Project 1: Let’s Play With Docker Containers

Zophia Kimberly Laud [029783219],

Ivy Le [030215976],

Mai Mai [030276114]

Instructor: Hailu Xu

California State University, Long Beach

College of Engineering

CECS 327, Sec 02, Spring 2026

February 28, 2026

Project 1: Let’s Play With Docker Containers

Links:

Video - https://youtu.be/qfrva-UUAYM

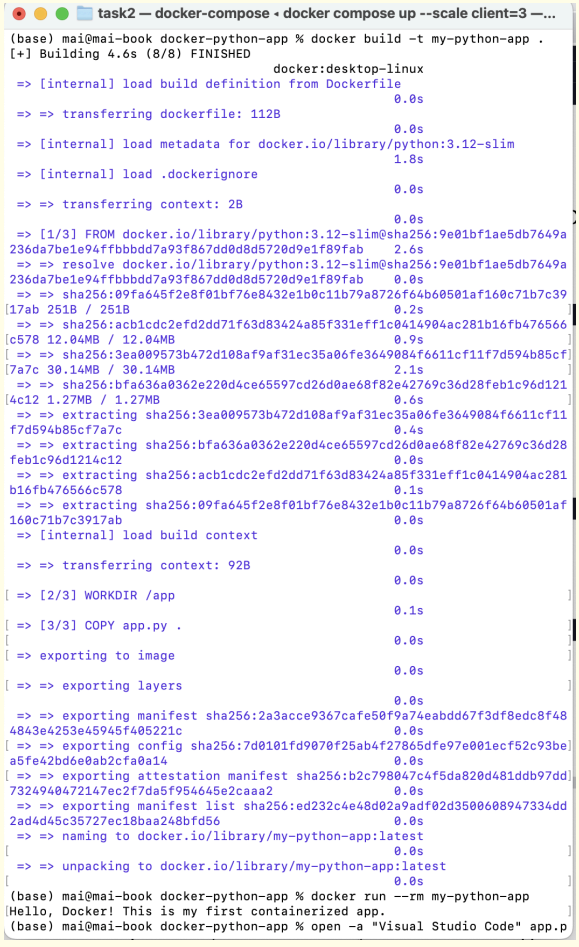
First Container

For the first container task, we created a simple Python script that prints a message to the console and then containerized it using Docker.

After writing the script, we created a Dockerfile to define the environment, including the Python base image and instructions to execute the script when the container starts:

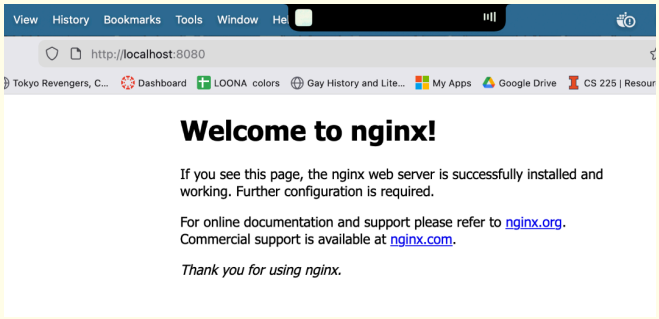
FROM python:3.12-slim (Uses the official Python 3.12 base image as the starting point)
WORKDIR /app (creates work directory app inside the container)
COPY app.py . (copies the python script we created into the container)
CMD ["python", "app.py"] (specifies instructions for the contianer)

We then built a Docker image from the Dockerfile and ran the container successfully. Here’s a screenshot of the container running:



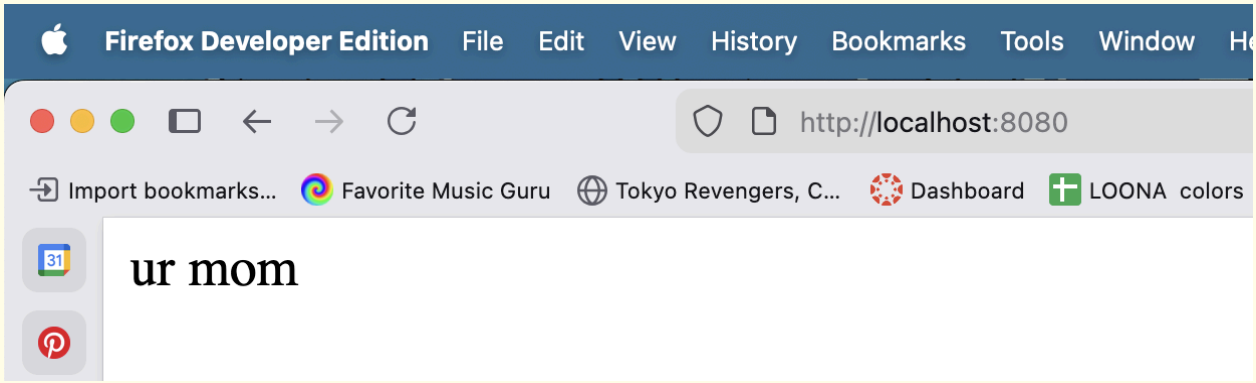
Project 1: Let's Play With Docker Containers

