Name - Mohit Shadija
Div - D15B
Roll - 54

Adv devops 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.

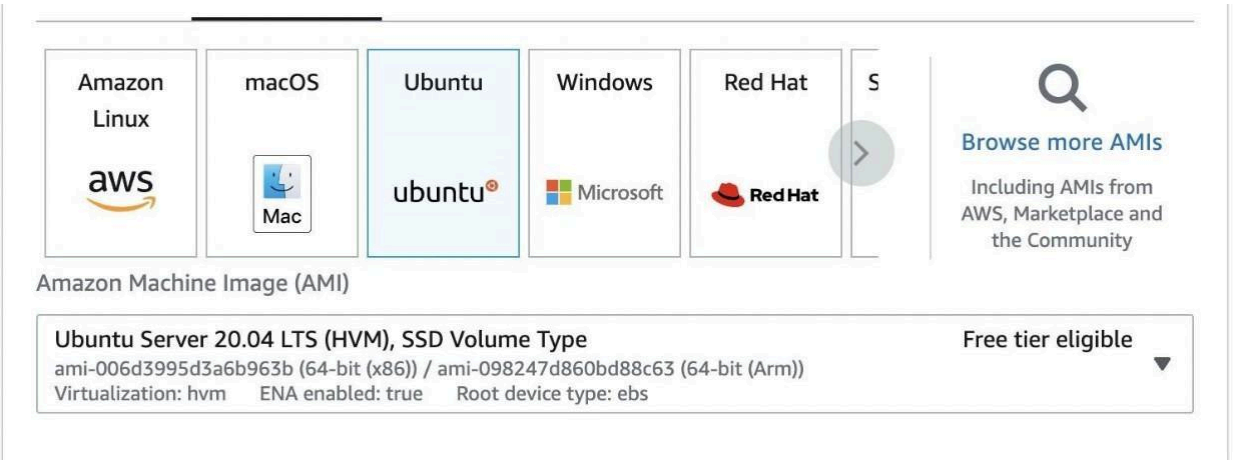
Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage the following activities:

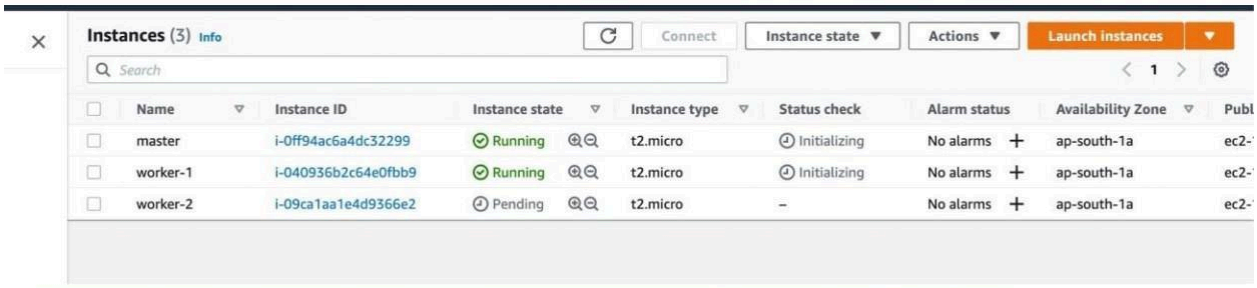
- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications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

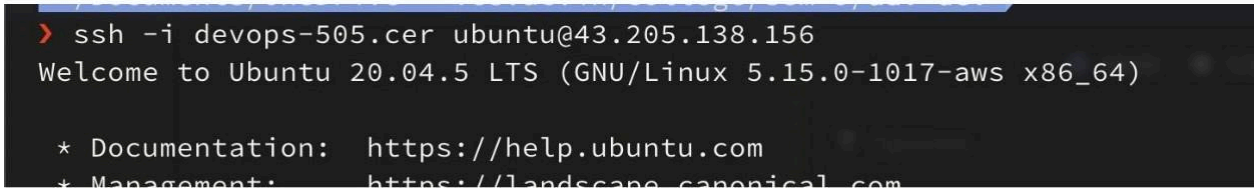
Step 1:

Create 3 EC2 Ubuntu Instances on AWS.

