Containerization & Virtualization

Building Modern Infrastructure

Why Do We Need Containers?

The Problem: "It works on my machine!"

You've just built your application:

- ✅ Works perfectly on your laptop
- ❌ Crashes on your colleague's computer
- ❌ Fails when deployed to the cloud

Why? Different OS, dependencies, configurations, missing libraries

What is an image?

The Solution: Containers

A container packages everything your app needs:

Your code + runtime + dependencies + system tools

Your image runs in a container

Result: Your app runs the **exact same way** everywhere!

> *Like a shipping container - travels safely anywhere, contents stay the same.*

What's a virtual machine?

Why Virtual Machines?

Before Containers, There Were VMs

The problem VMs solved:

- One physical server = one application
- Expensive hardware sitting idle
- Difficult to isolate applications from each other

Virtual Machines = Multiple Computers on One Machine

A VM creates:

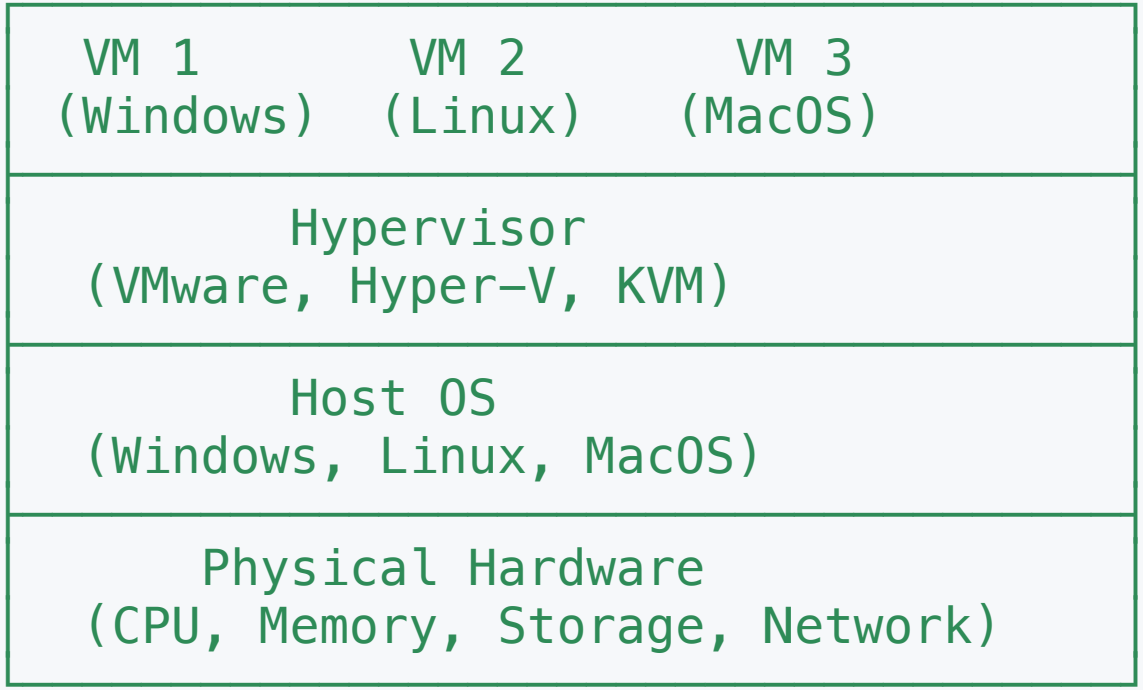
- Complete isolated environments
- Each with its own operating system
- Running on shared physical hardware

Benefits:

- Better hardware utilization
- Strong isolation between applications
- Flexibility to run different operating systems

