Docker

From Chaos to Containers

A beginner's guide to containerization

Shipping Containers

- Before Containers (1950s)
  - Different shapes, sizes, materials
  - Hard to stack, organize, transport
  - Lots of wasted space
  - Manual loading/unloading

- After Containers (1960s+)
    - **Standardized** size and shape
    - **Stackable** and organized
    - **Portable** across ships, trucks, trains
    - **Efficient** loading with cranes

Why Do We Need Docker?

- The Software Development Problem:
  - "It works on my machine!"
  - "But, I don't know why it does not work on users' machine!"

Traditional Problems

- Different operating systems
- Different software versions
- Missing dependencies
- Environment configuration issues
- Deployment headaches

Docker Solves These Problems

- Just like shipping containers Docker standardizes software deployment:
  - **Consistent** environments
  - **Portable** applications
  - **Isolated** processes
  - **Scalable** infrastructure
  - **Reproducible** builds

- The Software Development Problem Solved:
    - "It works on my machine!"
    - "So, I ship the environment that my machine is built together with the application.
- We already observed this solution pattern before
    - Python venv

What is Docker?

- Docker is a platform that uses **containerization** to
    - package applications
    - and their dependencies together.

Key Concepts

- **Container**: Like a shipping container for your app
- **Image**: Blueprint/template for containers
- **Dockerfile**: Recipe to build images
- **Registry**: Warehouse for storing images (Docker Hub)

Docker Architecture

- Applications are executed in isolation.

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Application   │    │   Application   │    │   Application   │
├─────────────────┤    ├─────────────────┤    ├─────────────────┤
│  Dependencies   │    │  Dependencies   │    │  Dependencies   │
├─────────────────┤    ├─────────────────┤    ├─────────────────┤
│   Docker Image  │    │   Docker Image  │    │   Docker Image  │
└─────────────────┘    └─────────────────┘    └─────────────────┘
        │                       │                       │
        └───────────────────────┼───────────────────────┘
                                │
                    ┌─────────────────┐
                    │  Docker Engine  │
                    ├─────────────────┤
                    │ Operating System│
                    └─────────────────┘
```

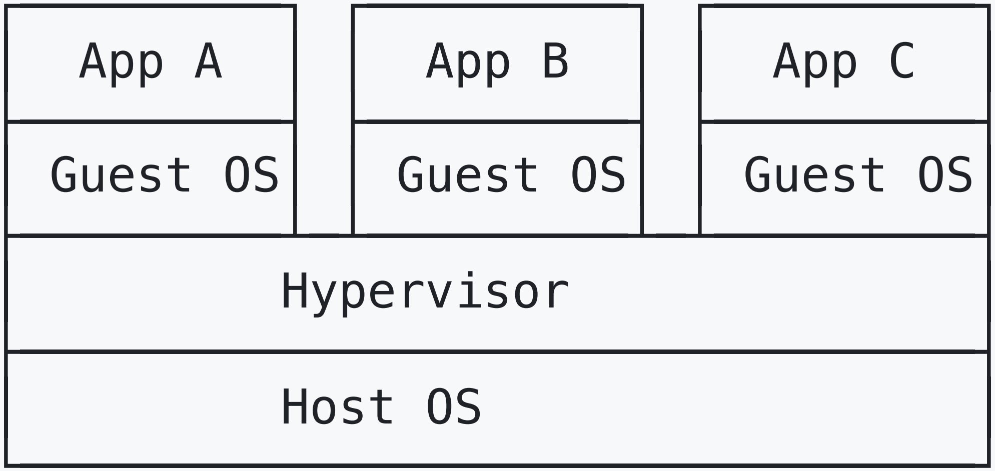
Warning: Docker is not

> Notice: **Docker packages your server environment, not your user application**

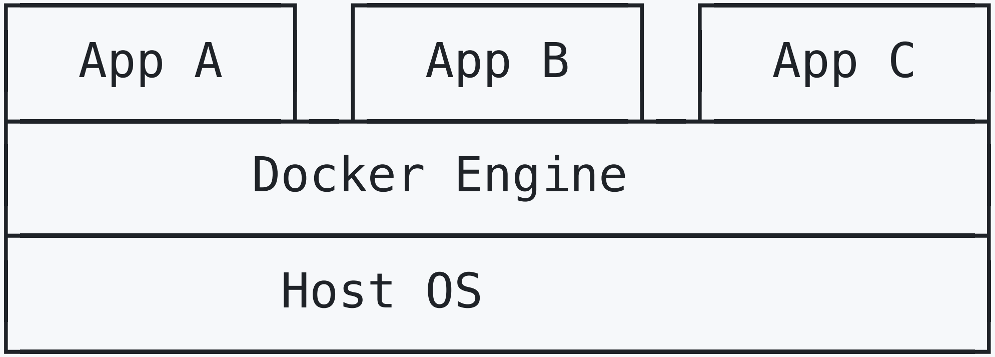
- **Servers**: Need consistent, isolated environments → Use Docker
- **End Users**: Need native, easy-to-install apps → Use traditional installers

Containers vs Virtual Machines

- Docker is not a virtual machine



- Docker is a container (isolator).
    - Practically no performance issue.
    - Easy to build and deploy.



App A

App B

App C

Docker Engine

Host OS

Basic Commands

- Essential Docker Commands

