Kubernetes Installation in Ubuntu Machines

In this Lab, you will learn how to deploy Kubernetes cluster using kubeadm tool with one Master node and two Worker nodes (Ubuntu Machines).

Prerequisites:

- A compatible Linux host. The Kubernetes project provides generic instructions for Linux distributions based on Debian and Red Hat, and those distributions without a package manager.
- 2 GB or more of RAM per machine (any less will leave little room for your apps).
- 2 CPUs or more.
- Full network connectivity between all machines in the cluster (public or private network is fine).
- Unique hostname, MAC address, and product_uuid for every node.
- Swap disabled. You **MUST** disable swap in order for the kubelet to work properly
- Certain ports are open on your machines which are shown below.

Control plane

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10259	kube-scheduler	Self
TCP	Inbound	10257	kube-controller-manager	Self

Although etcd ports are included in control plane section, you can also host your own etcd cluster externally or on custom ports.

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services†	All

Follow the below steps to install Kubernetes:

1. **Access the machines: -**

   Login to Master and Workers node as root and set the hostname of each machine.

hostnamectl set-hostname master

hostnamectl set-hostname worker1

hostnamectl set-hostname worker2

2. **Install the packages: -**

Use the below script to install Docker, Kubeadm, Kubelet and Kubectl packages in one go. Please make sure to use the script in all the machines and install the packages further.

2.1 create and script using any editor.

```
# vi script.sh
```

copy and paste the below script inside the script and save it.

```
#!/bin/bash
sudo apt update -y
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update -y
apt-cache policy docker-ce -y
sudo apt install docker-ce -y
wget -q -O - https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo deb http://apt.kubernetes.io/ kubernetes-xenial main | sudo tee /etc/apt/sources.list.d/kubernetes.list
apt update -y
apt install kubelet=1.21.1-00 kubeadm=1.21.1-00 kubectl=1.21.1-00 -y
sysctl net.bridge.bridge-nf-call-iptables=1
```

2.2 run the script to install all the packages.

```
# sh script.sh
```

Note: above script should be used in all the machines to install the required packages.

3. **Initializing your control-plane node: -**

We will be using **kubeadm init** command to initialize the cluster and will also provide the pod network CIDR which will be used to provide the IP addresses to the pods.

Use the below mentioned command only on the master node and save the output in any file.

```
# kubeadm init --pod-network-cidr=192.168.0.0/16 >> cluster_initialized.txt
```

Above command will start our cluster and install the main components as pods and create the necessary certificates. We have saved the output in **cluster_initialized.txt** file. Open this file for further instructions. The output of the file will look like below.

```
Your Kubernetes control-plane has initialized successfully!

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/

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

kubeadm join 172.31.0.123:6443 --token 6n58id.ejx5hwbwpvtjk5jf \
        --discovery-token-ca-cert-hash
sha256:189c11cb8cfafbccea50c90c2c81f74464f78d8209df88c3d887bcde25660ff5
```

Our Kubernetes control-plane has been installed successfully and now we have to implement the above commands to make it work.

Use the below command only on the **master node** as mentioned in above file.

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

4. **Install the network plugin: -**

   Kubernetes itself is not handling the networking part, instead it takes help from network plugins. Kubernetes is compatible with third party plugins like Calico, Canal, Cilium, Flannel, Weave Net etc.

   We will be using Calico plugins for the networking part which will be installed only on the **master node** which is also instructed in the above file (cluster_initialized.txt). Use the below command to install calico plugin.

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

5. **Join worker nodes to the cluster:**

   Now the last step is to use the token on the **worker nodes** to make them a part of this cluster which is also instructed in above file(cluster_initialized.txt).

   Use the below command only on the **worker nodes**. Note: every cluster would be generating their own token so below token is just for the reference. Use the token which will be generated by your kubeadm init command.

```
kubeadm join 172.31.0.123:6443 --token 6n58id.ejx5hwbwpvtjk5jf \
        --discovery-token-ca-cert-hash
sha256:189c11cb8cfafbccea50c90c2c81f74464f78d8209df88c3d887bcde25660ff5
```

6. **Check the status:**

   Now the cluster is ready and we can check the status by using the below command on the master node.

```
kubectl get nodes
```

   This will show us the number of nodes which are a part of this cluster. If our case we are having total 3 machines and below will be the output.

```
NAME      STATUS   ROLES                  AGE   VERSION
master    Ready    control-plane,master   2d9h  v1.21.1
worker1   Ready    <none>                 2d9h  v1.21.1
worker2   Ready    <none>                 2d9h  v1.21.1
```

   Now we will check the status of the main components which are running as pods and managing the cluster.

Kubectl get pods -n kube-system

Below we can see, we are having the management pods which are running.

```
NAME                                    READY   STATUS    RESTARTS   AGE
calico-kube-controllers-846d7f49d8-rwnn2   1/1     Running   40         2d17h
calico-node-7zc5t                       1/1     Running   5          2d17h
calico-node-bsbmx                       1/1     Running   5          2d17h
calico-node-dlnpv                       1/1     Running   5          2d17h
coredns-558bd4d5db-6czq9                1/1     Running   6          2d21h
coredns-558bd4d5db-lv5wk                1/1     Running   6          2d21h
etcd-master                             1/1     Running   4          45h
kube-apiserver-master                   1/1     Running   1          13h
kube-controller-manager-master          1/1     Running   25         2d21h
kube-proxy-8h4l2                        1/1     Running   6          2d21h
kube-proxy-bkrxq                        1/1     Running   6          2d21h
kube-proxy-vjfv2                        1/1     Running   6          2d21h
kube-scheduler-master                   1/1     Running   24         2d21h
```

Now your cluster is ready 😊