

PASOS A SEGUIR

Dejamos aquí una serie de pasos a seguir o una guía de cómo hemos levantado nuestro proyecto para que se pueda ejecutar sin problemas.

Tenéis todos los archivos en nuestro repositorio de GitHub mencionado a continuación. El **docker-compose** y el **Dockerfile** dentro de la carpeta **docker**. Serían lo referente al inicio de nuestra guía, por lo que recomendamos primero borrar el resto del contenido excepto la carpeta **docker** y seguir los pasos mencionados en la guía. También aconsejamos aumentar el Zoom a 110% para ver los contenidos mejor.

Organización :

- **Daniel** : **Scrum master**, el encargado de dirigir el equipo de proyecto. Después de asignar las tareas que van a llevar los colaboradores (incluido él) hemos quedado en lo siguiente: Su parte es la de usabilidad y accesibilidad de la aplicación web. Creación del repositorio remoto y actualización de contenidos.
- **Isla** : **Colaboradora del proyecto**, su parte ha sido el diseño de vistas. Colaboradora del repositorio remoto y actualización de cambios.
- **Rafa** : **Colaborador del proyecto**, gestión de contenedores en Docker y acceso a datos. Colaborador del repositorio remoto y actualización de cambios.

Sprints realizados: 1

Lenguajes usados: JS, PHP, SQL

Herramientas usadas :

- **Docker** - Levantar el servidor web y la base de datos con **Docker**.
- **GitHub** - Herramienta GitHub como medio colaborativo en **VSC**.



- Repositorio remoto:

<https://github.com/danielfgc/hitogrupalwordpress.git>

- **Visual Studio Code** - Para realizar el código del proyecto.

Paso a Paso :

En primer lugar vamos a montar el servidor web con **Apache+Postgres**. Para ello usaremos dos de los tres archivos que se encuentran dentro de la carpeta **docker**. (el otro con extensión **.SQL** hace referencia al back up de la base de datos)

	docker-compose	02/04/2022 19:42	Archivo de origen ...	1 KB
	Dockerfile	02/04/2022 17:43	Archivo	1 KB

Veamos la estructura :

Docker-compose :

```
docker-compose.yml
1  version: "3.8"
2
3  services:
4    postgres:
5      image: postgres
6      container_name: bdposgres
7      restart: always
8      ports:
9        - "5432:5432"
10     environment:
11       - DATABASE_HOST=localhost
12       - POSTGRES_USER=admin
13       - POSTGRES_PASSWORD=admin
14     volumes:
15       - ./database:/var/lib/postgresql/data
16     pgadmin:
17       image: dpage/pgadmin4
18       environment:
19         PGADMIN_DEFAULT_EMAIL: "admin@gmail.com"
20         PGADMIN_DEFAULT_PASSWORD: "admin"
21       volumes:
22         - ./pgadmin:/var/lib/pgadmin
23       ports:
24         - "8080:80"
25       depends_on:
26         - postgres
27     apache:
28       image: php:latest
29       build: .
30       container_name: PHP
31       ports:
32         - "80:80"
33       volumes:
34         - ./webs:/var/www/html
35
36     links:
37       - postgres
38
```

En rojo están las variables de entorno que podremos cambiar a nuestro gusto las cuales hacen referencia a la database y a la interfaz gráfica que vamos a usar. En nuestro caso hemos usado esas, pero se pueden cambiar a gusto de la persona que lo vaya a consumir.

Nota: Las dos primeras variables son las de la database mientras que las dos segundas hacen referencia a la interfaz gráfica como veremos más adelante.

Dockerfile :

El **Dockerfile** es un archivo muy importante con el que se pueden configurar muchas cosas. En él le damos la instrucciones que va a seguir el docker-compose así como las instrucciones para crear la imagen de nuestra App para subirla a docker hub y que todo el mundo la pueda consumir con las instrucciones que nosotros le hayamos dado.

