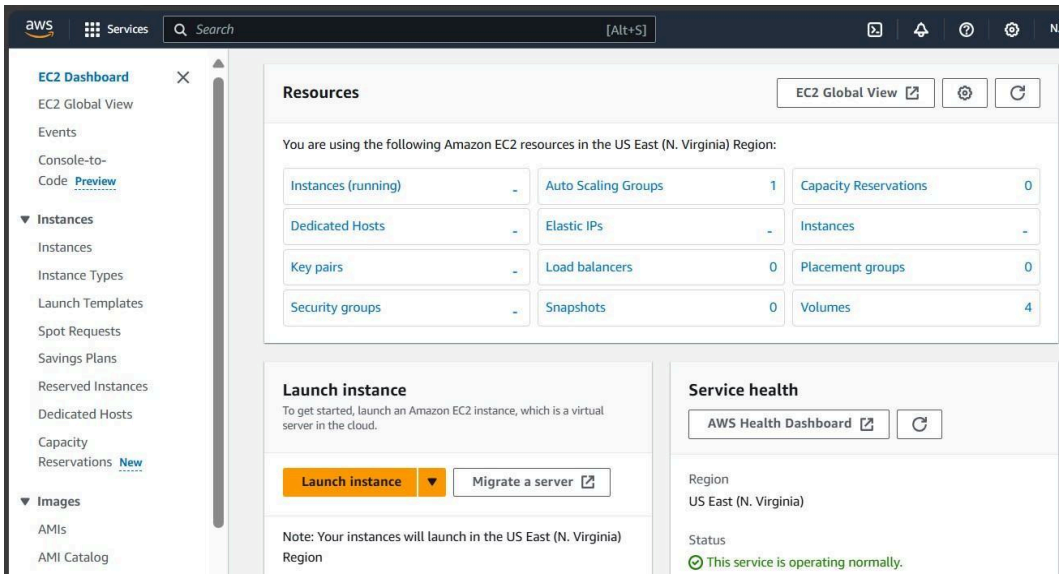
Experiment 4

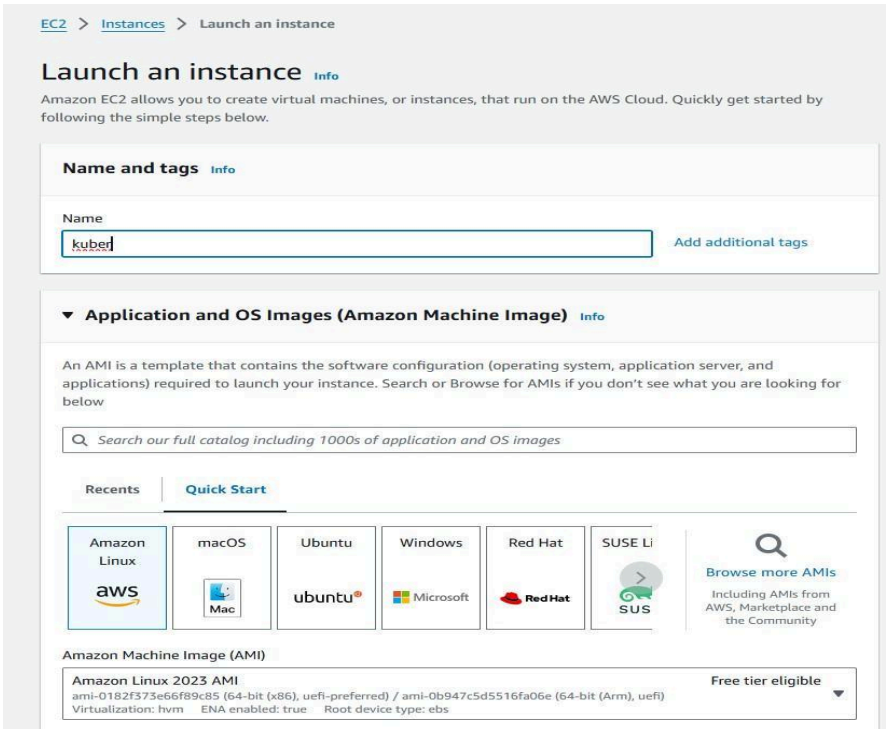
Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

