

# Experiment no. 3

**Aim :** To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

# Theory :

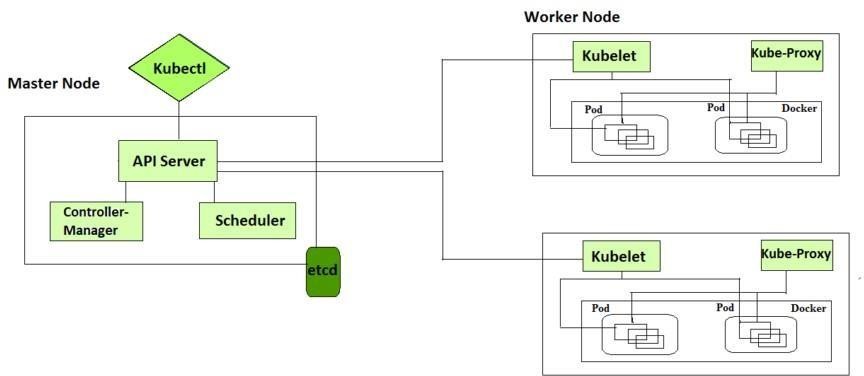
* **Kubernetes – Cluster Architecture**

[Kubernetes](https://www.geeksforgeeks.org/introduction-to-kubernetes-k8s/) comes with a client-server architecture. It consists of master and worker nodes, with the master being installed on a single Linux system and the nodes on many Linux workstations. The master node, contains the components such as [API](https://www.geeksforgeeks.org/what-is-an-api/) Server, controller manager, scheduler, and etcd [database](https://www.geeksforgeeks.org/what-is-database/) for stage storage. kubelet to communicate with the master, the kube-proxy for networking, and a container runtime such as [Docker](https://www.geeksforgeeks.org/docker-tutorial/) to manage containers.

# Kubernetes Components

Kubernetes is composed of a number of components, each of which plays a specific role in the overall system. These components can be divided into two categories:

* 1. nodes: Each [Kubernetes cluster](https://www.geeksforgeeks.org/kubernetes-cluster/) [r](https://www.geeksforgeeks.org/kubernetes-cluster/)equires at least one worker node, which is a collection of worker machines that make up the nodes where our container will be deployed.
  2. Control plane: The worker nodes and any pods contained within them will be under the control plane.



# Control Plane Components

It is basically a collection of various components that help us in managing the overall health of a cluster. For example, if you want to set up new pods, destroy pods, scale pods, etc. Basically, 4 services run on Control Plane:

1. Kube-API server

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. It is like an initial gateway to the cluster that listens to updates or queries via CLI like Kubectl. Kubectl communicates with API Server to inform what needs to be done like creating pods or deleting pods etc. It also works as a gatekeeper. It generally validates requests received and then forwards them to other processes. No request can be directly passed to the cluster, it has to be passed through the API Server.

1. Kube-Scheduler

When API Server receives a request for Scheduling Pods then the request is passed on to the Scheduler. It intelligently decides on which node to schedule the pod for better efficiency of the cluster.

1. Kube-Controller-Manager

The kube-controller-manager is responsible for running the controllers that handle the various aspects of the cluster’s control loop. These controllers include the replication controller, which ensures that the desired number of replicas of a given application is running, and the node controller, which ensures that nodes are correctly marked as “ready” or “not ready” based on their current state.

1. etcd

It is a key-value store of a Cluster. The Cluster State Changes get stored in the etcd. It acts as the Cluster brain because it tells the Scheduler and other processes about which resources are available and about cluster state changes.

# — Node Components

These are the nodes where the actual work happens. Each Node can have multiple pods and pods have containers running inside them. There are 3 processes in every Node that are used to Schedule and manage those pods.

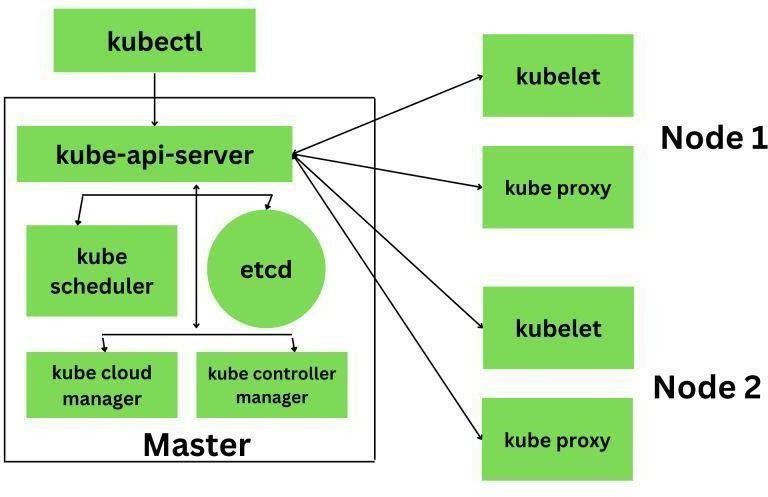
The following are the some of the components related to Node:

