# Kubernetes Cluster Setup Guide - Create Your Own K8s Cluster

## Overview

This guide will help you create a production-ready Kubernetes cluster using **kubeadm** with **containerd** as the container runtime. We'll set up a multi-node cluster with proper networking and security.

## Prerequisites

### System Requirements

- **Minimum 2 GB RAM** per node (4 GB recommended)
- **2 CPUs** per node
- **Network connectivity** between all nodes
- **Unique hostname, MAC address, and product_uuid** for every node

### Supported Operating Systems

- Ubuntu 16.04+ (Recommended: Ubuntu 20.04/22.04)
- CentOS 7+
- RHEL 7+
- Debian 9+

---

## Phase 1: Environment Setup (All Nodes)

### Step 1: Update System & Install Dependencies

```bash
# Update package index
sudo apt update && sudo apt upgrade -y

# Install required packages
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

# Install additional tools
sudo apt install -y wget gpg lsb-release
```

### Step 2: Disable Swap (Critical for Kubernetes)

bash

```
# Disable swap immediately
sudo swapoff -a

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

# Verify swap is disabled
free -h
```

## Step 3: Configure Kernel Modules & Networking

bash

```
# Load required kernel modules
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# Configure sysctl parameters
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl parameters
sudo sysctl --system
```

## Step 4: Configure Firewall (If Enabled)

bash

```bash
# For Ubuntu/Debian with ufw
sudo ufw allow 6443/tcp    # Kubernetes API server
sudo ufw allow 2379:2380/tcp  # etcd server client API
sudo ufw allow 10250/tcp   # Kubelet API
sudo ufw allow 10259/tcp   # kube-scheduler
sudo ufw allow 10257/tcp   # kube-controller-manager
sudo ufw allow 30000:32767/tcp  # NodePort Services

# For worker nodes, also allow:
sudo ufw allow 10250/tcp   # Kubelet API
sudo ufw allow 30000:32767/tcp  # NodePort Services
```

---

## Phase 2: Install Container Runtime (containerd)

### Step 5: Install containerd

bash

```bash
# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyr

# Add Docker repository
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyri

# Update package index
sudo apt update

# Install containerd
sudo apt install -y containerd.io
```

### Step 6: Configure containerd

bash

```bash
# Create containerd configuration directory
sudo mkdir -p /etc/containerd

# Generate default configuration
sudo containerd config default | sudo tee /etc/containerd/config.toml

# Enable SystemdCgroup (Important for kubeadm)
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml

# Restart and enable containerd
sudo systemctl restart containerd
sudo systemctl enable containerd

# Verify containerd is running
sudo systemctl status containerd
```

---

## Phase 3: Install Kubernetes Components

### Step 7: Add Kubernetes Repository

bash

```bash
# Add Kubernetes GPG key
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc

# Add Kubernetes repository
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/st

# Update package index
sudo apt update
```

### Step 8: Install Kubernetes Tools

bash

```bash
# Install kubelet, kubeadm, and kubectl
sudo apt install -y kubelet kubeadm kubectl

# Hold packages to prevent automatic updates
sudo apt-mark hold kubelet kubeadm kubectl

# Enable kubelet service
sudo systemctl enable kubelet
```

---

## Phase 4: Initialize Master/Control Plane Node

### Step 9: Initialize Kubernetes Cluster (Master Node Only)

bash

```bash
# Initialize the cluster
sudo kubeadm init \
  --pod-network-cidr=192.168.0.0/16 \
  --kubernetes-version=stable \
  --node-name=$(hostname -s)

# IMPORTANT: Save the join command output for worker nodes!
```

### Step 10: Configure kubectl (Master Node Only)

bash

```bash
# Set up kubectl for 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

# Verify cluster status
kubectl get nodes
kubectl get pods -A
```

### Step 11: Install Pod Network Add-on (Master Node Only)

bash

```
# Install Calico CNI (Recommended)
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.1/manifests/tige

# Download and apply Calico custom resources
curl https://raw.githubusercontent.com/projectcalico/calico/v3.28.1/manifests/custom-resources.

# Apply Calico configuration
kubectl create -f custom-resources.yaml

# Wait for all pods to be ready
kubectl get pods -n calico-system --watch
```

**Alternative CNI Options:**

bash

```
# Flannel (Simpler option)
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.ym

# Weave Net
kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k
```

