Docker Workshop

From Basics to Docker Compose

~by Pranavjeet Naidu

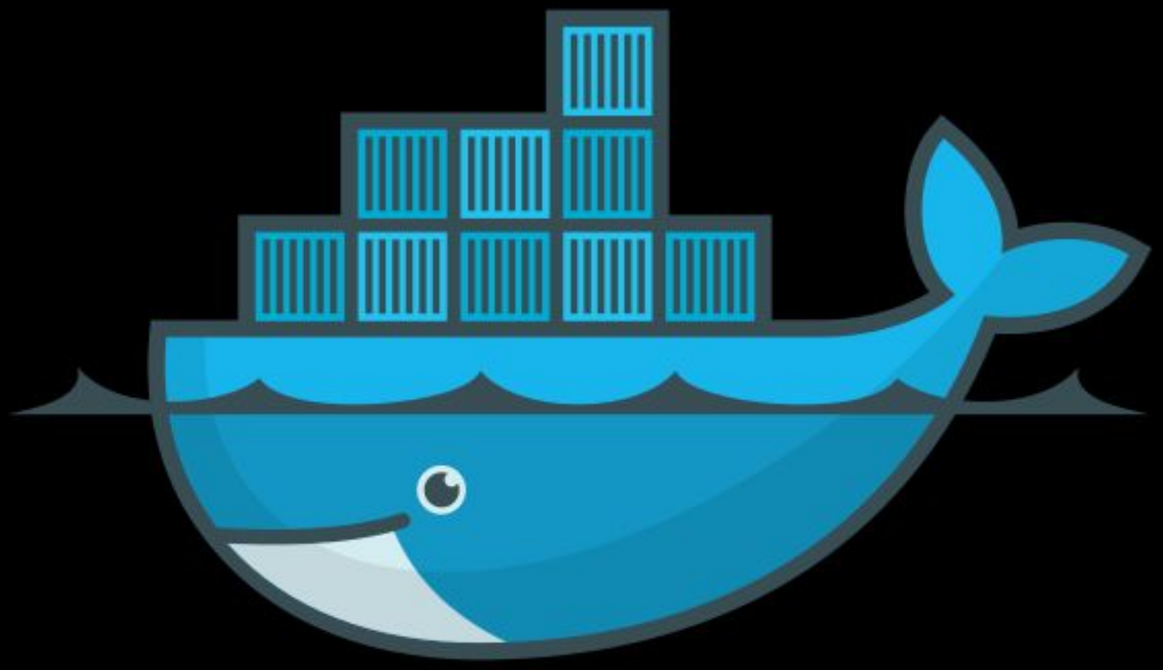
https://github.com/thealcodingclub/containerization101

What is Docker?

- A platform for developing, shipping, and running applications consistently
- Enables packaging applications with all dependencies
- Solves "it works on my machine" problem
- Platform independent deployment

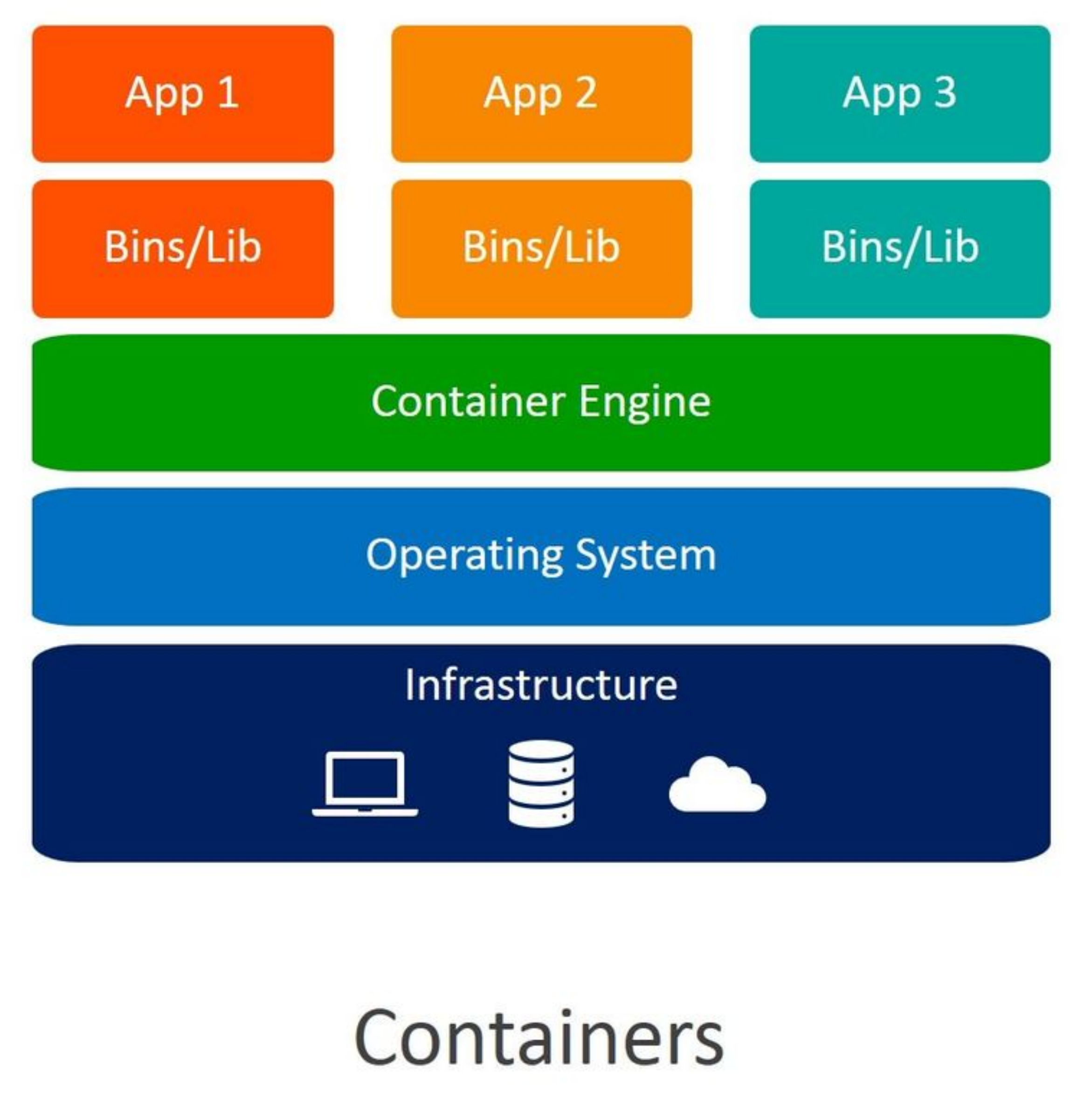
Key benefits:

- Consistency across environments
- Improved collaboration
- Rapid deployment
- Resource efficiency

