Advanced DevOps Lab Experiment 3

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

**Reference:** <https://www.youtube.com/watch?v=Cz7hSJNq2GU>

# 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.

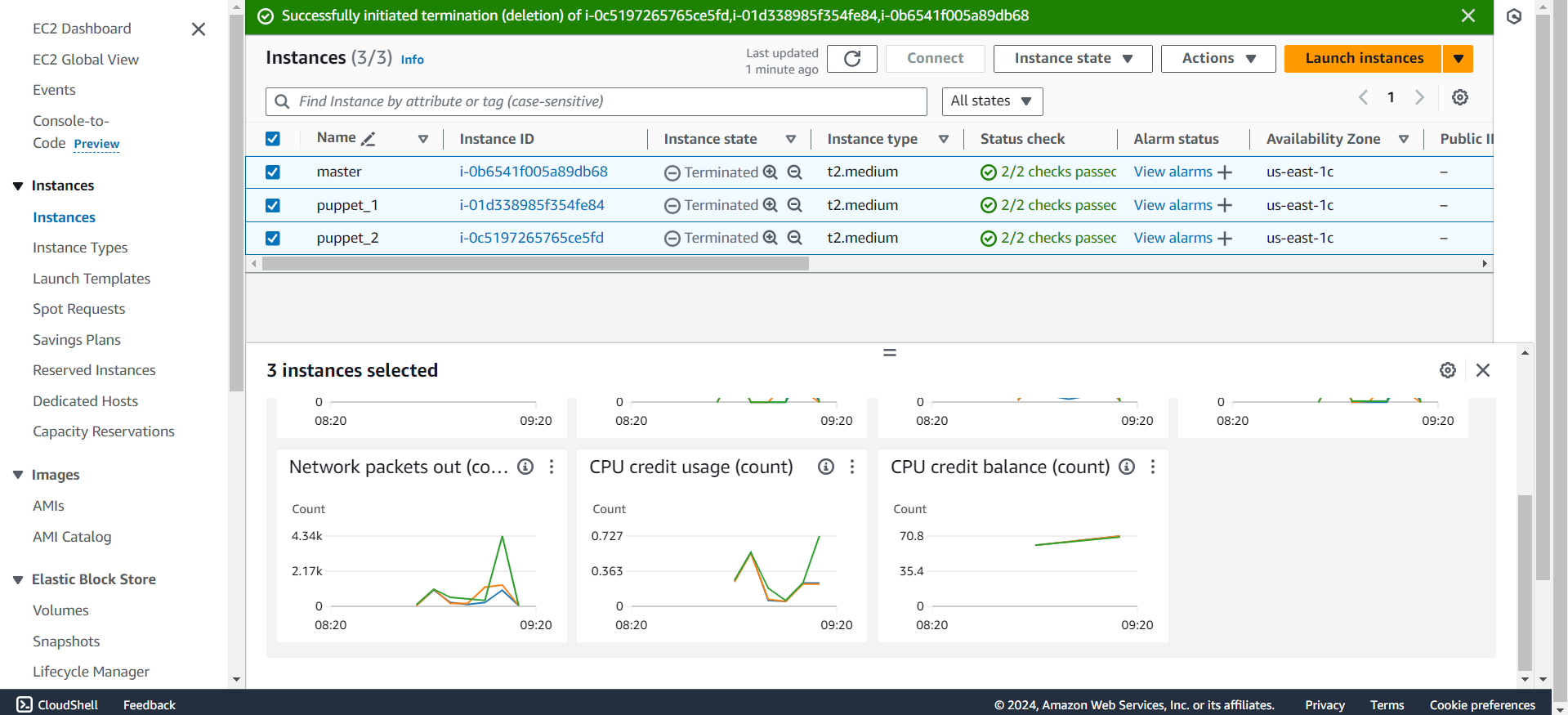
Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage the following activities:

