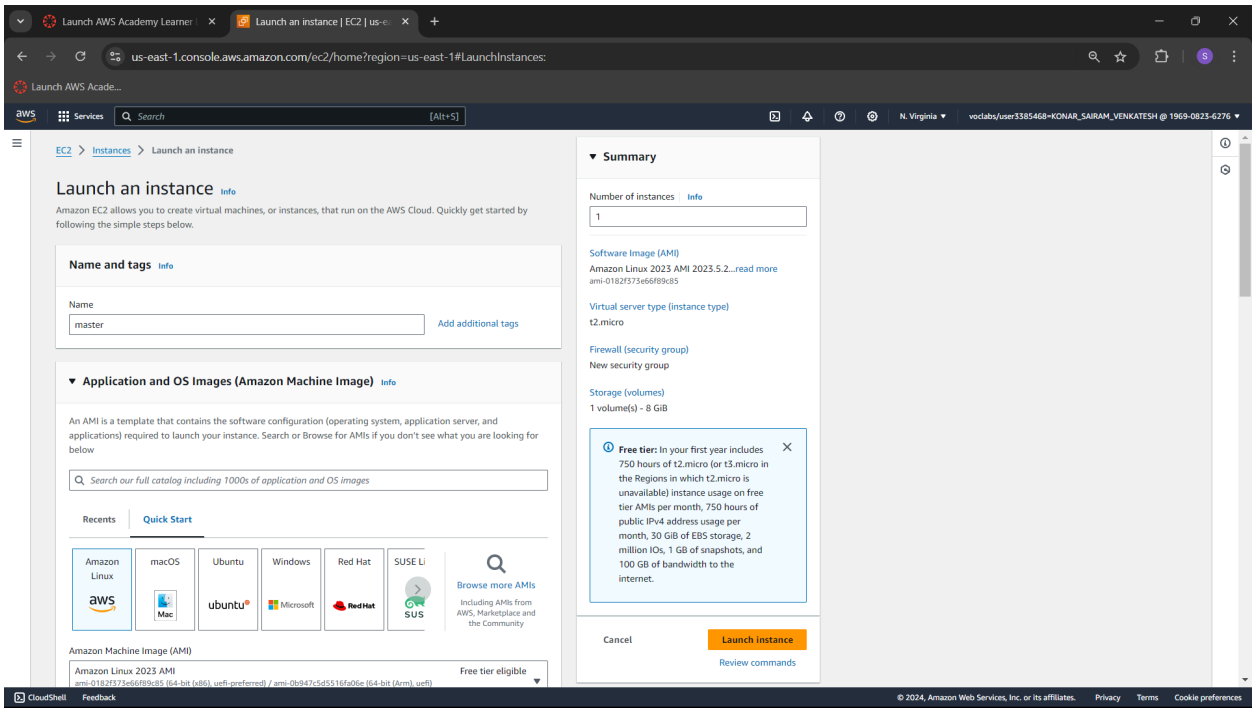
Sairam Konar D15C 27 2024-25

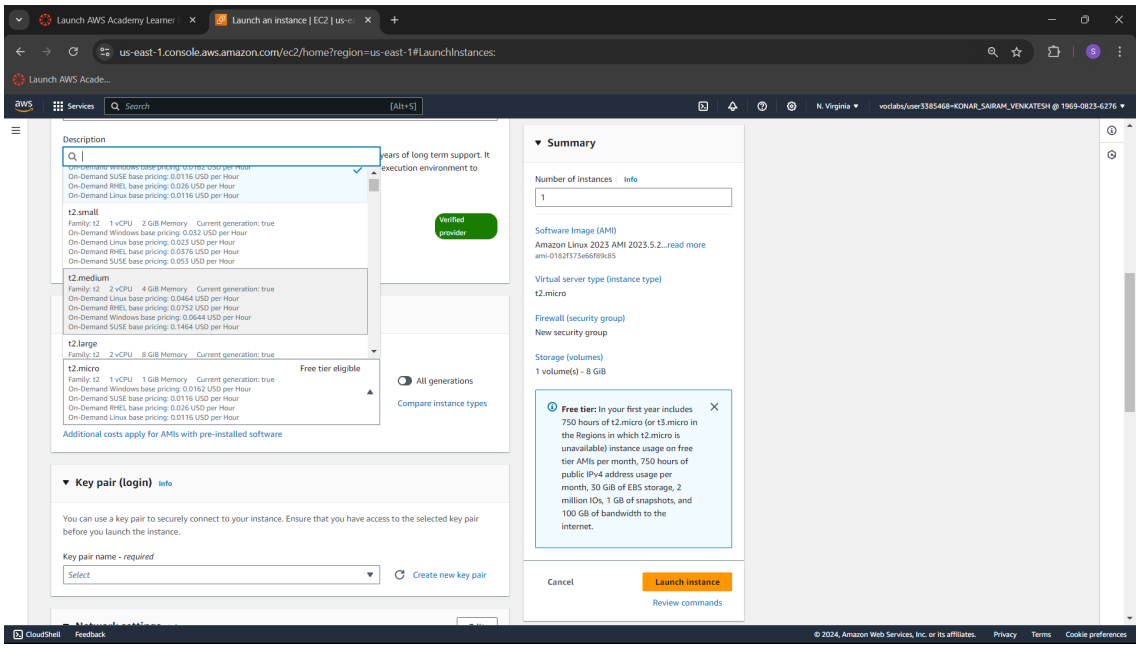
Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Step 1: Set Up EC2 Instances.

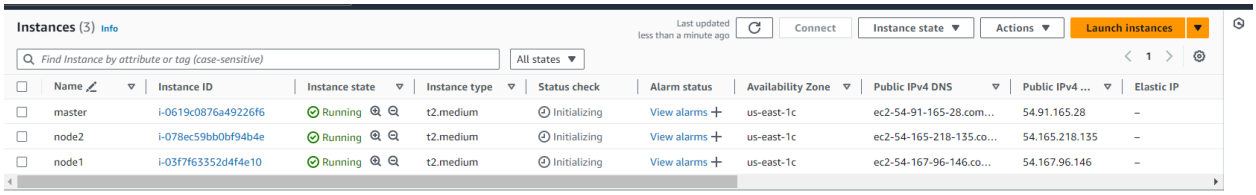
1) Set up 3 EC2 instances called master, node1, node2
   Select Amazon Linux as the OS image.



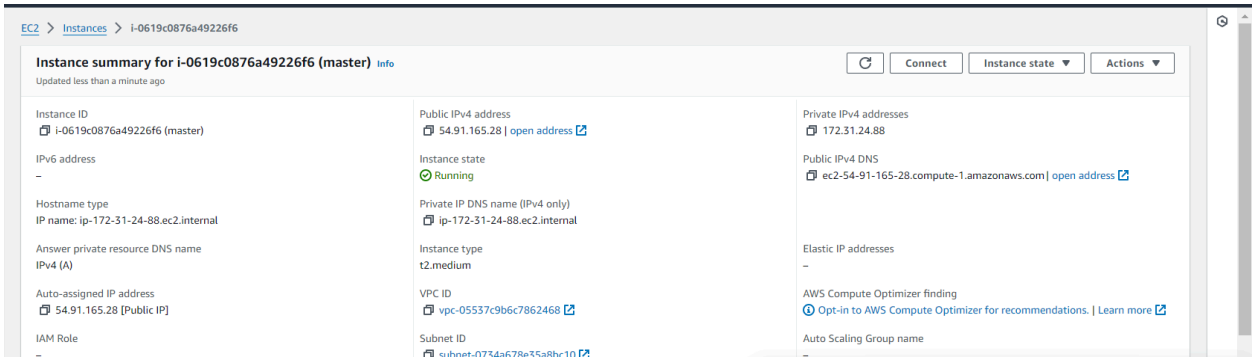
IMPORTANT: The default instance type and free one provided by AWS is t2.micro, which provides only 1CPU and 1 GiB of memory. For running Kubernetes, a minimum of 2 CPUs and 2GiB of RAM is required, hence change **t2.micro** to **t2.medium.**



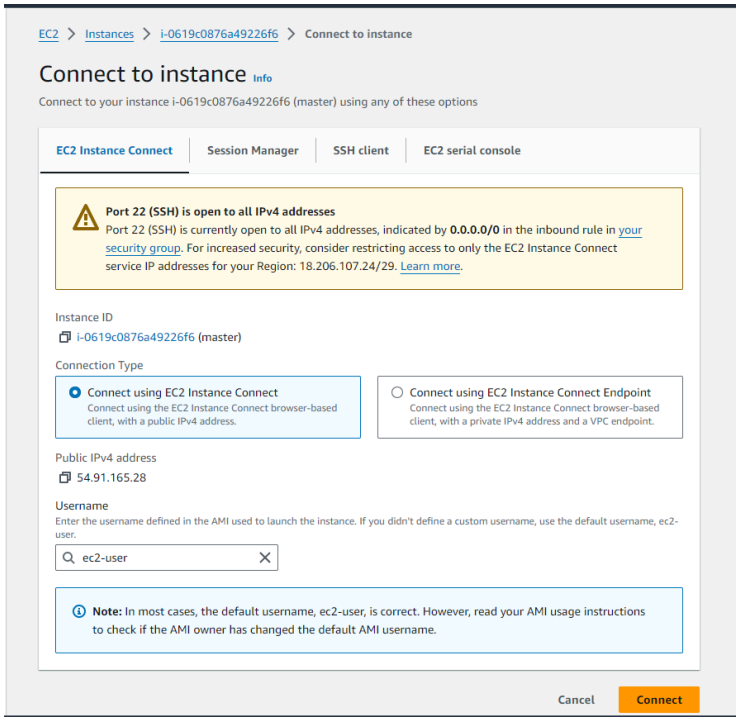
2) Select a key pair, it may be the default (vockey provided by AWS Academy) or you may create one.
These are the 3 instances created. Click on the instance ID of each.



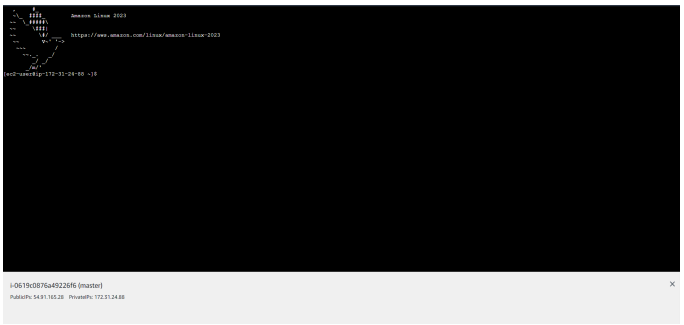
3) Click on connect.

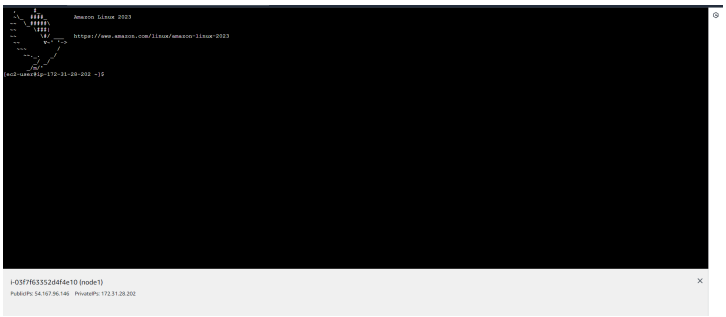


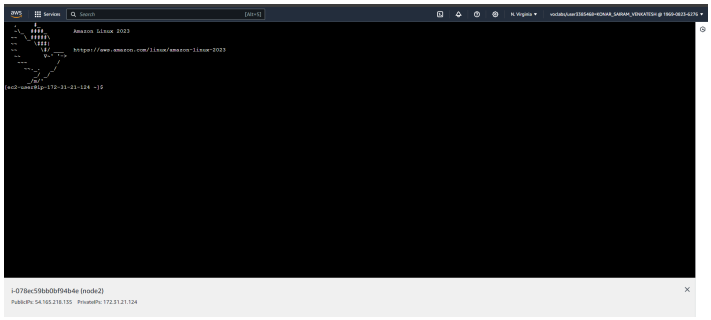
4) Maintain the default options and click on connect.



5) After doing the above steps, the terminal for all 3 nodes can be seen.







PERFORM THE FOLLOWING STEPS ON ALL 3 MACHINES

Step 2: Installation of Docker

1) Use command

‘sudo su’

This allows you to act as the root user of the terminal

```
[ec2-user@ip-172-31-21-124 ~]$ sudo su
[root@ip-172-31-21-124 ec2-user]#
```

2) We can install docker using YUM(Yellowdog Updater, Modified). Use the command

‘yum install docker -y’

```
[root@ip-172-31-21-124 ec2-user]# yum install docker -y
Last metadata expiration check: 0:05:01 ago on Fri Sep 13 16:47:44 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             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_64.rpm                    4.0 MB
/s | 401 kB     00:00
(2/10): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm                     3.1 MB
/s | 183 kB     00:00
(3/10): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm                          2.1 MB
/s |  75 kB     00:00
(4/10): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm           2.9 MB
/s |  58 kB     00:00
(5/10): libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64.rpm                    685 kB
/s |  30 kB     00:00
(6/10): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rpm                         1.5 MB
/s |  84 kB     00:00
(7/10): pigz-2.5-1.amzn2023.0.3.x86_64.rpm                               2.4 MB
/s |  83 kB     00:00
(8/10): runc-1.1.13-1.amzn2023.0.1.x86_64.rpm                             23 MB
/s | 3.2 MB     00:00
(9/10): docker-25.0.6-1.amzn2023.0.2.x86_64.rpm                           35 MB
/s |  44 MB     00:01
(10/10): containerd-1.7.20-1.amzn2023.0.1.x86_64.rpm                      22 MB
/s |  35 MB     00:01
--------------------------------------------------------------------------------
Total                                                                     51 MB
/s |  84 MB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
```

```
Running transaction
  Preparing        :                                                        1/1
  Installing       : runc-1.1.13-1.amzn2023.0.1.x86_64                      1/10
  Installing       : containerd-1.7.20-1.amzn2023.0.1.x86_64                2/10
  Running scriptlet: containerd-1.7.20-1.amzn2023.0.1.x86_64                2/10
  Installing       : pigz-2.5-1.amzn2023.0.3.x86_64                         3/10
  Installing       : libnftnl-1.2.2-2.amzn2023.0.2.x86_64                   4/10
  Installing       : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64              5/10
  Installing       : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64     6/10
  Installing       : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64              7/10
  Installing       : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64               8/10
  Running scriptlet: iptables-nft-1.8.8-3.amzn2023.0.2.x86_64               8/10
  Installing       : libcgroup-3.0-1.amzn2023.0.1.x86_64                    9/10
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64                    10/10
  Installing       : docker-25.0.6-1.amzn2023.0.2.x86_64                    10/10
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64                    10/10
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                1/10
  Verifying        : docker-25.0.6-1.amzn2023.0.2.x86_64                    2/10
  Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64              3/10
  Verifying        : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64               4/10
  Verifying        : libcgroup-3.0-1.amzn2023.0.1.x86_64                    5/10
  Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64     6/10
  Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64              7/10
  Verifying        : libnftnl-1.2.2-2.amzn2023.0.2.x86_64                   8/10
  Verifying        : pigz-2.5-1.amzn2023.0.3.x86_64                         9/10
  Verifying        : runc-1.1.13-1.amzn2023.0.1.x86_64                      10/10

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   iptables-nft-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   libnftnl-1.2.2-2.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!
[root@ip-172-31-21-124 ec2-user]#
```

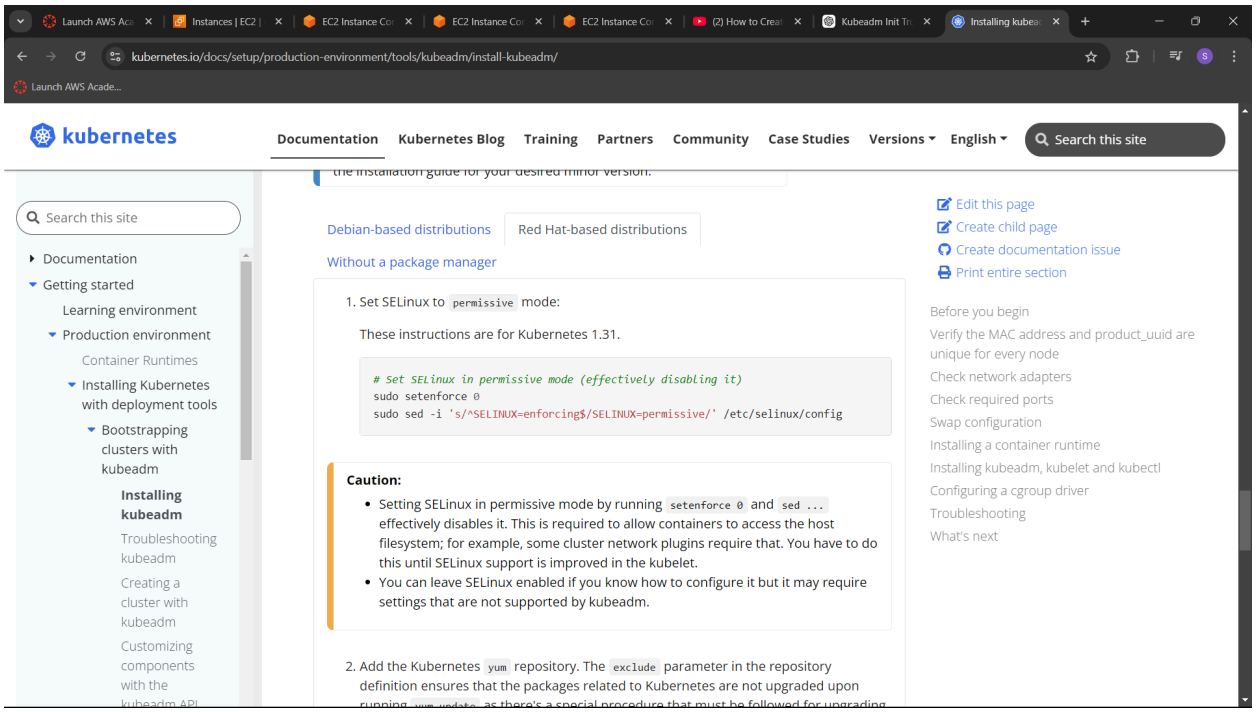
3) Now, configure a daemon.json file by using the following chain of commands.

- 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
- sudo systemctl enable docker
- sudo systemctl daemon-reload
- sudo systemctl restart docker

```
[root@ip-172-31-20-253 ec2-user]# 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
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-20-253 docker]#
```

Step 3: Installing Kubernetes

1) For installing kubernetes, we will be using kubeadm, a framework used for creating kubernetes clusters using command line.
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
The following will be visibe when you visit the website.



2) Select **red hat-based distributions** as amazon linux is based on red hat.

sudo setenforce 0
→ sets SELinux to permissive mode

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
→ edits the SELinux configuration file (/etc/selinux/config) to make the change persistent across reboots. If not used, SELinux reverts to enforcing mode after reboot.

Setting SELinux to permissive mode during Kubernetes installation prevents permission-related issues with container runtimes and components that may not function correctly under SELinux's enforcing policies.

Run the following commands:

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

- 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

This comamnd is a repository script to create a kubernetes repository

```
[root@ip-172-31-21-124 ec2-user]# 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
[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
```

- yum repolist

This command shows the repositories created on the machine.

```
[root@ip-172-31-21-124 ec2-user]# yum repolist
repo id                                   repo name
amazonlinux                               Amazon Linux 2023 repository
kernel-livepatch                          Amazon Linux 2023 Kernel Livepatch repository
kubernetes                                Kubernetes
```

Next step is to install kubelet, kubeadm, kubectl

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

```
[root@ip-172-31-21-124 ec2-user]# sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Kubernetes                                            [===                Kubernetes
                                                                          56 kB/s | 9.4 kB     00:
00
Dependencies resolved.
==========================================================================================
 Package                     Architecture      Version                     Repository
                  Size
==========================================================================================
Installing:
 kubeadm                     x86_64            1.31.1-150500.1.1           kubernetes
                  11 M
 kubectl                     x86_64            1.31.1-150500.1.1           kubernetes
                  11 M
 kubelet                     x86_64            1.31.1-150500.1.1           kubernetes
                  15 M
Installing dependencies:
 conntrack-tools             x86_64            1.4.6-2.amzn2023.0.2        amazonlinux
                 208 k
 cri-tools                   x86_64            1.31.1-150500.1.1           kubernetes
                 6.9 M
 kubernetes-cni              x86_64            1.5.1-150500.1.1            kubernetes
                 7.1 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
==========================================================================================
```

```
                                                                                                                                                9/9
  Running scriptlet: kubectl-1.31.1-150500.1.1.x86_64
                       9/9
  Verifying        : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
                       1/9
  Verifying        : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
                       2/9
  Verifying        : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
                       3/9
  Verifying        : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
                       4/9
  Verifying        : cri-tools-1.31.1-150500.1.1.x86_64
                       5/9
  Verifying        : kubeadm-1.31.1-150500.1.1.x86_64
                       6/9
  Verifying        : kubectl-1.31.1-150500.1.1.x86_64
                       7/9
  Verifying        : kubelet-1.31.1-150500.1.1.x86_64
                       8/9
  Verifying        : kubernetes-cni-1.5.1-150500.1.1.x86_64
                       9/9

Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64        cri-tools-1.31.1-150500.1.1.x86_64        kubeadm-1.31.1-150500.1.1.x86_64                    kubectl-1.31.1-150500.1.1.x86_
64
  kubelet-1.31.1-150500.1.1.x86_64                   kubernetes-cni-1.5.1-150500.1.1.x86_64    libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64  libnetfilter_cttimeout-1.0.0-1
9.amzn2023.0.2.x86_64
  libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
[root@ip-172-31-21-124 ec2-user]#
```

Now, we need to enable the kubelet service. Run the command

- sudo systemctl enable --now kubelet

```
[root@ip-172-31-21-124 ec2-user]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
```

PERFORM THE FOLLOWING ON ONLY THE MASTER MACHINE

1) Firstly, we need to initialize kubernetes. For this, run the command:
    - kubeadm init

```
[root@ip-172-31-21-124 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 connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0913 17:19:05.142402   28614 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm.It is rec
ommended 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-172-31-21-124.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and
IPs [10.96.0.1 172.31.21.124]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-21-124.ec2.internal localhost] and IPs [172.31.21.124 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-21-124.ec2.internal localhost] and IPs [172.31.21.124 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
```

```
[api-check] Waiting for a healthy API server. This can take up to 4m0s
[api-check] The API server is healthy after 5.002506688s
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node ip-172-31-21-124.ec2.internal as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-ext
ernal-load-balancers]
[mark-control-plane] Marking the node ip-172-31-21-124.ec2.internal as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: 76r1ql.qpqk9pmv8t9osgq4
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[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

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 172.31.21.124:6443 --token 76r1ql.qpqk9pmv8t9osgq4 \
        --discovery-token-ca-cert-hash sha256:a03f677cdc93d202c5f89e2370eb7231d212211bac6f816cddcb547b7f00a4f2
[root@ip-172-31-21-124 ec2-user]#
```

From the output, we will receive a command that is used to link the nodes to the master. Copy it and save it somewhere local.

```
kubeadm join 172.31.21.124:6443 --token 76r1ql.qpqk9pmv8t9osgq4 \
        --discovery-token-ca-cert-hash sha256:a03f677cdc93d202c5f89e2370eb7231d212211bac6f816cddcb547b7f00a4f2
```

2) From the output, we receive the following commands:

- mkdir -p $HOME/.kube
- sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
- sudo chown $ (id -u):$(id -g) $HOME/.kube/config

Run these commands.

```
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
```

```
[root@ip-172-31-21-124 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
```

3) To check whether nodes are connected, run the command
   - kubectl get nodes

   This output shows only master is connected right now.

```
[root@ip-172-31-20-253 docker]# kubectl get nodes
NAME                              STATUS     ROLES           AGE     VERSION
ip-172-31-20-253.ec2.internal   NotReady   control-plane   2m19s   v1.31.1
```

PERFORM THE FOLLOWING ONLY ON THE NODE MACHINES

Use the command that you had copied before and use them on the node machines.

```
[root@ip-172-31-23-39 docker]# kubeadm join 172.31.21.12:6443 --token 5mqobq.tz0ofaxdr0jhd384        --discovery-token-ca-cert-hash sha256:c7f71cc84c1dd813b0ea20de
72a188dab31a490d2aceea1c30d9744f5e1e7f59
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
error execution phase preflight: couldn't validate the identity of the API Server: failed to request the cluster-info ConfigMap: Get "https://172.31.21.12:6443/api/
v1/namespaces/kube-public/configmaps/cluster-info?timeout=10s": context deadline exceeded
To see the stack trace of this error execute with --v=5 or higher
[root@ip-172-31-23-39 docker]#
```

ERROR

Here, an error occurs as ‘Context Deadline Exceeded’. Hence, the final output is not obtained.

Conclusion:

In this experiment, we have learned how to create kubernetes clusters on a linux terminal, how the ssh command works on a local terminal and what requirements are necessary to create kubernetes clusters. In the final step, while working on the EC2 instance terminal, the connection of the master node to the slave nodes does not happen as it gives an error of ‘Context Deadline Exceeded’.