DEVOPS SHACK

KUBERNETES Local Setup

Detailed Kubernetes Installation on Ubuntu (Master and Worker Nodes)

This guide outlines the detailed steps for setting up a Kubernetes cluster on Ubuntu machines. It includes instructions for both master and worker nodes and covers the necessary configurations for networking and container runtimes. The cluster is initialized using kubeadm, with Calico as the network plugin.

Prerequisites

- All machines (both master and worker nodes) should run a compatible version of Ubuntu (e.g., Ubuntu 20.04 or 22.04).
- Ensure that you have root or sudo privileges on all machines.
- At least two nodes: one for the master and one or more for the worker nodes.
- Minimum resources:
    - **Master node**: 2 CPUs and 2GB RAM
    - **Worker nodes**: 1 CPU and 1GB RAM each

----------ON ALL NODES----------

```
sudo apt update && sudo apt upgrade -y

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

sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter

sudo tee /etc/sysctl.d/kubernetes.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

sudo sysctl --system

sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o
/etc/apt/trusted.gpg.d/docker.gpg
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"

sudo apt update
sudo apt install -y containerd.io

containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml

sudo systemctl restart containerd
sudo systemctl enable containerd

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg

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

----------On Master Node----------

```
sudo kubeadm init

# Run output command on worker node

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

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

 # Detailed Explaination added below

Step 1: Set Up on All Nodes (Master and Worker)

1. **Update and Upgrade Packages:**

   Update the package index and upgrade the installed packages to ensure that you have the latest security patches and updates.

   sudo apt update && sudo apt upgrade -y

2. **Disable Swap:**

   Kubernetes requires swap to be disabled for optimal performance. Disabling swap ensures that Kubernetes can efficiently allocate resources.

   sudo swapoff -a

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

   The first command disables swap for the current session, and the second command ensures that swap remains disabled after a reboot.

3. **Load Required Kernel Modules:**

   Kubernetes uses certain kernel modules to handle networking and container operations. These modules need to be loaded and configured.

   - Load kernel modules:

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

   - Activate the modules:

   sudo modprobe overlay

   sudo modprobe br_netfilter

4. **Network Configuration for Kubernetes:**

   Modify the system's network settings to enable bridge networking. This is necessary for Kubernetes networking components like Calico.

   sudo tee /etc/sysctl.d/kubernetes.conf <<EOF

   net.bridge.bridge-nf-call-ip6tables = 1

   net.bridge.bridge-nf-call-iptables = 1

   net.ipv4.ip_forward = 1

   EOF

   Apply the changes:

   sudo sysctl --system

5. **Install Required Dependencies:**

   Install packages that are necessary for adding new repositories and handling secure connections.

   sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates

---

Step 2: Install and Configure containerd (Container Runtime)

Kubernetes needs a container runtime to run the containers. We will use **containerd** as it is a lightweight and recommended container runtime for Kubernetes.

1. **Add Docker GPG Key and Repository:**

   Add the Docker repository that contains the containerd.io package.

   sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg

   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

2. **Install containerd.io:**

   After adding the Docker repository, install the containerd.io package.

   sudo apt update

   sudo apt install -y containerd.io

3. **Configure containerd:**

   Configure containerd to use the systemd cgroup driver, which is recommended for Kubernetes.

   - Create a default configuration file:

   containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1

   - Edit the configuration to set SystemdCgroup to true:

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

4. **Restart and Enable containerd:**

   Restart the containerd service to apply the changes and enable it to start on boot.

   sudo systemctl restart containerd

   sudo systemctl enable containerd

Step 3: Install Kubernetes Components (kubelet, kubeadm, kubectl)

1. **Add Kubernetes GPG Key and Repository:**

   Add the Kubernetes package repository and its GPG key.

   - Add the Kubernetes repository:

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

   - Add the GPG key:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

2. **Install Kubernetes Components:**

   Now, install kubelet, kubeadm, and kubectl.

```
sudo apt update
```

```
sudo apt install -y kubelet kubeadm kubectl
```

3. **Mark the Packages on Hold:**

   Prevent the installed packages from being updated accidentally to avoid breaking the cluster.

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

Step 4: Initialize the Master Node

On the master node, you will initialize the Kubernetes cluster using kubeadm. This step sets up the control plane (API Server, etcd, Scheduler, Controller Manager).

1. **Run kubeadm init:**

   This command initializes the master node and sets up the control plane.

```
sudo kubeadm init
```

   After running this command, kubeadm will output several important pieces of information:

   - A **join command** to be used on worker nodes to add them to the cluster.
   - The admin.conf file, which is needed to communicate with the cluster.

2. **Set Up kubectl for the Master Node:**

   To start using kubectl to manage your cluster, you need to copy the admin.conf file to your home directory.

```
mkdir -p $HOME/.kube
```

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

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

This configures the kubectl CLI to connect to the newly created cluster using the correct credentials.

Step 5: Set Up Networking (Calico)

Kubernetes requires a network plugin to manage the communication between nodes and pods. Here we are using **Calico** for this purpose.

1. **Install Calico Networking:**

   Apply the Calico manifest to set up networking in the cluster.

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

This command installs Calico as the network plugin for the cluster. Calico is a highly scalable networking and network security solution for Kubernetes.

Step 6: Join Worker Nodes to the Cluster

Once the master node is initialized, you need to join worker nodes to the cluster using the **join command** generated during the kubeadm init process.

1. **Run the Join Command on Each Worker Node:**

   On each worker node, run the join command from the master node's output. It will look something like this:

```
sudo kubeadm join <master-node-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

This command will register the worker node with the master and start the communication between them.

2. **Verify the Worker Nodes Have Joined:**

   On the master node, use the following command to verify that the worker nodes have successfully joined the cluster:

```
kubectl get nodes
```

You should see the master and worker nodes listed with their status as Ready.

Conclusion

You have now set up a basic Kubernetes cluster with one master node and one or more worker nodes using kubeadm, kubectl, and kubelet. The cluster is using containerd as the container runtime, and Calico is used as the network plugin to handle communication between the pods.

This cluster setup forms the foundation for deploying and managing containerized applications in a production or development environment. You can now deploy applications to your Kubernetes cluster, scale them, and manage their lifecycle using Kubernetes' powerful orchestration features.

Feel free to expand this basic setup with additional configurations, such as persistent storage, advanced security (RBAC), and monitoring tools.