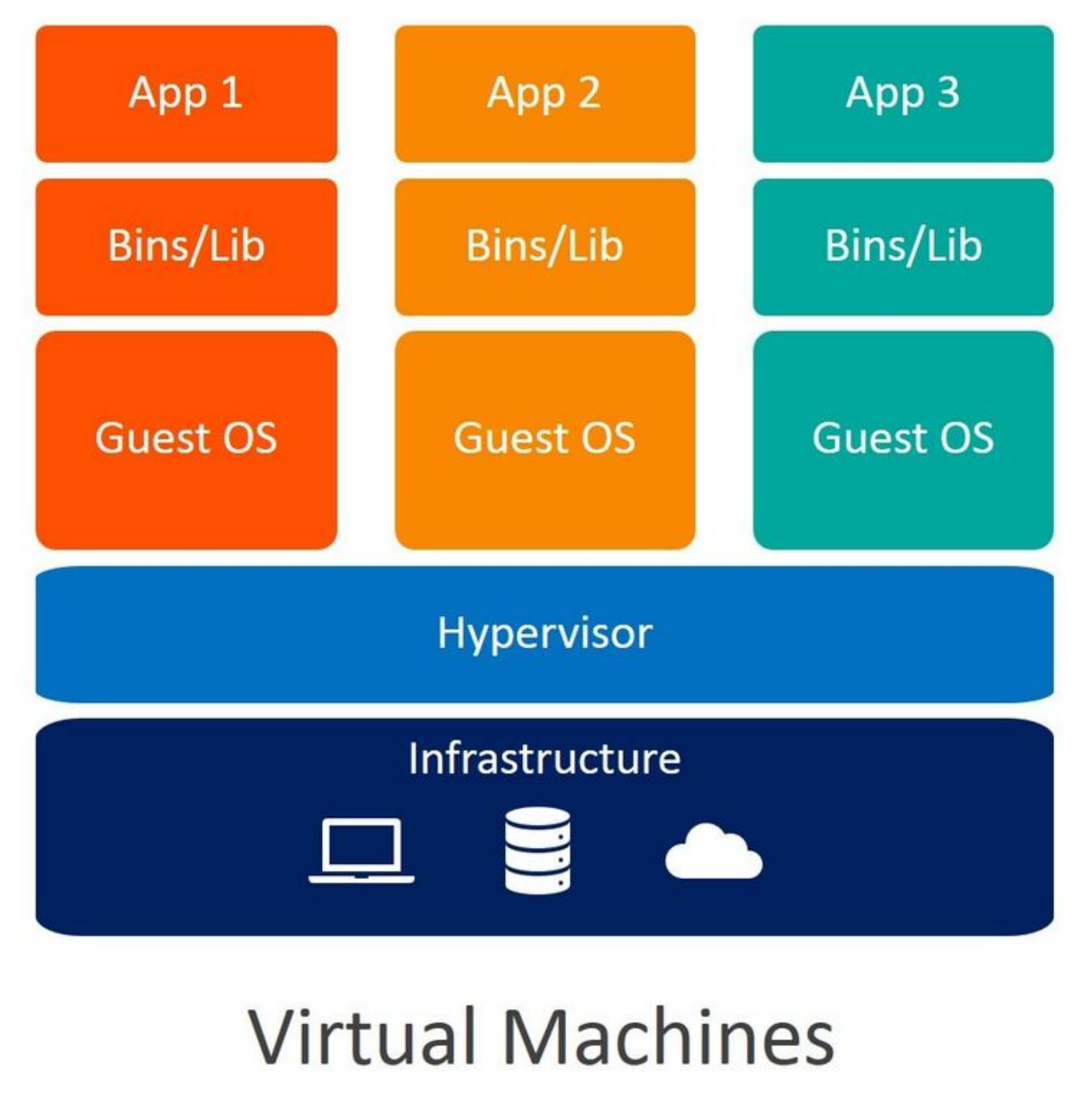


Containerization

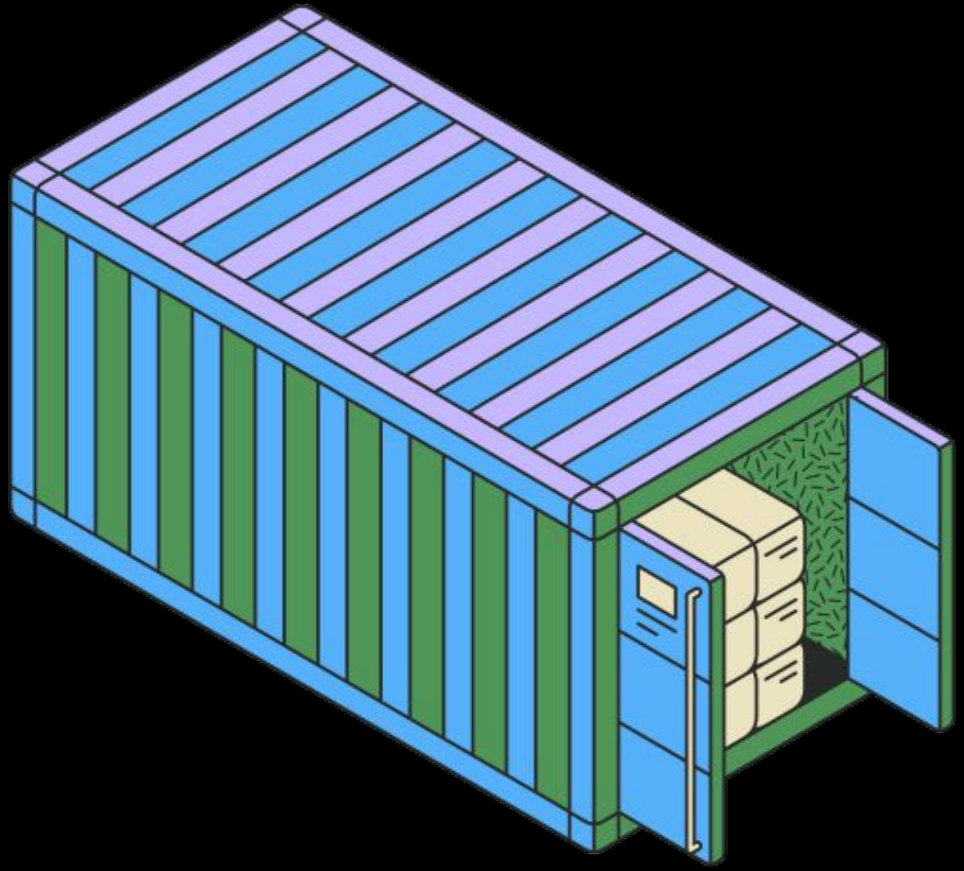
Virtualization



v
s



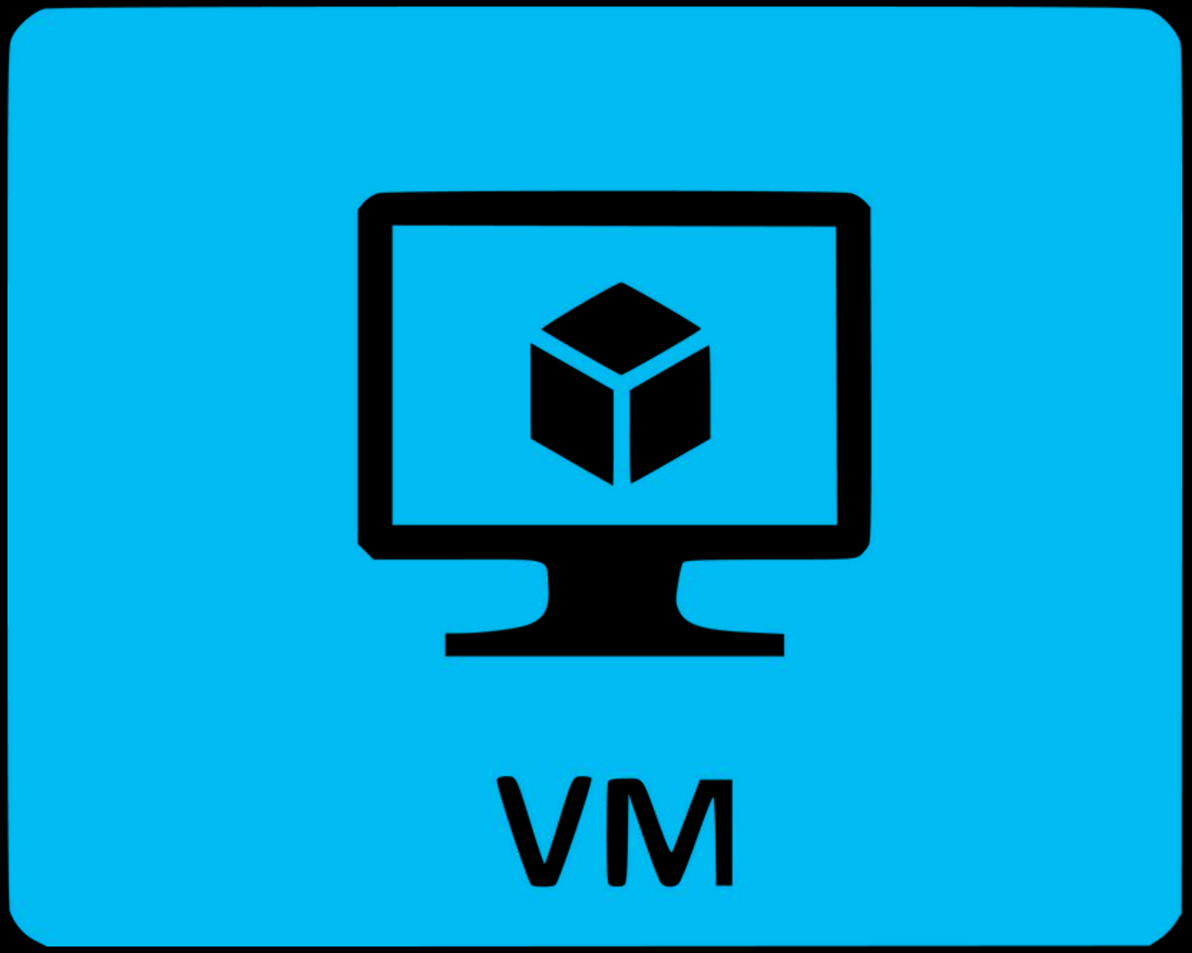
Containerization



- Containers:
  - Share host OS kernel
  - Lightweight (MBs)
  - Seconds to start
  - Less resource intensive
  - Perfect for microservices

Virtualization

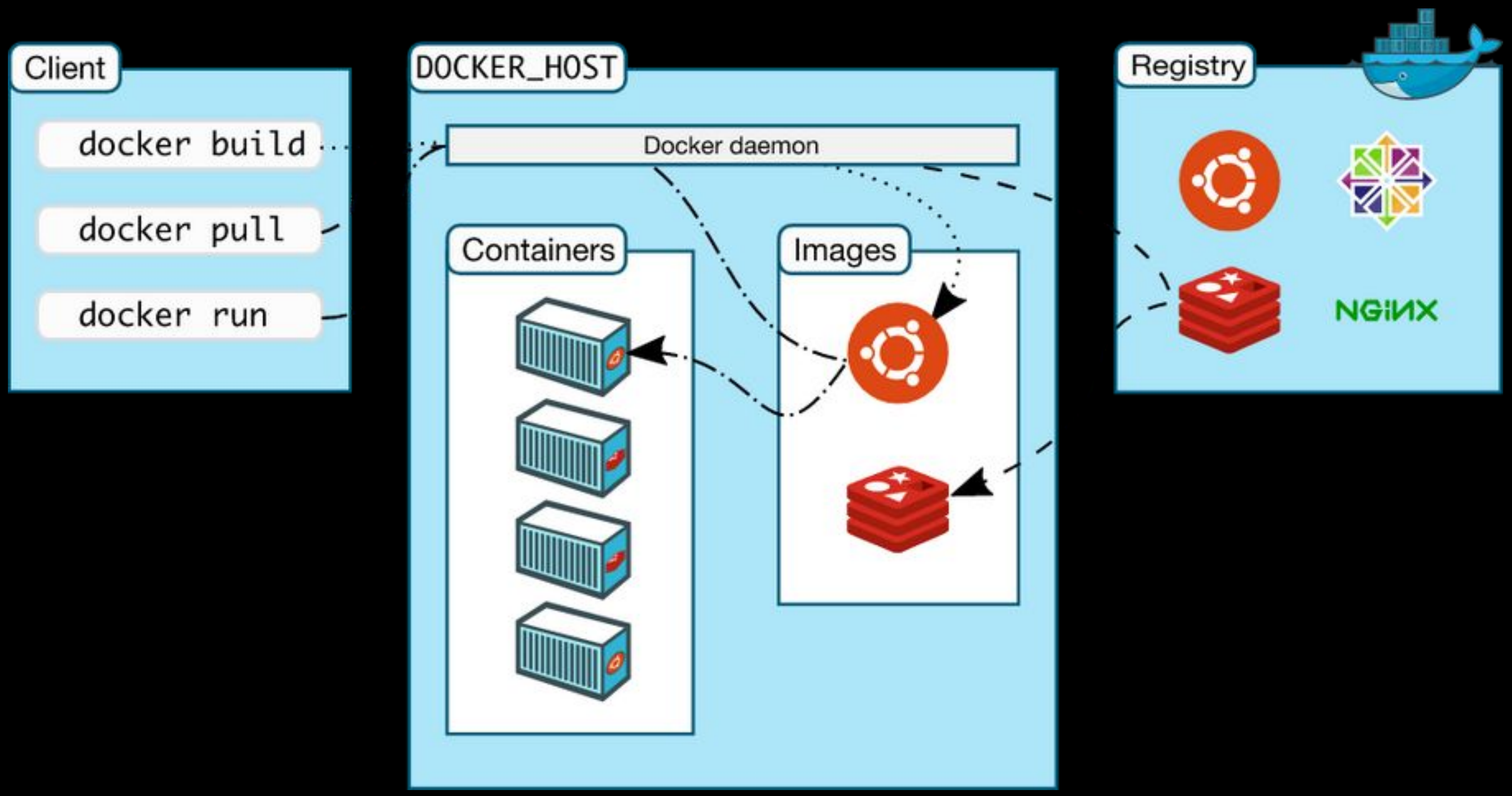
- Virtual Machines:
  - Complete OS copy
  - Heavy (GBs)
  - Minutes to start
  - More resource intensive
  - Better isolation



Docker Architecture

- Client-Server Architecture:
  a. Docker Client: Command line interface
  b. Docker Daemon: Builds, runs, and manages containers
  c. Docker Registry: Stores Docker images
  d. Docker Objects: Images and containers

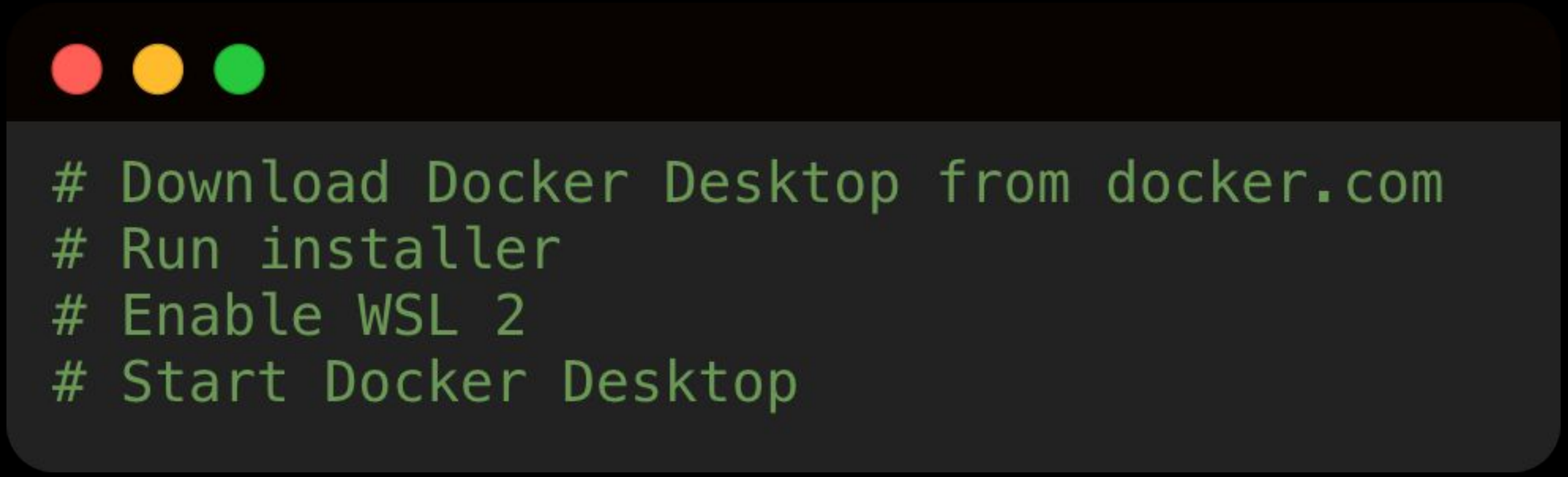
- Communication flow:
  - Client → REST API → Daemon
  - Daemon ↔ Registry



