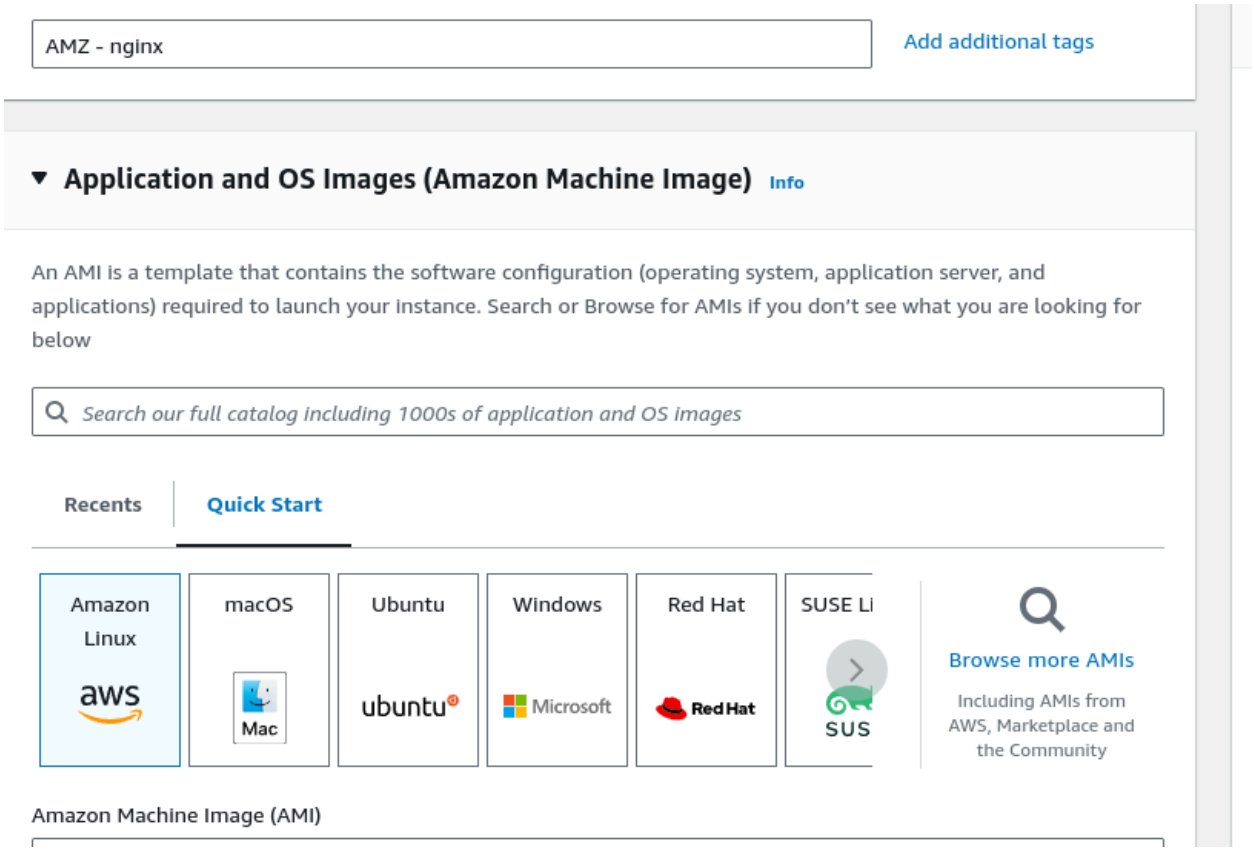
Name: Kshitij Hundre **Div:D15C** **Roll No:18**

EXPERIMENT-4 ADV-DEVOPS

Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

STEPS:

1. Select Amazon linux as OS image (You can use any but then modify commands accordingly)



2. Make ssh connection in terminal
   Note: If you have directly made connection through browser then skip this part

Name: Kshitij Hundre **Div:D15C** **Roll No:18**

```
quantum@machine ~/Downloads ssh -i "ec-2-ubuntu.pem" ec2-user@ec2-54-162-208-25.compute-1.amazonaws.com
The authenticity of host 'ec2-54-162-208-25.compute-1.amazonaws.com (54.162.208.25)' can't be established.
ED25519 key fingerprint is SHA256:lBrdxB+9Hn3kWlOYZWmzh1wg/R4s+e7QDAMBZJPvf/E.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-162-208-25.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-31-63 ~]$ sudo yum update
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[ec2-user@ip-172-31-31-63 ~]$ sudo yum upgrade
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[ec2-user@ip-172-31-31-63 ~]$
```

3. **Install Docker**

sudo dnf update
sudo dnf install docker
sudo systemctl enable docker
sudo systemctl start docker

