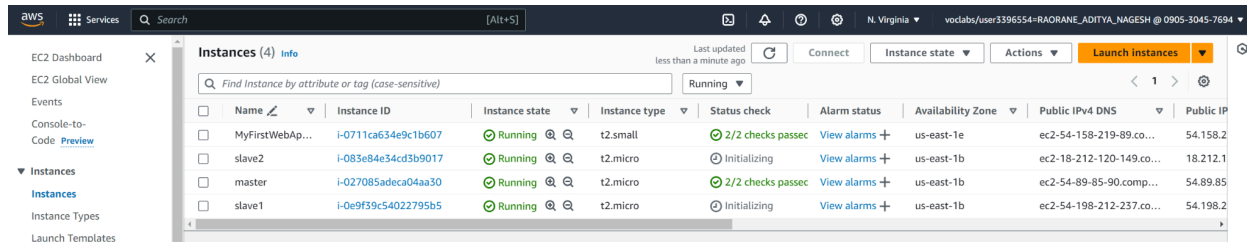
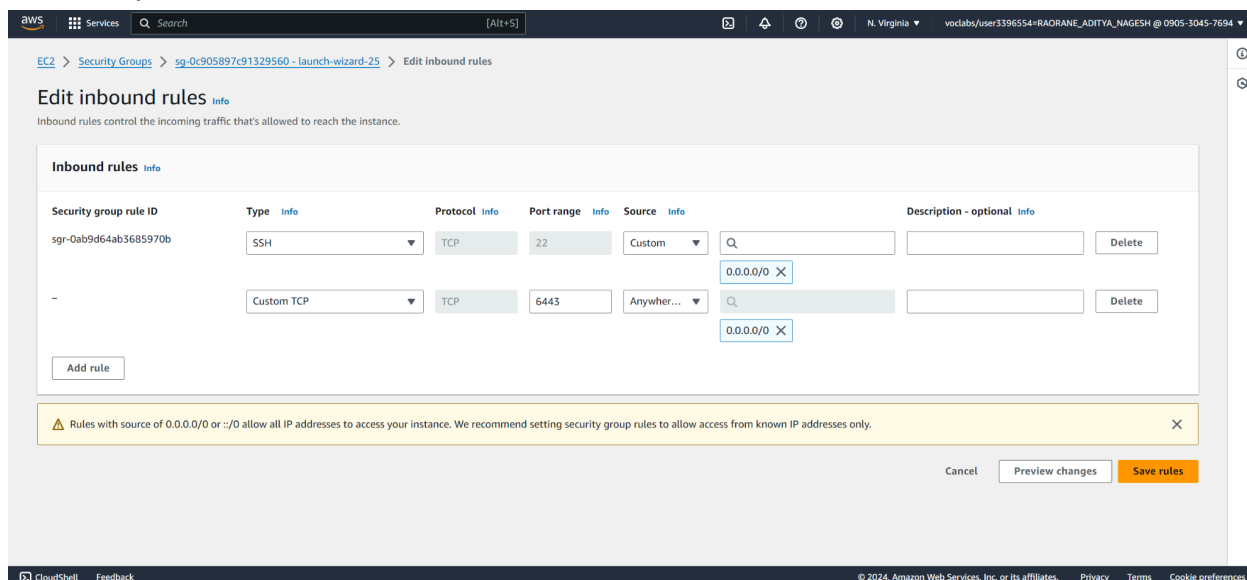


Aim: To understand the Kubernetes Cluster Architecture, install and spin up a Kubernetes Cluster on Linux Machines/Cloud.

