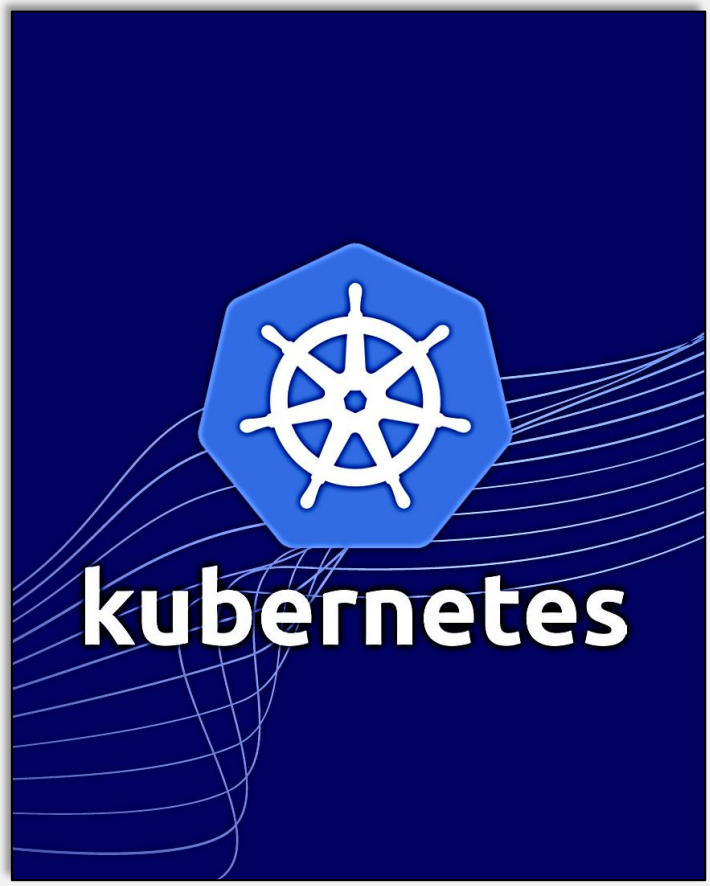


# K8S Guide:

## SETTING UP A KUBERNETES CLUSTER ON UBUNTU

**By : Oussama El Ouazdi**

# Comprehensive Guide to Setting Up a Kubernetes Cluster on Ubuntu

## Part 1: Introduction, System Requirements, and Prerequisites

### Introduction

Setting up a Kubernetes cluster on Ubuntu involves several key steps, including installing container runtimes, Kubernetes core components, and a network plugin. This guide provides a comprehensive approach to setting up a single-master, multi-node Kubernetes cluster. We use kubeadm, a tool to simplify Kubernetes cluster setup, and Docker as the container runtime.

This setup guide is ideal for students, professionals, and cloud practitioners who want to experiment with Kubernetes clusters in a development or testing environment. Follow along to create a fully functional Kubernetes cluster from scratch, ensuring that all essential components are configured for smooth operation.

### System Requirements

Before starting, ensure that each machine in your Kubernetes cluster meets the following requirements :

- **Operating System** : Ubuntu 20.04 or newer is generally recommended. Ensure that the operating system is updated to avoid compatibility issues.
- **CPU :**
  - Master Node: Minimum 2 cores (4 cores recommended for better performance).
  - Worker Nodes: Minimum 1 core (2 cores recommended).
- **RAM :**
  - Master Node: Minimum 2 GB (4 GB or more recommended).
  - Worker Nodes: Minimum 1 GB (2 GB or more recommended).
- **Storage :** Minimum 10 GB per node (20 GB or more recommended for production workloads).
- **Network :**
  - Each node must be able to communicate with each other over the network.
  - Recommended to disable swap, as Kubernetes does not support swap.

### Prerequisites

- **User Permissions :** Root or sudo access is required on all nodes.

