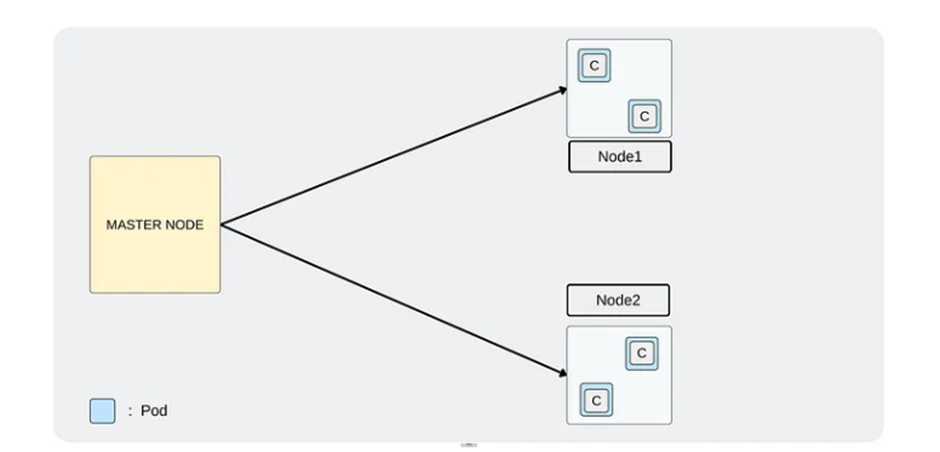
**Setting up a Kubernetes Cluster(Master+Worker Node) using kubeadm on AWS EC2**

**Objectives**  
By the end of this tutorial, you will:

Create three EC2 instances with Ubuntu 22.04 and the necessary security group settings.  
Configure the instances to prepare for Kubernetes installation.  
Install Docker and Kubernetes components on all nodes.  
Initialize the Kubernetes cluster on the master node.  
Join the worker nodes in the cluster.  
Deploy an Nginx application on the cluster for validation.



**Step 1:**

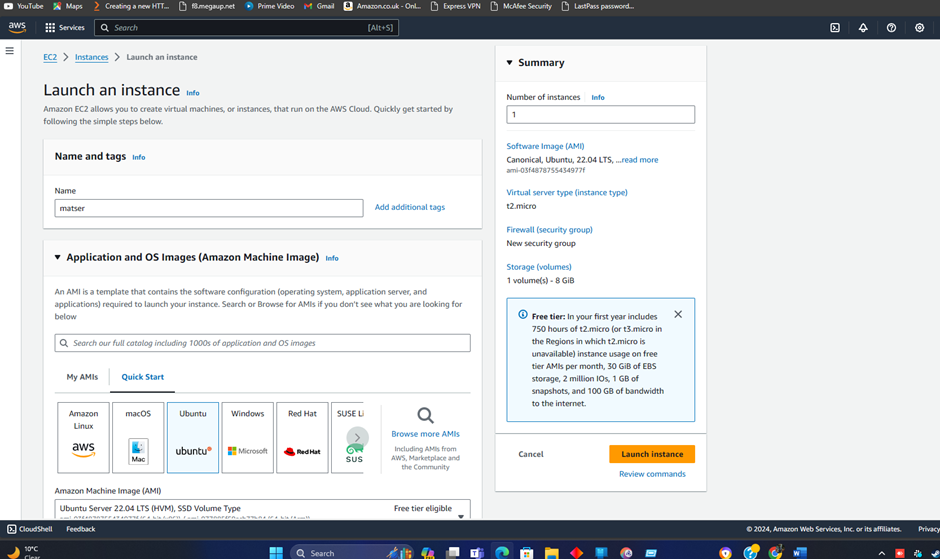
Create two EC2 Instances with the given configuration

