent that runs on each worker node. It ensures that containers are running in a pod as expected by the control plane. It communicates with the API server to receive instructions and report back the status of the node and its workloads.
- **Kube-proxy:** Kube-proxy is a network proxy that runs on each worker node. It manages the networking for the pods, ensuring that each pod can communicate with others, both within and outside the cluster.
- **Docker (or other container runtimes):** Docker is the container runtime that runs and manages the containers on the worker nodes. It pulls container images from a registry, starts and stops containers, and manages container storage and networking.

User Interfaces: Users interact with the Kubernetes cluster through two primary interfaces:

- **UI (User Interface):** A graphical interface that provides an easy way to manage and monitor the cluster.
- **CLI (Command-Line Interface, e.g., kubectl):** A more powerful tool for managing the cluster, allowing users to interact with the API server directly via command-line commands.

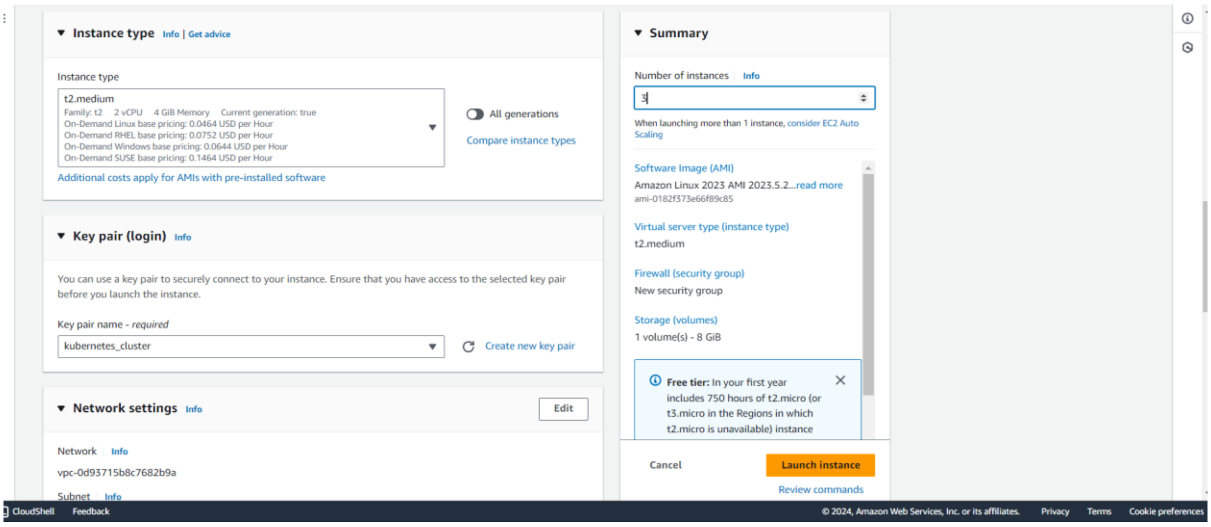
This architecture allows Kubernetes to abstract away the underlying infrastructure, providing a consistent and scalable environment for deploying and managing containerized applications across different environments.

Conclusion:-

In conclusion, Kubernetes is a powerful, open-source platform that automates the deployment and management of containerized applications. Its architecture ensures seamless operation across diverse environments, promoting flexibility and scalability. By abstracting the underlying infrastructure and supporting DevOps practices, Kubernetes enhances the software development lifecycle. This results in streamlined operations, reduced downtime, and consistent application performance.

Install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

- Create 3 EC2 instances, one for the master node and two for the worker nodes



Instance type

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)

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: kubernetes_cluster

Network settings

Network: vpc-0d93715b8c7682b9a

Summary

Number of instances: 3

When launching more than 1 instance, consider EC2 Auto Scaling

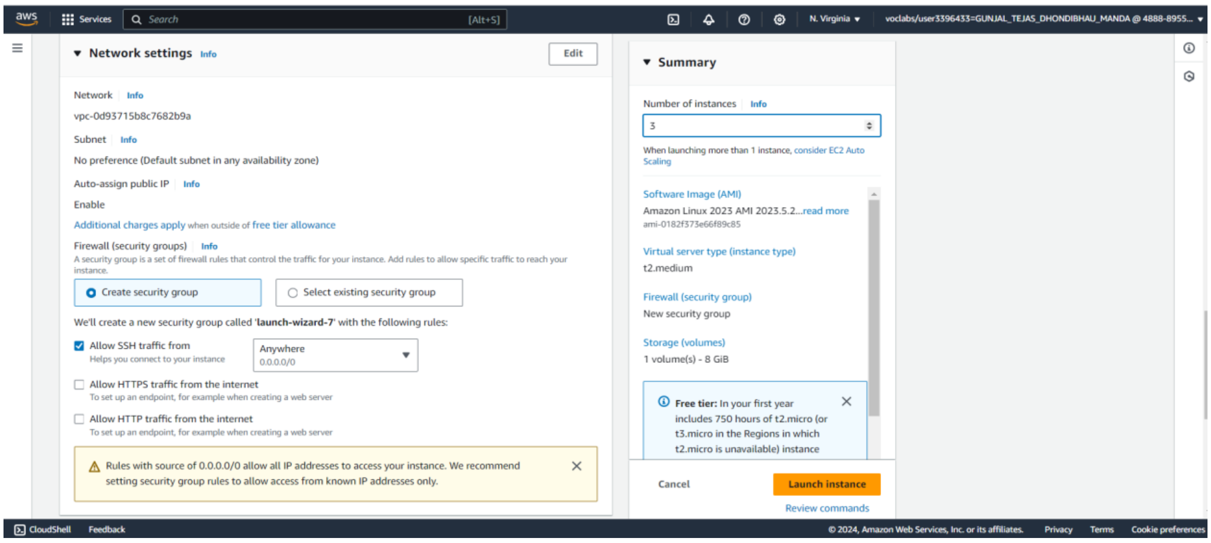
Software Image (AMI): Amazon Linux 2023 AMI 2023.5.2...read more
ami-0182f373e66f89c85

Virtual server type (instance type): t2.medium

Firewall (security group): New security group

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

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance



Network settings

Network: vpc-0d93715b8c7682b9a

Subnet: No preference (Default subnet in any availability zone)

Auto-assign public IP: Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)
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)

We'll create a new security group called 'launch-wizard-7' with the following rules:

☑ Allow SSH traffic from — Anywhere 0.0.0.0/0
Helps you connect to your instance

☐ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

☐ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Summary

Number of instances: 3

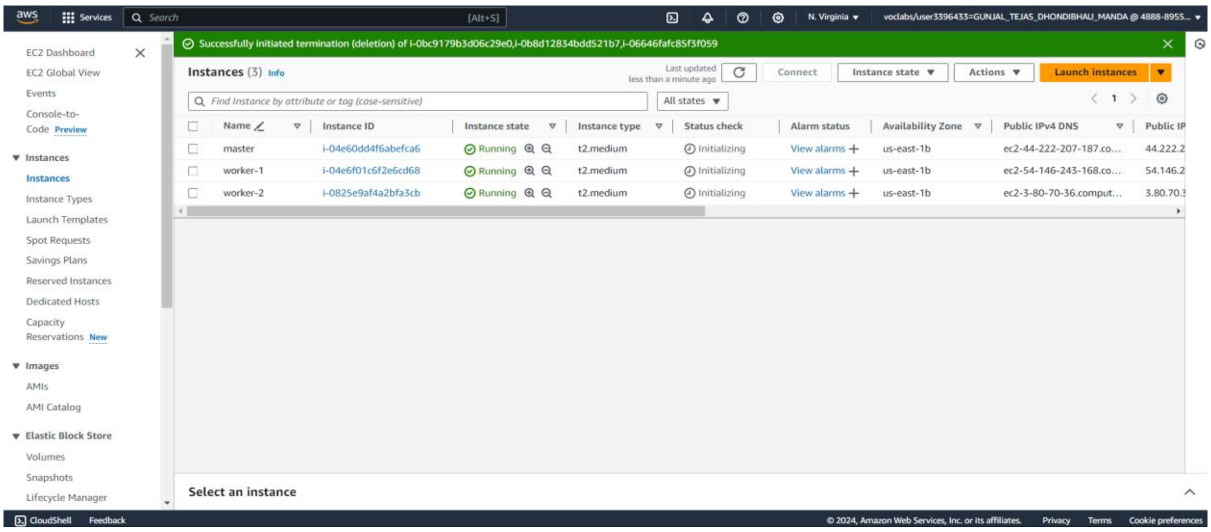
Software Image (AMI): Amazon Linux 2023 AMI 2023.5.2...read more
ami-0182f373e66f89c85

Virtual server type (instance type): t2.medium

Firewall (security group): New security group

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

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance



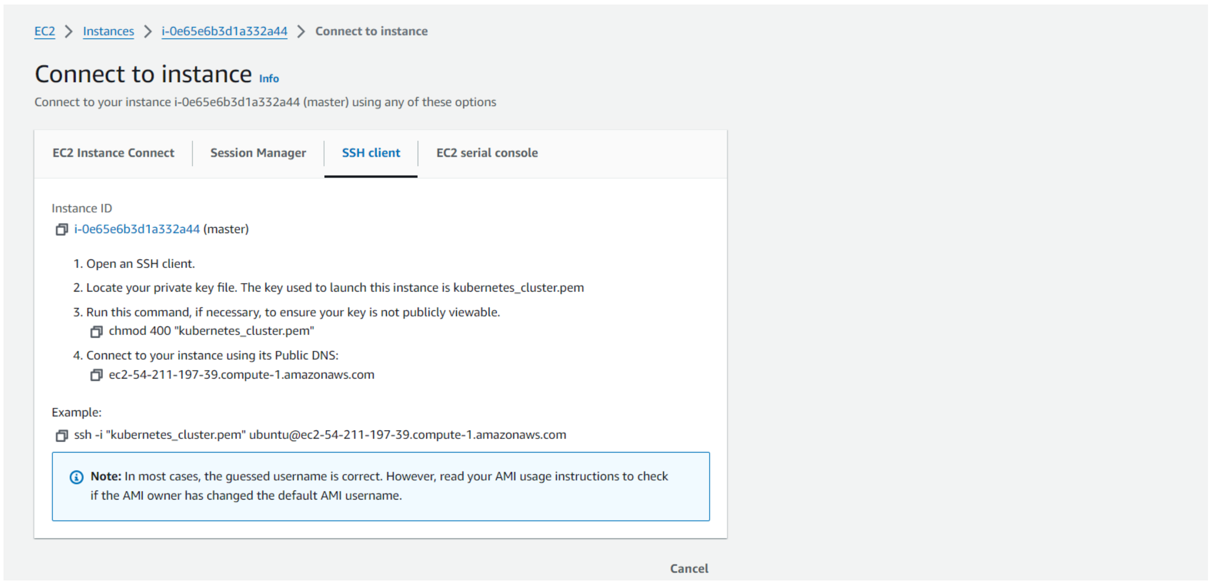
Successfully initiated termination (deletion) of i-0bc9179b3d06c29e0,i-0b8d12834bdd521b7,i-06646fafc85f3f059

Instances (3)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IP
master	i-04e60dd4f6abefca6	Running	t2.medium	Initializing	View alarms	us-east-1b	ec2-44-222-207-187.co...	44.222.2
worker-1	i-04e6f01c6f2e6cd68	Running	t2.medium	Initializing	View alarms	us-east-1b	ec2-54-146-243-168.co...	54.146.2
worker-2	i-0825e9af4a2bfa3cb	Running	t2.medium	Initializing	View alarms	us-east-1b	ec2-3-80-70-36.comput...	3.80.70.3

Select an instance

- After the instances have been created, copy the text given in the example part of each of the three instances into git bash.



EC2 > Instances > i-0e65e6b3d1a332a44 > Connect to instance

Connect to instance Info

Connect to your instance i-0e65e6b3d1a332a44 (master) using any of these options

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID

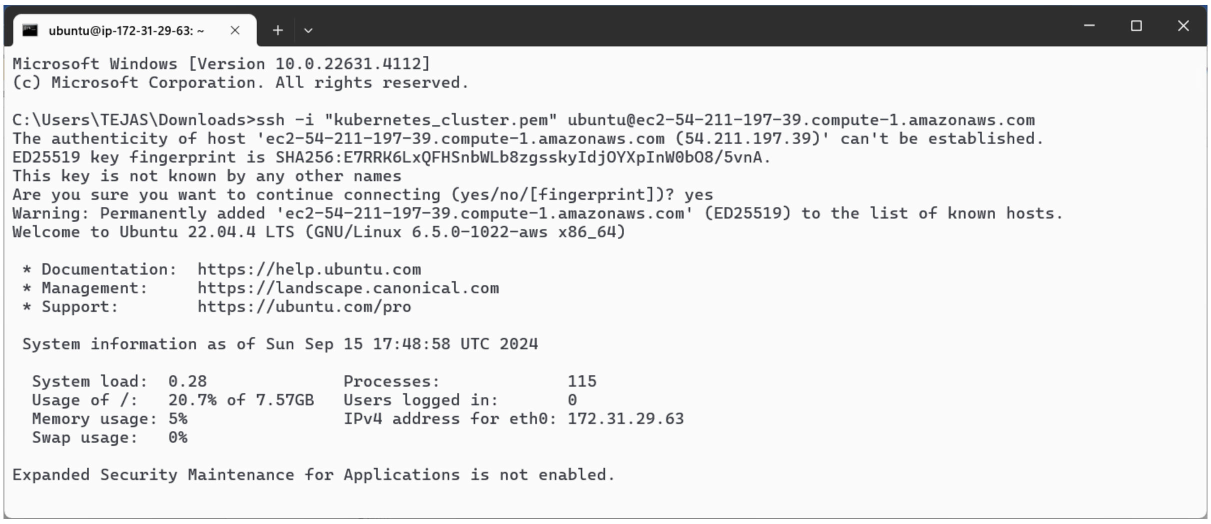
i-0e65e6b3d1a332a44 (master)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is kubernetes_cluster.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 "kubernetes_cluster.pem"
4. Connect to your instance using its Public DNS:
   ec2-54-211-197-39.compute-1.amazonaws.com