* 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

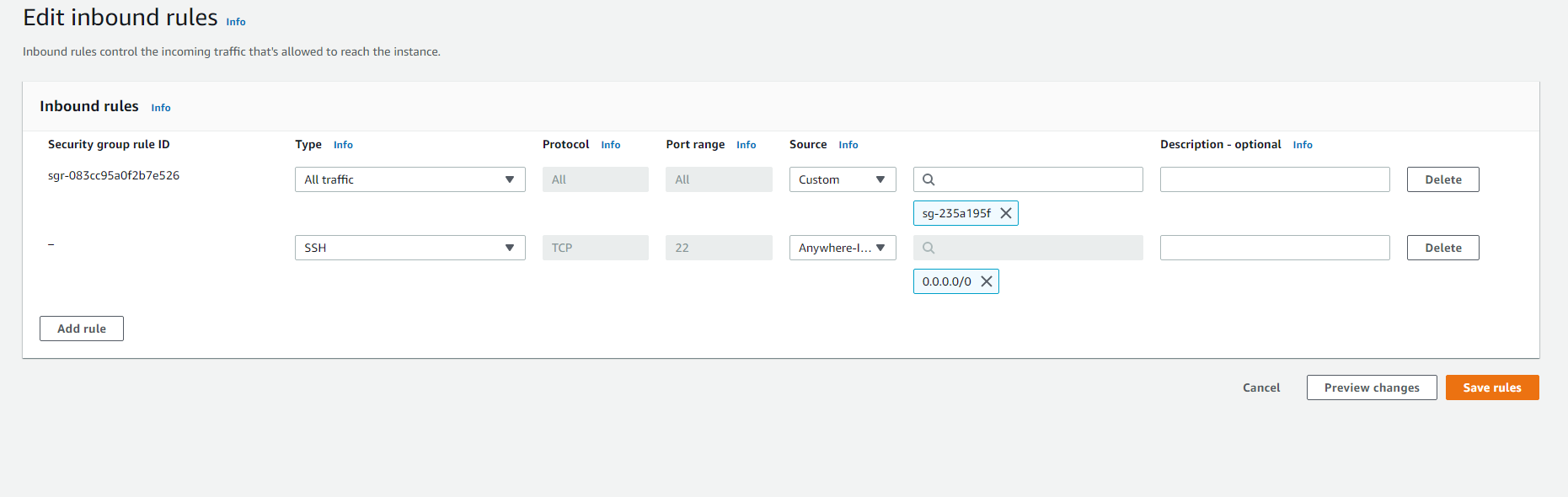
# Steps:

1. Create 3 EC2 Ubuntu Instances on AWS.

(Name 1 as Master, the other 2 as puppet1,2)

