3-Node Kubernetes Cluster Deployment

Prerequisites:

- ✓ 3 virtual machines (VMs) are up and running in the same subnet / VPC. We are using EC2 instances on the AWS Cloud for this how-to guide. One of the VMs needs to have 2 vCPUs and 4 GB RAM, which means at least a t2.medium instance type. The rest of the VMs can be t2.micro instances.
- ✓ Ubuntu 22.04 LTS OS on all machines.
- ✓ Familiarity with the Kubernetes Components.
- ✓ On-Premises should have three VMs or Three Physical servers with communication enabled between them (Network)

Part A - Controller and Worker Nodes

Configure Network Prerequisites
This step configures IPv4 forwarding and lets iptables see bridged traffic. The overlay module is needed by Docker to work with the overlay network driver. The br_netfilter module is needed by Kubernetes to enable network filtering and NAT (Network Address Translation).
Refer to this Kubernetes documentation for more information.

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

sudo sysctl –system

lsmod | grep overlay
```

Confirm that you see an output similar to the following:

```
overlay                     151552   0
```

```
lsmod | grep br_netfilter
```

Confirm that you see an output similar to the following:

```
br_netfilter                 28672   0
bridge                      307200   1 br_netfilter
```

Verify that the `net.bridge.bridge-nf-call-iptables`, `net.bridge.bridge-nf-call-ip6tables`, and `net.ipv4.ip_forward` system variables are set to 1 in your sysctl config by running the following command:

```
sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-
call-ip6tables net.ipv4.ip_forward
```

Output

```
net.bridge.bridge-nf-call-iptables = 1
 net.bridge.bridge-nf-call-ip6tables = 1
 net.ipv4.ip_forward = 1
```

Ensure to run these steps to configure the required network settings on all nodes.

Configure Container Runtime

This step configures container runtime for all nodes on the cluster. Kubernetes 1.28 requires that you use a runtime that conforms with the Container Runtime Interface (CRI), which is an API for container run-times to work with `kubelet`. We will use `containerd` as the container runtime for this Kubernetes cluster.

Refer to [this Kubernetes documentation] for more information.

i. Install Docker Engine (as it includes `containerd` runtime).

   Refer to [this Docker documentation] for more information.

Update the `apt` package index, install the required packages, and get the official GPG keys for the Docker package repositories:

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

Add the Docker repository to the `apt` sources:

```
echo \
"deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Update the `apt` package index, again:

```
sudo apt-get update
```

Install Docker (including `containerd`):

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

Make daemon file for docker to aviod service errors later

```
    sudo mkdir /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
```

Confirm that Docker Engine installation is successful by running the `hello-world` image:

```
sudo docker run hello-world
```

Switch-off the swap in all the nodes

```
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Configure the `containerd` runtime environment.

Refer to this Kubernetes documentation for more information.

Login as root:

```
sudo -i
```

Create a default `containerd` configuration file:

```
containerd config default > /etc/containerd/config.toml
```

Open `config.toml` in a text editor:

```
vi /etc/containerd/config.toml
```

Change the value of `SystemdCgroup` from `false` to `true` (it should be visible around line number 125 in `config.toml`):

```
SystemdCgroup = true
```

Restart `containerd`

```
systemctl restart containerd
```

Exit the `sudo` mode:

```
exit
```

Ensure to run these steps to configure the container runtime on **all** nodes.

Install `kubeadm`, `kubelet`, and `kubectl`

In this step, we install these packages on **all** nodes:

- `kubeadm`: The command to bootstrap the cluster
- `kubelet`: An agent that runs on all nodes in the cluster and does things like starting pods and containers
- `kubectl`: The command line utility to talk to the cluster

Refer to this Kubernetes documentation for more information.

Since we are using Ubuntu 22.04 LTS, we will use instructions for Debian-based distributions.

i. Update the `apt` package index, install the required packages, and get the official GPG keys for the Kubernetes package repositories:

```
   sudo apt-get update
   sudo apt-get install -y apt-transport-https ca-certificates
   curl
   curl -fsSL
   https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo
   gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Add the Kubernetes repository to the `apt` sources:

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-
keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' |
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Update the `apt` package index, again:

```
sudo apt-get update
```

Install `kubelet`, `kubeadm`, and `kubectl`:

```
sudo apt-get install -y kubelet kubeadm kubectl
```

Pin their version (avoids automatic updates):

```
sudo apt-mark hold kubelet kubeadm kubectl
```

Ensure to install `kubeadm`, `kubelet`, and `kubectl` on **all** nodes.

Part B - Controller Node ONLY

Run these commands only on the VM designated as the controller (master) node.

(RUN AS ROOT) Initiate API server:

```
sudo -i
```

Initiate the API server. Remember to change the <ControllerVM-PrivateIP> with the actual private IP address of the controller VM:

```
kubeadm init --apiserver-advertise-address=<ControllerVM-PrivateIP> --pod-network-cidr=10.244.0.0/16
```

Exit the sudo mode

```
exit
```

(RUN AS NORMAL USER) Add a user for kube config:

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

(RUN AS NORMAL USER) Deploy Weave network:

```
kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
```

(RUN AS ROOT) Create cluster join command:

```
sudo -i
```

```
kubeadm token create --print-join-command
```

Part C - Worker Nodes ONLY

Copy the output of the cluster join command from the previous step and run on the VMs designated as the worker nodes.

(EXAMPLE COMMAND - DO NOT USE):

```
sudo kubeadm join 192.168.175.100:6443 --token jno9md.v9u1snltrwkv3vix
--discovery-token-ca-cert-hash
sha256:74b58d0e840e43a7051dcc4d7388b836dbd187c732fbfd3008051d10a14e271
a
```

The Kubernetes cluster is now configured.

Now check the status of the nodes and cluster (RUN as NORMAL User)

```
kubectl get nodes
```