¡ IMPORTANTE ! Este archivo no tiene extensión y se escribe con “D”.

veamos la estructura de nuestro Dockerfile:

```
Dockerfile > ...
1  FROM php:7.4-apache
2
3  RUN apt-get update && apt-get install -y libpq-dev && docker-php-ext-install pdo pdo_pgsql
4
5  COPY webs/ /var/www/html
6
7
8  EXPOSE 80
```

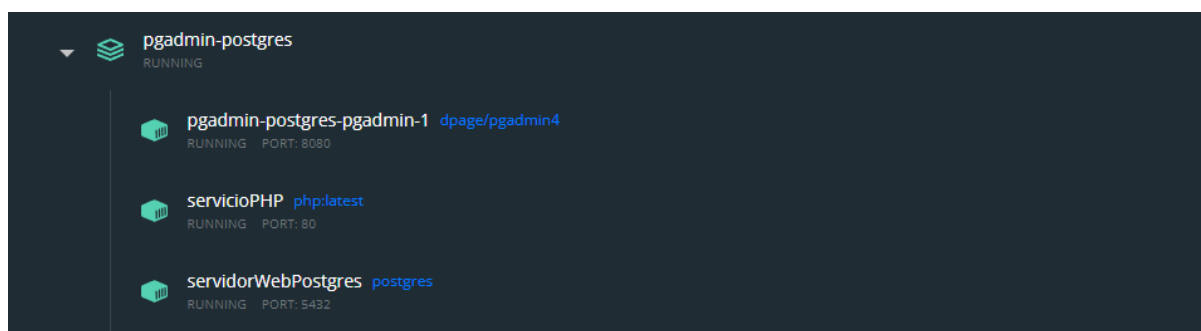
La **línea 3** es muy importante ya que va a ser la que se encargue de instalar las extensiones necesarias de nuestro **php.ini** para que funcione correctamente el acceso a datos (instalar la extensión pdo_pgsql).

Un vez analizada la estructura vamos a comenzar a levantar el proyecto, para ello haremos los siguientes pasos:

- Abrimos la carpeta docker en donde se encuentra nuestro docker-compose con VSC (Visual Studio Code). pulsamos “ F1 ” y seleccionaremos “nuevo terminal” (así estaremos en la ruta de nuestro archivo a ejecutar). Simplemente ejecutaremos el siguiente comando :

```
C:\Users\Rafa_\OneDrive\Desktop\proyectoGruaplPg>docker-compose up
```

- Cuando ya lo hayamos ejecutado se nos empezará a montar el servidor con las instrucciones del docker-compose y el Dockerfile. Una vez terminado el proceso, debemos dejar el terminal abierto ya que es el encargado de mantenernos nuestro servicio activo. En caso contrario, si lo hemos cerrado, podemos ir a docker desktop y ejecutarlo desde ahí:



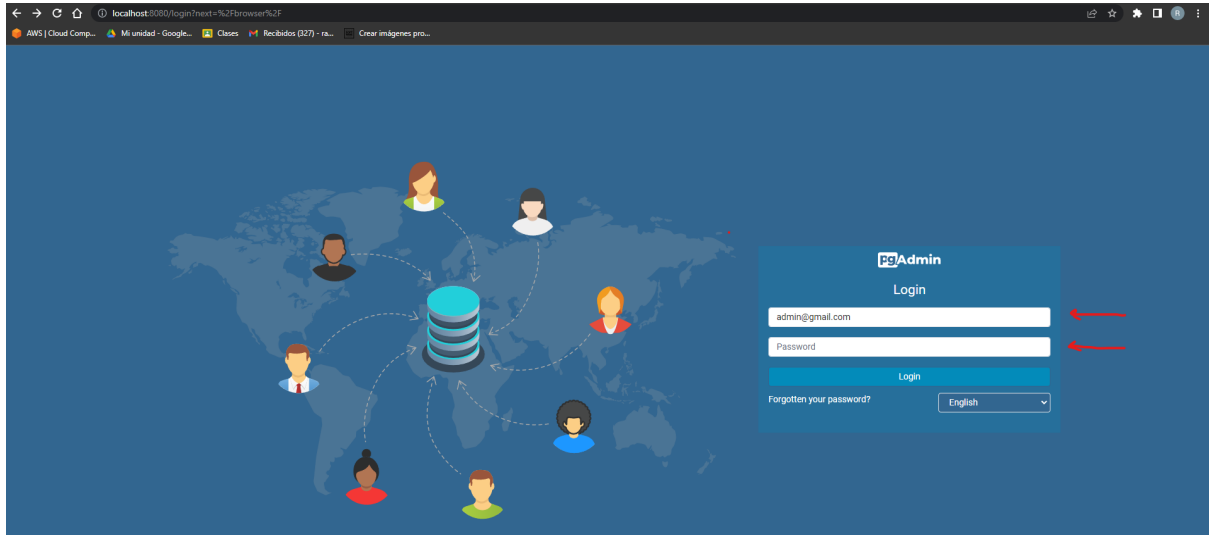
Como vemos tenemos los tres contenedores que nos dan servicio levantados. El nombre del compose hace referencia a la carpeta donde se ejecuta (en este caso tengo otro servidor de prueba ejecutándose, no el del ejercicio usado para esta guía) Aquí podemos asegurarnos de que todos los contenedores estén funcionando correctamente y en caso contrario ver cuál no, para saber dónde está el problema.

