**Setup a multi-master Kubernetes cluster with kubeadm**

[Kubernetes](https://kubernetes.io/docs/getting-started-guides/ubuntu/local/)is a system designed to manage containerized applications built within Docker containers in a clustered environment. It provides basic mechanisms for deployment, maintenance and scaling of applications on public, private or hybrid setups, means, it handles the entire life cycle of a containerized application. It also has intelligence of self-healing features where containers can be auto provisioned, restarted, or even replicated.

**Kubernetes Components**

**etcd** — A highly available key-value store for shared configuration and service discovery.

**kube-apiserver** — Provides the API for Kubernetes orchestration.

**kube-controller-manager** — Enforces Kubernetes services.

**kube-scheduler** — Schedules containers on hosts.

**kubelet** — Processes a container manifest so the containers are launched according to how they are described.

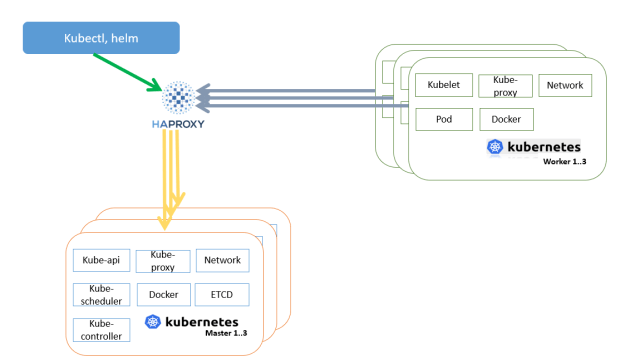
**kube-proxy** — Provides network proxy services.

**Read More:**[**What is Kubernetes, its basics and components**](https://foxutech.com/what-is-kubernetes/)

Okay, now let’s see how to setup multi-master kubernetes setup with kubeadm. Major advantage for this setup is keeping the cluster HA.

**Prerequisites**

In this example, we will be using an Ubuntu 18.04 as a base image for the seven machines needed. The machines will all be configured on the same network, 10.1.1.0/24, and this network needs to have access to the Internet.



As per the flow diagram, we are going to setup HAProxy first on machine 10.1.1.11. then we will be setup three Kubernetes master nodes. These machines will have the IPs 10.1.1.21, 10.1.1.22, and 10.1.1.23. Finally, lets setup three Kubernetes worker nodes with the IPs 10.1.1.31, 10.1.1.32, and 10.1.1.33.

We also need an IP range for the pods. Let’s set as 10.2.0.0/16, but it is only internal to Kubernetes.

In this setup we will be using ubuntu 18.04 installed to generate all the necessary certificates, and to manage the Kubernetes cluster. If you don’t have a Linux system, you can use the HAProxy machine to manage or generate the certificate.

**Client Tools**

We need two tools on the client machine: the Cloud Flare SSL tool to generate the different certificates, and the Kubernetes client, kubectl, to manage the Kubernetes cluster.

**Installing cfssl**

1. Download the binaries.

# wget <https://pkg.cfssl.org/R1.2/cfssl_linux-amd64>  
# wget <https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64>

