

# Kubernetes Multi-Master Setup - Manual Installation Guide

## Architecture Overview

This guide sets up a 3-master node Kubernetes cluster for high availability with:

- 3 Master nodes (Control Plane)
- 2+ Worker nodes
- External etcd cluster (optional but recommended)
- Load balancer for API server access

## Prerequisites

### Hardware Requirements

- **Master nodes:** 2 CPU, 4GB RAM, 20GB disk minimum
- **Worker nodes:** 2 CPU, 4GB RAM, 20GB disk minimum
- **Load balancer:** Can be a separate VM or cloud LB

### Network Requirements

- All nodes must have unique hostnames and MAC addresses
- All nodes must be able to communicate on the network
- Disable swap on all nodes
- Ports must be open (detailed below)

## Step 1: Infrastructure Setup

### 1.1 Server Inventory

```
Master-1: 192.168.1.10 (k8s-master-1)
Master-2: 192.168.1.11 (k8s-master-2)
Master-3: 192.168.1.12 (k8s-master-3)
Worker-1: 192.168.1.20 (k8s-worker-1)
Worker-2: 192.168.1.21 (k8s-worker-2)
Load Balancer: 192.168.1.100 (k8s-lb)
```

### 1.2 Load Balancer Configuration

Configure HAProxy or NGINX to balance traffic across master nodes:

HAProxy Configuration (`/etc/haproxy/haproxy.cfg`):

```
frontend k8s-api
  bind *:6443
  mode tcp
  default_backend k8s-masters

backend k8s-masters
  mode tcp
  balance roundrobin
  server master-1 192.168.1.10:6443 check
  server master-2 192.168.1.11:6443 check
  server master-3 192.168.1.12:6443 check
```

## Step 2: System Preparation (All Nodes)

### 2.1 Update System

```
bash

# Ubuntu/Debian
sudo apt update && sudo apt upgrade -y

# CentOS/RHEL
sudo yum update -y
```

### 2.2 Disable Swap

```
bash

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

### 2.3 Configure Hostname Resolution

Edit `/etc/hosts` on all nodes:

```
bash

sudo tee -a /etc/hosts <<EOF
192.168.1.10 k8s-master-1
192.168.1.11 k8s-master-2
192.168.1.12 k8s-master-3
192.168.1.20 k8s-worker-1
192.168.1.21 k8s-worker-2
192.168.1.100 k8s-lb
EOF
```

## 2.4 Enable Required Kernel Modules

```
bash

sudo tee /etc/modules-load.d/k8s.conf <<EOF
br_netfilter
ip_vs
ip_vs_rr
ip_vs_wrr
ip_vs_sh
nf_conntrack_ipv4
EOF

sudo modprobe br_netfilter
sudo modprobe ip_vs
sudo modprobe ip_vs_rr
sudo modprobe ip_vs_wrr
sudo modprobe ip_vs_sh
sudo modprobe nf_conntrack_ipv4
```

## 2.5 Configure Sysctl

```
bash

sudo tee /etc/sysctl.d/k8s.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
```

## 2.6 Configure Firewall Ports

### Master Nodes:

```
bash
```

```
# API Server
```

```
sudo ufw allow 6443/tcp
```

```
# etcd
```

```
sudo ufw allow 2379:2380/tcp
```

```
# Kubelet API
```

```
sudo ufw allow 10250/tcp
```

```
# kube-scheduler
```

```
sudo ufw allow 10259/tcp
```

```
# kube-controller-manager
```

```
sudo ufw allow 10257/tcp
```

## Worker Nodes:

```
bash
```

```
# Kubelet API
```

```
sudo ufw allow 10250/tcp
```

```
# NodePort Services
```

```
sudo ufw allow 30000:32767/tcp
```

## Step 3: Container Runtime Installation

### 3.1 Install containerd (All Nodes)

```
bash
```

```
# Install dependencies
```

```
sudo apt-get update
```

```
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release
```

```
# Add Docker's official GPG key
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-arc
```

```
# Add Docker repository
```

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com
```

```
# Install containerd
```

```
sudo apt-get update
```

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

```
# Configure containerd
```

```
sudo mkdir -p /etc/containerd
```

```
sudo containerd config default | sudo tee /etc/containerd/config.toml
```

```
# Enable SystemdCgroup
```

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

```
# Start and enable containerd
```

```
sudo systemctl restart containerd
```

```
sudo systemctl enable containerd
```

## Step 4: Kubernetes Components Installation

### 4.1 Install kubeadm, kubelet, kubectl (All Nodes)

```
bash
```

```
# Add Kubernetes repository
```

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

```
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
# Install Kubernetes components
```

```
sudo apt-get update
```

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

```
# Hold packages to prevent automatic updates
```

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

```
# Enable kubelet
```

```
sudo systemctl enable kubelet
```

