Kubernetes Installation Guide

Complete Setup on Ubuntu 24.04

Using kubeadm, kubelet, kubectl, containerd & Calico CNI

Kubernetes v1.31

Introduction

This document provides a comprehensive step-by-step guide to install and configure a production-ready Kubernetes cluster on Ubuntu 24.04. The installation utilizes modern containerization technologies and industry-standard tools to ensure reliability, scalability, and ease of management.

What's Included

- kubeadm - Kubernetes cluster bootstrapping tool
- kubelet - Primary node agent that runs on each cluster node
- kubectl - Command-line tool for controlling Kubernetes clusters
- containerd - Industry-standard container runtime
- Calico CNI - Network plugin for pod networking

This Setup is Suitable For

- DevOps laboratory environments
- Vagrant-based development environments
- On-premises production clusters
- Learning and training Kubernetes
- Production-like testing and staging environments

Cluster Topology

The following table outlines the example cluster architecture used in this guide. You can customize node names and IP addresses according to your environment requirements.

Role	Hostname	Example IP Address
Master	k8s-master-node	192.168.56.10
Worker	k8s-worker-node	192.168.56.11

Installation Steps

Step 1 — Configure Hostnames (All Nodes)

Proper hostname configuration ensures that nodes can communicate with each other using meaningful names instead of IP addresses. This step must be performed on all nodes (master and workers).

Edit the Hosts File

Open the hosts file with elevated privileges:

```
- sudo nano /etc/hosts
```

Add the following entries to map hostnames to IP addresses:

```
192.168.56.10 k8s-master-node
192.168.56.11 k8s-worker-node
Set Hostname on Each Node
```

On the Master Node:

```
- sudo hostnamectl set-hostname k8s-master-node
```

On Worker Node 1:

```
- sudo hostnamectl set-hostname k8s-worker-node
```

Verify the hostname change:

```
- hostname
```

Test Network Connectivity

From the master node, test connectivity to worker nodes:

```
- ping k8s-master-node
- ping k8s-worker-node
```

Step 2 — Disable Swap (All Nodes)

Kubernetes requires swap to be disabled for optimal performance and to ensure proper resource management. Swap memory can interfere with Kubernetes pod scheduling and memory limits.

Temporarily Disable Swap

Immediately disable swap for the current session:

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

Permanently Disable Swap

To ensure swap remains disabled after reboot, edit the fstab file:

```
- sudo nano /etc/fstab
```

Comment out any line containing 'swap' by adding a # at the beginning:

```
# /swapfile none swap sw 0 0
```

Verify Swap is Disabled

Check that swap is completely disabled:

```
- swapon --show
```

Note: No output indicates that swap is successfully disabled.

Step 3 — Install Docker and Containerd (All Nodes)

Containerd is the container runtime that Kubernetes uses to run containers. While Docker includes containerd, we'll configure containerd specifically for Kubernetes use.

Install Docker

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

Enable and Start Docker Service

```
- sudo systemctl enable dockersudo systemctl start docker
```

Configure Containerd

Create the containerd configuration directory:

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

Generate and save the default configuration:

```
- sudo sh -c "containerd config default > /etc/containerd/config.toml"
```

Enable SystemdCgroup

Kubernetes requires SystemdCgroup to be enabled. Update the configuration using sed:

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

Restart and Verify Containerd

```
- sudo systemctl restart containerd
- sudo systemctl status containerd
```

Step 4 — Install Kubernetes Components (All Nodes)

This step installs the core Kubernetes tools: kubeadm for cluster bootstrapping, kubelet as the node agent, and kubectl for cluster management.

Install Required Dependencies

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

Add Kubernetes GPG Key

Download and install the Kubernetes package signing key:

```
-   curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg
    --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Add Kubernetes Repository

```
-   echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
    https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /" | sudo tee
    /etc/apt/sources.list.d/kubernetes.list
```

Install Kubernetes Tools

Update package index and install kubeadm, kubelet, and kubectl:

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

Step 5 — Initialize the Master Node

This step initializes the Kubernetes control plane on the master node. The pod network CIDR must match what you'll configure in the CNI plugin (Calico in this case).

Initialize Kubernetes Cluster

Run kubeadm init with the pod network CIDR. This process may take several minutes:

```
-   sudo kubeadm init --pod-network-cidr=10.10.0.0/16
```

Important: Save the kubeadm join command output at the end of initialization. You will need it to join worker nodes to the cluster.

Configure kubectl for Current User

Set up kubectl access for the non-root user:

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

Verify Master Node Status

Check that the master node is visible (it will show as NotReady until CNI is installed):

```
- kubectl get nodes
```

Step 6 — Install Calico CNI Plugin (Master Node)

Calico provides networking and network policy for Kubernetes. Installing the CNI plugin will enable pod-to-pod communication across the cluster.

Install Tigera Operator

Deploy the Calico operator:

```
- kubectl create -f
  https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/tigera-o
  perator.yaml
```

Download Custom Resources Manifest

```
- curl
  https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/custom-r
  esources.yaml -O
```

Update CIDR to Match Cluster Configuration

Modify the CIDR in the manifest to match the pod network CIDR used during cluster initialization:

```
- sed -i 's/cidr: 192\.168\.0\.0\/16/cidr: 10.10.0.0\/16/g' custom-resources.yaml
```

Apply Calico Configuration

```
- kubectl create -f custom-resources.yaml
```

Wait a few minutes for Calico pods to start. After installation, your master node should show as Ready.

Step 7 — Join Worker Nodes to the Cluster

Now that the master node is configured and the CNI is installed, you can join worker nodes to the cluster using the join command generated during kubeadm init.

Execute Join Command on Worker Nodes

On each worker node, run the join command with sudo. The command will look similar to this:

```
- sudo kubeadm join <MASTER-IP>:6443 --token <TOKEN>
  --discovery-token-ca-cert-hash sha256:<HASH>
```

Replace <MASTER-IP>, <TOKEN>, and <HASH> with the actual values from your kubeadm init output.

Verify Worker Nodes Have Joined

On the master node, verify that all nodes are in the Ready state:

```
- kubectl get nodes
```

All nodes should show STATUS as Ready. If any node shows NotReady, wait a minute and check again as pods may still be initializing.

Step 8 — Test the Cluster

Verify that your Kubernetes cluster is functioning correctly by deploying a test application and exposing it as a service.

Create a Test Namespace

```
- kubectl create namespace demo-namespace
```

Deploy Nginx Application

Create a deployment with 2 replicas of nginx:

```
- kubectl create deployment my-app --image nginx --replicas 2 -n demo-namespace
```

Expose the Deployment as a Service

```
- kubectl expose deployment my-app -n demo-namespace --type NodePort --port 80
```

Check Service Details

Get the NodePort assigned to the service:

```
- kubectl get svc -n demo-namespace
```

Test Application Access

Use curl to access the nginx service through any worker node IP and the NodePort:

- curl http://<worker-ip>:<nodeport>

You should see the default nginx welcome page HTML output, confirming that your cluster is operational.

Useful Kubernetes Commands

The following commands are essential for day-to-day Kubernetes cluster management and troubleshooting.

Cluster and Node Management

Command	Description
kubectl get nodes	List all nodes in the cluster
kubectl get pods -A	List all pods in all namespaces
kubectl get svc	List all services

Pod Management and Troubleshooting

Command	Description
kubectl logs <pod>	View logs from a specific pod
kubectl describe pod <pod>	Show detailed information about a pod
kubectl delete -f manifest.yaml	Delete resources defined in a manifest file

frontend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: frontend
    tier: ui
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
        tier: ui
    spec:
      containers:
      - name: frontend
        # Image updated automatically via CI/CD
        image: awaisumarhayatofficial/frontend:latest
        imagePullPolicy: Always
        ports:
        - name: http-port
          containerPort: 80
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "200m"
            memory: "256Mi"
```

frontend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: frontend
    tier: ui
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - name: http
```

```
      protocol: TCP
      port: 80          # Port inside the cluster
      targetPort: 80    # Port on the container
      nodePort: 30080   # External port for browser access
```

backend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  labels:
    app: backend
    tier: api
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
        tier: api
    spec:
      containers:
      - name: backend
        image: awaisumarhayatofficial/backend:latest
        imagePullPolicy: Always
        ports:
        - name: api-port
          containerPort: 80
        env:
        - name: REDIS_HOST
          value: "redis" # ClusterIP service name of Redis
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "250m"
            memory: "512Mi"
```

backend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: backend
  labels:
    app: backend
    tier: api
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - name: http
      protocol: TCP
      port: 8001        # Internal Cluster Port
      targetPort: 80    # Application Port inside Container
      nodePort: 30001   # External Port for Access
```

Dockerfile (Frontend Application Dockerize)