```
# Download an image
docker pull nginx
# Run a container
docker run -d -p 8080:80 nginx
# List running containers
docker ps
# Stop a container
docker stop <container_id>
```

Docker Desktop

- Download and use Docker Desktop.
  - It supports all the Docker CLI.
  - We can manage Docker images and containers.

Dockerfile: The Recipe

- What is a Dockerfile?
    - A text file with instructions to build a Docker image
- Java Analogy
    - Source code: Dockerfile
    - Class file: Docker Image
    - Object in memory: Docker Container

Simple Dockerfile Example

- Building a Node.js Application
  - We write down the same process we do when we use Node.js in the Dockerfile.
- The last line has the command that we use for starting the application.
  - `CMD ...`

```
# Start with a base image (like choosing your kitchen)
FROM node:18-alpine

# Set working directory (prepare your workspace)
WORKDIR /app

# Copy dependency list (shopping list)
COPY package*.json ./

# Install dependencies (buy ingredients)
RUN npm install

# Copy application code (bring your recipe)
COPY . .

# Expose port (set the table)
EXPOSE 3000

# Start the application (cook and serve)
CMD ["node", "app.js"]
```

Building and Running

```
# 1. Create the Docker image (t means tag)
docker build -t my-node-app .

# 2. Run the container (p means port)
docker run -p 3000:3000 my-node-app

# 3. Access your app
# Open browser: http://localhost:3000
```

What happened?

- Docker read your recipe (Dockerfile)
    - Java compiler (javac) reads your source file (.java)
- Docker compiles the Dockerfile into a Docker Image.
    - javac compiles the code into a class file.

- Docker Runs the Image in Memory; Now the Image becomes a Container.
    - java virtual machine loads the class in memory and execute it; Now the class becomes an object.

Docker Compose: The Orchestra

- What is Docker Compose?
    - Tool for defining and running **multi-container** applications

Real-world analogy

- **Single Container = Solo musician**
- **Docker Compose = Full orchestra**
    - Perfect for:
        - Web app + Database
        - Frontend + Backend + Database
        - Microservices architecture

Docker Compose Example

- Web Application with Database

```yaml
version: '3.8'

services:
  # Web application (Frontend)
  web:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - database
    environment:
      - DATABASE_URL=postgresql://user:password@database:5432/myapp
  # Database (Backend storage)
  database:
    image: postgres:15
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=myapp
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

- In this example, we use two containers:
    - Web application
    - Database backend storage that the web application relies upon.

```
  # Web application (Frontend)
  web:
    build: . # searches for Dockerfile
...
  database:
    image: postgres:15
```

- Based on this compose file:
  - Docker builds the web application Docker image from the Dockerfile in this (.) directory.
  - Docker builds the database server by pulling image from a repository.

Docker mapping

- Docker maps local port/directory to the port/directory in the Docker image.

```
ports:
  - "3000:3000"
volumes:
  - postgres_data:/var/lib/postgresql/data
```

Docker Compose Commands

- Managing Your Orchestra

```
# Start all services (entire orchestra plays)
docker-compose up
# Start in background
docker-compose up -d
# Stop all services (silence the orchestra)
docker-compose down
# Rebuild services
docker-compose build
```