Configurar la database :

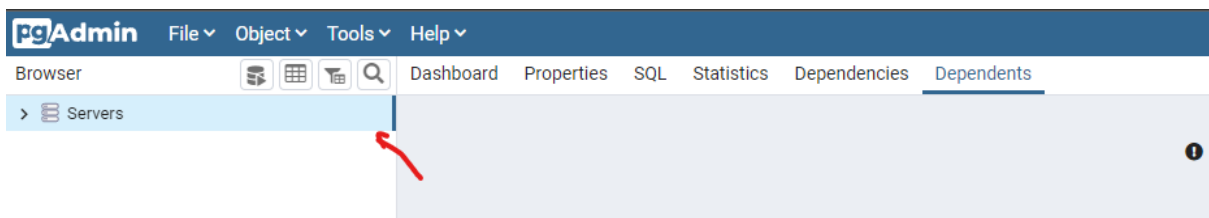
Una vez tengamos todo funcionando correctamente, vamos a ir al puerto **8080**. Para ello iremos a nuestro navegador favorito y lo haremos de la siguiente manera :

```
localhost:8080
```

Se nos tiene que abrir la interfaz gráfica de postgres que veremos a continuación en donde meteremos los datos anteriormente mencionados:



Introducimos los datos y ya tendríamos nuestra database con la interfaz gráfica funcionando, ahora vamos a configurar la conexión de postgres.



Para ello, haremos click derecho en “ **Servers** ” > “ **Register** ” > “ **Servers..** ” y nos aparecerá la siguiente ventana:

Register - Server

General | Connection | SSL | SSH Tunnel | Advanced

Name ❗

Server group ☰ Servers ▼

Background ✕

Foreground ✕

Connect now? 🔵

Shared? 🔴

Comments

❗ 'Name' cannot be empty. ✕


ℹ ? ✕ Close ↺ Reset 💾 Save

Primero indicamos el nombre de la conexión, en nuestro caso será “**conexiondocker**”

Después nos vamos a la pestaña “**Connection**” para configurar esa conexión:

Register - Server

General Connection SSL SSH Tunnel Advanced

Host name/address 

Port 5432

Maintenance database postgres

Username admin



Kerberos authentication? ☐



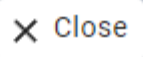


Password

Save password? ☐

Role

Service

 Either Host name, Address or Service must be specified. 