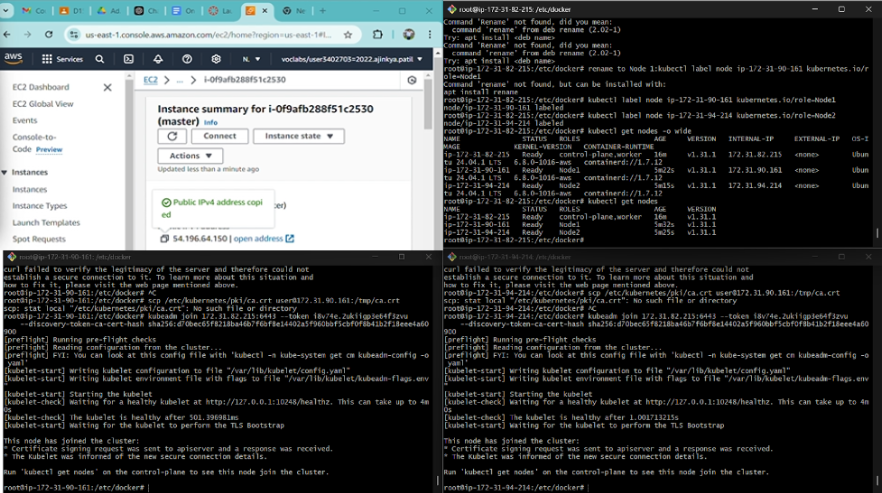


1. Edit the Security Group Inbound Rules to allow SSH



1. SSH into all 3 machines

# ssh -i <keyname>.pem ubuntu@<public\_ip\_address>



1. 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

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",

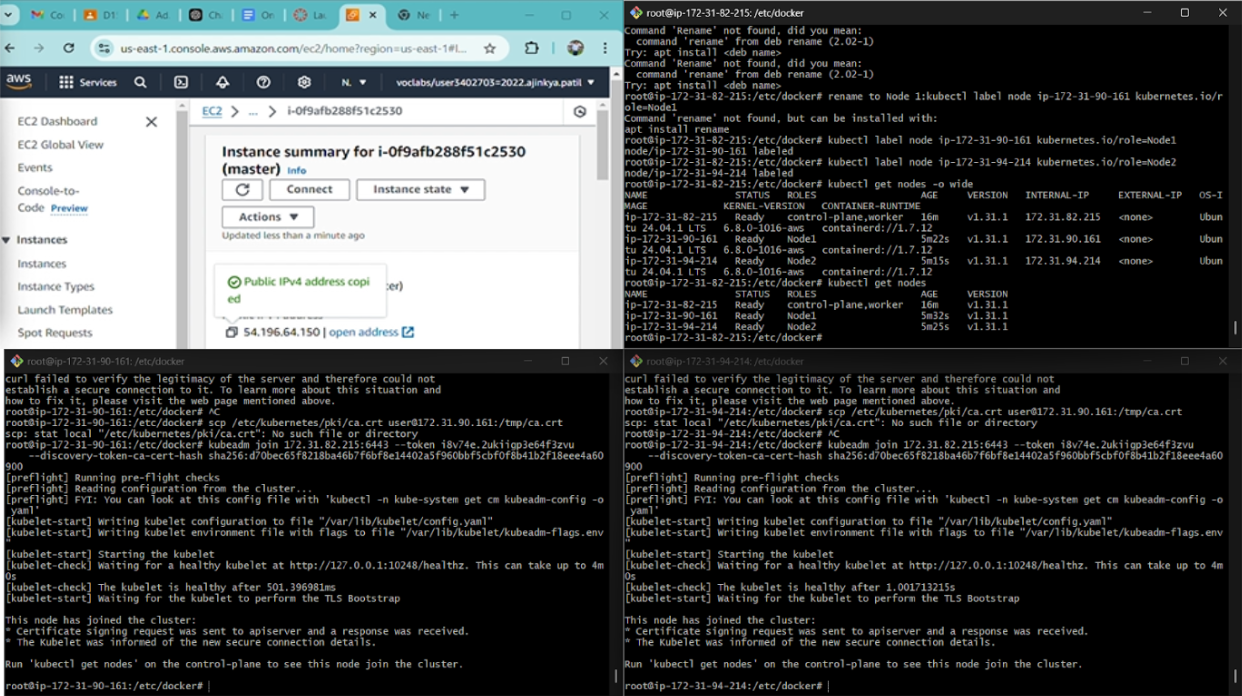
"log-opts": {

"max-size": "100m"

