

Project 1: Let's Play With Docker Containers

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)

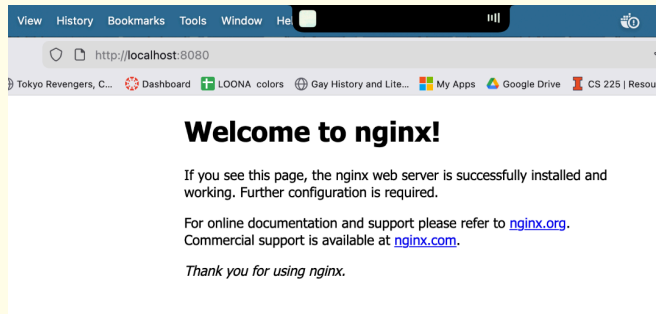
CMD ["python", "app.py"] (specifies instructions for the container)

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

```
task2 — docker-compose • docker compose up --scale client=3 —...
(base) mai@mai-book docker-python-app % docker build -t my-python-app .
[+] Building 4.6s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 112B
=> [internal] load metadata for docker.io/library/python:3.12-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/python:3.12-slim@sha256:9e01bf1ae5db7649a236da7be1e94ffbbdd7a93f867dd0d8d5720d9e1f89fab
=> => resolve docker.io/library/python:3.12-slim@sha256:9e01bf1ae5db7649a236da7be1e94ffbbdd7a93f867dd0d8d5720d9e1f89fab
=> => sha256:09fa645f2e8f01bf76e8432e1b0c11b79a8726f64b60501af160c71b7c3917ab 251B / 251B
=> => sha256:acblcdc2efd2dd71f63d8342a85f331eff1c0414904ac281b16fb476566c578 12.04MB / 12.04MB
[=> => sha256:3ea09573b472d108af9af31ec35a06fe3649084f6611cf11f7d594b85cf7a7c 30.14MB / 30.14MB
=> => sha256:bfa636a0362e220d4ce6597cd26d0ae68f82e42769c36d28feb1c96d1214c12 1.27MB / 1.27MB
=> => extracting sha256:3ea09573b472d108af9af31ec35a06fe3649084f6611cf11f7d594b85cf7a7c
=> => extracting sha256:bfa636a0362e220d4ce6597cd26d0ae68f82e42769c36d28feb1c96d1214c12
=> => extracting sha256:acblcdc2efd2dd71f63d8342a85f331eff1c0414904ac281b16fb476566c578
=> => extracting sha256:09fa645f2e8f01bf76e8432e1b0c11b79a8726f64b60501af160c71b7c3917ab
=> [internal] load build context
=> => transferring context: 92B
[=> [2/3] WORKDIR /app
[=> [3/3] COPY app.py .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:2a3acce9367cfe50f9a74eabdd67f3df8edc8f484843e4253e45945f405221c
[=> => exporting config sha256:7d010fd9070f25ab4f27865df97e001ecf52c93bea5fe42bd0e0ab2cfa0a14
[=> => exporting attestation manifest sha256:b2c798047c4f5da820d481ddb97dd73249d0472147cc27d5f954645e2c2caaa2
=> => exporting manifest list sha256:ed232c4e48d02a9adf02d3500608947334dd2ad4d45c35727ec18baa248bfd56
=> => naming to docker.io/library/my-python-app:latest
=> => unpacking to docker.io/library/my-python-app:latest
(base) mai@mai-book docker-python-app % docker run --rm my-python-app
Hello, Docker! This is my first containerized app.
(base) mai@mai-book docker-python-app % open -a "Visual Studio Code" app.p
```

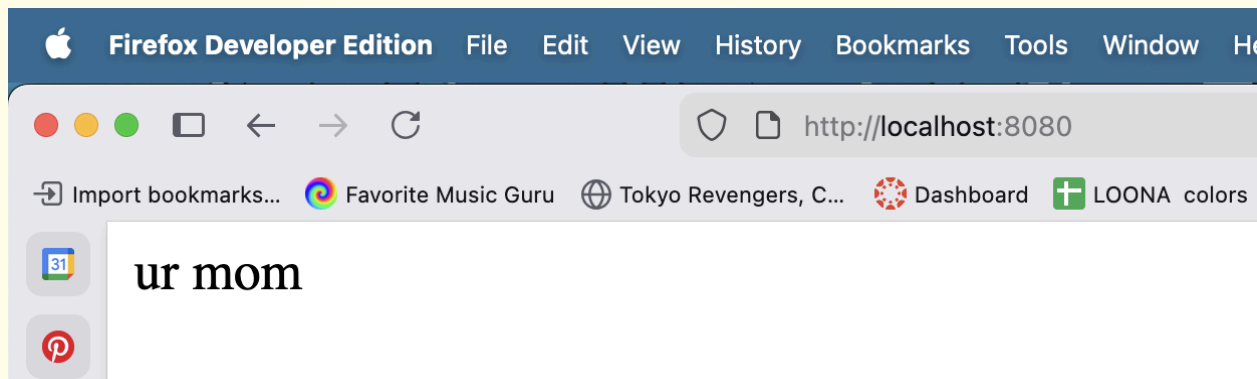
Project 1: Let's Play With Docker Containers

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
```

 **task2**
/Users/mai/docker-python-app/task2

[View configurations](#)

 server ● python:3.12-slim	 		server SERVER: listening on port 50007
 client ○ python:3.12-slim	 		client CLIENT: received b'Hello, world'
 client ○ python:3.12-slim	 		server SERVER: connected by ('172.18.0.3', 44512)
 client ○ python:3.12-slim	 		client CLIENT: received b'Hello, world'
 client ○ python:3.12-slim	 		server SERVER: connected by ('172.18.0.4', 49782)
 client ○ python:3.12-slim	 		client CLIENT: received b'Hello, world'
			server SERVER: connected by ('172.18.0.3', 44526)

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/>