MODULE 7

Complete Mastery Guide

From Zero to

Advanced

DOCKER CONTAINERIZATION

Master the complete Docker ecosystem from fundamentals to advanced deployment strategies

Containers Orchestration DevOps

COURSE STRUCTURE

Learning Journey

01 **Fundamentals**

Understanding Docker, containers, VM comparisons, and core problems solved

02 **Installation & Setup**

Prerequisites, Windows/Mac/Linux installation, Docker Desktop setup

03 **Essential Commands**

Master CLI with pull, run, ps, stop, start, port mapping, logs, exec

04 **Building & Development**

Creating custom images, Dockerfiles, multi-container applications

05 **Advanced Concepts**

Networking, Compose, volumes, private repositories, deployment

06 **Interview & Exam Prep**

Q&A, MCQs, workflows, cheat sheets, and hands-on projects

Chapter One

01

FUNDAMENTALS

Understanding Docker and Containerization

FOUNDATION

What is Docker?

Containerization Platform

Docker is a **containerization platform** that enables you to package applications and their dependencies into isolated containers.

These containers can run **anywhere** - on your laptop, a colleague's machine, or in the cloud - without environment-related issues.

"Works on My Machine" Problem: SOLVED

Docker eliminates the classic developer dilemma where code works on one machine but fails on another due to environment differences.

Package

Dependencies

Run Anywhere

Core Capabilities

Package Applications

Bundle code + dependencies

Include Dependencies

Libraries, runtime, system tools

Run Anywhere

Consistent across environments

Isolate Environments

Prevent conflicts between apps

CORE CONCEPT

Understanding Containers

What is a Container?

A container is a **lightweight, isolated environment** that packages an application along with all its dependencies, libraries, and runtime.

- Lightweight
- Isolated
- Shares OS

Key Characteristics

- ✓ Contains application code + dependencies
- ✓ Shares the host OS kernel
- ✓ Faster startup than VMs
- ✓ Efficient resource utilization

Container vs Virtual Machine

Aspect	Container	VM
Weight	Lightweight	Heavy
Startup Speed	Fast (seconds)	Slow (minutes)
OS Sharing	Shares Host OS	Separate OS
Resource Usage	Efficient	High overhead
Isolation	Process-level	Hardware-level
Portability	High	Medium

VALUE PROPOSITION

Problems Docker Solves

Docker addresses critical challenges in modern software development and deployment

Environment Mismatch

Eliminates "works on my machine" by ensuring consistency across development, testing, and production environments.

Dependency Issues

Packages all dependencies with the application, preventing conflicts and version mismatches.

Deployment Inconsistency

Ensures identical deployments every time, reducing manual configuration errors and downtime.

Scaling Difficulty

Enables rapid horizontal scaling with lightweight containers that start in seconds, not minutes.

CI/CD Complexity

Simplifies continuous integration and deployment pipelines by providing consistent, reproducible builds across all stages of the development lifecycle.

DISTRIBUTION

Container Repository System

Where Do Containers Live?

Containers are created from **images**. These images are stored in centralized repositories called registries, making them accessible to developers and deployment systems worldwide.

Relationship: Image (blueprint) → Container (running instance)