To test whether docker is successfully running, use command **sudo docker run hello-world**

```
[ec2-user@ip-172-31-24-190 ~]$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Then, configure cgroup in a daemon.json file.This allows kubernetes to manage host more efficiently

```
cd /etc/docker

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

4. Install Kubernetes

Note: I'm directly installing binary package you may install from package repository of your distribution

Install CNI plugins (required for most pod network):

```
CNI_PLUGINS_VERSION="v1.3.0"
ARCH="amd64"
DEST="/opt/cni/bin"
sudo mkdir -p "$DEST"
curl -L
"https://github.com/containernetworking/plugins/releases/download/${CNI_PLUGINS_V
ERSION}/cni-plugins-linux-${ARCH}-${CNI_PLUGINS_VERSION}.tgz" | sudo tar -C
"$DEST" -xz
```

Define the directory to download command files:

```
DOWNLOAD_DIR="/usr/local/bin"
sudo mkdir -p "$DOWNLOAD_DIR"
```

Optionally install crictl (required for interaction with the Container Runtime Interface (CRI), optional for kubeadm):

```
CRICTL_VERSION="v1.31.0"
ARCH="amd64"
```

curl -L "https://github.com/kubernetes-sigs/cri-tools/releases/download/${CRICTL_VERSION}/crictl-${CRICTL_VERSION}-linux-${ARCH}.tar.gz" | sudo tar -C $DOWNLOAD_DIR -xz

Install `kubeadm`, `kubelet` and add a `kubelet` systemd service:

RELEASE="$(curl -sSL https://dl.k8s.io/release/stable.txt)"
ARCH="amd64"
cd $DOWNLOAD_DIR
sudo curl -L --remote-name-all https://dl.k8s.io/release/${RELEASE}/bin/linux/${ARCH}/{kubeadm,kubelet}
sudo chmod +x {kubeadm,kubelet}

RELEASE_VERSION="v0.16.2"
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}/cmd/krel/templates/latest/kubelet/kubelet.service" | sed "s:/usr/bin:${DOWNLOAD_DIR}:g" | sudo tee /usr/lib/systemd/system/kubelet.service
sudo mkdir -p /usr/lib/systemd/system/kubelet.service.d
curl -sSL "https://raw.githubusercontent.com/kubernetes/release/${RELEASE_VERSION}/cmd/krel/templates/latest/kubeadm/10-kubeadm.conf" | sed "s:/usr/bin:${DOWNLOAD_DIR}:g" | sudo tee /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf

Now we need to install kubectl

```
Set up repository:

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
EOF

sudo yum install -y kubectl
```

Name: Kshitij Hundre **Div:D15C** **Roll No:18**

```
ec2-user@ip-172-31-24-190 ~ $ kubectl version
Client Version: v1.31.1
Kustomize Version: v5.4.2
```

We have installed successfully installed kubernetes

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
```

```
[root@ip-172-31-24-190 bin]# sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-iptables = 1
```

Disable SELINUX

Type **sudo nano /etc/selinux/config** and set the value of **SELINUX=disabled** instead of **SELINUX=permissive**
Save the file by pressing ctrl+o then press enter then press ctrl+x

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
# See also:
# https://docs.fedoraproject.org/en-US/quick-docs/getting-started-with-selinux/#getting-started-with-selinux-selinux-states-and-modes
#
# NOTE: In earlier Fedora kernel builds, SELINUX=disabled would also
# fully disable SELinux during boot. If you need a system with SELinux
# fully disabled instead of SELinux running with no policy loaded, you
# need to pass selinux=0 to the kernel command line. You can use grubby
# to persistently set the bootloader to boot with selinux=0:
#
#    grubby --update-kernel ALL --args selinux=0
#
# To revert back to SELinux enabled:
#
#    grubby --update-kernel ALL --remove-args selinux
#
SELINUX=disabled
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Then reboot the system using **sudo reboot**
After rebooting we need to make ssh connection with machine after it gets disconnected

Now if we type command **sestatus,** then it show disabled

```
ec2-user@ip-172-31-24-190 ~ $ sestatus
SELinux status:                 disabled
```

5. Initialize the Kubecluster

Install packages socat and iproute-tc and conntrack to avoid prelight errors
sudo dnf install socat iproute-tc conntrack-tools -y

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

Name: Kshitij Hundre **Div:D15C** **Roll No:18**

```
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 172.31.24.190:6443 --token xsbsq1.6ro11sawnvttbsvu \
        --discovery-token-ca-cert-hash sha256:10d2b67f4f4749b51854065a554c74e6a956e4782d9ab4bb79b8591648b3edef
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
```

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

sudo systemctl restart kubelet

Then, add a common networking plugin called flannel as mentioned in the code.
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```
ec2-user@ip-172-31-24-190 ~ $ 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
```

