Setting up Kubernetes from VMs

Lets setup a basic Kubernetes cluster using Virtual Machines (VMs).

Plan

Here's the overall plan:

- [x] Install VirtualBox
- [x] Install AlmaLinux 9.6 Minimal ISO (choose ARM64 for Apple Silicon)
- [x] Spin up two VMs: one for the master node, and one for the worker node.
- [x] Install Kubernetes using `kubeadm`.

Workflow

Let's break down the steps:

- [x] **Install VirtualBox**
- [x] **Download ISO**
  - Download the AlmaLinux 9.6 Minimal ISO.
- [x] **Configure VirtualBox Network Settings**
  - Chosen **NAT Network** as the primary network mode.
  - Created a network named `k8s-network`.
  - Defined the CIDR (Classless Inter-Domain Routing): `10.0.2.0/24`.
    - `/24` means that 24 out of the 32 bits are used for the network address, leaving 8 bits for host addresses.
    - This gives us the IP address range: `10.0.2.0 - 10.0.2.255`.
  - Disabled DHCP to assign static IP addresses to our VMs.
- [x] **Create VM1: Master Node** (4GB RAM, 15GB storage)
  - Name: `k8s-master1`
  - ISO: Path to the downloaded ISO.
  - Type: Linux
  - SubType: Oracle Linux
  - Version: Oracle Linux 9.x
  - Network: Attached to **NAT Network** (`k8s-network`)
  - **Installing OS:**
    - Created a root user password.
    - Created a new sudo user `k8s` with a password.
    - Disabled KDUMP.

        - Configured a static IP address:
            - IP: `10.0.2.10`
            - NetMask: `255.255.255.0`
            - Gateway: `10.0.2.1`
            - DNS: `8.8.8.8`
        - Began the installation.
        - Rebooted the system.
        - Installation completed, got the shell.
        - Checked IP address: `ip addr show` (should show `10.0.2.10/24`).
        - Checked internet connectivity: `ping hrushispace.com`.
        - Checked OS info: `cat /etc/os-release`.
        - Checked memory: `free -h`.
        - Checked disk: `df -h`.
        - Checked CPU: `lscpu`.
- [x] **Create VM2: Worker Node** (4GB RAM, 15GB storage)
    - Name: `k8s-worker1`
    - ISO: Path to the downloaded ISO.
    - Type: Linux
    - SubType: Oracle Linux
    - Version: Oracle Linux 9.x
    - Network: Attached to **NAT Network** (`k8s-network`)
    - **Installing OS:**
        - Created a root user password.
        - Created a new sudo user `k8s` with a password.
        - Disabled KDUMP.
        - Configured a static IP address:
            - IP: `10.0.2.11`
            - NetMask: `255.255.255.0`
            - Gateway: `10.0.2.1`
            - DNS: `8.8.8.8`
        - Began the installation.
        - Rebooted the system.
        - Installation completed, got the shell.
        - Checked IP address: `ip addr show` (should show `10.0.2.11/24`).
        - Checked internet connectivity: `ping hrushispace.com`.
        - Checked OS info: `cat /etc/os-release`.
        - Checked memory: `free -h`.
        - Checked disk: `df -h`.

    - Checked CPU: `lscpu`.

- [x] **Verify VM Communication**
    - On `k8s-master1`: `ping 10.0.2.11`
    - On `k8s-worker1`: `ping 10.0.2.10`
- [x] **Check Active Services**
    - On each VM: `systemctl list-units --type=service --state=actives`
- [x] **Disable Firewall**
    - By default, the firewall is active, which can block necessary communication. While you can open specific ports, disabling it for this basic setup is simpler.
    - On both master and worker nodes:

```
sudo systemctl disable firewalld --now
```

- [x] **Disable Swap**
    - Swap is virtual memory on disk.
    - Kubernetes prefers real memory for predictable resource allocation.
    - Disable swap:

```
sudo swapoff -a
```

    - To prevent swap from being enabled on reboot, comment it out in `/etc/fstab`:

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

- [x] **Adding Kernel Modules**
    - Think of kernel modules as extensions for your Linux OS, providing extra functionality when needed. Kubernetes networking relies on `overlay` and `br_netfilter`.
    - Go to `/etc/modules-load.d/`.
    - Create a file `k8s.conf` and add these two modules:

```
overlay
br_netfilter
```

        - `overlay`: Used by container runtimes like Podman for efficient management of container image layers.
        - `br_netfilter`: Crucial for allowing network traffic through Linux bridges to be processed by the firewall (iptables), which Kubernetes often uses for its networking.
    - Load the modules using `modprobe`:

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

- Alternatively, rebooting the VMs will automatically load these modules.

