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:

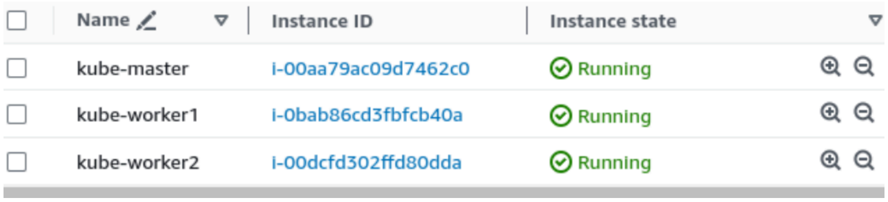
- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application ● Monitoring resource consumption and resource limits to automatically stop applications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

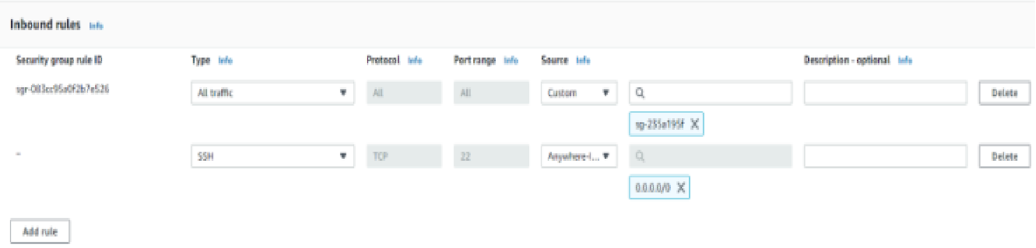
1. Create 3 EC2 Ubuntu Instances on AWS.

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



□	Name	Instance ID	Instance state
□	kube-master	i-00aa79ac09d7462c0	Running
□	kube-worker1	i-0bab86cd3fbfcb40a	Running
□	kube-worker2	i-00dcfd302ffd80dda	Running

2. Edit the Security Group Inbound Rules to allow SSH

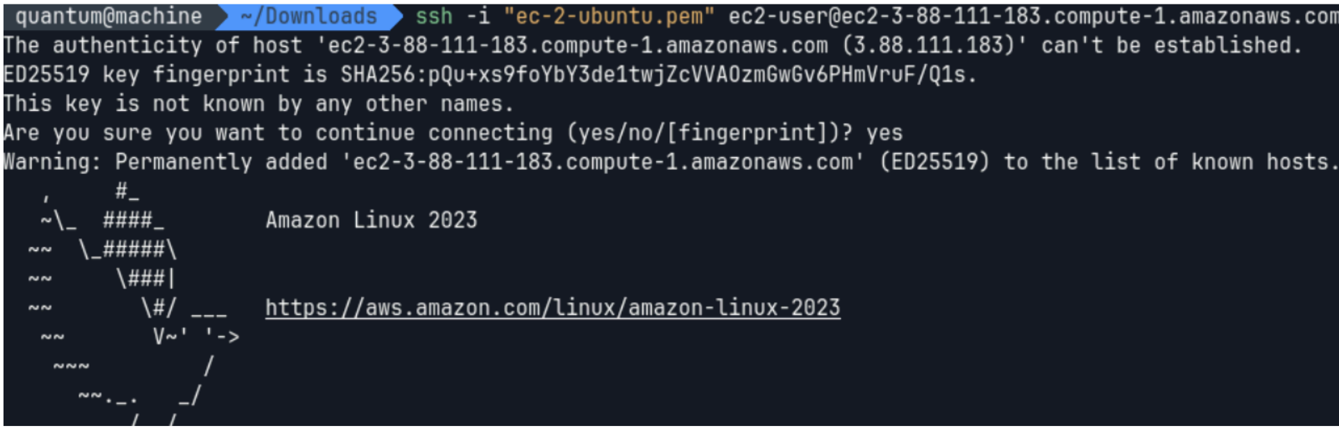


Inbound rules Info

Security group rule ID	Type	Protocol	Port range	Source		Description - optional	
sgr-083cc95a0f2b7e526	All traffic	All	All	Custom	sg-235a195f		Delete
-	SSH	TCP	22	Anywhere-I...	0.0.0.0/0		Delete

3. SSH into all 3 machines

   ssh -i <keyname>.pem ubuntu@<public_ip_address>



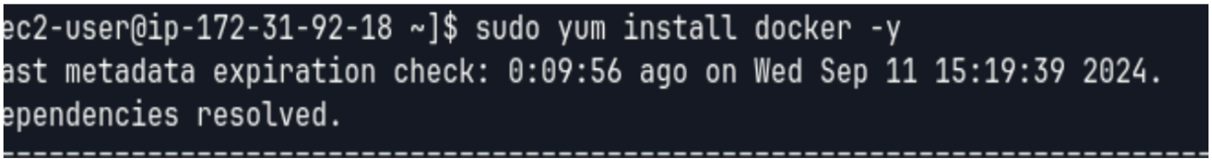
```
 quantum@machine  ~/Downloads  ssh -i "ec-2-ubuntu.pem" ec2-user@ec2-3-88-111-183.compute-1.amazonaws.com
The authenticity of host 'ec2-3-88-111-183.compute-1.amazonaws.com (3.88.111.183)' can't be established.
ED25519 key fingerprint is SHA256:pQu+xs9foYbY3de1twjZcVVA0zmGwGv6PHmVruF/Q1s.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-88-111-183.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
```

