**PART A**

**Experiment No. 11**

**A.1 Aim: Deploying a Django Application with MySQL Using Docker**

**Objective:**  To understand how to deploy a Django application using MySQL database within Docker containers and establish connectivity between them using direct network links.

**A.2 Prerequisite:**

* Knowledge of Docker, Docker networking, Django, and MySQL.
* Basic understanding of web applications and REST APIs.

**A.3 Outcome:**  
After successful completion of this experiment, students will be able to:

* Deploy Django applications within a Docker container.
* Connect the Django application to MySQL running in another container.
* Understand the process of containerization and networking in Docker.

**A.4 Theory**  
In this experiment, we deploy a Django application that uses MySQL as its database. Each service runs in its separate container, and the Django app will connect to the MySQL container over the Docker network. This approach ensures portability and consistency across environments.

**Steps for Setting up Django with MySQL using Docker:**

1. **Create a Docker Network**  
   To allow both containers to communicate, create a network:

docker network create **my\_django\_network**

1. **Set up the MySQL Container**  
   Run the MySQL container using the official image, specifying the network and environment variables:

docker run --name mysql-db --network my\_django\_network

-e MYSQL\_ROOT\_PASSWORD=rootpass

-e MYSQL\_DATABASE=mydb

e MYSQL\_USER=myuser

-e MYSQL\_PASSWORD=mypassword

-d **mysql:latest**

1. **Build the Django Application Container**  
   Create a Dockerfile for your Django app:

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt /app/

RUN pip install -r requirements.txt

COPY . /app/

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

Build the Docker image:

docker build -t django-app . (Don’t forget the dot)

1. **Run the Django Container**  
   Start the Django container, connecting it to the same Docker network:

docker run --name django-app --network **my\_django\_network**

-e DB\_HOST=mysql-db

-e DB\_USER=myuser

-e DB\_PASSWORD=mypassword

-e DB\_NAME=mydb

-p 8000:8000 django-app

1. **Configure Django Settings**  
   Update the DATABASES section of settings.py to use environment variables:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': os.getenv('DB\_NAME'),

'USER': os.getenv('DB\_USER'),

'PASSWORD': os.getenv('DB\_PASSWORD'),

'HOST': os.getenv('DB\_HOST'),

'PORT': '3306',

}

}

Or

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'your\_database\_name',

'USER': 'your\_username',

'PASSWORD': 'your\_password',

**'HOST': 'localhost', # or your database host**

'PORT': '3306', # default MySQL port

}

}

1. **Run Migrations**  
   Once the containers are up, run the migrations to set up the database schema:

docker exec -it django-app python manage.py migrate

1. **Access the Application**  
   Open your browser and navigate to http://localhost:8000 to access the Django app connected to the MySQL database.

**Scenario Problem Statement:**  
**Problem Statement 1:** Deploy a Django-based web application with a MySQL database using Docker. Ensure the Django app can communicate with the MySQL database container.  
**Hints:**

* Use Docker networking to establish communication between the Django and MySQL containers.
* Ensure correct environment variables are set for database connection.
* Use Django’s migration feature to set up the database schema.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. : S020 | Name: Husain Chhil |
| Class : MBA Tech Data Science | Batch : J1 |
| Date of Experiment : 05-11-2024 | Date/Time of Submission : 05-11-2024 |
| Grade : |  |

**B.1 Code:**

**Dockerfile:**

*# Use the official Python image*

FROM **python**:3.12

*# Set the working directory*

WORKDIR /app

*# Copy the requirements file and install dependencies*

COPY requirements.txt /app/

RUN pip install -r requirements.txt

*# Copy the project files*

COPY . /app/

*# Run Django's development server on port 8000*

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

**Settings.py**

"""

Django settings for myproject project.

Generated by 'django-admin startproject' using Django 5.1.1.

For more information on this file, see

https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.1/ref/settings/

"""

from pathlib import Path

import os

*# Build paths inside the project like this: BASE\_DIR / 'subdir'.*

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

*# Quick-start development settings - unsuitable for production*

*# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/*

*# SECURITY WARNING: keep the secret key used in production secret!*

SECRET\_KEY = 'django-insecure-kq&wh#z6pxalkoc9u!9xv5!p$\*-v$8wj#6kx@\*%l40r)4m8i)a'

*# SECURITY WARNING: don't run with debug turned on in production!*

DEBUG = True

ALLOWED\_HOSTS = []

*# Application definition*

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'myproject.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'myproject.wsgi.application'

*# Database*

*# https://docs.djangoproject.com/en/5.1/ref/settings/#databases*

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': os.getenv('DB\_NAME', 'mydb'),

        'USER': os.getenv('DB\_USER', 'myuser'),

        'PASSWORD': os.getenv('DB\_PASSWORD', 'mypassword'),

        'HOST': os.getenv('DB\_HOST', 'mysql-db'),  *# Hostname of the MySQL container*

        'PORT': '3306',  *# Port mapped for MySQL on host*

    }

}

*# Password validation*

*# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators*

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

*# Internationalization*

*# https://docs.djangoproject.com/en/5.1/topics/i18n/*

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

*# Static files (CSS, JavaScript, Images)*

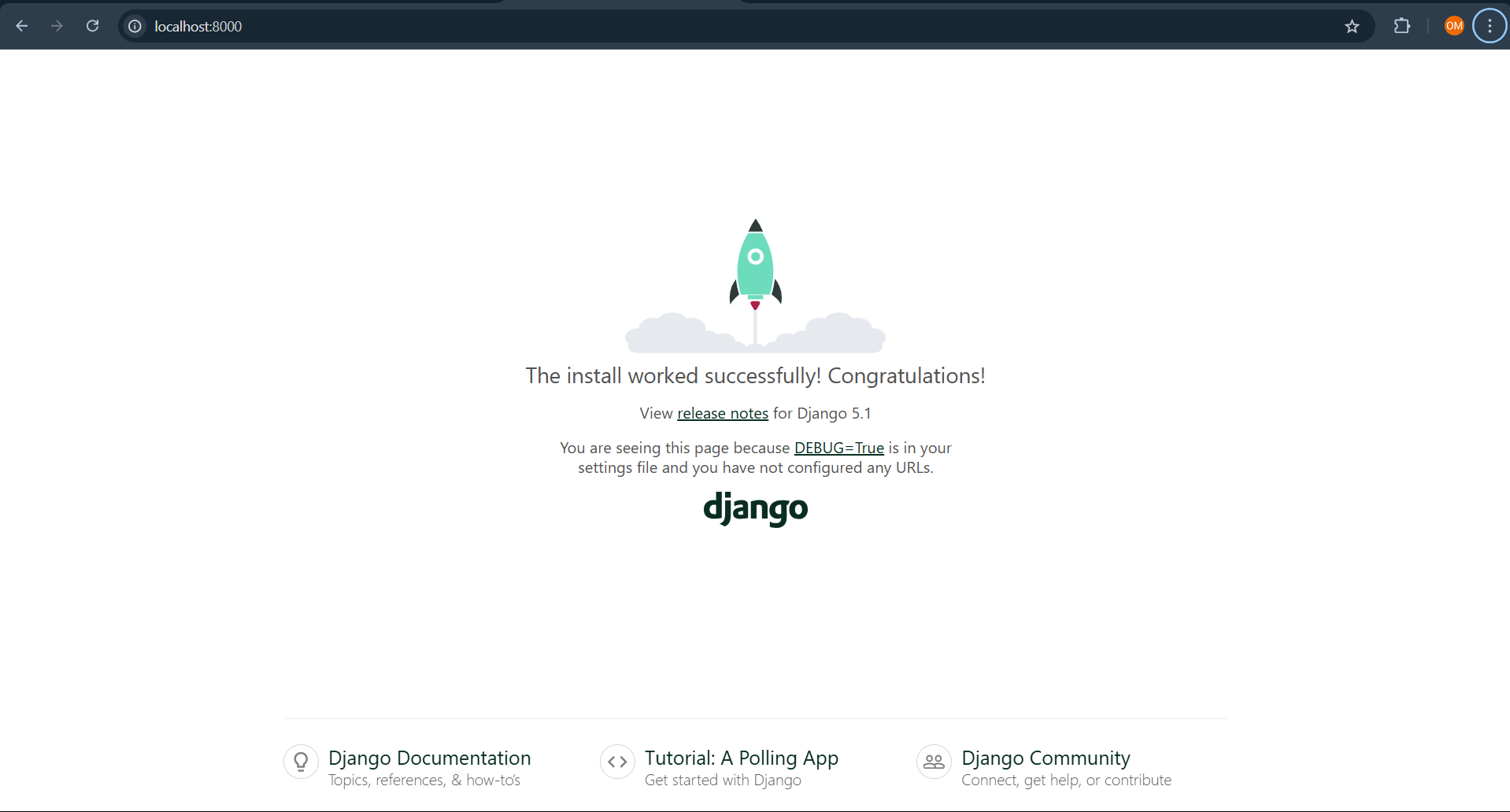
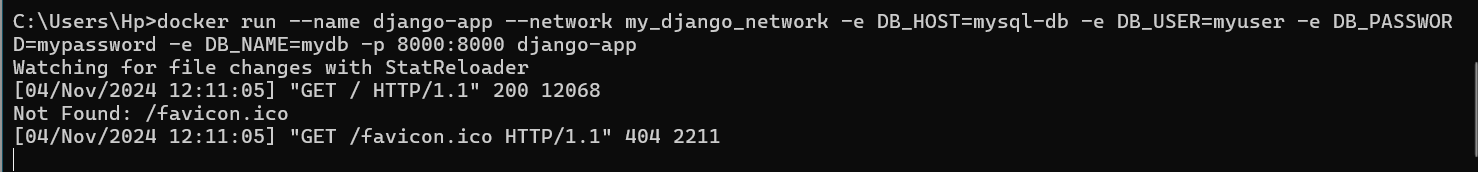
*# https://docs.djangoproject.com/en/5.1/howto/static-files/*

