**Create Master and Worker Nodes**

1. **Create** k8s-master-vm**:**

A screenshot of a computer

Description automatically generated

**Networking Tab:**

A screenshot of a computer

Description automatically generated

1. Create k8s-master-vm using same steps.

**Creating a cluster with kubeadm**

Using **kubeadm**, we can create a minimum viable Kubernetes cluster. We can use kubeadm to set up a cluster that will pass the Kubernetes Conformance tests. kubeadm also supports other cluster lifecycle functions, such as bootstrap tokens and cluster upgrades.

The **kubeadm** tool is good if we need:

* A simple way for us to try setting up Kubernetes cluster on our own.
* A building block to explore Kubernetes.
* A way for existing users to automate setting up a cluster and test their application.

**Pre-requisites:**

* One or more machines running a docker installable Linux OS like Ubuntu.
* 2 GiB or more of RAM per machine
* At least 2 CPUs on the machine that we use as a control-plane node.
* Full network connectivity among all machines in the cluster. We can use either a public or a private network.

**Objectives**

* Install a single control-plane Kubernetes cluster
* Install a Pod network on the cluster so that your Pods can talk to each other

**Steps to set up the cluster**

1. Set an appropriate hostname for each node in case we want to change the host names

On Control plane:

sudo hostnamectl set-hostname k8s-master-vm

On Worker plane:

sudo hostnamectl set-hostname k8s-worker-vm

All subsequent commands must be executed on both Control Plane and Worker Plane nodes

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

sudo nano /etc/hosts

and enter as follows:

10.0.0.4 k8s-master-vm

10.0.0.5 k8s-worker-vm

Save and Exit

1. If host names are changed, log out of both the servers and log back in to see these changes take effect

**Setup Docker Engine and containerd**

1. We 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

**To check if the modules are loaded**

sudo modprobe overlay

sudo modprobe br\_netfilter

1. Set sysctl params required by setup (these 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

1. Apply sysctl params without reboot

sudo sysctl --system

1. Set up the Docker Engine repository

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

1. 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

sudo chmod a+r /etc/apt/keyrings/docker.gpg

1. Set up the repository for docker

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

1. Update the apt package index

sudo apt-get update

1. Install Docker Engine, containerd, and Docker Compose

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

1. Add the current logged in user to the docker group

sudo usermod -aG docker $USER

1. Run the command to reevaluate the group membership

newgrp docker

1. Make sure that 'disabled\_plugins' is commented out in our config.toml file

sudo sed -i 's/disabled\_plugins/#disabled\_plugins/' /etc/containerd/config.toml

1. Restart containerd

sudo systemctl restart containerd

1. Disable swap on all the nodes

sudo swapoff -a

sudo sed -i '/swap/d' /etc/fstab

1. We can use ‘sed‘ to swap in true

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

1. Verify the change was made, The following command shouldn’t return anything.

grep 'SystemdCgroup = true' /etc/containerd/config.toml

1. On all the nodes, install kubeadm, kubelet and kubectl

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

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

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

1. Execute the following only on the control plane node, initialize the cluster and set up kubectl access

sudo kubeadm init --pod-network-cidr 192.168.0.0/16 --kubernetes-version 1.29.1

1. Copy the o/p of the above command into Notepad
2. Copy the cluster config file to $HOME/.kube folder

mkdir -p $HOME/.kube

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

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

1. Verify the cluster is working

kubectl get nodes

1. Install the Calico network add-on

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

1. Get the join command (this command is also printed during kubeadm init . We can copy it during that step as well)

kubeadm token create --print-join-command

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

sudo kubeadm join ...

1. On the control plane node, verify all nodes in the cluster are ready. (It may take a few minutes for all of the nodes to enter the READY state)

kubectl get nodes

**Note:**

If the command kubectl get nodes shows the following error, then we need to turn off swap and restart Kubelet service again

Error: The connection to the server 10.0.x.x:6443 was refused - did you specify the right host or port?

Solution:

sudo -i #Switches user to root

swapoff -a

exit

sudo systemctl restart kubelet.service