Docker Lab Session - Report

This report explains the steps taken to build a batch of containers, consisting of:

- 2 flask containers
- one MySQL container

Folder structure

```
.
├── App_1
│   ├── Dockerfile
│   ├── app1.py
│   └── requirements.txt
├── App_2
│   ├── Dockerfile
│   ├── app2.py
│   └── requirements.txt
├── db
│   └── init.sql
├── docker-compose.yml
├── rapport.md
└── readme.md
```

Definition of the app.py files

App1

```
from flask import Flask, jsonify
import mysql.connector

app = Flask(__name__)


def start_connection():
    config = {
        "host": "db",
        "user": "root",
        "passwd": "mypassword",
        "port": "3306",
        'database': 'employees'
        }
    connection = mysql.connector.connect(**config)
    cursor = connection.cursor(dictionary=True)
    cursor.execute('SELECT Employee_Name, Title FROM employee_data')
    results = cursor.fetchall()
    cursor.close()
    connection.close()
    return (jsonify({'Employee Data': results}).get_data(as_text=True)
            + "\n"
            + "It works well")


@app.route('/')
def hello_world():
    return start_connection()


if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5000)
```

This Flask app works with a function start_connection(), which connects to a specific SQL database hosted by the MySQL container and selects all employee names and titles from this database.

App2

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World! You are in App 2'

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0",port=5001)
```

A simple second app that prints "Hello, World! You are in App 2." The port is adjusted to 5001 to avoid conflicts between the two apps (Flask app defaults to port 5000).

SQL

init.sql

```
CREATE DATABASE employees;
USE employees;


CREATE TABLE employee_data (
  Employee_Name VARCHAR(50),
  Title VARCHAR(50)
);


INSERT INTO employee_data
  (Employee_Name, Title)
VALUES
  ('Amit Khanna', 'Manager'),
  ('Anjali Gupta', 'Engineer');
```

This file initializes the SQL database, creating a database "employees" and a table "employee_data" with columns for employee name and title. Two rows of data are inserted into the table.

Dockerfiles

The Dockerfiles in the `App_1` and `App_2` folders are identical:

```
FROM python:3.9.6
WORKDIR /app
```

```
COPY requirements.txt /app
RUN pip install --no-cache-dir -r requirements.txt
COPY app1.py /app
CMD ["python", "app1.py"]
```

These files are used to create the image that will run in the app containers. They copy the requirements.txt file, install required Python packages, and copy the app1.py (or app2.py) file into the container's `/app` folder.

Note: The Dockerfile refers to app1.py but it's also applicable to app2.py.

Remarks : Thanks to binds volume, when the app.py is modified locally, the changes are directly reflected into the container folder. Thus, we can check directly the changes in the corresponding http adress.

docker-compose file

```
services:
  app1:
    build: ./App_1
    networks:
      - app1_backend
    ports:
      - 5000:5000
    volumes:
      - ./App_1:/app
  app2:
    build: ./App_2
    networks:
      - app2_backend
    ports:
      - 5001:5001
  db:
    image: mysql:5.7
    ports:
      - "32000:3306"
    networks:
      - app1_backend
      - app2_backend
    environment:
      MYSQL_ROOT_PASSWORD: "####"
    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./data:/var/lib/mysql

networks:
  app1_frontend:
  app2_frontend:
```

This file is maybe the most important because it sets all the parameters to create the differents containers based on each image. Let's try to understand all of it.

The first line is services, it is where we are going to declare of our differents containers and their properties.

app1 container

We create a first container app1 which is builded base on the image created by the Dockerfile located in ./App_1 folder.

Then we define on which network this container will be able to interact with. Here the network is app1_backend, it means that all container connected to this network can interact with each others.

We also match our port 5000 to the port 5000 of the container.

Finally, we set a bind volume between our local repository and the container's one.

app2 container

We don't bind a volume to this app. In the lab context, we need at least one app with bind volume.

We match our localhost 5001 port to the port 5000 of the container. Indeed, we already have a connexion which use 5000 port therefore we have to change.

And we link this container to another network as we don't want any possible interactions between app1 and app2 container.

db container

This SQL database container is linked with the two app newtorks. So each app can interact with db container.

And we bind a volume, and incorporate the init.sql file in the good folder of the MySQL API to run it at the initialization of the container.

Conclusion

Finally, we just have to run the docker-compose.yml file with `docker compose up` and find our differents applications on http://127.0.0.1:5000. and http://127.0.0.1:5001.