Docker Installation

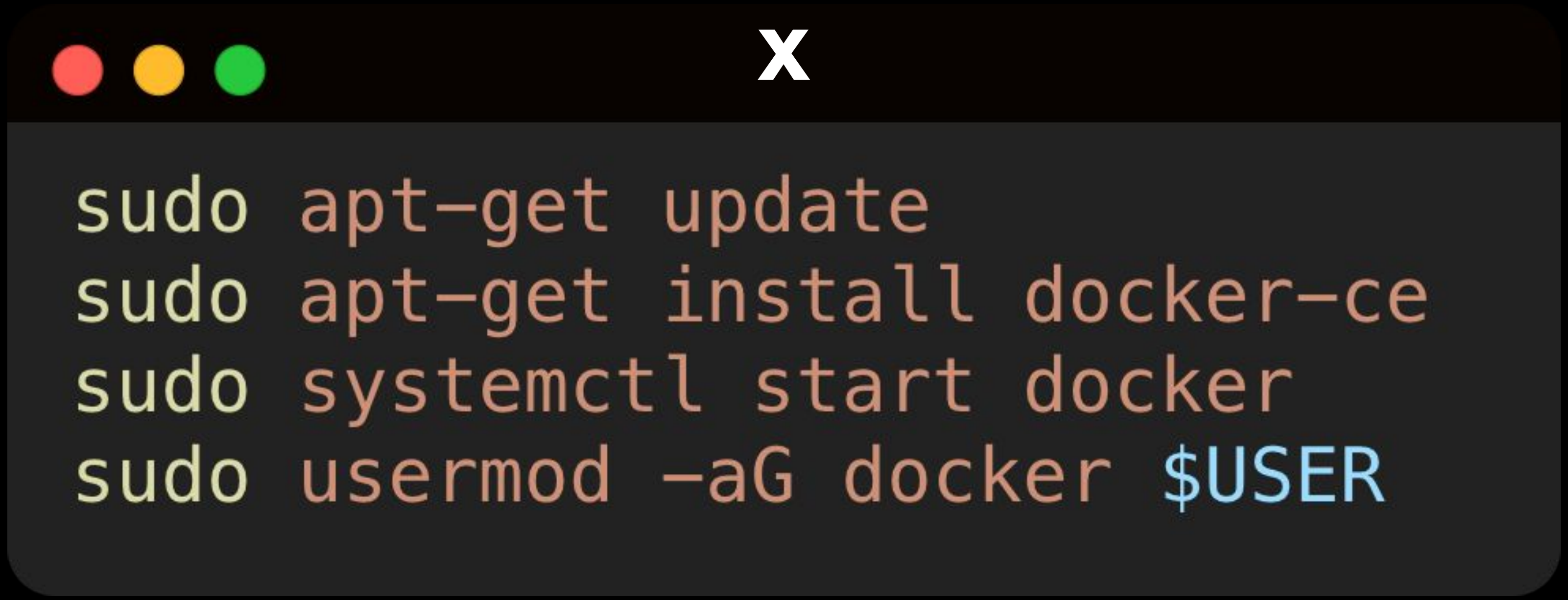
(via CLI)

Windows



```
# Download Docker Desktop from docker.com
# Run installer
# Enable WSL 2
# Start Docker Desktop
```

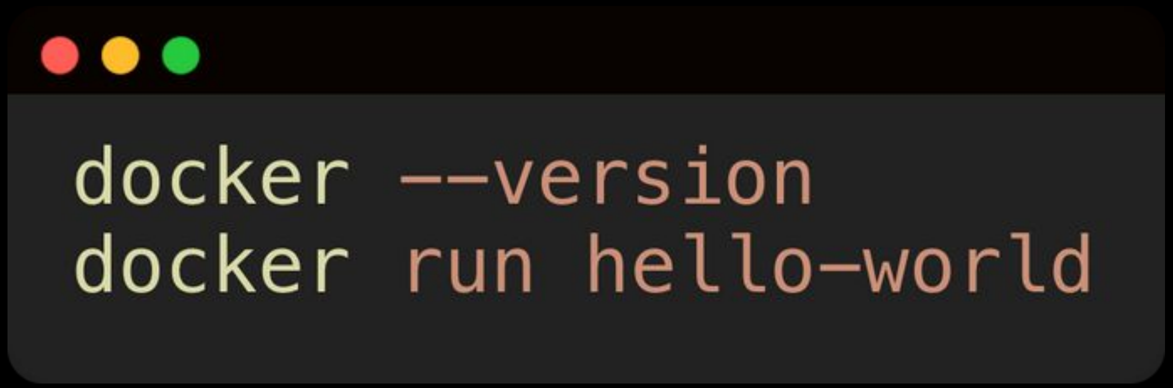
Linu



x

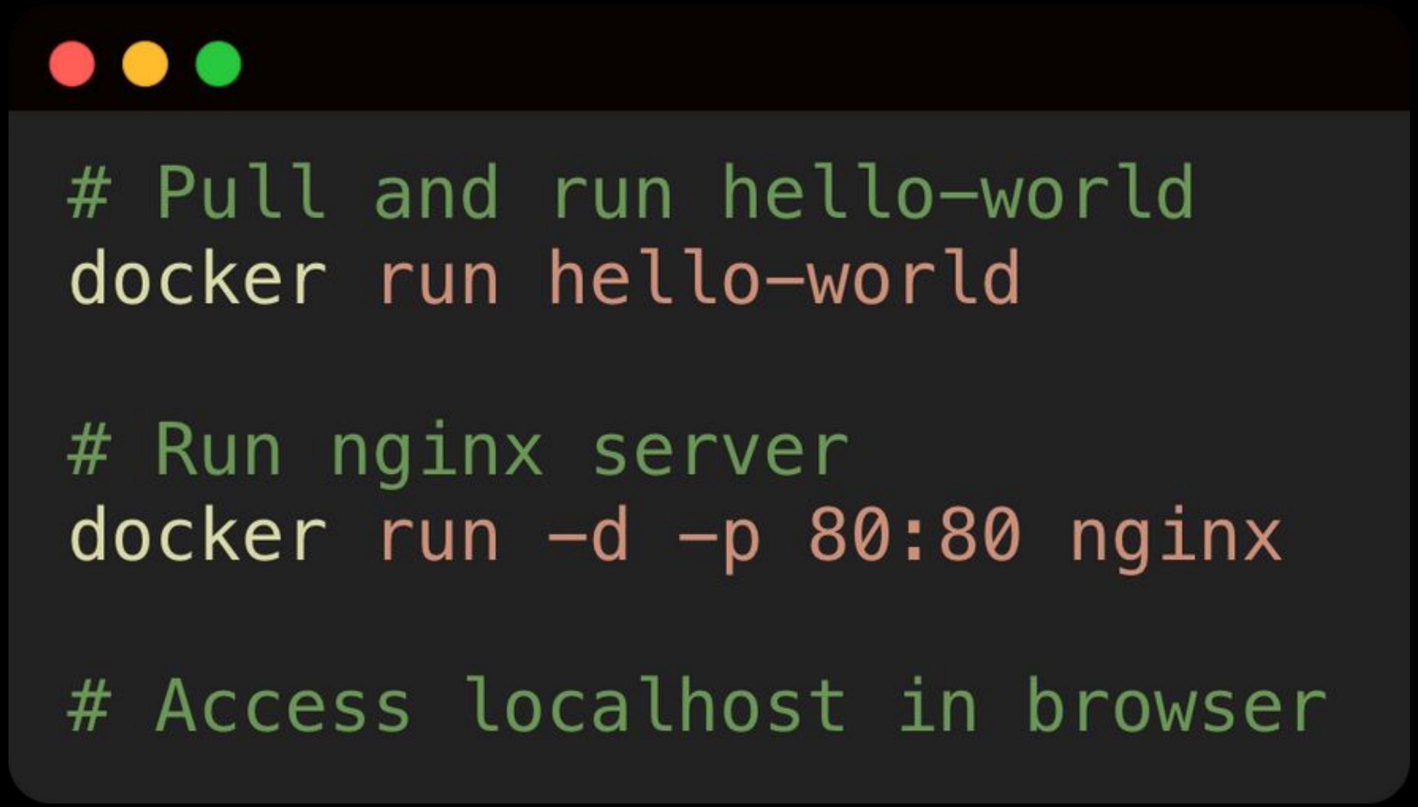
```
sudo apt-get update
sudo apt-get install docker-ce
sudo systemctl start docker
sudo usermod -aG docker $USER
```

Verification



```
docker --version
docker run hello-world
```

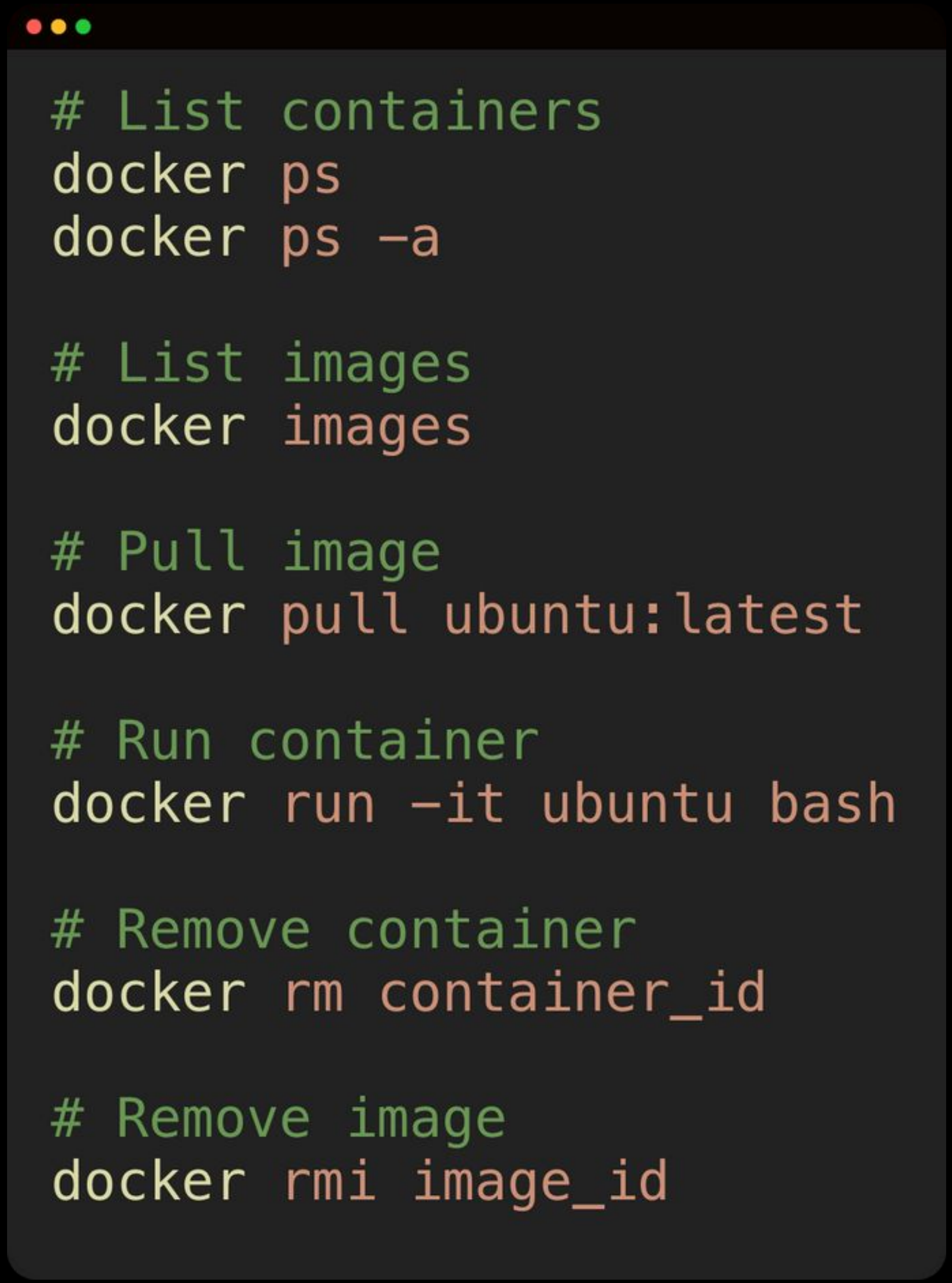
First Container Demo



```
# Pull and run hello-world
docker run hello-world

# Run nginx server
docker run -d -p 80:80 nginx

# Access localhost in browser
```

