Advanced DevOps
Lab Experiment: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:

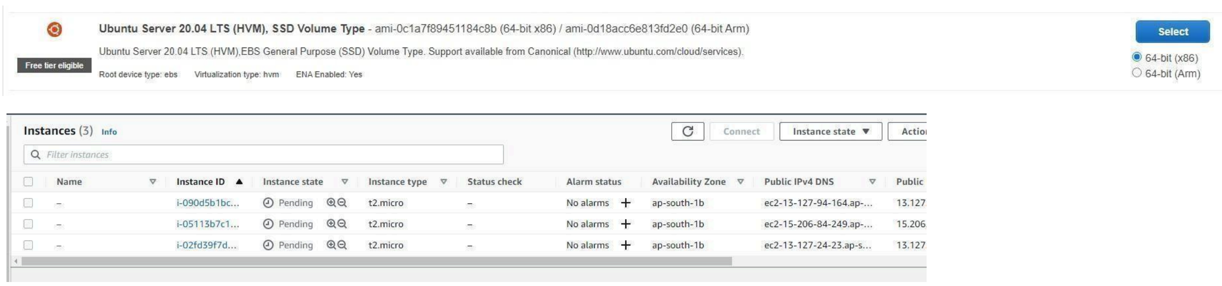
- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications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

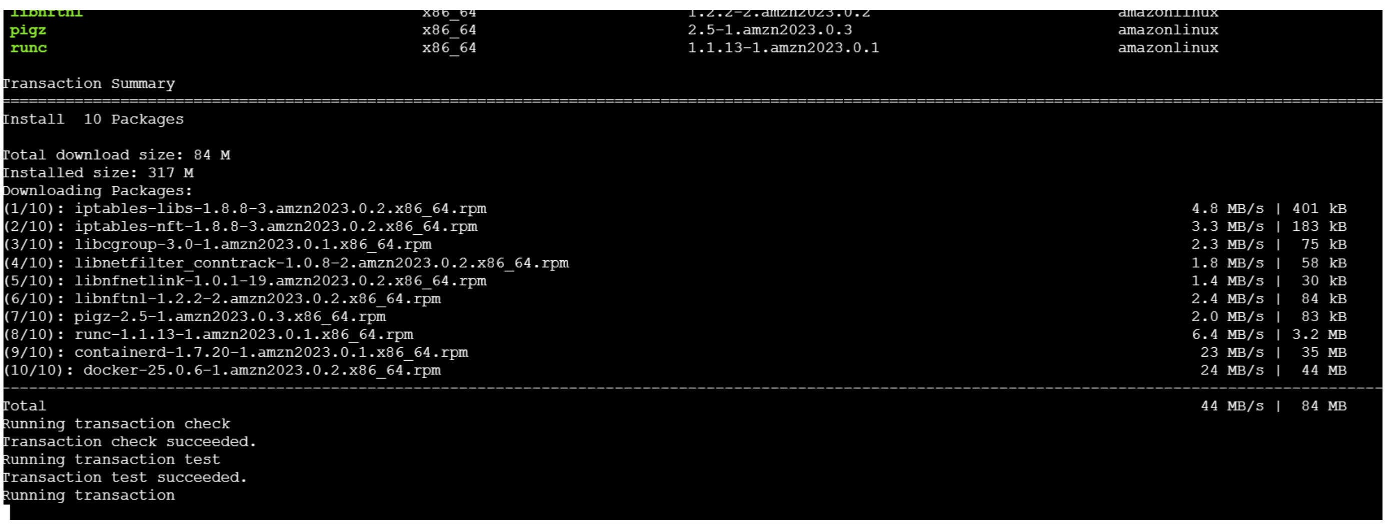
   (Name 1 as Master, the other 2 as worker-1 and worker-2)

