Introducción a Docker  
UD 06. Caso práctico 02 - Django con Docker Compose

short line



Autor: Sergi García Barea

Actualizado Febrero 2025

Licencia

**Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA)**: No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

**1.** [**Introducción**](#_dbh0n1vac4c8) **3**

**2.** [**Ficheros “Dockerfile” y “requirements.txt” del caso práctico**](#_vyhbfp4t666x) **3**

**3.** [**Fichero “docker-compose.yml” del caso práctico**](#_ejyez7bpu11s) **3**

**4.** [**Paso 1: Poniendo en marcha el sistema**](#_3b05p2f899fh) **4**

**5.** [**Paso 2: Creando proyecto Django y conectando a la base de datos**](#_af8o4jtq1igj) **5**

**6.** [**Paso 3: Reiniciando el sistema**](#_z42tus8s8rh0) **5**

**7.** [**Bibliografía**](#_aktea99seyu7) **7**

UD06. Caso práctico 02

# Introducción

En este caso práctico vamos a poner en marcha un servidor web en Python con Django, conectado a una base de datos PostgreSQL. Construiremos la imagen del servidor a partir de un Dockerfile y estableceremos que tenga persistencia tanto el código de la aplicación como el contenido de la base de datos.

# Ficheros “Dockerfile” y “requirements.txt” del caso práctico

El contenido del fichero “***Dockerfile***” que incluimos comentado, es el siguiente:

| *#Imagen base* **FROM** python:3.7 *#Definimos la variable de entorno PYTHONBUFFERED* **ENV** PYTHONUNBUFFERED=1 *#Establecemos como directorio de trabajo /code* **WORKDIR** /code *#Copiamos requirements.txt del anfitrión a la imagen* **COPY** requirements.txt /code/ *#Instalamos las dependencias de Python indicadas en requirements* **RUN** pip install -r requirements.txt |
| --- |

Básicamente, este Dockerfile, a partir de la versión 3 de la imagen “***python***”, establece un directorio de trabajo (/code), copia del anfitrión “***requirements.txt***” (que contiene dependencias que deseamos instalar de Python) y las instala usando “pip”.

El contenido del fichero “***requirements.txt***” es el siguiente:

| Django>=3.0,<4.0 psycopg2-binary>=2.8 |
| --- |

El resultado de construir esta imagen, será una imagen con Python 3, Django con una versión de la rama 3.X y la biblioteca psycopg2 con una versión superior o igual a la 2.8.

# Fichero “docker-compose.yml” del caso práctico

El contenido del fichero “***docker-compose.yml***” que incluimos comentado, es el siguiente:

| *#Versión del fichero docker-compose 3.9. No obligatorio desde la versión de docker-compose 1.27.0* version: "3.9"  *#Indicamos los servicios* services:  *#Base de datos*  db:  *#Se basa en Postgres*  image: postgres  *#Guarda la persistencia de la base de datos en el directorio*  *#./datos/db de donde lancemos Docker Compose*  volumes:  - ./datos/db:/var/lib/postgresql/data  *#Establece variables de entorno para indicar base de datos, usuario y password*  environment:  - POSTGRES\_DB=postgres  - POSTGRES\_USER=postgres  - POSTGRES\_PASSWORD=postgres  *#Crea una aplicación web con Django*  web:  *#Construye la imagen a partir de un Dockerfile del directorio actual*  build: .  *#Comando por defecto al crear contenedor, lanzar manage.py para que*  *#lance el servidor web con Django en el puerto 8000*  command: python manage.py runserver 0.0.0.0:8000  *#Mapea el código del proyecto Django*  *# dentro de la carpeta ./codigo del anfitrión*  volumes:  - ./codigo:/code  *#Enlaza puerto 8000 de contenedor con puerto 8000 de anfitrión*  ports:  - "8000:8000"  *#Este contenedor depende de "db"*  depends\_on:  - db |
| --- |

En este caso concreto, lo que estamos haciendo es:

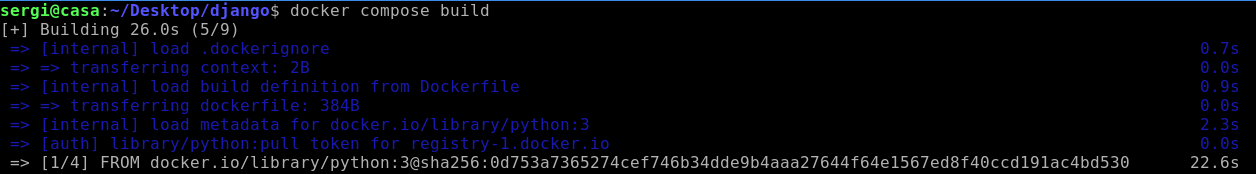
* Poner en marcha la base de datos.
* Enlazar la persistencia de la base de datos a la carpeta local “***./datos/db***”, que se creerá en el directorio local donde lancemos “***Docker Compose***”.
* Crear una imagen a partir del “***Dockerfile***” de nuestro directorio actual y una vez creada:
  + Si se cumple la dependencia con “***db***”, lanzar un contenedor con dicha imagen.
  + Establecer como comando de inicio del contenedor el comando para iniciar el servidor web Python con “***Django***”.
  + Enlazar la persistencia del código del servidor a una carpeta “***./codigo***” que se creará en el directorio local donde lancemos “***Docker Compose***”-
  + Mapee puerto 8000 de contenedor con puerto 8000 del anfitrión.

# Paso 1: Poniendo en marcha el sistema

Previamente a poner en marcha el sistema y de manera opcional, podemos usar el comando

| docker compose build |
| --- |

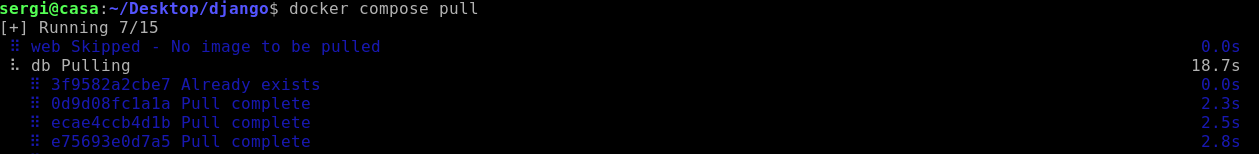
para que construya la imagen del Dockerfile previamente a lanzar el servicio, obteniendo:



Sí queremos descargar imágenes ya creadas antes de poner en marcha el sistema, con:

| docker compose pull |
| --- |

Si las descargamos, obtenemos algo similar a:



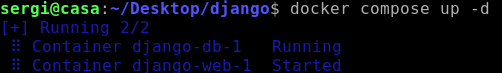
Para poner en marcha el sistema, simplemente nos situamos en el directorio donde tengamos el fichero “***docker-compose.yml***” de este caso práctico y escribimos:

| docker compose up -d |
| --- |

La opción “***-d***” indica que “***Docker Compose***” se ejecute en segundo plano.

La opción “***up***”, descarga y construye imágenes (si no estaban ya). Tras ello lanza los contenedores asociados, siguiendo orden de dependencia.

Si todo ha ido bien, obtendremos un mensaje similar a este:



# Paso 2: Creando proyecto Django y conectando a la base de datos

Si tras el paso anterior, intentamos acceder a <http://localhost:8000>, veremos que no se puede acceder. Eso es porque no se está sirviendo ningún proyecto Django y deberemos crear una base.

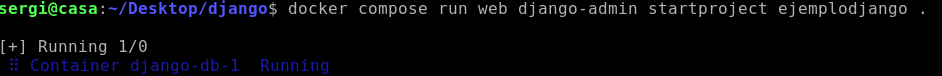
Podremos crearla con el siguiente comando

| docker compose run web django-admin startproject ejemplodjango . |
| --- |

En este comando las opciones indicadas son:

* “***run***” indica que ejecutaremos un comando.
* “***web***” indica para qué servicio es el comando.
* “***django-admin startproject ejemplodjango .***” es un comando que crea un proyecto con nombre “***ejemplodjango***” en el directorio actual del contenedor (indicado por “***.***”). Recordemos que el directorio actual es “***/code***”, que fue definido en el “***Dockerfile***”.

