

Web Development Report for Assignment-2

Assignment: Exploring Django with Docker

Software Engineering, 2nd year
Nurzhan Momynkul
ID: 23MD0414

13.10.2024

1. Docker Compose.....	2
- Configuration.....	2
- Docker-compose.yml, brief explanation:.....	4
- Build and Run, The output in the terminal:.....	5
2. Docker Networking and Volumes.....	5
- Set Up Docker Networking.....	5
- Implement Docker Volumes.....	7
- Findings.....	7
3. Django Application Setup.....	7
- Project Structure.....	8
- Database Configuration.....	9
- Findings.....	9
Conclusion.....	10

1. Docker Compose

- Configuration

Components of the Project:

Dockerfile - Python dependency file

requirements.txt - file of dependencies to be used in Dockerfile

docker-compose.yml - file describing the build services of a multi-container application, in our case: Web (Django) and db (Postgre)

```
docker-compose.yml U .env U Dockerfile U X
django-docker > Dockerfile > ...
1 FROM python:3.9-slim
2
3 # Set the working directory in the container
4 WORKDIR /code
5
6 # Copy requirements.txt
7 COPY requirements.txt /code/
8
9 # Install dependencies
10 RUN pip install -r requirements.txt
11
12 # Copy the rest of the project files
13 COPY . /code/
14
15 # Expose the default Django port
16 EXPOSE 8000
```

```
docker-compose.yml U .env U
django-docker > requirements.txt
1 Django>=3.2,<4.0
2 psycopg2-binary>=2.9,<3.0
3 |
```

version: '3'

```
services:
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - app-network

  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./code
      - static_volume:/code/static
      - media_volume:/code/media
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_HOST: db
      DB_PORT: 5432
    networks:
      - app-network

volumes:
  postgres_data:
  static_volume:
  media_volume:
















networks:
  app-network:
```

- Docker-compose.yml, brief explanation:

The file describes two services db and web that create the application. These names then can be used in place of IP in container configs. The services are a web server and a database. The interconnection of these services, their volumes, ports, variables are described. Volumes are needed to save the results of work after containers are stopped. Ports describe the port forwarding from the local machine to the container. In variables we specify login, password, database from .env file.

- Build and Run, The output in the terminal:

```
PS N:\Murzhan\KBTU\VIII-semester\Web-app-dev-Msc\django-docker> docker-compose up --build
time="2024-10-13T14:19:07+05:00" level=warning msg="N:\Murzhan\KBTU\VIII-semester\Web-app-dev-Msc\django-docker\docker-compose.yml: the attribute 'version' is obsolete, it will
be ignored, please remove it to avoid potential confusion"
[+] Building 0.7s (11/11) FINISHED
=> [web internal] load build definition from Dockerfile
=> [web internal] load metadata for docker.io/library/python:3.9-slim
=> [web internal] load .dockerignore
=> [web internal] load build context
=> [web 1/5] FROM docker.io/library/python:3.9-slim
=> [web 2/5] WORKDIR /code
=> [web 3/5] COPY requirements.txt /code/
=> [web 4/5] RUN pip install -r requirements.txt
=> [web 5/5] COPY . /code/
=> [web] exporting to image
=> [web] exporting layers
=> [web] writing image sha256:6e4feb33adf79378c89f9a0f7ad732ee4cf41525c750a73b22426d6f454afcc
=> [web] naming to docker.io/library/django-docker-web
=> [web] resolving provenance for metadata file
[+] Running 2/2
✔ Container django-docker-db-1 Created
✔ Container django-docker-web-1 Recreated
Attaching to db-1, web-1
db-1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
db-1 |
db-1 | 2024-10-13 09:19:11.241 UTC [1] LOG: starting PostgreSQL 13.16 (Debian 13.16-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
db-1 | 2024-10-13 09:19:11.242 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1 | 2024-10-13 09:19:11.243 UTC [1] LOG: listening on IPv6 address ":::", port 5432
db-1 | 2024-10-13 09:19:11.257 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1 | 2024-10-13 09:19:11.274 UTC [27] LOG: database system was shut down at 2024-10-13 09:15:46 UTC
db-1 | 2024-10-13 09:19:11.303 UTC [1] LOG: database system is ready to accept connections
web-1 | Watching for file changes with statreloader
web-1 | [13/oct/2024 09:19:38] "GET / HTTP/1.1" 200 10697
web-1 | [13/oct/2024 09:19:38] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
web-1 | [13/oct/2024 09:19:38] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
web-1 | [13/oct/2024 09:19:38] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
web-1 | [13/oct/2024 09:19:38] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
```

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last start... ↓	Actions
<input type="checkbox"/>	 django-docker		Running (2/2)		0.95%	10 minutes ago	  
<input type="checkbox"/>	 web-1 1b752e204603 	django-docker-web <none>	Running	8000:8000 	0.95%	10 minutes ago	  
<input type="checkbox"/>	 db-1 778fa83dcd97 	postgres:13	Running		0%	10 minutes ago	  