STATIC\_URL = 'static/'

*# Default primary key field type*

*# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field*

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

**B.2 Output**

**B.3 Conclusion:**

Create and connected a Django Project and mysql using Docker

**B.3 Observations and Learning:**

Deployment of Projects on Docker

**B.4 Question of Curiosity**

1. How do you establish communication between the Django and MySQL containers using Docker?

Ans:To enable communication between the Django and MySQL containers, you can:

1. **Create a Docker network** (e.g., docker network create my\_django\_network).
2. **Attach both containers to this network** using the --network option when you run each container (e.g., docker run --network my\_django\_network).
3. **Reference the MySQL container by its container name** (e.g., mysql-db) as the DB\_HOST in Django’s settings, so Django can locate it within the network.

Containers on the same network can communicate via internal Docker DNS using container names as hostnames, facilitating easy access without exposing ports to the host.

1. What are the benefits of containerizing web applications and databases?

Ans:

 **Consistency and Portability**: Containers provide a consistent environment across development, testing, and production. Applications run identically across different setups.

 **Isolation**: Each service runs in an isolated container, preventing interference and making it easier to manage dependencies.

 **Scalability**: Containers enable easy horizontal scaling. For example, adding more instances of a service can be automated.

 **Efficient Resource Usage**: Containers are lightweight compared to virtual machines, so they use resources more efficiently.

 **Simplified Deployment and Management**: Containers simplify deployment and rollback procedures, especially in CI/CD pipelines.

1. How do environment variables enhance the portability of Dockerized applications?

Ans:

 Environment variables allow configuration values (e.g., database credentials, ports, hostnames) to be set externally, making the application’s code independent of specific values.

 They provide flexibility for developers to change settings easily across environments (development, staging, production) without modifying code.

 This approach enhances **security** by keeping sensitive data like passwords out of the codebase and **portability** by letting the same container image run in different environments with environment-specific variables.

1. Why is it essential to use docker network when setting up multi-container applications?

Ans:

 Docker networks provide isolated communication channels for containers, enabling seamless communication between them while keeping external access optional and secure.

 They allow containers to communicate using internal DNS, referencing each container by its name within the network.

 Docker networks also enhance security by limiting which containers can access each other, reducing the attack surface.

1. What is docker-compose? What is the significance of running migrations in Django using docker-compose.xml?

Ans:

 **Docker Compose** is a tool that defines and manages multi-container Docker applications using a docker-compose.yml file. This file contains configuration for each containerized service, allowing you to start, stop, and manage them together as a unified application.

 **Running migrations**: In a Django app, migrations need to be run after setting up the database to ensure the database schema is up to date with the models. Using docker-compose, you can specify a command to run Django migrations automatically or run migrations manually via docker-compose run or docker-compose exec. This ensures that the database is initialized properly, preventing errors due to missing tables or fields.