1. Container runtime

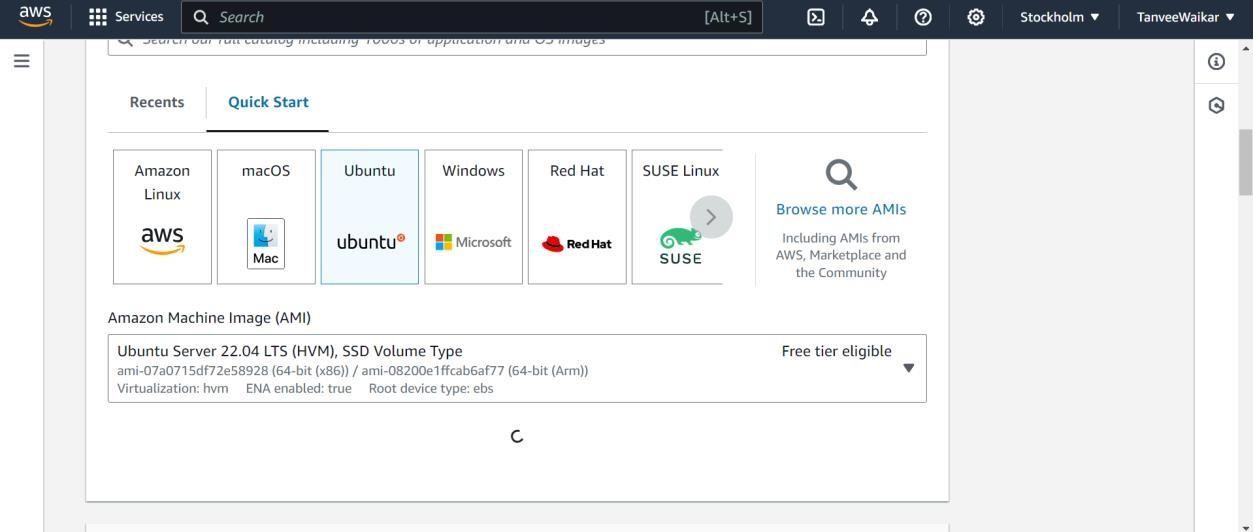
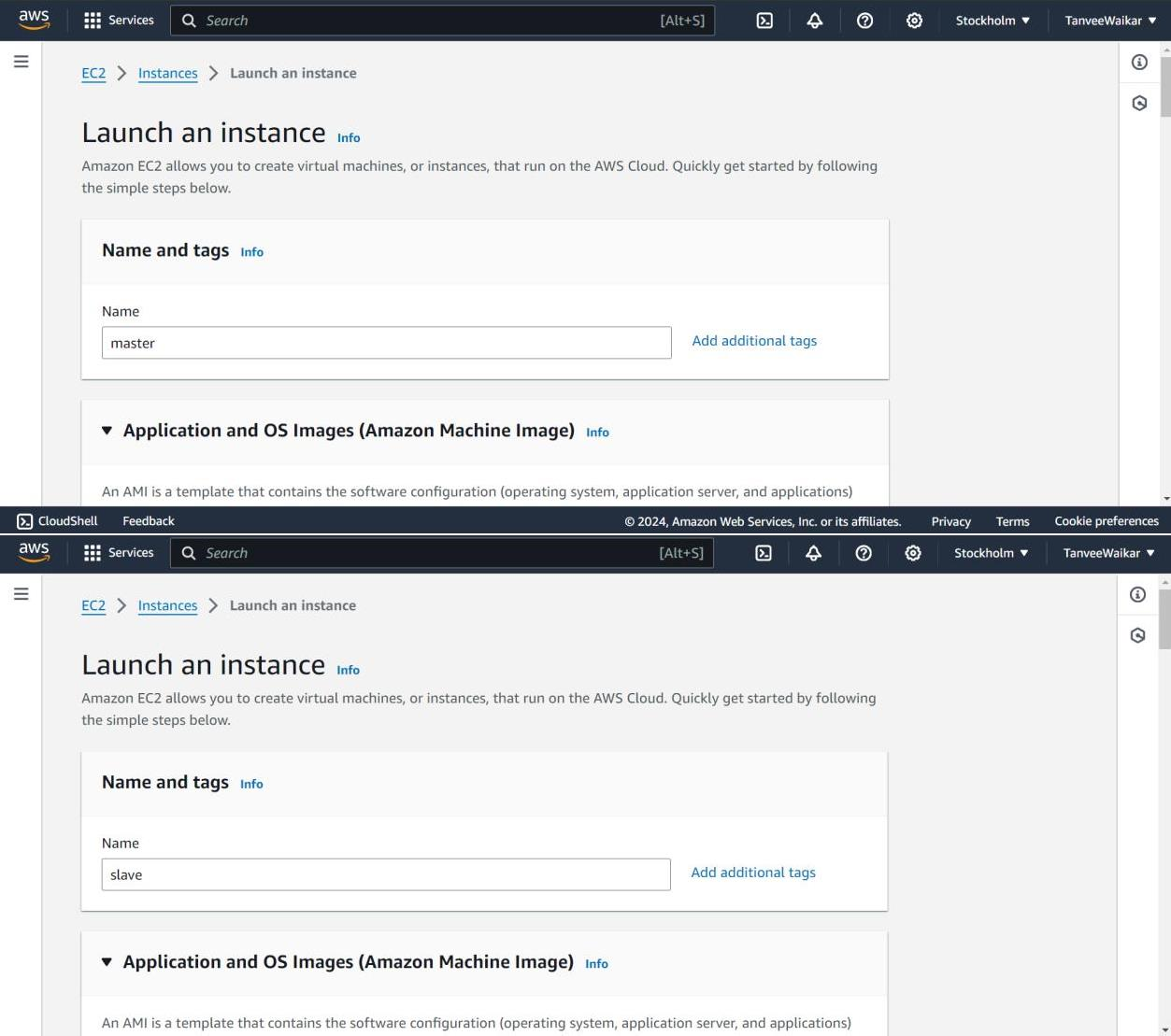
A container runtime is needed to run the application containers running on pods inside a pod. Example-> Docker