1. Checks for local image
2. Pulls from Docker Hub
3. Creates container
4. Runs container

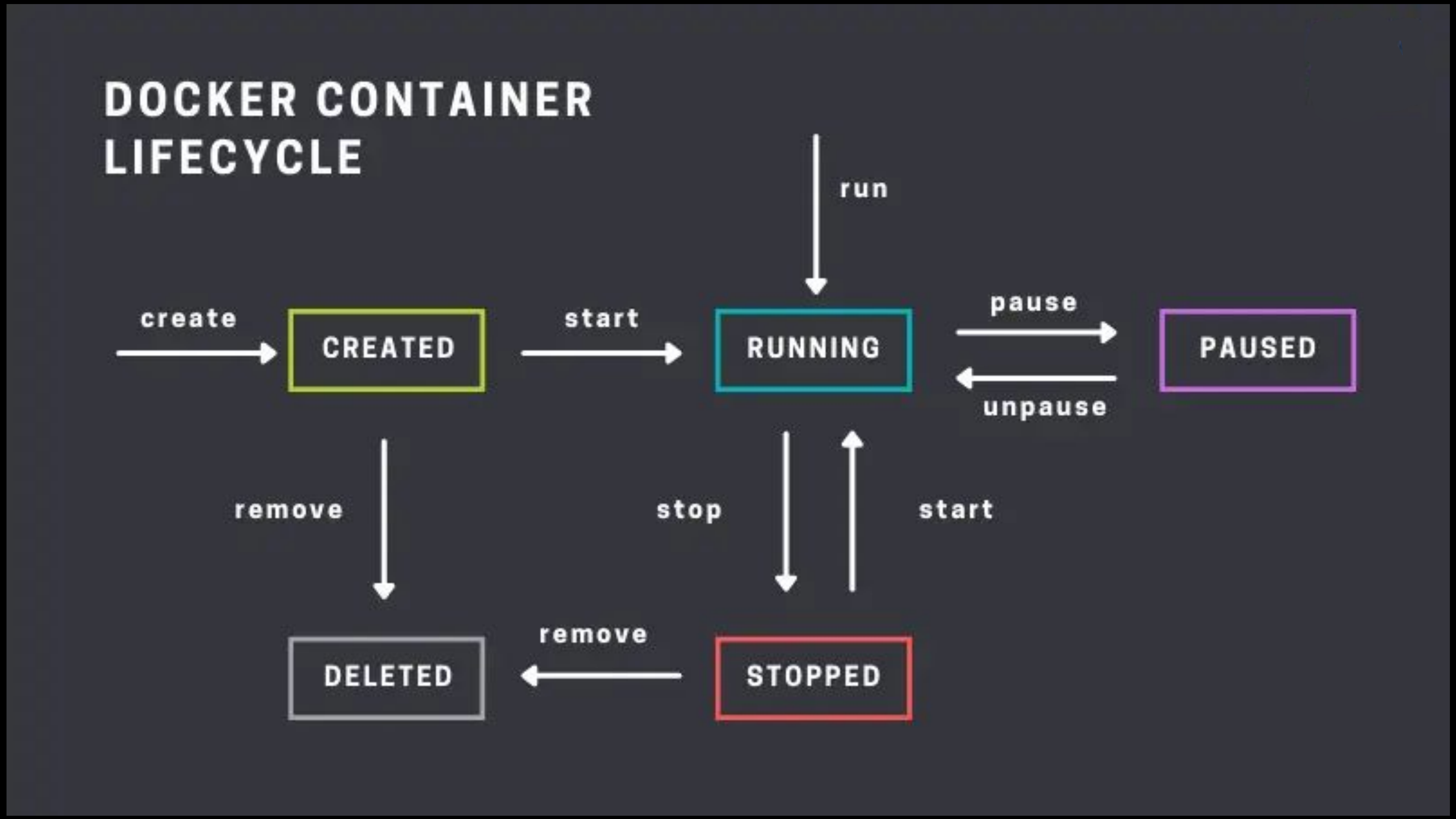
Basic



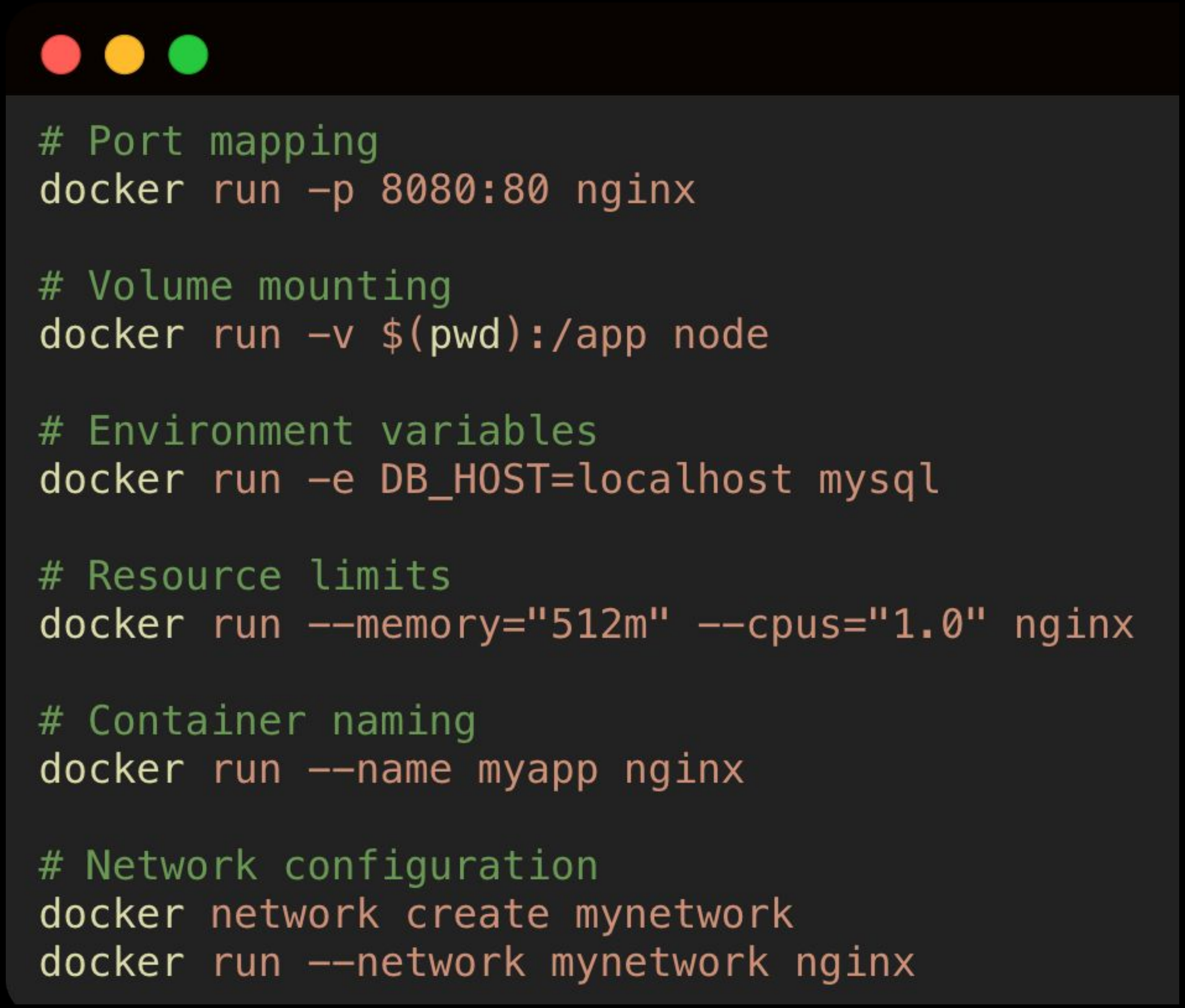
```
# List containers
docker ps
docker ps -a

# List images
docker images

# Pull image
docker pull ubuntu:latest

# Run container
docker run -it ubuntu bash

# Remove container
docker rm container_id

# Remove image
docker rmi image_id
```

Container Lifecycle

1. Created: docker create
2. Running: docker start/run
3. Paused: docker pause/unpause
4. Stopped: docker stop
5. Deleted: docker rm


