**🌐 Virtualization vs Containerization**

| **Aspect** | **Virtualization (VM)** | **Containerization** |
| --- | --- | --- |
| **Definition** | Virtualization uses a **hypervisor** to emulate entire hardware and run multiple OS instances on a host machine. Each **VM** has its **own OS kernel**, libraries, and apps. | Containerization shares the host OS kernel but isolates apps in user spaces (**containers**). Containers are lightweight and portable. |
| **OS Requirements** | Each VM includes a **guest OS** (heavyweight). | Containers **share host OS kernel** (lightweight). |
| **Resource Overhead** | Higher (due to multiple OS kernels and hypervisor layer). | Lower (no need for multiple OS kernels). |
| **Boot Time** | Slow (minutes). | Fast (seconds). |
| **Examples** | VMware, VirtualBox, Hyper-V, KVM. | Docker, Podman, containerd. |

**👉 Summary**:

* Virtualization → Creates **Virtual Machines (VMs)**.
* Containerization → Creates **Containers** (lighter than VMs).

**🐳 Containerization**

Containerization is about **packaging applications with their dependencies** (binaries, libraries, config files) into **containers** so they can run anywhere (dev, test, prod).

**🛠 Container Runtimes**

These are the engines that run containers:

* **Docker** (most popular)
* **Podman** (daemonless, rootless option from RedHat)
* **containerd** (lightweight container runtime used under Docker and Kubernetes)
* **CRI-O** (used in OpenShift and Kubernetes)

**☸️ Container Orchestration**

When you need to manage **hundreds/thousands of containers**, orchestration tools help:

| **Tool** | **Description** |
| --- | --- |
| **Kubernetes** | ✅ *Industry-standard*, open-source orchestrator. Manages deployment, scaling, networking, health-checks. |
| **Docker Swarm** | 🐳 Docker’s native clustering. Easier to use but less powerful than Kubernetes. |
| **Apache Mesos** | A general-purpose cluster manager. Can run containers and other workloads. |

**☸️ Kubernetes**

* **What is it?**  
  Kubernetes (aka **K8s**) is an **open-source orchestrator** for **deploying, scaling, and managing containers**.
* It abstracts the underlying infrastructure and provides **declarative configuration** (via YAML).
* Works with container runtimes like **Docker**, **containerd**, and **CRI-O**.

**🏢 Enterprise Kubernetes Platforms**

| **Platform** | **Built On Kubernetes?** | **Notes** |
| --- | --- | --- |
| **RedHat OpenShift** | ✅ Yes | Adds developer tooling, security, UI dashboard, uses **oc** CLI. |
| **VMware Tanzu** | ✅ Yes | Integrates with VMware infra. |
| **Rancher** | ✅ Yes | Lightweight and easy to manage multiple K8s clusters. |

**🔑 OpenShift Highlights:**

* Built on **Kubernetes** but adds enterprise features.
* CLI tools:
  + **oc** → OpenShift CLI (like kubectl, but OpenShift-specific).
  + **kubectl** → Standard Kubernetes CLI.

**🛠 How to Install & Deploy Kubernetes**

There are **4 main ways** to deploy Kubernetes:

**1️⃣ Install & Deploy from Scratch (kube binaries)**

👉 This is the **hardcore, manual approach**:

* Download Kubernetes core binaries (kube-apiserver, kube-scheduler, kube-controller-manager, kubelet, kubectl, etc.).
* Set up networking (CNI), certificates, and configuration **yourself**.
* You need deep Kubernetes internals knowledge to do this.

✅ **Use Case**:  
✔️ Great for **learning** and **custom implementations**.  
❌ Not recommended for production unless you are a **cloud vendor** or **building a custom K8s distribution**.

**2️⃣ kubeadm**

* **What is it?**  
  A CLI tool that helps you **bootstrap** a Kubernetes cluster easily.  
  It sets up all the components (control plane + worker nodes) with minimal manual effort.
* Installs:
  + Kubernetes control plane on the master node.
  + Kubelet and kube-proxy on worker nodes.
* After installation, you still need to configure **CNI plugins** (like Calico, Flannel).