Example:

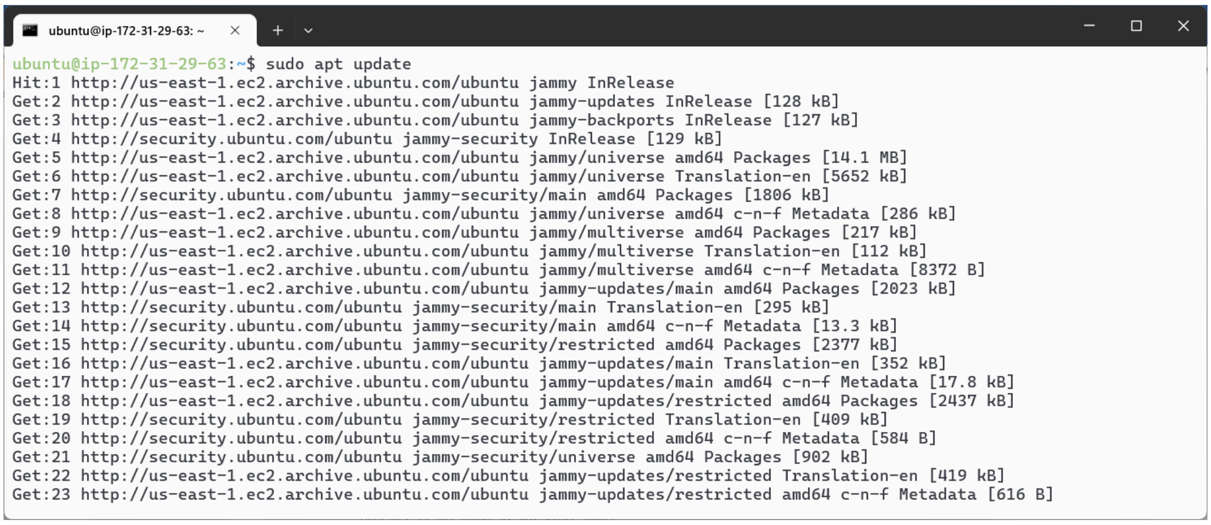
ssh -i "kubernetes_cluster.pem" ubuntu@ec2-54-211-197-39.compute-1.amazonaws.com

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.



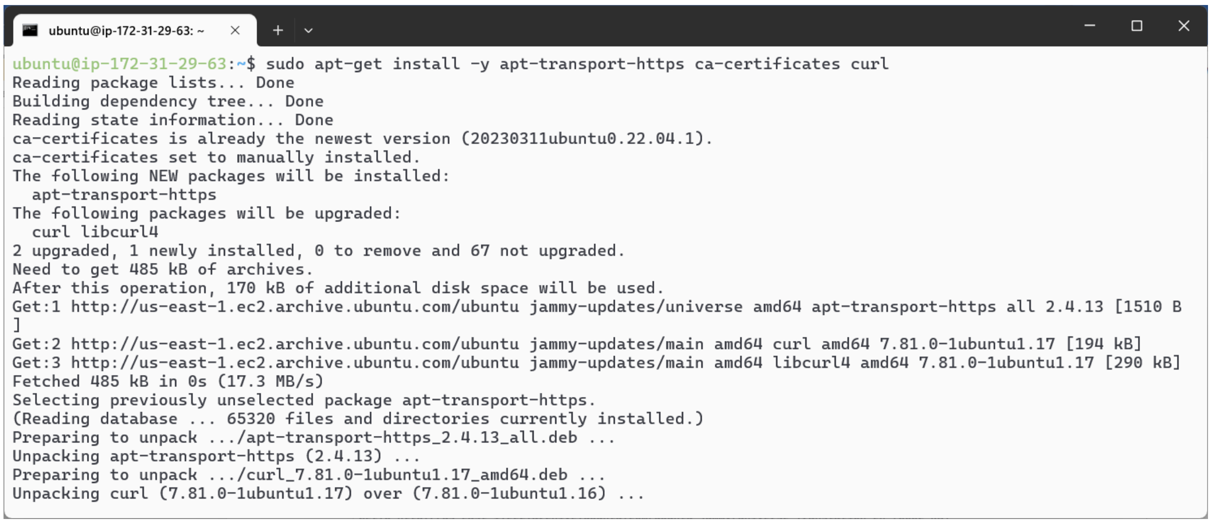
```
Microsoft Windows [Version 10.0.22631.4112]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TEJAS\Downloads>ssh -i "kubernetes_cluster.pem" ubuntu@ec2-54-211-197-39.compute-1.amazonaws.com
The authenticity of host 'ec2-54-211-197-39.compute-1.amazonaws.com (54.211.197.39)' can't be established.
ED25519 key fingerprint is SHA256:E7RRK6LxQFHSnbWLb8zgsskyIdjOYXpInW0bO8/5vnA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-211-197-39.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1022-aws x86_64)

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

 System information as of Sun Sep 15 17:48:58 UTC 2024

  System load:  0.28               Processes:             115
  Usage of /:   20.7% of 7.57GB    Users logged in:       0
  Memory usage: 5%                 IPv4 address for eth0: 172.31.29.63
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.
```

- Update the package manager on all nodes:



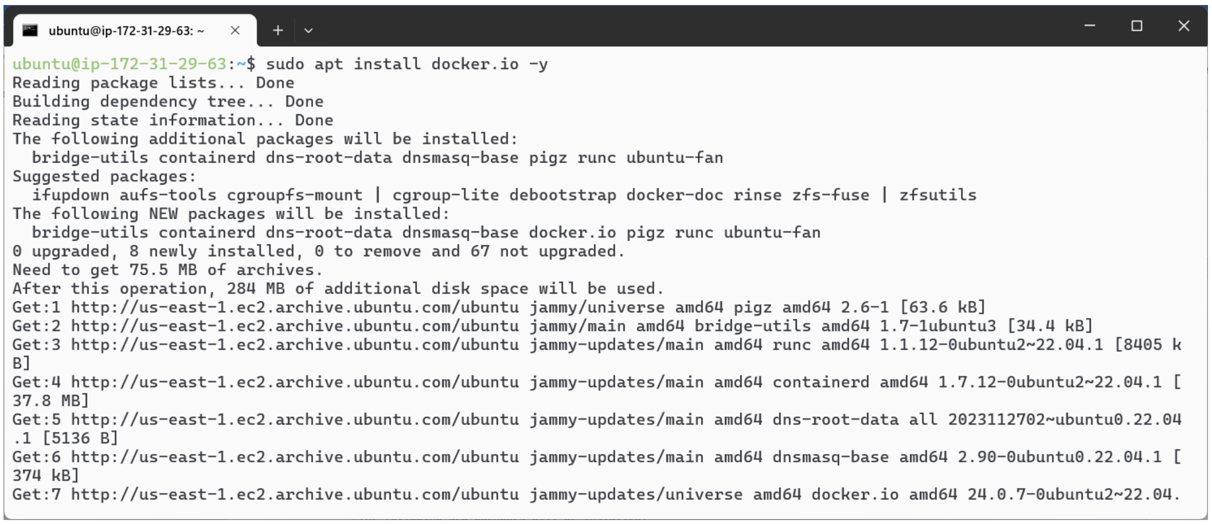
```
ubuntu@ip-172-31-29-63:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1806 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2023 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [295 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.3 kB]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2377 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [352 kB]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.8 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2437 kB]
Get:19 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [409 kB]
Get:20 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 c-n-f Metadata [584 B]
Get:21 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [902 kB]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [419 kB]
Get:23 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [616 B]
```

➢ Installing Required Packages for HTTPS and Certificate Transport on Ubuntu



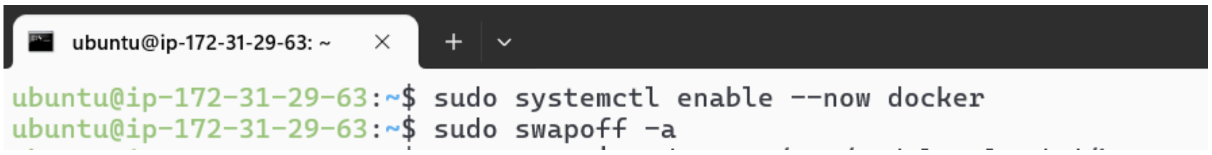
```
ubuntu@ip-172-31-29-63:~$ sudo apt-get install -y apt-transport-https ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
The following packages will be upgraded:
  curl libcurl4
2 upgraded, 1 newly installed, 0 to remove and 67 not upgraded.
Need to get 485 kB of archives.
After this operation, 170 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 apt-transport-https all 2.4.13 [1510 B
]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.17 [194 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libcurl4 amd64 7.81.0-1ubuntu1.17 [290 kB]
Fetched 485 kB in 0s (17.3 MB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 65320 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.4.13_all.deb ...
Unpacking apt-transport-https (2.4.13) ...
Preparing to unpack .../curl_7.81.0-1ubuntu1.17_amd64.deb ...
Unpacking curl (7.81.0-1ubuntu1.17) over (7.81.0-1ubuntu1.16) ...
```

➢ Installing Docker on Ubuntu



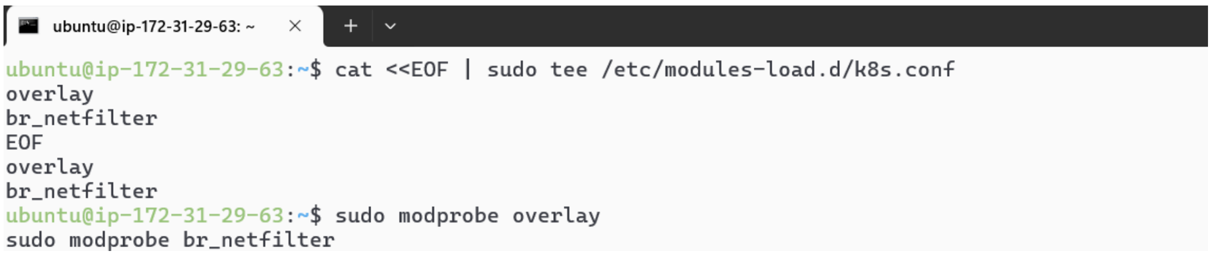
```
ubuntu@ip-172-31-29-63:~$ sudo apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 67 not upgraded.
Need to get 75.5 MB of archives.
After this operation, 284 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.12-0ubuntu2~22.04.1 [8405 k
B]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.7.12-0ubuntu2~22.04.1 [
37.8 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dns-root-data all 2023112702~ubuntu0.22.04
.1 [5136 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.90-0ubuntu0.22.04.1 [
374 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 docker.io amd64 24.0.7-0ubuntu2~22.04.
```

➢ Enabling Docker and Disabling Swap on Ubuntu



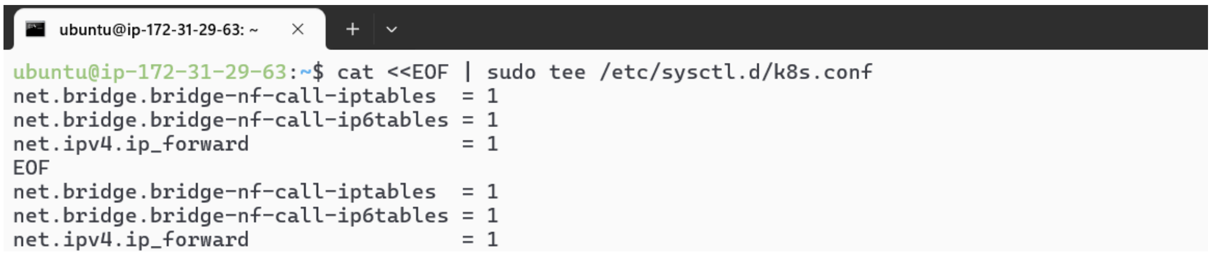
```
ubuntu@ip-172-31-29-63:~$ sudo systemctl enable --now docker
ubuntu@ip-172-31-29-63:~$ sudo swapoff -a
```

➢ Load necessary kernel modules for networking and iptables:

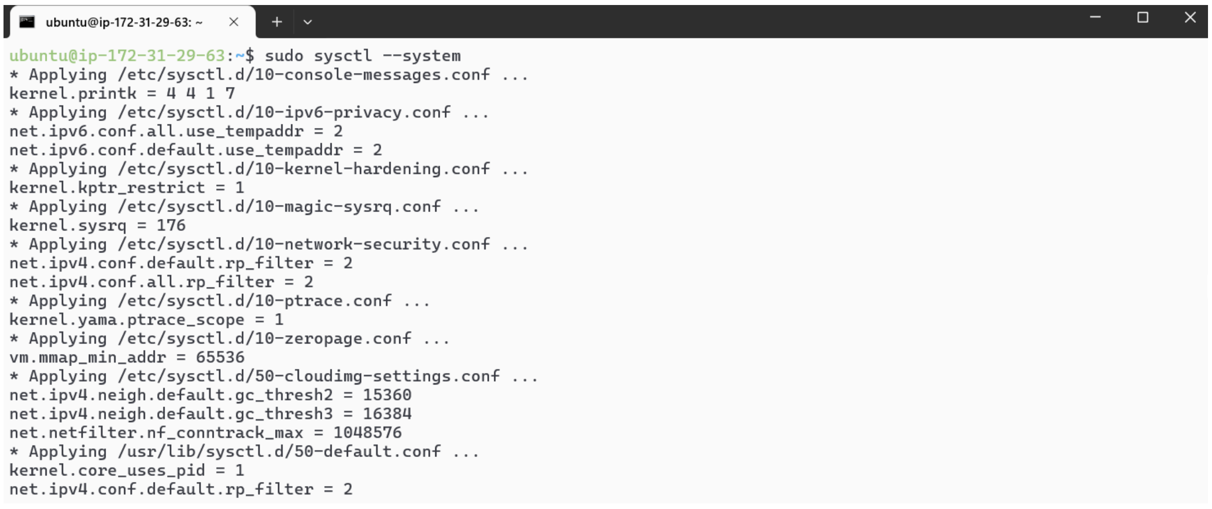


```
ubuntu@ip-172-31-29-63:~$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
overlay
br_netfilter
ubuntu@ip-172-31-29-63:~$ sudo modprobe overlay
sudo modprobe br_netfilter
```

➢ Configure sysctl settings for Kubernetes networking:

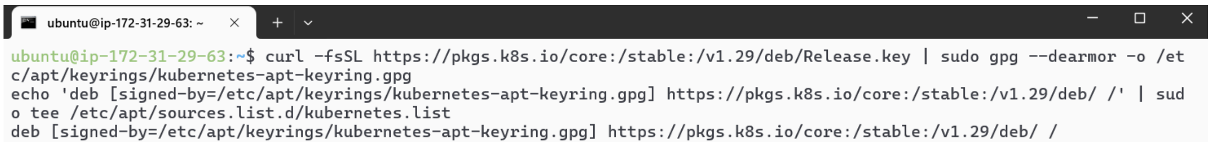


```
ubuntu@ip-172-31-29-63:~$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
```

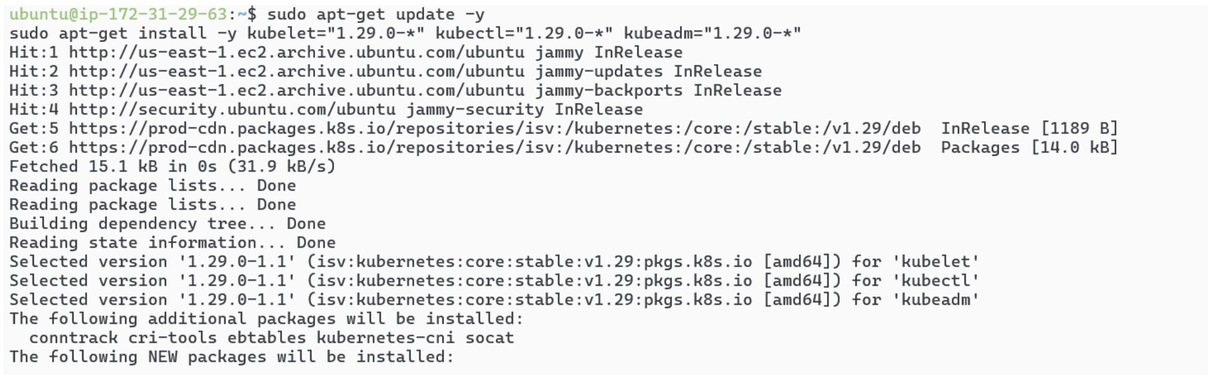


```
ubuntu@ip-172-31-29-63:~$ sudo sysctl --system
* Applying /etc/sysctl.d/10-console-messages.conf ...
kernel.printk = 4 4 1 7
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
kernel.kptr_restrict = 1
* Applying /etc/sysctl.d/10-magic-sysrq.conf ...
kernel.sysrq = 176
* Applying /etc/sysctl.d/10-network-security.conf ...
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.all.rp_filter = 2
* Applying /etc/sysctl.d/10-ptrace.conf ...
kernel.yama.ptrace_scope = 1
* Applying /etc/sysctl.d/10-zeropage.conf ...
vm.mmap_min_addr = 65536
* Applying /etc/sysctl.d/50-cloudimg-settings.conf ...
net.ipv4.neigh.default.gc_thresh2 = 15360
net.ipv4.neigh.default.gc_thresh3 = 16384
net.netfilter.nf_conntrack_max = 1048576
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 2
```

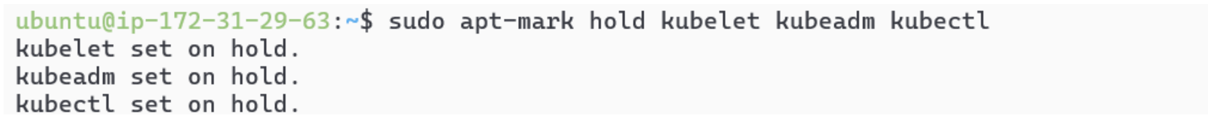
➢ Install Kubernetes tools on all nodes.



```
ubuntu@ip-172-31-29-63:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /et
c/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sud
o tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /
```