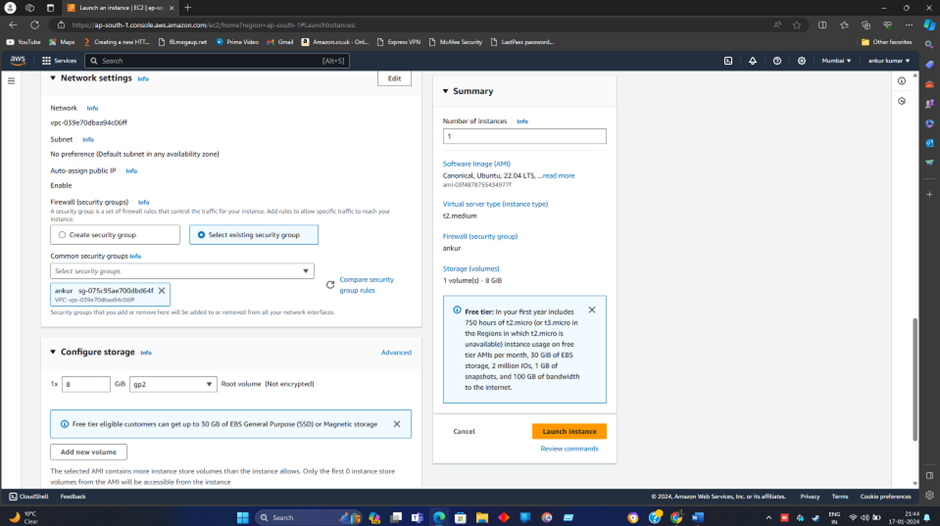
Instance type- t2. medium

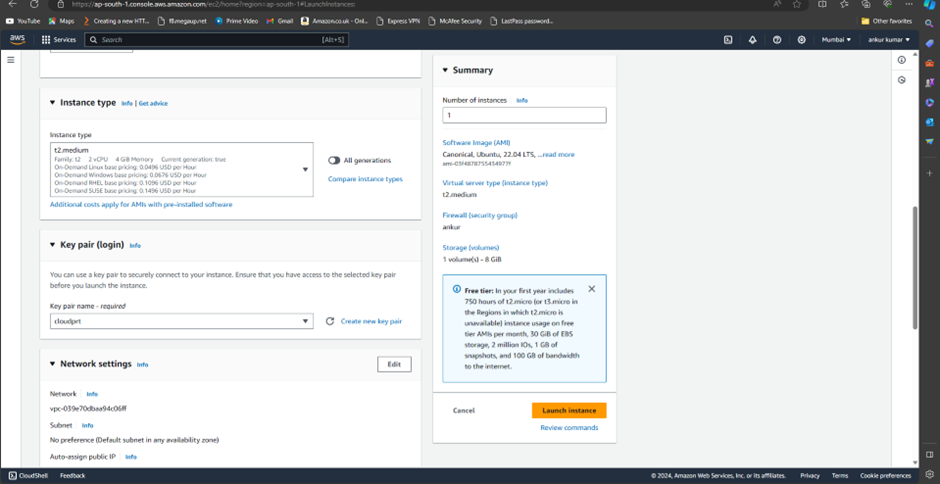
Ubuntu Version- 22.04

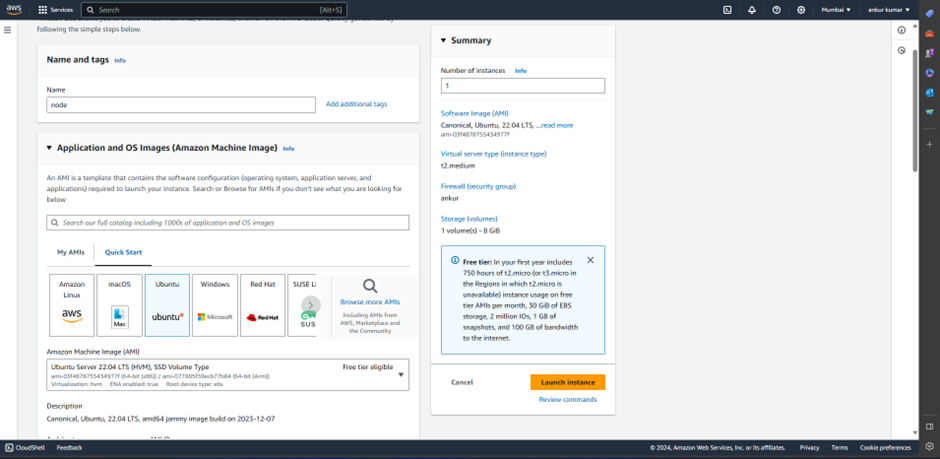
Create the keypair so you can connect to the instance using SSH.

