Lab – Kubernetes cluster installation

F. Lassabe

Introduction

This lab aims at deploying a Kubernetes cluster, then to use it for deploying a sample application.

1 Deploying Kubernetes

You shall deploy your K8s cluster into a VirtualBox machines. You may either do it on your own computer, or on the proxmox server that you used for Proxmox lab (since it allows accessing container repositories). For each VM, allocate the following hardware resources:

- Debian 12
- 2 cores
- 4GB RAM
- 20 GB hard drive

The VM's network shall use the range 10.0.2.0/24 (default for NAT network), and the pod network range shall be 192.168.51.0/24. Your hosts IPs shall be:

- Control plane: 10.0.2.100/24, named control
- Worker 1: 10.0.2.101/24, named wrk1
- Worker 2: 10.0.2.102/24, named wrk2

Set your `/etc/hosts` and `/etc/hostname` files accordingly.

1.1 Preparation – all VM

1.1.1 Disabling swap

You shall remove swap from the hosts, as well as enabling bridged traffic on all nodes.

```
sudo swapoff -a
```

Do not forget to comment out the line related to swap in /etc/fstab.

1.1.2 Open required ports

You shall allow access to the following ports (or disable Debian firewall, but never do that in production environments):

- On control plan: TCP ports 6443, 2379, 2380, 10250, 10251, 10252, and 10255
- On workers: TCP ports 10250, and 30000 to 32767

1.1.3 Installing containerd runtime

Add the following lines to file `/etc/modules-load.d/containerd.conf` (create it if it doesn't exist):

```
overlay
br_netfilter
```

And enable both modules:

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

You may load these modules at startup by creating a file, e.g. `99-kubernetes.conf`, in `/etc/sysctl.d` whose content is:

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

Apply changes with:

```
sudo sysctl --system
```

Install and configure containerd:

```
sudo apt update
sudo apt -y install containerd
containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
```

Edit file `/etc/containerd/config.toml` and find section `[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]`, and change entry `SystemdCgroup` from false to true.
Restart and enable containerd, so it will start automatically:

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```

1.1.4 Install Kubernetes

Add Kubernetes repository and key, then install its components on all nodes:

```
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
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key \
  | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
sudo apt update
sudo apt install kubelet kubeadm kubectl -y
sudo apt-mark hold kubelet kubeadm kubectl
```

1.2 Initializing the cluster

1.2.1 On the control plan

Create a configuration file, e.g. k8s.yaml, with the following content:

```
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: "1.28.0"
controlPlaneEndpoint: "control" # change according to your naming
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
```

Then, initialize the cluster with your configuration:

```
sudo kubeadm init --config kubelet.yaml
```

Note it will output the required command to join a worker to the cluster. To check your cluster, use the following commands on the control plan:

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

If you need to remind the command used to join the cluster, you may invoke this command:

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

1.2.2 Join the cluster with workers

You'll also need to setup your configuration in `.kube` directory:

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

On your worker nodes, join them to the cluster by running the command that was displayed when you initialized the master node:

```
sudo kubeadm join control:6443 --token YOUR TOKEN HERE \
--discovery-token-ca-cert-hash sha256:YOUR CERT HERE
```

After joining, check from the control plan that you see your workers:

```
sudo kubectl get nodes
```

You see all nodes are not ready, since they have no network plugin. We'll add the Calico network plugin.

1.3 Networking plugin

We'll use Calico as networking plugin, although there are plenty of such plugins:

```
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/calico.yaml
```

If you have a firewall, open the following ports:

- TCP: 179
- UDP: 4789, 51820, and 51821

You shall see calico services with the kube-system list after a couple minutes:

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

Your nodes shall now appear ready.

2 Using your Kubernetes cluster

Use the following commands to create a service nginx with two replicas of the server:

```
kubectl create deployment nginx-app --image=nginx --replicas 2
kubectl expose deployment nginx-app --name=nginx-web-svc --type NodePort --port 80 --target-port 80
kubectl describe svc nginx-web-svc
```

The last command will list the service status, along with the port available to access it. Add a Ubuntu VM to your setup and try to access the service from a web browser.