Project 1: Let's play with Docker Containers

CECS 327 – Networks & Distributed Comp

You should submit the required deliverable materials on Canvas by **11:55pm, September 20th (Saturday), 2025**. **Only one submission** of each group.

Objective

This project will guide undergraduate students through the basics of Docker, enabling them to:

1. Understand containerization and its benefits.
2. Install Docker and run their first container.
3. Create a custom Docker image and containerize a simple programming task.
4. Apply Docker to deploy a basic application (e.g., a Python script or web server).

Prerequisites

1. Basic familiarity with the command line (Terminal/CMD/PowerShell).
2. Fundamental programming knowledge (e.g., Python, Node.js, or Java).
3. A computer with at least 4GB RAM and 20GB free storage.

Step-by-Step Guide

1. Introduction to Docker

What is Docker?

- Docker is a platform for developing, shipping, and running applications in isolated environments called **containers**.
- Containers package code and dependencies, ensuring consistency across environments (e.g., your laptop, a server, or the cloud).

Why Use Docker?

- Eliminates "works on my machine" problems.
- Simplifies dependency management.
- Enables scalable deployments.

Key Concepts:

- **Image**: A read-only template with instructions to create a container (e.g., ubuntu:20.04, python:3.9).
- **Container**: A runnable instance of an image.
- **Dockerfile**: A text file defining how to build an image.
- **Docker Hub**: A registry to share and pull pre-built images.

2. Install Docker

1. **Download Docker Desktop** (for Windows/macOS) or **Docker Engine** (for Linux):
   - Windows/macOS
   - Linux: Follow instructions for your distribution (e.g., apt install docker.io for Ubuntu).
2. Verify installation:

```
docker --version       # Check Docker version
docker run hello-world  # Run a test container
```

   If successful, you'll see: *"Hello from Docker!"*

3. Run Your First Container

Task: Run a lightweight Alpine Linux container.

```
docker run -it alpine:latest sh
```

- -it: Run interactively with a terminal.
- alpine:latest: The image name (a minimal Linux distribution).
- sh: Start a shell session inside the container.

Explore the container:

```
ls /      # List files
echo "Hello Docker!" > test.txt
exit      # Exit the container
```

4. Create a Custom Docker Image

Task: Containerize a Python script that prints "Hello, Docker!".

Step 1: Write a Python script (app.py):

```
print("Hello, Docker! This is my first containerized app.")
```

Step 2: Create a Dockerfile:

Please explore how to create the Dockerfile

Step 3: Build the Docker image:

```
docker build -t my-python-app .
```

- -t my-python-app: Tag the image with a name.

Step 4: Run the container:

```
docker run my-python-app
```

Output: Hello, Docker! This is my first containerized app.

5. Task1: Deploy a Web Server

Task: Run an Nginx web server using Docker.

1. Pull the Nginx image:
```
docker pull nginx:latest
```
2. Run the container with port mapping:
```
docker run -d -p 8080:80 --name my-nginx nginx:latest
```
   - -d: Run in detached mode (background).
   - -p 8080:80: Map host port 8080 to container port 80.
3. Visit http://localhost:8080 in your browser to see the Nginx welcome page.

Task: Modify the default page by mounting a volume:

1. Create a local index.html file.
2. Run the container with a volume:
```
docker run -d -p 8080:80 -v $(pwd)/index.html:/usr/share/nginx/html/index.html nginx:latest
```

6. Task 2: Multi-Container Setup (Server + Clients)

Task 2.1: Create a Server Container

- Write a simple Python TCP/UDP server (server.py) that listens for incoming messages.
- Create a Dockerfile for the server container.

Task 2.2: Connect Server and Clients Using Docker Compose

- Write a docker-compose.yml file to define one server container and multiple clients.
  Please explore how to design yml file.
- Run and show communication in logs.

7. Troubleshooting

- **Permission denied?** Run Docker commands with sudo (Linux) or ensure Docker Desktop is running (Windows/macOS).
- **Port already in use?** Change the host port (e.g., -p 8081:80).
- **Debug a container**:
```
docker logs <container_id>   # View logs
docker exec -it <container_id> sh  # Enter a running container
```

8. Project Submission

1. Submit your code, Dockerfile, and README to zip file.
2. Submit a report include your outputs and logs.

Submission Requirements

You need to strictly follow the instructions listed below:

1) This is a **group project**, please submit a .zip/.rar file that contains all files (report, code, Dockerfile, and README), only one submission from one group.

2) Make a **video** to record your code execution and outputs. The video should present your name or time as identification (You are suggested to upload the video to YouTube and put the link into your report).

3) The submission should include your **source code** and **project report**. Project report should contain your groupmates name and ID. Document your work in a **report** file:

- Explain your Dockerfile steps.
- Include screenshots of successful runs.
- List challenges faced and solutions.

4) Your code must **be able to compile**; otherwise, you will receive a grade of zero.

5) Your code should not produce anything else other than the required information in the output file.

6) If you code is **partially completed**, please explain the details in the report what has been completed and the status of the missing parts, we will grade it based on the entire performance.

7) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	15 pts
Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your group mates	5 pts
Code can be compiled and shows correct outputs	40 pts

Other Policies

1) Late submissions will be graded based on our policy discussed in the course syllabus.
2) Code-level discussion is prohibited. We will use anti-plagiarism tools to detect violations of this policy.
3) AI-tools such as ChatGPT, Claude, etc. are prohibited in both code and contents.

References

- Docker Official Documentation
- Docker Curriculum
- Play with Docker Labs