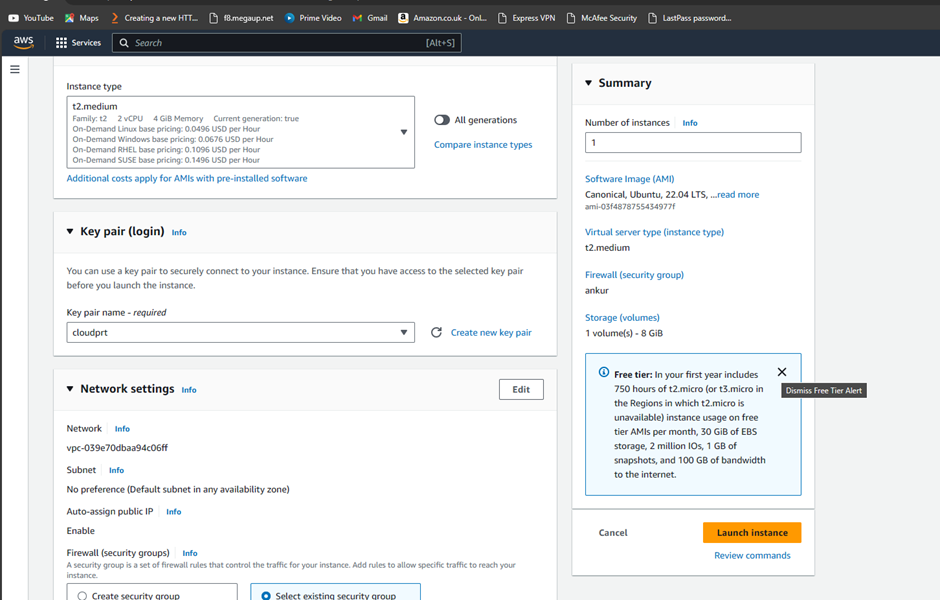
Create the new Security group and once the instances are initialized/created then, make sure to add the Allow All traffic in inbound rule in the attached security group.











After creating the instances, we have to configure the all instances. Let’s do that and follow the steps carefully.

Commands need to run on all Nodes(Master and Worker)

Once we log in the all three instances, run the following command.

Step 2:

sudo su

swapoff -a; sed -i ‘/swap/d’ /etc/fstab

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

sudo modprobe overlay

sudo modprobe br\_netfilter

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

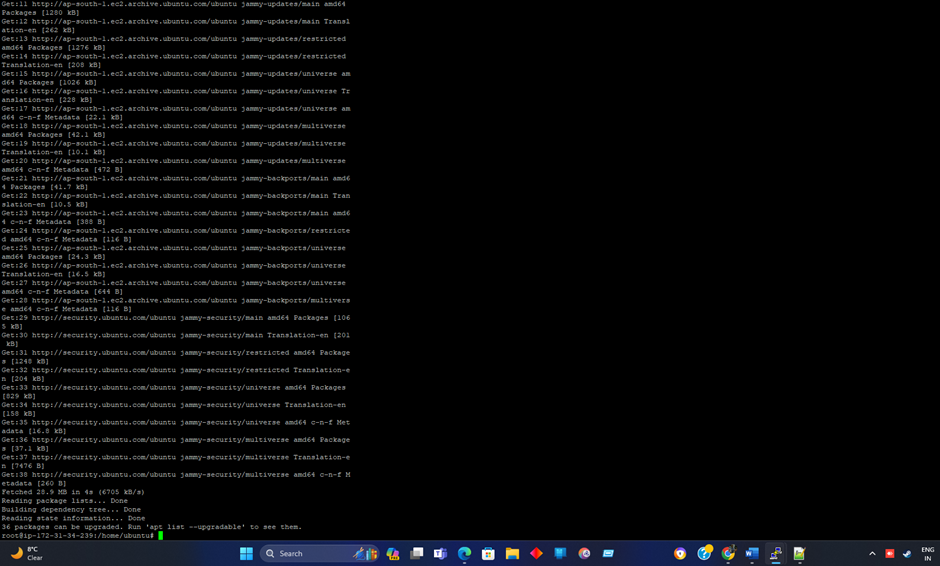
net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip\_forward

EOF

**Step3**

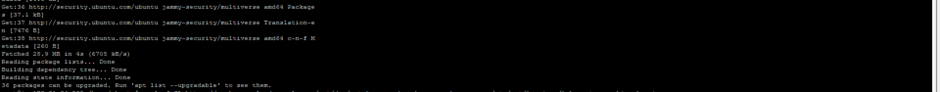
apt update



**Step4**

Fetch the public key from Google to validate the Kubernetes packages once it will be installed.

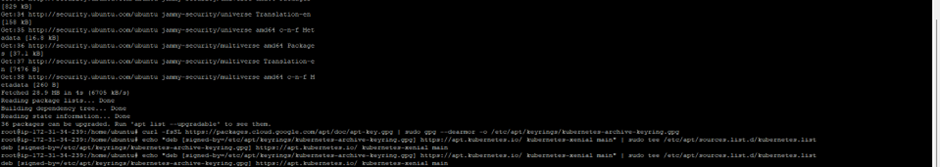
curl -fsSL <https://packages.cloud.google.com/apt/doc/apt-key.gpg> | sudo gpg — dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg



**Step5**

Add the Kubernetes package in the sources.list.d directory

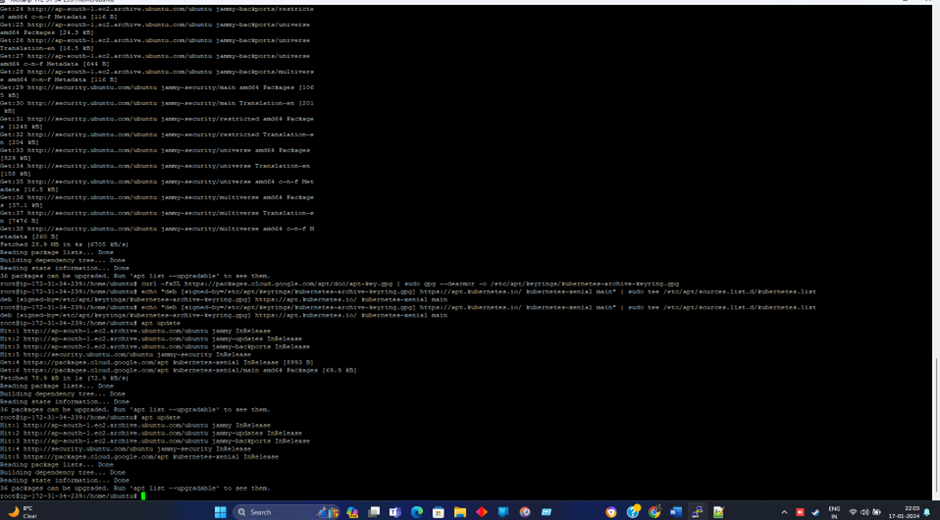
echo “deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] <https://apt.kubernetes.io/> kubernetes-xenial main” | sudo tee /etc/apt/sources.list.d/kubernetes.list



**Step6**

Update the packages as we have added some keys and packages.

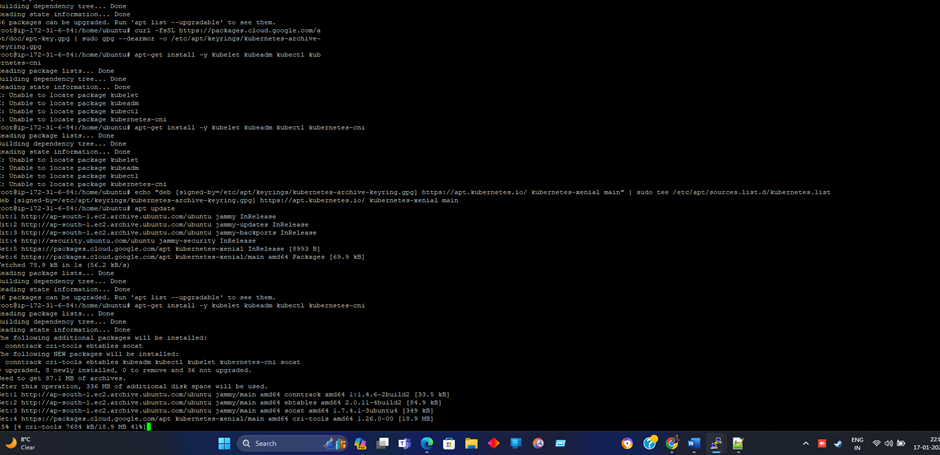
apt update



**Step7**

Install kubelet, kubeadm, kubectl and kubernets-cni

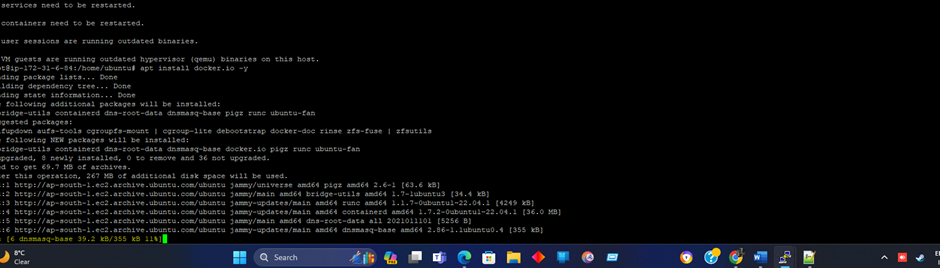
apt-get install -y kubelet kubeadm kubectl kubernetes-cni



**Step8**

This is one of the important dependencies to setting up the Master and Worker nodes. Installing docker.

apt install docker.io -y



**Step9**

Configuring containerd to ensure compatibility with Kubernetes

sudo mkdir /etc/containerd

sudo sh -c “containerd config default > /etc/containerd/config.toml”

sudo sed -i ‘s/ SystemdCgroup = false/ SystemdCgroup = true/’ /etc/containerd/config.toml