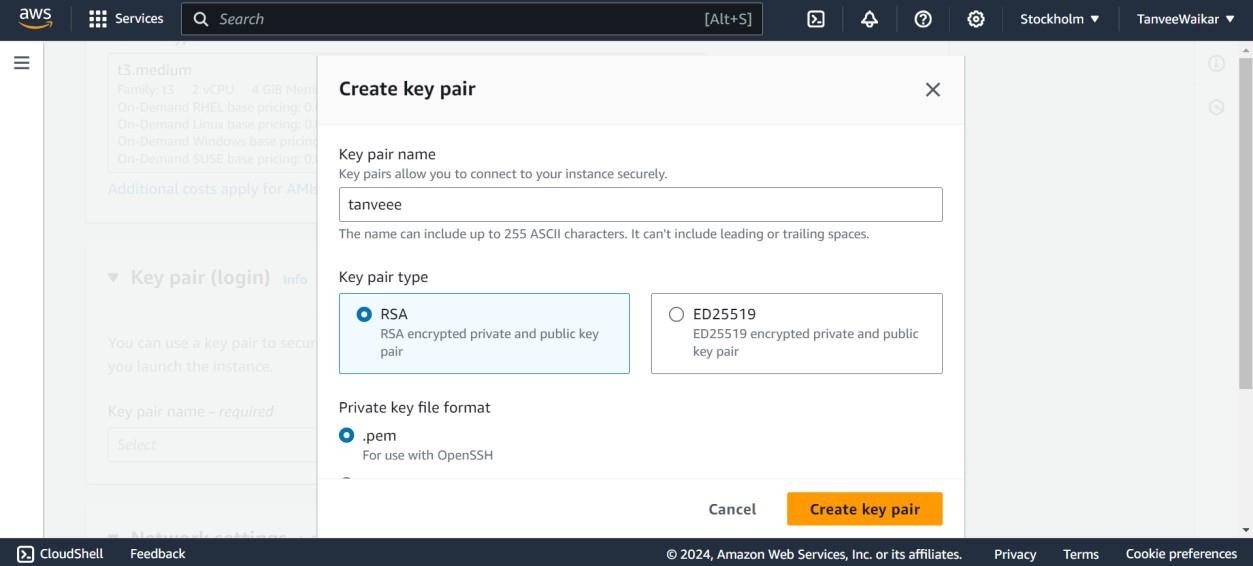
1. kubelet kubelet interacts with both the container runtime as well as the Node. It is the process responsible for starting a pod with a container inside.
2. kube-proxy

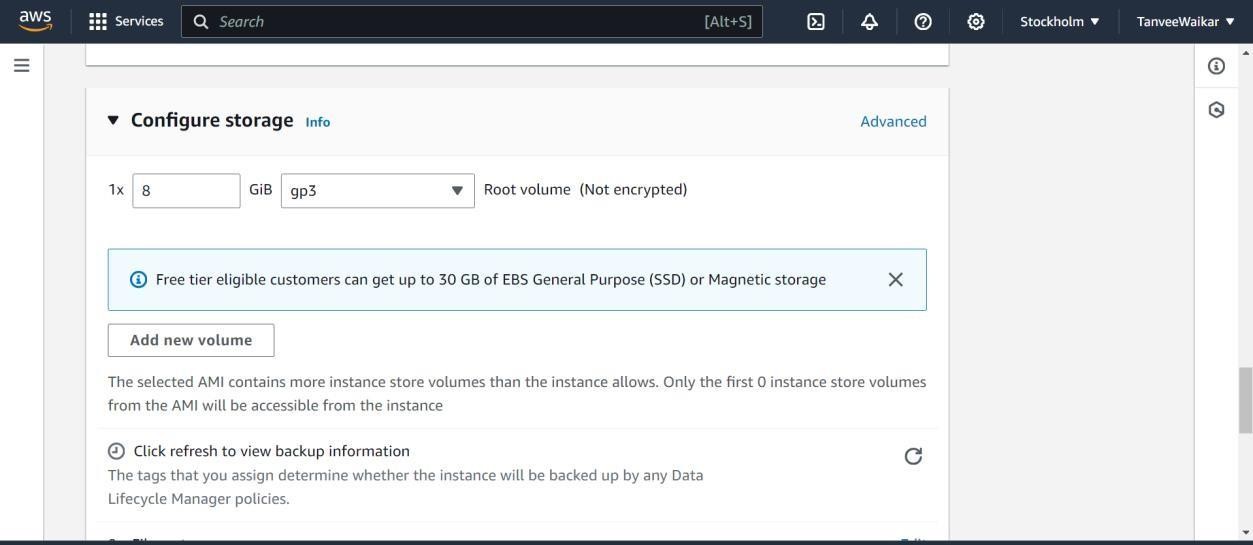
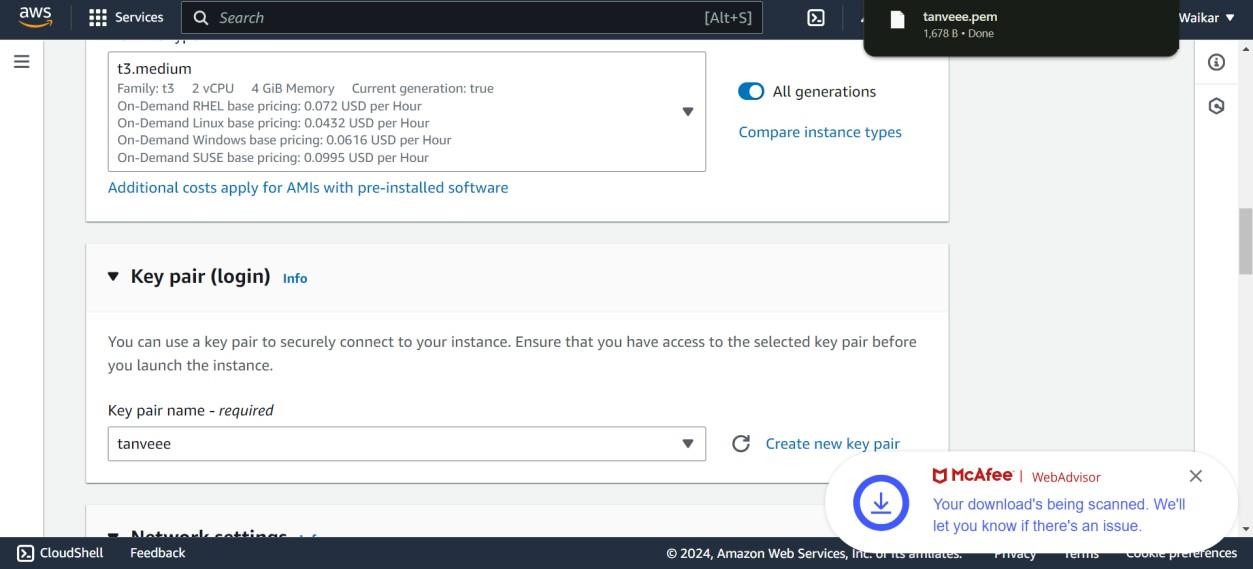
It is the process responsible for forwarding the request from Services to the pods. It has intelligent logic to forward the request to the right pod in the worker node.