How Virtual Machines Work

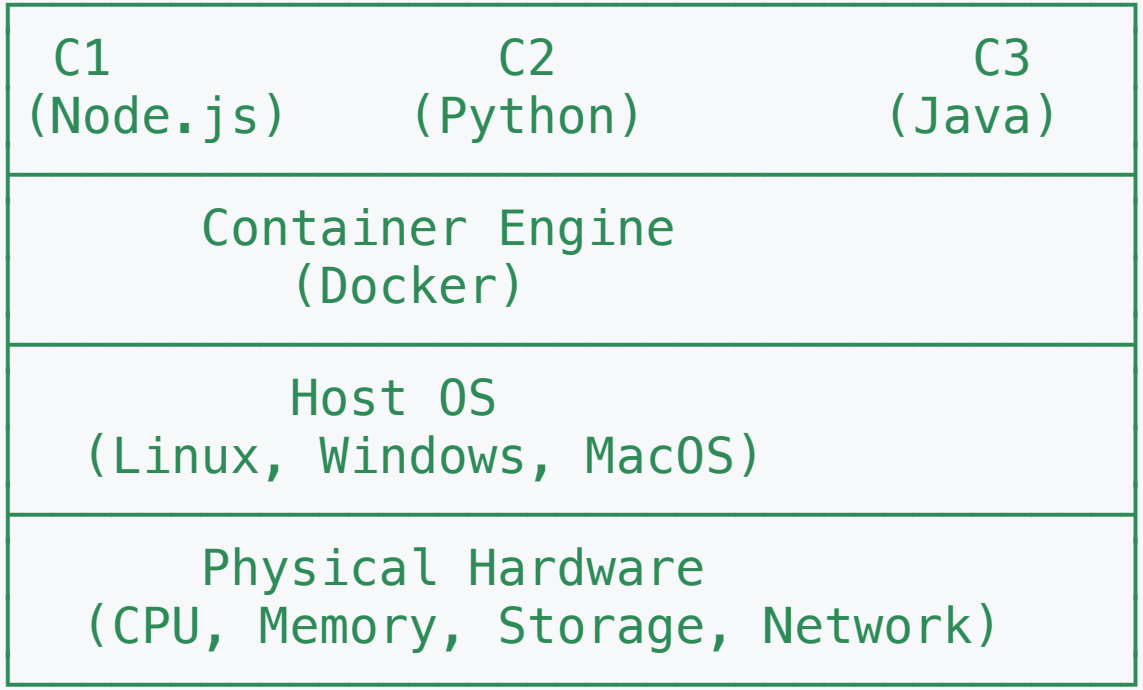
Architecture:



The Hypervisor:

- Sits between host OS and virtual machines
- Allocates physical resources to each VM
- Provides isolation between VMs
- Each VM thinks it has its own computer!

How Containers Work



Containers vs Virtual Machines

Key difference for platform engineers:

Containers	Virtual Machines
Share host OS kernel	Each VM has full OS
Lightweight (MBs)	Heavy (GBs)
Start in seconds	Start in minutes
More efficient	More overhead

Why containers? Portable, consistent, efficient, fast to deploy

Docker Essentials

🐳 Docker = Industry Standard

Core concepts you need to know:

- **Images** - Blueprint/template (read-only, built from a Dockerfile)
- **Containers** - Running instances of images (your actual application)
- **Registries** - Where images are stored and shared

Images vs Containers

Understanding the Difference

Docker Image:

- **Read-only** template or blueprint
- Contains your app code, dependencies, and configuration
- Built once from a Dockerfile
- Stored in registries (Docker Hub, GHCR)
- Can create many containers from one image

Container:

- **Running instance** of an image
- Your actual application executing
- Has its own writable layer
- Can start, stop, restart, delete
- Multiple containers can run from the same image

> ***Analogy:*** *An image is like a class in programming, a container is like an object/instance of that class!*

Container Registries

Container Registries (like GitHub for images):

- Docker Hub, GitHub Container Registry (GHCR), AWS ECR, Azure ACR

Why registries matter:

- Share images across teams
- Version control for images
- Deploy the same image everywhere (dev → staging → prod)

Platform Engineering Essentials

What You Need to Know:

1. Image Optimization

- Smaller images = faster deployments
- Use minimal base images (Alpine Linux)
- Remove unnecessary dependencies

2. Versioning & Tagging

- Tag images properly (v1.0.0, v1.0.1)
- Never use `:latest` in production!
- Link images to source code

3. Security

- Scan images for vulnerabilities
- Use trusted base images
- Run as non-root users

4. Orchestration

- Kubernetes for production scale
- Auto-scaling, self-healing, load balancing

Your Tasks Today

Learn by doing:

1. **Build** Docker images for your applications
2. **Optimize** image size and security
3. **Push** to Container Registry
4. **Version** your images properly

Remember: Use AI (Copilot) to help you learn!