DOCKER CONTAINER LIFECYCLE

Container Management

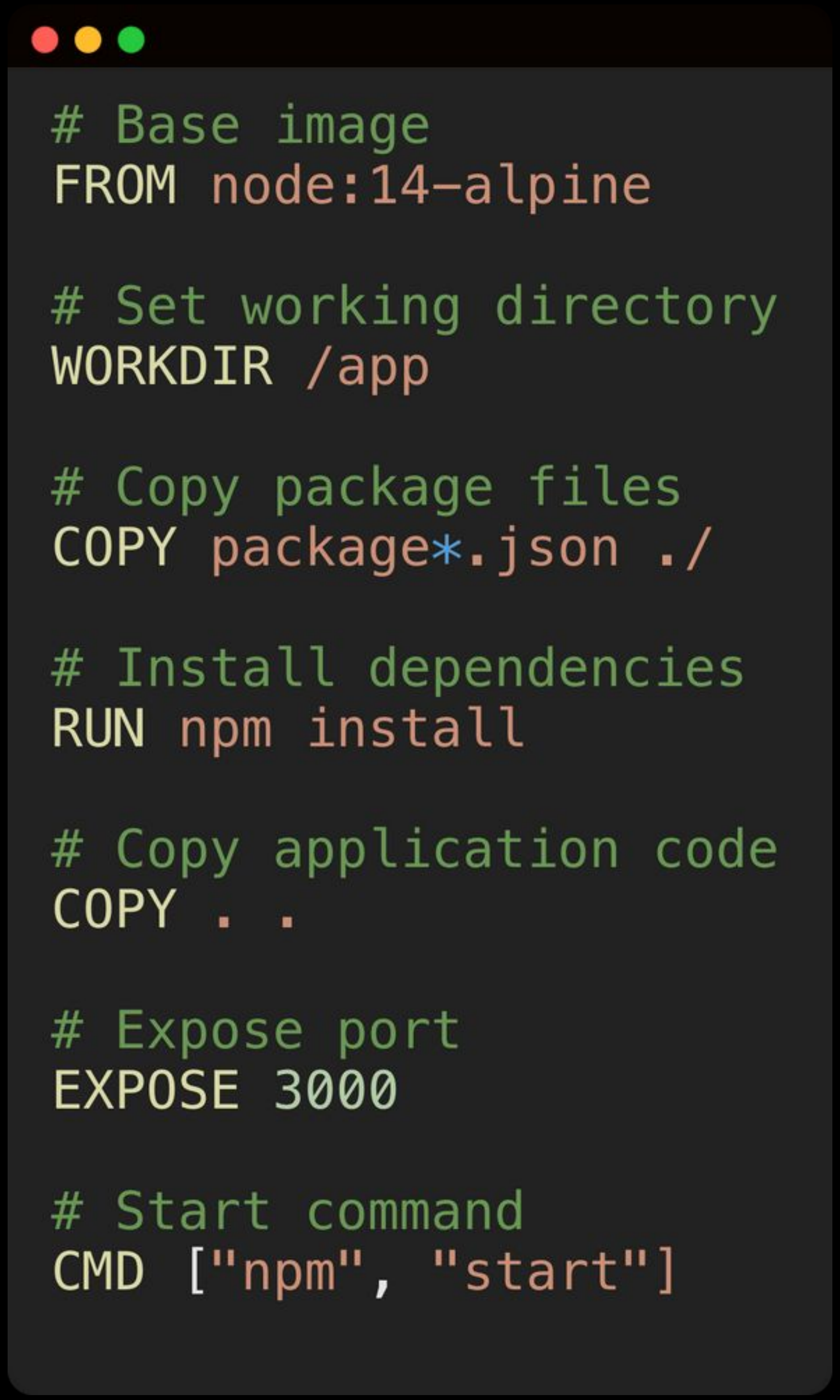


```
# Port mapping
docker run -p 8080:80 nginx

# Volume mounting
docker run -v $(pwd):/app node

# Environment variables
docker run -e DB_HOST=localhost mysql

# Resource limits
docker run --memory="512m" --cpus="1.0" nginx

# Container naming
docker run --name myapp nginx

# Network configuration
docker network create mynetwork
docker run --network mynetwork nginx
```

Docker Desktop

Docker Hub

Dockerfile Basics



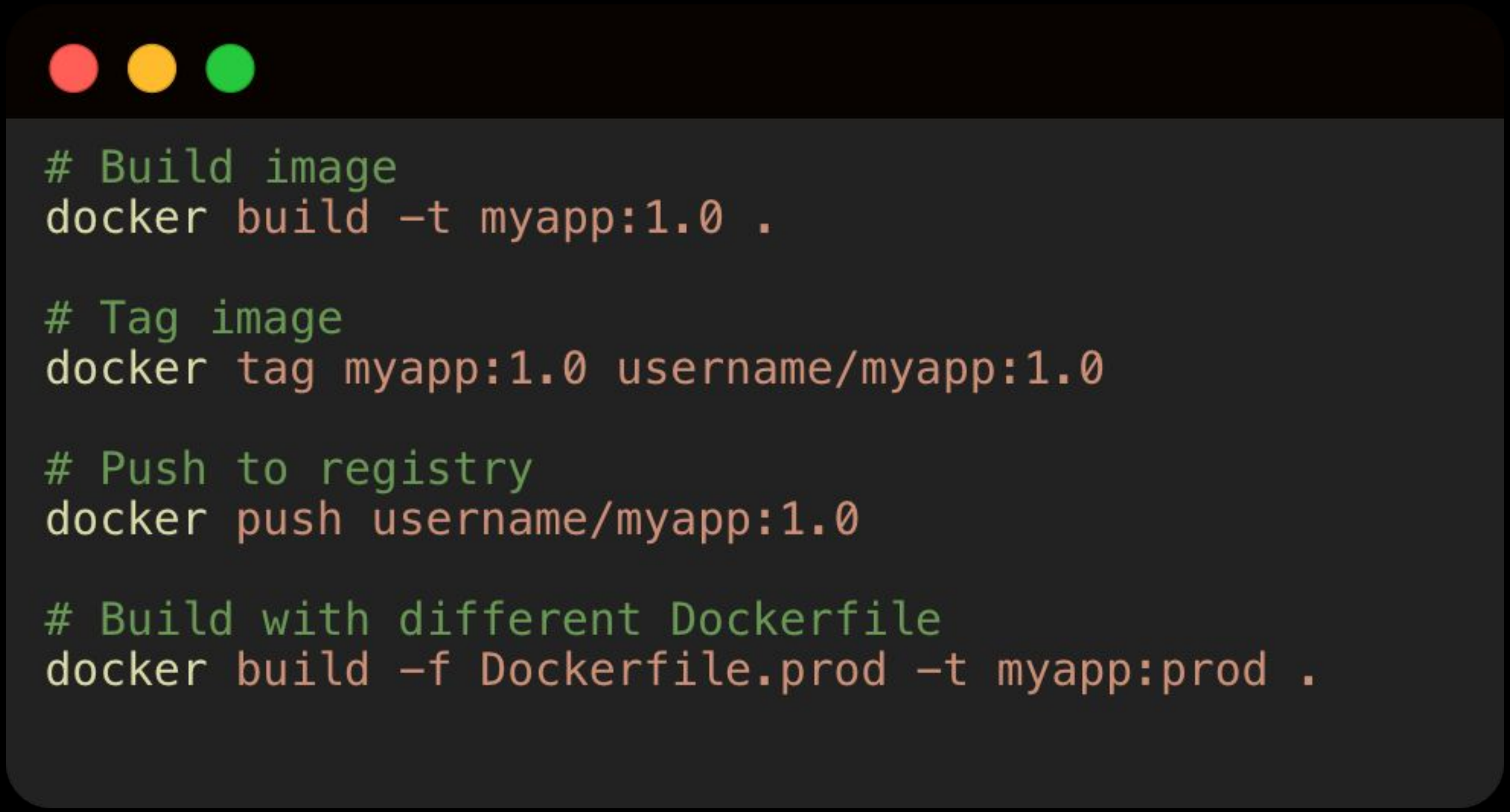
```dockerfile
# Base image
FROM node:14-alpine

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy application code
COPY . .

# Expose port
EXPOSE 3000

# Start command
CMD ["npm", "start"]
```

Dockerfile Basics

Best Practices:

- ✓ Use specific base image tags
- ✓ Minimize layers
- ✓ Use .dockerignore
- ✓ Security considerations

Building Images

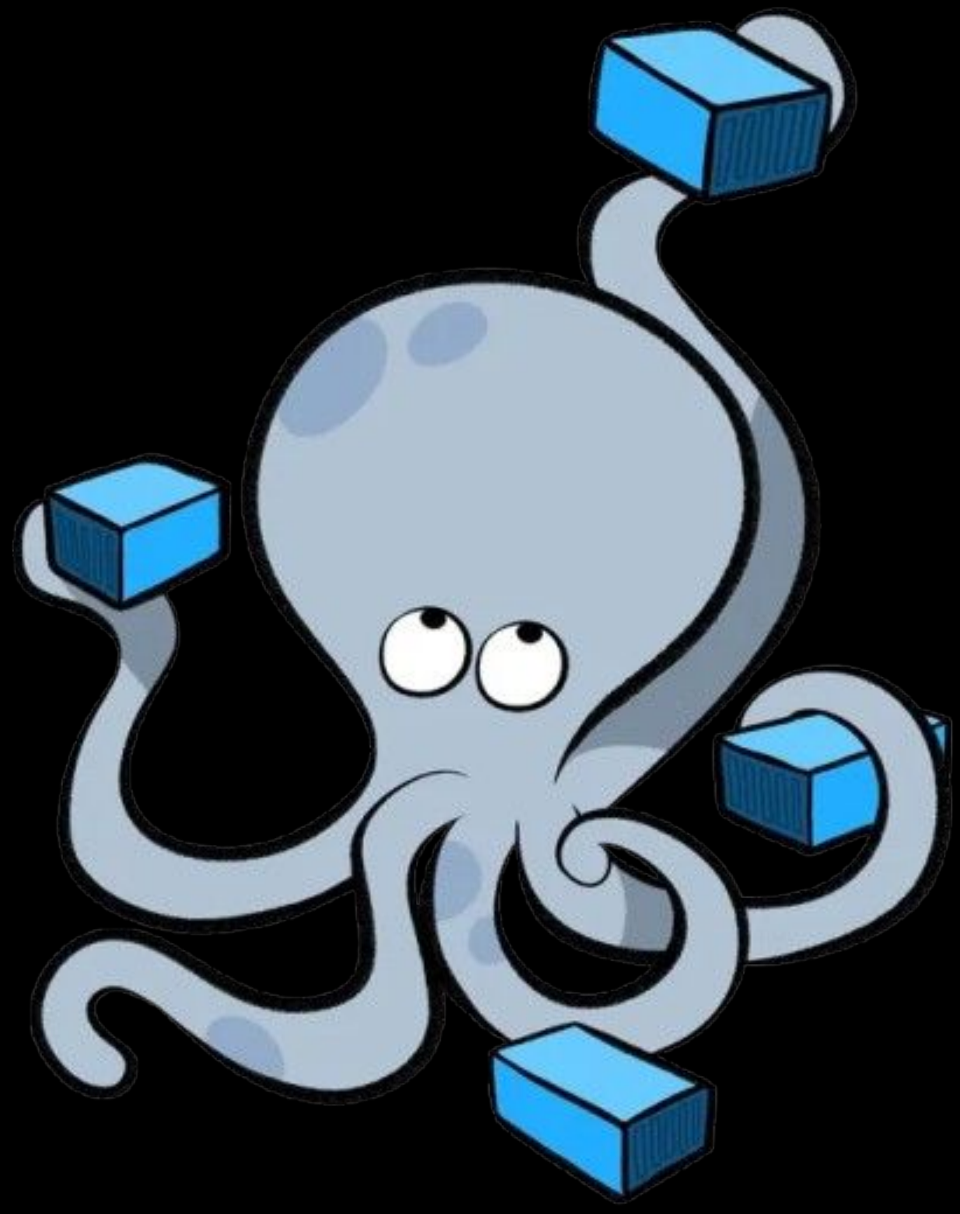


```
# Build image
docker build -t myapp:1.0 .

# Tag image
docker tag myapp:1.0 username/myapp:1.0

# Push to registry
docker push username/myapp:1.0

# Build with different Dockerfile
docker build -f Dockerfile.prod -t myapp:prod .
```