- [x] **Configure `sysctl` for Network**
  - For communication between pods and to enable firewall rules for pod traffic, configure `sysctl`.
  - Create `k8s.conf` inside `/etc/sysctl.d/` and add these lines:

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

    - `net.bridge.bridge-nf-call-iptables = 1` : Enables iptables rules for traffic traversing Linux bridges.
    - `net.bridge.bridge-nf-call-ip6tables = 1` : Same as above, but for IPv6.
    - `net.ipv4.ip_forward = 1` : Allows IP forwarding, necessary for inter-pod communication across different nodes.
  - Reload the `sysctl` configuration:

```
sudo sysctl --system
```

- [x] **Install CRI-O Container Runtime**
  - Create a repository file for CRI-O:

```
cat <<EOF | sudo tee /etc/yum.repos.d/cri-o.repo
[cri-o]
name=CRI-O
baseurl=[https://download.opensuse.org/repositories/isv:/cri-o:/stable:/$](https:
enabled=1
gpgcheck=1
gpgkey=[https://download.opensuse.org/repositories/isv:/cri-o:/stable:/$](https
EOF
```

  - Install CRI-O and `container-selinux` :

```
sudo dnf install -y container-selinux cri-o
sudo systemctl start crio.service
```

  - CRI (Container Runtime Interface) is an abstraction layer that allows Kubernetes to work with different container runtimes.

- [x] **Install Kubernetes**
  - Add the Kubernetes repository:

    ```
    cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
    [kubernetes]
    name=Kubernetes
    baseurl=[https://pkgs.k8s.io/core:/stable:/v1.29/rpm/](https://pkgs.k8s.io/core
    repo_gpgcheck=1
    gpgcheck=1
    gpgkey=[https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key](ht
    enable=1
    EOF
    ```

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

    ```
    sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
    ```

- [x] **Configure Master Node**
  - Initialize the Kubernetes control plane on the master node:

    ```
    sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --cri-socket unix:///var/run/
    ```

  - Configure `kubectl` to work with your cluster:

    ```
    mkdir -p /home/k8s/.kube
    sudo cp /etc/kubernetes/admin.config /home/k8s/.kube/config
    sudo chown k8s:k8s /home/k8s/.kube/config
    ```

- [x] **Configure Worker Node**
  - You'll see a `kubeadm join` command output by the `kubeadm init` command on the master node. Copy and run that command on the worker node. It will look something like this:

    ```
    kubeadm join <your_master_ip>:<port> --token <token> --discovery-token-ca-cert-h
    ```

- [x] **Install a CNI Plugin**
  - Kubernetes needs a Container Network Interface (CNI) plugin to enable communication between pods. We'll install Calico:

    ```
    kubectl apply -f [https://raw.githubusercontent.com/projectcalico/calico/v3.27.0
    ```

  - Once the CNI plugin is deployed, your worker node should be able to join the cluster.

```
[k8s@10 kubernetes]$ kubectl get nodes
NAME         STATUS   ROLES           AGE   VERSION
10.0.2.10    Ready    control-plane   66m   v1.29.15
10.0.2.11    Ready    <none>          28m   v1.29.15
[k8s@10 kubernetes]$
```

VirtualBox Network Modes

Understanding VirtualBox network modes is key:

- **1. NAT (Default)** - [Room with no door between rooms, only enter/exit from/to outside]
  - VM gets internet through the host.
  - VMs are isolated from each other.
  - VM can access the internet.
  - VMs cannot reach each other.
- **2. NAT Network** - [Rooms connected by Hallway with access to outside]
  - VMs share a virtual network and can communicate.
  - VM can access the internet.
  - Host cannot directly reach the VM.
  - VMs can reach each other.
- **3. Host-only Adapter** - [Same as NAT but no access to outside]
  - Creates a private network between the host and VMs.
  - VM cannot access the internet.
  - Host can reach the VM.
  - VMs can reach each other.
- **4. Bridged Adapter** - [Each room has access to outside]
  - VM connects directly to your physical network.
  - VM can access the internet.
  - Host can reach the VM.
  - VMs can reach each other.

DHCP

Dynamic Host Configuration Protocol - An automatic IP address assignment service that gives devices network settings when they connect.

Modprobe

Modprobe is a command-line utility in Linux used to add and remove modules from the Linux kernel.