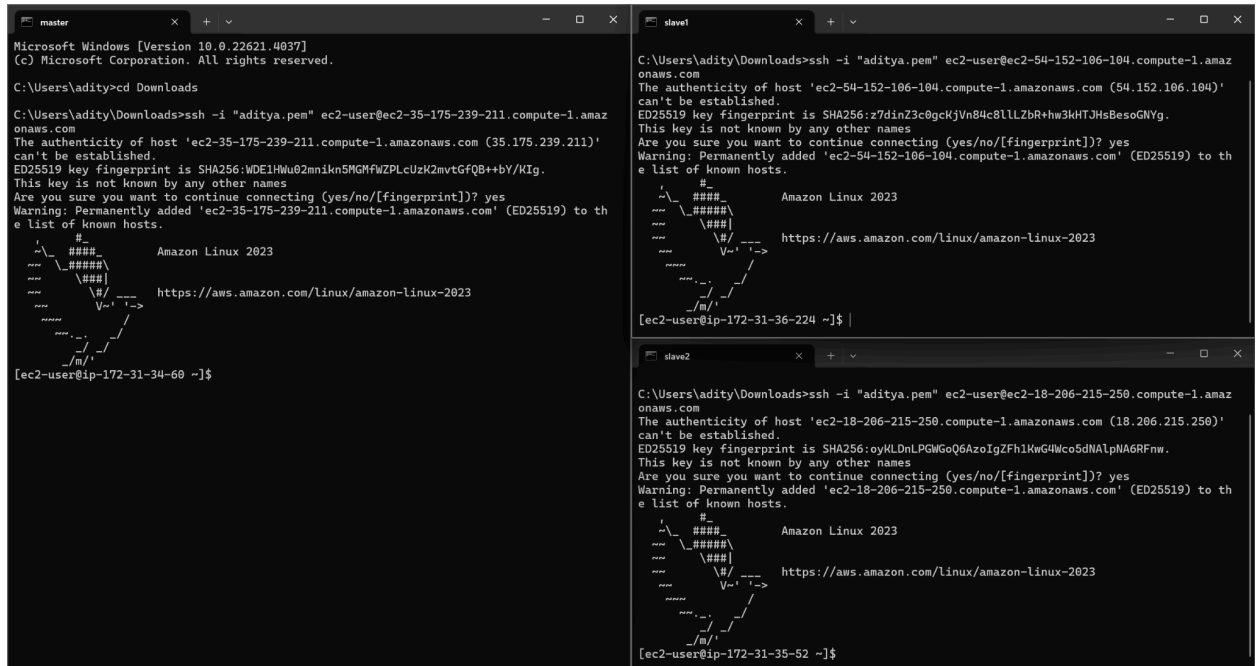
1] Create 3 EC2 instances with all running on Amazon Linux as OS with inbound SSH allowed. To efficiently run the machines it is advised to select the instance type as t2.medium which comes with 4 GiB of memory and 2 vCPU's.



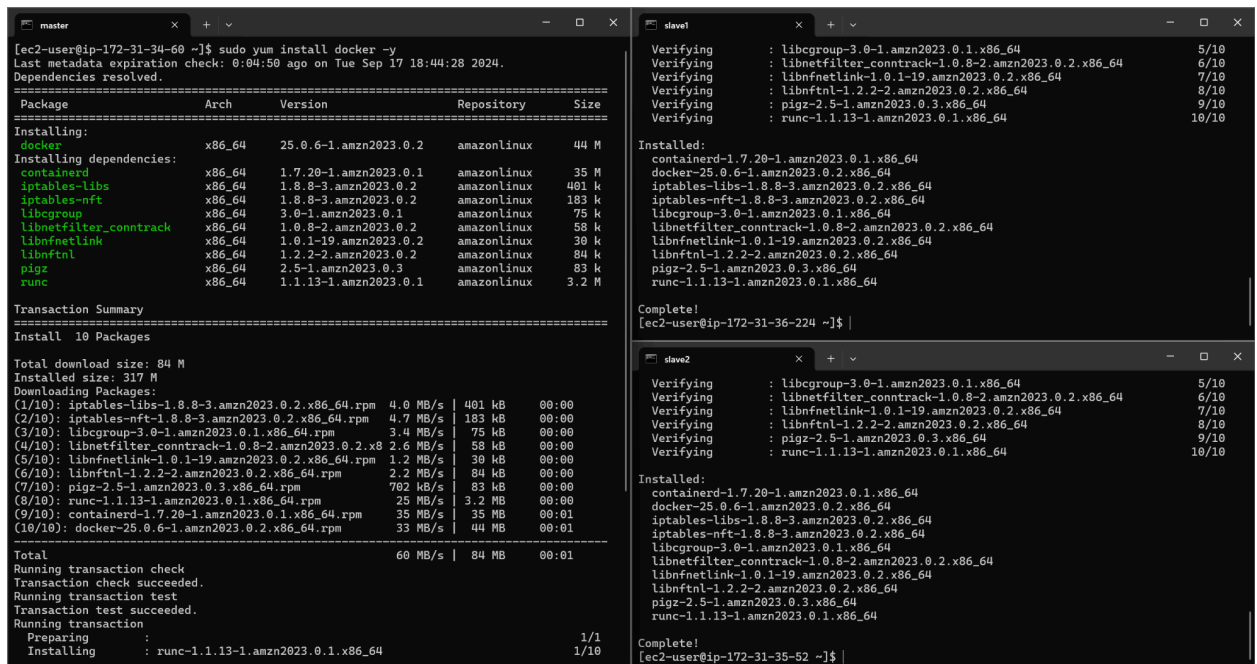
2] Update the inbound rules for the security groups on all three machines to allow TCP traffic on port 6443 from source 0.0.0.0/0.



