How to Setup Kubernetes Cluster on Debian Based systems

Debian12, Ubuntu 22.04, LinuxMint21.4, LMDE6

Kubernetes is a free container orchestration tool that helps you achieve automated deployment, scaling, and management of containerized applications. A Kubernetes cluster consists of a Master node and a Worker node. The master node is responsible for managing nodes and pods in the cluster. The worker node is used to deploy the application workload. With Kubernetes, you can deploy and manage cloud-native applications using on-premises infrastructure or public cloud platforms.

This tutorial will show you how to set up the Kubernetes cluster on an Ubuntu 22.04 server.
Prerequisites

- A server running Debian based system. In my case LMDE6.
- Unrestricted internet access

Getting Started

First please verify that you don’t have any K8s or K3s installed and disabled:

```
ss -tulpn | grep 1025[0-9] # if the answer is not blank, then you have k8s or k3s is installed
systemctl status k3s
systemctl disable --now k3s
```

You will need to update and upgrade your system packages to the latest version. You can do it with the following command:

```
apt-get update -y
apt-get upgrade -y
apt-get install -y vim
```

Once all the packages are updated, you will also need to disable the Swap on your system. You can disable it with the following command:

```
swapoff -a
```

Next, you will need to load some required kernel modules on all nodes. To do so, edit the /etc/modules-load.d/containerd.conf file:

```
echo -n '\noverlay\nbr_netfilter\n' >> /etc/modules-load.d/containerd.conf
```

Save and close the file, then load the modules with the following command:

```
modprobe overlay
modprobe br_netfilter
```

Next, you will also need to create a fine and define some required kernel parameters:

```
echo "
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
" >>  /etc/sysctl.d/kubernetes.conf
```

Run the following command to apply the changes:

```
sysctl --system
```

Install Container Run Environment

Next, you will need to install the Docker for the Kubernetes cluster. First, install all the required dependencies using the following the easy use script by docker themselves:

```
curl -L https://get.docker.com | sudo bash
usermod -aG docker ${USER}
```

Next, you will need to configure the Containers so it starts by systemd. You can do it with the following command:

```
sudo sed -i 's/disabled_plugins.*/#disabled_plugins = ["cri"]/g'\
    /etc/containerd/config.toml
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g'\
    /etc/containerd/config.toml
```

Next, restart the Containerd service to apply the changes:

```
sudo systemctl restart containerd
```

Install Kubernetes Components

By default, Kubernetes components are NOT included in the default repositories. So you will need to add the Kubernetes repository to your system. You can add it with the following command:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/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:/v1.30/deb/ /" |\
    sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Next, update the repository and install all Kubernetes components with the following command:

```
apt-get update -y
apt-get install kubelet kubeadm kubectl -y
```

Once all the packages are installed, you can proceed to the next step.

Initialize Kubernetes Cluster

At this point, All Kubernetes components are installed. Now, run the following command on the master node **with root user** to initialize the cluster:

```
sudo su
kubeadm init --control-plane-endpoint=master-ip \
    --cri-socket=unix:///run/containerd/containerd.sock
```

You will get the following output:

```
To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of control-plane nodes by copying certificate authorities
and service account keys on each node and then running the following as root:

  kubeadm join master-ip:6443 --token chmz7m.fbjgdcqne1q0ff4t \
    --discovery-token-ca-cert-hash sha256:c614bf14af27472e470546539a9a2ff63e5d558dbbb3cc06d6f7a030fcb5
    --control-plane
```

Then you can join any number of worker nodes by running the following on each as root:

```
 kubeadm join master-ip:6443 --token chmz7m.fbjgdcqne1q0ff4t \
    --discovery-token-ca-cert-hash sha256:c614bf14af27472e470546539a9a2ff63e5d558dbbb3cc06d6f7a030fcb5
  --control-plane
```

Note: Copy the `kubeadm join` command from the above output. You will need to run this command on the Worker node to join the cluster. Next, you will need to run the following commands to interact with the Kubernetes cluster:

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

Join Worker Node to the Cluster

Next, log in to the Worker node and run the following command to join the Worker node to the Kubernetes cluster:

```
 kubeadm join master-ip:6443 --token chmz7m.fbjgdcqne1q0ff4t \
    --discovery-token-ca-cert-hash sha256:c614bf14af27472e470546539a9a2ff63e5d558dbbb3cc06d6f7a030fcb5
    --control-plane
```

You will get the following output:

```
[preflight] Running pre-flight checks
    [WARNING SystemVerification]: missing optional cgroups: blkio
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.en
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
```

This node has joined the cluster: * Certificate signing request was sent to apiserver and a response was received. * The Kubelet was informed of the new secure connection details.

Run ‘kubectl get nodes’ on the control-plane to see this node join the cluster.

Install Calico Pod Network Add-on

Next, you will need to install the Calico Pod Network on the Kubernetes Master node to manage the network.

You can download and install it with the following command:

```
curl https://projectcalico.docs.tigera.io/manifests/calico.yaml -O calico.yaml
kubectl apply -f calico.yaml
```

You will get the following output:

```
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org crea
```

```
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
```

Next, verify the status of pods using the following command:

```
kubectl get pods -n kube-system
NAME                                          READY   STATUS    RESTARTS   AGE
calico-kube-controllers-58dbc876ff-nh2st      1/1     Running   0          5m58s
calico-node-7cfz7                             1/1     Running   0          5m58s
calico-node-lt5cv                             1/1     Running   0          5m58s
coredns-565d847f94-dm6qc                      1/1     Running   0          21m
coredns-565d847f94-zhng9                      1/1     Running   0          21m
etcd-k8smaster.example.net                    1/1     Running   0          22m
kube-apiserver-k8smaster.example.net          1/1     Running   0          22m
kube-controller-manager-k8smaster.example.net 1/1     Running   0          22m
kube-proxy-9w2xp                              1/1     Running   0          14m
kube-proxy-gdb97                              1/1     Running   0          21m
kube-scheduler-k8smaster.example.net          1/1     Running   0          22m
```

You can now check the status of the Kubernetes cluster with the following command:

```
kubectl get nodes
NAME                   STATUS   ROLES           AGE   VERSION
k8smaster.example.net   Ready    control-plane   22m   v1.25.0
kubernetes             Ready              14m   v1.25.0
```

Deploy an Nginx Application on Kubernetes

To test the Kubernetes, we will deploy an Nginx application on the cluster.

Run the following command to deploy an Nginx app:

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

You can verify your app with the following command:

```
kubectl get deployment nginx-app
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
nginx-app   2/2     2            2           13s
```

Next, expose your application on port 80 with the following command:

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

Next, verify the Nginx service status with the following command:

```
kubectl get svc nginx-app
NAME        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nginx-app   NodePort   10.109.89.196           80:30921/TCP   14s
```

You can also see the detailed information of your Nginx app using the following command:

```
kubectl describe svc nginx-app
```

You should see the following output:

```
Name:                     nginx-app
Namespace:                default
```

```
Labels:                   app=nginx-app
Annotations:
Selector:                 app=nginx-app
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.109.89.196
IPs:                      10.109.89.196
Port:                       80/TCP
TargetPort:               80/TCP
NodePort:                   30921/TCP
Endpoints:                192.168.192.129:80,192.168.192.130:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
```

Now, note down the Nginx application IP address from the above output and verify your Nginx app using the curl command:

```
curl http://10.109.89.196
```

If everything is fine, you will get the following output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
```

```
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Congratulations! you have successfully deployed the Kubernetes cluster on Debian based system.