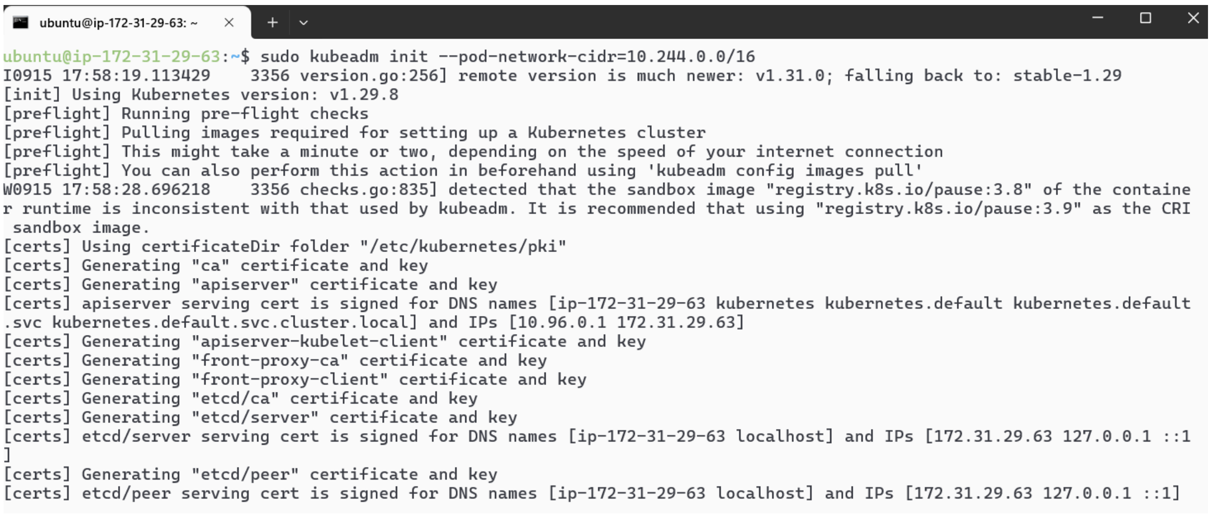
```
ubuntu@ip-172-31-29-63:~$ sudo apt-get update -y
sudo apt-get install -y kubelet="1.29.0-*" kubectl="1.29.0-*" kubeadm="1.29.0-*"
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:5 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.29/deb  InRelease [1189 B]
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.29/deb  Packages [14.0 kB]
Fetched 15.1 kB in 0s (31.9 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Selected version '1.29.0-1.1' (isv:kubernetes:core:stable:v1.29:pkgs.k8s.io [amd64]) for 'kubelet'
Selected version '1.29.0-1.1' (isv:kubernetes:core:stable:v1.29:pkgs.k8s.io [amd64]) for 'kubectl'
Selected version '1.29.0-1.1' (isv:kubernetes:core:stable:v1.29:pkgs.k8s.io [amd64]) for 'kubeadm'
The following additional packages will be installed:
  conntrack cri-tools ebtables kubernetes-cni socat
The following NEW packages will be installed:
```



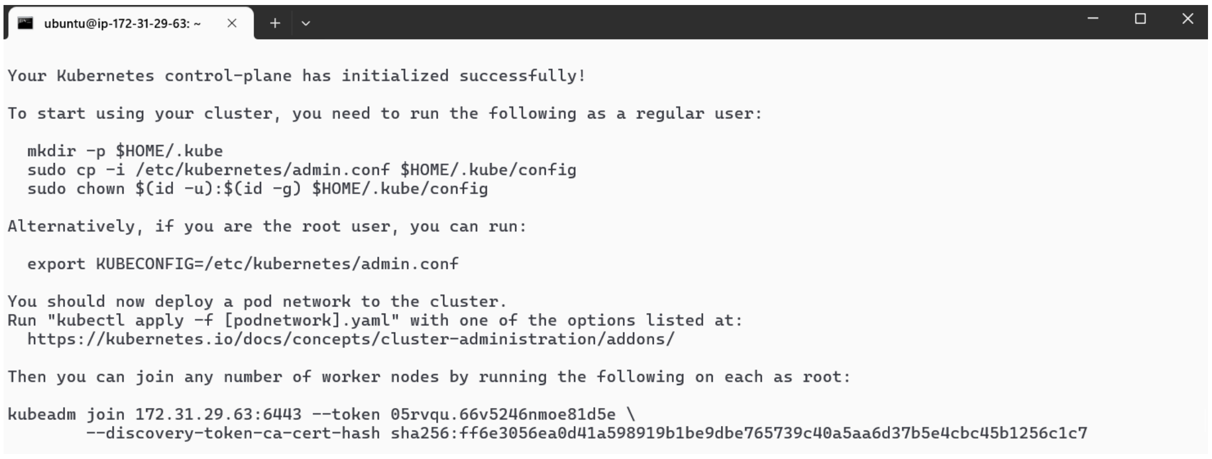
```
ubuntu@ip-172-31-29-63:~$ sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

Only on master node

➢ Initialize the Kubernetes Cluster on Master Node

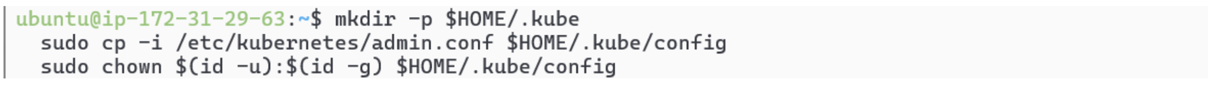


```
ubuntu@ip-172-31-29-63:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
I0915 17:58:19.113429    3356 version.go:256] remote version is much newer: v1.31.0; falling back to: stable-1.29
[init] Using Kubernetes version: v1.29.8
[preflight] Running pre-flight checks
[preflight] Pulling images 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 in beforehand using 'kubeadm config images pull'
W0915 17:58:28.696218    3356 checks.go:835] detected that the sandbox image "registry.k8s.io/pause:3.8" of the containe
r runtime is inconsistent with that used by kubeadm. It is recommended that using "registry.k8s.io/pause:3.9" as the CRI
 sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-29-63 kubernetes kubernetes.default kubernetes.default
.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.29.63]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-29-63 localhost] and IPs [172.31.29.63 127.0.0.1 ::1
]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-29-63 localhost] and IPs [172.31.29.63 127.0.0.1 ::1]
```

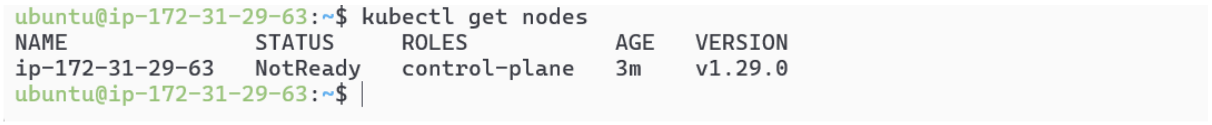


```
Your Kubernetes control-plane has initialized successfully!

