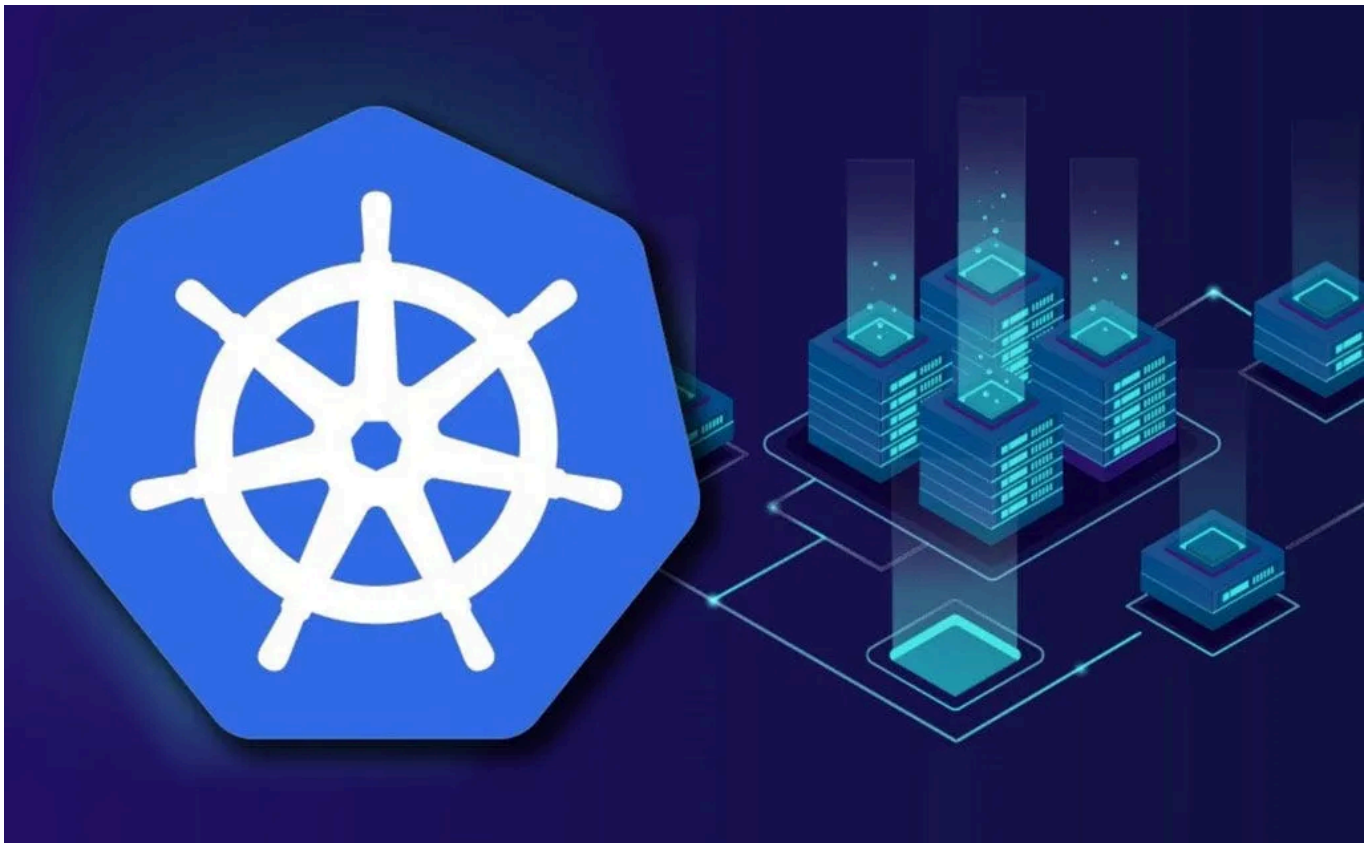
Step-by-Step: Building a Kubernetes Cluster from Scratch on Ubuntu 24.04 (2025 Guide)

Osama Abusitta Follow 6 min read · Sep 13, 2025

8 2

Kubernetes powers modern container orchestration, running everything from small test environments to enterprise-scale platforms. While cloud providers like GKE, EKS, and AKS make cluster creation easy, they often come with higher costs and limited flexibility. Many engineers therefore choose to build Kubernetes manually — not only to achieve a production-ready cluster at a lower cost , but also to gain deeper understanding and full control over their infrastructure.



In this guide, we'll walk through creating a three-node Kubernetes cluster using the latest tooling and practices as of 2025.