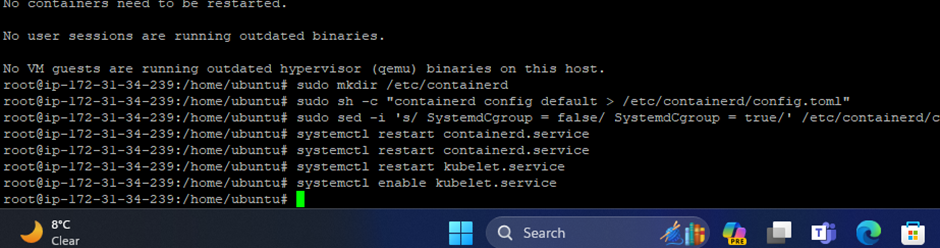
Step 10

Restart containerd, kubelet, and enable kubelet so when we reboot our machine the nodes will restart it as well and connect properly.

systemctl restart containerd.service

systemctl restart kubelet.service

systemctl enable kubelet.service



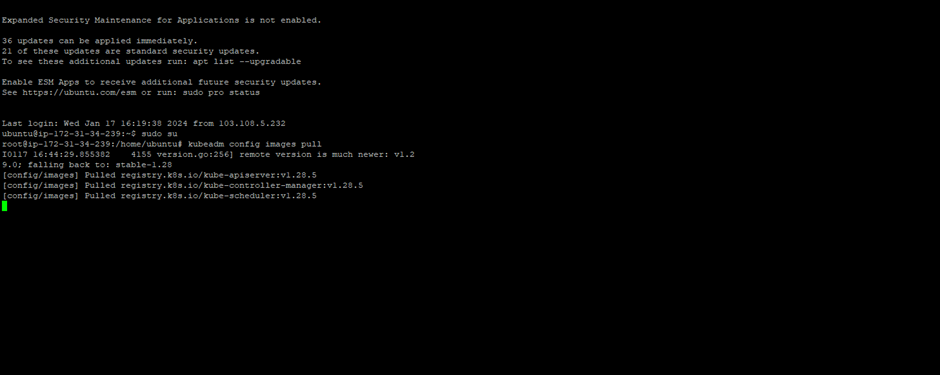
Now, we have completed the installation of the things that are needed on both nodes (Master and Worker). But in the next steps, we have to configure things only on the Master Node.

**Only on the Master Node**

**Step 11**

Initialize the Kubernetes cluster and it will pull some images such as kube-apiserver, kube-controller, and many other important components.

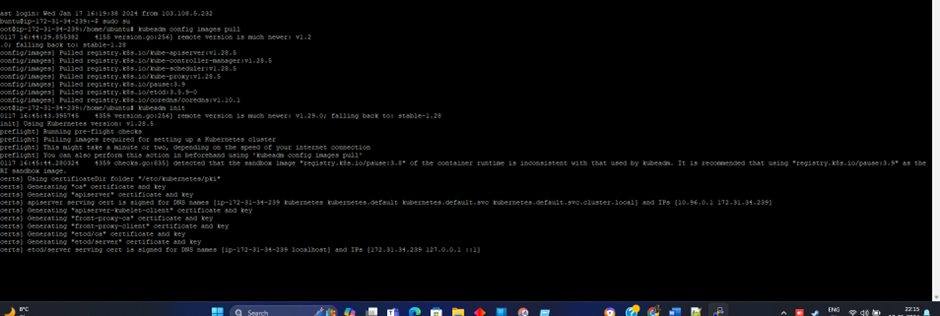
kubeadm config images pull



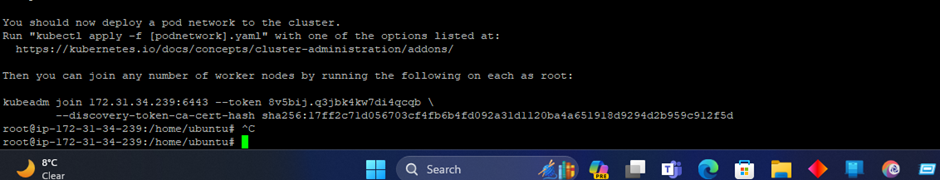
**Step 12**

Now, initialize the Kubernetes cluster which will give you the token or command to connect with this Master node from the Worker node. At the end of this command, you will get some commands that need to run and at the bottom, you will get the **kubeadm join**command that will be run from the **Worker Node**to connect with the Master Node. I have highlighted the commands in the second next snipped. Please keep the **kubeadm join**command somewhere in the notepad.

kubeadm init



Keep the **kubeadm join**command in your notepad or somewhere for later.