3] SSH into all 3 machines each in a separate terminal.



4]To install Docker, use the following command:
sudo yum install docker -y



5] Then, configure cgroup in a daemon.json file by using the following commands. This allows kubernetes to manage host more efficiently

- `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
```

The image shows three terminal windows side-by-side, each representing a different node in a Kubernetes cluster: master, slave1, and slave2. Each window shows the execution of the same command to configure the Docker daemon.json file. The command is: `cat <<EOF | sudo tee /etc/docker/daemon.json`. The output of the command is a JSON configuration for the Docker daemon, which sets the cgroup driver to systemd, the log driver to json-file with a max-size of 100m, and the storage driver to overlay2. The master node's IP is 172-31-34-60, slave1's is 172-31-36-224, and slave2's is 172-31-35-52. All three nodes show the command being executed successfully, with the JSON configuration being written to the daemon.json file.

```
master
[ec2-user@ip-172-31-34-60 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-172-31-34-60 docker]$

slave1
Complete!
[ec2-user@ip-172-31-36-224 ~]$ cd /etc/docker
[ec2-user@ip-172-31-36-224 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-172-31-36-224 docker]$

slave2
Complete!
[ec2-user@ip-172-31-35-52 ~]$ cd /etc/docker
[ec2-user@ip-172-31-35-52 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-172-31-35-52 docker]$
```

6] After configuring restart docker service service :

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

The screenshot shows three terminal windows. The 'master' window on the left shows the configuration of `/etc/docker/daemon.json` with settings for `exec-opts`, `log-driver`, `log-opts`, `max-size`, and `storage-driver`. It then shows the execution of `sudo systemctl enable docker`, `sudo systemctl daemon-reload`, and `sudo systemctl restart docker`. The 'slave1' window on the top right shows the same configuration and startup steps. The 'slave2' window on the bottom right shows the same configuration and startup steps.

```

[ec2-user@ip-172-31-34-60 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-172-31-34-60 docker]$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/s
systemd/system/docker.service.
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-34-60 docker]$

[ec2-user@ip-172-31-36-224 docker]$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/s
systemd/system/docker.service.
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-36-224 docker]$

[ec2-user@ip-172-31-35-52 docker]$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/s
systemd/system/docker.service.
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-35-52 docker]$

```

7]SELinux needs to be disabled before configuring kubelet to avoid interference with kubernetes api server

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

Add kubernetes repository (paste in terminal)

