Lab 5 Report

Course: Cloud_Computing
Student: ZhengYang
GitHub Repo: y*angzheng.github.io — Repository for personal website project using GitHub Pages*
Date: 2025.11.1

1. Abstract

This laboratory session provides a hands-on introduction to core Docker technologies. The experiment covers fundamental concepts including containerization, image management, and orchestration of multi-container applications using Docker Compose. Key practical skills gained include running isolated containers, managing Docker images, publishing container ports, overriding default configurations, persisting data with volumes, and sharing files between the host and containers. The objective is to demonstrate how Docker solves environment inconsistency and dependency management problems in software development and deployment.

2. Introduction & Objectives

Docker containers package an application and its dependencies into a standardized, isolated unit, ensuring consistency across different environments (development, testing, production). This lab explores the following key concepts and objectives:

Container:An isolated process that runs an application component.

Image:An immutable,layered package containing everything needed to run a container.

Docker Compose:A tool for defining and running multi-container applications.

Learning Objectives:

- Run and manage Docker containers via both the CLI and GUI.
- Search, pull, and inspect Docker images.
- Use Docker Compose to define, build, and run a multi-service application.
- Configure network access by publishing and exposing container ports.
- Override default container settings for environment variables, resources, and commands.
- Implement data persistence using Docker volumes.
- Share files between the host system and a container using bind mounts.

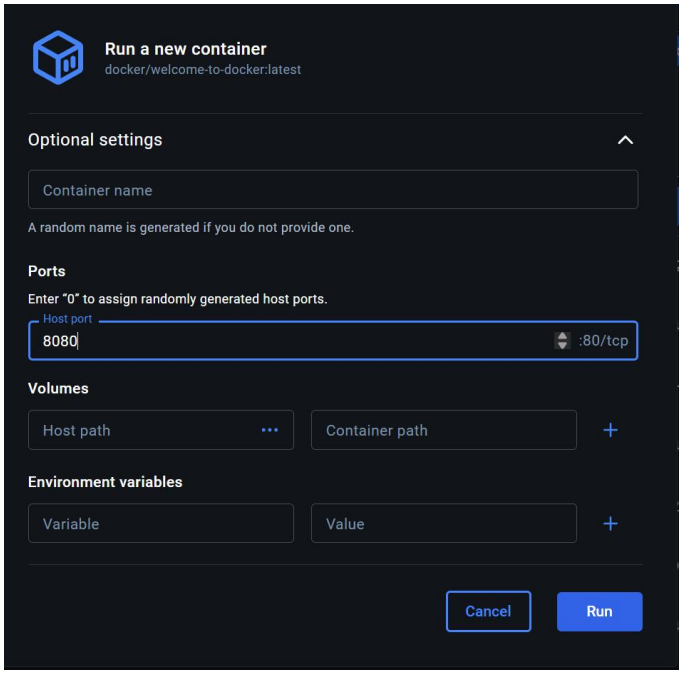
3. Experimental Procedures & Results

Experiment 1: Running Your First Container

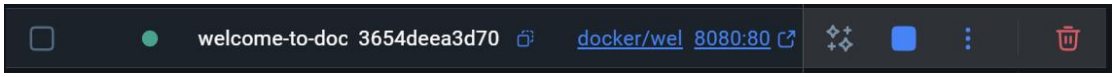
Objective: To understand the basic lifecycle of a Docker container.

Procedure:

- **CLI Method:**Run the command: docker run -d -p 8080:80 docker/welcome-to-docker



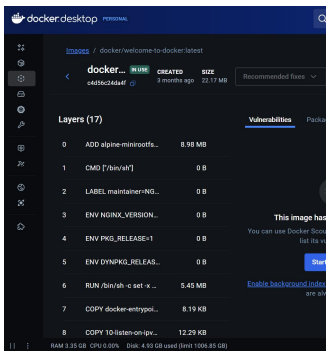
- **Verification:**Use docker psto confirm the container is running. Note the PORTScolumn showing 0.0.0.0:8080->80/tcp.



- **Access Application:**Open a web browser and navigate to http://localhost:8080.
- **Stop Container**:Run docker stop <container_id>.

Results:The welcome-to-dockerweb server started successfully.

The application was accessible on the host machine at port 8080, which was forwarded to the container's port 80.



Experiment 2: Working with Docker Images

Objective: To learn how to find, acquire, and analyze Docker images.

Procedure:

- **Search Image**:Run docker search docker/welcome-to-docker.
- **Pull Image**:Run docker pull docker/welcome-to-docker.

- **List Images:**Run docker image ls to view all local images.

```
C:\Users\zy>docker image ls
REPOSITORY
mysql
postgres
postgres
gitea/gitea
<none>
registry.cn-hangzhou.aliyuncs.com/google_c
httpd
httpd
docker/welcome-to-docker
node
```

- **Inspect Layers:**Run docker image history docker/welcome-to-dockerto see the image's layered filesystem.