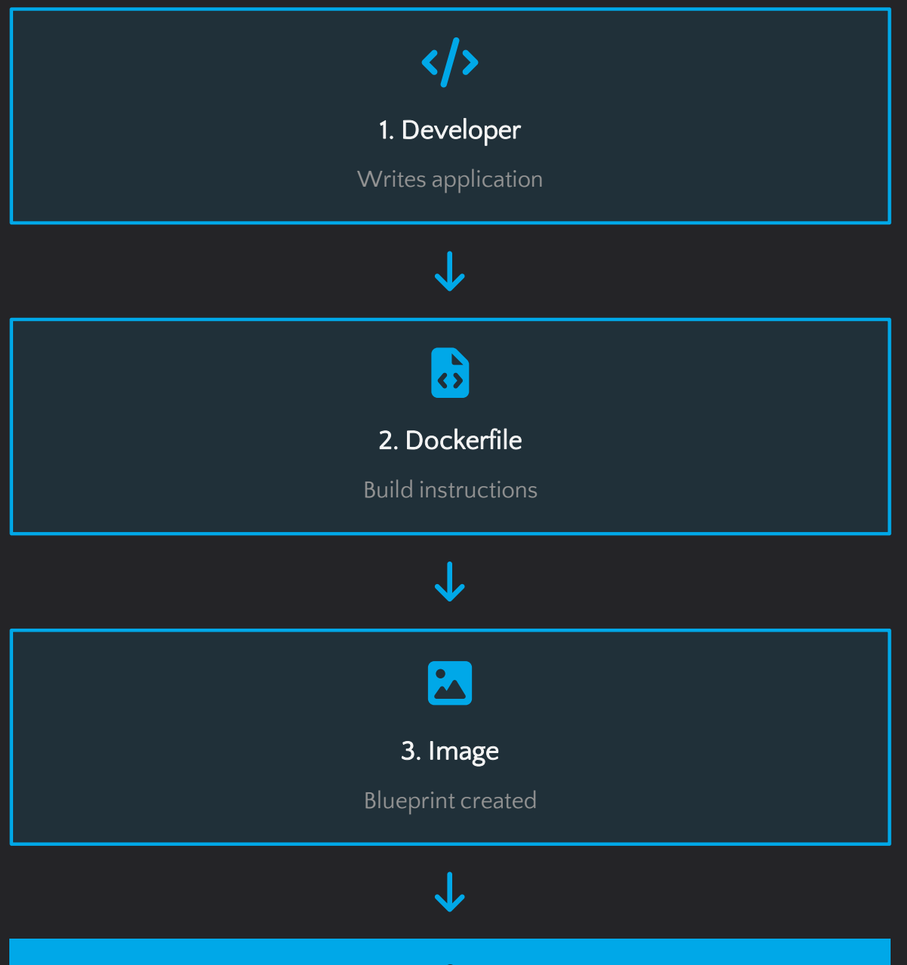
Registry Types

Docker Hub
Public registry with official images (nginx, ubuntu, python)