2. Add the execution permission to the binaries.

# chmod +x cfssl\*

3. Move the binaries to /usr/local/bin.

# mv cfssl\_linux-amd64 /usr/local/bin/cfssl  
# mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

4. Verify the installation.

# cfssl version

**Installing kubectl**

1. Download the binary.

# wget <https://storage.googleapis.com/kubernetes-release/release/v1.15.0/bin/linux/amd64/kubectl>

2. Add the execution permission to the binary.

# chmod +x kubectl

3. Move the binary to /usr/local/bin.

# mv kubectl /usr/local/bin

4. Verify the installation.

# kubectl version

# Setup a multi-master Kubernetes cluster with kubeadm

By [**motoskia**](https://foxutech.com/author/motoskia/) -

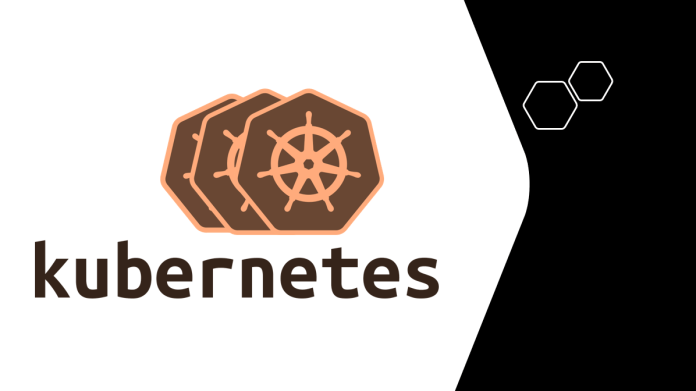
April 10, 2022

[0](https://foxutech.com/setup-a-multi-master-kubernetes-cluster-with-kubeadm/#respond)

1913

[Share on Facebook](http://www.facebook.com/sharer.php?u=https%3A%2F%2Ffoxutech.com%2Fsetup-a-multi-master-kubernetes-cluster-with-kubeadm%2F)

[Tweet on Twitter](https://twitter.com/intent/tweet?text=Setup+a+multi-master+Kubernetes+cluster+with+kubeadm&url=https%3A%2F%2Ffoxutech.com%2Fsetup-a-multi-master-kubernetes-cluster-with-kubeadm%2F&via=FoxuTech)

[](https://i0.wp.com/foxutech.com/wp-content/uploads/2022/04/multi-master-Kubernetes-cluster-with-kubeadm.png?fit=1196%2C672&ssl=1)

[Kubernetes](https://kubernetes.io/docs/getting-started-guides/ubuntu/local/)is a system designed to manage containerized applications built within Docker containers in a clustered environment. It provides basic mechanisms for deployment, maintenance and scaling of applications on public, private or hybrid setups, means, it handles the entire life cycle of a containerized application. It also has intelligence of self-healing features where containers can be auto provisioned, restarted, or even replicated.

**Kubernetes Components**

**etcd** – A highly available key-value store for shared configuration and service discovery.

**kube-apiserver** – Provides the API for Kubernetes orchestration.

**kube-controller-manager** – Enforces Kubernetes services.

**kube-scheduler** – Schedules containers on hosts.

**kubelet** – Processes a container manifest so the containers are launched according to how they are described.

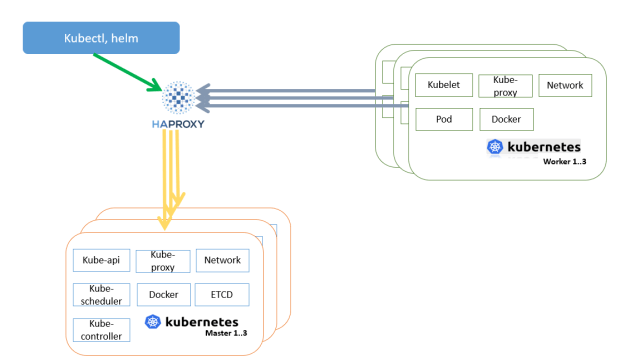
**kube-proxy** – Provides network proxy services.

**Read More:**[**What is Kubernetes, its basics and components**](https://foxutech.com/what-is-kubernetes/)

Okay, now let’s see how to setup multi-master kubernetes setup with kubeadm. Major advantage for this setup is keeping the cluster HA.

### **Prerequisites**

In this example, we will be using an Ubuntu 18.04 as a base image for the seven machines needed. The machines will all be configured on the same network, 10.1.1.0/24, and this network needs to have access to the Internet.

Setup a multi-master Kubernetes cluster with kubeadm

As per the flow diagram, we are going to setup HAProxy first on machine 10.1.1.11.  then we will be setup three Kubernetes master nodes. These machines will have the IPs 10.1.1.21, 10.1.1.22, and 10.1.1.23. Finally, lets setup three Kubernetes worker nodes with the IPs 10.1.1.31, 10.1.1.32, and 10.1.1.33.