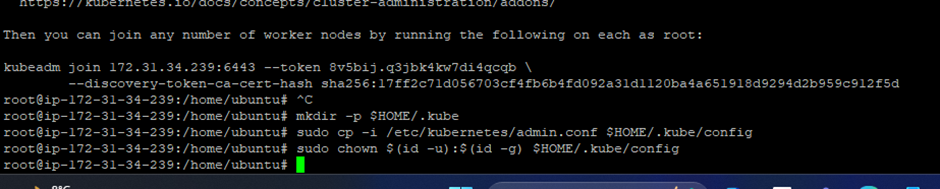
**Step 13**

As you have to manage the cluster that’s why you need to create a .kube file copy it to the given directory and change the ownership of the file.

mkdir -p $HOME/.kube

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

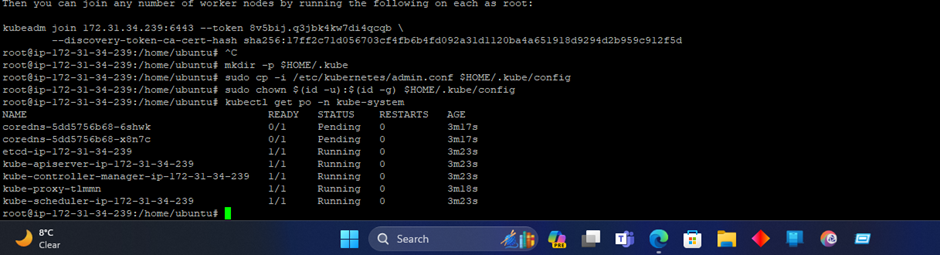
sudo chown $(id -u):$(id -g) $HOME/.kube/config



**Step 14**

Verify the kubeconfig by kube-system which will list all the pods. If you observe, there are starting twopods that are not ready status because the network plugin is not installed.

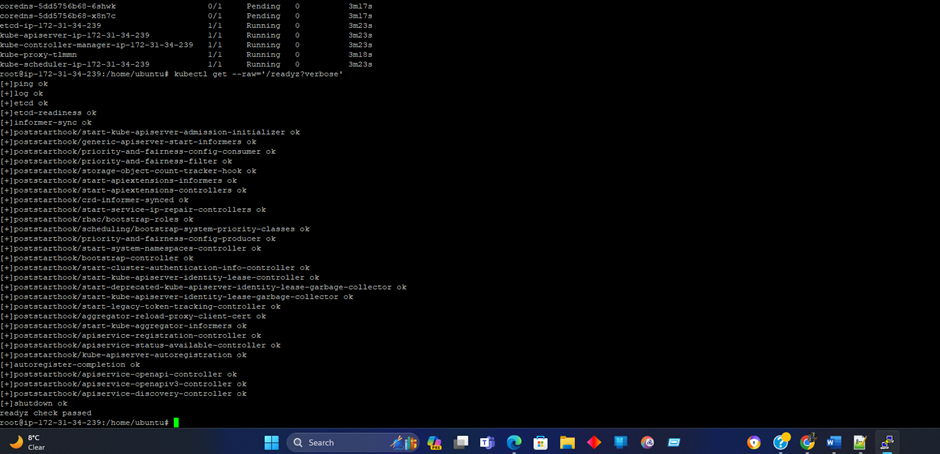
kubectl get po -n kube-system



**Step 15**

Verify all the cluster component health statuses

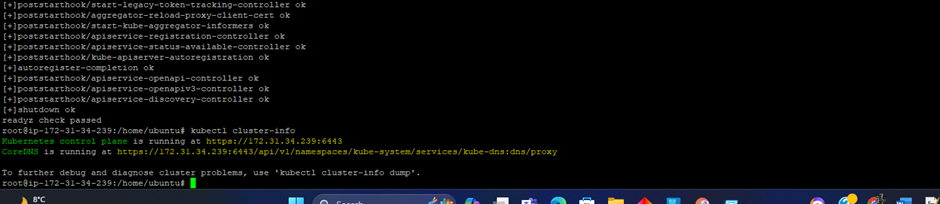
kubectl get — raw=’/readyz?verbose’