```
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/r
```

```
epomd.xml.key
```

```
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
```

```
EOF
```

```

master
"storage-driver": "overlay2"
[ec2-user@ip-172-31-34-60 docker]$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
docker -v
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/
systemd/system/docker.service.
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-34-60 docker]$ sudo setenforce 0
[ec2-user@ip-172-31-34-60 docker]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive
/'
/etc/selinux/config
sudo: sed-i: command not found
-bash: /etc/selinux/config: Permission denied
[ec2-user@ip-172-31-34-60 docker]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.rep
o
[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
> ^C
[ec2-user@ip-172-31-34-60 docker]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.rep
o
[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-172-31-34-60 docker]$

slave1
/etc/selinux/config
sudo: sed-i: command not found
-bash: /etc/selinux/config: Permission denied
[ec2-user@ip-172-31-35-224 docker]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.rep
o
[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-172-31-35-224 docker]$

slave2
/etc/selinux/config
sudo: sed-i: command not found
-bash: /etc/selinux/config: Permission denied
[ec2-user@ip-172-31-35-52 docker]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.rep
o
[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-172-31-35-52 docker]$

```

8] Type following command to install set of kubernetes packages:

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

```

master
[ec2-user@ip-172-31-34-60 docker]$ sudo yum install -y kubelet kubeadm kubectl --disabl
eexcludes=kubernetes
Last metadata expiration check: 0:00:43 ago on Tue Sep 17 18:55:30 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:
contrack-tools x86_64    1.4.6-2.amzn2023.0.2 amazonlinux     200 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.amzn2023.0.2.x86_253 kB/s | 24 kB  00:00
(2/9): libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_225 kB/s | 24 kB  00:00
(3/9): libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64_1.3 MB/s | 30 kB  00:00
(4/9): contrack-tools-1.4.6-2.amzn2023.0.2.x86_64.rpm 1.6 MB/s | 208 kB  00:00
(5/9): cri-tools-1.30.1-150500.1.1.x86_64.rpm 33 MB/s | 8.6 MB  00:00
(6/9): kubelet-1.30.5-150500.1.1.x86_64.rpm 30 MB/s | 10 MB  00:00
(7/9): kubeadm-1.30.5-150500.1.1.x86_64.rpm 20 MB/s | 10 MB  00:00
(8/9): kubelet-1.30.5-150500.1.1.x86_64.rpm 38 MB/s | 17 MB  00:00
(9/9): kubernetes-cni-1.4.0-150500.1.1.x86_64.rpm 16 MB/s | 6.7 MB  00:00
-----
Total                    55 MB/s | 53 MB  00:00
Kubernetes              19 kB/s | 1.7 kB  00:00
Importing GPG key 0x9A296436:
Userid      "Isv:Kubernetes OBS Project <isv.kubernetes@build.opensuse.org>"
Fingerprint: D216 B104 86C0 377B 9E87 6E1A 2346 54DA 9A29 6436
From        : https://pkgs.k8s.io/core/stable/v1.30/rpm/repodata/repomd.xml.key
Key imported successfully
Running transaction check
Transaction check succeeded.

slave1
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-1.30.1-150500.1.1.x86_64 5/9
Verifying      : kubeadm-1.30.5-150500.1.1.x86_64 6/9
Verifying      : kubelet-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:
contrack-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-172-31-36-224 docker]$

slave2
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-1.30.1-150500.1.1.x86_64 5/9
Verifying      : kubeadm-1.30.5-150500.1.1.x86_64 6/9
Verifying      : kubelet-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:
contrack-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-172-31-35-52 docker]$

```

9] 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

```

master
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      : 1/1
  Installing     : kubernetes-cni-1.4.0-150500.1.1.x86_64 1/9
  Installing     : cri-tools-1.30.1-150500.1.1.x86_64 1/9
  Installing     : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64 2/9
  Installing     : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64 3/9
  Installing     : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64 4/9
  Installing     : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 5/9
  Running scriptlet: conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 6/9
  Installing     : kubelet-1.30.5-150500.1.1.x86_64 7/9
  Running scriptlet: kubelet-1.30.5-150500.1.1.x86_64 7/9
  Installing     : kubeadm-1.30.5-150500.1.1.x86_64 8/9
  Installing     : kubectrl-1.30.5-150500.1.1.x86_64 9/9
  Running scriptlet: kubectrl-1.30.5-150500.1.1.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_queue-1.0.5-2.amzn2023.0.2.x86_64 3/9
  Verifying     : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64 4/9
  Verifying     : cri-tools-1.30.1-150500.1.1.x86_64 5/9
  Verifying     : kubeadm-1.30.5-150500.1.1.x86_64 6/9
  Verifying     : kubectrl-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
kubectrl-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-172-31-34-60 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
[ec2-user@ip-172-31-34-60 docker]$

slave1
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
kubectrl-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-172-31-36-224 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
[ec2-user@ip-172-31-36-224 docker]$

slave2
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
kubectrl-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-172-31-35-52 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
[ec2-user@ip-172-31-35-52 docker]$

```

10] Perform this ONLY on the MASTER machine:-

a) Initialize Kubernetes By Typing Below Command:

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