---

# Phase 5: Join Worker Nodes

## Step 12: Join Worker Nodes to Cluster

bash

```
# On each worker node, run the join command from kubeadm init output:
sudo kubeadm join <MASTER-IP>:6443 \
  --token <TOKEN> \
  --discovery-token-ca-cert-hash sha256:<HASH>

# If you lost the join command, generate a new one on master:
kubeadm token create --print-join-command
```

## Step 13: Verify Cluster Setup

```bash
# On master node, check all nodes
kubectl get nodes -o wide

# Check system pods
kubectl get pods -n kube-system

# Check cluster info
kubectl cluster-info
```

---

## Phase 6: Post-Installation Configuration

### Step 14: Remove Taint from Master (Optional - for single node)

```bash
# Allow scheduling pods on master node (for development/testing)
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

### Step 15: Install Kubernetes Dashboard (Optional)

bash

```bash
# Deploy Dashboard
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recom

# Create admin user
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
EOF

# Get access token
kubectl -n kubernetes-dashboard create token admin-user

# Access dashboard
kubectl proxy
# Visit: http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes
```

## Step 16: Install Helm (Package Manager)

bash

```bash
# Install Helm
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://ba
sudo apt update
sudo apt install helm

# Verify Helm installation
helm version
```

---

## Testing Your Cluster

### Step 17: Deploy Test Application

bash

```bash
# Create a test deployment
kubectl create deployment nginx --image=nginx:latest

# Expose the deployment
kubectl expose deployment nginx --port=80 --type=NodePort

# Get service details
kubectl get services

# Test the application
curl http://<NODE-IP>:<NODE-PORT>
```

---

## Cluster Management Commands

### Essential kubectl Commands

```bash
# Node management
kubectl get nodes
kubectl describe node <NODE-NAME>
kubectl drain <NODE-NAME> --ignore-daemonsets
kubectl uncordon <NODE-NAME>

# Pod management
kubectl get pods -A
kubectl logs <POD-NAME> -n <NAMESPACE>
kubectl exec -it <POD-NAME> -- /bin/bash

# Cluster health
kubectl get componentstatuses
kubectl top nodes
kubectl top pods -A
```

### Cluster Maintenance

```bash
# Backup etcd (Important!)
ETCDCTL_API=3 etcdctl snapshot save backup.db \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key

# Upgrade cluster
sudo kubeadm upgrade plan
sudo kubeadm upgrade apply v1.31.x
```

## Troubleshooting

### Common Issues & Solutions

#### 1. Node Not Ready

bash

```bash
# Check kubelet logs
sudo journalctl -u kubelet -f

# Check node conditions
kubectl describe node <NODE-NAME>
```

### 2. Pods Stuck in Pending

bash

```bash
# Check pod events
kubectl describe pod <POD-NAME>

# Check resource availability
kubectl top nodes
```

### 3. CNI Issues

bash

```bash
# Restart CNI pods
kubectl delete pods -n kube-system -l k8s-app=calico-node

# Check CNI logs
kubectl logs -n kube-system -l k8s-app=calico-node
```

### 4. Reset Cluster (if needed)

bash

```bash
# Reset kubeadm
sudo kubeadm reset
sudo rm -rf /etc/cni/net.d
sudo rm -rf $HOME/.kube/config
```

---

## Security Considerations

### Production Hardening

- **Enable RBAC** (enabled by default in kubeadm)
- **Use Network Policies** for pod-to-pod communication

- **Regular security updates** for all components
- **Proper secret management** with tools like Sealed Secrets or External Secrets
- **Enable audit logging**
- **Use Pod Security Standards**

### Resource Limits

```yaml
# Example resource limits
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
spec:
  hard:
    requests.cpu: "1000m"
    requests.memory: 1Gi
    limits.cpu: "2000m"
    limits.memory: 2Gi
```

## Next Steps

1. **Set up monitoring** with Prometheus & Grafana
2. **Configure logging** with ELK stack or Loki
3. **Implement CI/CD** pipelines
4. **Set up backup strategies** for etcd and persistent volumes
5. **Plan for high availability** with multiple master nodes
6. **Implement security policies** and network segmentation

---

**Congratulations!** 🎉 You now have a fully functional Kubernetes cluster running with containerd. Your cluster is ready for deploying applications and learning advanced Kubernetes concepts.