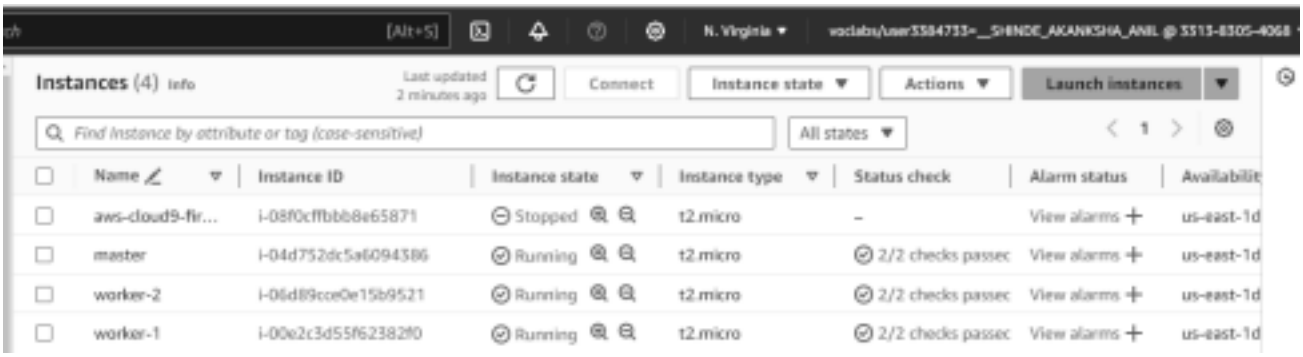
Experiment: 3

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

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 and do it for all the three machines.

```
Akanksha Shinde@AkankshaShinde MINGW64 ~/downloads (master)
$ ssh -i "server.pem" ec2-user@ec2-54-174-206-93.compute-1.amazonaws.com
The authenticity of host 'ec2-54-174-206-93.compute-1.amazonaws.com (54.174.206.
93)' can't be established.
ED25519 key fingerprint is SHA256:T+tsGyI15gAvUvjeAZ7GjDIWXHOaI4EPF5g5oICrkoQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-174-206-93.compute-1.amazonaws.com' (ED25519)
 to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
```

3. From now on, until mentioned, perform these steps on all 3 machines. Install Docker for all the 3 machines

```
[root@ip-172-31-90-172 ec2-user]# yum install docker -y
Last metadata expiration check: 0:21:16 ago on Fri Aug 30 04:01:12 2024.
Dependencies resolved.
```

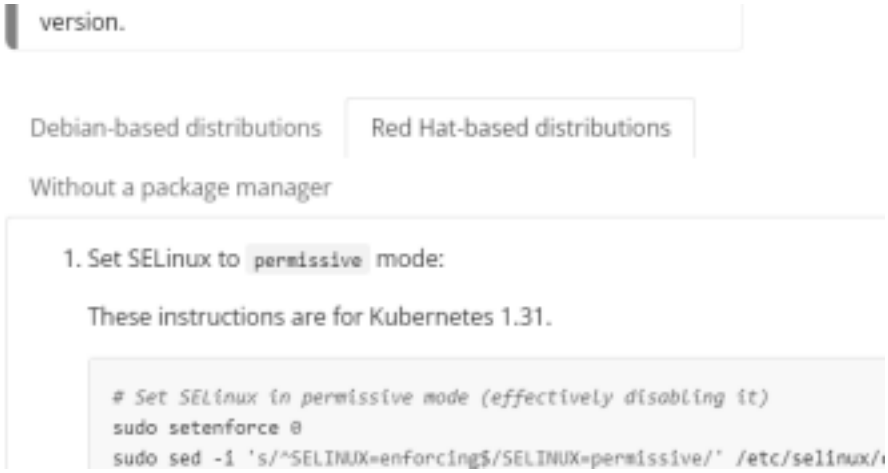
Package	Architecture	Version
Installing:		
docker	x86_64	25.0.6-1.amzn2023.0.1
Installing dependencies:		
containerd	x86_64	1.7.20-1.amzn2023.0.1
iptables-libs	x86_64	1.8.8-3.amzn2023.0.2
iptables-nft	x86_64	1.8.8-3.amzn2023.0.2
libcgroup	x86_64	3.0-1.amzn2023.0.1
libnetfilter_conntrack	x86_64	1.0.8-2.amzn2023.0.2
libnfnetlink	x86_64	1.0.1-19.amzn2023.0.2
libnftnl	x86_64	1.2.2-2.amzn2023.0.2
pigz	x86_64	2.5-1.amzn2023.0.3

Start the docker by running the command systemctl start docker in the terminal of all the ec2 instance.