Obtendremos algo similar a:



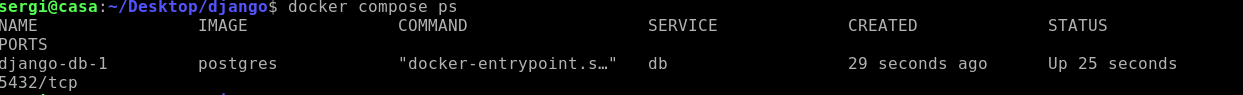
Tras esto aún no veremos nada, comentamos en el punto siguiente el porqué de esto y como solucionarlo.

# Paso 3: Reiniciando el sistema

Si lanzamos el siguiente comando

| docker compose ps |
| --- |

Observamos lo siguiente:



Vemos que el servidor de la web no está en ejecución (está detenido). Eso es porque en el momento de lanzarlo no existía el proyecto Django y el comando por defecto del contenedor que habíamos definido en “***docker-compose.yml***” (“***python manage.py runserver 0.0.0.0:8000***”) no podía lanzarse, ya que “***manage.py***” no existía.

Si ahora visitamos nuestro directorio “***/codigo***” del anfitrión, observamos que tenemos mapeado el proyecto Django ahí y podemos modificarlo desde nuestra máquina anfitriona.

Como el usuario del contenedor es “***root***”, todos los ficheros mapeados pertenecen a “***root***”, pero podemos cambiarlos a un usuario local nuestro para facilitar la tarea con un comando similar a:

| sudo chown -R $USER:$USER ./codigo |
| --- |

Al crear el proyecto, por defecto Django utiliza una base de datos **SQLite** en un fichero, por lo cual el servidor de bases de datos que hemos lanzado, no tiene ningún uso.

Antes de relanzar el proyecto, podemos modificar el fichero “***./codigo/ejemplodjango/settings.py***” y comentar/eliminar la conexión a base de datos **SQLite** e indicar una conexión a la base de datos **PostgreSQL**. Para ello, el siguiente código lo comentaremos o eliminaremos:

| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.sqlite3',  'NAME': BASE\_DIR / 'db.sqlite3',  } } |
| --- |

y en su lugar colocaremos:

| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.postgresql',  'NAME': 'postgres',  'USER': 'postgres',  'PASSWORD': 'postgres',  'HOST': 'db',  'PORT': 5432,  } } |
| --- |

Los datos de este fichero, deben coincidir con los indicados como variables de entorno que hemos colocado en “***docker-compose.yml***”.

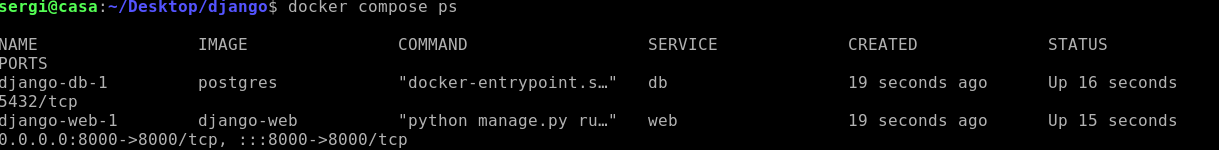
Tras este cambio, paramos el sistema y lo arrancamos de nuevo con