```

master
[ec2-user@ip-172-31-34-60 docker]$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all
I0917 18:58:49.232613 27391 version.go:256] remote version is much newer: v1.31.0; falling back to: stable-1.30
[init] Using Kubernetes version: v1.30.5
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (949 MB) is less than the minimum 1700 MB
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[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 in beforehand using 'kubeadm config images pull'
W0917 18:58:49.564960 27391 checks.go:844] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm. It is recommended to use "registry.k8s.io/pause:3.9" 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-172-31-34-60.ec2.internal kubernet
es.kubernetes.default.kubernetes.default.svc.kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.34.60]
[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-172-31-34-60.ec2.internal localhost] and IPs [172.31.34.60 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-34-60.ec2.internal localhost] and IPs [172.31.34.60 127.0.0.1 ::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

```

b) Copy the mkdir and chown commands from 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`


```

master
[mark-control-plane] Marking the node ip-172-31-34-60.ec2.internal as control-plane by
adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-fr
om-external-load-balancers]
[mark-control-plane] Marking the node ip-172-31-34-60.ec2.internal as control-plane by
adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: f70o2w.perwilygbt7qu724
[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 automatical
ly approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node clie
nt 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 kube
let 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 172.31.34.60:6443 --token f70o2w.perwilygbt7qu724 \
--discovery-token-ca-cert-hash sha256:df9aa1a237965f949c055717ba438799f890a8647
b8cd10386fa7ed864e59096
[ec2-user@ip-172-31-34-60 docker]$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
[ec2-user@ip-172-31-34-60 docker]$

```

c) Copy the Kubernetes join command from your output to the clipboard

- `kubeadm join 172.31.34.60:6443 --token f70o2w.perwilygbt7qu724 \`
`--discovery-token-ca-cert-hash`
`sha256:df9aa1a237965f949c055717ba438799f890a8647b8cd10386fa7ed`
`864e59096`

d) Then, add a common networking plugin called flannel file as mentioned code.

- `kubectl apply -f`
<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>


```
[ec2-user@ip-172-31-34-60 docker]$ kubectl apply -f https://raw.githubusercontent.com/oreos/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
[ec2-user@ip-172-31-34-60 docker]$ |
```

e) Check the created node using this command

- kubectl get nodes

```
[ec2-user@ip-172-31-34-60 docker]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-172-31-34-60.ec2.internal       Ready    control-plane   5m41s   v1.30.5
```

11] Perform this ONLY on the WORKER machines:-

a) Paste the below command on all 2 worker machines

- sudo yum install iproute-tc socat-y (necessary packages required by kubernetes)
- sudo systemctl enable kubelet
- sudo systemctl restart kubelet

```

slave1
[ec2-user@ip-172-31-36-224 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
[ec2-user@ip-172-31-36-224 docker]$ sudo yum install -y iproute-tc socat # Install nec
essary packages required by Kubernetes
sudo systemctl enable kubelet      # Enable kubelet to start on boot
sudo systemctl restart kubelet     # Restart the kubelet service
Last metadata expiration check: 0:07:42 ago on Tue Sep 17 18:55:36 2024.
Dependencies resolved.
=====
Package                Architecture Version                      Repository                  Size
=====
Installing:
iproute-tc             x86_64          5.10.0-2.amzn2023.0.5      amazonlinux                 455 k
socat                  x86_64          1.7.4.2-1.amzn2023.0.2     amazonlinux                 303 k