Task 1: Deploying an Nginx Web Server



Docker was used to retrieve and run the official nginx:latest image from Docker Hub.
Once that happens, the container we created was then executed in detached mode with port mapping, allowing for access through our browser at localhost:8080

To customize the content being shown, a local index.html file was mounted into the container using a bind volume, mapping it to /usr/share/nginx/html/index.html, which is Nginx's default web root



Task 2: Multi-Container Setup Using Docker Compose

A TCP-based server was implemented with Python's socket module. The server binds to a port, listens for incoming connections, accepts client connections, and then receives data using recv().
The server is based on the Python socket echo server example, binding to port 50007 and listening for incoming connections. Docker Compose creates a default bridge network, allowing containers to communicate using service names as DNS hostnames.

Compose run command:

- docker compose up --scale client=3

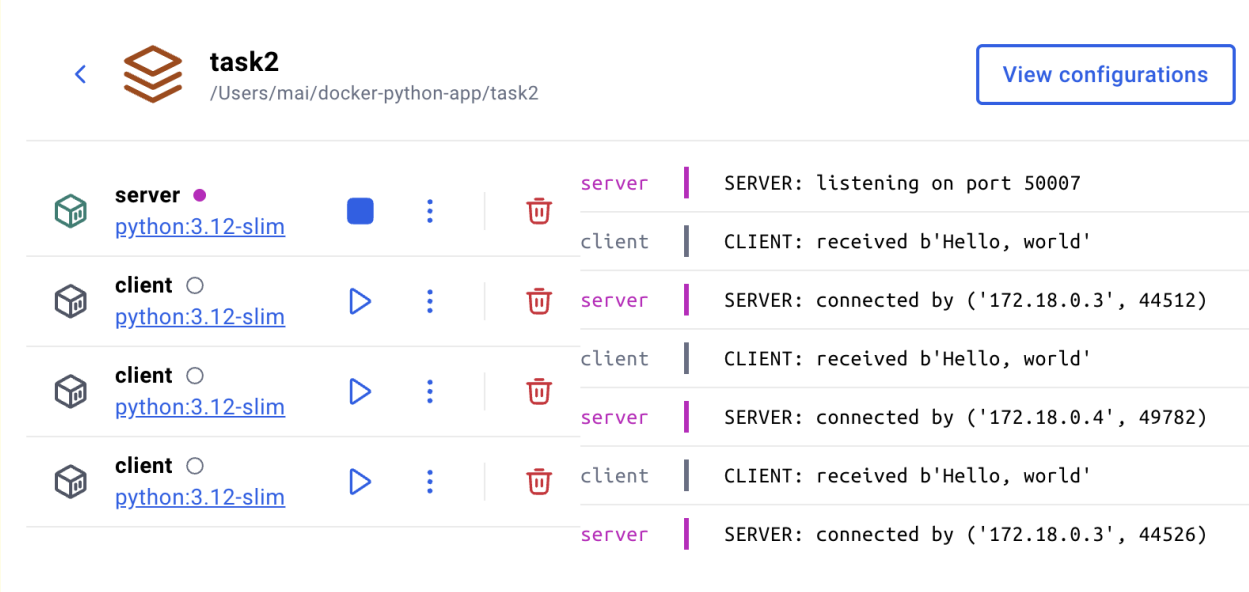
Project 1: Let’s Play With Docker Containers

```
client-1 exited with code 1
(base) mai@mai-book task2 % docker compose up --scale client=3
WARN[0000] No services to build
[+] up 4/4
 ✔ Container tcp-server     Recreated
 ✔ Container task2-client-1 Recreated
 ✔ Container task2-client-2 Recreated
 ✔ Container task2-client-3 Recreated
Attaching to client-1, client-2, client-3, server-1
server-1  | SERVER: listening on port 50007
client-3  | CLIENT: received b'Hello, world'
server-1  | SERVER: connected by ('172.18.0.3', 44512)
client-3 exited with code 0


server-1  | SERVER: connected by ('172.18.0.4', 49782)
client-2  | CLIENT: received b'Hello, world'


server-1  | SERVER: connected by ('172.18.0.3', 44526)
client-1  | CLIENT: received b'Hello, world'
client-2 exited with code 0
client-1 exited with code 0

v View in Docker Desktop   o View Config   w Enable Watch   d Detach
```



Project 1: Let’s Play With Docker Containers

Contributions

Mai Mai - Server/client implementation, Docker Compose configuration, debugging, testing

Ivy Le - Nginx deployment, volume mounting, port mapping, testing.

Zophia Kimberly Laud - Multi-container testing, networking validation, report preparation.

Challenges

1. When first using Docker, writing a correct Dockerfile was consuming because we didn’t know which base image to use, and we didn’t understand how WORKDIR, COPY, and CMD work.
    a. **Solution**: We looked through documentation and figured out how to create a simple Dockerfile for a Python app.
2. Containers would have issues with executing the code
    a. Solution: We found out that docker logs existed, and we can use those to debug.

References

https://docs.python.org/3/library/socket.html

https://docs.docker.com/compose/how-tos/networking/