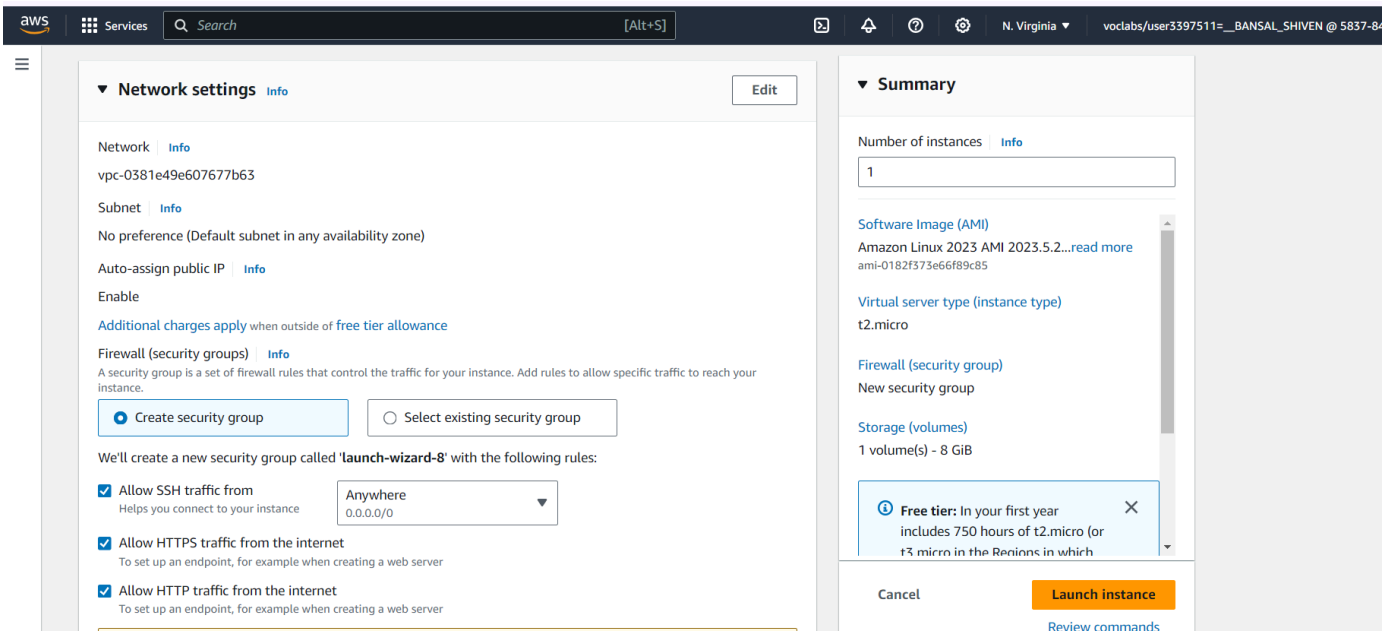
Procedure:

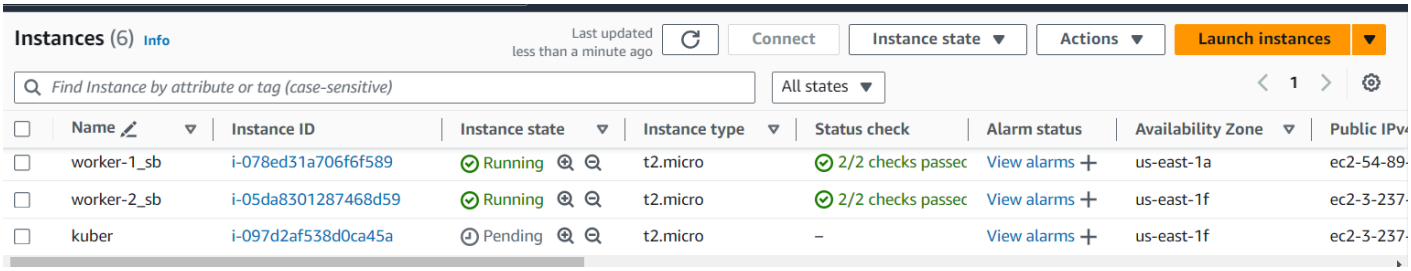
1. **Creation Of EC-2 instance**

- Create an EC2 AWS Linux instance on AWS .also edit the Security Group Inbound Rules to allow SSH. then select the t2.micro instance type