## Step 5: Initialize First Master Node

### 5.1 Create kubeadm Config File

Create `/root/kubeadm-config.yaml` on Master-1:

```
yaml

apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: v1.28.0
controlPlaneEndpoint: "k8s-lb:6443"
networking:
  serviceSubnet: "10.96.0.0/16"
  podSubnet: "10.244.0.0/16"
etcd:
  local:
    dataDir: "/var/lib/etcd"
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: "192.168.1.10"
  bindPort: 6443
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: JoinConfiguration
discovery:
  bootstrapToken:
    apiServerEndpoint: "k8s-lb:6443"
    token: ""
    unsafeSkipCAVerification: true
controlPlane:
  localAPIEndpoint:
    advertiseAddress: ""
    bindPort: 6443
```

### 5.2 Initialize Cluster

```
bash

sudo kubeadm init --config=/root/kubeadm-config.yaml --upload-certs
```

### 5.3 Configure kubectl for root user

```
bash
```

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

## 5.4 Save Join Commands

Save the output from kubeadm init - you'll need:

- Master join command with certificate key
- Worker join command

Example:

```
bash

# Master join command
kubeadm join k8s-lb:6443 --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash> \
  --control-plane --certificate-key <cert-key>

# Worker join command
kubeadm join k8s-lb:6443 --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash>
```

## Step 6: Join Additional Master Nodes

### 6.1 Join Master-2

On Master-2 node:

```
bash

sudo kubeadm join k8s-lb:6443 --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash> \
  --control-plane --certificate-key <cert-key>

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

### 6.2 Join Master-3

Repeat the same process on Master-3.

## Step 7: Install CNI Plugin

### 7.1 Install Flannel (from Master-1)

```
bash

kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

### 7.2 Verify Master Nodes

```
bash

kubectl get nodes -o wide
kubectl get pods -n kube-system
```

## Step 8: Join Worker Nodes

### 8.1 Join Worker Nodes

On each worker node:

```
bash

sudo kubeadm join k8s-lb:6443 --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash>
```

### 8.2 Label Worker Nodes (from Master)

```
bash

kubectl label node k8s-worker-1 node-role.kubernetes.io/worker=worker
kubectl label node k8s-worker-2 node-role.kubernetes.io/worker=worker
```

## Step 9: Verification and Testing

### 9.1 Verify Cluster Status

```
bash
```



```
# Check all nodes
```

```
kubectl get nodes
```

```
# Check system pods
```

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

```
# Check cluster info
```

```
kubectl cluster-info
```

```
# Check component status
```

```
kubectl get cs
```

## 9.2 Test Pod Scheduling

```
bash
```

```
# Create test deployment
```

```
kubectl create deployment test-nginx --image=nginx --replicas=3
```

```
# Check pod distribution
```

```
kubectl get pods -o wide
```

```
# Clean up
```

```
kubectl delete deployment test-nginx
```

## Step 10: Configure High Availability Testing

### 10.1 Test API Server HA

```
bash
```

```
# Test from different masters
```

```
kubectl --server=https://192.168.1.10:6443 get nodes
```

```
kubectl --server=https://192.168.1.11:6443 get nodes
```

```
kubectl --server=https://192.168.1.12:6443 get nodes
```

### 10.2 Test Master Node Failure

Stop one master node and verify cluster continues to function:

```
bash
```

```
# On one master node
sudo systemctl stop kubelet
```

```
# From another master
kubectl get nodes
```

## Troubleshooting Common Issues

### Issue 1: Certificate Errors

```
bash

# Regenerate certificates if needed
sudo kubeadm init phase upload-certs --upload-certs
```

### Issue 2: Token Expiration

```
bash

# Generate new token
kubeadm token create --print-join-command
```

### Issue 3: etcd Issues

```
bash

# Check etcd health
kubectl get pods -n kube-system | grep etcd
kubectl logs -n kube-system etcd-k8s-master-1
```

### Issue 4: Network Issues

```
bash

# Check CNI pods
kubectl get pods -n kube-system | grep flannel
kubectl logs -n kube-system ds/kube-flannel-ds
```

## Maintenance Commands

### Backup etcd

```
bash
```

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=/etc/kubernetes/pki/etcd/ca.crt \  
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \  
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key \  
snapshot save /backup/etcd-snapshot-$(date +%Y%m%d_%H%M%S).db
```

## Upgrade Cluster

```
bash  
  
# Plan upgrade  
sudo kubeadm upgrade plan  
  
# Upgrade first master  
sudo kubeadm upgrade apply v1.28.1  
  
# Upgrade other masters  
sudo kubeadm upgrade node
```

## Security Hardening

### Enable RBAC (Already enabled by default)

```
bash  
  
kubectl get clusterrolebinding
```

## Network Policies

```
bash  
  
# Example network policy  
kubectl apply -f - <<EOF  
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: default-deny-ingress  
spec:  
  podSelector: {}  
  policyTypes:  
  - Ingress  
EOF
```

This completes the manual setup of a highly available Kubernetes cluster with multiple master nodes. The cluster is now ready for production workloads with built-in redundancy and failover capabilities.