We also need an IP range for the pods. Let’s set as 10.2.0.0/16, but it is only internal to Kubernetes.

In this setup we will be using ubuntu 18.04 installed to generate all the necessary certificates, and to manage the Kubernetes cluster. If you don’t have a Linux system, you can use the HAProxy machine to manage or generate the certificate.

#### **Client Tools**

We need two tools on the client machine: the Cloud Flare SSL tool to generate the different certificates, and the Kubernetes client, kubectl, to manage the Kubernetes cluster.

### **Installing cfssl**

1. Download the binaries.

# wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64  
# wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

2. Add the execution permission to the binaries.

# chmod +x cfssl\*

3. Move the binaries to /usr/local/bin.

# mv cfssl\_linux-amd64 /usr/local/bin/cfssl  
# mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

4. Verify the installation.

# cfssl version

#### **Installing kubectl**

1. Download the binary.

# wget https://storage.googleapis.com/kubernetes-release/release/v1.15.0/bin/linux/amd64/kubectl

2. Add the execution permission to the binary.

# chmod +x kubectl

3. Move the binary to /usr/local/bin.

# mv kubectl /usr/local/bin

4. Verify the installation.

# kubectl version

### **Setup HAProxy load balancer**

As we will deploy three Kubernetes master nodes, we need to deploy an HAPRoxy load balancer in front of them to distribute the traffic.

1. SSH to the 10.1.1.11 Ubuntu machine.

2. Update the machine.

# apt-get update  
# apt-get upgrade

3. Install HAProxy.

# apt-get install haproxy

4. Configure HAProxy to load balance the traffic between the three Kubernetes master nodes.

# vim /etc/haproxy/haproxy.cfg

global

...

default

...

frontend kubernetes

bind 10.1.1.11:6443

option tcplog

mode tcp

default\_backend kubernetes-master-nodes

backend kubernetes-master-nodes

mode tcp

balance roundrobin

option tcp-check

server k8s-master-0 10.1.1.21:6443 check fall 3 rise 2

server k8s-master-1 10.1.1.22:6443 check fall 3 rise 2

server k8s-master-2 10.1.1.23:6443 check fall 3 rise 2

5. Restart HAProxy.

# systemctl restart haproxy

### **Generating the TLS certificates**

These steps can be done on your Linux box if you have one or use HAProxy machine depending on where you installed the cfssl tool.

#### **Creating a certificate authority**

1. Create the certificate authority configuration file.

# vim ca-config.json

{

"signing": {

"default": {

"expiry": "8760h"

},

"profiles": {

"kubernetes": {

"usages": ["signing", "key encipherment", "server auth", "client auth"],

"expiry": "8760h"

}

}

}

}

2. Create the certificate authority signing request configuration file.

# vim ca-csr.json

{

"CN": "Kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "IE",

"L": "Cork",

"O": "Kubernetes",

"OU": "CA",

"ST": "Cork Co."

}

]

}

3. Generate the certificate authority certificate and private key.

# cfssl gencert -initca ca-csr.json | cfssljson -bare ca

4. Verify that the ca-key.pem and the ca.pem were generated.

#### **Creating the certificate for the Etcd cluster**

1. Create the certificate signing request configuration file.

# vim kubernetes-csr.json

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "IE",

"L": "Cork",

"O": "Kubernetes",

"OU": "Kubernetes",

"ST": "Cork Co."

}

]

}

2. Generate the certificate and private key.

# cfssl gencert \  
-ca=ca.pem \  
-ca-key=ca-key.pem \  
-config=ca-config.json \  
-  
hostname=10.1.1.21,10.1.1.22,10.1.1.23,10.1.1.11,127.0.0.1,kubernetes.default \  
-profile=kubernetes kubernetes-csr.json | \  
- cfssljson -bare kubernetes

3. Verify that the kubernetes-key.pem and the kubernetes.pem file were generated.

4. Copy the certificate to each node (you can use following command to copy the files to all the nodes together or you can do scp individually).