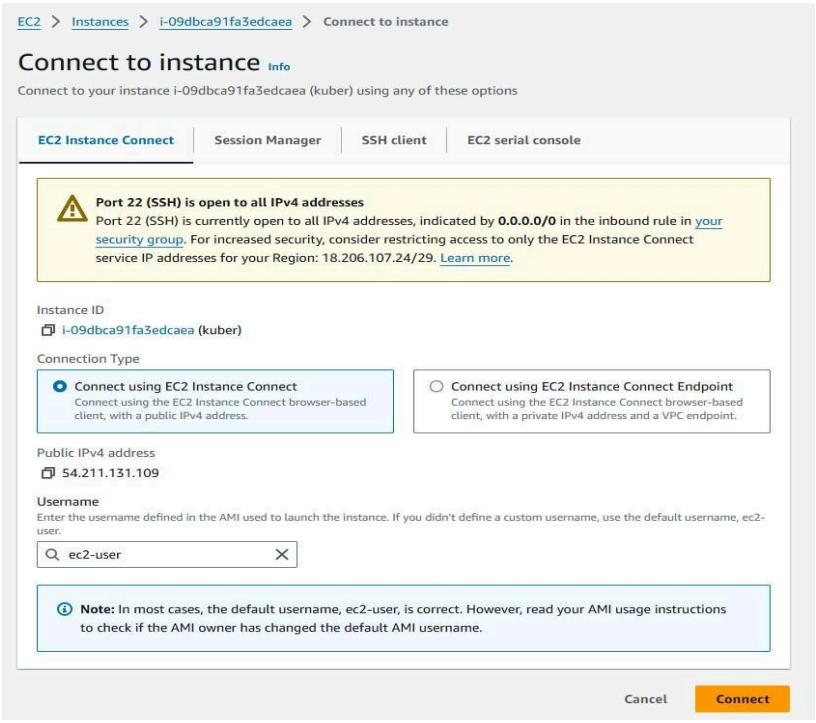








- Thus Kuber named -instance gets created.Then click on Id of that instance then click on connect button you will see this:



- Then go into SSH client where you will get this command
  Chmod 400 “keyname.pem”
  ssh -i <keyname>.pem ubuntu@<public_ip_address> copy it and then connect it and run the following command for establishing connection.(I have entered this command on git bash where i entered in downloads where server.pem is stored then as the key is not accessible hence we need to change its mode using chmod 400 “key name.pem”. Then use the given command for making connections).

```
ADMIN@DESKTOP-LPV2RP5 MINGW64 ~
$ cd Downloads/

ADMIN@DESKTOP-LPV2RP5 MINGW64 ~/Downloads
$ chmod 400 "server.pem"

ADMIN@DESKTOP-LPV2RP5 MINGW64 ~/Downloads
$ ssh -i "server.pem" ec2-user@ec2-3-237-37-35.compute-1.amazonaws.com
The authenticity of host 'ec2-3-237-37-35.compute-1.amazonaws.com (3.237.37.35)'
 can't be established.
ED25519 key fingerprint is SHA256:uhsLQC4WN+6i3np6iMFwW+vGF8aGkTrm89ryXQ4wT1g.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-237-37-35.compute-1.amazonaws.com' (ED25519) t
o the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sat Sep 14 11:47:26 2024 from 18.206.107.28
[ec2-user@ip-172-31-65-181 ~]$
```