Private Registries
Self-hosted or enterprise solutions for proprietary images

Cloud Registries
AWS ECR, Google Container Registry, Azure Container Registry

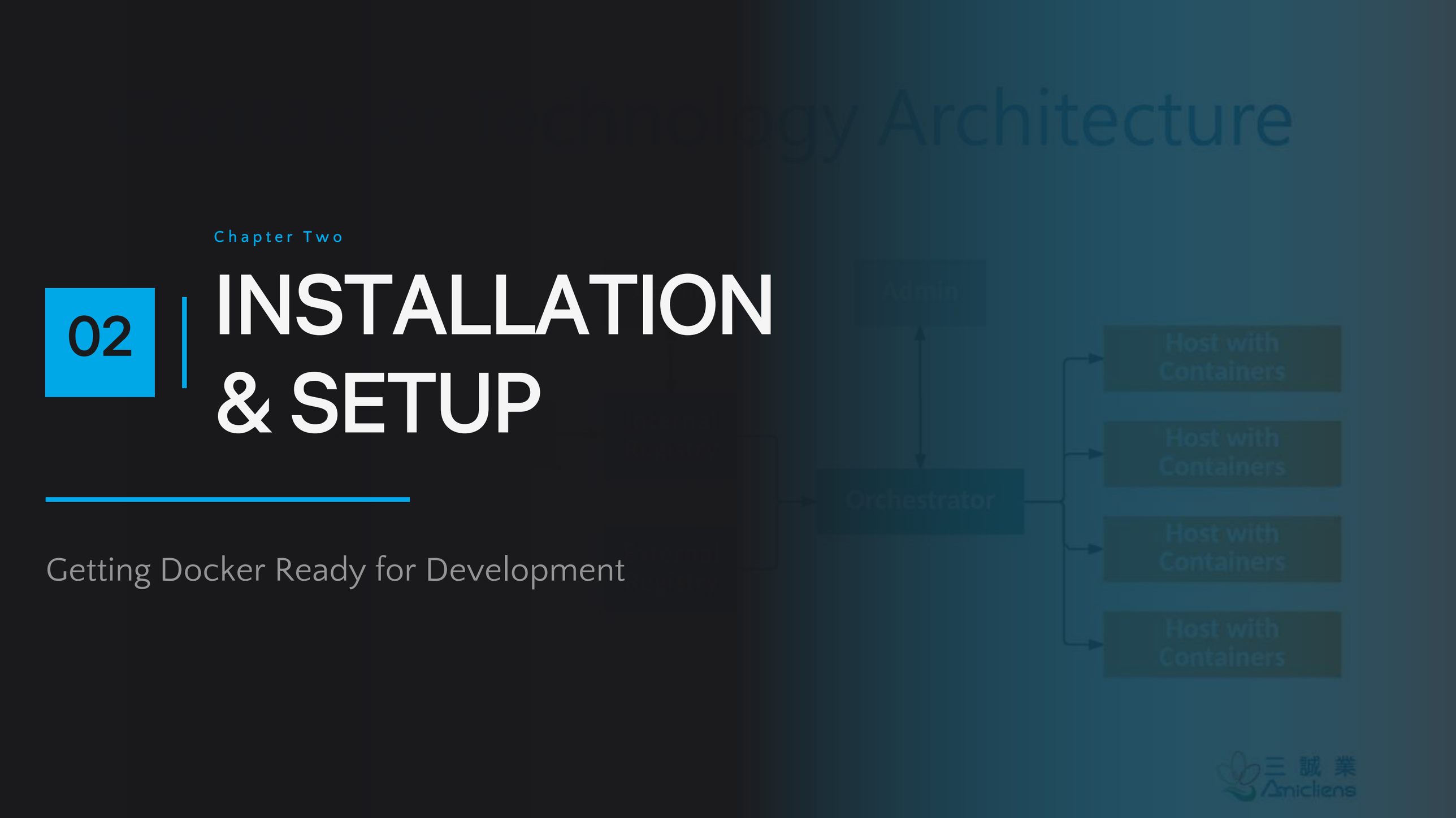
Image Flow



Chapter Two

02 | INSTALLATION & SETUP

Getting Docker Ready for Development



SETUP

Docker Installation Guide

Prerequisites

Before installing Docker, ensure your system meets these requirements:

64-bit Operating System
Windows 10/11, macOS 10.15+, or 64-bit Linux

Virtualization Enabled
BIOS/UEVT settings (VT-x, SVM, EPT)

Internet Connection
Required for downloading images and updates

Administrator Access
Required for installation and configuration

Windows / macOS

1. Visit docker.com
2. Download Docker Desktop
3. Run installer and follow prompts
4. Restart if required

Linux (Ubuntu/Debian)