# for f in 10.1.1.21 10.1.1.22 10.1.1.23 10.1.1.31 10.1.1.32 10.1.1.33; do scp ca.pem kubernetes.pem kubernetes-key.pem ubuntu@f:~; done

### **Kubeadm Setup**

**Preparing the 10.1.1.21/22/23/31/32/33 machine**

Performing below steps on all systems

Installing Docker latest version

# curl -fsSL https://get.docker.com -o get-docker.sh  
# sh get-docker.sh  
# usermod -aG docker your-user

**Installing kubeadm, kublet, and kubectl**

1. Add the Google repository key.

# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

2. Add the Google repository.

# vim /etc/apt/sources.list.d/kubernetes.list  
deb http://apt.kubernetes.io kubernetes-xenial main

3. Update the list of packages and install kubelet, kubeadm and kubectl. In this example, we will be installing version 1.22.8-00. Also make sure you are hold the package from automatic update. If already hold, you need to “apt-mark unhold kubeadm” and then continue update/install.

# apt-get update  
# apt-get install kubelet=1.22.8-00 kubeadm=1.22.8-00 kubectl=1.22.8-00  
# apt-mark hold kubeadm

4. Disable the swap.

# swapoff -a  
# sed -i '/ swap / s/^/#/' /etc/fstab

#### **Installing and configuring Etcd**

Installing and configuring Etcd on the 10.1.1.21/22/23 machine (All 3 master)

1. SSH to the 10.1.1.21 machine.  
2. Create a configuration directory for Etcd.

# mkdir /etc/etcd /var/lib/etcd

3. Move the certificates to the configuration directory.

# mv ~/ca.pem ~/kubernetes.pem ~/kubernetes-key.pem /etc/etcd

4. Download the etcd binaries.

# wget https://github.com/etcd-io/etcd/releases/download/v3.3.13/etcd-v3.3.13-linux-amd64.tar.gz

5. Extract the etcd archive.

# tar xvzf etcd-v3.3.13-linux-amd64.tar.gz

6. Move the etcd binaries to /usr/local/bin.

# mv etcd-v3.3.13-linux-amd64/etcd\* /usr/local/bin/

7. Create an etcd systemd unit file.

# vim /etc/systemd/system/etcd.service

[Unit]

Description=etcd

Documentation=https://github.com/coreos

[Service]

ExecStart=/usr/local/bin/etcd \

--name 10.1.1.21 \

--cert-file=/etc/etcd/kubernetes.pem \

--key-file=/etc/etcd/kubernetes-key.pem \

--peer-cert-file=/etc/etcd/kubernetes.pem \

--peer-key-file=/etc/etcd/kubernetes-key.pem \

--trusted-ca-file=/etc/etcd/ca.pem \

--peer-trusted-ca-file=/etc/etcd/ca.pem \

--peer-client-cert-auth \

--client-cert-auth \

--initial-advertise-peer-urls https://10.1.1.21:2380 \

--listen-peer-urls https:// 10.1.1.11:2380 \

--listen-client-urls https://10.1.1.21:2379,http://127.0.0.1:2379 \

--advertise-client-urls https://10.1.1.21:2379 \

--initial-cluster-token etcd-cluster-0 \

--initial-cluster 10.1.1.21=https://10.1.1.21:2380,10.1.1.22=https://10.1.1.22:2380,10.1.1.23=https://10.1.1.23:2380 \

--initial-cluster-state new \

--data-dir=/var/lib/etcd

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

8. Reload the daemon configuration.

# systemctl daemon-reload

9. Enable etcd to start at boot time.

# systemctl enable etcd

10. Start etcd.

# systemctl start etcd

11. Verify that the cluster is up and running.

# ETCDCTL\_API=3 etcdctl member list

Perform all the steps on other Master (91 and 92) by replacing IP

### **Initializing the master nodes**

#### **Initializing the Master node 10.1.1.21**

1. SSH to the 10.1.1.21 machine.  
2. Create the configuration file for kubeadm.

# vim config.yaml

apiVersion: kubeadm.k8s.io/v1alpha3

kind: ClusterConfiguration

kubernetesVersion: stable

apiServerCertSANs:

