### Añadir archivos de configuración

Cada plataforma de alojamiento requiere alguna configuración para que un proyecto se ejecute correctamente en sus servidores. En esta sección, añadiremos tres archivos de configuración:

*.platform.app.yaml* Este es el archivo de configuración principal del proyecto. Indica a Platform.sh qué tipo de proyecto estamos intentando desplegar y qué tipo de recursos necesita nuestro proyecto, e incluye comandos para construir el proyecto en el servidor.

*.platform/routes.yaml* Este archivo define las rutas a nuestro proyecto. Cuando Platform.sh recibe una solicitud, ésta es la configuración que ayuda a dirigir estas solicitudes a nuestro proyecto específico.

*.platform/services.yaml* Este archivo define cualquier servicio adicional que necesite nuestro proyecto.

Todos estos son archivos YAML (YAML Ain't Markup Language). *YAML* es un lenguaje diseñado para escribir archivos de configuración; está hecho para ser leído fácilmente tanto por humanos como por ordenadores. Puedes escribir o modificar a mano un archivo YAML típico, pero un ordenador también puede leerlo e interpretarlo sin ambigüedades.

Los archivos YAML son magníficos para la configuración del despliegue, porque te dan un gran control sobre lo que ocurre durante el proceso de despliegue.

#### Hacer visibles los archivos ocultos

La mayoría de los sistemas operativos ocultan los archivos y carpetas que empiezan por un punto, como *.platform*. Cuando abras un explorador de archivos, no verás este tipo de archivos y carpetas por defecto. Pero como programador, necesitarás verlos. He aquí cómo ver los archivos ocultos, dependiendo de tu sistema operativo:

* En Windows, abre el Explorador de Windows y, a continuación, abre una carpeta como *Desktop*. Haz clic en la pestaña **Ver** y asegúrate de que están marcadas las opciones **Extensiones de nombre de archivo** y **Elementos ocultos**.
* En macOS, puedes pulsar ⌘-SHIFT-. (punto) en cualquier ventana del Finder para ver los archivos y carpetas ocultos.
* En sistemas Linux como Ubuntu, puedes pulsar CTRL-H en cualquier explorador de archivos para ver los archivos y carpetas ocultos. Para que esta configuración sea permanente, abre un explorador de archivos como Nautilus y haz clic en la pestaña de opciones (indicada por tres líneas). Selecciona la casilla **Mostrar** archivos ocultos.

#### El archivo de configuración .platform.app.yaml

El primer archivo de configuración es el más largo, porque controla el proceso general de despliegue. Lo mostraremos por partes; puedes introducirlo a mano en tu editor de texto o descargar una copia de los recursos en línea en [https://ehmatthes.github.io/pcc\_3e.](https://ehmatthes.github.io/pcc_3e)

Aquí tienes la primera parte de *.platform.app.yaml*, que debes guardar en el mismo directorio que *manage.py*:

**.platform.app.yaml**

❶ name: "ll\_project"  
type: "python:3.10"  
  
❷ relationships:  
 database: "db:postgresql"  
  
# The configuration of the app when it's exposed to the web.  
❸ web:  
 upstream:  
 socket\_family: unix  
 commands:  
❹ start: "gunicorn -w 4 -b unix:$SOCKET ll\_project.wsgi:application"  
❺ locations:  
 "/":  
 passthru: true  
 "/static":  
 root: "static"  
 expires: 1h  
 allow: true  
  
# The size of the persistent disk of the application (in MB).  
❻ disk: 512

Cuando guardes este archivo, asegúrate de incluir el punto al principio del nombre del archivo. Si omites el punto, Platform.sh no encontrará el archivo y tu proyecto no se desplegará.