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 172.31.29.63:6443 --token 05rvqu.66v5246nmoe81d5e \
        --discovery-token-ca-cert-hash sha256:ff6e3056ea0d41a598919b1be9dbe765739c40a5aa6d37b5e4cbc45b1256c1c7
```

➢ Set up kubectl on the master node

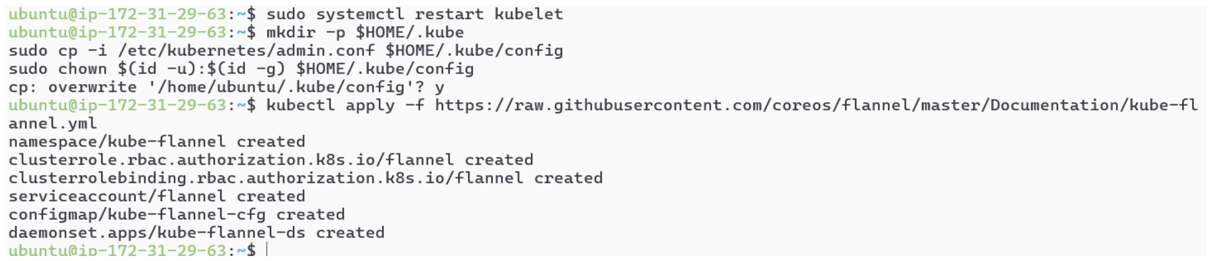


```
ubuntu@ip-172-31-29-63:~$ mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

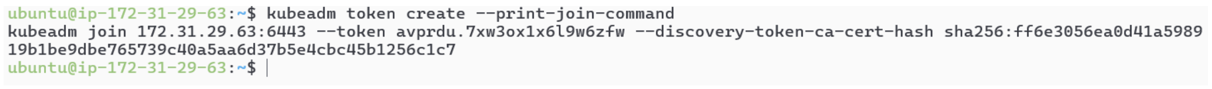


```
ubuntu@ip-172-31-29-63:~$ kubectl get nodes
NAME              STATUS     ROLES           AGE   VERSION
ip-172-31-29-63   NotReady   control-plane   3m    v1.29.0
ubuntu@ip-172-31-29-63:~$
```

➢ To enable communication between pods, install a pod network plugin like Flannel or Calico



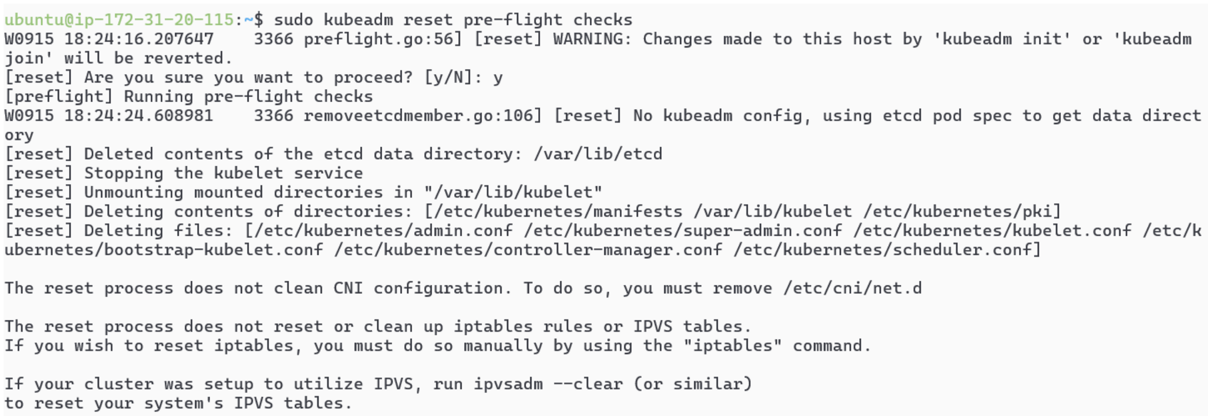
```
ubuntu@ip-172-31-29-63:~$ sudo systemctl restart kubelet
ubuntu@ip-172-31-29-63:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
cp: overwrite '/home/ubuntu/.kube/config'? y
ubuntu@ip-172-31-29-63:~$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-fl
annel.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
ubuntu@ip-172-31-29-63:~$
```



```
ubuntu@ip-172-31-29-63:~$ kubeadm token create --print-join-command
kubeadm join 172.31.29.63:6443 --token avprdu.7xw3ox1x6l9w6zfw --discovery-token-ca-cert-hash sha256:ff6e3056ea0d41a5989
19b1be9dbe765739c40a5aa6d37b5e4cbc45b1256c1c7
ubuntu@ip-172-31-29-63:~$
```

