Name: Shivpratik Hande **Div/Roll no.:D15C/14**

Advanced DevOps Lab

Experiment No: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:

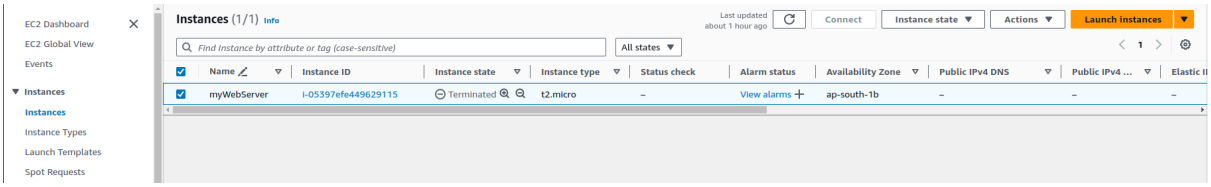
- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications

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

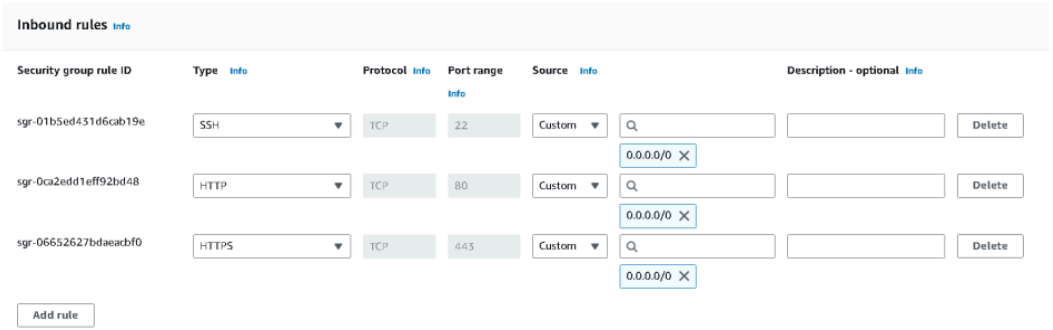
Steps:

1. Create 3 EC2 Ubuntu Instances on AWS.

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



2. Edit the Security Group Inbound Rules to allow SSH



3.SSH into all 3 machines

a. You can do it through the aws console directly
or
b. Locate your key from the Downloads folder and open it in cmd and paste this command
ssh -i <-your-key->.pem ec2-user<ip-address of instance>

Name: Shivpratik Hande Div/Roll no.:D15C/14

3. 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

```
index.html
ubuntu@ip-172-31-13-78:/var/www/html$ curl -fsSL https://download.docker.com/li
ux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubu
tu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install -y docker-ce
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (
ee apt-key(8)).
OK
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble st
ble'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_c
m_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_down
oad_docker_com_linux_ubuntu-noble.list
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
```

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

```
ubuntu@ip-172-31-13-78:/$ curl -s https://packages.cloud.google.com/apt/doc/apt
key.gpg | sudo apt-key add -
sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (
ee apt-key(8)).
OK
deb https://apt.kubernetes.io/ kubernetes-xenial main
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [
26 kB]
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64
ackages [530 kB]
Get:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe am
64 Packages [374 kB]
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Tr
nslation-en [154 kB]
Ign:7 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
```

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

4. Perform this ONLY on the Master machine
Initialize the Kubecluster

Name: Shivpratik Hande Div/Roll no.:D15C/14

sudo kubeadm init --pod-network-cidr=10.244.0.0/16
--ignore-preflight-errors=all

```
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 172.31.45.229:6443 --token s9zq75.bsi7js5f62ridulc \
        --discovery-token-ca-cert-hash sha256:91eae090fdd49337bf70d5bf7478e60bc85820d0996651871129a082db6fa8f1
ubuntu@ip-172-31-45-229:~$
```

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

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

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

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

```
ubuntu@ip-172-31-45-229:~$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.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
```

Check the created pod using this command
Now, keep a watch on all nodes using the following command
watch kubectl get nodes
5. 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

```
Every 2.0s: kubectl get nodes
24
NAME              STATUS   ROLES           AGE   VERSION
ip-172-31-45-229  Ready    control-plane   28m   v1.31.1
```

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 setup, we established a Kubernetes cluster using three AWS EC2 instances, successfully deploying Docker and Kubernetes components on each. While the master node is operational and the initial configuration is complete, the worker nodes are encountering challenges when attempting to join the cluster. These issues appear to stem from configuration or networking problems. To finalize the cluster setup and ensure its proper functionality, further troubleshooting on the worker nodes is necessary to resolve these connectivity issues. Once these challenges are addressed, the cluster will be fully operational, allowing for efficient management and scaling of containerized applications across the instances.