| docker compose down; docker compose up -d |
| --- |

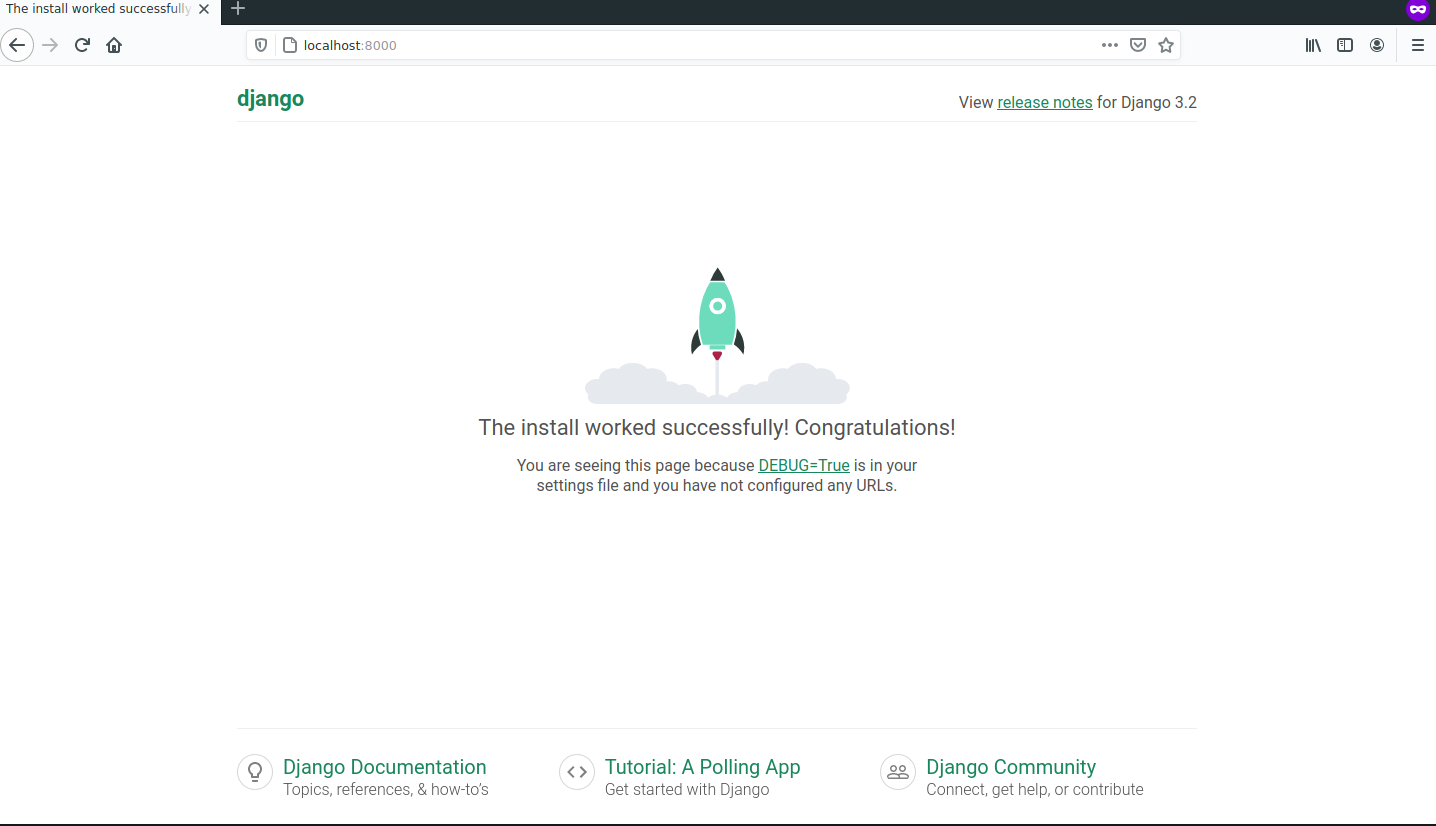
Podemos comprobar, que al existir ya el proyecto **Django**, el contenedor no está parado:

| docker compose ps |
| --- |

Y observamos que así es con algo similar a



Tras ello, accedemos a <http://localhost:8000> y observamos que todo funciona adecuadamente:



Finalmente, recordar que la persistencia de la base de datos la tenemos enlazada a nuestro directorio “***./datos/bd***” y el código que podemos manipular dentro de “***./codigo***”.

# Bibliografía

[1] Docker Docs <https://docs.docker.com/>

[2] Docker Compose Docs <https://docs.docker.com/compose/>