

Advanced DevOps Lab

Experiment 3

Roll No.	24
Name	Iyer Sreekesh Subramanian
Class	D15-A
Subject	Advanced DevOps Lab

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.

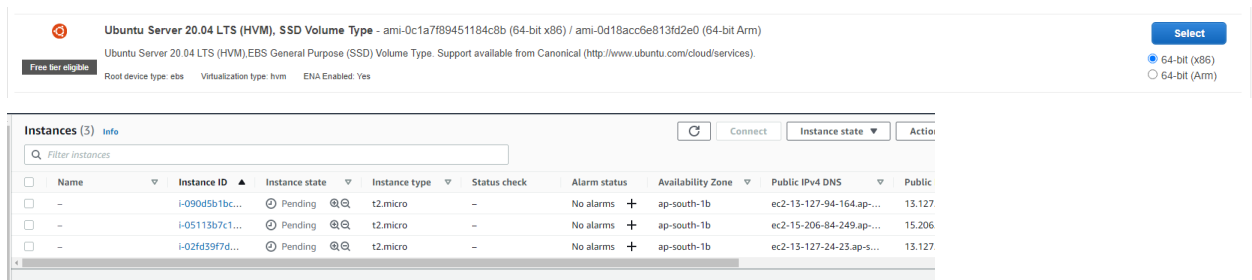
Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage the following activities:

- 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

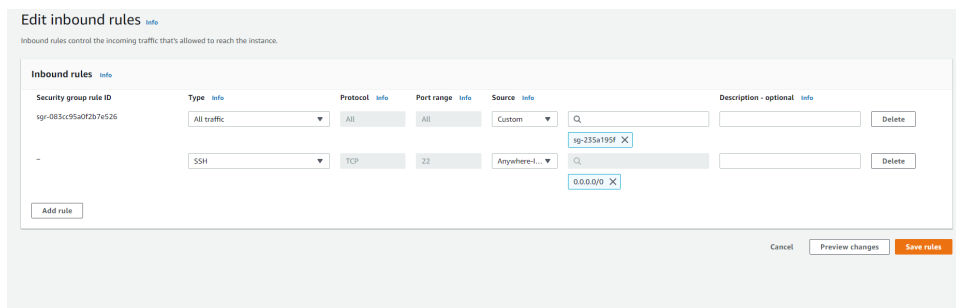
Steps:

1. Create 3 EC2 Ubuntu Instances on AWS.

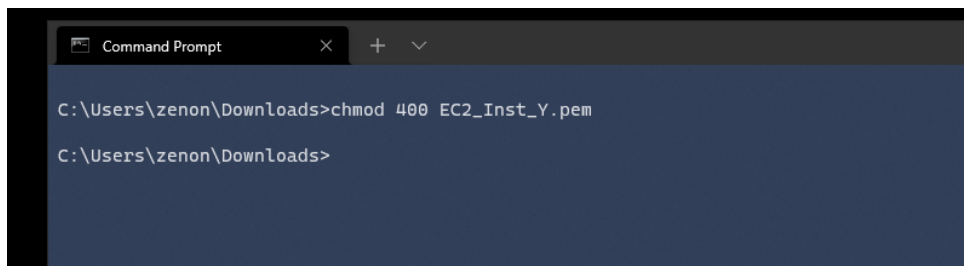
(Name 1 as Master, the other 2 as worker-1 and worker-2)



2. Edit the Security Group Inbound Rules to allow SSH



3. SSH into all 3 machines



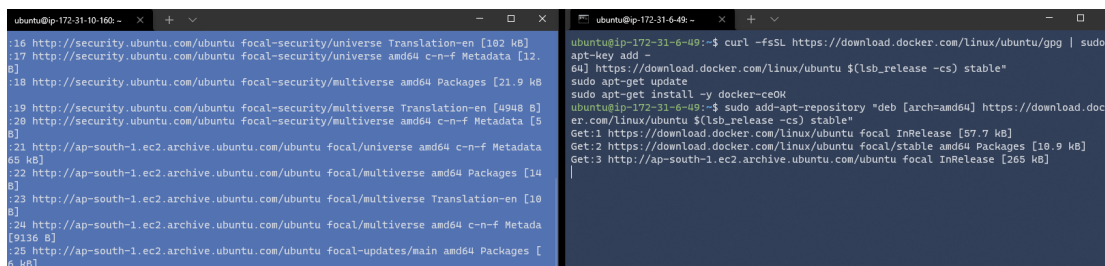
ssh -i <keyname>.pem ubuntu@<public_ip_address>

```
C:\Users\zenon\Downloads>ssh -i EC2_Inst_Y.pem ubuntu@13.127.94.164
The authenticity of host '13.127.94.164 (13.127.94.164)' can't be established.
ECDSA key fingerprint is SHA256:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

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
```



Then, configure cgroup in a daemon.json file.