The setup includes:

- Ubuntu Server 24.04 LTS (64-bit) on both nodes
- Docker Engine as the container runtime (via cri-dockerd)
- Calico for pod networking and network policies
- A control plane node and two worker nodes

We'll use kubeadm, the official installation tool recommended by the Kubernetes project, and validate every step to ensure the cluster is stable and ready for workloads. By the end, you'll have a fully functioning Kubernetes cluster.

Prerequisites

- Three Ubuntu 24.04 LTS servers:
- Master node: IP (xx.xxx.x.x1) replace this value with you master node IP
- Two worker nodes: IP (xx.xxx.x.x2 , xx.xxx.x.x3) replace this value with you worker node IP
- At least 2 CPUs and 4GB RAM per node
- Sudo access
- Basic Linux knowledge

Basic prep (do this on all nodes)

Optional set hostnames

- On master node

```
# (optional) set hostnames
# on master node xx.xxx.x.x1
sudo hostnamectl set-hostname master-1
```

- On worker node 1

```
# on worker node 1 ip xx.xxx.x.x2
sudo hostnamectl set-hostname worker-1
```

- On Worker node 2

```
# on worker node 1 ip xx.xxx.x.x3
sudo hostnamectl set-hostname worker-2
```

- On All nodes

```
ries on BOTH nodes (adjust if you use internal/private IPs)
x.x.x1 master-1" | sudo tee -a /etc/hosts
x.x.x2 worker-1" | sudo tee -a /etc/hosts
x.x.x3 worker-1" | sudo tee -a /etc/hosts

ackages & basic tools
t update && sudo apt-get install -y curl ca-certificates gpg apt-transport-https
```

Validate:

```
hostnamectl
getent hosts master-1 worker-1 worker-2
```


trusted sources, especially port 6443/tcp (API server). Required ports reference

1) Kernel & networking sysctls & disable swap (all nodes)

kubeadm expects ip forwarding enabled and bridged traffic visible to iptables; swap must be off. Container-Runtimes

```
# load modules on boot
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter

# sysctls for routing & bridged traffic
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
sudo sysctl --system

# disable swap (runtime + persistent)
sudo swapoff -a
sudo sed -ri '/\sswap\s/s/^/#/' /etc/fstab
```

Validate

```
lsmod | grep -E 'br_netfilter|overlay'
sysctl net.ipv4.ip_forward
sysctl net.bridge.bridge-nf-call-iptables
swapon --show     # (should print nothing)
```

2) Install Docker Engine (all nodes)

Kubernetes provides several common container runtimes (in this guide we will use docker)

- Containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime

Will use Docker's official packages, then ensure the Docker cgroup driver is systemd (recommended with systemd-based hosts).

```
# Docker official repo
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg - dearmor -o /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg - print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu noble stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

Set cgroup driver to systemd:

```
sudo mkdir -p /etc/docker

cat <<'EOF' | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": { "max-size": "100m" },
  "storage-driver": "overlay2"
}
EOF


sudo systemctl daemon-reload
sudo systemctl enable --now docker
sudo systemctl restart docker
```

You can set Docker's cgroup driver via exec-opts and, on cgroup v2 systems, Docker defaults to systemd when available. Docker Document

Make docker available without sudo access

```
sudo usermod -aG docker $USER
newgrp docker
docker ps
```

Validate

```
docker info --format '{{.CgroupDriver}}' # should be: systemd
systemctl is-active docker # active
```

3) Install cri-dockerd (all nodes)

Kubernetes removed dockershim; Docker now needs the cri-dockerd adapter.

The default CRI socket is /run/cri-dockerd.sock. [Kubernetes Docker Engine]

```
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.20/cri-docke

sudo dpkg -i cri-dockerd_0.3.20.3-0.debian-bookworm_amd64.deb
sudo apt-get install -f -y

which cri-dockerd

# enable socket/service
sudo systemctl enable --now cri-docker.socket
sudo systemctl enable --now cri-docker.service
```

Validate

```
systemctl is-active cri-docker.socket

systemctl is-active cri-docker.service
```

```
ss -ltn | grep -E '(/run/cri-dockerd.sock)?' # optional

which cri-dockerd
```

4) Install kubeadm, kubelet, kubectl (all nodes)

Use the pkgs.k8s.io repos for v1.34 (current stable docs) [docs](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/):

```
sudo apt-get update

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

# keyring (create dir if needed)
sudo mkdir -p -m 755 /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.34/deb/Release.key | \
        sudo gpg - dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.
  sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl
```

Validate