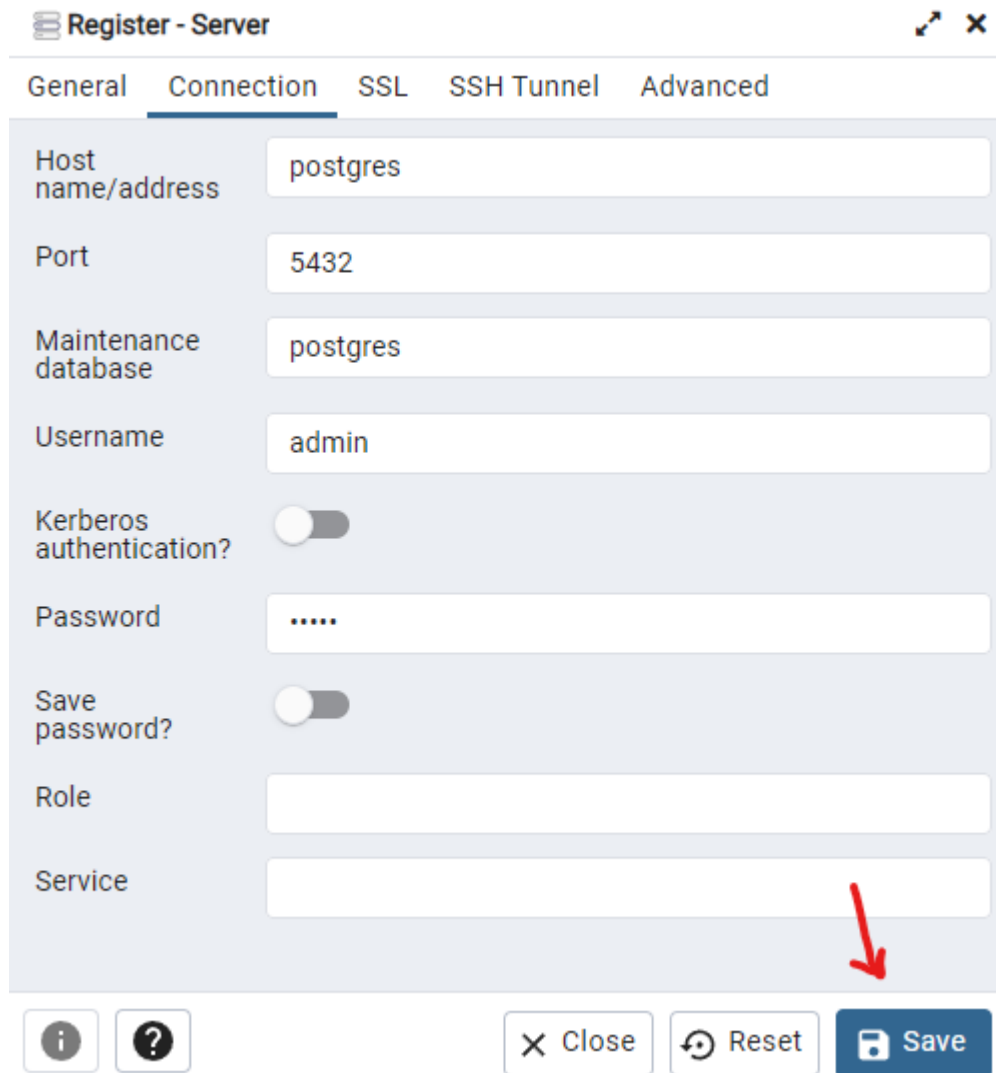
   Close  Reset  Save

1. Indicamos el host que vamos a usar. ! **IMPORTANTE** ! Debemos poner el host del servicio de nuestra database que en este caso se llama postgres , ya que este es el que llama al localhost :

```
4 postgres:
5   image: postgres
6   container_name: bdpostgres
7   restart: always
8   ports:
9     - "5432:5432"
10  environment:
11    - DATABASE_HOST=localhost
12    - POSTGRES_USER=admin
13    - POSTGRES_PASSWORD=admin
```

Más adelante haremos lo mismo para la conexión de nuestro proyecto.

2. Nombre del usuario que hayamos especificado en el docker-compose (después de localhost). En nuestro caso es “ **admin** ”.
3. Contraseña que hayamos especificado en el docker-compose (después de user). En nuestro caso es “ **admin** ”.



Register - Server

General **Connection** SSL SSH Tunnel Advanced

Host name/address: postgres

Port: 5432

Maintenance database: postgres

Username: admin

Kerberos authentication? ☐

Password:


Save password? ☐

Role:

Service:

Guardamos y listo, ya deberíamos tener la conexión correctamente hecha con la que podríamos usar postgres para lo que queramos. Ahora vamos a configurar la DB de nuestro ejercicio con el **backup.sql** que hemos proporcionado.

Para ello, vamos a crear una nueva base de datos en postgres . Una vez la tengamos pinchamos con click derecho sobre ella y se nos desplegará una ventana, pinchamos en “ **restore..** ” y se nos abrirá lo siguiente :



Restore (Database: hitogrupalPG)

General Data/Objects Options

Format Custom or tar

Filename ! 📁

Number of jobs

Role name Select an item...

! Please provide a filename. ×

i ? ✕ Close ↺ Reset ↗ Restore

Pinchamos en la carpeta y se nos abrirá otra ventana, en donde deberá estar nuestro **backup.sql** (o base de datos que queramos con extensión .SQL). Como vemos no tenemos nada ya que la información que hay almacenada hace referencia al contenedor, pero sí que podemos acceder a través de nuestro ordenador ya que se lo hemos especificado en el docker-compose con “volumes:” (datos persistentes). Entonces para que nuestro archivo se vea, vamos a ir a la carpeta de nuestro proyecto :

Nombre	Fecha de modificación	Tipo	Tamaño
database	03/04/2022 10:00	Carpeta de archivos	
pgadmin	03/04/2022 10:00	Carpeta de archivos	
webs	02/04/2022 19:51	Carpeta de archivos	
docker-compose	02/04/2022 19:42	Archivo de origen ...	1 KB
Dockerfile	03/04/2022 10:16	Archivo	1 KB

Deberíamos tener la siguiente estructura una vez montado el docker-compose, después, pinchamos en “ **pgadmin** “ En esta carpeta podemos ingresar las bases de datos así como los archivos SQL con las sentencias que hayamos guardado.