2. Installation of Docker

1. . For installation of Docker into the machines run the following command: sudo yum install docker -y

```
[ec2-user@ip-172-31-65-181 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:08:49 ago on Sat Sep 14 11:42:25 2024.
Dependencies resolved.
================================================================================
 Package                  Arch     Version                  Repository     Size
================================================================================
Installing:
 docker                   x86_64   25.0.6-1.amzn2023.0.2    amazonlinux    44 M
Installing dependencies:
 containerd               x86_64   1.7.20-1.amzn2023.0.1    amazonlinux    35 M
 iptables-libs            x86_64   1.8.8-3.amzn2023.0.2     amazonlinux   401 k
 iptables-nft             x86_64   1.8.8-3.amzn2023.0.2     amazonlinux   183 k
 libcgroup                x86_64   3.0-1.amzn2023.0.1       amazonlinux    75 k
 libnetfilter_conntrack   x86_64   1.0.8-2.amzn2023.0.2     amazonlinux    58 k
 libnfnetlink             x86_64   1.0.1-19.amzn2023.0.2    amazonlinux    30 k
 libnftnl                 x86_64   1.2.2-2.amzn2023.0.2     amazonlinux    84 k
 pigz                     x86_64   2.5-1.amzn2023.0.3       amazonlinux    83 k
 runc                     x86_64   1.1.13-1.amzn2023.0.1    amazonlinux   3.2 M

Transaction Summary
================================================================================
Install  10 Packages

Total download size: 84 M
Installed size: 317 M
Downloading Packages:
(1/10): iptables-libs-1.8.8-3.amzn2023.0.2.x86_ 4.5 MB/s | 401 kB     00:00
(2/10): iptables-nft-1.8.8-3.amzn2023.0.2.x86_6 4.5 MB/s | 183 kB     00:00
(3/10): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm 3.9 MB/s |  75 kB     00:00
(4/10): libnetfilter_conntrack-1.0.8-2.amzn2023 2.6 MB/s |  58 kB     00:00
(5/10): libnfnetlink-1.0.1-19.amzn2023.0.2.x86_ 1.2 MB/s |  30 kB     00:00
(6/10): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rp 2.2 MB/s |  84 kB     00:00
(7/10): pigz-2.5-1.amzn2023.0.3.x86_64.rpm      2.8 MB/s |  83 kB     00:00
(8/10): runc-1.1.13-1.amzn2023.0.1.x86_64.rpm    30 MB/s | 3.2 MB     00:00
(9/10): containerd-1.7.20-1.amzn2023.0.1.x86_64  38 MB/s |  35 MB     00:00
(10/10): docker-25.0.6-1.amzn2023.0.2.x86_64.rp  34 MB/s |  44 MB     00:01
--------------------------------------------------------------------------------
Total                                            62 MB/s |  84 MB     00:01
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.

  Verifying        : containerd-1.7.20-1.amzn2023.0.1.x86_64
  Verifying        : docker-25.0.6-1.amzn2023.0.2.x86_64
  Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  Verifying        : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  Verifying        : libcgroup-3.0-1.amzn2023.0.1.x86_64
  Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64
  Verifying        : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  Verifying        : pigz-2.5-1.amzn2023.0.3.x86_64
  Verifying        : runc-1.1.13-1.amzn2023.0.1.x86_64

Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64         docker-25.0.6-1.amzn2023.0.2.x86_64                  iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  libcgroup-3.0-1.amzn2023.0.1.x86_64             libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64   libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64
  pigz-2.5-1.amzn2023.0.3.x86_64                  runc-1.1.13-1.amzn2023.0.1.x86_64

Complete!
```

- Then, configure cgroup in a daemon.json file by using following commands cd /etc/docker
  cat <<EOF | sudo tee /etc/docker/daemon.json
  {
  "exec-opts":
  ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
  "max-size": "100m"
  },
  "storage-driver": "overlay2"

}
EOF

```
[ec2-user@ip-172-31-65-181 ~]$ cd /etc/docker
[ec2-user@ip-172-31-65-181 docker]$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts":
["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
EOF
{
"exec-opts":
["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
```

- Then after this run the following command to enable and start docker and also to load the daemon.json file.
  sudo systemctl enable docker
  sudo systemctl daemon-reload
  sudo systemctl restart docker

```
[ec2-user@ip-172-31-81-216 docker]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /us
r/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-81-216 docker]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-81-216 docker]$ sudo systemctl restart docker
```

- docker -v

```
[ec2-user@ip-172-31-65-181 docker]$ docker -v
Docker version 25.0.5, build 5dc9bcc
```

3. Then Install Kubernetes with the following command.

- SELinux needs to be disable before configuring kubelet thus run the following command
  sudo setenforce 0
  sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

```
[ec2-user@ip-172-31-65-181 docker]$ sudo setenforce 0
[ec2-user@ip-172-31-65-181 docker]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=p
ermissive/' /etc/selinux/config
```