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

5. 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-172-31-40:~$ sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
ubuntu@ip-172-31-40:~$ sudo apt-get update
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [265 kB]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease [9383 B]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 https://packages.cloud.google.com/apt/kubernetes-xenial InRelease [9383 B]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [181 kB]
Get:6 https://packages.cloud.google.com/apt/kubernetes-xenial/main amd64 Packages [59.0 kB]
Fetched 339 kB in 1s (480 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-40:~$ sudo apt-get install -y kubelet kubeadm kubectl

ubuntu@ip-172-31-10-100:~$ sudo apt-get update
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [265 kB]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease [9383 B]
Get:3 https://packages.cloud.google.com/apt/kubernetes-xenial InRelease [9383 B]
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [181 kB]
Get:6 https://packages.cloud.google.com/apt/kubernetes-xenial/main amd64 Packages [59.0 kB]
Fetched 339 kB in 1s (480 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-10-100:~$ sudo apt-get install -y kubelet kubeadm kubectl

ubuntu@ip-172-31-6-48:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo apt-key add -
gpg: key 0x034989EE: public key "Kubernetes Release Key" imported
gpg: Total number processed: 1
gpg:   imported: 1 (0 new, 1 reused, 0 failed)
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
ubuntu@ip-172-31-6-48:~$ sudo apt-get update
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [265 kB]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease [9383 B]
Get:3 https://packages.cloud.google.com/apt/kubernetes-xenial InRelease [9383 B]
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [181 kB]
Get:6 https://packages.cloud.google.com/apt/kubernetes-xenial/main amd64 Packages [59.0 kB]
Fetched 339 kB in 1s (480 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-6-48:~$ 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
```

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

Initialize the Kubecluster

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
ubuntu@ip-172-31-10-100:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```

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 <ip> --token <token> \
  --discovery-token-ca-cert-hash sha256:ab3fb9f05059f2f4829be48caa1830254e6c49218c9aeffc4

```

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 flannel file as mentioned in the code.

```

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

ubuntu@ip-172-31-4-0:/etc/docker$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```

Check the created pod using this command

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

```
watch kubectl get nodes
```

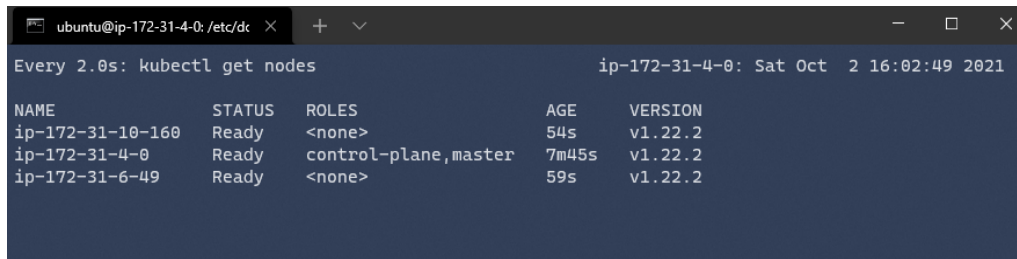
7. Perform this **ONLY** on the worker machines

```

sudo kubeadm join <ip> --token <token> \
  --discovery-token-ca-cert-hash <hash>

```

8. Now, notice the changes on the master terminal

A terminal window titled 'ubuntu@ip-172-31-4-0: /etc/dc' with standard window controls. The command 'Every 2.0s: kubectl get nodes' is shown at the top. The output is a table with 5 columns: NAME, STATUS, ROLES, AGE, and VERSION. It lists three nodes: 'ip-172-31-10-160' (Ready, <none>, 54s, v1.22.2), 'ip-172-31-4-0' (Ready, control-plane,master, 7m45s, v1.22.2), and 'ip-172-31-6-49' (Ready, <none>, 59s, v1.22.2). The terminal background is dark blue.

NAME	STATUS	ROLES	AGE	VERSION
ip-172-31-10-160	Ready	<none>	54s	v1.22.2
ip-172-31-4-0	Ready	control-plane,master	7m45s	v1.22.2
ip-172-31-6-49	Ready	<none>	59s	v1.22.2

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.