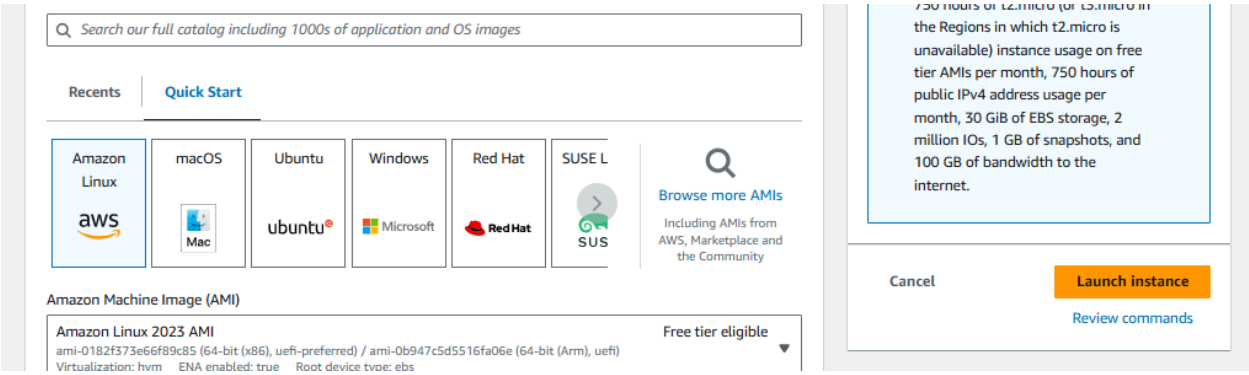
Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

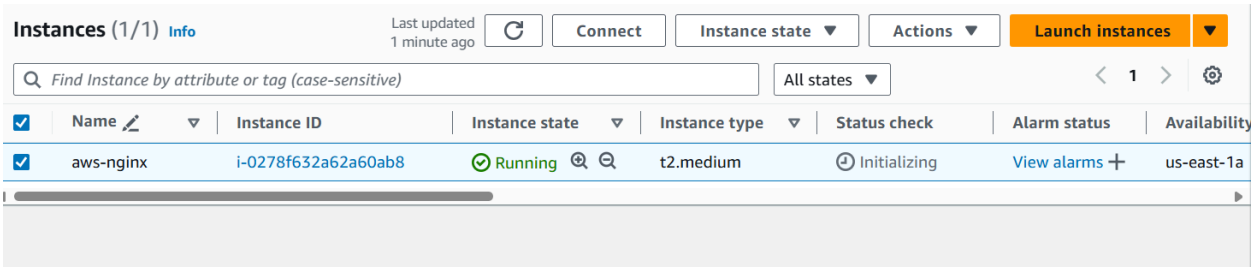
STEPS:

#Creating EC2 instance:

1. Create 1 EC-2 instances with all running on Amazon Linux as OS with inbound SSH allowed and the proper key

- Select Amazon linux as OS image
- To efficient run kubernetes cluster select instance type of at least t2.medium as kubernetes recommends at least 2 vCPU to run smoothly





2. **ssh connection checking through the local terminal with the help of key**

```
C:\Users\Avan>ssh -i "C:\Users\Avan\Downloads\kub1.pem" ec2-user@3.85.237.93
The authenticity of host '3.85.237.93 (3.85.237.93)' can't be established.
ED25519 key fingerprint is SHA256:Nz2iC26abFyhATf/8i4F0IgWmDoxTXBbzY9/NkMwYyM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.85.237.93' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Thu Sep 12 13:11:49 2024 from 18.206.107.29
```

3. Install Docker

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

```
[ec2-user@ip-172-31-16-177 ~]$ sudo dnf update
sudo dnf install docker
sudo systemctl enable docker
sudo systemctl start docker
Last metadata expiration check: 0:01:07 ago on Sat Sep 14 11:26:29 2024.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:01:08 ago on Sat Sep 14 11:26:29 2024.
Dependencies resolved.
================================================================================================================================================
 Package                                 Architecture                    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
```

```
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!
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
```

To test whether docker is successfully running, use command

- **sudo docker run hello-world**

```
[ec2-user@ip-172-31-16-177 ~]$ 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

```
[ec2-user@ip-172-31-16-177 ~]$ cd /etc/docker
[ec2-user@ip-172-31-16-177 docker]$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
[ec2-user@ip-172-31-16-177 docker]$ sudo systemctl daemon-reload
sudo systemctl restart docker
[ec2-user@ip-172-31-16-177 docker]$
```

4. **Install Kubernetes**
   SELinux needs to be disabled before configuring kubelet

- sudo setenforce 0
- sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

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

```
[ec2-user@ip-172-31-16-177 docker]$ sudo yum update
Kubernetes
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-16-177 docker]$ sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Last metadata expiration check: 0:00:12 ago on Sat Sep 14 11:53:54 2024.
Dependencies resolved.
================================================================================
 Package                    Architecture          Version
================================================================================
Installing:
 kubeadm                    x86_64                1.30.5-150500.1.1
 kubectl                    x86_64                1.30.5-150500.1.1
 kubelet                    x86_64                1.30.5-150500.1.1
Installing dependencies:
 conntrack-tools            x86_64                1.4.6-2.amzn2023.0.2
 cri-tools                  x86_64                1.30.1-150500.1.1
 kubernetes-cni             x86_64                1.4.0-150500.1.1
 libnetfilter_cthelper      x86_64                1.0.0-21.amzn2023.0.2
 libnetfilter_cttimeout     x86_64                1.0.0-19.amzn2023.0.2
 libnetfilter_queue         x86_64                1.0.5-2.amzn2023.0.2

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

Checking the version

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl version
Client Version: v1.30.5
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

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

```
[ec2-user@ip-172-31-16-177 docker]$ 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 the following command

- **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-st
#
# 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

```
[ec2-user@ip-172-31-16-177 docker]$ sudo reboot

Broadcast message from root@localhost on pts/1 (Sat 2024-09-14 11:58:50 UTC):

The system will reboot now!
```

Now if we type command

- **sestatus (***then it show disabled*)

```
[ec2-user@ip-172-31-16-177 docker]$ sestatus
SELinux status:                 disabled
[ec2-user@ip-172-31-16-177 docker]$
```

5. **Initialize the Kubecluster**

Install packages socat and iproute-tc and conntrack to avoid prelight errors

- sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all

```
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 172.31.16.177:6443 --token z7ersu.jg6wsvrvkywtavqd \
        --discovery-token-ca-cert-hash sha256:c3f37da241eae40a745ab6c8ab9f631c88778927d75ebeb8f37ce1a6f911c8cb
```

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

Restart the kubelet server

- 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-172-31-16-177 docker]$   mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[ec2-user@ip-172-31-16-177 docker]$ sudo systemctl restart kubelet
[ec2-user@ip-172-31-16-177 docker]$ kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
error: flag needs an argument: 'f' in -f
See 'kubectl apply --help' for usage.
-bash: https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml: No such file or directory
[ec2-user@ip-172-31-16-177 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
```

Now to see your nodes are running type this command

- **kubectl get nodes**

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl get nodes
NAME                              STATUS   ROLES           AGE     VERSION
ip-172-31-16-177.ec2.internal   Ready    control-plane   6m23s   v1.30.5
```

(***If you face the connectivity is not established/ refused try this command for 2-3 times and wait for a while the nodes will be set to status ready and if not happening then restart the kubelet server again)***

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-172-31-16-177 docker]$ 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-172-31-16-177 docker]$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-77d8468669-9kkmf   0/1     Pending   0          29s
nginx-deployment-77d8468669-zcdsc   0/1     Pending   0          29s
[ec2-user@ip-172-31-16-177 docker]$
```

As we can see our pods are in pending state we need to make pods untainted
Type kubectl get nodes to see name of node

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl get nodes
NAME                              STATUS   ROLES           AGE     VERSION
ip-172-31-16-177.ec2.internal   Ready    control-plane   6m23s   v1.30.5
```

Copy the name of the node (ip-172-31-16-177.ec2.internal)
Then type command

- **kubectl taint nodes ip-172-31-16-177.ec2.internal node-role.kubernetes.io/control-plane-**

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl taint nodes ip-172-31-16-177.ec2.internal node-role.kubernetes.io/control-plane-
node/ip-172-31-16-177.ec2.internal untainted
```

After executing above command, check again status of pods if still pending then restart kubelet wait for sometime and let the server load everything and check by running the pods command

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS        AGE
nginx-deployment-77d8468669-9kkmf   1/1     Running   3 (3m5s ago)    14m
nginx-deployment-77d8468669-zcdsc   1/1     Running   3 (50s ago)     14m
[ec2-user@ip-172-31-16-177 docker]$
```

As we can see our pods are running and status is set

6. **Deployment of Nginx Server**

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

- kubectl port-forward <POD_NAME> 8080:80

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

```
[ec2-user@ip-172-31-16-177 docker]$ kubectl port-forward nginx-deployment-77d8468669-9kkmf 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**

```
[ec2-user@ip-172-31-16-177 ~]$ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sat, 14 Sep 2024 12:28:13 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
```

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

Conclusion:

The deployment was completed successfully, but I ran into connection issues shortly after. Even though I tried troubleshooting through the terminal, the problem persisted, forcing me to restart the server several times before the issue was resolved. The pods created were not running because nodes were tainted so we had to make them untainted. During this process, I gained valuable insights into deployment installation, understanding both the setup and the challenges that can arise with Kubernetes. It was a bit frustrating at times, but the experience taught me how critical it is to troubleshoot effectively and manage the deployment environment. Ultimately, and the deployment was fully functional.