Ubuntu 20.04 LTS

1) Create 3 ubuntu VM, 1 control plane (master) and 2 worker nodes.

2) Ports enable for master node: (Inbound)
TCP 6443 → For Kubernetes API server
TCP 2379–2380 → For etcd server client API
TCP 10250 → For Kubelet API
TCP 10259 → For kube-scheduler
TCP 10257 → For kube-controller-manager
TCP 22 → For remote access with ssh
UDP 8472 → Cluster-Wide Network Comm. — Flannel VXLAN

3) Ports enabled for worker nodes (Inbound)
TCP 10250 → For Kubelet API
TCP 30000–32767 → NodePort Services†
TCP 22 → For remote access with ssh
UDP 8472 → Cluster-Wide Network Comm. — Flannel VXLAN

4) Login into the controller instance and change its hostname for our convenience

```
sudo hostnamectl set-hostname k8s-controller
```

Now logout and log back in to see the change.

Similarly, do this for both the worker nodes.

```
sudo hostnamectl set-hostname k8s-worker-1

sudo hostnamectl set-hostname k8s-worker-2
```

5) Now, we will declare the known hosts in all the three nodes so that the name that we give to each host could be mapped to their **private** IP. It is done so that we dont have to be confused and memorize the ip address.
We use private IP because incase of EC2, everytime the machine restarts the public IP changes.

```
sudo nano /etc/hosts
```

We need to enter the below line in each node having all the hosts as a key value pair (where key is private ip and value is the name given by us (any random name))
Example:

```
172.31.80.250 k8s-controller
172.31.82.32 k8s-worker-1
172.31.92.113 k8s-worker-2
```

Once we enter the hosts we need to logout and log back in.

6) Run the below commands on all the nodes (both controller and worker)

**# On all nodes, set up Docker Engine and containerd. You will need to load some kernel modules and modify some system settings as part of this process**

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

**# Set up the Docker Engine repository**

```
sudo apt-get update && sudo apt-get install -y ca-certificates curl gnupg lsb-release
apt-transport-https
```

**# Add Docker's official GPG key**

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

**# Set up the repository**

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

**# Update the apt package index**

```
sudo apt-get update
```

**# Install Docker Engine, containerd, and Docker Compose**

```
VERSION_STRING=5:23.0.1-1~ubuntu.20.04~focal
sudo apt-get install -y docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING
containerd.io docker-buildx-plugin docker-compose-plugin
```

**# Add your 'cloud_user' to the docker group**

```
sudo usermod -aG docker <USER>
```

Example:

```
sudo usermod -aG docker ubuntu
```

**# Log out and log back in so that your group membership is re-evaluated**

**# Make sure that 'disabled_plugins' is commented out in your config.toml file**

```
sudo sed -i 's/disabled_plugins/#disabled_plugins/' /etc/containerd/config.toml
```

**# Restart containerd**

```
sudo systemctl restart containerd
```

**# On all nodes, disable swap.**

```
sudo swapoff -a
```

**# On all nodes, install kubeadm, kubelet, and kubectl**

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

sudo apt-get update && sudo apt-get install -y kubelet=1.24.0-00 kubeadm=1.24.0-00
kubectl=1.24.0-00

sudo apt-mark hold kubelet kubeadm kubectl
```

7) Now go back to the controller plane (Master), and run the below command

**# On the control plane node only, initialize the cluster and set up kubectl access**

```
sudo kubeadm init --pod-network-cidr 192.168.0.0/16 --kubernetes-version 1.24.0
```

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

**# Verify the cluster is working**

```
kubectl get nodes
```

**# Install the Calico network add-on**

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

**# Get the join command (this command is also printed during kubeadm init . Feel free to simply copy it from there)**

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

**# The above command prints a command for the worker nodes to join the cluster. Paste the returned command and execute it in all the worker nodes.**

8) Copy the command returned from the last command in the controller and paste it into all the worker nodes

NOTE: you might need to use 'sudo' if it denies and gives error

9) Go back to controller node and check the nodes in the cluster

```
        kubectl get nodes
```

NOTE: It might take a couple of minutes to reflect and come in ready mode.

**Docs on installing Kubeadm:**
**https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/**

**To see more on creating a cluster using kubeadm:**
**https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluste**

ORIGINAL content:

# Building a Kubernetes Cluster
# Lesson URL:
https://learn.acloud.guru/course/introduction-to-kubernetes/learn/9c48bcf2-2573-485f-8906-6977bed23fc0/10592c01-d22c-4620-bace-28de4de4cf4b/watch

# Relevant Documentation

- Installing kubeadm:
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
- Creating a cluster with kubeadm:
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

# Lesson Reference

# If you are using cloud playground, create three servers with the following settings:

- Distribution: Ubuntu 20.04 Focal Fossa LTS
- Size: medium

# If you wish, you can set an appropriate hostname for each node.

# On the control plane node:

```
sudo hostnamectl set-hostname k8s-control
```

# On the first worker node:

```
sudo hostnamectl set-hostname k8s-worker1
```

# On the second worker node:

```
sudo hostnamectl set-hostname k8s-worker2
```

# On all nodes, set up the hosts file to enable all the nodes to reach each other using these hostnames

```
sudo vi /etc/hosts
```

# On all nodes, add the following at the end of the file. You will need to supply the actual private IP address for each node

```
<control plane node private IP> k8s-control
<worker node 1 private IP> k8s-worker1
<worker node 2 private IP> k8s-worker2
```

```
# Log out of all three servers and log back in to see these changes take effect

# On all nodes, set up Docker Engine and containerd. You will need to load some kernel
modules and modify some system settings as part of this
process

cat <<EOF | sudo tee /etc/modules-load.d/k8s.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/k8s.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

# Set up the Docker Engine repository

sudo apt-get update && sudo apt-get install -y ca-certificates curl gnupg lsb-release
apt-transport-https

# Add Docker's official GPG key

sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

# Set up the repository

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

# Update the apt package index

sudo apt-get update
```

```
# Install Docker Engine, containerd, and Docker Compose

VERSION_STRING=5:23.0.1-1~ubuntu.20.04~focal
sudo apt-get install -y docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING
containerd.io docker-buildx-plugin docker-compose-plugin

# Add your 'cloud_user' to the docker group

sudo usermod -aG docker $USER

# Log out and log back in so that your group membership is re-evaluated

# Make sure that 'disabled_plugins' is commented out in your config.toml file

sudo sed -i 's/disabled_plugins/#disabled_plugins/' /etc/containerd/config.toml

# Restart containerd

sudo systemctl restart containerd

# On all nodes, disable swap.

sudo swapoff -a

# On all nodes, install kubeadm, kubelet, and kubectl

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

sudo apt-get update && sudo apt-get install -y kubelet=1.24.0-00 kubeadm=1.24.0-00
kubectl=1.24.0-00

sudo apt-mark hold kubelet kubeadm kubectl

# On the control plane node only, initialize the cluster and set up kubectl access

sudo kubeadm init --pod-network-cidr 192.168.0.0/16 --kubernetes-version 1.24.0

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

# Verify the cluster is working
```

```
kubectl get nodes

# Install the Calico network add-on

kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

# Get the join command (this command is also printed during kubeadm init . Feel free to
simply copy it from there)

kubeadm token create --print-join-command

# Copy the join command from the control plane node. Run it on each worker node as root
(i.e. with sudo )

sudo kubeadm join ...

# On the control plane node, verify all nodes in your cluster are ready. Note that it may take
a few moments for all of the nodes to
enter the READY state

kubectl get nodes
```