No es necesario que lo entiendas todo en *.platform.app.yaml* en este momento; destacaré las partes más importantes de la configuración. El archivo comienza especificando el name del proyecto, que llamaremos 'll\_project' para ser coherentes con el nombre que utilizamos al iniciar el proyecto ❶. También tenemos que especificar la versión de Python que estamos utilizando (3.10 en el momento de escribir esto). Puedes encontrar una lista de las versiones compatibles en [https://docs.platform.sh/languages/python.html.](https://docs.platform.sh/languages/python.html)

A continuación hay una sección llamada relationships que define otros servicios que necesita el proyecto ❷. Aquí la única relación es con una base de datos Postgres. Después está la sección web ❸. La sección commands:start indica a Platform.sh qué proceso debe utilizar para atender las peticiones entrantes. Aquí estamos especificando que gunicorn gestionará las peticiones ❹. Este comando sustituye al comando python manage.py runserver que hemos estado utilizando localmente.

La sección locations indica a Platform.sh dónde debe enviar las solicitudes entrantes ❺. La mayoría de las peticiones deben pasarse a gunicorn; nuestros archivos *urls.py* dirán a gunicorn exactamente cómo manejar esas peticiones. Las solicitudes de archivos estáticos se gestionarán por separado y se actualizarán una vez cada hora. La última línea muestra que estamos solicitando 512 MB de espacio en disco en uno de los servidores de Platform.sh ❻.

El resto de [.platform.app.yaml](http://.platform.app.yaml) es como sigue:

--snip--  
disk: 512  
  
# Set a local read/write mount for logs.  
❶ mounts:  
 "logs":  
 source: local  
 source\_path: logs  
  
# The hooks executed at various points in the lifecycle of the application.  
❷ hooks:  
 build: |  
❸ pip install --upgrade pip  
 pip install -r requirements.txt  
 pip install -r requirements\_remote.txt  
  
 mkdir logs  
❹ python manage.py collectstatic  
 rm -rf logs  
❺ deploy: |  
 python manage.py migrate

La sección mounts ❶ nos permite definir directorios donde podemos leer y escribir datos mientras se ejecuta el proyecto. Esta sección define un directorio *logs/* para el proyecto desplegado.

La sección hooks ❷ define acciones que se llevan a cabo en varios momentos del proceso de despliegue. En la sección build, instalamos todos los paquetes necesarios para servir el proyecto en el entorno en vivo ❸. También ejecutamos collectstatic ❹, que reúne todos los archivos estáticos necesarios para el proyecto en un solo lugar para que puedan ser servidos de forma eficiente.

Por último, en la sección deploy ❺, especificamos que las migraciones se ejecuten cada vez que se despliegue el proyecto. En un proyecto sencillo, esto no tendrá ningún efecto cuando no haya habido cambios.

Los otros dos archivos de configuración son mucho más cortos; vamos a escribirlos ahora.

#### El archivo de configuración routes.yaml

Un *route* es la ruta que toma una petición cuando es procesada por el servidor. Cuando Platform.sh recibe una solicitud, necesita saber dónde enviarla.

Crea una nueva carpeta llamada *.platform*, en el mismo directorio que *manage.py*. Asegúrate de incluir el punto al principio del nombre. Dentro de esa carpeta, crea un archivo llamado *routes.yaml* e introduce lo siguiente:

**.platform/routes.yaml**

# Each route describes how an incoming URL will be processed by Platform.sh.  
  
"https://{default}/":  
 type: upstream  
 upstream: "ll\_project:http"  
  
"https://www.{default}/":  
 type: redirect  
 to: "https://{default}/"

Este archivo se asegura de que las peticiones como *https://project\_url.com* y *www.project\_url.com* se dirijan todas al mismo lugar.

#### El archivo de configuración services.yaml

Este último archivo de configuración especifica los servicios que nuestro proyecto necesita para funcionar. Guarda este archivo en el directorio *.platform/*, junto a *routes.yaml*:

**.platform/routes.yaml**

# Each service listed will be deployed in its own container as part of your  
# Platform.sh project.  
  
db:  
 type: postgresql:12  
 disk: 1024

Este archivo define un servicio, una base de datos Postgres.

[anterior](c20_20.html)[Subtema 21 de 37: (Ver todo)](c20.html)[siguiente](c20_22.html)