- Here We are adding kubernetes using the repository whose command is given below. cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
  [kubernetes]

name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF

```
[ec2-user@ip-172-31-65-181 docker]$ sudo setenforce 0
[ec2-user@ip-172-31-65-181 docker]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=p
ermissive/' /etc/selinux/config
[ec2-user@ip-172-31-65-181 docker]$  cat <<EOF | sudo tee /etc/yum.repos.d/kuber
netes.repo
[kubernetes]
name=Kubernetes baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/ enabled=1
gpgcheck=1 gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xm
l.key exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
[kubernetes]
name=Kubernetes baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/ enabled=1
gpgcheck=1 gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xm
l.key exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
[ec2-user@ip-172-31-65-181 docker]$
```

- After that Run following command to make the updation and also to install kubelet ,kubeadm, kubectl: sudo yum update

```
[ec2-user@ip-172-31-65-181 docker]$ sudo yum update
Invalid configuration value: gpgcheck=1 gpgkey=https://pkgs.k8s.io/core:/stable:
/v1.30/rpm/repodata/repomd.xml.key exclude=kubelet kubeadm kubectl cri-tools kub
ernetes-cni in /etc/yum.repos.d/kubernetes.repo; invalid boolean value '1 gpgkey
=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key exclude=kub
elet kubeadm kubectl cri-tools kubernetes-cni'
Error: Cannot find a valid baseurl for repo: kubernetes
Ignoring repositories: kubernetes
Last metadata expiration check: 0:19:14 ago on Sat Sep 14 11:42:25 2024.
Dependencies resolved.
Nothing to do.
Complete!
```

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

```
[ec2-user@ip-172-31-81-216 ~]$ sudo yum install -y kubelet kubeadm kubectl --dis
ableexcludes=kubernetes
Last metadata expiration check: 0:00:35 ago on Sat Sep 14 07:38:02 2024.
Dependencies resolved.
================================================================================
 Package                  Arch     Version                  Repository     Size
================================================================================
Installing:
 kubeadm                  x86_64   1.30.5-150500.1.1        kubernetes     10 M
 kubectl                  x86_64   1.30.5-150500.1.1        kubernetes     10 M
 kubelet                  x86_64   1.30.5-150500.1.1        kubernetes     17 M
Installing dependencies:
 conntrack-tools          x86_64   1.4.6-2.amzn2023.0.2     amazonlinux   208 k
 cri-tools                x86_64   1.30.1-150500.1.1        kubernetes    8.6 M
 kubernetes-cni           x86_64   1.4.0-150500.1.1         kubernetes    6.7 M
 libnetfilter_cthelper    x86_64   1.0.0-21.amzn2023.0.2    amazonlinux    24 k
 libnetfilter_cttimeout   x86_64   1.0.0-19.amzn2023.0.2    amazonlinux    24 k
 libnetfilter_queue       x86_64   1.0.5-2.amzn2023.0.2     amazonlinux    30 k

Transaction Summary
================================================================================
Install  9 Packages

Total download size: 53 M
Installed size: 292 M
Downloading Packages:
(1/9): libnetfilter_cthelper-1.0.0-21.amzn2023. 436 kB/s |  24 kB     00:00
(2/9): libnetfilter_queue-1.0.5-2.amzn2023.0.2. 1.5 MB/s |  30 kB     00:00
(3/9): libnetfilter_cttimeout-1.0.0-19.amzn2023 309 kB/s |  24 kB     00:00
(4/9): conntrack-tools-1.4.6-2.amzn2023.0.2.x86 2.3 MB/s | 208 kB     00:00
(5/9): cri-tools-1.30.1-150500.1.1.x86_64.rpm    34 MB/s | 8.6 MB     00:00
(6/9): kubectl-1.30.5-150500.1.1.x86_64.rpm      21 MB/s |  10 MB     00:00
(7/9): kubeadm-1.30.5-150500.1.1.x86_64.rpm      17 MB/s |  10 MB     00:00
(8/9): kubelet-1.30.5-150500.1.1.x86_64.rpm      32 MB/s |  17 MB     00:00
(9/9): kubernetes-cni-1.4.0-150500.1.1.x86_64.r  21 MB/s | 6.7 MB     00:00
--------------------------------------------------------------------------------
Verifying        : kubectl-1.30.5-150500.1.1.x86_64                      7/9
Verifying        : kubelet-1.30.5-150500.1.1.x86_64                      8/9
Verifying        : kubernetes-cni-1.4.0-150500.1.1.x86_64                9/9

Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
  cri-tools-1.30.1-150500.1.1.x86_64
  kubeadm-1.30.5-150500.1.1.x86_64
  kubectl-1.30.5-150500.1.1.x86_64
  kubelet-1.30.5-150500.1.1.x86_64
  kubernetes-cni-1.4.0-150500.1.1.x86_64
  libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
  libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
  libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
[ec2-user@ip-172-31-81-216 ~]$
```