- **Firewall Rules :** Ensure the following ports are open on the nodes :
  - **Master Node :**
    - Port 6443 (Kubernetes API server)
    - Port 2379-2380 (etcd server client API)
    - Port 10250, 10251, 10252 (Kubelet, Kube-Scheduler, Kube-Controller-Manager)
  - **Worker Node :**
    - Port 10250 (Kubelet API)
    - Port 30000-32767 (NodePort Services)
- **Disable Swap :** Kubernetes requires swap to be disabled for proper memory management.

  sudo swapoff -a
- **Update and Upgrade :** Run an update and upgrade on each node to ensure all packages are up-to-date.

  sudo apt update && sudo apt upgrade -y

With these prerequisites in place, we're ready to move to Part 2: Uninstalling and Cleaning Previous Installations.

## Part 2 : Uninstalling and Cleaning Previous Installations

Before proceeding, ensure that any existing Kubernetes or Docker installations are fully removed to avoid conflicts during setup. This section covers how to reset `kubeadm` and clean up any leftover files from previous installations.

### Step 1 : Reset `kubeadm` and Clean Up Kubernetes Components

If you've previously attempted a Kubernetes installation on any of the nodes, reset `kubeadm` and delete related files to start with a clean slate.

sudo kubeadm reset -f

After running the above command, remove Kubernetes-related directories and configurations.

sudo rm -rf /etc/cni/net.d

sudo iptables -F

sudo iptables -t nat -F

sudo iptables -t mangle -F

sudo iptables -X

sudo ipvsadm --clear

sudo rm -rf /etc/kubernetes

sudo rm -rf /var/lib/etcd

```
sudo rm -rf /var/lib/kubelet
sudo rm -rf /etc/kubernetes /var/lib/kubelet /var/lib/etcd /etc/cni /opt/cni /var/lib/cni
```

### Step 2 : Remove Docker and Container Runtime Files

If Docker, containerd, or other container runtimes are installed, purge them as well to avoid conflicts with new installations.

```
sudo apt-get purge -y kubeadm kubectl kubelet kubernetes-cni kube*
sudo apt-get purge -y containerd docker-ce docker-ce-cli containerd.io
sudo rm -rf /var/lib/docker /etc/docker /etc/containerd /var/lib/containerd
```

### Step 3 : Remove Network Interfaces Created by Previous Installations

Kubernetes often creates additional network interfaces (e.g., `cni0`, `flannel.1`). To ensure a fresh start, delete these interfaces if they exist :

```
sudo ip link delete cni0
sudo ip link delete flannel.1
```

### Step 4 : Autoremove and Clean Up Residual Files

Use the `autoremove` and `autoclean` commands to free up space and ensure no unwanted dependencies are left behind.

```
sudo apt-get autoremove -y sudo apt-get autoclean -y
```

### Step 5 : Reload Daemon and Remove Kubernetes Service Files

To finalize the cleanup, reload the systemd daemon and remove any remaining Kubernetes service files :

```
sudo rm -rf /etc/systemd/system/kubelet.service.d
sudo rm -rf /etc/systemd/system/kubelet.service
sudo systemctl daemon-reload
```

With this, the nodes are now cleared of any previous Kubernetes or Docker setups, and you can proceed to the next step of installing Docker and container runtimes.

## Part 3 : Installing Docker and Containerd as the Container Runtime

For Kubernetes clusters, containerd and Docker are common container runtimes. In this guide, we'll use containerd configured with Docker.

Now that your system is clean, let's proceed to install Docker and configure containerd, a lightweight runtime that Kubernetes commonly uses to manage containerized applications.

### Step 1 : Install Docker

Docker provides the necessary tooling and resources for container management. Run the following commands to install Docker :

```
sudo apt update
sudo apt install -y docker.io
```

Enable Docker to start on boot and start the Docker service :

```
sudo systemctl enable docker
sudo systemctl start docker
```

### Step 2 : Install Containerd

Containerd is the container runtime that will work behind Docker to run and manage containers in the Kubernetes cluster.