Docker Compose

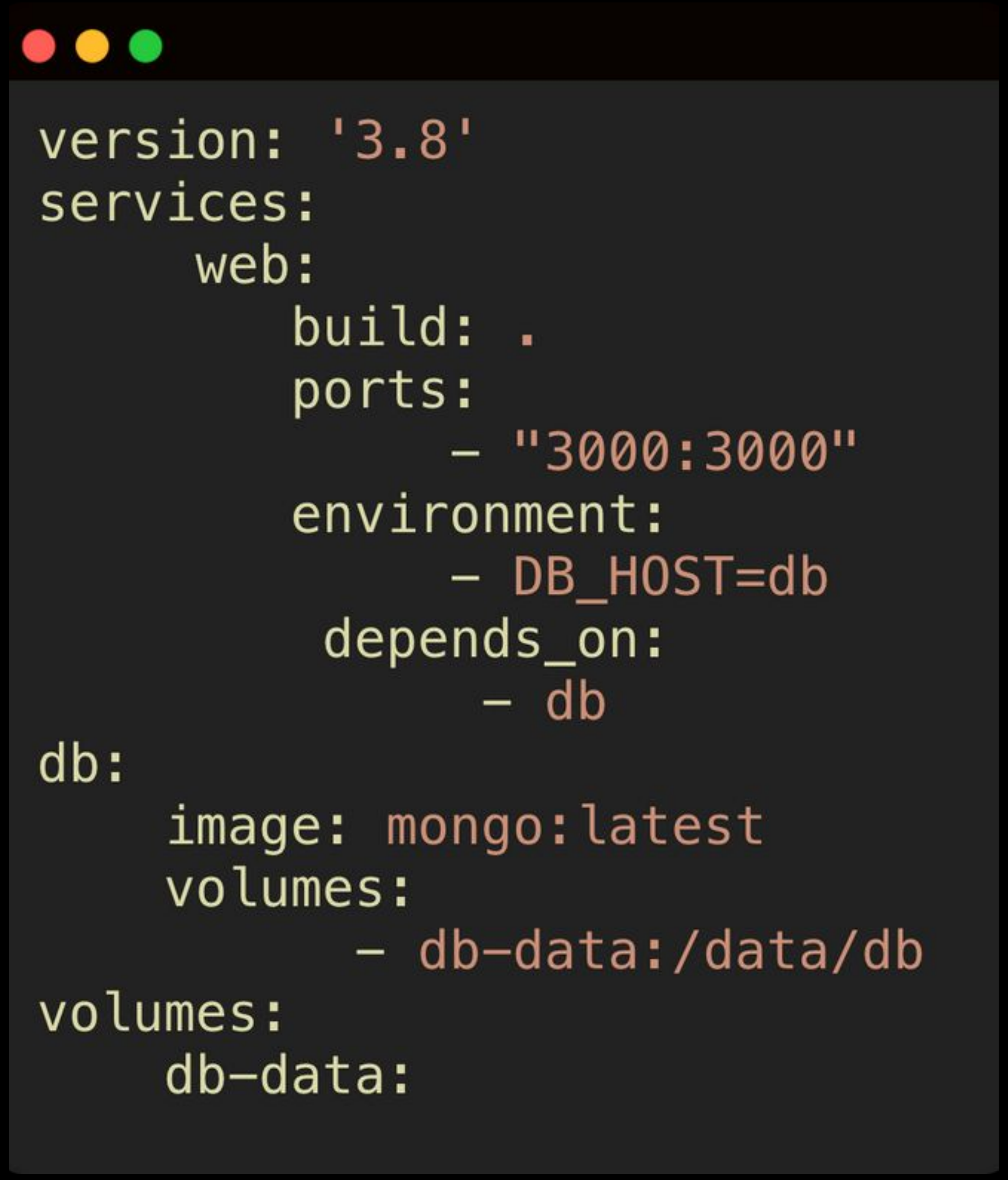
Purpose:

- Define multi-container applications
- Single source of truth for app configuration
- Simplified deployment



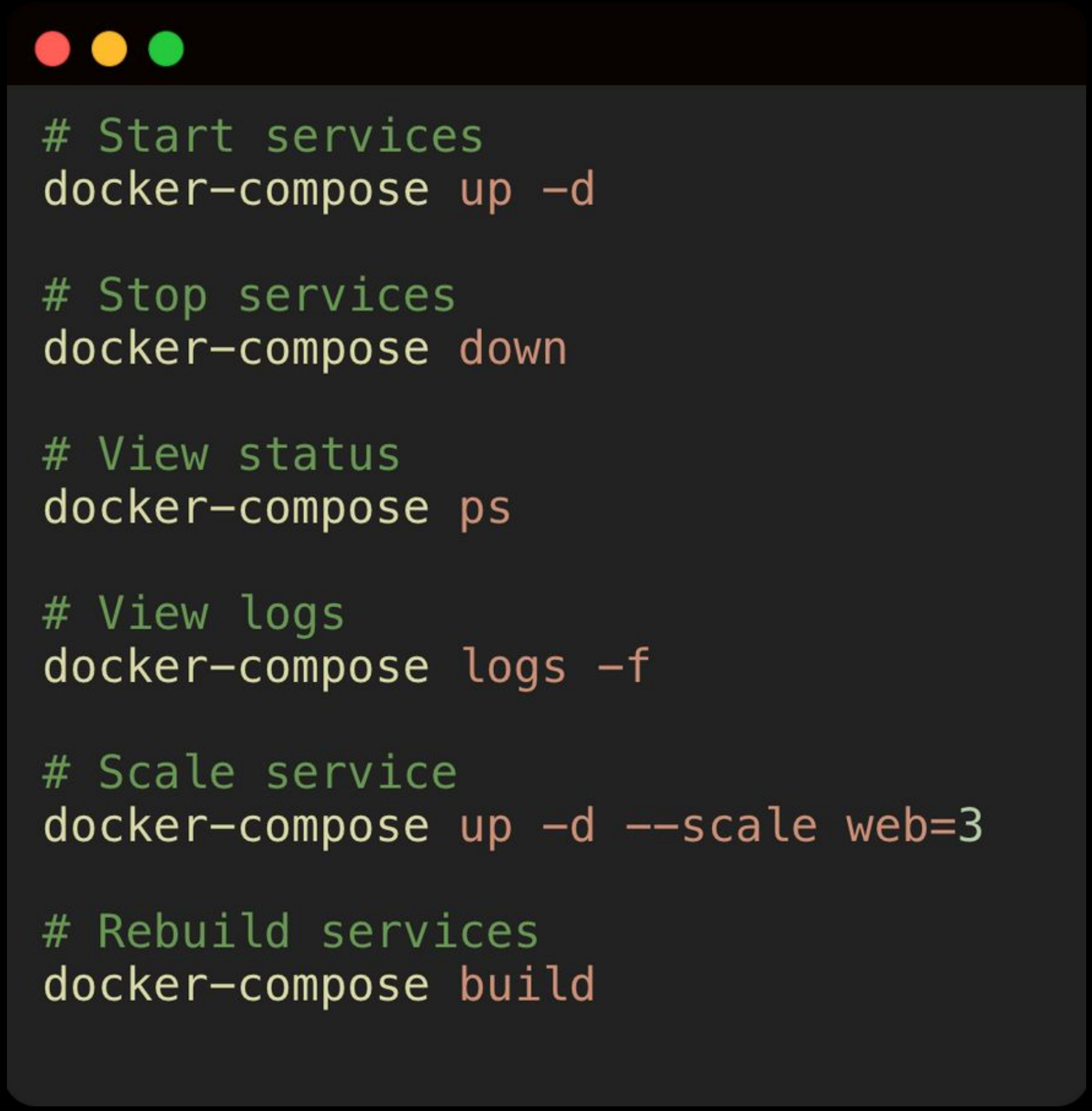
Docker Compose

Example docker-compose.yml



```
version: '3.8'
services:
     web:
        build: .
        ports:
             - "3000:3000"
        environment:
             - DB_HOST=db
         depends_on:
              - db
db:
    image: mongo:latest
    volumes:
          - db-data:/data/db
volumes:
    db-data:
```

Docker Compose Commands

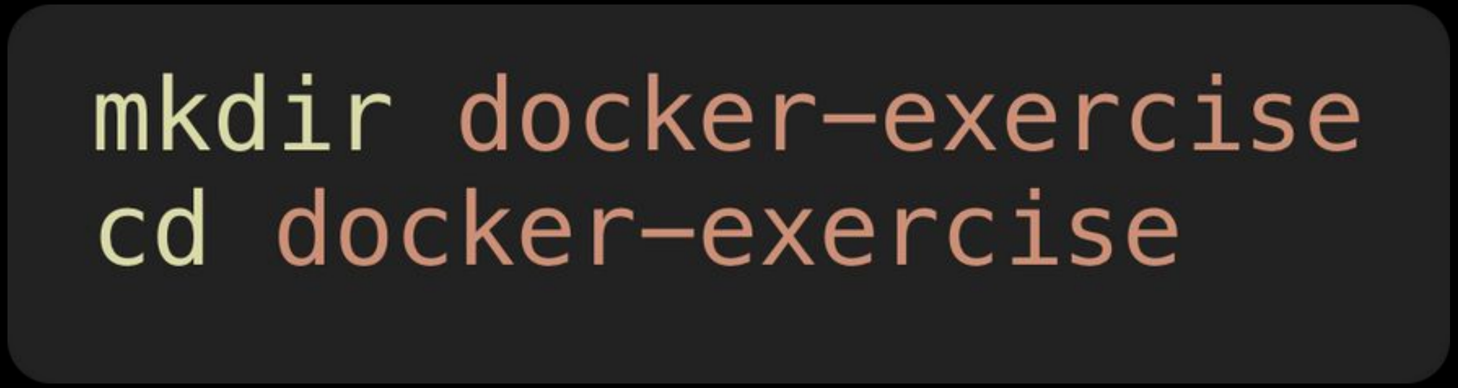


```
# Start services
docker-compose up -d

# Stop services
docker-compose down

# View status
docker-compose ps

# View logs
docker-compose logs -f

# Scale service
docker-compose up -d --scale web=3

# Rebuild services
docker-compose build
```

