Practical 6

Title: Docker Compose for Multi-Container Applications (Flask + PostgreSQL)

Objective:

1. Create a Flask web application that connects to a PostgreSQL database.
2. Use Docker Compose to orchestrate both containers.
3. Test communication between the services.

<u>For windows:</u>

Here's your **entire Dockerized Flask + PostgreSQL project**, optimized step-by-step **for Windows users**, especially those using **PowerShell** or **CMD**. This includes file creation, permission handling, and running Docker from Windows.

Open Docker Desktop

1. Create Project Structure (PowerShell Recommended)

```
# Create main project folder
mkdir flask_postgres_app
cd flask_postgres_app

# Create folders and files
mkdir app
New-Item app\app.py -ItemType File
New-Item app\requirements.txt -ItemType File
New-Item Dockerfile -ItemType File
New-Item docker-compose.yml -ItemType File
New-Item wait-for-postgres.sh -ItemType File

#optional
Pip install psycopg2-binary
```

✅ 2. Fill the Files

🔹 app/app.py

```
from flask import Flask
import psycopg2

app = Flask(__name__)
```

```
@app.route('/')
def index():
    try:
        conn = psycopg2.connect(
            host='db',
            database='mydb',
            user='myuser',
            password='mypassword'
        )
        cur = conn.cursor()
        cur.execute('SELECT version();')
        db_version = cur.fetchone()
        cur.close()
        conn.close()
        return f'Connected to PostgreSQL: {db_version}'
    except Exception as e:
        return f'Failed to connect to PostgreSQL: {e}'

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

- app/requirements.txt

```
flask
psycopg2-binary
```

- Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

# Install PostgreSQL client
RUN apt-get update && \
    apt-get install -y postgresql-client && \
    rm -rf /var/lib/apt/lists/*

COPY app/ .

RUN pip install --no-cache-dir -r requirements.txt

COPY wait-for-postgres.sh /wait-for-postgres.sh
RUN chmod +x /wait-for-postgres.sh

CMD ["/wait-for-postgres.sh", "db", "python", "app.py"]
```

🔹 docker-compose.yml

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5001:5000"
    depends_on:
      - db

  db:
    image: postgres:14
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

🔹 wait-for-postgres.sh

```
#!/bin/bash
# wait-for-postgres.sh

set -e

host="$1"
shift
cmd="$@"

until PGPASSWORD=mypassword psql -h "$host" -U "myuser" -d "mydb" -c '\q'; do
  >&2 echo "Postgres is unavailable - sleeping"
  sleep 1
done

>&2 echo "Postgres is up - executing command"
exec $cmd
```

✅ 3. Make Shell Script Executable (for Windows users)

If you're on **Windows**, there's **no native chmod**, so Docker will use the Linux environment to handle file permissions during build.

But to avoid issues:

- Use **Git Bash** or **WSL** to run chmod +x wait-for-postgres.sh, **or**

- Let the Dockerfile handle it as it already does:
- RUN chmod +x /wait-for-postgres.sh

So, **you can skip this step on Windows**.

✅ 4. Optional Cleanup (if needed)

If you’ve previously built the containers and want a fresh start:

```
docker-compose down -v
```

✅ 5. Build and Start the Application

```
docker-compose up --build
```

This command will:

- Build the Docker image
- Start Flask app and PostgreSQL
- Wait for the DB to be ready before launching the Flask app

✅ 6. Test in Browser

Open your browser and visit:

👉 http://localhost:5001

You should see:

```
Connected to PostgreSQL: ('PostgreSQL 14.x...',)
```

💡 Tips for Windows Users

- **Use PowerShell or Git Bash** for best compatibility with file scripts.
- Avoid running Docker commands in **CMD**, as it can behave unpredictably with shell scripts.
- If you're using **WSL**, everything will work as if you're on Linux, no changes needed.

On Linux:

```
mkdir flask_postgres_app
```

```
cd flask_postgres_app
```

```
mkdir app
touch app/app.py app/requirements.txt
touch Dockerfile docker-compose.yml wait-for-postgres.sh
```

Create app/app.py (in app.py file)

```
from flask import Flask
import psycopg2
app = Flask(__name__)
@app.route('/')
def index():
    try:
        conn = psycopg2.connect(
            host='db',
            database='mydb',
            user='myuser',
            password='mypassword'
        )
        cur = conn.cursor()
        cur.execute('SELECT version();')
        db_version = cur.fetchone()
        cur.close()
        conn.close()
        return f'Connected to PostgreSQL: {db_version}'
    except Exception as e:
        return f'Failed to connect to PostgreSQL: {e}'

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

Create app/requirements.txt (in requirements.txt file)

```
flask
psycopg2-binary
```

Create Dockerfile (in Dockerfile)

```
FROM python:3.9-slim
WORKDIR /app
# Install PostgreSQL client
RUN apt-get update && \
    apt-get install -y postgresql-client && \
    rm -rf /var/lib/apt/lists/*
COPY app/ .
RUN pip install --no-cache-dir -r requirements.txt
COPY wait-for-postgres.sh /wait-for-postgres.sh
RUN chmod +x /wait-for-postgres.sh
CMD ["/wait-for-postgres.sh", "db", "python", "app.py"]
```

Create docker-compose.yml (in docker-compose.yml file)

```
services:
  web:
    build: .
    ports:
      - "5001:5000"
    depends_on:
      - db

  db:
    image: postgres:14
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Create wait-for-postgres.sh (in wait-for-postgres.sh file)

```
#!/bin/bash
# wait-for-postgres.sh
set -e
host="$1"
shift
cmd="$@"
until PGPASSWORD=mypassword psql -h "$host" -U "myuser" -d "mydb" -c '\q'; do
  >&2 echo "Postgres is unavailable - sleeping"
  sleep 1
done

>&2 echo "Postgres is up - executing command"
exec $cmd
```

Make it executable:

```
chmod +x wait-for-postgres.sh
```

Optional Cleanup (if any prior builds): (in terminal)

```
docker-compose down -v
```

Build & Start the Application (in terminal)

```
docker-compose up --build
```

Test in Browser

Visit:

http://localhost:5001

You should see:

```
Connected to PostgreSQL: ('PostgreSQL 14.x...',)
```