},

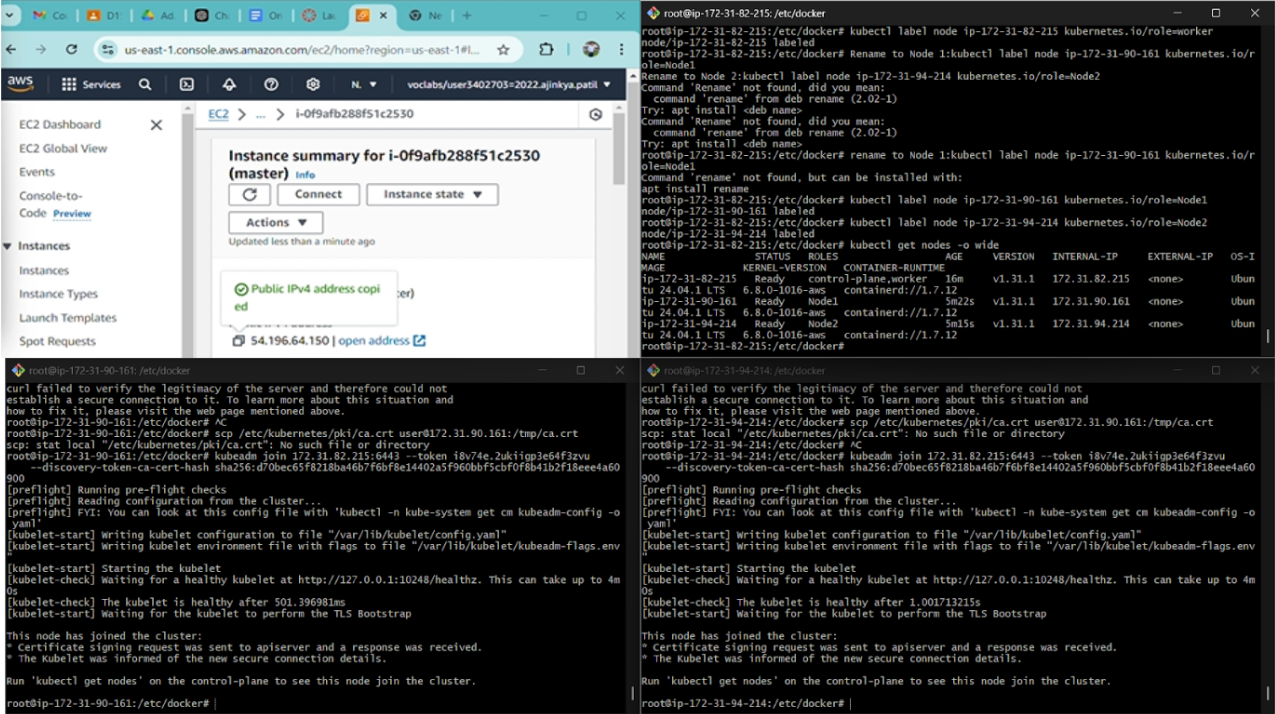
"storage-driver": "overlay2"

}

EOF

sudo systemctl enable docker sudo systemctl daemon-reload sudo systemctl restart docker

Install Kubernetes on all 3 machines



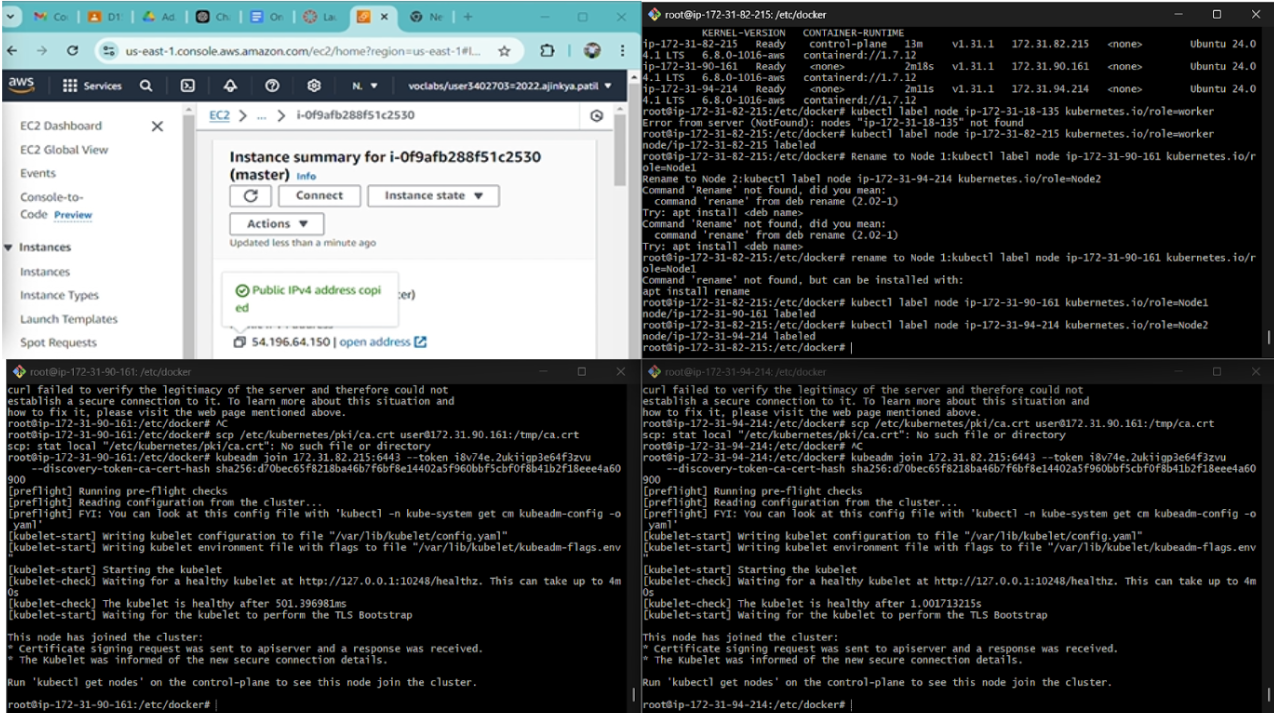
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