Results:

The image was successfully located on Docker Hub and downloaded to the local machine.

Experiment 3: Orchestrating with Docker Compose

Objective: To deploy a multi-container application (Node.js frontend, MySQL database) using a declarative YAML file.

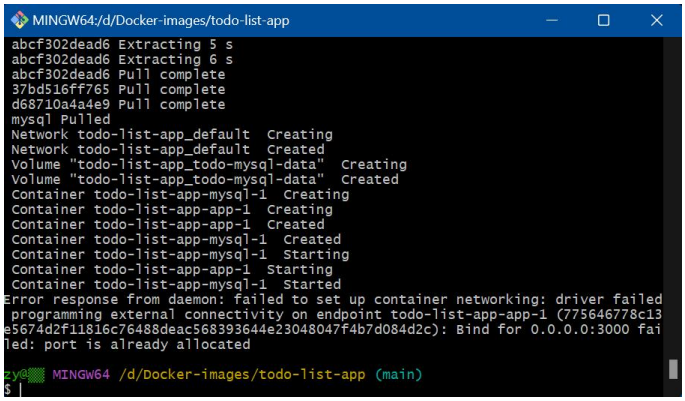
Procedure:

- **Clone Repository:** git clone https://github.com/dockersamples/todo-list-app

```
zy@ MINGW64 /d/Docker-images
$ git clone https://github.com/dockersamples/todo-list-app
Cloning into 'todo-list-app'...
remote: Enumerating objects: 93, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 93 (delta 0), reused 0 (delta 0), pack-reused 91 (from 2)
Receiving objects: 100% (93/93), 1.68 MiB | 105.00 KiB/s, done.
Resolving deltas: 100% (15/15), done.
```

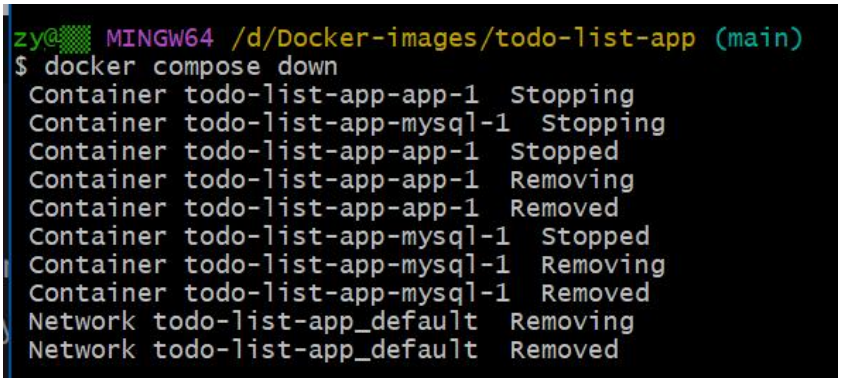
- **Navigate:**cd todo-list-app
- **Examine File:**Review the compose.yamlfile to understand the service definitions for appand mysql.

- **Start Application:**Run docker compose up -d --build



- **Access Application:**Open http://localhost:3000in a browser.

- **Teardown:**Run docker compose down --volumesto stop and remove all resources.
- **Results:**



Docker Compose automatically created a network, volume, and started two containers. The todo-list application functioned correctly, demonstrating interaction between the frontend and database containers.

Experiment 4: Network Configuration - Port Publishing

Objective: To understand how to make containerized services accessible from the host network.

Procedure:

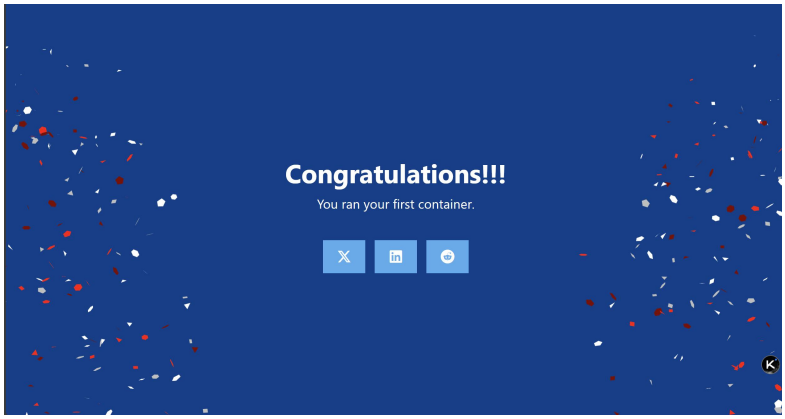
- **Explicit Port Mapping:**Re-run the container from Experiment 1 using -p 8080:80.

```
PS C:\Users\zy> docker run -d -p 8080:80 docker/welcome-to-docker
b837c7b5b60376c7820a5c78f4e76fa728f1c92e7d4aa7a8c6c26c6d38255ed8
```

- **Compose Port Mapping:**Create a compose.yamlfile with a portssection mapping host port 8080 to container port 80 for a service.