- 10.1.1.11

controlPlaneEndpoint: "10.1.1.11:6443"

etcd:

external:

endpoints:

- https://10.1.1.21:2379

- https://10.1.1.22:2379

- https://10.1.1.23:2379

caFile: /etc/etcd/ca.pem

certFile: /etc/etcd/kubernetes.pem

keyFile: /etc/etcd/kubernetes-key.pem

networking:

podSubnet: 10.2.0.0/24

apiServerExtraArgs:

apiserver-count: "3"

3. Initialize the machine as a master node.

# kubeadm init --config=config.yaml

4. Copy the certificates to the two other masters.

# scp -r /etc/kubernetes/pki ubuntu@10.1.1.22:~  
# scp -r /etc/kubernetes/pki ubuntu@10.1.1.23:~

**Initializing the 2nd and 3rd master node 10.1.1.22/23**

1- SSH to the 10.1.1.22 machine.  
2- Remove the apiserver.crt and apiserver.key.

# rm ~/pki/apiserver.\*

3. Move the certificates to the /etc/kubernetes directory.

# mv ~/pki /etc/kubernetes/

4. Create the configuration file for kubeadm.

# vim config.yaml

apiVersion: kubeadm.k8s.io/v1alpha3

kind: ClusterConfiguration

kubernetesVersion: stable

apiServerCertSANs:

- 10.1.1.11

controlPlaneEndpoint: "10.1.1.11:6443"

etcd:

external:

endpoints:

- https://10.1.1.21:2379

- https://10.1.1.22:2379

- https://10.1.1.23:2379

caFile: /etc/etcd/ca.pem

certFile: /etc/etcd/kubernetes.pem

keyFile: /etc/etcd/kubernetes-key.pem

networking:

podSubnet: 10.2.0.0/24

apiServerExtraArgs:

apiserver-count: "3"

Run the kubeadm join command copied from the Master one with –control-plane

#### **Initializing the worker node 10.1.1.31/32/33**

1. Login each worker and run the kubeadm join command copied from master 1 for workers.

# kubeadm join 10.1.1.21:6443 --token [your\_token] --discovery-token-ca-cert-hash sha256:[your\_token\_ca\_cert\_hash]

Run same command on worker node 32 and 33

#### **Verifying that the workers joined the cluster**

1. SSH to one of the master node  
2. Get the list of the nodes.

# kubectl --kubeconfig /etc/kubernetes/admin.conf get nodes

#### **Configuring kubectl on the client machine**

1. SSH to one of the master node. 10.1.1.21  
2. Add permissions to the admin.conf file.

# chmod +r /etc/kubernetes/admin.conf

3. From the client machine(Should be your local linux box or Haproxy machine), copy the configuration file.

# scp ubuntu@10.1.1.21:/etc/kubernetes/admin.conf .

4. Create and configure the kubectl configuration directory.

# mkdir ~/.kube  
# mv admin.conf ~/.kube/config  
# chmod 600 ~/.kube/config

5. Go back to the SSH session on the master and change back the permissions of the configuration file.

# chmod 600 /etc/kubernetes/admin.conf

6. check that you can access the Kubernetes API from the client machine.

# kubectl get nodes

#### **Deploy overlay network**

We are going to use Calico as the overlay network. You can also use static route or another overlay network tool like Weavenet or Flannel.

1. Deploy the overlay network pods from the client machine.

# kubectl apply -f https://docs.projectcalico.org/archive/v3.21/manifests/calico.yaml

2. Check that the pods are deployed properly.

# kubectl get pods -n kube-system

3. Check that the nodes are in Ready state.

# kubectl get nodes