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

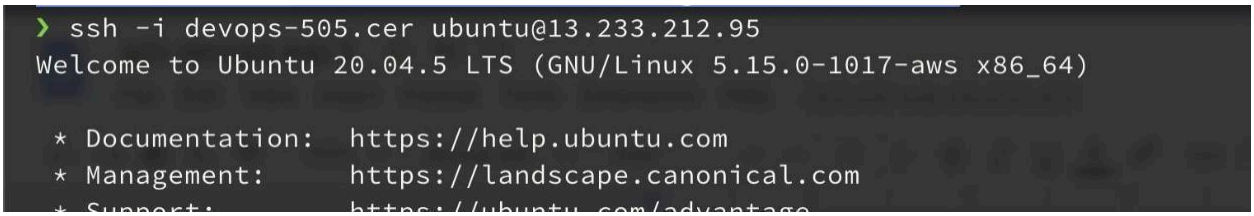






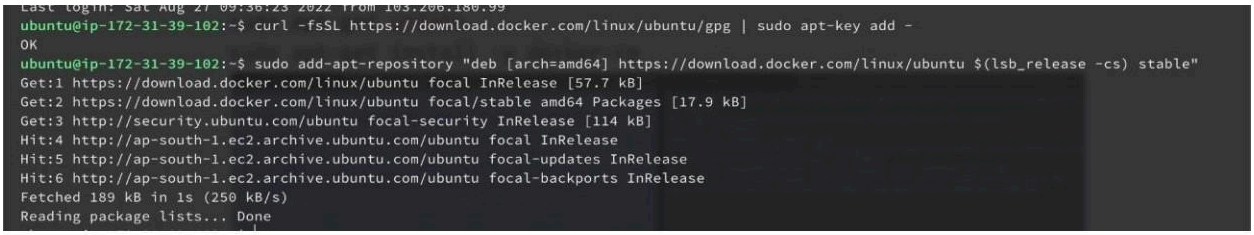
Step 2:

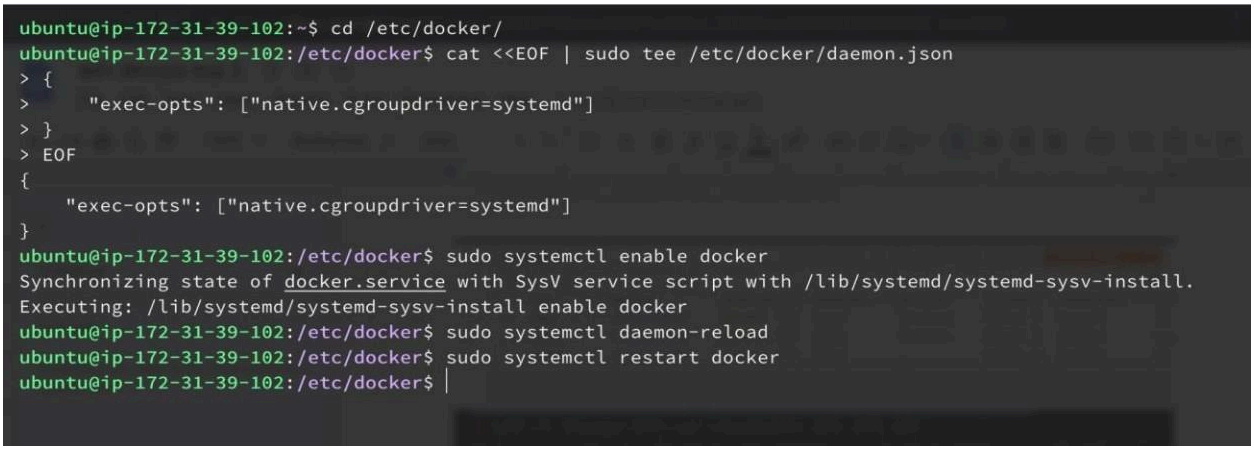
SSH into all 3 machines



Step 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





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

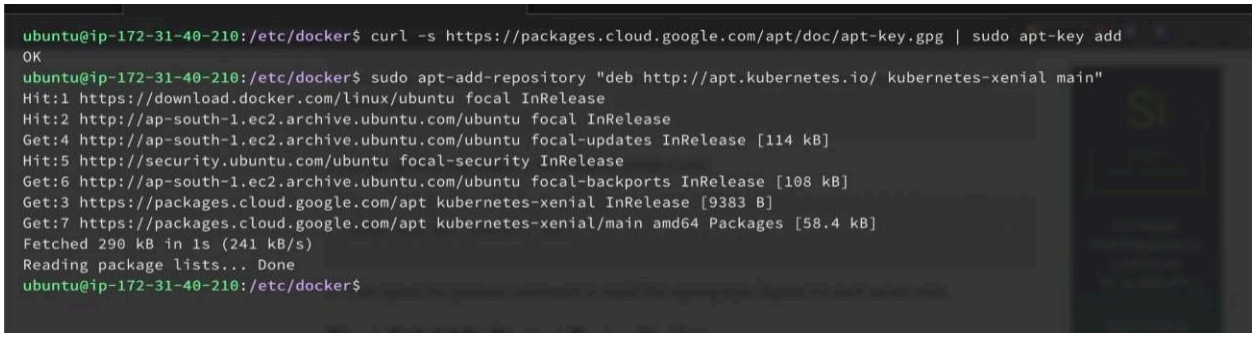
sudo systemctl restart docker

Step 4:

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

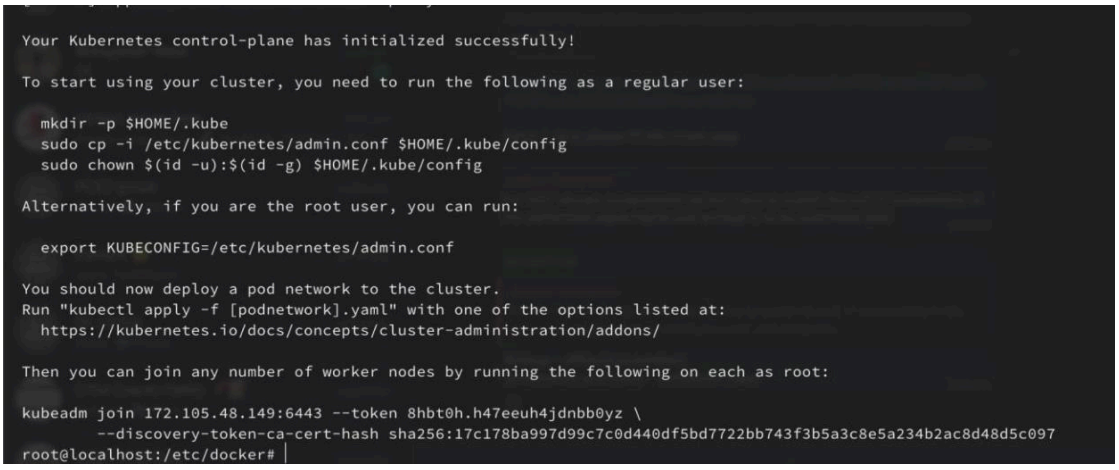


After installing Kubernetes, we need to configure internet options to allow
bridging. sudo swapoff -a echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a
/etc/sysctl.conf sudo
sysctl -p

Step 5:

Perform this ONLY on the Master machine

Initialize the Kubecluster sudo kubeadm init --podnetwork-cidr=10.244.0.0/16



kubeadm join 172.105.48.149:6443 --token mwjrs9.n9m68s5x1a3r0dka
--discovery-token-ca-cert-hash

sha256:94174c906096688053622fcc60e48b71b934a4074ce79c5f2334f69016ae9a19

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 ubeflannel.yml

```
root@localhost:/etc/docker# 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 **Step 6:**

Perform this ONLY on the worker machines

sudo kubeadm join --token \ --discovery-token-ca-cert-hash

```
root@master:~# kubectl get nodes
NAME     STATUS   ROLES           AGE     VERSION
master   Ready    control-plane   3m58s   v1.25.0
root@master:~#
```

Step 7:

Now, notice the changes on the master terminal

```
root@master:~# kubectl get nodes
NAME        STATUS     ROLES           AGE     VERSION
localhost   Ready      <none>          8m27s   v1.25.0
master      Ready      control-plane   122m    v1.25.0
```

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.