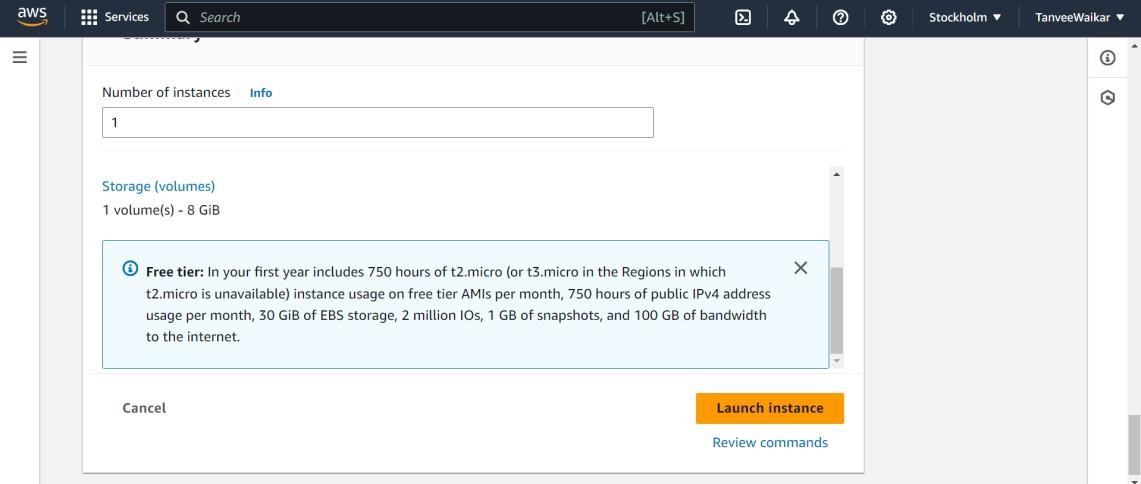


#Create master and slave

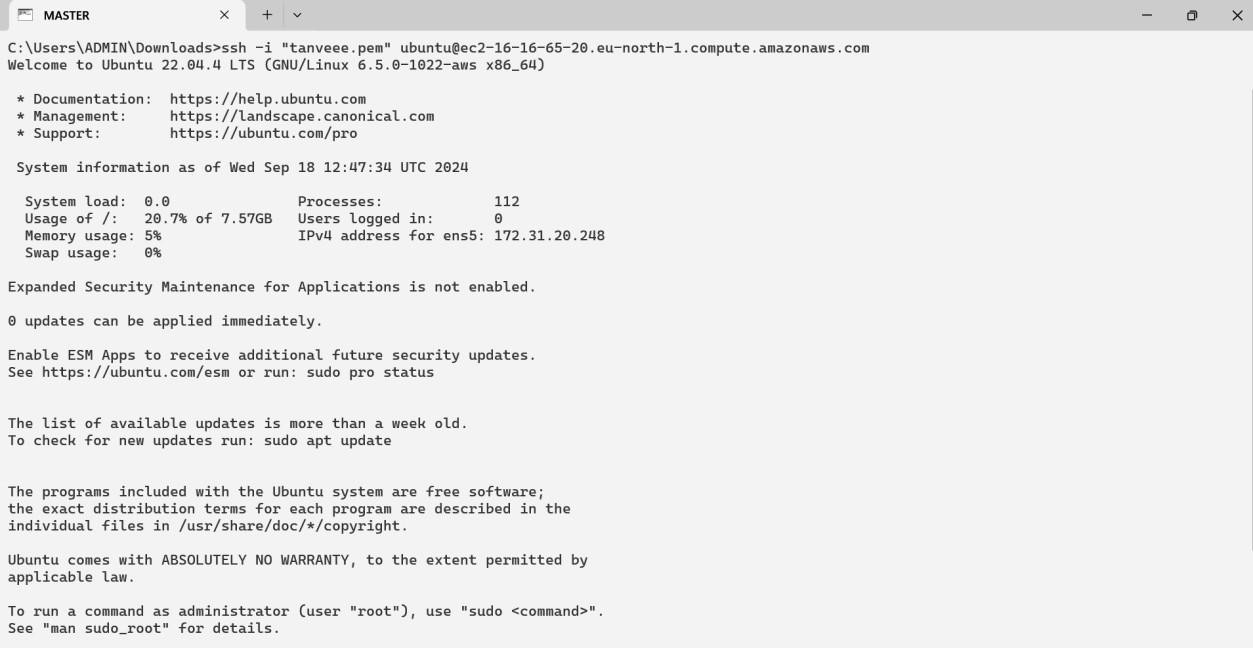




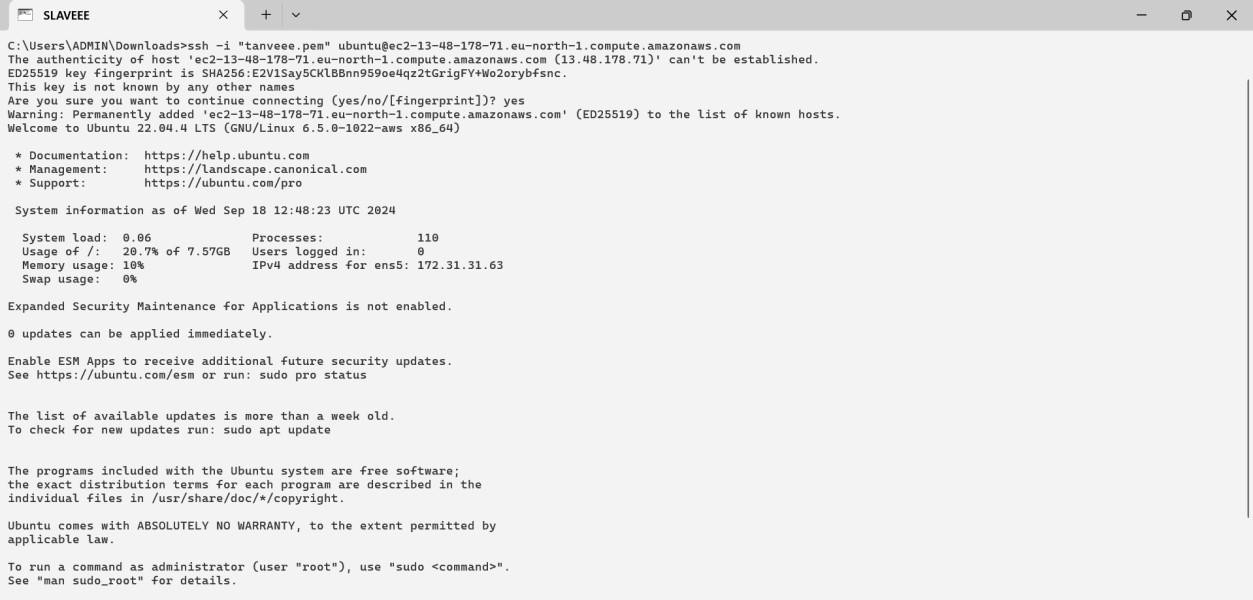




#MASTER CONNECT



#SLAVE CONNECT



**#Execute on Both "Master" & "Worker Node"# disable swap**

**sudo swapoff -a**

**# Create the .conf file to load the modules at bootup**

**cat** <<EOF | sudo tee /etc/modules-load.d/k8s.confoverlay br\_netfilter

EOF

sudo modprobe overlay sudo modprobe br\_netfilter

# sysctl params required by setup, params persist across rebootscat

<<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 =

1EOF

# Apply sysctl params without reboot sudo sysctl --system

## Install CRIO Runtimesudo apt-get update -y

sudo apt-get install -y software-properties-common curl apt-transport-https ca-certificates gpg sudo curl -fsSL https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/Release.key | sudo gpg --dearmor - o /etc/apt/keyrings/cri-o-apt-keyring.gpg

echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg] https://pkgs.k8s.io/addons:/cri- o:/prerelease:/main/deb/ /" | sudo tee /etc/apt/sources.list.d/cri-o.list

sudo apt-get update -y sudo apt-get install -y cri-o

sudo systemctl daemon-reload sudo systemctl enable crio --now sudo systemctl start crio.service

echo "CRI runtime installed successfully"