```
FROM ubuntu/apache2:2.4-20.04_beta

# Remove default files as root
RUN rm -rf /var/www/html/*

# Copy website files as root
COPY ./html/ /var/www/html/

# Create non-root user
#RUN useradd -m WebUI

# Give WebUI ownership of Apache directories
#RUN mkdir -p /var/run/apache2 /var/lock/apache2 \
#    && chown -R WebUI:WebUI /var/www/html /var/run/apache2
/var/lock/apache2

# Switch to non-root
#USER WebUI
WORKDIR /var/www/html
EXPOSE 80
CMD ["apache2-foreground"]
```

Dockerfile (Backend Application Dockerize)

```
# 1. PHP 8.2 with Apache
FROM php:8.2-apache

# 2. Install essential system dependencies
RUN apt-get update && apt-get install -y \
    libpng-dev \
    zip \
    unzip \
    git \
    && docker-php-ext-install pdo_mysql

# 3. Enable Apache Rewrite Module (Laravel routes ke liye zaroori hai)
RUN a2enmod rewrite

# 4. Get latest Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# 5. Set working directory
WORKDIR /var/www/html

# 6. Create fresh Laravel project
RUN composer create-project laravel/laravel .

# 7. Setup Environment (.env) and SQLite
# Note: Yahan 'sed' mein humne '|' use kiya hai taake path ka slash '/'
masla na kare
RUN cp .env.example .env && \
    sed -i 's/DB_CONNECTION=mysql/DB_CONNECTION=sqlite/g' .env && \
    sed -i
's|DB_DATABASE=laravel|DB_DATABASE=/var/www/html/database/database.sqlite|g'
.env && \
    mkdir -p /var/www/html/database && \
    touch /var/www/html/database/database.sqlite

# 8. Set Apache Document Root to Laravel's public folder
RUN sed -i 's|/var/www/html|/var/www/html/public|g'
/etc/apache2/sites-available/000-default.conf

# 9. Final Permissions
RUN chown -R www-data:www-data /var/www/html && \
    chmod -R 775 /var/www/html/storage /var/www/html/bootstrap/cache
/var/www/html/database
```

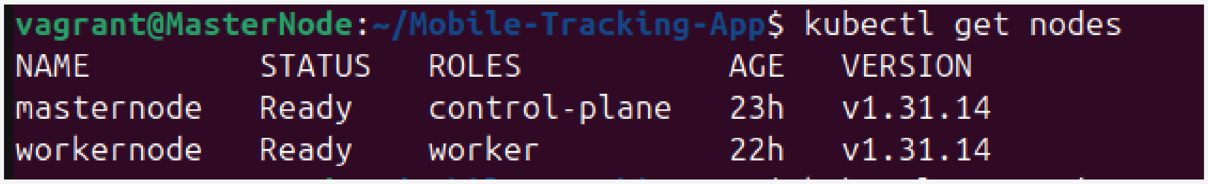
```
# 10. Generate Laravel Application Key
RUN php artisan key:generate

EXPOSE 80
```

Docker Compose file (docker-compose.yml)

```
version: "3.9"

services:
  frontend:
    build: ./frontend
    ports:
      - "80:80"
    restart: always

  backend:
    build: ./backend
    ports:
      - "8001:80"
    environment:
      REDIS_HOST: redis
    depends_on:
      - redis
    restart: always

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    restart: always
```

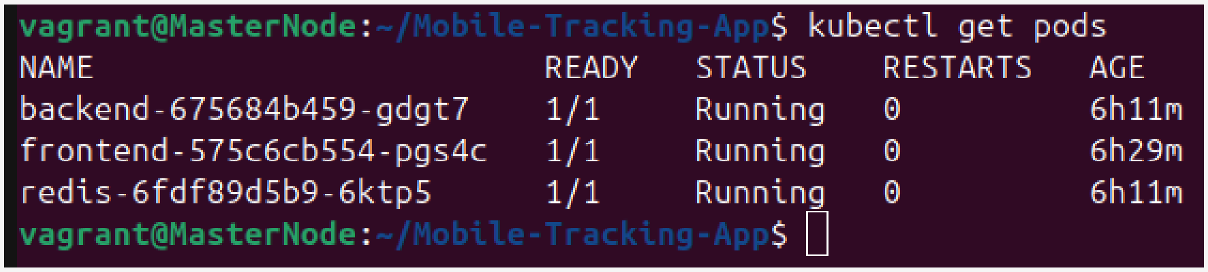
```
cd ~/Mobile-Tracking-App

kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
kubectl apply -f backend-deployment.yaml
kubectl apply -f backend-service.yaml
kubectl apply -f redis-deployment.yaml
kubectl apply -f redis-service.yaml
```

