Exploring Django with Docker

Makhatbek Iliyas
School of Information Technology and Engineering
Kazakh-British Technical University
Almaty, Kazakhstan
Email: il_makhatbek@kbtu.kz

October 13, 2024

Contents

1 Introduction

This assignment is focused on creating a Django web application using Docker. The goal was to learn how to set up a Django project with Docker Compose, configure Docker networking, and use Docker volumes for data persistence. I also learned how to connect Django to a PostgreSQL database running in a Docker container. Below, I will explain each step of the process and show the configurations I used.

2 Docker Compose

2.1 Configuration

For the Docker configuration, I created a `docker-compose.yml` file. This file defines two main services: the Django web service and the PostgreSQL database. The web service runs Django, and the database service uses PostgreSQL.

Here is the key part of my `docker-compose.yml` file:

```yaml
services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    environment:
      - DB_NAME=django_db
      - DB_USER=iliyas
      - DB_PASSWORD=iliyaspassword
      - DB_HOST=db
    depends_on:
      - db

  db:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=django_db
      - POSTGRES_USER=iliyas
      - POSTGRES_PASSWORD=iliyaspassword

volumes:
  postgres_data:
```

2.2 Build and Run

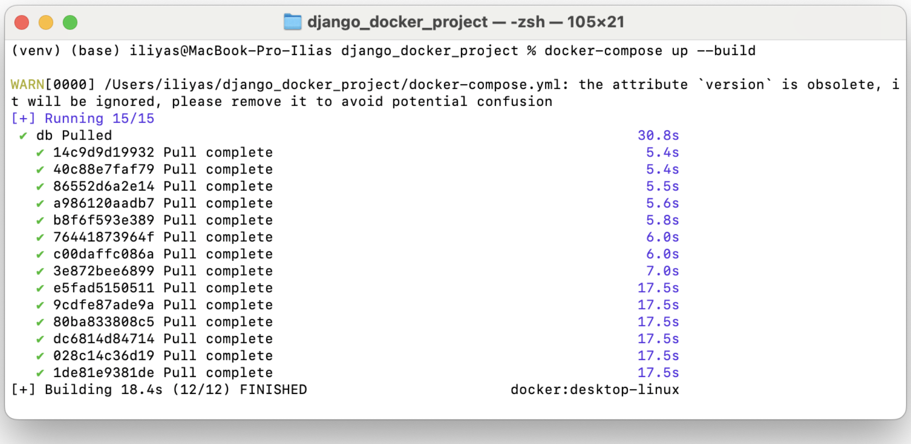
To build and run the containers, I used the command:

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

This command builds the containers and starts the application. After the build, I checked that the application was running by going to `http://localhost:8000`, and I saw the default Django welcome page.

2.3 Challenges faced

There was a small warning regarding the `version` attribute in the `docker-compose.yml`, but it didn't affect the functionality. I removed it later to avoid any future confusion.



```
(venv) (base) iliyas@MacBook-Pro-Ilias django_docker_project % docker-compose up --build

WARN[0000] /Users/iliyas/django_docker_project/docker-compose.yml: the attribute `version` is obsolete, i
t will be ignored, please remove it to avoid potential confusion
[+] Running 15/15
 ✔ db Pulled                                                                    30.8s
   ✔ 14c9d9d19932 Pull complete                                                  5.4s
   ✔ 40c88e7faf79 Pull complete                                                  5.4s
   ✔ 86552d6a2e14 Pull complete                                                  5.5s
   ✔ a986120aadb7 Pull complete                                                  5.6s
   ✔ b8f6f593e389 Pull complete                                                  5.8s
   ✔ 76441873964f Pull complete                                                  6.0s
   ✔ c00daffc086a Pull complete                                                  6.0s
   ✔ 3e872bee6899 Pull complete                                                  7.0s
   ✔ e5fad5150511 Pull complete                                                 17.5s
   ✔ 9cdfe87ade9a Pull complete                                                 17.5s
   ✔ 80ba833808c5 Pull complete                                                 17.5s
   ✔ dc6814d84714 Pull complete                                                 17.5s
   ✔ 028c14c36d19 Pull complete                                                 17.5s
   ✔ 1de81e9381de Pull complete                                                 17.5s
[+] Building 18.4s (12/12) FINISHED                               docker:desktop-linux
```