# Add Kubernetes APT repository and install required packages

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/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:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.listsudo apt- get update -y

sudo apt-get install -y kubelet="1.29.0-\*" kubectl="1.29.0-\*" kubeadm="1.29.0-\*"sudo apt-get update -y

sudo apt-get install -y jq

sudo systemctl enable --now kubelet sudo systemctl start kubelet



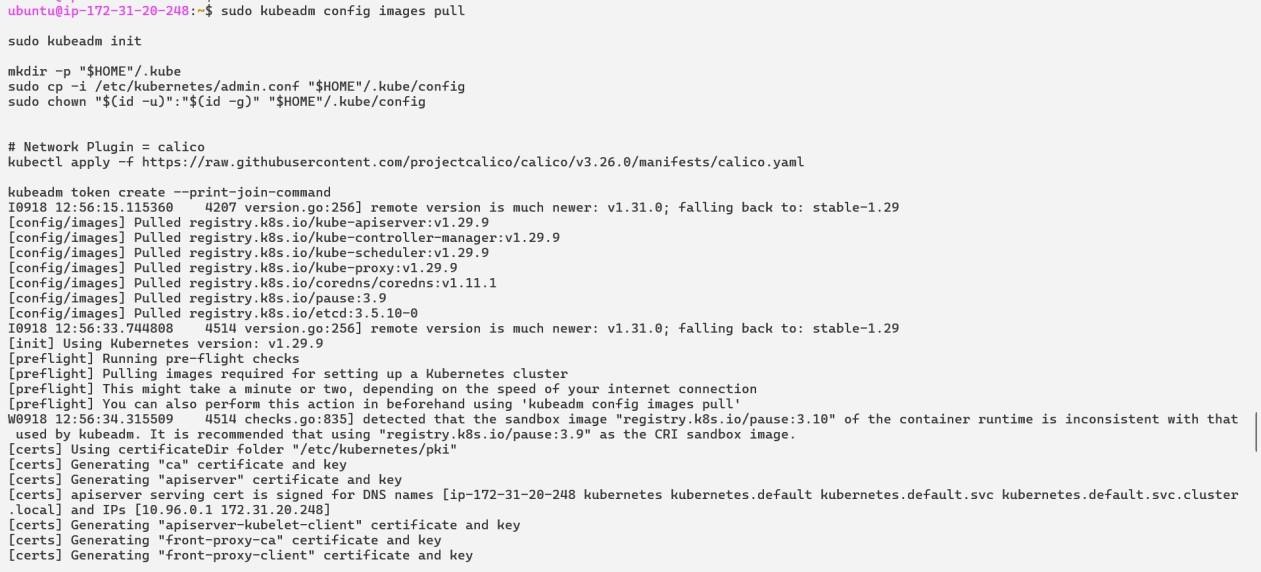


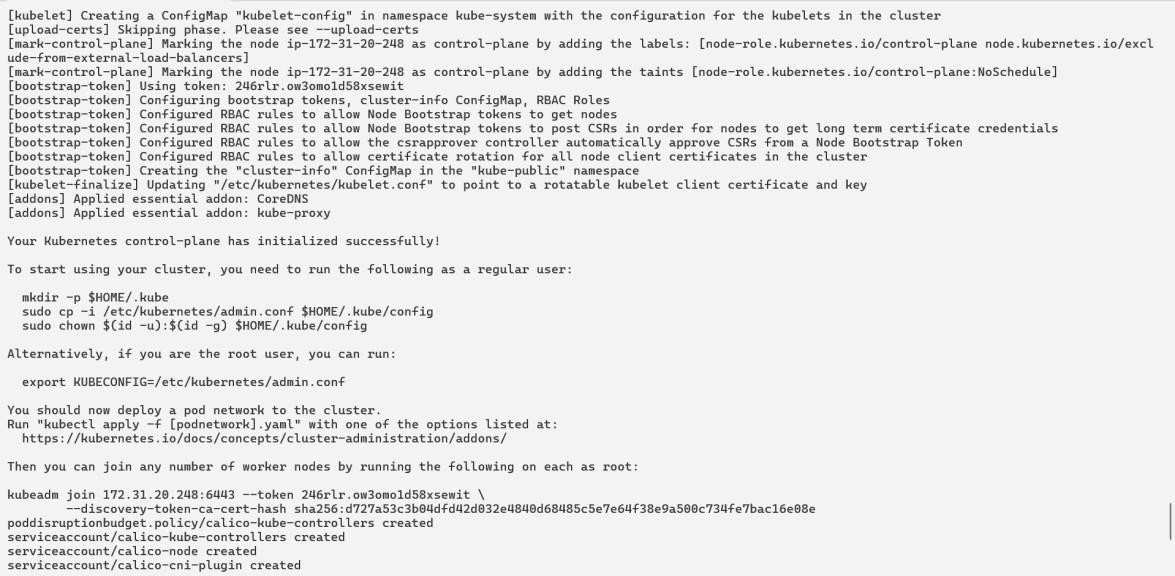
**Execute ONLY on "Master Node"** sudo kubeadm config images pullsudo kubeadm init mkdir -p "$HOME"/.kube