Exercise one

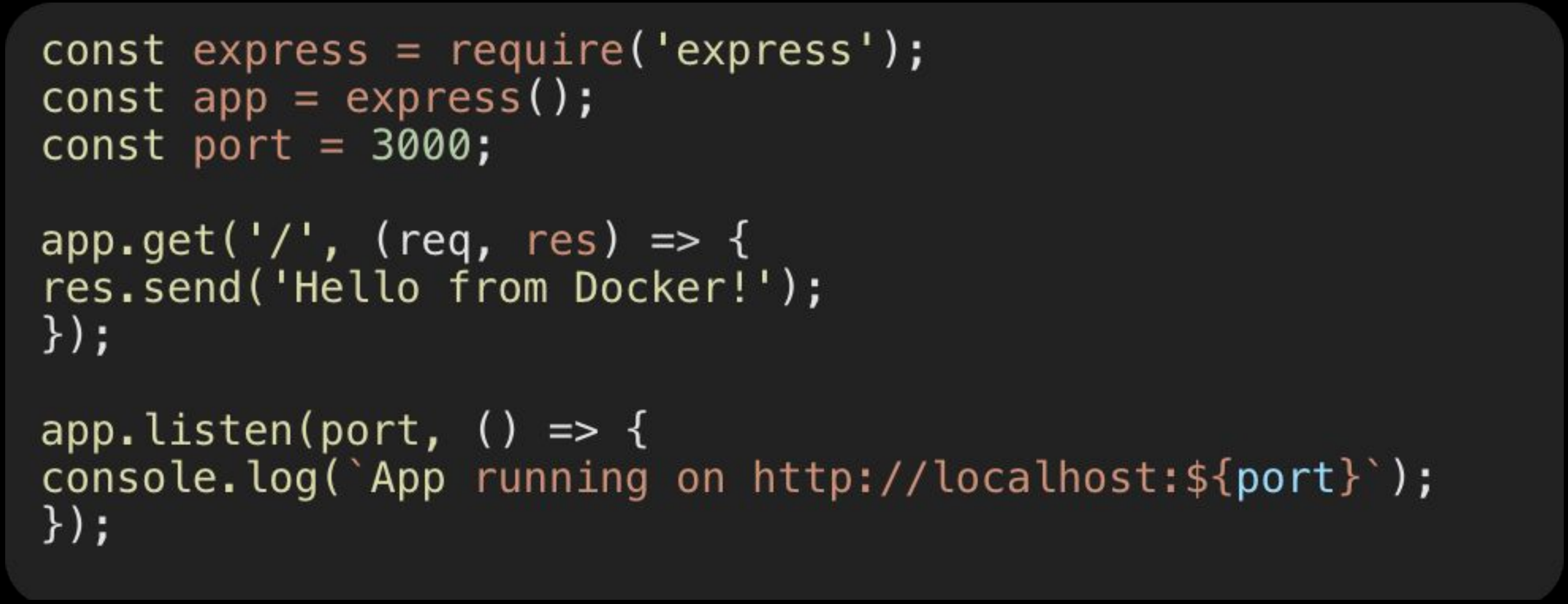
“Hello World” web application

1. Create project structure



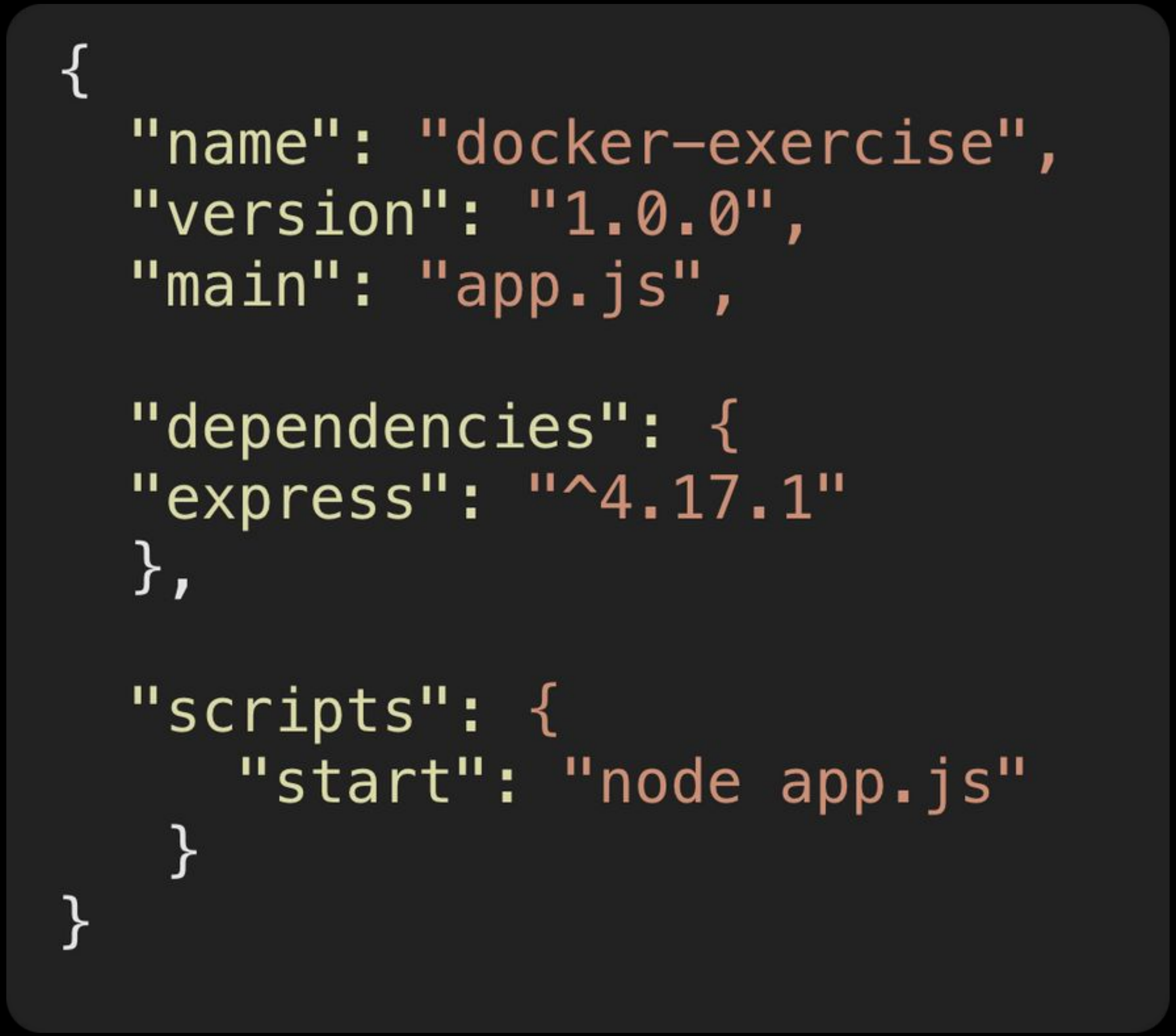
```
mkdir docker-exercise
cd docker-exercise
```

2. Create app.js



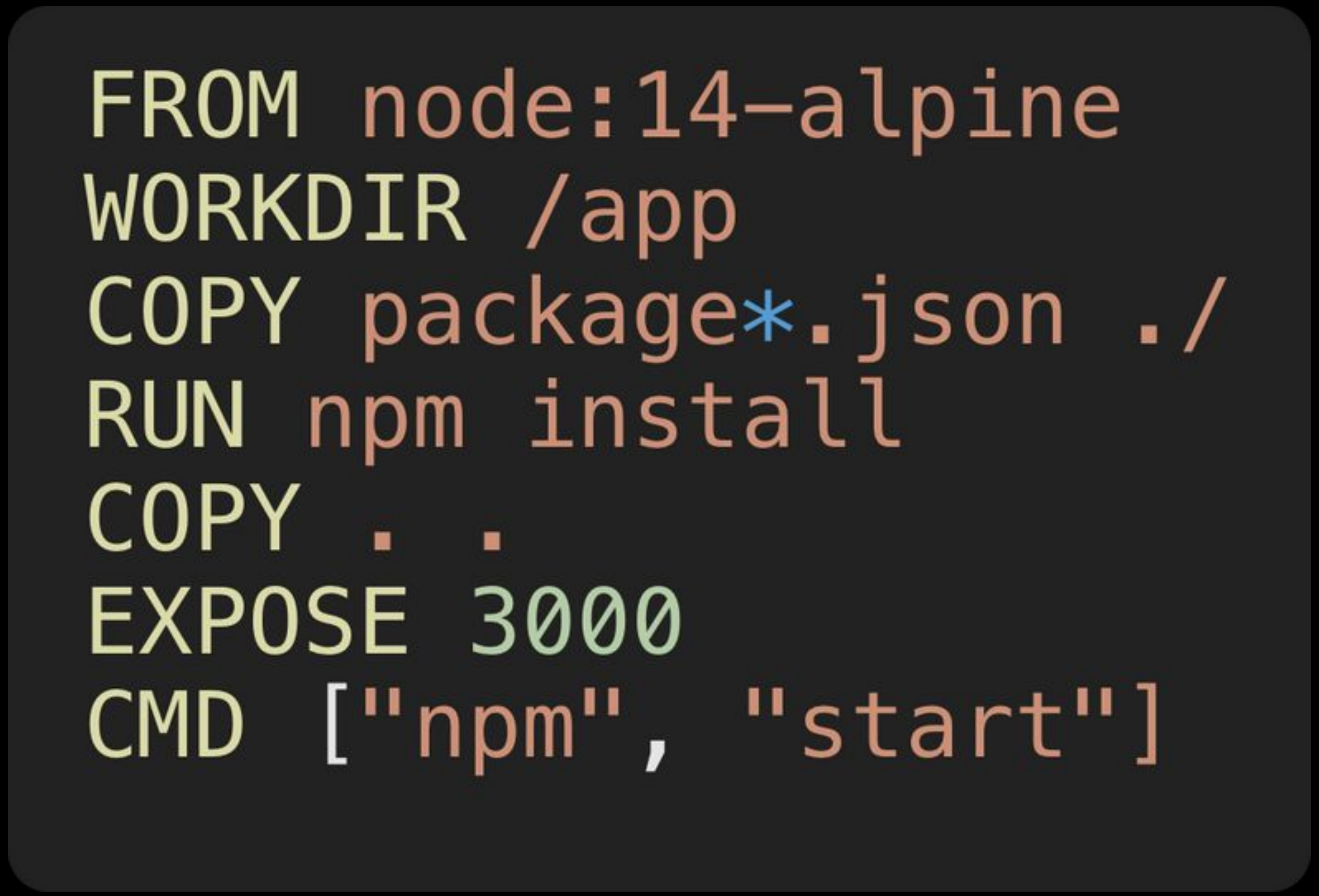
```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
res.send('Hello from Docker!');
});

app.listen(port, () => {
console.log(`App running on http://localhost:${port}`);
});
```

3. Create package.json



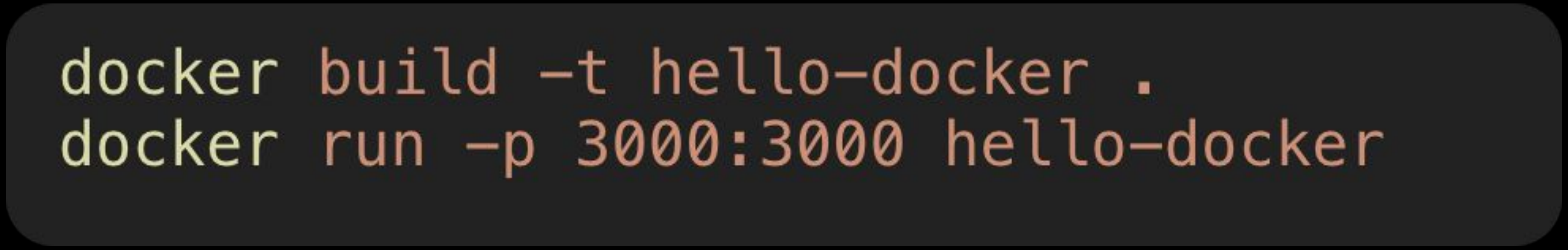
```
{
  "name": "docker-exercise",
  "version": "1.0.0",
  "main": "app.js",

  "dependencies": {
  "express": "^4.17.1"
  },

  "scripts": {
     "start": "node app.js"
   }
}
```