✅ **Use Case**:  
✔️ For **on-premise clusters** or when using VMs in a public cloud.  
✔️ Production-ready if you add HA and secure it properly.  
❌ Requires ops expertise for maintenance.

**3️⃣ Minikube**

* **What is it?**  
  A lightweight Kubernetes distribution to run **single-node clusters** locally (for testing and learning).
* Installs everything (control plane + worker) on **your laptop/VM**.

✅ **Use Case**:  
✔️ **Not for production**.  
✔️ Great for **learning** and **developer testing**.

**4️⃣ Managed Kubernetes Services**

These are **cloud provider-managed** Kubernetes offerings.  
The cloud vendor manages:  
✅ Kubernetes installation & upgrades.  
✅ Control plane availability.  
✅ Security patches.

You manage:  
✅ Your apps + worker nodes (depending on configuration).

| **Cloud Provider** | **Service Name** | **Notes** |
| --- | --- | --- |
| **GCP** | **GKE** | Google Kubernetes Engine. Supports autopilot mode (fully serverless). |
| **Azure** | **AKS** | Azure Kubernetes Service. Integrates with Azure AD & DevOps pipelines. |
| **AWS** | **EKS** | Elastic Kubernetes Service. Works well with AWS ecosystem (IAM, VPC). |
| **Oracle** | **OKE** | Oracle Kubernetes Engine. |
| **IBM** | **IKS** | IBM Kubernetes Service. |
| **Red Hat** | **OpenShift** | Adds enterprise features on top of Kubernetes. |

💰 **Charges:**

* You pay **only for the underlying infrastructure** (VMs, storage, networking).
* Kubernetes control plane is often **free or very low cost** (GKE, AKS, and EKS charge ~$0 for control plane in small clusters).

✅ **Use Case**:  
✔️ Perfect for **production workloads** if you want to avoid managing Kubernetes internals.  
✔️ Faster deployment and scaling.

**📝 CLI Tools Comparison**

| **Tool** | **Use Case** |
| --- | --- |
| kubectl | Standard Kubernetes CLI (works everywhere). |
| kubeadm | Bootstrap your own cluster on VMs/Bare-metal. |
| minikube | Single-node dev/test clusters locally. |
| oc | OpenShift CLI (adds OpenShift-specific commands). |

**⚡ Quick Analogy**

* 🛠 **Install from binaries** → *Building a car from individual engine parts* 🧑‍🔧
* 🚧 **kubeadm** → *Assembling a car from a DIY kit* 🔩
* 🚲 **minikube** → *A bicycle for practice in your backyard*
* 🛞 **Managed Service (GKE/AKS/EKS)** → *Ordering an Uber – you just drive.* 🚗

**🚀 Kubernetes Installation Lab Guide (kubeadm + cri-dockerd + Calico)**

This guide walks you through setting up a **Kubernetes cluster** on multiple Ubuntu 20.04/22.04 machines using **kubeadm**, **cri-dockerd**, and **Calico CNI**.

✅ **Cluster Topology**:

* **1 Master Node** (Control Plane)
* **N Worker Nodes**

✅ **Container Runtime**: Docker (via cri-dockerd)  
✅ **Pod Network**: Calico

**🖥️ 1. Prerequisites**

* Ubuntu 20.04/22.04 servers (minimum 2 CPUs, 2GB RAM per node).
* Proper **hostname** set on each node:

bash

CopyEdit

sudo hostnamectl set-hostname <name>

* **Swap disabled**:

bash

CopyEdit

sudo swapoff -a

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

* Internet access on all nodes.
* Root or sudo user access.