Terminal Commands:

```
sudo apt update
sudo apt install docker.io
```

Verification

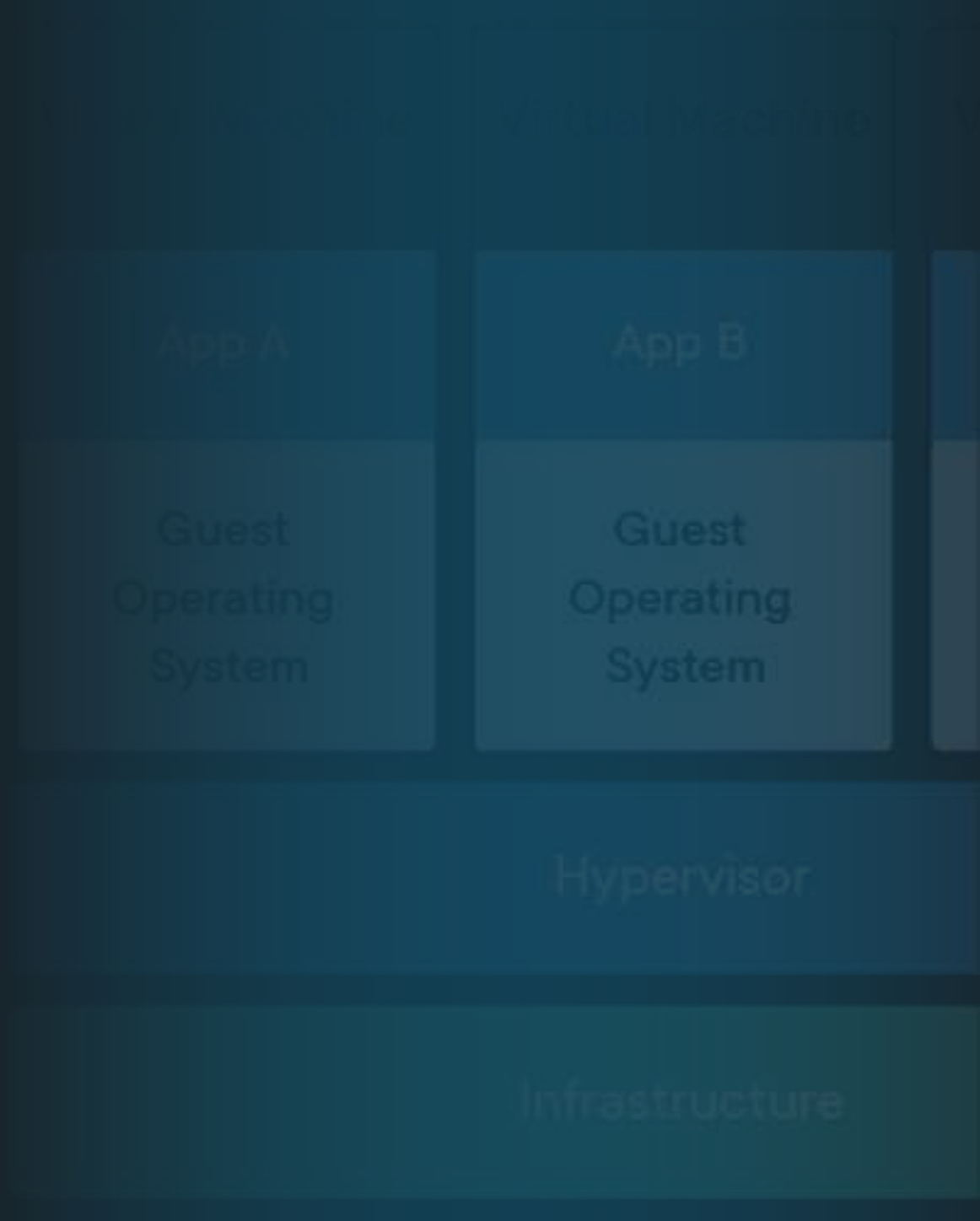
Confirm successful installation

```
docker --version
```

Chapter Three

03 | ESSENTIAL COMMANDS

Master Docker Command Line Interface



CORE COMMANDS

Main Docker Commands

docker pull

Download image from registry

Syntax:
`docker pull`

Example:
`docker pull nginx`

docker run

Create & start container

Foreground:
`docker run nginx`

Background (-d):
`docker run -d nginx`

docker ps

List containers

Running containers:
`docker ps`

All containers:
`docker ps -a`

docker stop / start

Control container state

Stop container:
`docker stop`

Start container:
`docker start`

Pro Tip: Use container IDs from `docker ps` with stop/start commands

ADVANCED COMMANDS

Port Mapping & Additional Commands

Port Mapping

Maps container port to host port for external access

Syntax:

```
docker run -p host:container
```

Example:

```
docker run -p 8080:80 nginx
```

Access: localhost:8080

docker logs

View container logs

```
docker logs
```

docker exec

Run command inside container

Interactive shell access:

```
docker exec -it bash
```

docker inspect

View container details

```
docker inspect
```

docker networks

List networks:

```
docker network ls
```

Create network:

```
docker network create mynet
```

-it flag: Interactive mode with TTY

Port mapping: Essential for web services

Chapter Four

04 BUILDING & DEV

Creating Custom Images and Applications

CUSTOM IMAGES

Building Docker Images

What is a Dockerfile?

A Dockerfile is a **text file** containing instructions to build a Docker image automatically.

