Jeremiah B. Dela Vega
IV-BCSAD
ELECTIVE 3 - Assignment # 3

STUDY PLAN: Docker and Containerization

Overview

This study plan will help you:

- Understand what Docker and containerization are
- Learn their history and why they matter
- Practice the actual workflow (building, running, and deploying containers)
- Follow best practices for using Docker in real projects

Estimated total duration: 2–3 weeks (adjust as needed)

WEEK 1: Foundations of Docker and Containerization

Day 1: What is Containerization?
Goal: Understand what "containerization" means and why developers use it.

Concept	Explanation	Example
Virtualization	Running multiple OSes on one machine using virtual machines (VMs).	Example: Running Windows and Linux together using VirtualBox.
Containerization	Running multiple isolated applications using the same OS kernel.	Docker lets you run isolated apps without full OS overhead.
Container	A lightweight, standalone package with everything needed to run a piece of software.	A "container" for a Python web app has the code, Python runtime, and dependencies.

Key takeaway:
Containers are like lightweight virtual machines but faster, smaller, and easier to move.

Day 2: History of Docker and Containerization

Year	Event	Description
1979	chroot command (Unix)	Allowed creating isolated file systems, a primitive form of containers.
2000s	FreeBSD Jails & LXC (Linux Containers)	Provided OS-level isolation and process separation.
2013	Docker released	Made containerization simple, portable, and developer-friendly.
2017–2020	Kubernetes + Docker combo	Became the standard for scaling containers in production.
2022+	Containerd, Podman, Docker Desktop	Modern tools now offer different ways to manage containers.

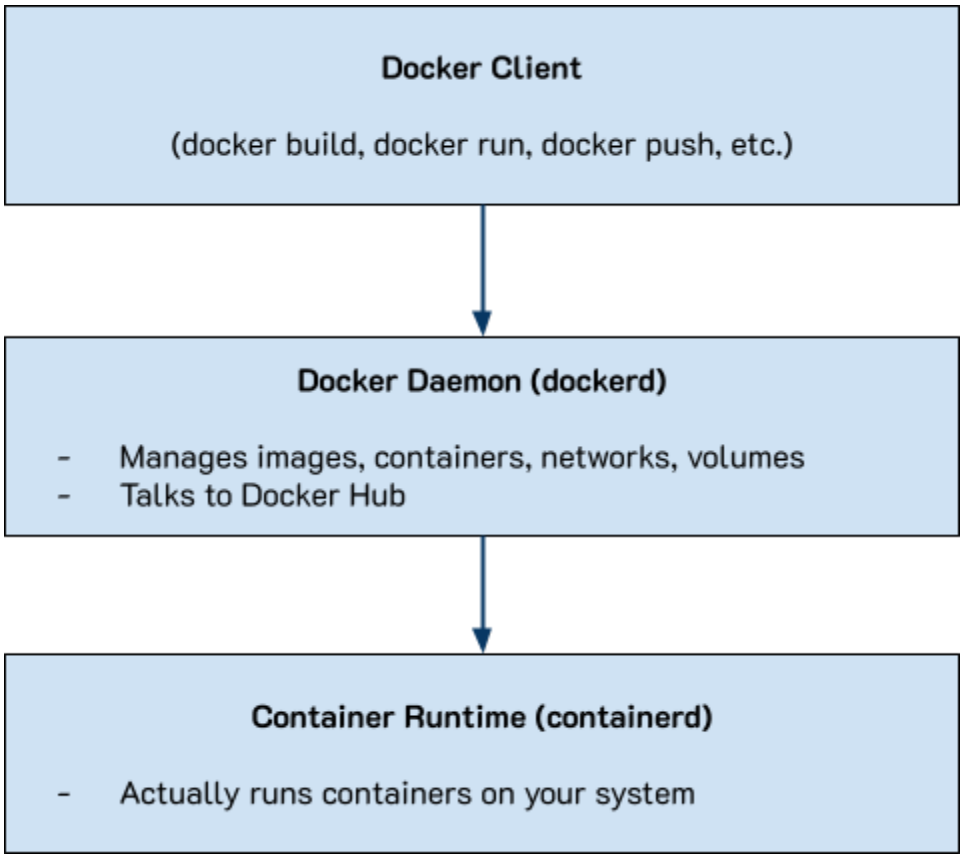
Docker popularized containers by making them easier to build, share, and deploy anywhere.

Day 3-4: Installing Docker and Understanding Its Architecture

Steps:
Install Docker Desktop (Windows/Mac) or Docker Engine (Linux).
Open a terminal and test:

Syntax:
docker --version
docker run hello-world

Docker Architecture (simple figure):



The Docker client sends commands to the daemon, which manages images and containers on your system.

WEEK 2: Hands-On Docker Basics

Day 5–6: Docker Images and Containers

Term	Meaning	Example Command
Image	A blueprint or template for containers.	docker pull ubuntu
Container	A running instance of an image.	docker run -it ubuntu
Registry	A storage for images (like GitHub but for containers).	docker push myapp:latest

Example Workflow:

```
# Step 1: Get an image
docker pull nginx

# Step 2: Run it as a container
docker run -d -p 8080:80 nginx

# Step 3: See it running
docker ps

# Step 4: Stop and remove it
docker stop <container_id>
docker rm <container_id>
```

You can access the app in your browser at **http://localhost:8080**

Day 7–8: Dockerfiles (Building Your Own Image)

A Dockerfile defines how to build your image.

Example: Dockerfile for a simple **Node.js** app

Dockerfile:

```
# Use official Node image
FROM node:18

# Set working directory
WORKDIR /app

# Copy files and install dependencies
COPY package*.json ./
RUN npm install

# Copy app code
COPY . .

# Expose port
EXPOSE 3000

# Run the app
CMD ["npm", "start"]
```

Build and run it:

```
docker build -t my-node-app .
docker run -d -p 3000:3000 my-node-app
```

The Dockerfile acts like a recipe, defining everything needed to create a working app environment.

Day 9–10: Docker Compose (Managing Multiple Containers)

If your app needs a database + backend, you can use Docker Compose.

Example: **docker-compose.yml**

```
version: '3'
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

Then just run:

```
docker compose up
```

Docker Compose simplifies running multi-container apps (like full stacks).

WEEK 3: Best Practices and Advanced Concepts

Day 11–12: Best Practices in Docker Implementation

Category	Best Practice	Example
Image Size	Use lightweight base images (like **alpine**)	FROM node:18-alpine
Security	Avoid running as root, use **.dockerignore**	USER appuser
Caching	Order Dockerfile steps efficiently	Place COPY package*.json before code copy
Environment Configs	Use environment variables	ENV NODE_ENV=production
Volumes	Store data persistently	docker volume create mydata
Tagging	Always tag images (avoid **latest** in prod)	myapp:v1.0.0
Cleanup	Regularly remove unused containers/images	docker system prune

Day 13–14: Container Orchestration (Intro to Kubernetes)

Once you know Docker, you can learn **Kubernetes (K8s)**, a tool that automates deploying and scaling containers.

Simple analogy:

Tool	Description
Docker	Builds and runs containers on a single machine.
Kubernetes	Description

You don't need to dive deep yet, but know that Docker containers often run inside Kubernetes clusters in production.

Day 15: Mini Project – Deploy Your App

Project idea:
Build a simple web app (Node.js or Python Flask), containerize it with Docker, and run it alongside a database using Docker Compose.

Checklist:

- Create Dockerfile
- Create docker-compose.yml
- Run app locally with docker compose up
- Push image to Docker Hub
- Test on another computer or cloud VM