4. 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
```



```
ec2-user@ip-172-31-92-18 ~]$ sudo yum install docker -y
ast metadata expiration check: 0:09:56 ago on Wed Sep 11 15:19:39 2024.
ependencies resolved.
--------------------------------------------------------------------------------
```

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.

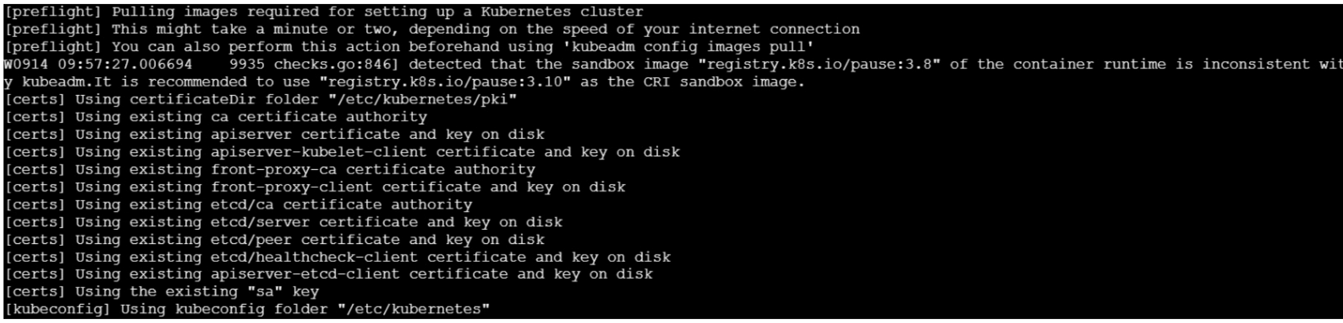
```
       sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
       sudo sysctl -p
```

5. 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



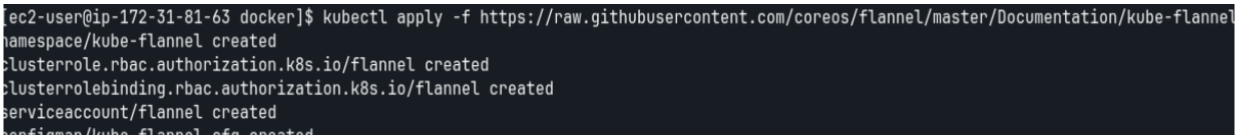
```
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0914 09:57:27.006694    9935 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent wit
y kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Using existing ca certificate authority
[certs] Using existing apiserver certificate and key on disk
[certs] Using existing apiserver-kubelet-client certificate and key on disk
[certs] Using existing front-proxy-ca certificate authority
[certs] Using existing front-proxy-client certificate and key on disk
[certs] Using existing etcd/ca certificate authority
[certs] Using existing etcd/server certificate and key on disk
[certs] Using existing etcd/peer certificate and key on disk
[certs] Using existing etcd/healthcheck-client certificate and key on disk
[certs] Using existing apiserver-etcd-client certificate and key on disk
[certs] Using the existing "sa" key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
```

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/
k ube-flannel.yml
```



```
[ec2-user@ip-172-31-81-63 docker]$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
```

Check the created pod using this command

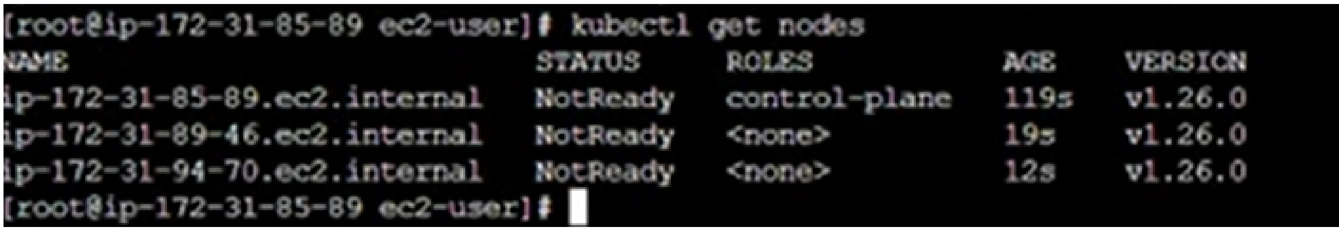
Now, keep a watch on all nodes using the following command

```
watch kubectl get nodes
```

6. 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



```
[root@ip-172-31-85-89 ec2-user]# kubectl get nodes
NAME                           STATUS     ROLES           AGE    VERSION
ip-172-31-85-89.ec2.internal   NotReady   control-plane   119s   v1.26.0
ip-172-31-89-46.ec2.internal   NotReady   <none>          19s    v1.26.0
ip-172-31-94-70.ec2.internal   NotReady   <none>          12s    v1.26.0
[root@ip-172-31-85-89 ec2-user]#
```

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, the goal was to set up and configure a Kubernetes cluster using kubeadm on a Linux environment with yum as the package manager.