2. Docker Networking and Volumes

- Set Up Docker Networking

```
db:
  image: postgres:13
  environment:
    POSTGRES_DB: ${DB_NAME}
    POSTGRES_USER: ${DB_USER}
```

```

    POSTGRES_PASSWORD: ${DB_PASSWORD}
volumes:
  - postgres_data:/var/lib/postgresql/data
networks:
  - app-network
web:
  build: .
  command: python manage.py runserver 0.0.0.0:8000
  volumes:
    - ./code
    - static_volume:/code/static
    - media_volume:/code/media
  ports:
    - "8000:8000"
  depends_on:
    - db
  environment:
    DB_NAME: ${DB_NAME}
    DB_USER: ${DB_USER}
    DB_PASSWORD: ${DB_PASSWORD}
    DB_HOST: db
    DB_PORT: 5432
  networks:
    - app-network

networks:
  app-network:

```

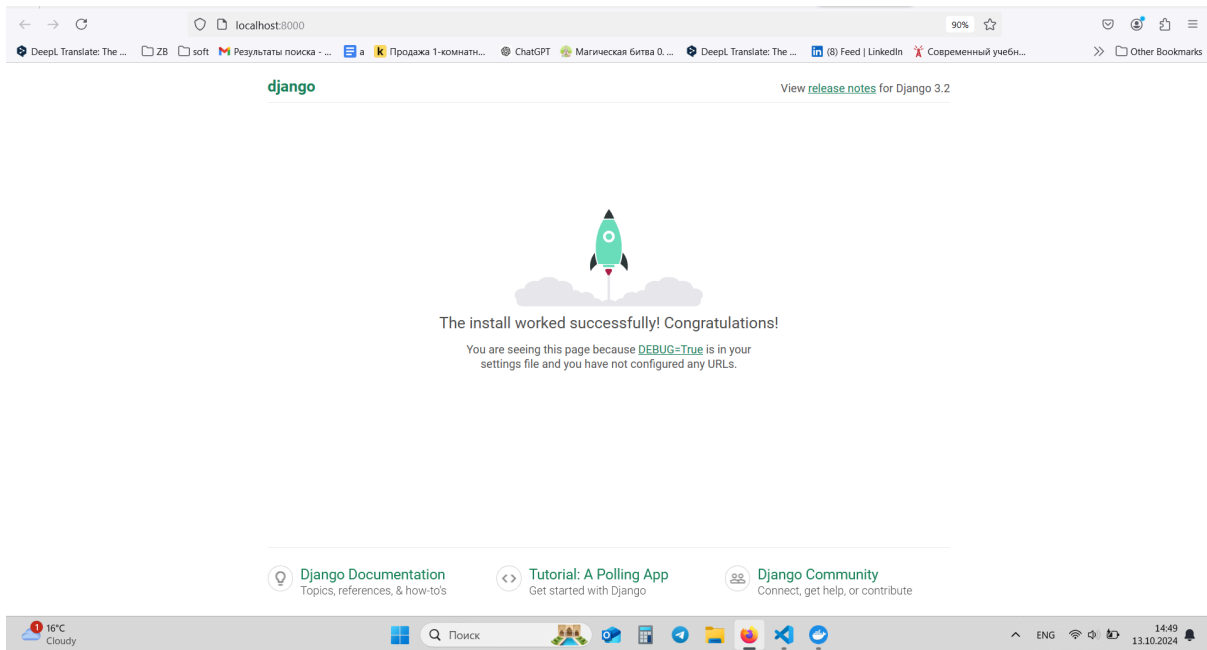
By specifying **networks: app-network**, we created a custom Docker network. Both the web (Django) and db (Postgre) services are connected to this network.

Benefits

Containers on the same network can communicate with each other by their names.

Isolation from other Docker containers outside the network. Easy scaling in the future, as all services communicate over a custom network.

Django connects to the PostgreSQL database. We can confirm this by visiting <http://localhost:8000> once the server is running.



- Implement Docker Volumes

Volumes are crucial for persisting data, there are risks that data would be lost if a container is stopped or removed. That's why we can specify volumes too:

PostgreSQL Data Volume: Persist the PostgreSQL database data using a volume so that the database retains its state even when the container is recreated.

Static and Media Files Volumes: Set up volumes for static and media files in Django, ensuring that these files persist across container restarts.

volumes:

```
postgres_data: # Volume for persisting PostgreSQL data
```

```
static_volume: # Volume for persisting static files
```

```
media_volume: # Volume for persisting media files
```