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

```
[ec2-user@ip-172-31-81-216 ~]$ sudo swapoff -a
[ec2-user@ip-172-31-81-216 ~]$ echo "net.bridge.bridge-nf-call-iptables=1" | sud
o tee -a /etc/sysctl.conf
net.bridge.bridge-nf-call-iptables=1
[ec2-user@ip-172-31-81-216 ~]$ sudo sysctl -p
net.bridge.bridge-nf-call-iptables = 1
```

4. Initialize the Kubecluster

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

```
[ec2-user@ip-172-31-80-126 docker]$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
I0913 10:32:44.629146   26680 version.go:256] remote version is much newer: v1.31.0; falling back to: stable-1.30
[init] Using Kubernetes version: v1.30.4
[preflight] Running pre-flight checks
```

```
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 172.31.26.174:6443 --token pv0yyi.xhllqhclfjr50pt8 \
        --discovery-token-ca-cert-hash sha256:8293b2f6d29de466bd859007f5adbcdb3a
ecb0c446ba09033d32a5846b3d434f
```

- copy the token and save for future use .
  kubeadm join 172.31.26.174:6443 --token pv0yyi.xhllqhclfjr50pt8 \--discovery-token-ca-cert-hash sha256:8293b2f6d29de466bd859007f5adbcdb3aecb0c446ba09033d32a5846b3d434f
- 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

```
[ec2-user@ip-172-31-80-126 docker]$ ^C
[ec2-user@ip-172-31-80-126 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 flannel as mentioned in the code.

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/D
o cumentation/kube-flannel.yml
```

```
[ec2-user@ip-172-31-26-174 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
```

5. Now that the cluster is up and running,we can deploy our nginx server on this cluster. Apply deployment using this following command:

```
kubectl apply -f

https://k8s.io/examples/pods/simple-pod.yaml
```

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl apply -f https://k8s.io/examples/pods/s
imple-pod.yaml
pod/nginx created
```

Then use kubectl get nodes to check whether the pod gets created or not.

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   0/1     Pending   0          12s
```

To convert state from pending to running use following command:

kubectl describe pod nginx This command will help to describe the pods it gives reason for failure as it shows the untolerated taints which need to be untainted.

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           <none>
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  nginx:
    Image:        nginx:1.14.2
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-k4lj6 (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  kube-api-access-k4lj6:
    Type:                     Projected (a volume that contains injected data from m
ultiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 3
00s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for
 300s
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  7s    default-scheduler  0/1 nodes are available: 1 no
de(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption:
0/1 nodes are available: 1 Preemption is not helpful for scheduling.
```

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl taint nodes --all node-role.kubernetes.io
/control-plane-
node/ip-172-31-26-174.ec2.internal untainted
```

6. Now check pod status is is running

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl get pods
NAME    READY   STATUS    RESTARTS     AGE
nginx   1/1     Running   1 (6s ago)   90s
```

7. Lastly, mention the port you want to host. Here i have used localhost 8081 then check it.

```
kubectl port-forward nginx 8081:80
```

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl port-forward nginx 8081:80
Forwarding from 127.0.0.1:8081 -> 80
Forwarding from [::1]:8081 -> 80
```

8. Verify your deployment

Open up a new terminal and ssh to your EC2 instance.

Then, use this curl command to check if the Nginx server is running.

```
curl --head http://127.0.0.1:8080
```

If the response is 200 OK and you can see the Nginx server name, your deployment was successful. We have successfully deployed our Nginx server on our EC2 instance.

Conclusion:

First, I successfully launched an AWS EC2 instance running Amazon Linux. After that, I installed Docker and Kubernetes on the instance. Following the installation, I initialized the Kubernetes cluster, which provided me with a token, along with chown and mkdir commands. I then executed both the mkdir and chown commands successfully. Next, I installed the Flannel networking plugin without any issues. Initially, there was an error while deploying Nginx, but after correcting it, I successfully deployed Nginx using a simple-pod.yml file. I confirmed its deployment with the get pods command and hosted it locally on http://localhost:8081, which worked as expected.