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



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

sudo swapoff -a

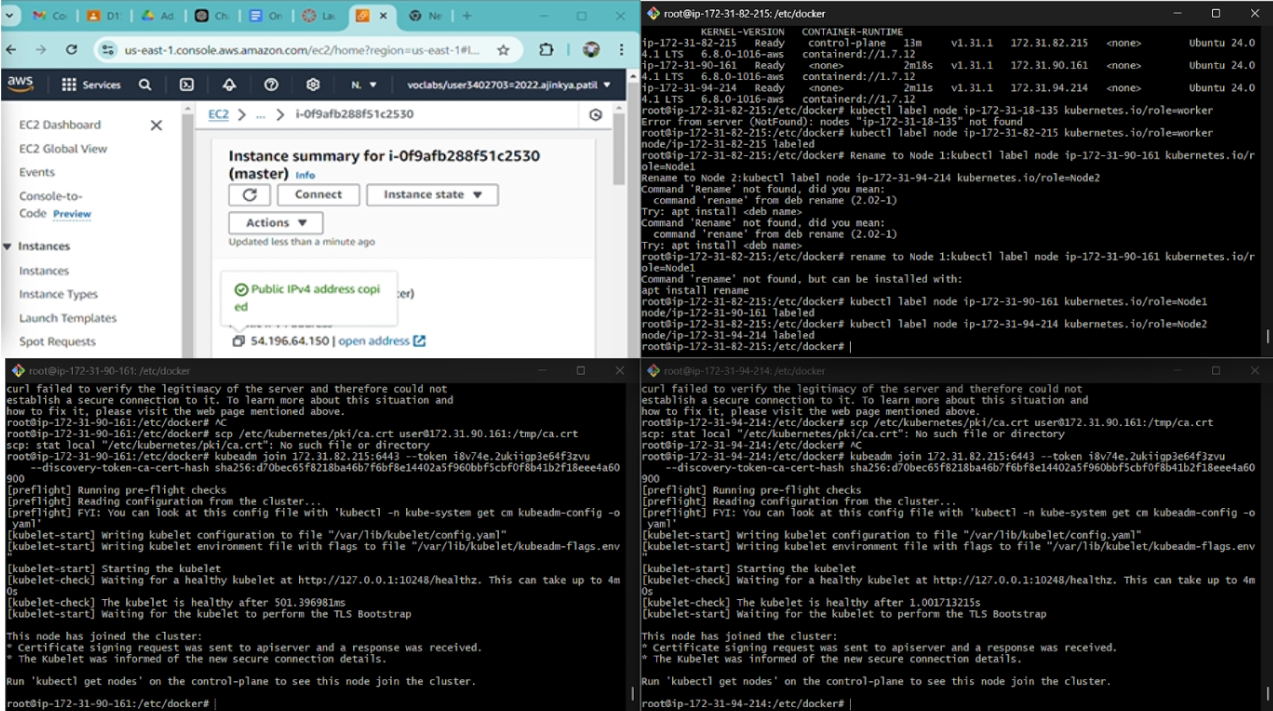
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p