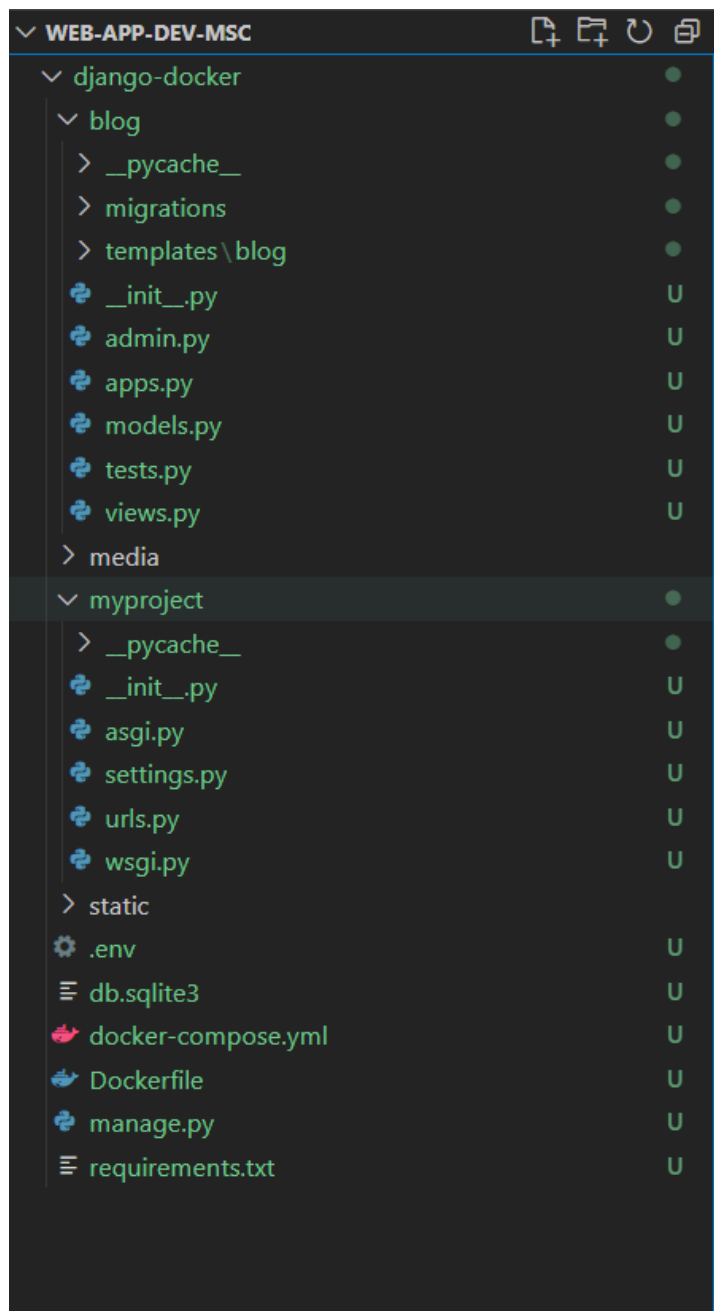
- Findings

Networking: The custom network (app-network) ensures that both the web (Django) and db (PostgreSQL) services can easily communicate using the service names (e.g., db as the hostname for PostgreSQL). This setup isolates the services from other Docker containers that may be running, reducing potential conflicts.

Volumes: Using volumes for Postgre, static, and media files ensures that data persists across container restarts. This is essential for database-driven applications and for handling user uploads or static assets. Even if the db container is removed or restarted, the database data is preserved due to the mounted volume. And any user-uploaded files or static assets are not lost between restarts, ensuring a stable environment.

3. Django Application Setup

- Project Structure



The Post model defined in models.py represents a blog post with a title, content, and timestamp.

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    def __str__(self):
        return self.title
```

Then migrations created with makemigrations and migrate commands. It will set up the database schema for the Post model.

- Database Configuration

The PostgreSQL database is configured in settings.py to connect to the database container. The HOST value is set to the service name (db), which allows Django to connect to the PostgreSQL database running in Docker.

- Findings

The ability to easily configure and connect to a PostgreSQL database generates productivity. I found out that Docker containers isolate the application and its dependencies, reducing compatibility issues. Using Docker Compose simplifies multi-container applications, making it easier to manage services like the web server and database. However, setting up the database connection correctly took some time due to potential misconfigurations in environment variables or service names. Navigating the command line interface within the Docker container was initially confusing but became easier with practice.

```
django-docker > myproject > settings.py > ...
1  import os
2
3  DATABASES = {
4      'default': {
5          'ENGINE': 'django.db.backends.postgresql',
6          'NAME': os.getenv('DB_NAME', 'postgres'),
7          'USER': os.getenv('DB_USER', 'postgres'),
8          'PASSWORD': os.getenv('DB_PASSWORD', '123123'),
9          'HOST': 'db',
10         'PORT': '5432',
11     }
12 }
13
14 INSTALLED_APPS = [
15     'django.contrib.admin',
16     'django.contrib.auth',
17     'django.contrib.contenttypes',
18     'django.contrib.sessions',
19     'django.contrib.messages',
20     'django.contrib.staticfiles',
21     'blog',
22 ]
23 DEBUG = False
24
25 ALLOWED_HOSTS = ['*']
26
```

Conclusion

In this assignment, I learned how to set up a Django application using Docker, create a PostgreSQL database, and run migrations. Docker provided a consistent environment, making it easier to manage dependencies and isolate services. The use of Docker simplified the development process and ensured that the application was portable and scalable. Overall, Docker significantly simplified the integration of Django and PostgreSQL for web development.