- Text-based
- Version controlled
- Reproducible
- Automated builds

Dockerfile Instructions

FROM	Base image
WORKDIR	Working directory
COPY	Copy files
RUN	Execute commands
CMD	Default command

Sample Dockerfile (Flask App)

```
# Use Python 3.10 base imageFROM python:3.10 # Set working directoryWORKDIR /app # Copy all filesCOPY . . # Install FlaskRUN pip install flask # Run appCMD ["python", "app.py"]
```

Build Image

Command (run in Dockerfile directory):

```
docker build -t flask-app .
```

-t : Tag/name

. : Build context

REAL-WORLD APPLICATION

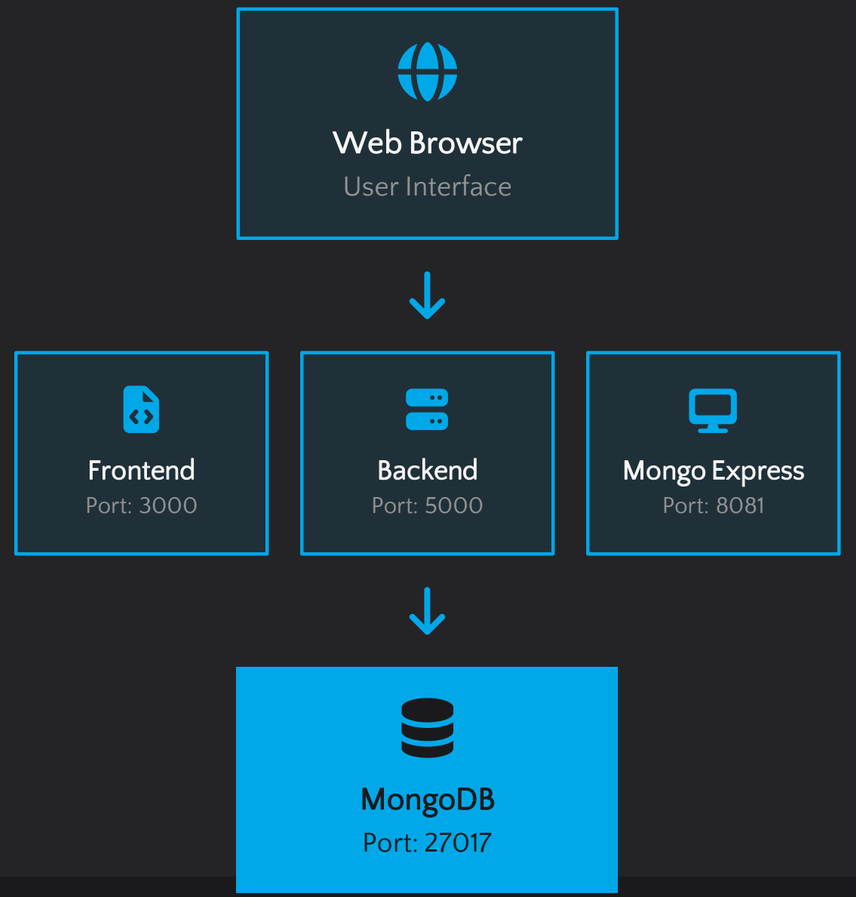
Developing with Containers

Multi-Container Architecture

Real applications typically consist of **multiple services** running in separate containers that communicate with each other.

- **Frontend (HTML/Node.js)**
  User interface layer
- **Backend (Flask)**
  API logic and business rules
- **Database (MongoDB)**
  Data persistence layer
- **Mongo Express UI**
  Database management interface

Architecture Diagram



Best Practice: Each service runs in its own isolated container with specific configurations

DATABASE CONTAINERS

MongoDB & Mongo Express with Docker

MongoDB Container

Run MongoDB database in a container with persistent data storage

Command:

```
docker run -d --name mongo mongo
```

- **-d:** Runs in background
- **--name:** Names container "mongo"
- **mongo:** Latest MongoDB image
- **Port:** 27017 (default)

Mongo Express UI

Web-based MongoDB admin interface for database management

Command:

```
docker run -d -p 8081:8081 mongo-express
```

- **-p 8081:8081:** Maps host port
- **Mongo Express:** Admin UI image
- **Access:** localhost:8081

Auto-Connection

Mongo Express auto-detects MongoDB container

No Configuration

Works out-of-the-box with default settings

Visual Management

Browser-based database administration

Chapter Five

05 | ADVANCED CONCEPTS

Docker Networks, Compose, and Volumes

NETWORKING

Docker Networking

Docker networking enables containers to communicate with each other and the outside world

What is Docker Network?