```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get nodes

NAME          STATUS   ROLES           AGE   VERSION
masternode    Ready    control-plane   22h   v1.31.14
workernode    Ready    worker          21h   v1.31.14
```



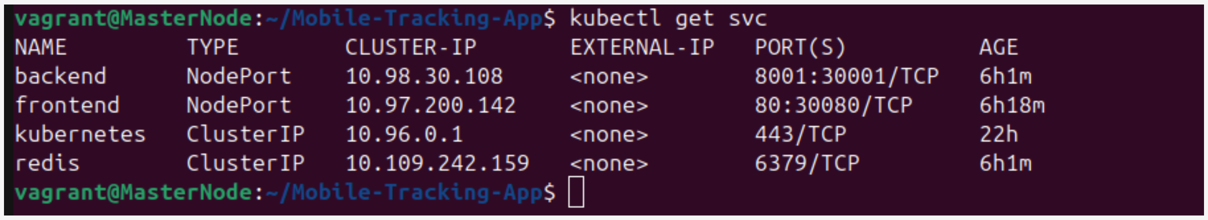
```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get nodes
NAME          STATUS    ROLES           AGE    VERSION
masternode    Ready     control-plane   23h    v1.31.14
workernode    Ready     worker          22h    v1.31.14
```

```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get pods

NAME                         READY   STATUS    RESTARTS    AGE
backend-675684b459-gdgt7     1/1     Running   0           5h24m
frontend-575c6cb554-pgs4c    1/1     Running   0           5h42m
redis-6fdf89d5b9-6ktp5       1/1     Running   0           5h24m
```



```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get pods
NAME                         READY    STATUS     RESTARTS    AGE
backend-675684b459-gdgt7     1/1      Running    0           6h11m
frontend-575c6cb554-pgs4c    1/1      Running    0           6h29m
redis-6fdf89d5b9-6ktp5       1/1      Running    0           6h11m
vagrant@MasterNode:~/Mobile-Tracking-App$
```

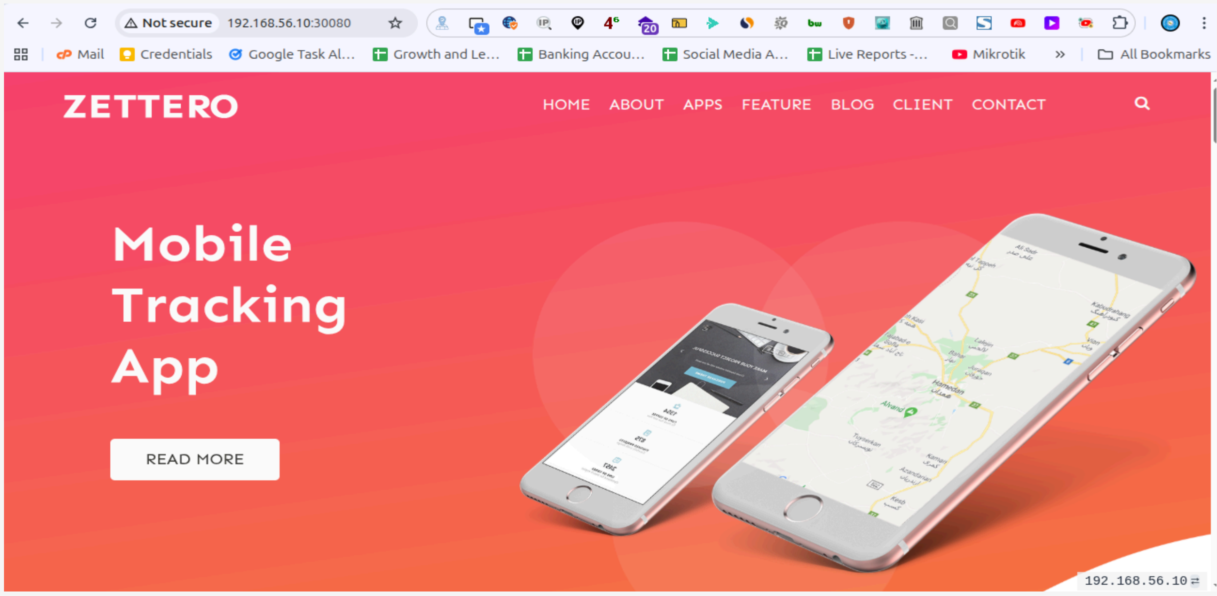
```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
backend      NodePort    10.98.30.108     <none>        8001:30001/TCP   6h1m
frontend     NodePort    10.97.200.142    <none>        80:30080/TCP     6h18m
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          22h
redis        ClusterIP   10.109.242.159   <none>        6379/TCP         6h1m