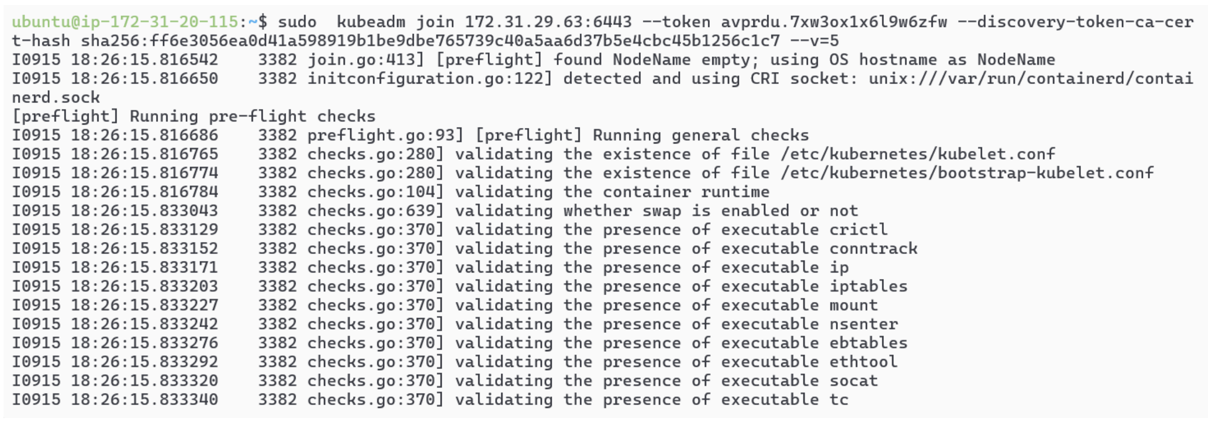
Only on Worker nodes

- Join Worker Nodes to the Cluster

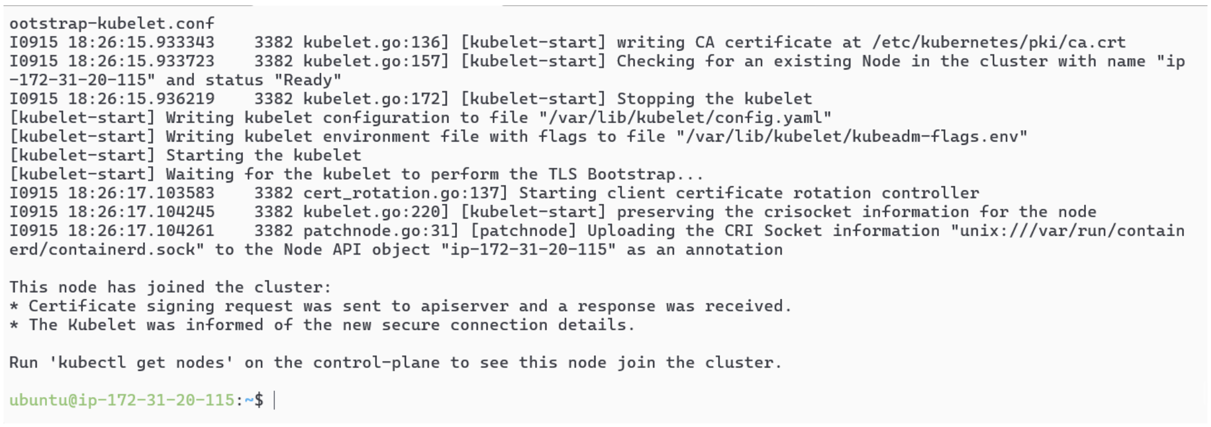


```
ubuntu@ip-172-31-20-115:~$ sudo kubeadm reset pre-flight checks
W0915 18:24:16.207647    3366 preflight.go:56] [reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm
join' will be reverted.
[reset] Are you sure you want to proceed? [y/N]: y
[preflight] Running pre-flight checks
W0915 18:24:24.608981    3366 removeetcdmember.go:106] [reset] No kubeadm config, using etcd pod spec to get data direct
ory
[reset] Deleted contents of the etcd data directory: /var/lib/etcd
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Deleting contents of directories: [/etc/kubernetes/manifests /var/lib/kubelet /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/super-admin.conf /etc/kubernetes/kubelet.conf /etc/k
ubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf /etc/kubernetes/scheduler.conf]

The reset process does not clean CNI configuration. To do so, you must remove /etc/cni/net.d

The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables" command.

If your cluster was setup to utilize IPVS, run ipvsadm --clear (or similar)
to reset your system's IPVS tables.
```

- On the worker nodes, run the command provided by the master node during initialization . It looks something like this: sudo kubeadm join :6443 --token --discovery-token-ca-cert-hash sha256:

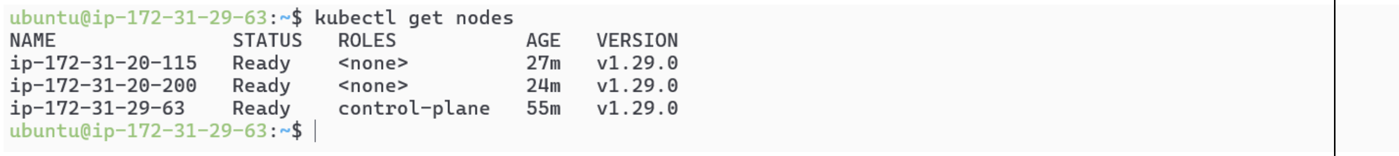


```
ubuntu@ip-172-31-20-115:~$ sudo  kubeadm join 172.31.29.63:6443 --token avprdu.7xw3ox1x6l9w6zfw --discovery-token-ca-cer
t-hash sha256:ff6e3056ea0d41a598919b1be9dbe765739c40a5aa6d37b5e4cbc45b1256c1c7 --v=5
I0915 18:26:15.816542    3382 join.go:413] [preflight] found NodeName empty; using OS hostname as NodeName
I0915 18:26:15.816650    3382 initconfiguration.go:122] detected and using CRI socket: unix:///var/run/containerd/contai
nerd.sock
[preflight] Running pre-flight checks
I0915 18:26:15.816686    3382 preflight.go:93] [preflight] Running general checks
I0915 18:26:15.816765    3382 checks.go:280] validating the existence of file /etc/kubernetes/kubelet.conf
I0915 18:26:15.816774    3382 checks.go:280] validating the existence of file /etc/kubernetes/bootstrap-kubelet.conf
I0915 18:26:15.816784    3382 checks.go:104] validating the container runtime
I0915 18:26:15.833043    3382 checks.go:639] validating whether swap is enabled or not
I0915 18:26:15.833129    3382 checks.go:370] validating the presence of executable crictl
I0915 18:26:15.833152    3382 checks.go:370] validating the presence of executable conntrack
I0915 18:26:15.833171    3382 checks.go:370] validating the presence of executable ip
I0915 18:26:15.833203    3382 checks.go:370] validating the presence of executable iptables
I0915 18:26:15.833227    3382 checks.go:370] validating the presence of executable mount
I0915 18:26:15.833242    3382 checks.go:370] validating the presence of executable nsenter
I0915 18:26:15.833276    3382 checks.go:370] validating the presence of executable ebtables
I0915 18:26:15.833292    3382 checks.go:370] validating the presence of executable ethtool
I0915 18:26:15.833320    3382 checks.go:370] validating the presence of executable socat
I0915 18:26:15.833340    3382 checks.go:370] validating the presence of executable tc
```



```
ootstrap-kubelet.conf
I0915 18:26:15.933343    3382 kubelet.go:136] [kubelet-start] writing CA certificate at /etc/kubernetes/pki/ca.crt
I0915 18:26:15.933723    3382 kubelet.go:157] [kubelet-start] Checking for an existing Node in the cluster with name "ip
-172-31-20-115" and status "Ready"
I0915 18:26:15.936219    3382 kubelet.go:172] [kubelet-start] Stopping the kubelet
[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...
I0915 18:26:17.103583    3382 cert_rotation.go:137] Starting client certificate rotation controller
I0915 18:26:17.104245    3382 kubelet.go:220] [kubelet-start] preserving the crisocket information for the node
I0915 18:26:17.104261    3382 patchnode.go:31] [patchnode] Uploading the CRI Socket information "unix:///var/run/contain
erd/containerd.sock" to the Node API object "ip-172-31-20-115" as an annotation

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.

ubuntu@ip-172-31-20-115:~$ |
```

- Verify the Cluster
  Once the worker node joins, check the status on the master node



```
ubuntu@ip-172-31-29-63:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE   VERSION
ip-172-31-20-115   Ready    <none>          27m   v1.29.0
ip-172-31-20-200   Ready    <none>          24m   v1.29.0
ip-172-31-29-63    Ready    control-plane   55m   v1.29.0
ubuntu@ip-172-31-29-63:~$
```