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

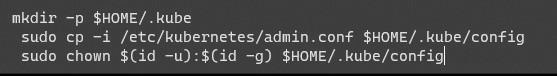
Initialize the Kubecluster

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.

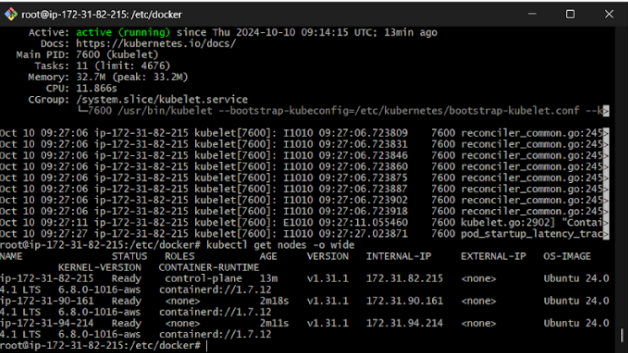
Copy the mkdir and chown commands from the top and execute them



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/kube-flannel.yml

activate running status



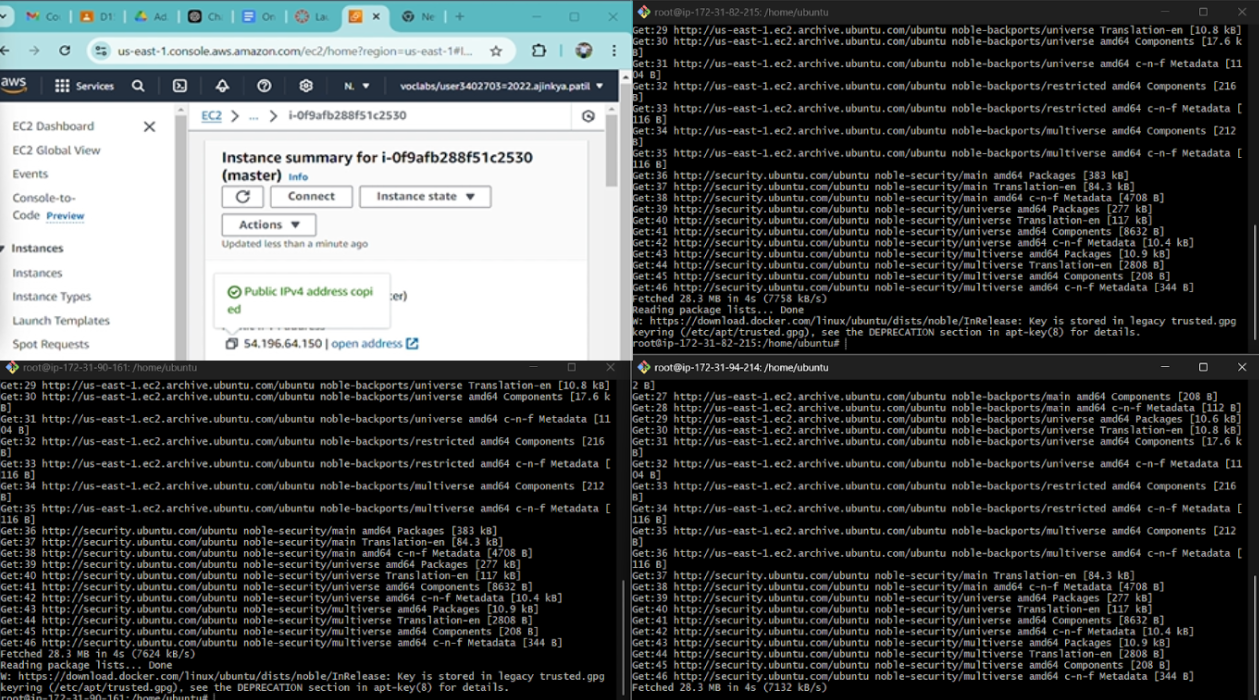
Check the created pod using this command