Docker networking allows containers to **communicate using container names** instead of IP addresses, providing isolation from the host network.

- Container communication
- Network isolation

Network Commands

List all networks:

```
docker network ls
```

Create network:

```
docker network create mynetwork
```

Run container on network:

```
docker run --network mynetwork ...
```

Example Workflow

1. Create network: `docker network create app-network`
2. Run containers on same network

ORCHESTRATION

Docker Compose Mastery

What is Docker Compose?

Docker Compose is a tool used to **define and run multiple containers** using a single YAML file.

- Single YAML file
- Multi-service
- One command
- Auto-networking

Essential Commands

Start all services:

```
docker-compose up
```

Stop all services:

```
docker-compose down
```

Build and start:

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

docker-compose.yml Example

```
# Docker Compose versionversion: "3" # Services definitionservices: # Web serviceweb: image: nginx ports: - "8080:80" # Database servicemongo: image: mongo
```

Key Components

- **version:** Compose file format
- **services:** Individual containers
- **image:** Base Docker image
- **ports:** Port mapping

DATA MANAGEMENT

Docker Volumes & Data Persistence

Why Volumes?

Containers are **temporary**. When a container stops or is removed, all data inside is lost.

Problem: Data loss on container restart

Solution: Volumes store data permanently

What is Docker Volume?

A volume is a **storage mechanism** managed by Docker for persisting data generated by containers.

Volume Types

1. Named Volume
Managed by Docker with custom name

2. Anonymous Volume
Auto-generated name by Docker

3. Bind Mount
Maps host directory to container

Volume Commands

List volumes:
`docker volume ls`

Create volume:
`docker volume create mydata`

Usage Example

Run with named volume:
`docker run -v data:/data mongo`

- ✓ Data persists after restart
- ✓ Shared across containers
- ✓ Managed by Docker

Best Practice: Use named volumes for databases and stateful applications

DISTRIBUTION

Private Docker Repository

Docker Login

Authenticate with Docker registry to push and pull private images

Command:

```
docker login
```

- Enter username
- Enter password

Docker Tag

Name your image for registry

```
docker tag flask-app username/flask-app
```

Docker Push

Upload image to registry

```
docker push username/flask-app
```

Can be private repository

Workflow Summary

1. **Build Image**
   Create application image
2. **Tag Image**
   Name with registry
3. **Login**
   Authenticate
4. **Push**
   Upload to registry

PRODUCTION

Deploying Containerized Applications

Deployment Options

Virtual Machine
Traditional server deployment

Cloud Server
AWS EC2, Google Cloud, Azure

Kubernetes
Container orchestration platform

Docker Swarm
Native Docker clustering

Deployment Workflow

1. **Build Image**
   Create application image
2. **Push to Registry**
   Upload to Docker Hub
3. **Pull on Server**
   Download on production
4. **Run Container**
   Start application

Chapter Six

06

INTERVIEW & EXAM

Quick Reference for Success

INTERVIEW PREP

Docker Interview Q&A

1. What is Docker?

Docker is a containerization platform used to package applications with dependencies and run them anywhere consistently.

2. What is a Docker container?

A container is a lightweight, isolated runtime environment created from a Docker image.

3. Difference between Docker and VM?

Docker shares the host OS kernel and is lightweight, while VMs run a full OS and are heavy.

4. What is a Docker image?

A Docker image is a blueprint used to create containers.

5. What is Dockerfile?

A text file that contains instructions to build a Docker image.

6. What is Docker Hub?

Docker Hub is a container registry used to store and share Docker images.