**🛠️ 2. Prepare All Nodes (Master & Workers)**

**Step 2.1: Create and Run a Bootstrap Script**

Create a script to install Docker, kubeadm, kubectl, kubelet, and configure cri-dockerd:

bash

CopyEdit

vi script.sh

Paste the following:

bash

CopyEdit

#!/bin/bash

# Update package lists and install Docker

sudo apt update -y

sudo apt install docker.io -y

# Set up Kubernetes repository and install kubelet, kubeadm, and kubectl

sudo mkdir -p -m 755 /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/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.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

# Enable net.bridge.bridge-nf-call-iptables

sudo sysctl net.bridge.bridge-nf-call-iptables=1

# Download and install cri-dockerd

wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.14/cri-dockerd-0.3.14.amd64.tgz

tar -xvf cri-dockerd-0.3.14.amd64.tgz

cd cri-dockerd

sudo install -o root -g root -m 0755 cri-dockerd /usr/local/bin/cri-dockerd

# Download and set up cri-dockerd systemd service

cd ..

wget https://github.com/Mirantis/cri-dockerd/archive/refs/tags/v0.3.14.tar.gz

tar -xvf v0.3.14.tar.gz

cd cri-dockerd-0.3.14/

sudo cp packaging/systemd/\* /etc/systemd/system

sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

# Enable and start cri-docker service

sudo systemctl daemon-reload

sudo systemctl enable --now cri-docker.socket

sudo systemctl enable cri-docker

sudo systemctl start cri-docker

sudo systemctl status cri-docker

Make it executable and run it:

bash

CopyEdit

chmod +x script.sh

./script.sh

✅ Repeat this on **all nodes** (master and workers).

**🎯 3. Initialize the Master Node**

On the **Master Node only**:

**Step 3.1: Create a kubeadm config file**

bash

CopyEdit

vi config.yaml

Paste this, updating the advertiseAddress to your master node’s **private IP**:

yaml

CopyEdit

apiVersion: kubeadm.k8s.io/v1beta3

kind: InitConfiguration

localAPIEndpoint:

advertiseAddress: 172.31.3.203

bindPort: 6443

nodeRegistration:

criSocket: unix:///var/run/cri-dockerd.sock

---

apiVersion: kubeadm.k8s.io/v1beta3

kind: ClusterConfiguration

networking:

podSubnet: 192.168.0.0/16

**Step 3.2: Enable kernel modules and sysctl settings**

bash

CopyEdit

sudo modprobe br\_netfilter

sudo tee /etc/modules-load.d/k8s.conf <<EOF

br\_netfilter

EOF

sudo tee /etc/sysctl.d/k8s.conf <<EOF

net.bridge.bridge-nf-call-iptables = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip\_forward = 1

EOF

sudo sysctl --system

**Step 3.3: Initialize the Cluster**

bash

CopyEdit

sudo kubeadm init --config=config.yaml >> cluster\_initialized.txt

✅ Save the join command from cluster\_initialized.txt for worker nodes.

**📂 4. Configure kubectl (Master Node Only)**

bash

CopyEdit

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

**🌐 5. Install Calico Network Plugin**

On **Master Node**:

bash

CopyEdit

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

**🖥️ 6. Join Worker Nodes**

On **each Worker Node**, run the kubeadm join command from cluster\_initialized.txt:

bash

CopyEdit

kubeadm join 172.31.3.203:6443 --token <token> \

--discovery-token-ca-cert-hash sha256:<hash> \

--cri-socket unix:///var/run/cri-dockerd.sock

**✅ 7. Verify Cluster Status**

**On the Master Node:**

bash

CopyEdit

kubectl get nodes

kubectl get pods -A

Expected output:

* Master node: Ready
* Worker nodes: Ready
* Calico pods: Running

**🆘 Troubleshooting**

* If initialization fails:

bash

CopyEdit

sudo kubeadm reset -f --cri-socket=unix:///var/run/cri-dockerd.sock

Then re-run initialization.

* Check service status:

bash

CopyEdit

systemctl status kubelet

systemctl status cri-docker

**📝 Useful Commands Reference**

| **Command** | **Description** |
| --- | --- |
| kubectl get nodes | Check status of all nodes |
| kubectl get pods -A | View all system and user pods |
| kubectl get pods -A -o wide | Pods with node and IP details |
| kubectl apply -f <file.yaml> | Deploy manifests |
| kubeadm reset | Reset cluster setup (use with caution) |

**📊 Expected Final Status**

| **Component** | **Status** |
| --- | --- |
| Master Node | Ready |
| Worker Nodes | Ready |
| Calico Pods | Running |
| CoreDNS Pods | Running |