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

watch kubectl get nodes

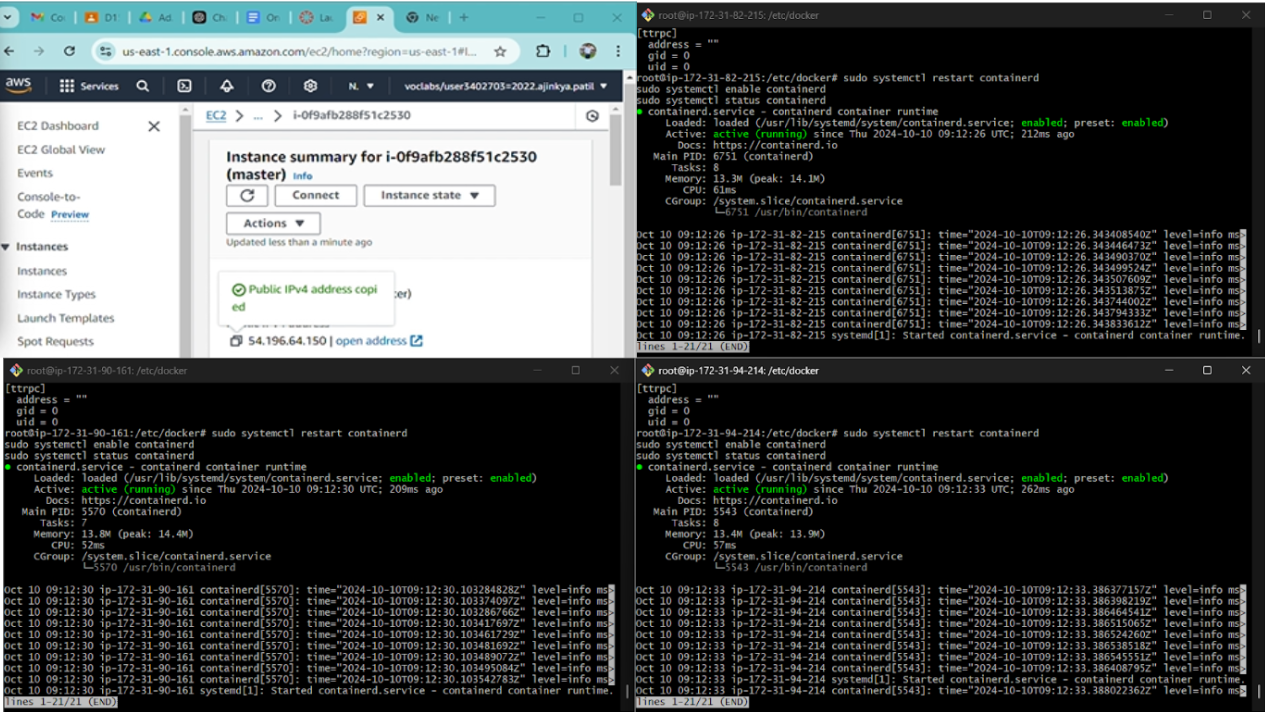
1. Perform this **ONLY on the worker machines**

sudo kubeadm join <ip> --token <token> \



--discovery-token-ca-cert-hash <hash>

Now, notice the changes on the master terminal



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:

In this experiment, we learned how to install Kubernetes create a Kubernetes Cluster in AWS EC2 instances and get them up and running.