Nombre	Fecha de modificación	Tipo	Tamaño
sessions	03/04/2022 10:00	Carpeta de archivos	
storage	02/04/2022 19:46	Carpeta de archivos	
pgadmin4	03/04/2022 10:00	Data Base File	176 KB

Después en “ **storage** ”

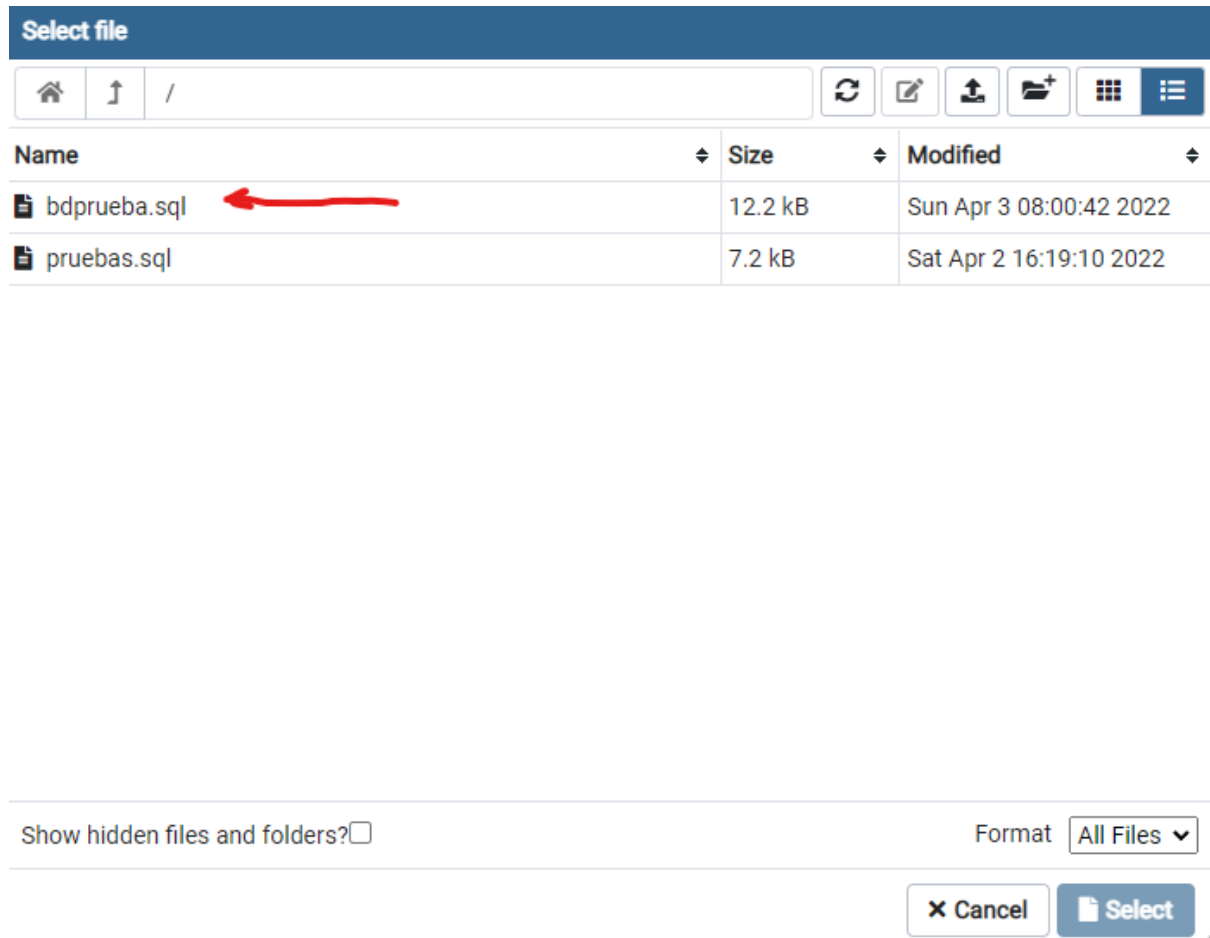
rafael.martinezl_campusfp.es	03/04/2022 10:02	Carpeta de archivos
------------------------------	------------------	---------------------

O el nombre que hayamos usado para acceder a la interfaz gráfica

Nombre	Fecha de modificación	Tipo	Tamaño
pruebas	02/04/2022 18:19	SQL Text File	8 KB
pruebas2	03/04/2022 9:57	Archivo	13 KB
SQLpruebas	02/04/2022 21:16	Archivo	1 KB


Y es aquí donde pegaremos el archivo **backup.sql**. Como veis en el ejemplo, “ **pruebas** “ si que tiene la extensión **.SQL** así que lo leerá. En el caso de los otros dos archivos al no tener la extensión, postgres no los reconocerá, por lo que es importante especificar la extensión a la hora de guardar la DB.