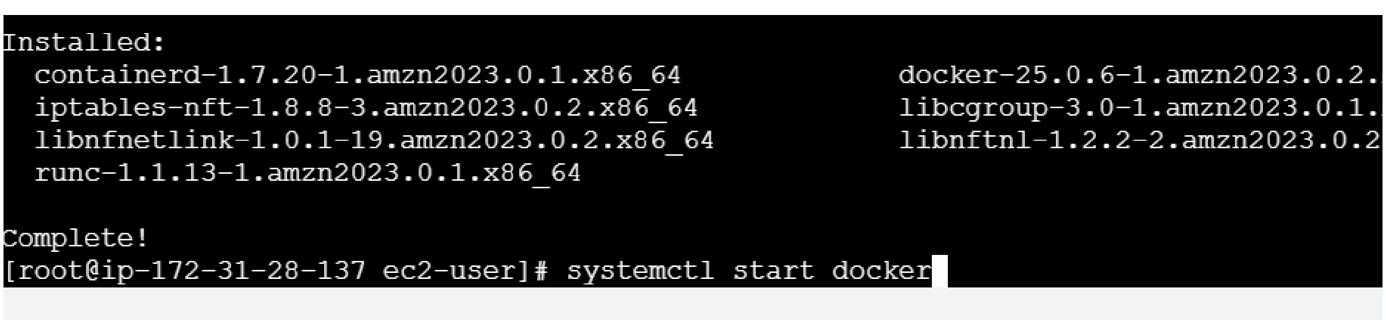


2. From now on, until mentioned, perform these steps on all 3 machines.

   Install Docker

```
Yum install docker -y
```





Install Kubernetes on all 3 machines

```
# Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

# This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes

(Optional) Enable the kubelet service before running kubeadm:
sudo systemctl enable --now kubelet
```

```
Total
Kubernetes
Importing GPG key 0x9A296436:
 Userid     : "isv:kubernetes OBS Project <isv:kubernetes@build.opensuse.org>"
 Fingerprint: DE15 B144 86CD 377B 9E87 6E1A 2346 54DA 9A29 6436
 From       : https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.ke
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
```

```
Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64          cri-tools-1.31.1-150500.1.1.x86_64                  kubeadm-1.31.
  kubectl-1.31.1-150500.1.1.x86_64                     kubelet-1.31.1-150500.1.1.x86_64                    kubernetes-cn
  libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64   libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64  libnetfilter_

Complete!
[root@ip-172-31-29-112 ec2-user]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
[root@ip-172-31-29-112 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-172-31-29-112 ec2-user]#
```

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

   Initialize the Kubecluster

```
[root@ip-172-31-29-112 ec2-user]# kubeadm init
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
[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 connecti
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0914 12:33:27.963484   27696 checks.go:846] detected that the sandbox image "registry.k8s.io
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] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-29-112.ec2.internal kuberne
```

```
kubeadm join 172.31.29.112:6443 --token zmjfoc.fz9s8bca9ulmwqfr \
        --discovery-token-ca-cert-hash sha256:1b3c21ddca2149f5f9168c68e51c1fd8d158818c2d8f678344d9004d9155c27e
[root@ip-172-31-29-112 ec2-user]#  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@ip-172-31-29-112 ec2-user]# export KUBECONFIG=/etc/kubernetes/admin.conf
[root@ip-172-31-29-112 ec2-user]#
```

```
Last login: Sat Sep 14 12:27:27 2024 from 18.206.107.27
[ec2-user@ip-172-31-29-12 ~]$ sudo su
[root@ip-172-31-29-12 ec2-user]# kubeadm join 172.31.29.112:6443 --token zmjfoc.fz9s8bca9ulmwqfr \
        --discovery-token-ca-cert-hash sha256:1b3c21ddca2149f5f9168c68e51c1fd8d158818c2d8f678344d9004d9155c27e
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
```

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 simplifies containerized application management by automating deployment, scaling, and resource allocation. It efficiently handles load balancing, resource optimization, and fault tolerance, making it the standard for scalable and reliable microservice architectures.