Lab : How to Install Kubernetes Cluster on Ubuntu 22.04

Prerequisites

In this Lab, we are using one master node and two worker nodes. Following are system requirements on each node,

- Minimal install Ubuntu 22.04
- Minimum 2GB RAM or more
- Minimum 2 CPU cores / or 2 vCPU
- 20 GB free disk space on /var or more
- Sudo user with admin rights
- Internet connectivity on each node

Lab Setup

- Master Node: 192.168.1.173 – k8smaster.example.net
- First Worker Node: 192.168.1.174 – k8sworker1.example.net
- Second Worker Node: 192.168.1.175 – k8sworker2.example.net

1) Set hostname on Each Node

Login to to master node and set hostname via hostnamectl command,

```
$ sudo hostnamectl set-hostname "k8smaster.example.net"
```

On the worker nodes, run

```
$ sudo hostnamectl set-hostname "k8sworker1.example.net"   // 1st worker node
```

```
$ sudo hostnamectl set-hostname "k8sworker2.example.net"   // 2nd worker node
```

Add the following lines in /**etc/hosts** file on each node

```
192.168.1.173   k8smaster.example.net k8smaster

192.168.1.174   k8sworker1.example.net k8sworker1

192.168.1.175   k8sworker2.example.net k8sworker2
```

2) Disable Swap & Add kernel Parameters on each node

Execute beneath swapoff and sed command to disable swap. Make sure to run the following commands on all the nodes.

```
$ sudo swapoff -a

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

Load the following kernel modules on all the nodes,

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

overlay

br_netfilter

EOF

$ sudo modprobe overlay

$ sudo modprobe br_netfilter
```

Set the following Kernel parameters for Kubernetes, run beneath tee command

```
$ sudo tee /etc/sysctl.d/kubernetes.conf <<EOT

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

net.ipv4.ip_forward = 1

EOT
```

Reload the above changes, run

```
$ sudo sysctl --system
```

3) Install Containerd Runtime

In this guide, we are using containerd runtime for our Kubernetes cluster. So, to install containerd, first install its dependencies.

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

Enable docker repository

```
$ 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"
```

Now, run following apt command to install containerd

```
$ sudo apt update

$ sudo apt install -y containerd.io
```

Configure containerd so that it starts using systemd as cgroup.

```
$ 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
```

Restart and enable containerd service

```
$ sudo systemctl restart containerd

$ sudo systemctl enable containerd
```

4) Add Apt Repository for Kubernetes

Kubernetes package is not available in the default Ubuntu 22.04 package repositories. So we need to add Kubernetes repository. run following command to download public signing key,

```
$sudo mkdir -p -m 755 /etc/apt/keyrings

$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Next, run following echo command to add Kubernetes apt repository.

```
$ 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
```

Note: At the time of writing this Lab, Kubernetes v1.28 was available, replace this version with new higher version if available.

5) Install Kubectl, Kubeadm and Kubelet (on each node)

Post adding the repositories, install Kubernetes components like kubectl, kubelet and Kubeadm utility on all the nodes. Execute following set of commands,

```
$ sudo apt update

$ sudo apt install -y kubelet kubeadm kubectl

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

6) Install Kubernetes Cluster on Ubuntu 22.04

Now, we are all set to initialize Kubernetes cluster. Run the following Kubeadm command on the master node only.

```
$ sudo kubeadm init --control-plane-endpoint=k8smaster.example.net
```

After the initialization is complete, you will see a message with instructions on how to join worker nodes to the cluster. Make a note of the kubeadm join command for future reference.

So, to start interacting with cluster, run following commands on the master node,

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

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

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

next, try to run following kubectl commands to view cluster and node status

```
$ kubectl cluster-info

$ kubectl get nodes
```

7) Join Worker Nodes to the Cluster

On each worker node, use the kubeadm join command you noted down earlier after initializing the master node on step 6. It should look something like this:

```
$ sudo kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>
```

If you do not have the token, you can get it by running the following command on the master node:

```
$ kubeadm token list
```

Above output from worker nodes confirms that both the nodes have joined the cluster.Check the nodes status from master node using kubectl command,

```
$ kubectl get nodes
```

As we can see nodes status is ‘NotReady’, so to make it active. We must install CNI (Container Network Interface) or network add-on plugins like Calico, Flannel and Weave-net.

8) Install Calico Network Plugin

A network plugin is required to enable communication between pods in the cluster. Run following kubectl command to install Calico network plugin from the master node,

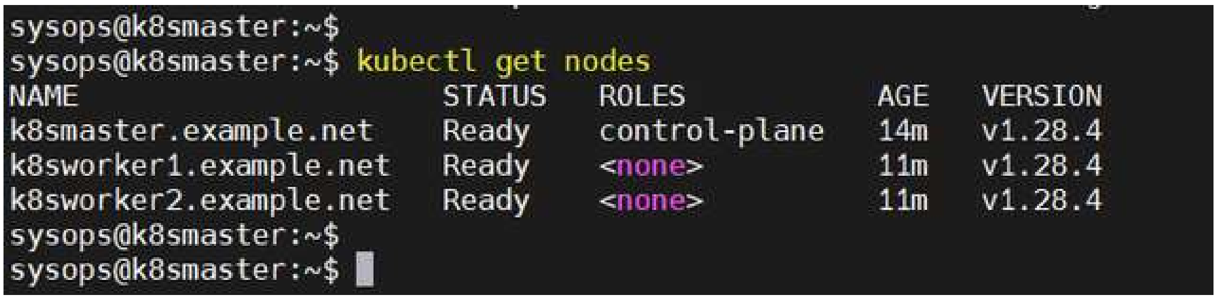
```
$ kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml
```

Verify the status of pods in kube-system namespace,

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

Verify the status of nodes.

```
$ kubectl get nodes
```



```
sysops@k8smaster:~$
sysops@k8smaster:~$ kubectl get nodes
NAME                      STATUS   ROLES           AGE   VERSION
k8smaster.example.net     Ready    control-plane   14m   v1.28.4
k8sworker1.example.net    Ready    <none>          11m   v1.28.4
k8sworker2.example.net    Ready    <none>          11m   v1.28.4
sysops@k8smaster:~$
sysops@k8smaster:~$
```

Great, above confirms that nodes are active node. Now, we can say that our Kubernetes cluster is functional.

9) Test Your Kubernetes Cluster Installation

To test Kubernetes installation, let’s try to deploy nginx based application and try to access it.

```
$ kubectl create deployment nginx-app --image=nginx --replicas=2
```

Check the status of nginx-app deployment

```
$ kubectl get deployment nginx-app
```

Expose the deployment as NodePort,

```
$ kubectl expose deployment nginx-app --type=NodePort --port=80
```

Run following commands to view service status

```
$ kubectl get svc nginx-app

$ kubectl describe svc nginx-app
```

Use following curl command to access nginx based application,

$ curl http://<woker-node-ip-addres>:<NodePort>