Table of Content

⚠ **Attention**

In this method , I've used Containerd CRI and the Container Network Interface (CNI) is *Flannel*

Step 0 : Configure VMs

First , we should have 3 VMs : 1 for Master Node and 2 for Worker Node.
The OS is Ubuntu 22.04.3 Server LTS.
I use Bridge Network in Virtual Box with Static IP.

Part 1 : Disable swap

Kubernetes schedules work based on the understanding of available resources. If workloads start using swap, it can become difficult for Kubernetes to make accurate scheduling decisions. Therefore, it’s recommended to disable swap before installing Kubernetes.

You can do it with the following command. The `sudo swapoff -` command temporarily disables swap on your system. Then, the `sudo sed -i '/ swap / s/^/#/' /etc/fstab` command modifies a configuration file to keep the swap remains off even after a system reboot.

```
sudo swapoff -a

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

To check if the swap is off , we can execute this command :

```
free -h
```

If Swap has 0 value , it will be off correctly !

Part 2 : Change Host Name

First change the hostname like this :

```
# Master Node
sudo hostnamectl set-hostname master-node
```

```
# Worker 01
sudo hostnamectl set-hostname worker01
```

```
# Worker 02
sudo hostnamectl set-hostname worker02
```

Part 3 : Edit `netplan` config

Then, edit the `/etc/netplan/00-**.yaml` config .

Mater Node

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      addresses:
        - 192.168.1.180/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [10.202.10.202, 10.202.10.102]
  version: 2
```

Worker 01

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      addresses:
        - 192.168.1.181/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [10.202.10.202, 10.202.10.102]
  version: 2
```

Worker 02

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      addresses:
        - 192.168.1.182/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [10.202.10.202, 10.202.10.102]
  version: 2
```

Then we should apply this config .

```
sudo netplan try
sudo netplan apply
```

So the VMs IP is like this :

```
192.168.1.180 master-node
192.168.1.181 worker01
192.168.1.182 worker02
```

So in `master-node` we should edit `/etc/hosts` like this :

```
127.0.0.1 localhost
127.0.1.1 master-node

192.168.1.180 master-node
192.168.1.181 worker01
192.168.1.182 worker02
```

Step 1 : Set up the IPV4 bridge on all nodes

To configure the IPV4 bridge on all nodes, execute the following commands on each node.

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

```
sudo modprobe overlay
sudo modprobe br_netfilter
```

```
# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/kubernetes.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
```

```
# Apply sysctl params without reboot
sudo sysctl --system
```

Step 2 : Installing Containerd

First we should install `containerd` :

```
sudo apt install containerd -y
```

Step 3 : Installing Docker

Now we should install `docker.io` . So we have

```
sudo apt install docker.io -y
```

Step 4 : Install kubelet, kubeadm, and kubectl

Let’s install kubelet, kubeadm, and kubectl on each node to create a Kubernetes cluster. They play an important role in managing a Kubernetes cluster.

Kubelet is the node agent that runs on every node and is responsible for ensuring containers are running in a Pod as specified by the Pod's specifications. (Pods are the smallest deployable units in a Kubernetes cluster).

Then we need to install kubeadm, which is used to bootstrap a Kubernetes cluster, including setting up the master node and helping worker nodes join the cluster.

Kubectl is a CLI tool for Kubernetes to run commands to perform various actions such as deploying applications, inspecting resources, and managing cluster operations directly from the terminal.

Before installing them, you must update the package index with the `sudo apt-get update` command.

These instructions are for Kubernetes 1.29

Part 1 : Updating `apt` Packages

```
sudo apt-get update
# apt-transport-https may be a dummy package; if so, you can skip that package
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

Part 2 : Download the public signing key for the Kubernetes package repositories

```
# If the folder `/etc/apt/keyrings` does not exist, it should be created before the curl command, read
# sudo mkdir -p -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key \
| sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Part 3 : Add the appropriate Kubernetes `apt` repository.

```
# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \ https://pkgs.k8s.io/core:/stable:
```

Part 4 : Update the `apt` package index, install kubelet, kubeadm and kubectl, and pin their version

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Step 4 : Configuring `containerd.toml`

First we should create the containerd directory

```
sudo mkdir /etc/containerd
```

Then create a default configuration file for containerd and save it as `config.toml` using the following command:

```
sudo sh -c "containerd config default > /etc/containerd/config.toml"
```

After running these commands, you need to modify the config.toml file to locate the entry that sets "SystemdCgroup" to false and changes its value to true. This is important because Kubernetes requires all its components, and the container runtime uses systemd for cgroups.

```
sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml
```

Next, restart containerd and kubelet services to apply the changes you made on all nodes.

```
sudo systemctl daemon-reload
```

```
sudo systemctl restart containerd.service
```

```
sudo systemctl restart kubelet
```

⚠ Warning

All these Steps from 0 to 4 should be done on Master Nodes and Worker Nodes

Step 5 : Initialize the Kubernetes cluster on the master node

First we should pull the Kubernetes image using following command :

```
sudo kubeadm config images pull
```

Next, initialize our master node. The `--pod-network-cidr` flag is setting the IP address range for the pod network.

```
kubeadm init --cri-socket=unix:///var/run/containerd/containerd.sock \
--control-plane-endpoint "192.168.1.180:6443" \
--pod-network-cidr=10.244.0.0/16 --upload-certs
```

To manage the cluster, you should configure kubectl on the master node. Create the `.kube` directory in your home folder and copy the cluster's admin configuration to your personal `.kube` directory. Next, change the ownership of the copied configuration file to give the user the permission to use the configuration file to interact with the cluster.

```
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
```

Step 6 : Installing Flannel

Flannel is a simple and easy way to configure a layer 3 network fabric designed for Kubernetes. Flannel GitHub Repo

For Kubernetes v1.17+:

Deploying Flannel with `kubectl` :

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/\
latest/download/kube-flannel.yml
```

⚠ **Warning**

If you use custom `podCIDR` (not `10.244.0.0/16`) you first need to download the above manifest and modify the network to match your one.

Step 7 : Join Worker Node to Cluster

Apply the `kubeadm` **join** command on worker nodes to connect them to the master node. Prefix the command with `sudo`:

```
sudo kubeadm join [master-node-ip]:6443 --token [token] \
--discovery-token-ca-cert-hash sha256:[hash]
```

For example like this :

```
kubeadm join 192.168.1.180:6443 --token umcrji.6pxewrn6jqfdjbyb \
        --discovery-token-ca-cert-hash
    sha256:77582f0618a37e2fd1fefa4d64b679dc631269961c1f3f6fd2e749e0e3dec82f
```

Step 8 : Verify the cluster and test (Master Node)

Finally, we want to verify whether our cluster is successfully created.

```
kubectl get pods -n kube-system
kubectl get nodes
```

Step 9 : Deploy test application on cluster (Master Node)

```
kubectl run nginx --image=nginx
```