- **Download the latest containerd release** from GitHub. Replace the filename below with the latest version.

  ```
  cd ~/Downloads
  wget https://github.com/containerd/containerd/releases/download/v1.7.23/containerd-1.7.23-linux-amd64.tar.gz
  ```

- **Extract and install containerd :**

  ```
  sudo tar Cxzvf /usr/local containerd-1.7.23-linux-amd64.tar.gz
  ```

### Step 3 : Install Runc

Runc is a low-level container runtime used by containerd to run containers.

- **Download the latest `runc` binary from GitHub :**

  ```
  wget https://github.com/opencontainers/runc/releases/download/v1.1.8/runc.amd64
  ```

- **Move the binary to the appropriate directory and set permissions :**

  ```
  sudo install -m 755 runc.amd64 /usr/local/sbin/runc
  ```

### Step 4 : Install CNI Plugins

Container Network Interface (CNI) plugins are necessary for Kubernetes to manage networking in containers. Follow these steps to install the CNI plugins :

- **Extract the plugins to the CNI directory :**

```
sudo mkdir -p /opt/cni/bin
sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.6.0.tgz
```

- **Download the CNI plugins archive :**

```
wget https://github.com/containernetworking/plugins/releases/download/v1.6.0/cni-plugins-linux-amd64-v1.6.0.tgz
```

### Step 5 : Configure Containerd

Containerd requires a configuration file for runtime settings. Generate and configure the default settings with the following steps :

- **Create the configuration file :**

```
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
```

- **Enable Systemd as the Cgroup Driver (recommended for Kubernetes) :**

Edit the containerd configuration to use `SystemdCgroup`. This change aligns with Kubernetes' use of `systemd` for managing resources.

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

- **Download the containerd service file and enable the containerd service :**

```
sudo curl -L https://raw.githubusercontent.com/containerd/containerd/main/containerd.service -o /etc/systemd/system/containerd.service
sudo systemctl daemon-reload
sudo systemctl enable --now containerd
```

- **Check the containerd status to ensure it's running :**

```
sudo systemctl status containerd
```

With Docker and containerd configured, we're ready to install the Kubernetes components (`kubelet`, `kubeadm`, and `kubectl`), which we'll cover in Part 4.

## Part 4 : Installing Kubernetes Components (kubelet, kubeadm, and kubectl)

With Docker and containerd installed, you can now set up the essential Kubernetes components on each node: `kubelet`, `kubeadm`, and `kubectl`. These tools will enable you to manage and orchestrate your cluster.

### Step 1 : Install Prerequisite Packages

Install essential packages required for downloading Kubernetes components and setting up IP-based virtual server management.

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg ipvsadm
```

### Step 2 : Add the Kubernetes GPG Key

To securely install Kubernetes packages, add Google's GPG key:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

### Step 3 : Add the Kubernetes Repository

Add the Kubernetes repository to your system's sources list :

```
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
```

### Step 4 : Install kubelet, kubeadm, and kubectl

Now, update your package list and install the Kubernetes components :

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

To prevent these packages from being accidentally updated, hold them at their current versions :

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

### Step 5 : Enable and Start kubelet

Enable the `kubelet` service to start automatically and ensure it's running :