Ahora volvamos al restore anterior en la interfaz gráfica, pinchamos nuevamente en el icono de la carpeta y deberíamos tener nuestro **backup.sql**:



En nuestro caso se llama “ **bdpruebas.sql** ” ya que seguimos haciendo pruebas para que todo funcione correctamente. Pero esto valdría con la base de datos que queramos. La seleccionamos y le damos a “ **Restore** ” y ya tendríamos la base de datos configurada para usar nuestra app.

Ahora volvamos a nuestro directorio del proyecto. Como vemos, tenemos una carpeta llamada “ **webs** ”:

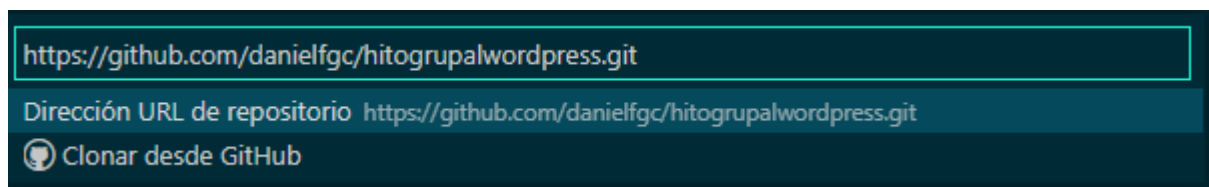
Nombre	Fecha de modificación	Tipo	Tamaño
database	03/04/2022 10:00	Carpeta de archivos	
pgadmin	03/04/2022 10:00	Carpeta de archivos	
webs 	02/04/2022 19:51	Carpeta de archivos	
docker-compose	02/04/2022 19:42	Archivo de origen ...	1 KB
Dockerfile	03/04/2022 10:16	Archivo	1 KB

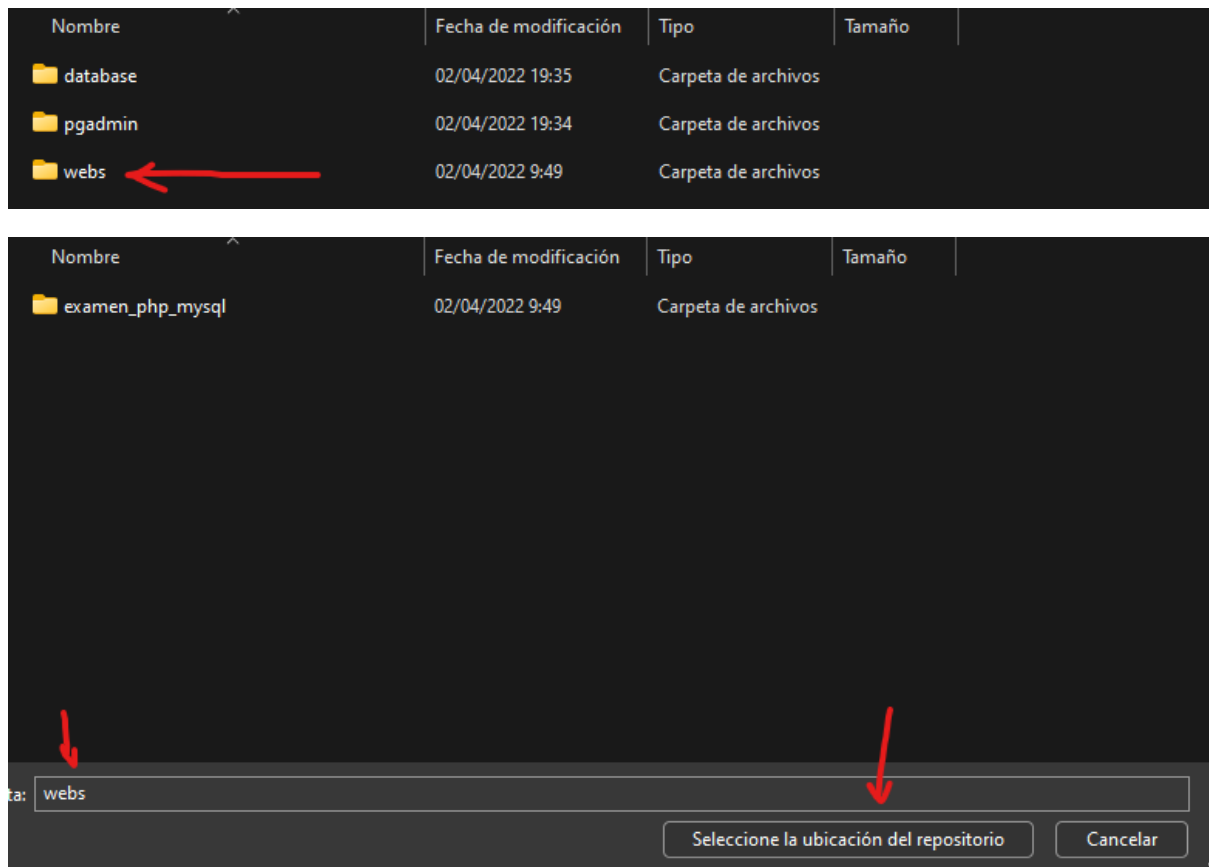
Pues bien, esta carpeta es nuestro **HTDOCS de Apache**. Podéis meter cualquier proyecto con PHP y veréis como funciona. Si queréis usar nuestro proyecto podéis copiar el proyecto en caso de que no lo hayáis borrado y meterlo en la carpeta que se indica más adelante (**No recomendable**) o si estáis siguiendo la guía debemos abrir el **VSC** y pulsamos “ **F1** ”, se nos abrirá un recuadro como el de antes para el terminal pero en este caso pondremos “ **clonar** ” y veremos como nos marca un comando “ **Git:clone** ” y ahí le especificamos la URL de nuestro proyecto:

<https://github.com/danielfgc/hitogrupalwordpress.git>

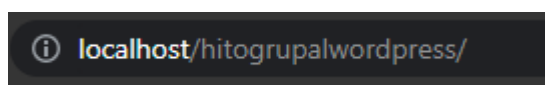
(En este caso pasamos el proyecto principal ya que los demás estamos como colaboradores, para evitar posibles fallos).

Nos pedirá la ubicación de nuestro proyecto, entonces aquí ya iremos a la carpeta del proyecto, después a nuestra carpeta “ **webs** ” y lo meteremos ahí.

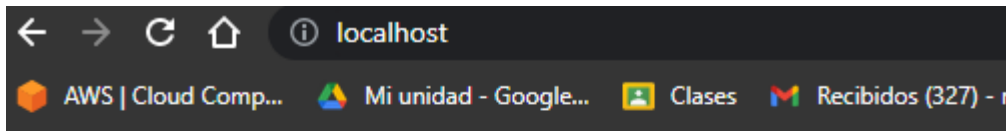




Comprobamos que estemos dentro de la carpeta y le damos a “ **seleccionar la ubicación del repositorio** ”. (En caso de que hayais usado nuestro proyecto podréis borrar la carpeta **docker** que acabáis de bajar nuevamente con el proyecto, para que no os moleste) Una vez tengamos nuestro proyecto en la carpeta accederemos a él de la siguiente manera :



Nota: si ponemos solamente localhost sin especificar la ruta del proyecto que queremos abrir nos saldrá lo siguiente ya que no tenemos permisos, no tenemos nuestro **phpMyAdmin**



Forbidden

You don't have permission to access this resource.

Apache/2.4.52 (Debian) Server at localhost Port 80

Así que no os olvidéis de poner la ruta para que os aparezca.

Antes de terminar esta guía, nos gustaría hacer un último paso para que todo el mundo tenga claro el hacer la conexión del proyecto con la DB o con cualquier otro proyecto que queramos. Como dijimos anteriormente no estamos usando localhost, sino el contenedor de postgres por ello debemos especificar lo siguiente en nuestra configuración del proyecto.

```
connection.php > ...
1  <?php
2
3  class Connect
4  {
5      static function getConnection()
6      {
7          $conexion = new PDO("pgsql:host=postgres;port=5432;dbname=hitogrupal", "admin", "admin");
8          return $conexion;
9      }
10 }
11
```

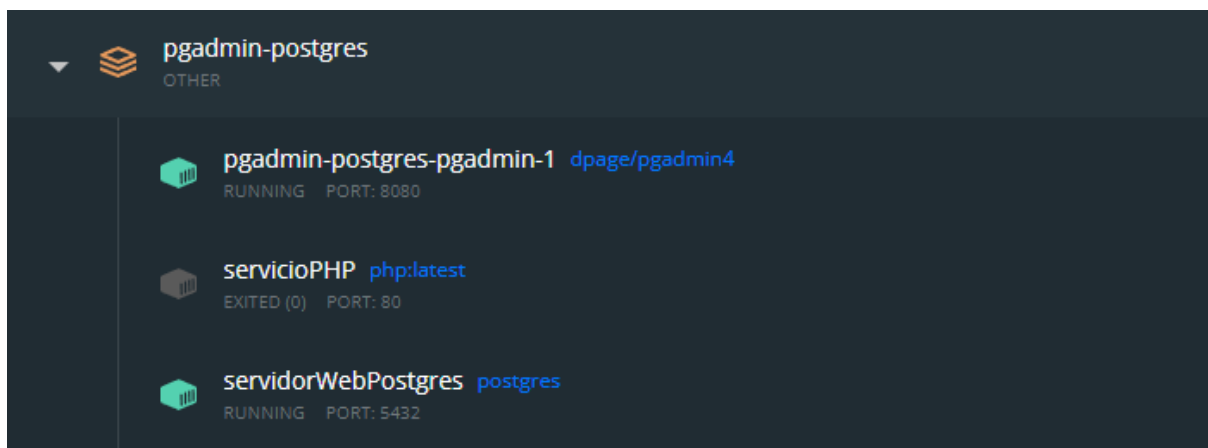
1 2 3 4 5

1. Aquí al igual que en la conexión de la DB usaremos “ postgres ”.
2. El puerto que hemos habilitado en el contenedor de postgres mapeado a nuestro ordenador local.
3. El nombre de la base de datos a la que queremos conectar.
4. El nombre de usuario de la base de datos (el que usamos para acceder a la conexión).
5. La contraseña de usuario de la base de datos (el que usamos para acceder a la conexión).

Y listo, ya tendríamos nuestro proyecto listo para trabajar con postgres.

Posibles errores :

- Si vemos que a la hora de la conexión con la BD en nuestro proyecto nos da error por temas de Drivers eso es porque la extensión especificada en el [Dockerfile](#) no se ha instalado correctamente. Para ello deberemos borrar la imagen que hayamos creado de PHP para el servidor y volver a hacer el docker-compose up, se debería volver a descargar la imagen con la extensión.
- Si vemos que algún contenedor nos da fallo como el que veremos a continuación es posible que debamos comprobar que todos los puertos están libres (por ejemplo no tener Xampp levantado) sino esto nos impedirá levantar los contenedores correctamente.



- Si vemos que a la hora de hacer la conexión nos da un error que nos indica que postgres necesita una versión diferente o más antigua, deberemos ir al [Dockerfile](#) y en donde pone “ **FROM** “ deberemos usar una versión de apache que coincida con la versión de postgres usada. **No recomiendo tocar mucho los archivos si queréis que todo funcione correctamente. En caso contrario, sois libres de investigar y de comeros la cabeza.**
- En caso de que hayamos parado todos los servicios , todos los puertos están libres y no podamos levantar un contenedor es posible que si

hemos hecho el docker-compose up con algún puerto ocupado no se nos haya instalado correctamente, **mi recomendación es eliminar el proyecto desde el docker desktop y volver a ejecutar el docker-compose. En caso de que siga sin ir, borrar el contenedor junto con las imágenes(ordenadamente así) y volver a ejecutar el docker-compose up. Con esto no deberías tener problemas.**

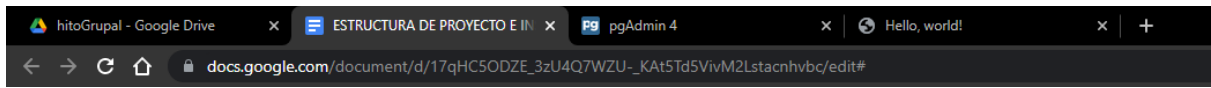
- Si se ejecuta correctamente todo pero aún así no se insertan los datos en la DB, es posible que alguna de las sentencias usadas en el código estén mal o que en la DB los campos especificados no sean los mismos que usamos en la sentencia de nuestra aplicación web.
- En caso de que montemos más servidores iguales y estéis probando, es posible que os de un error al hacer el docker-compose por el nombre del contenedor (ya que existe uno con el mismo nombre). Para ello cambiaremos el nombre del contenedor en el **docker-compose**, borramos el proyecto creado recientemente (en el docker desktop) y ejecutaremos nuevamente:

```
postgres:
  image: postgres
  container_name: bdpostgres ←
  restart: always
  ports:
    - "5432:5432"
```

```
apache:
  image: php:latest
  build: .
  container_name: PHP ←
  ports:
```

Datos a cambiar.

Como vemos es una genial idea para no llenar nuestro ordenador de programas y que sea algo descontrolado, además de poder trabajar cómodamente desde nuestro navegador.



¡Esperamos que os haya servido!