Now type **kubectl get nodes**

```
The connection to the server 172.31.24.190:6443 was refused - did you specify the right host or port?
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
The connection to the server 172.31.24.190:6443 was refused - did you specify the right host or port?
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
^[[AError from server (Forbidden): nodes is forbidden: User "kubernetes-admin" cannot list resource "nodes" in
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
NAME                            STATUS   ROLES           AGE   VERSION
ip-172-31-24-190.ec2.internal   Ready    control-plane   34m   v1.31.0
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
NAME                            STATUS   ROLES           AGE   VERSION
ip-172-31-24-190.ec2.internal   Ready    control-plane   34m   v1.31.0
ec2-user@ip-172-31-24-190 ~ $
```

Note: If any time of get error of connection refused just restart the kubelet service (sudo systemctl restart kubelet)

Now that the cluster is up and running, we can deploy our nginx server on this cluster. Apply this deployment file using this command to create a deployment

```
ec2-user@ip-172-31-24-190 ~ $ kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

Use ‘kubectl get pods’ to verify if the deployment was properly created and the pod is working correctly.

```
ec2-user@ip-172-31-24-190 ~ $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-mwd8p    0/1     Pending   0          7s
nginx-deployment-d556bf558-zc25s    0/1     Pending   0          7s
```

As we can see our pods are in pending state
On checking logs to we came to know the pods are in tainted state (using command **kubectl describe pod nginx-deployment-d556bf558-mwd8p**)

```
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  56s   default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption: 0/1 nodes are available:
```

To make pods untainted
Type kubectl get nodes to see name of node

```
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
NAME                              STATUS   ROLES           AGE   VERSION
ip-172-31-24-190.ec2.internal   Ready    control-plane   43m   v1.31.0
ec2-user@ip-172-31-24-190 ~ $
 3  1:ec2-user@ip-172-31-24-190:~*  2:~/Downloads-  3:~/Downloads#  4:~/Downloads
```

Copy the name of the node (ip-172-31-24-190.ec2.internal)

Then type command **kubectl taint nodes <NODE_NAME> - -all**

Name: Kshitij Hundre **Div:D15C** **Roll No:18**

In my case **kubectl taint nodes ip-172-31-24-190.ec2.internal node-role.kubernetes.io/control-plane-**

```
ec2-user@ip-172-31-24-190 ~ $ kubectl taint nodes ip-172-31-24-190.ec2.internal node-role.kubernetes.io/control-plane-
node/ip-172-31-24-190.ec2.internal untainted
```

After executing above command, check again status of pods if still pending then restart kubelet wait for 1-2 minutes and check again

```
ec2-user@ip-172-31-24-190 ~ $ kubectl get pods
NAME                                READY   STATUS    RESTARTS      AGE
nginx-deployment-d556bf558-mwd8p    1/1     Running   2 (73s ago)   12m
nginx-deployment-d556bf558-zc25s    1/1     Running   2 (73s ago)   12m
```

As we can see our pods are running

Lastly, port forward the deployment to your localhost so that you can view it.

kubectl port-forward <POD_NAME> 8080:80
In my case : **kubectl port-forward nginx-deployment-d556bf558-mwd8p 8080:80**

Note: if you are getting connection refused error then restart kubelet

```
ec2-user@ip-172-31-24-190 ~ $  kubectl port-forward  nginx-deployment-d556bf558-mwd8p 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

As port forwarding is active so we cannot type other commands.
Open new terminal window and make ssh connection to same machine OR we can open instance of same machine in new browser tab
And type command **curl --head http://127.0.0.1:8080**

Name: Kshitij Hundre **Div:D15C** **Roll No:18**

```
ec2-user@ip-172-31-24-190 ~ $ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sat, 14 Sep 2024 06:54:21 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 04 Dec 2018 14:44:49 GMT
Connection: keep-alive
ETag: "5c0692e1-264"
Accept-Ranges: bytes

ec2-user@ip-172-31-24-190 ~ $
 3  1:ec2-user@ip-172-31-24-190:~#-  2:ec2-user@ip-172-31-24-190:~*  3:~/Downloads
```

Response status 200 (OK) indicates that our nginx server is running successfully on kubernetes

Conclusion: We started by installing and setting up Docker and Kubernetes. Initially, there were issues with the Kubernetes API server, but restarting the kubelet service resolved them. The pods weren't running at first due to node tainting, which we corrected by untainting the nodes. After addressing all the errors, we successfully deployed the NGINX server pods, which are now accessible via the forwarded port. The NGINX server can be accessed either through different terminals or by running the port forwarding process in the background, achieved by appending `&` to the command.