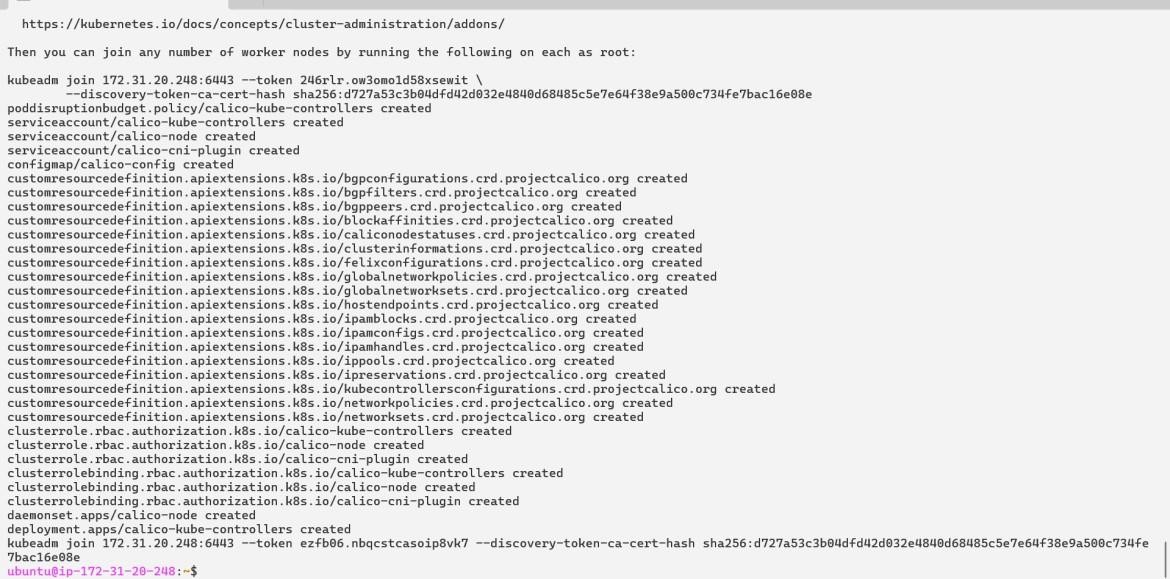
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/configsudo chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config

# Network Plugin = calico kubectl apply -f

https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml kubeadm token create --print-join-command