```
sudo systemctl enable --now kubelet
```

With Kubernetes components installed, you're now ready to initialize the Kubernetes cluster on the master node, which we'll cover in Part 5.

## Part 5 : Initializing the Kubernetes Cluster (Master Node Setup)

In this section, we'll configure the master node and initialize the Kubernetes control plane.

The master node controls the cluster and orchestrates tasks among the worker nodes. In this step, you'll initialize the control plane on the master node, configure kubeconfig, and set up a networking plugin for communication between nodes.

### Step 1 : Initialize the Kubernetes Control Plane

To initialize the Kubernetes master node, run the following command on the master node:

`sudo kubeadm init`

This command will :

1. Set up the Kubernetes control plane on the master node.
2. Generate a kubeadm join command that worker nodes will use to connect to this master node. **Make sure to save this join command, as you'll need it later.**

**Note**: If your master node has limited resources, you can specify a pod network CIDR, for example --pod-network-cidr=192.168.0.0/16, to ensure compatibility with the network plugin you'll install later (e.g., Calico).

### Step 2 : Configure kubectl for the Master Node

To interact with your cluster from the command line, configure `kubectl` on the master node :

- **Create the `.kube` directory :**
  `mkdir -p $HOME/.kube`
- **Copy the kubeconfig file (which stores cluster configuration) to this directory :**
  `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
- **Change ownership of the kubeconfig file to the current user :**
  `sudo chown $(id -u):$(id -g) $HOME/.kube/config`

At this point, you should be able to run `kubectl` commands and see the master node status.

### Step 3 : Install a Pod Network Add-On

Kubernetes requires a network add-on to enable communication between pods across nodes. Calico is a popular choice, and it's compatible with Kubernetes' default configurations.

- **Apply the Calico network manifest :**
  `kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.29.0/manifests/calico.yaml`
- **Verify the Node and Pod Status :**
  It may take a few moments for the network plugin to initialize. Check the status of nodes and pods with the following commands :
  `kubectl get nodes`
  `kubectl get pods --all-namespaces`
  Ensure that all nodes are in a "Ready" state, and that Calico pods are running correctly.

With the master node initialized and the network plugin installed, you're ready to join worker nodes to the cluster, which we'll cover in Part 6.

## Part 6 : Adding Worker Nodes to the Kubernetes Cluster

With the master node set up, it's time to connect the worker nodes to your cluster. Each worker node will run Kubernetes workloads and communicate with the master node to receive instructions.

### Step 1: Retrieve the Join Command from the Master Node

During the `kubeadm init` process on the master node, you received a `kubeadm join` command, which will look something like this :

kubeadm join <master-node-ip>:6443 --token <your-token> --discovery-token-ca-cert-hash sha256:<hash>

If you didn't save the join command, you can generate a new one by running the following command on the master node :

kubeadm token create --print-join-command

### Step 2: Run the Join Command on Each Worker Node

On each worker node, execute the `kubeadm join` command you obtained. This will connect the worker node to the master node and integrate it into the cluster.

Example :

sudo kubeadm join <master-node-ip>:6443 --token <your-token> --discovery-token-ca-cert-hash sha256:<hash>

### Step 3: Verify the Worker Nodes Are Connected

Once you've run the join command on each worker node, return to the master node and verify that the worker nodes are successfully added to the cluster.

kubectl get nodes

You should see all the nodes listed, with each in a "Ready" state. If any nodes show a different status, they may still be initializing or have a connection issue.

## Part 7: Verifying Cluster Health and Troubleshooting Common Issues

After adding all nodes to the cluster, it's important to confirm that everything is running smoothly. Here's a list of commands to verify the cluster's health and troubleshoot common issues if necessary.

### Step 1: Check the Status of All Nodes and Pods

On the master node, run the following commands to check the status of nodes and pods across all namespaces :

kubectl get nodes

kubectl get pods --all-namespaces

### Step 2: Common Issues and Solutions

- **Node Not Ready :**

  Check if `kubelet` is running on each node:

  sudo systemctl status kubelet

  Ensure that all required ports are open (firewall settings may need to be adjusted).

- **Network Plugin Issues :**

  Verify that the network plugin (e.g., Calico) pods are running properly.

  Restart the networking plugin if needed by reapplying the manifest :

  kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.29.0/manifests/calico.yaml

- **kubeadm join Token Expiry :**

  If the join token has expired, create a new token on the master node :

  kubeadm token create --print-join-command

- **Containerd Issues :**

  If containerd is not starting properly, ensure `SystemdCgroup` is set to `true` in `/etc/containerd/config.toml`.

  Restart containerd to apply changes :

  sudo systemctl restart containerd

## Conclusion

Congratulations! You've successfully set up a multi-node Kubernetes cluster with Docker, containerd, and Calico networking. This guide provides a solid foundation for experimenting with Kubernetes clusters and learning how to manage workloads in a distributed environment.