```
kubeadm version && kubelet --version && kubectl version --client

systemctl enable - now kubelet || true # kubelet may stay in a waiting state unt
```

5) Control-plane init (run ONLY on master-1)

Initialize the cluster, pointing kubeadm at cri-dockerd, and set the pod CIDR for Calico.

kubeadm init flags, CRI sockets and cluster creation flow are covered in the kubeadm docs. [Creating a cluster with kubeadm]

replace xx.xxx.x.x1 with mster node IP

```
sudo kubeadm init \
 - apiserver-advertise-address=xx.xxx.x.x1 \
 - pod-network-cidr=192.168.0.0/16 \
 - cri-socket=unix:///run/cri-dockerd.sock
```

When it finishes, set your kubeconfig:

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

Validate

```
kubectl cluster-info

kubectl get nodes -o wide # control-plane will be NotReady until CNI is installe
```

6) Install Calico CNI (on master-1)

Use the Tigera Operator (recommended). With our chosen 192.168.0.0/16, the default manifest already matches — you can apply it as-is.

[Install Calico networking and network policy for on-premises deployments]

```
# Install operator + CRDs (use latest v3.xx from docs page; example uses v3.30.3
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.30.3

kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.30.3

# Get default custom-resources (edit if you want to change IP pool/CIDR/encapsul
curl -O -L https://raw.githubusercontent.com/projectcalico/calico/v3.30.3/manife

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

Validate

```
kubectl -n tigera-operator rollout status deploy/tigera-operator
watch kubectl get pods -n calico-system
kubectl get nodes # should turn Ready once calico-node is up
```

Calico on-premises install via operator, with default pool 192.168.0.0/16; the operator manages lifecycle and is the recommended path.

7) Join the worker (run on master-1 to get the command, then run it on worker-1 and worker-2)

kubeadm provides a ready join command; add the CRI socket for Docker (cri-dockerd).

```
# on master-1, print a join command (no TTL so you can reuse it today)
kubeadm token create --print-join-command --ttl 0
```

The output look like (xx.xxx.x.x1 is you master node IP)

```
kubeadm join xx.xxx.x.x1:6443 - token <TOKEN> \
 --discovery-token-ca-cert-hash sha256:25a08a5bba53f27d76c944a7a1b41b35e8f138cc8
```

Copy the full kubeadm join command and run it on worker-1 and worker-2

*** Sometimes you need to use sudo and add ` — cri-socket=unix:///var/run/cri-dockerd.sock` to the end of command*

To be like :

```
sudo kubeadm join xx.xxx.x.x1:6443 \
 --token <TOKEN> \
 --discovery-token-ca-cert-hash sha256:<HASH> \
 --cri-socket=unix:///var/run/cri-dockerd.sock
```

Validate on master node (master-1)

```
kubectl get nodes -o wide
kubectl get pods -A
```

(Optional) UFW / security-group ports to allow

Control-plane node needs inbound:

6443/tcp (API)

10250/tcp (kubelet)

10257/tcp (kube-controller-manager)

10259/tcp (kube-scheduler).

etcd (2379–2380) is local unless externalized.

NodePort range 30000–32767/tcp;

kubelet 10250/tcp.

Ports and Protocols

When running Kubernetes in an environment with strict network boundaries, such as on-premises datacenter with physical...

kubernetes.io

On master node (master-1)

```
sudo ufw allow 6443,10250,10257,10259/tcp
sudo ufw allow 30000:32767/tcp
```

On worker nodes (worker-1, worker-2)

```
sudo ufw allow 10250/tcp
sudo ufw allow 30000:32767/tcp
```

Cluster sanity checks

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

References

Installing kubeadm/kubelet/kubectl & pkgs.k8s.io (v1.34 docs).

Create a cluster with kubeadm / init & join flow.

Container runtimes.

Calico install on-prem.

Required Kubernetes ports.

Docker cgroup driver configuration.

Next : Install Kubernetes cluster dashboard

Kubernetes | Kubernetes Cluster | Containers | Container Orchestration | Ubuntu

Written by Osama Abusitta

23 followers · 1 following

Follow

Responses (2)

Write a response

What are your thoughts?