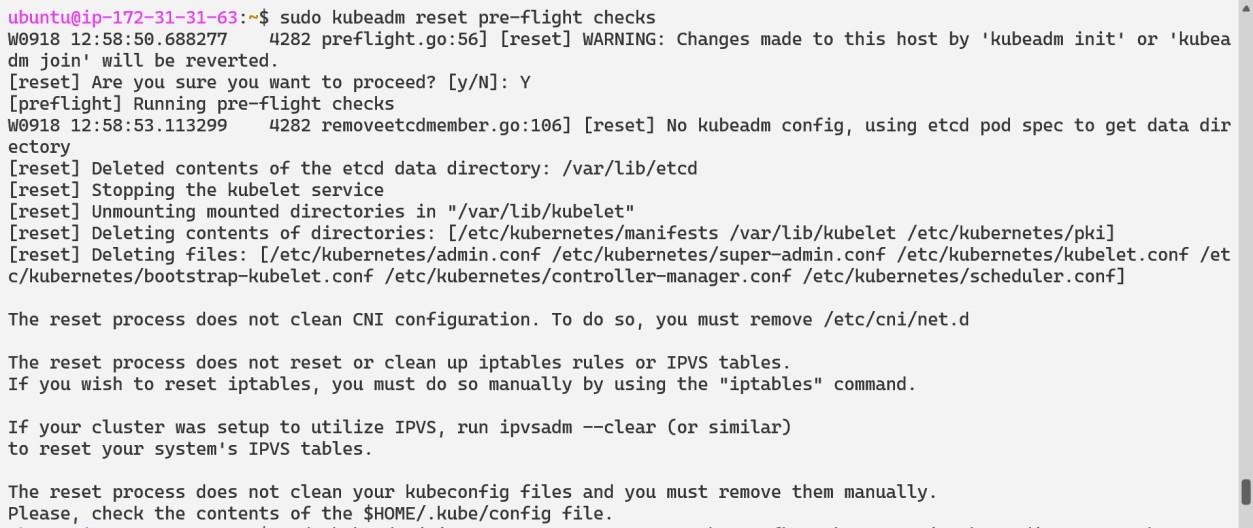




**Execute on ALL of your Worker Node's**

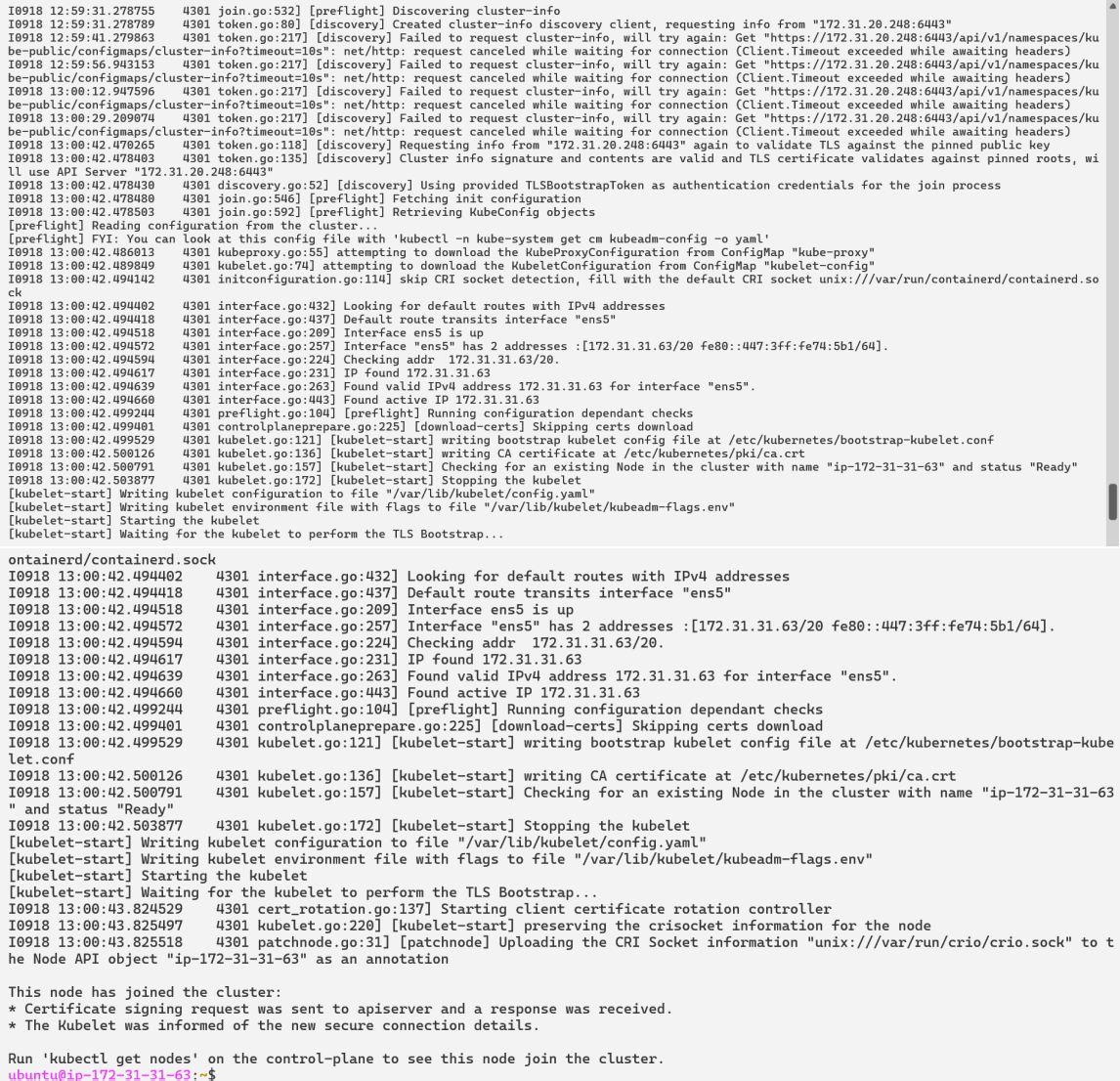
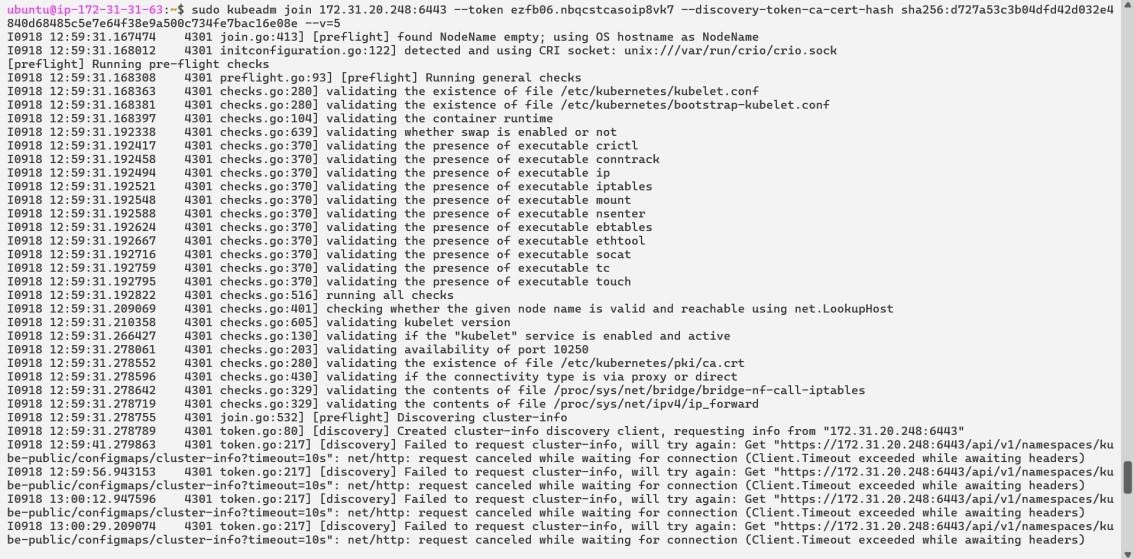
1. **Perform pre-flight checks**

sudo kubeadm reset pre-flight checks



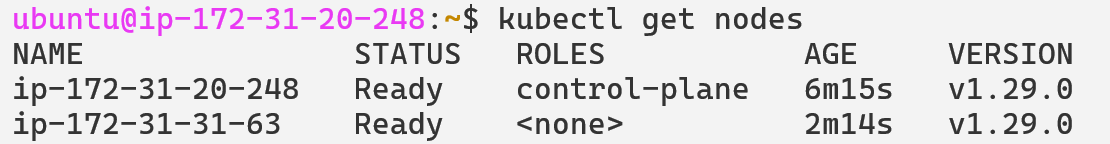
1. **Paste the join command you got from the master node and append --v=5 at the end.sudo your-token --v=5**

**Use sudo before the token.**



**Verify Cluster ConnectionOn Master Node:**

kubectl get nodes



* + **Conclusion :** Hence we have studied the Kubernetes Cluster Architecture, and successfully install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms