1.-Crear una carpeta llamado docker_django

2.-dentro de la carpeta crear un archivo de requirements.txt con el siguiente

contenido:

Django>=2.0,<3.0

psycopg2>=2.7,<3.0

3.-Ahora creamos un archivo Dockerfile:

FROM python:3

ENV PYTHONUNBUFFERED 1

RUN mkdir /code

WORKDIR /code

COPY requirements.txt /code/

RUN pip install -r requirements.txt

COPY . /code/

4.-vamos a crear una carpeta llamada .envs y dentro de ella un archivo llamado .postgres donde guardaremos nuestras variables de entorno que se utilizarán más adelante en el contenedor de postgres: este es el contenido:

PostgreSQL

POSTGRES_HOST=db

POSTGRES_PORT=5432

POSTGRES_DB=mydb

POSTGRES_USER=Ds5DTAxP

POSTGRES_PASSWORD=5nFyHgRtx8z3Mf5kcar8d2D4yQrVgFE2

5.-Ahora crearemos el archivo docker-compose.yml.

version: '3'

services:

  db:

```
  image: postgres
  env_file:
   - ./.envs/.postgres
web:
  build: .
  command: python manage.py runserver 0.0.0.0:8000
  env_file:
   - ./.envs/.postgres
  volumes:
   - .:/code
  ports:
   - "8000:8000"
  depends_on:
   - db
```

6-Una vez hecho esto crearemos el proyecto con la siguiente orden, sustituir <nombre-proyecto> por el nombre que le queráis poner.

```
docker-compose run web django-admin startproject ecommerce .
```

7.-Para finalizar la construcción del contenedor lanzaremos la ordern build

```
docker-compose -f docker-compose.yml build
```

8.-Ahora que ya tenemos el proyecto creado, modificaremos el archivo settings.py del proyecto de Django, para ello en nuestra carpeta raíz, buscamos en <nombre-proyecto/settings.py y realizamos las siguientes modificaciones para poder comunicarnos con nuestra base de datos.

```
DATABASES = {
  'default': {
     'ENGINE': 'django.db.backends.postgresql',
```

```
        'NAME': os.environ['POSTGRES_DB'],
        'USER': os.environ['POSTGRES_USER'],
        'PASSWORD': os.environ['POSTGRES_PASSWORD'],
        'HOST': os.environ['POSTGRES_HOST'],
        'PORT': os.environ['POSTGRES_PORT'],
    }
}
```

modifica esta linea tambien:#USE_TZ = True

9.-Con esto ya estaría todo listo, lanzamos up para levantar el servicio y ya tendríamos nuestro proyecto completamente listo.

docker-compose up

10.-Ahora accedemos a http://localhost:8000 y ya podremos ver nuestro proyecto en funcionamiento.

11.-Bueno bueno, esto está muy bien pero cada vez que paremos el contenedor perderemos los datos almacenados en la base de datos por lo que sería un rollazo tener que estar creando los datos de nuevo cada vez que corramos nuestro contenedor. Para solucionar esto abrimos el archivo docker-compose.yml y añadiremos unas modificaciones en el servicio db.

```
version: '3'

services:
  db:
    image: postgres
    env_file:
      - ./.envs/.postgres
    volumes:
      - /opt/postgres-data:/var/lib/postgresql/data
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
```

```
    env_file:
      - ./.envs/.postgres
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
```

12.-Lanzamos de nuevo el build y el up y ya estaría listo.

docker-compose -f docker-compose.yml build

docker-compose up

13.-Para comprobar que podemos acceder al admin de Django haremos el makemigrations y migrate para crear las tablas en la base de datos, para ello ahora lo tendremos que lanzar de esta forma:

docker-compose -f .\docker-compose.yml run --rm web python manage.py makemigrations

docker-compose -f .\docker-compose.yml run --rm web python manage.py migrate

14.-Y ahora crearemos el super usuario para poder entrar al panel con la siguiente orden:

docker-compose -f .\docker-compose.yml run --rm web python manage.py createsuperuser

15.-Inicializa tu contenedor y prueba el administrador:

docker-compose up

http://localhost:8000/admin

creando la vista

==========

1>ingrese a la carpeta del proyecto principal ecommerce

y cree el app:

docker-compose run web python manage.py startapp core

2>agregue el nombre de la aplicacion en settings

3>cree los modelos

4>sal de la carpeta ecoomerce y regresa a la principal y migra los nuevos modelos de la aplicacion cors

docker-compose -f .\docker-compose.yml run --rm web python manage.py makemigrations

docker-compose -f .\docker-compose.yml run --rm web python manage.py migrate

5>agrega en el archivo admin que aparezca lo modelos en el admin

6>sube los cambios y prueba

docker-compose -f docker-compose.yml build

docker-compose up

7>terminar de subir todo el proyecto y carga los cambios luego

docker-compose -f docker-compose.yml build

docker-compose up