Introduction to Docker

From Containers to Compose

Introduction to Docker

What are Containers?

Using Basic Docker Commands

Example: NGINX Hello World

Build Your Own Container

Example: Dockerizing a Flask App

Docker Compose

Introduction to Docker

⇢ Open-source

⇢ Easy creation, deployment, and debugging of containers

⇢ Easy to move apps between dev, testing, and prod env

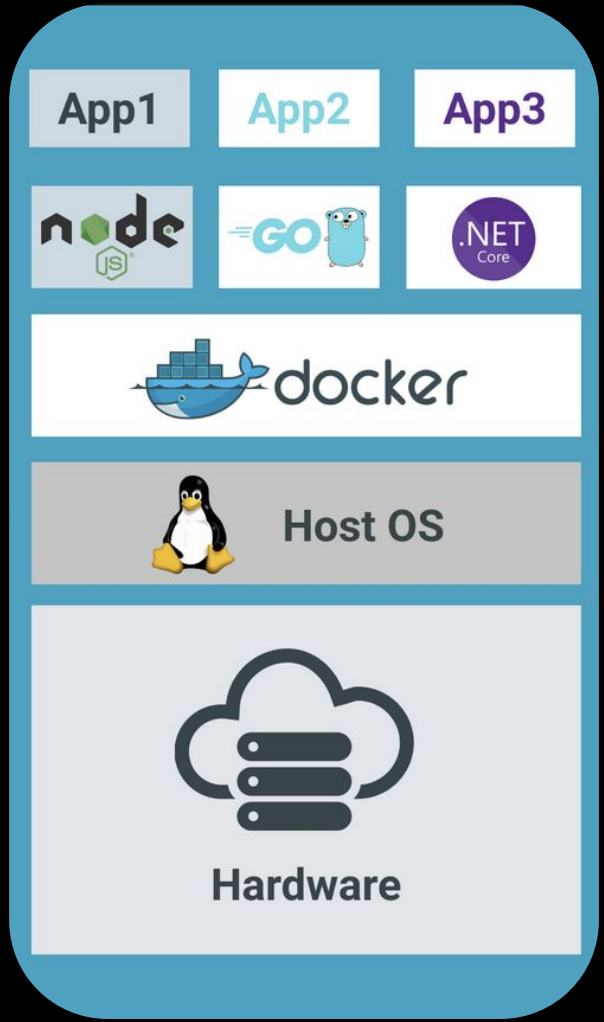
Compared to VMs:

⇢ Faster startup times

⇢ Lower resource usage

⇢ Easier management

⇢ Easier scaling

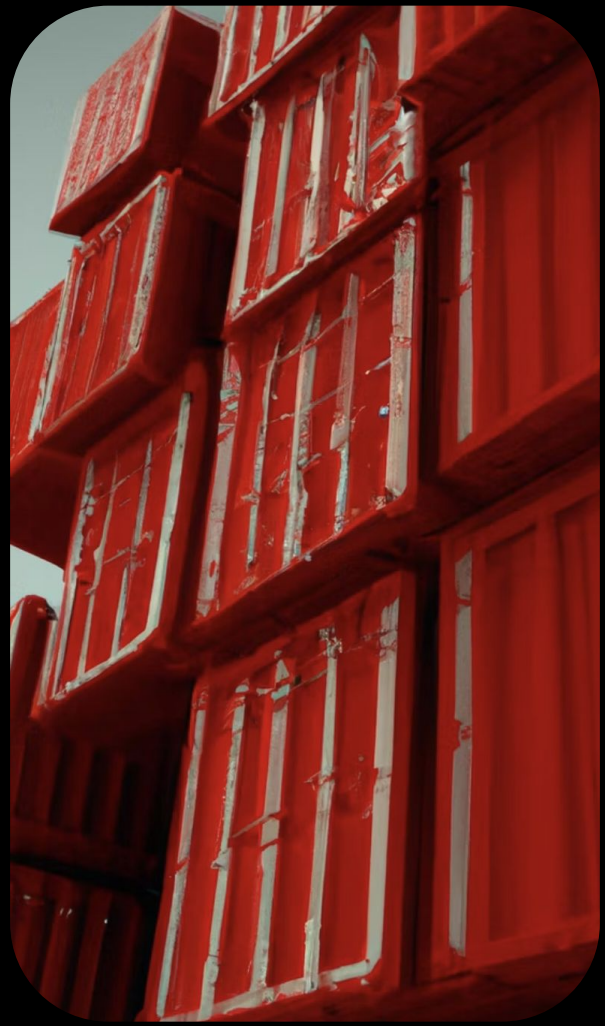


What are Containers?

- Lightweight virtual environments
- Allow packaging of apps and dependencies
- Isolated file system, network, and resources

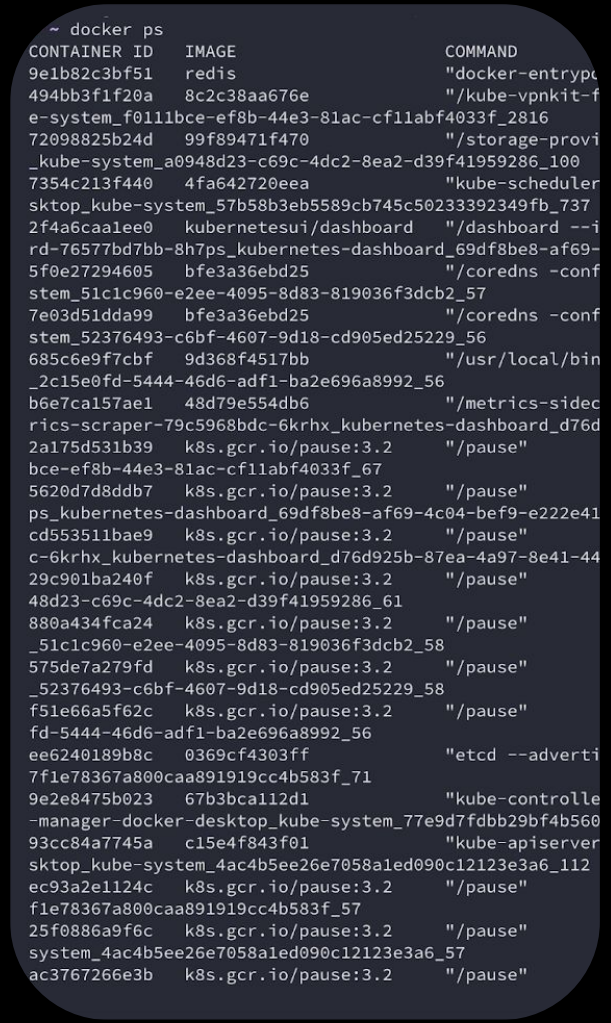
Compared to VMs:

- Same benefits as Docker
- Uses OS kernel, reducing overhead



Docker Commands

- docker pull
- docker images
- docker run
- docker ps
- docker stop



Example: NGINX Hello World

docker run -p 8080:80 nginx

Visit 127.0.0.1:8080 in browser

docker ps -a

(Look for ID)

docker rm <id>

docker image rm nginx

Building Your Own Container

- Built from Dockerfiles and existing containers
- Several different instructions you can use
- Run using “docker run” or (later) “docker compose”

Dockerfile Instructions:

FROM ENV RUN WORKDIR

COPY LABEL EXPOSE CMD

```
FROM python:3.10.0-slim-buster

# Keeps Python from generating .pyc files in the container
# Turns off buffering for easier container logging
# Force UTF8 encoding for funky character handling
# Needed so imports function properly
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
ENV PYTHONIOENCODING=utf-8
ENV PYTHONPATH=/app

# Install MySQL and Poetry
RUN apt-get update -y
RUN apt-get install --no-install-recommends -y build-essential libmariadb-dev-com
RUN pip install https://github.com/python-poetry/poetry/releases/download/1.2.0b2

# Add Poetry path to PATH
ENV PATH="${PATH}:/root/.local/bin"

# Install project dependencies with Poetry
COPY pyproject.toml .
RUN poetry config virtualenvs.create false
RUN poetry install --no-interaction --no-ansi --only main --all-extras --no-root

# Place where the app lives in the container
WORKDIR /app
COPY . /app

# During debugging, this entry point will be overridden.
CMD ["python", "/app/src/bot.py"]
```

Example: Dockerizing a Flask App

Download boilerplate from the BUILDS Workshop GitHub Repo

Fill in the Dockerfile based on the comments

Run “docker build . -t docker-workshop”

Run “docker run -p 8080:8000 docker-workshop”

Visit 127.0.0.1:8080 in your browser

CMD/CTRL-C when finished

Docker Compose

⇢ YML File Format

⇢ Used to define container configurations

⇢ Allows for automatic dependency management

⇢ Easy scaling with Kubernetes

Compared to Docker Run:

⇢ One file for multiple services

⇢ Removes having to remember command-line args

⇢ Lower likelihood to err in deployment

```yaml
version: "3.9"
services:
  character-bot:
    container_name: 1up-call-bot
    image: braxton/1up-call-bot:latest
    restart: unless-stopped
    environment:
      # Discord
      DISCORD_GUILD_ID:
      DISCORD_BOT_TOKEN:
      # Logging
      LOG_LEVEL: INFO
      # Database
      DB_HOST: database
      DB_DATABASE: 1up-call-bot
      DB_USER: 1up-call-bot
      DB_PASSWORD:
    depends_on:
      mariadb:
        condition: service_healthy
    links:
      - mariadb:database
  mariadb:
    container_name: mariadb
    image: mariadb
    restart: unless-stopped
    volumes:
      - ./config/db:/var/lib/mysql
    environment:
      MYSQL_DATABASE: 1up-call-bot
      MYSQL_USER: 1up-call-bot
      MYSQL_PASSWORD:
      MYSQL_ROOT_PASSWORD:
    healthcheck:
      test: "/usr/bin/mysql --user=$$MYSQL_USER --password=$$MYSQ
      # test: "/usr/local/mysql/bin/mysql --user=foo --password=f
      interval: 3s
      timeout: 1s
      retries: 5
```