Transaction Summary
=====
Install 2 Packages


slave2
-----
Total                               5.2 MB/s | 758 kB      00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :                                1/1
  Installing    : socat-1.7.4.2-1.amzn2023.0.2.x86_64 1/2
  Installing    : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64 2/2
  Running scriptlet: iproute-tc-5.10.0-2.amzn2023.0.5.x86_64 2/2
  Verifying     : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64 1/2
  Verifying     : socat-1.7.4.2-1.amzn2023.0.2.x86_64 2/2

Installed:
  iproute-tc-5.10.0-2.amzn2023.0.5.x86_64      socat-1.7.4.2-1.amzn2023.0.2.x86_64

Complete!
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/
systemd/system/kubelet.service.
[ec2-user@ip-172-31-35-52 docker]$

```

```
b) kubeadm join 172.31.34.60:6443 --token f70o2w.perwi1ygbt7qu724 \
    --discovery-token-ca-cert-hash
    sha256:df9aa1a237965f949c055717ba438799f890a8647b8cd10386fa7ed
    864e59096
```



```
slave1
[ec2-user@ip-172-31-36-224 docker]$ sudo kubeadm join 172.31.34.60:6443 --token f70o2w.perwi1ygbt7qu724 \
    --discovery-token-ca-cert-hash sha256:df9aa1a237965f949c055717ba438799f890a8647b8cd10386fa7ed864e59096 --v=5
I0917 19:04:45.857329 28291 join.go:417] [preflight] found NodeName empty; using OS hostname as NodeName
I0917 19:04:45.858112 28291 initconfiguration.go:122] detected and using CRI socket: unix:///var/run/containerd/containerd.sock
[preflight] Running pre-flight checks
I0917 19:04:45.858444 28291 preflight.go:93] [preflight] Running general checks
I0917 19:04:45.858592 28291 checks.go:278] validating the existence of file /etc/kubernetes/kubelet.conf
I0917 19:04:45.858911 28291 checks.go:278] validating the existence of file /etc/kubernetes/bootstrap-kubelet.conf
I0917 19:04:45.859018 28291 checks.go:102] validating the container runtime
I0917 19:04:45.910166 28291 checks.go:637] validating whether swap is enabled or not
I0917 19:04:45.910447 28291 checks.go:368] validating the presence of executable crictl
I0917 19:04:45.910556 28291 checks.go:368] validating the presence of executable conntrack
I0917 19:04:45.910709 28291 checks.go:368] validating the presence of executable ip

slave2
Node in the cluster with name "ip-172-31-35-52.ec2.internal" and status "Ready"
I0917 19:05:00.951572 28278 kubelet.go:175] [kubelet-start] Stopping the kubelet
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.00439379s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap
I0917 19:06:45.405069 28278 kubelet.go:242] [kubelet-start] preserving the cri socket information for the node
I0917 19:06:45.405225 28278 patchnode.go:31] [patchnode] Uploading the CRI Socket information "unix:///var/run/containerd/containerd.sock" to the Node API object "ip-172-31-35-52.ec2.internal" as an annotation
I0917 19:06:45.405575 28278 cert_rotation.go:137] Starting client certificate rotation controller

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

c) With the help of command the worker nodes are connected to the master node and are ready to do tasks assigned by the master node.

Now we can see in the master/control node of kubernetes that worker nodes are connected by typing **watch kubectl get nodes** in the **master** node instance.

```
[ec2-user@ip-172-31-34-60 docker]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-172-31-34-60.ec2.internal        Ready     control-plane  9m18s   v1.30.5
ip-172-31-35-52.ec2.internal        Ready     <none>      112s    v1.30.5
ip-172-31-36-224.ec2.internal        Ready     <none>      110s    v1.30.5
[ec2-user@ip-172-31-34-60 docker]$
```

Conclusion: We commenced the installation and configuration of the essential Kubernetes packages. While some were accessible via the default Linux repositories, others required the addition of external repositories for proper installation. During the setup, we encountered an issue where the nodes were tainted, resulting in the Kubernetes API server crashing. This was mitigated by removing the taints from the nodes. Additionally, SELinux was disabled to prevent potential conflicts with Kubernetes operations. In conclusion, we successfully integrated the worker nodes with the Kubernetes master node, achieving a fully operational cluster setup.