Figure 1: Terminal Output from `docker-compose up --build`

3 Docker Networking and Volumes

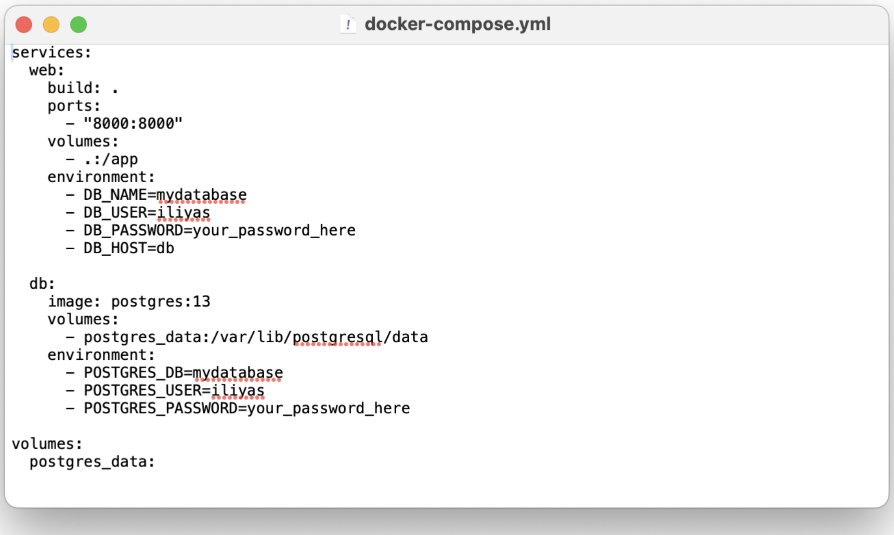
3.1 Networking Setup

In Docker Compose, services are automatically connected through an internal network, so I didn't need to do much. By default, the Django web service connects to the PostgreSQL service using the hostname `db` (as configured in the environment variable `DB_HOST`).

3.2 Volume Setup

For data persistence, I set up a volume for PostgreSQL. This ensures that even if the PostgreSQL container is stopped or removed, the database data will still be available. The volume is defined in the `docker-compose.yml` file under the `volumes` section.

Additionally, I used a volume for the Django application so that the code and any uploaded files or static files could be saved on the host system.



```
services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    environment:
      - DB_NAME=mydatabase
      - DB_USER=iliyas
      - DB_PASSWORD=your_password_here
      - DB_HOST=db

  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=mydatabase
      - POSTGRES_USER=iliyas
      - POSTGRES_PASSWORD=your_password_here

volumes:
  postgres_data:
```

Figure 2: Updated `docker-compose.yml` with Volumes

3.3 Findings

Using Docker networking allowed the services (Django and PostgreSQL) to communicate easily without manual network configuration. Docker volumes were helpful because they preserved the database data and project files between container restarts.

4 Django Application Setup

4.1 Project Structure

Inside the Django container, I created a new Django project by running the following command:

```
django-admin startproject myproject
```

Then, I created a simple app named `blog` using this command:

```
python manage.py startapp blog
```

This app has one model called `Post`, which stores a title and content. Here is a simple example of the `Post` model:

```python
from django.db import models

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

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

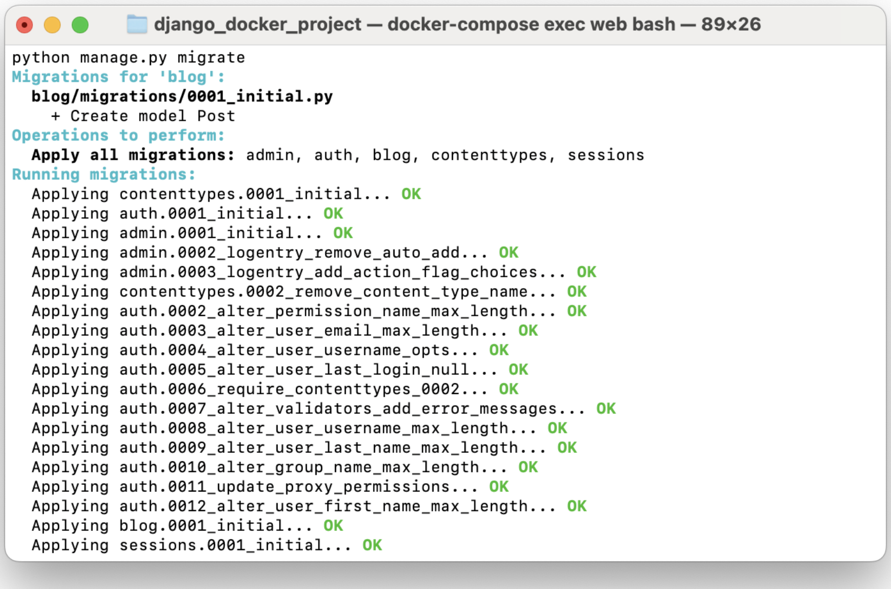
4.2 Database Configuration

In the Django settings, I updated the database configuration to connect to the PostgreSQL database running in the `db` container. The `settings.py` file was modified as follows:

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME'),
        'USER': os.environ.get('DB_USER'),
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': os.environ.get('DB_HOST'),
        'PORT': '5432',
    }
}
```

After configuring the database, I ran the migrations using this command:

```
python manage.py migrate
```


django_docker_project — docker-compose exec web bash — 89×26

```
python manage.py migrate
Migrations for 'blog':
  blog/migrations/0001_initial.py
    + Create model Post
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying blog.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Figure 3: Terminal Output from Running Migrations

4.3 Findings

Setting up Django with Docker was straightforward after configuring the `docker-compose.yml` file. I was able to easily manage the environment variables and make sure that Django connected to PostgreSQL. Running migrations in Docker was also similar to running them on a local setup, and the volume setup ensured data persistence.

5 Conclusion

This assignment gave me hands-on experience with Docker, Docker Compose, and setting up a Django application in a containerized environment. I learned how to configure services, set up networking between containers, and use Docker volumes to persist data. Working with Docker made it easier to manage dependencies and create an isolated development environment for Django and PostgreSQL.

6 References

- Docker Documentation: `https://docs.docker.com`
- Django Documentation: `https://docs.djangoproject.com`
- PostgreSQL Documentation: `https://www.postgresql.org/docs/`