vagrant@MasterNode:~/Mobile-Tracking-App$
```



```
vagrant@MasterNode:~/Mobile-Tracking-App$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
backend      NodePort    10.98.30.108     <none>        8001:30001/TCP   6h1m
frontend     NodePort    10.97.200.142    <none>        80:30080/TCP     6h18m
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          22h
redis        ClusterIP   10.109.242.159   <none>        6379/TCP         6h1m
vagrant@MasterNode:~/Mobile-Tracking-App$
```

Frontend

- http://192.168.56.10:30080

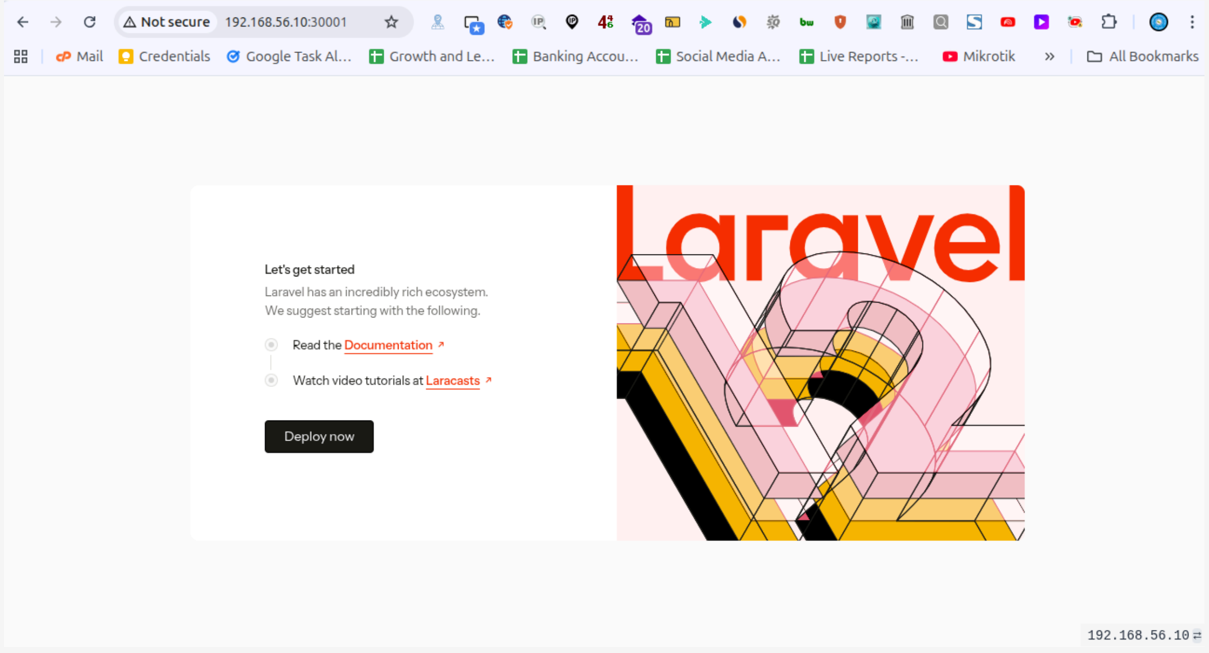


Backend API

- http://192.168.56.10:30001

Example:

- http://192.168.56.10:30001/health
- http://192.168.56.10:30001/api


192.168.56.10:30001
Let's get started
Laravel has an incredibly rich ecosystem.
We suggest starting with the following.
Read the Documentation
Watch video tutorials at Laracasts
Deploy now
Laravel