7. What is Docker Compose?

A tool to run multiple containers using a single YAML file.

8. Why use Docker volumes?

Volumes are used to persist data even after a container stops or restarts.

9. What is Docker networking?

Docker networking allows containers to communicate with each other securely.

10. Docker in DevOps?

Used for CI/CD pipelines, application deployment, scaling, and environment consistency.

QUICK QUIZ

Docker MCQs with Answers

1. Docker is a:

B) Container platform ✓

2. Which file builds an image?

C) Dockerfile ✓

3. Which command runs a container?

B) docker run ✓

4. Command to list running containers?

B) docker ps ✓

5. Which option maps ports?

C) -p ✓

6. Docker Compose file extension?

C) .yml ✓

7. Which command builds image?

C) docker build ✓

8. Data persistence is achieved using?

C) Volumes ✓

INDUSTRY PRACTICE

Real DevOps Docker Workflow

Standard Production Workflow

1. Developer writes code
2. Create Dockerfile
3. Build image

```
docker build -t myapp .
```

4. Push image to Docker Hub

```
docker push username/myapp
```

5. CI/CD pipeline pulls image
6. Container deployed on server / Kubernetes
7. Monitoring & scaling

Tools Integration

Jenkins
CI/CD automation

GitHub Actions
Workflow automation

Kubernetes
Container orchestration

Benefits

- ✓ Automated deployments
- ✓ Consistent environments
- ✓ Scalable infrastructure
- ✓ Faster release cycles

REFERENCE

Docker Commands Cheat Sheet

Basics

Version
```
docker --version
```

Info
```
docker info
```

Images

List
```
docker images
```

Pull
```
docker pull nginx
```

Build
```
docker build -t myapp .
```

Remove
```
docker rmi image-id
```

Containers

Run
```
docker run -d nginx
```

List
```
docker ps
```

Stop
```
docker stop container-id
```

Remove
```
docker rm container-id
```

Networks

List
```
docker network ls
```

Create
```
docker network create mynet
```

Port Mapping

```
docker run -p 8080:80 nginx
```

Logs & Exec

```
docker logs container-id
```

```
docker exec -it container-id bash
```

Volumes

```
docker run -v data:/data mongo
```

Docker Compose

Start services:
```
docker-compose up
```

Stop services:
```
docker-compose down
```

HANDS-ON

Mini Project: Flask + MongoDB

Project Structure

```
docker-flask-mongo/ ├── app.py ├── Dockerfile ├── requirements.txt └── docker-compose.yml
```

Flask App (app.py)

```
# Import required librariesfrom flask import Flask, jsonify from pymongo import MongoClient # Create Flask app app = Flask(__name__) # MongoDB connection client = MongoClient("mongo", 27017) # Routes and logic @app.route("/") def home(): return jsonify({"message": "Data inserted"}) if __name__ == "__main__": app.run(host="0.0.0.0", port=5000)
```

Dockerfile

```
FROM python:3.10 WORKDIR /app COPY requirements.txt . RUN pip install -r requirements.txt COPY . . CMD ["python", "app.py"]
```

docker-compose.yml

```
version: "3" services: web: build: . ports: - "5000:5000" mongo: image: mongo volumes: - mongo-data:/data/db volumes: mongo-data:
```

Run Project

SUMMARY

Final Quick Revision

Docker

Container platform for packaging and running applications consistently across environments

Image

Blueprint or template used to create containers. Contains application code and dependencies

Container

Running instance of an image. Lightweight, isolated environment with its own filesystem

Dockerfile

Build instructions for creating Docker images. Defines base image, dependencies, and commands

Compose

Tool for defining and running multi-service applications with a single YAML file

Volumes

Persistent storage mechanism for containers. Stores data beyond container lifecycle

DevOps Backbone

Docker is the foundation of modern DevOps practices, enabling consistent, scalable, and automated software delivery pipelines

Master Docker,
Master DevOps

You now have the knowledge to containerize any application.

From zero to advance – you're ready for real-world Docker challenges.

Containerize **Deploy** **Scale**

Docker Mastery Complete