Results:

Successfully configured network access to the containerized application.



Experiment 5: Overriding Container Defaults

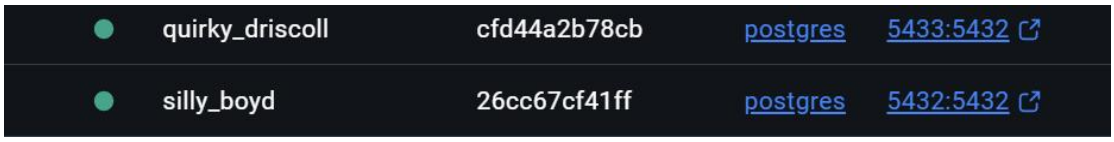
Objective:To customize container runtime behavior.

Procedure:

- **Environment Variables:**Start a PostgreSQL container with -e POSTGRES_PASSWORD =secret.

```
C:\Users\zy>docker run -d -e POSTGRES_PASSWORD=secret -p 5433:5432 postgres
cfd44a2b78cb6f58102a2273e989afc6a6b1025d00935299bb253ef23e022ebc
```

```
C:\Users\zy>docker run -d -e POSTGRES_PASSWORD=secret -p 5434:5432 --network mynetwork postgres
641436ec04634fb4a247893415a594d6c9a4876f0dc89118b27bb64e4cb2edd4
```



- **Resource Limits:**Start a container with resource constraints: --memory="512m" --cpus="0.5".

```
C:\Users\zy>docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
552c67d9b05f   bridge        bridge    local
26548b93872a   host          host      local
bc876f925e87   minikube      bridge    local
a138fc4a881e   mkdir_gitea   bridge    local
d6c57bfa5208   mynetwork     bridge    local
f26b1d11b68b   none          null      local
```

Custom Network: Create a network with docker network create mynetworkand run a container on it using --network mynetwork.

Results:

Successfully configured a database password and limited container resource usage.

Experiment 6: Data Persistence with Volumes

Objective: To prevent data loss when a container is removed by persisting data to a volume.

Procedure:

- **Run DB with Volume:** StartPostgreSQL: docker run -d -v postgres_data: /var/lib/postgresql/data -e POSTGRES_PASSWORD = secret --name db postgres

```
C:\Users\zy>docker run -d -p 80:80 -v log-data:/logs docker/welcome-to-docker
f9067f83634b73cad01089d00fc1b58257b448810ab20be22b26157b7e0af60f
```

- **CreateData:**

```
C:\Users\zy>docker run --name=db -e POSTGRES_PASSWORD=secret -d -v postgres_data:/var/lib/postgre
de7222b778d52e457bbc3b5509ac6155af1d95719968dba2a55b1ab6b5617464

C:\Users\zy>docker exec -ti db psql -U postgres
psql (14.19 (Debian 14.19-1.pgdg13+1))
Type "help" for help.

postgres=#
```

```
postgres=# CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    description VARCHAR(100)
);
INSERT INTO tasks (description) VALUES ('Finish work'), ('Have fun');
CREATE TABLE
INSERT 0 2
postgres=# SELECT * FROM tasks;
 id | description
----+-------------
  1 | Finish work
  2 | Have fun
(2 rows)
```

Connect to the DB (docker exec -ti db psql -U postgres) and create a table with sample data.

- **Remove and Recreate Container:**Run docker rm -f db, then start a new container (new-db) mounting the same postgres_datavolume.
- **Verify Data:**Connect to new-dband confirm the data still exists.

```
C:\Users\zy>docker stop db
db

C:\Users\zy>docker rm db
db

C:\Users\zy>docker run --name=new-db -d -v postgres_data:/var/lib/postgresql/data postgres
a7e74d1d791625564384a192e9c1c05593aaa7b88bf728ea8fbe0d4ef7903139

C:\Users\zy>docker exec -ti new-db psql -U postgres -c "SELECT * FROM tasks"
Error response from daemon: container a7e74d1d791625564384a192e9c1c05593aaa7b88bf728ea8fbe0d4ef7903139 is not running

C:\Users\zy>docker run --name=new-db -d -v postgres_data:/var/lib/postgresql/data postgres:14
5f601dc2408ffa3a80f347da9edb180c04b1488b0fbe95b154c11c0e9c30c757

C:\Users\zy>docker exec -ti new-db psql -U postgres -c "SELECT * FROM tasks"
 id | description
----+-------------
  1 | Finish work
  2 | Have fun
(2 rows)
```

Results:

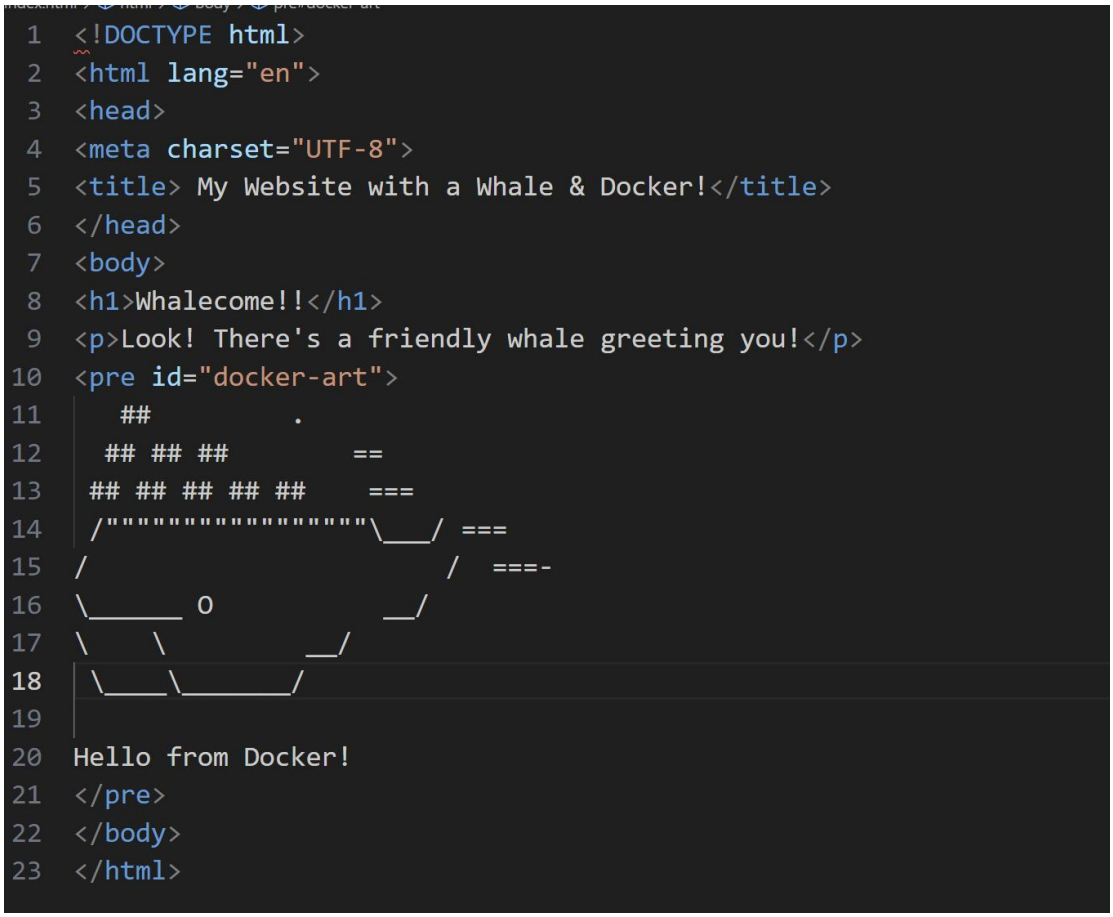
Data persisted in the volume survived the deletion and recreation of the container. Demonstrated the critical use case for stateful services like databases.

Experiment 7: File Sharing with Bind Mounts

Objective: To enable real-time file synchronization between the host and a container for development.

Procedure:

- **Create Web Content:** Create a local directory public_htmlwith a custom index.htmlfile.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title> My Website with a Whale & Docker!</title>
</head>
<body>
<h1>Whalecome!!</h1>
<p>Look! There's a friendly whale greeting you!</p>
<pre id="docker-art">
   ##         .
  ## ## ##        ==
 ## ## ## ## ##    ===
 /"""""""""""""""""\___/ ===
/                      /  ===-
\_____ O           __/
\    \         __/
 \____\_______/

Hello from Docker!
</pre>
</body>
</html>
```

- **Run with Bind Mount:**Start an HTTPD container,mounting the host's public_htmlto thecontainer'swebroot(/usr/local/apache2/htdocs/)using-v$(pwd)/public_html:/usr/local/apache2/htdocs.
- **Test Live Editing:** Modify the host's index.htmlfile and refresh the browser to see changes reflected immediately.

Results:

The container served the custom HTML page from the host.

4. Conclusion

This lab provided a comprehensive, practical foundation in Docker. The core concepts of containers, images, and Docker Compose were explored through hands-on experiments. The skills acquired — including container lifecycle management, multi-service orchestration, networking, configuration, and data persistence—are essential for modern software development and deployment. Docker effectively addresses environment inconsistency, enabling reliable application delivery from development to production.