4. Create Dockerfile



```
FROM node:14-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

5. Create Dockerfile



```
docker build -t hello-docker .
docker run -p 3000:3000 hello-docker
```

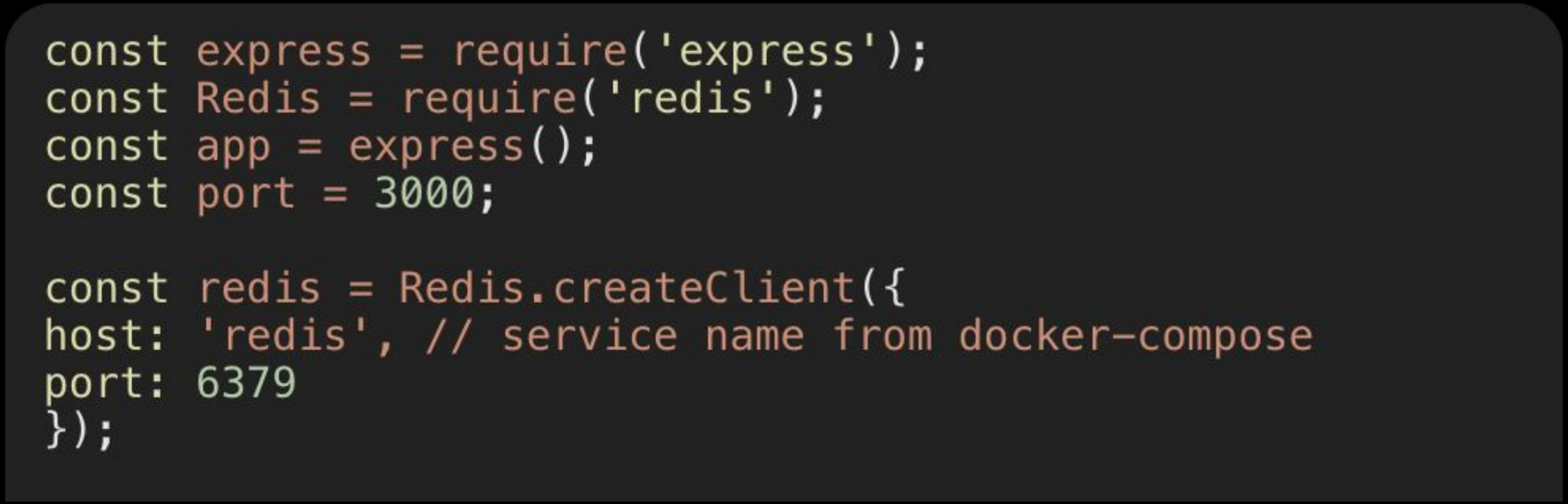
6. Test: Visit http://localhost:3000

Exercise two

Multi-container application

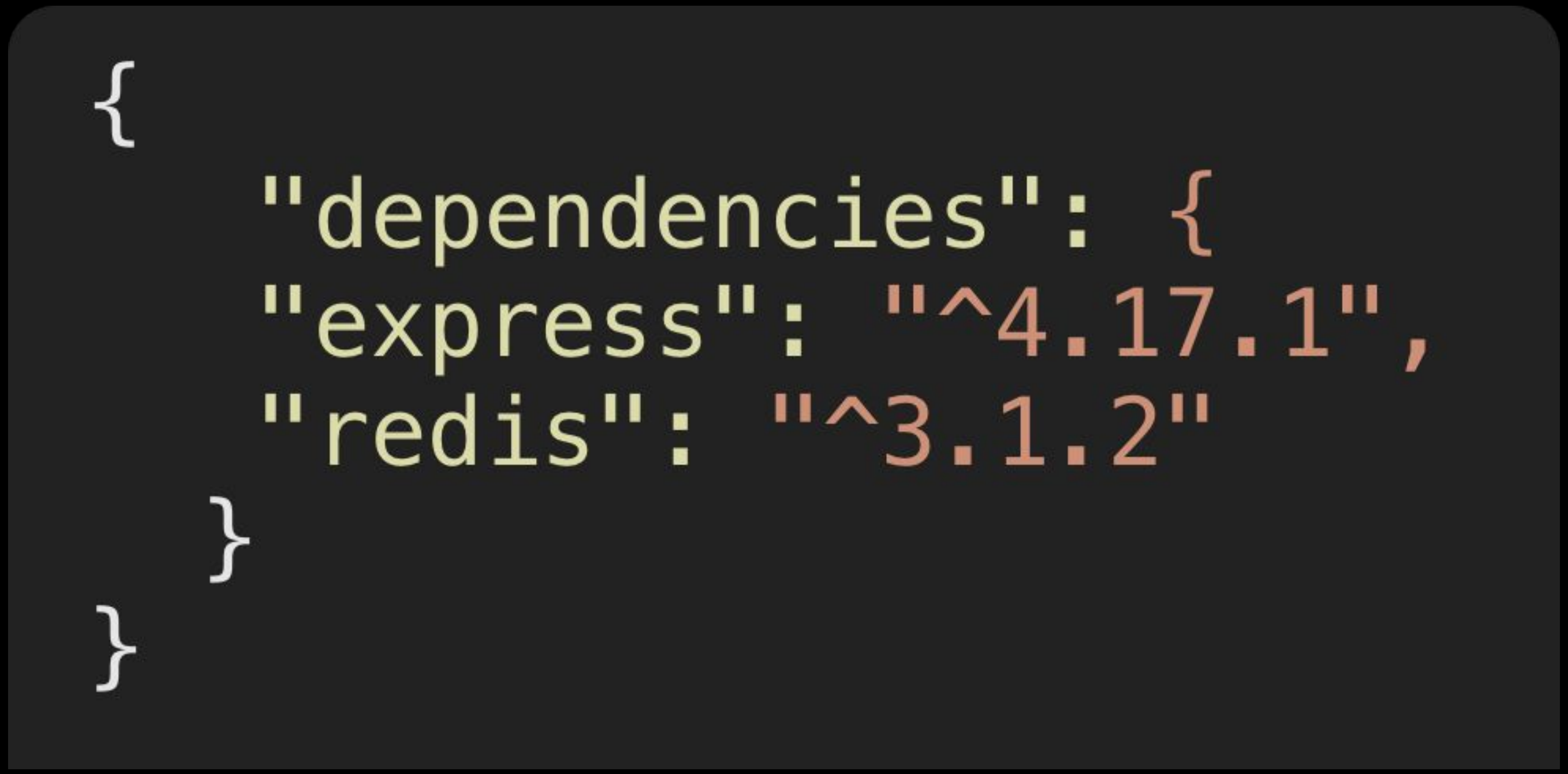
Objective: Create a Node.js application with Redis counter using Docker Compose

1. Update app.js



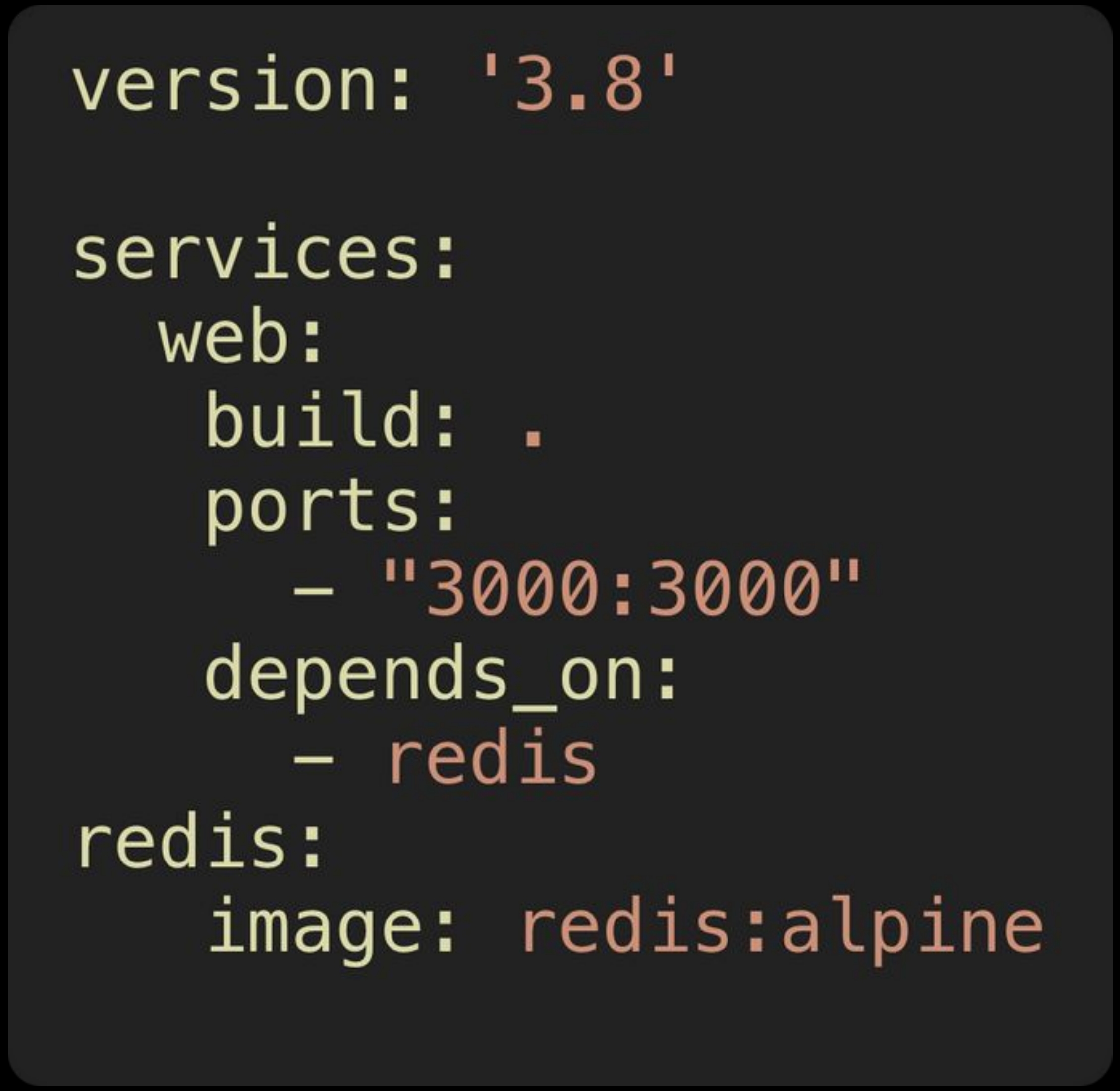
```
const express = require('express');
const Redis = require('redis');
const app = express();
const port = 3000;

const redis = Redis.createClient({
host: 'redis', // service name from docker-compose
port: 6379
});
```

2. Update package.json



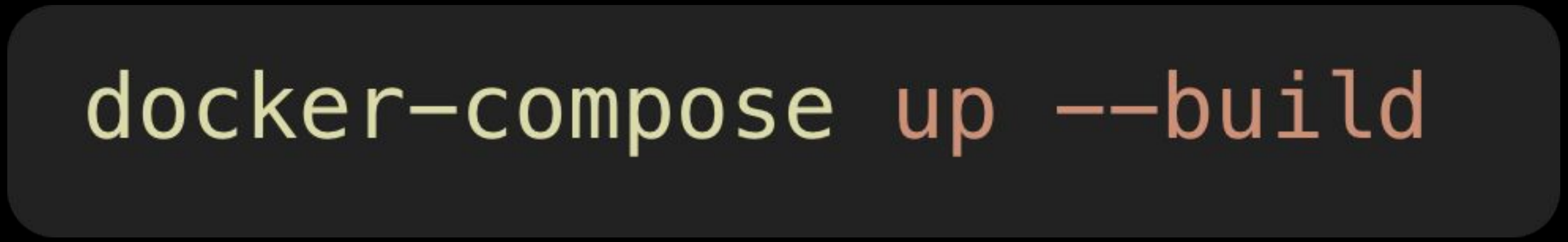
```json
{
   "dependencies": {
   "express": "^4.17.1",
   "redis": "^3.1.2"
  }
}
```

3. Create docker-compose.yml



```
version: '3.8'

services:
  web:
   build: .
   ports:
     - "3000:3000"
   depends_on:
     - redis
redis:
   image: redis:alpine
```

4. Build and run



```
docker-compose up --build
```

5. Build and run

- Visit http://localhost:3000 multiple times to see counter increase
- Try stopping and starting the containers to see data persistence