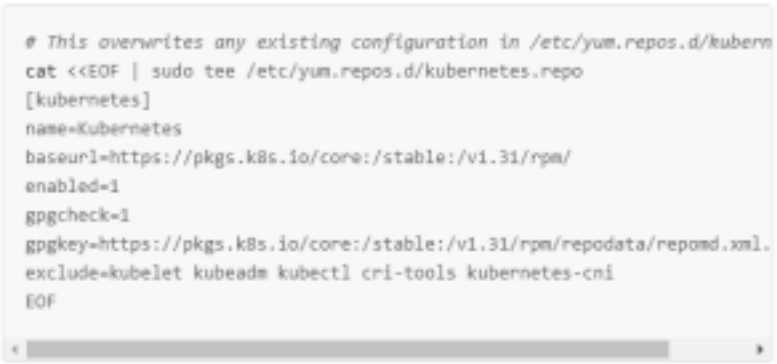
```
Complete!
[root@ip-172-31-82-133 ec2-user]# systemctl start docker
[root@ip-172-31-82-133 ec2-user]#
```

4. Install the kubernetes on all 3 machines by searching for kubeadm and click on install kubernetes.

Select the red hat based distribution. This process will automatically disable SELinux before configuring kubelet so no need to run it separately in terminal**.**



Copy the below script, to install kubernetes we need a kubernetes repo so this script helps us in getting that and paste it in the terminal.



```
Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64       docker-25.0.6-1.amzn2023.0.1.x86_64      iptables-libs-1
  iptables-nft-1.8.8-3.amzn2023.0.2.x86_64      libcgroup-3.0-1.amzn2023.0.1.x86_64      libnetfilter_cc
  libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64     libnftnl-1.2.2-2.amzn2023.0.2.x86_64     pigz-2.5-1.amzn
  runc-1.1.11-1.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-90-172 ec2-user]# systemctl start docker
[root@ip-172-31-90-172 ec2-user]# sudo su
[root@ip-172-31-90-172 ec2-user]# yum repolist
repo id                                                    repo name
amazonlinux                                                Amazon Linux 2023 repository
```

Run the command yum repolist to check whether the kubernetes repo has installed or not if successful installed then you can see a repo named as kubernetes

```
[root@ip-172-31-90-172 ec2-user]# yum repolist
repo id                                                    repo name
amazonlinux                                                Amazon Linux 2023 repository
kernel-livepatch                                           Amazon Linux 2023 Kernel Livepatch repository
kubernetes                                                 Kubernetes
[root@ip-172-31-90-172 ec2-user]#
```

Do the above steps for all the instances i.e for worker-1 and worker-2.

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

```
[addons] Applied essential addon: kube-proxy

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

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 172.31.12.28:6443 --token 4bqwb8.lua2ud01lr02uu55 \
    --discovery-token-ca-cert-hash sha256:b4edc7948be9bca50767f623b58e0612feedc144a7364f95be8dbd8c4614a169
```

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

```
[ec2-user@ip-172.31.12.28 docker]$ 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

```
[ec2-user@ip-172.31.12.28 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

6. Perform this ONLY on the worker machines

Run the following command

sudo yum install iproute-tc -y

sudo systemctl enable kubelet
sudo systemctl restart kubelet

Check the status of the pods using the following command
This command will show the status of all the pods.
kubectl get pods -n kube-system
Following command will show the status of the pod named daemonset.
kubectl get daemonset -n kube-system

```
[ec2-user@ip-172.31.12.28 docker]$ kubectl get pods -n kube-system
NAME                                                READY   STATUS             RESTARTS   AGE
coredns-55cb5b8774-fx12f                            1/1     Running            0          100s
coredns-55cb5b8774-xn14v                            1/1     Running            0          100s
etcd-ip-172.31.12.28.ec2.internal                   1/1     Running            0          75s
kube-apiserver-ip-172.31.12.28.ec2.internal         1/1     Running            1          2m
kube-controller-manager-ip-172.31.12.28.ec2.internal 0/1 CrashLoopBackOff   1              70s
kube-proxy-4dv8m                                    1/1     Running            2          100s
kube-scheduler-ip-172.31.12.28.ec2.internal         1/1     Running            1          76s

[ec2-user@ip-172.31.12.28 docker]$ kubectl get daemonset -n kube-system
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
kube-proxy   1         1         1       1            1           kubernetes.io/os=linux   3m
```

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:

Kubernetes cluster was successfully established using three AWS EC2 instances, which includes one Master and two Worker nodes. The process began with the creation of instances and configuration of settings to begin the communication. Docker was installed on all machines followed by the installation of Kubernetes components and the necessary repositories. The Master node was initialized with the `kubeadm init` command, and a plugin called Flannel was deployed to enable pod communication. Error incurred during initialization can be solved by changing the instance type to t3.medium or t3.large and thus 3 nodes were connected successfully