**Step 16**

Check the cluster-info

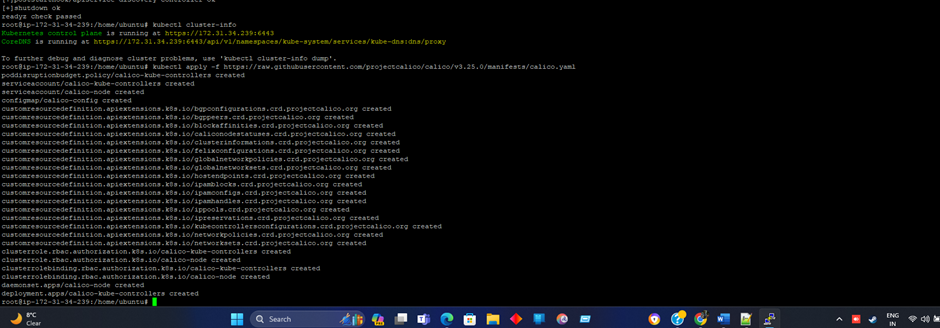
kubectl cluster-info



**Step 17**

To install the network plugin on the Master node

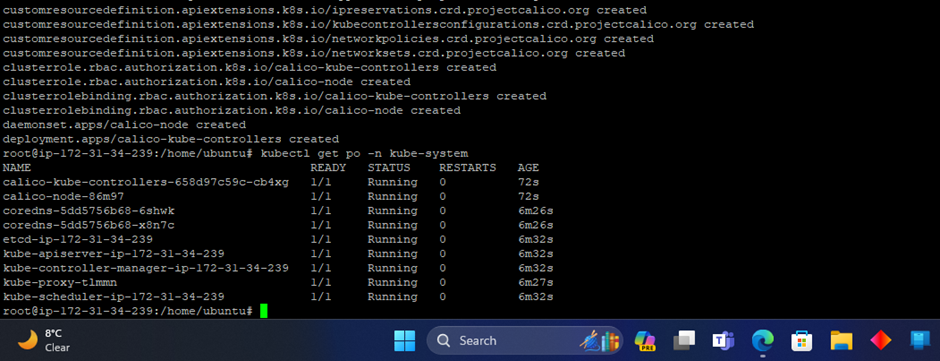
kubectl apply -f <https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml>



**Step 18**

now, If you run the below command, you will observe the two remaining pods are in ready status which means we are ready to bootstrap by our Workers Node or connect to the Master node through the Worker Node.

kubectl get po -n kube-system

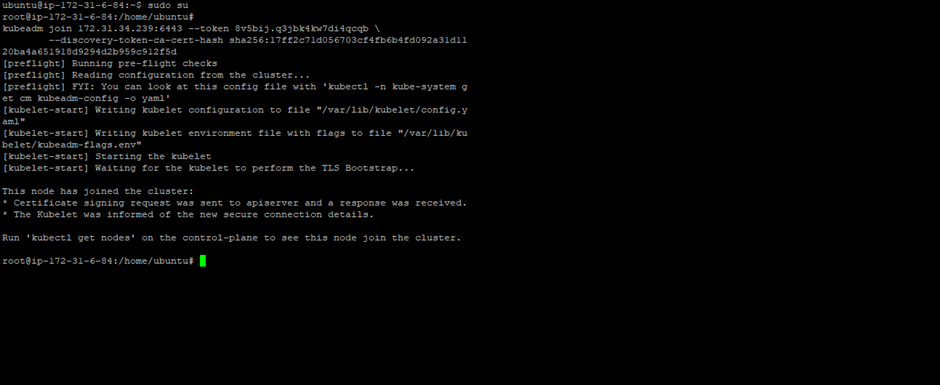


**Step 19**

Now Join your worker node in our cluster using this command

kubeadm join 172.31.34.239:6443 — token 8v5bij.q3jbk4kw7di4qcqb \

— discovery-token-ca-cert-hash sha256:17ff2c71d056703cf4fb6b4fd092a31d1120ba4a651918d9294d2b959c912f5d

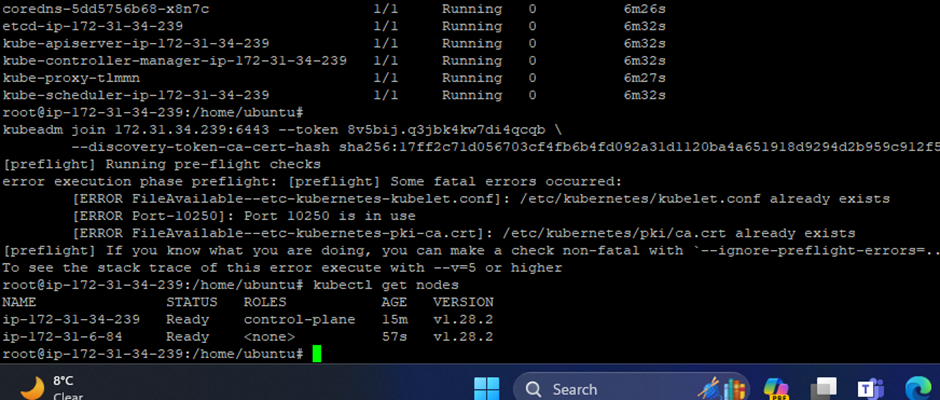


**Step 20**

**Now, from here all the commands will be run on Master Node only.**

If you run the below command you will see that the Worker Nodes are present with their respective Private IPs and it is in Ready. status

kubectl get nodes



apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2

template:

metadata:

labels:

app: nginx

spec:

containers:

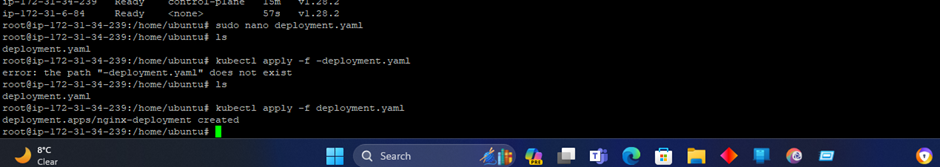
- name: nginx

image: nginx:latest

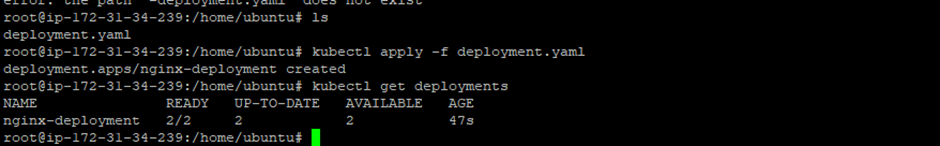
ports:

- containerPort: 80

kubectl apply -f -deployment.yaml



kubectl get deployments



**Step 22**

To expose your deployment on NodePort 32000 which means you can access your nginx application on port 32000 through your browser easily.

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

type: NodePort

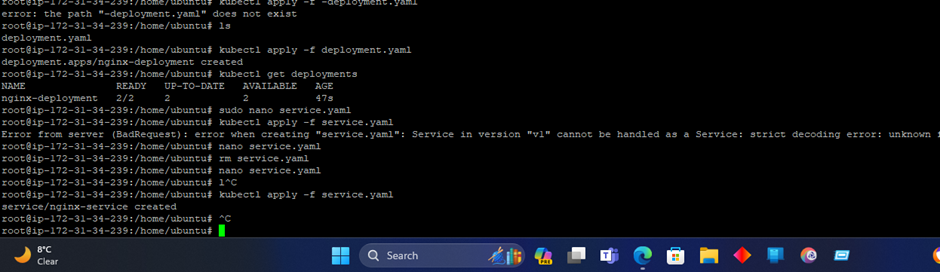
ports:

- port: 80

targetPort: 80

nodePort: 32000

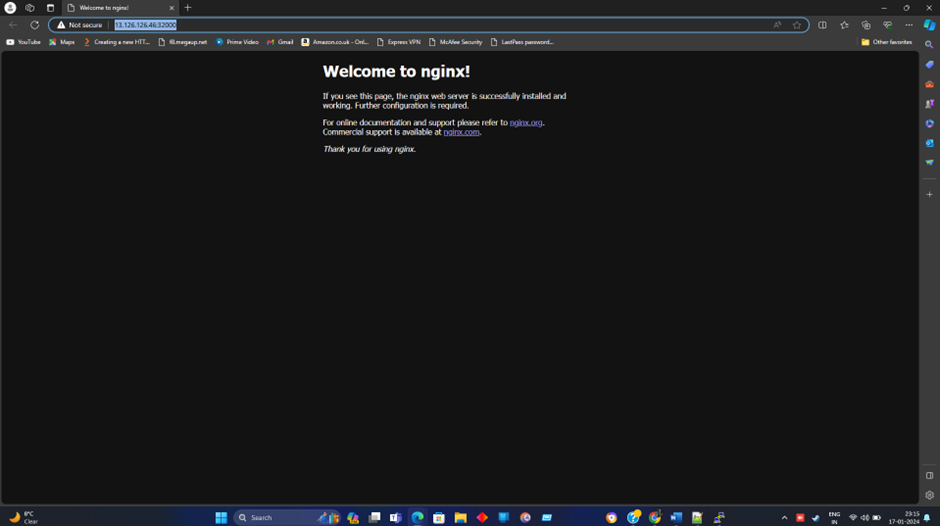
kubectl apply -f service.yaml



Now, check the pods by the below command and you can see that your pods are in running status.

You can validate your deployment by copying the Public IP of your WorkerNode

<http://13.126.126.46:32000/>



**Conclusion:**

Today, you’ve successfully set up a Kubernetes cluster with a master and two worker nodes. This foundational step is crucial for your Kubernetes journey. You’ve paved the way for deploying and managing containerized applications effectively. Keep exploring, and learning.