DOCKER-COMPOSE.YAML

version: '3.8'

services:

db:

image: postgres:13

environment:

POSTGRES\_DB: ${POSTGRES\_DB}

POSTGRES\_USER: ${POSTGRES\_USER}

POSTGRES\_PASSWORD: ${POSTGRES\_PASSWORD}

volumes:

- postgres\_data:/var/lib/postgresql/data

networks:

- mynetwork

web:

build: .

command: bash -c "python manage.py migrate && python manage.py runserver 0.0.0.0:9000"

volumes:

- .:/app

ports:

- "9000:9000"

depends\_on:

- db

networks:

- mynetwork

volumes:

postgres\_data:

networks:

mynetwork:

driver: bridge

#### 1. ****Version****

Version: “3.8”

**Purpose**: Specifies the version of Docker Compose syntax to use. Version 3.8 is a widely supported version that includes all the features needed for most common use cases.

2. **Services**

The services section defines the containers (or services) that Docker Compose will manage. Here we have two services: db (PostgreSQL database) and web (Django web application)

* **db (PostgreSQL Database)**

db:

image: postgres:13

environment:

POSTGRES\_DB: ${POSTGRES\_DB}

POSTGRES\_USER: ${POSTGRES\_USER}

POSTGRES\_PASSWORD: ${POSTGRES\_PASSWORD}

volumes:

- postgres\_data:/var/lib/postgresql/data

networks:

- mynetwork

db:: The name of this service is db. You can refer to this service in other parts of your configuration, such as the web service.

* image: postgres:13: Specifies the Docker image to use for the db service. Here, it is using the official PostgreSQL 13 image from Docker Hub.
* environment:: Defines environment variables for the db service. The following environment variables are set here:
* POSTGRES\_DB: The name of the database to create on startup (usually you would use a variable from an .env file, like ${POSTGRES\_DB}).
* POSTGRES\_USER: The username to connect to the database.
* POSTGRES\_PASSWORD: The password for the PostgreSQL user.
* volumes:: This maps a named volume (postgres\_data) to the PostgreSQL data directory in the container (/var/lib/postgresql/data):
* The named volume postgres\_data is defined below in the volumes section, and it persists the PostgreSQL data outside of the container, ensuring that the data is retained between container restarts.
* networks:: This defines which network the db container is connected to. Here, it is part of the mynetwork.

**web (Django Application)**

web:

build: .

command: bash -c "python manage.py migrate && python manage.py runserver 0.0.0.0:9000"

volumes:

- .:/app

ports:

- "9000:9000"

depends\_on:

- db

networks:

- mynetwork

web:: The name of this service is web. This will run your Django application.

* build: .: Specifies the build context. The . means the current directory. Docker Compose will look for a Dockerfile in the current directory and build an image from it.
* command:: Defines the command that will be run when the container starts. Here, it runs two commands:
* python manage.py migrate: This applies the database migrations.
* python manage.py runserver 0.0.0.0:9000: This starts the Django development server on port 9000. The 0.0.0.0 allows the server to be accessed from outside the container.
* volumes:: The volume mounts the current directory (.) from your local machine to /app in the container. This makes your local code accessible within the container. Any changes made to your code locally will immediately reflect inside the container.
* ports:: Maps port 9000 inside the container to port 9000 on your host machine. This allows you to access the Django app by visiting http://localhost:9000 from your browser.
* depends\_on:: Specifies that the web service depends on the db service. This means Docker Compose will wait for the db service to be started before starting the web service.
* networks:: Connects the web container to the same mynetwork network as the db container, allowing the two containers to communicate with each other.

3. **Volumes**

volumes:

postgres\_data:

volumes: **defines persistent storage volumes. In this case,** postgres\_data **is a named volume used to store the database data outside of the container.**

**4. Networks**

networks:

mynetwork:

driver: bridge

networks: **defines custom networks to allow containers to communicate. Here, it creates a network named** mynetwork **using the** bridge **driver, which is the default Docker network type. Both the** db **and** web **services are connected to this network.**