Feras Tamimi
Dec 18, 2025

Well thought out steps, thank you! :)

Reply

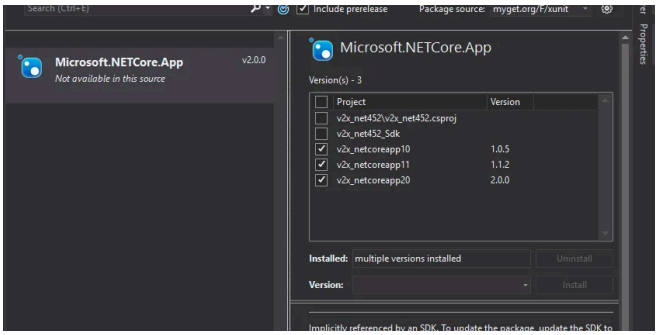
이상빈
Oct 16, 2025

i like you

excellent ❤️

Reply

More from Osama Abusitta



Osama Abusitta

Resolving Dependency Conflicts in .NET Projects: A Comprehensiv...

In Part 1 of this series, we explored how dependencies are declared, restored, and...

Aug 29, 2025 2

Osama Abusitta

Comprehensive Guide to .NET 8 Project Dependencies: Part 1

This is the first part of a multi-part series exploring .NET 8 project dependencies. In th...

Sep 11, 2024 3 1

Osama Abusitta

Dependency Management in .NET Libraries: A Guide for Library...

Consumers don't just get your code—they inherit your mistakes.

Sep 13, 2025

See all from Osama Abusitta

Recommended from Medium

In Write A Catalyst by DevOps voice

COMPLETE DEVOPS CHEAT SHEET HANDBOOK

A practical, example-driven DevOps command reference covering Linux, Git,...

Jan 14 210 2

Taimur Ijlal

If I Had 90 Days to Future-Proof My Cybersecurity Career .. I Would D...

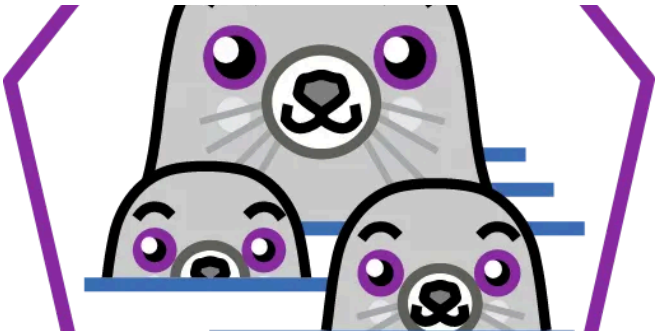
A Step by Step Guide To Surviving AI In Your Cybersecurity Career

4d ago 105 5

Ashish Singh

Linux Outage Triage: 20 Commands I Run Before I Touch...

5d ago 14 1

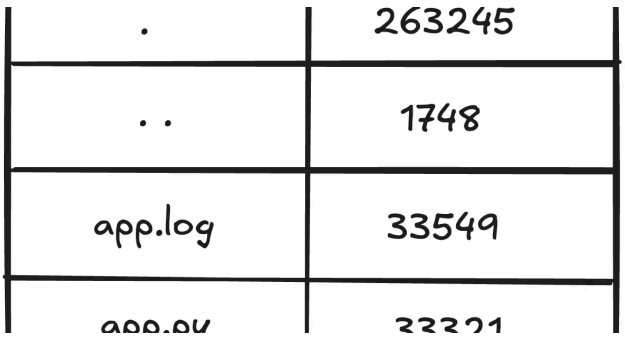


Umair

A Practical Way to Integrate Podman Containers into an Open...

I use Open vSwitch to connect my virtual machines, which works well when everythin...

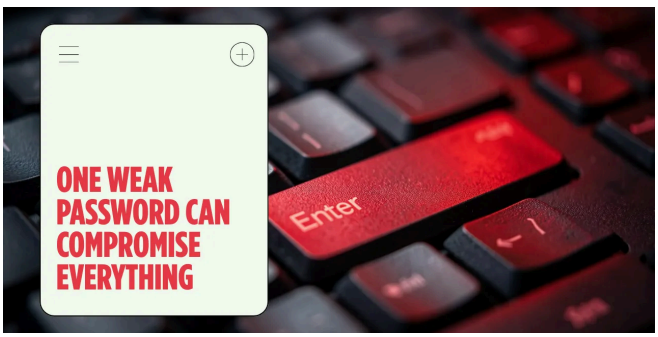
Jan 4



In FAUN.dev() by B Srinivasa Vinay

What Are Inodes? The Hidden Brain Behind Every Linux File

A practical deep dive into inode architecture, how Linux stores files internally, and why...



In CodeX by Pawan Natekar

One Weak Password Is All It Takes: Brute Force Attacks Explained

A beginner-friendly, defensive guide to the most underestimated cyber threat

Oct 25, 2025 6

6d ago 92 1

See more recommendations