

Para realizar esta práctica he creado un entorno web completo utilizando Docker y Docker Compose. El objetivo era tener funcionando un servidor PHP con Apache, una base de datos MySQL y la herramienta phpMyAdmin para poder gestionar la base de datos desde el navegador.

Lo primero que hice fue preparar la estructura de carpetas. Dentro de mi proyecto tengo una carpeta principal llamada “php” y dentro de ella creé otra carpeta llamada “docker”, donde coloqué todos los archivos relacionados con Docker. También dentro de “php” tengo la carpeta “app”, que contiene mis archivos PHP. La estructura quedó así:

```
C:\Users\pablg\Desktop\DAW\Entorno de Servidor\php\  
├── app\  
└── docker\  
    ├── web\  
    │   └── Dockerfile  
    └── docker-compose.yaml
```

Dentro de la carpeta “docker” creé el archivo docker-compose.yaml, que es el que define los tres servicios que forman el sistema: el servicio web, el servicio mysql y el servicio phpmyadmin. En este archivo configuré los puertos, los volúmenes y las variables de entorno necesarias. También configuré que el servicio web se construya desde un Dockerfile que está en la carpeta “web”.

Después, en la carpeta “web”, creé el archivo Dockerfile. Este archivo se basa en la imagen oficial php:8.4-apache. Añadí los comandos para instalar la extensión Xdebug, que se utiliza para depurar código PHP, y además modifiqué la configuración de Apache para permitir el listado de ficheros con la opción “Options +Indexes”. Con esto, al entrar a una carpeta desde el navegador, puedo ver los archivos que contiene, algo útil para comprobar que todo funciona.

En la carpeta “app” tengo mi archivo index.php, con un simple phpinfo() para comprobar que PHP se ejecuta correctamente.

Una vez configurado todo, abrí la terminal de PhpStorm (también se podría hacer desde PowerShell o CMD) y me moví hasta la carpeta donde está el archivo docker-compose.yaml, que en mi caso es:





```
cd "C:\Users\pablg\Desktop\DAW\Entorno de Servidor\php\docker"
```

Desde ahí ejecuté los siguientes comandos:

- Para construir la imagen del servicio web por primera vez:  
docker compose build web
- Para levantar todos los servicios en segundo plano:  
docker compose up -d

Cuando se completó la construcción de las imágenes y los contenedores se iniciaron correctamente, abrí el navegador para comprobar que todo funcionaba.

## Index of /

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Juego/</a>	2025-10-18 19:07	-	
 <a href="#">_index.php</a>	2025-10-17 07:18	21	
 <a href="#">clicks/</a>	2025-10-17 06:49	-	
 <a href="#">formularios/</a>	2025-10-09 07:15	-	

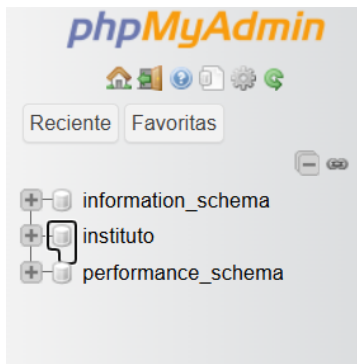
*Apache/2.4.65 (Debian) Server at localhost Port 8100*

Después abrí phpMyAdmin en la dirección <http://localhost:8101> y comprobé que se podía acceder correctamente a la interfaz.



The image shows the phpMyAdmin login page. At the top, there is a logo with a sailboat and the text 'phpMyAdmin' and 'Bienvenido a phpMyAdmin'. Below this, there is a section for language selection with a dropdown menu currently set to 'Español - Spanish'. Underneath, there is a login form with fields for 'Servidor:', 'Usuario:', and 'Contraseña:'. At the bottom right of the form is a button labeled 'Iniciar sesión'.

Dentro de phpMyAdmin verifiqué que la base de datos “instituto” se había creado correctamente gracias a las variables definidas en el docker-compose.yaml.



También comprobé que el servidor web mostraba el listado de archivos gracias a la configuración “Options +Indexes”.

En cuanto a los comandos de gestión, utilicé los siguientes de manera habitual:

- Para parar los contenedores sin borrar los datos: `docker compose down`
- Para ver los contenedores activos: `docker ps`
- Para acceder dentro del contenedor web y poder explorar archivos: `docker compose exec web bash`
- Para ver los logs y comprobar posibles errores: `docker compose logs -f`

Probé a apagar y volver a levantar el sistema con “`docker compose down`” y luego “`docker compose up -d`”, y comprobé que la base de datos seguía existiendo, lo cual demuestra que el volumen de MySQL mantiene los datos de forma persistente.

Respecto a las preguntas de reflexión, entendí varios conceptos importantes. En el servicio MySQL, si no se define la variable `MYSQL_ROOT_PASSWORD` el contenedor no puede iniciarse correctamente, ya que esa variable es obligatoria. También comprendí que si no se define un volumen, los datos se guardan dentro del contenedor y se pierden al eliminarlo.

Sobre los volúmenes, aprendí que los volúmenes nombrados los gestiona Docker y son ideales para mantener datos persistentes sin depender de rutas locales, mientras que los volúmenes bind se enlazan a carpetas del host y son más útiles para desarrollo, aunque pueden dar problemas de permisos. En mi caso utilicé un volumen nombrado para el servicio MySQL.

En cuanto a la comunicación entre servicios, descubrí que no hace falta exponer el puerto 3306 del contenedor MySQL porque phpMyAdmin se conecta a él usando la red interna de Docker, gracias a la variable `PMA_HOST` que apunta al servicio “mysql”.

En el servicio web entendí que es necesario construir la imagen desde un Dockerfile en

lugar de usar directamente la imagen `php:8.4-apache` porque quería personalizarla instalando Xdebug y modificando la configuración de Apache. También vi que si no monto el volumen `../app:/var/www/html`, no se verían mis archivos locales dentro del contenedor.

Sobre Xdebug, aprendí que es muy útil para depurar código en desarrollo, pero no se debe dejar activado en producción porque consume recursos y puede mostrar información sensible. Lo mismo ocurre con la opción “Options +Indexes”, que es útil para desarrollo, pero insegura en un entorno de producción porque muestra los archivos del servidor.

Otra cosa que comprendí es la diferencia entre los comandos `docker exec` y `docker compose exec`. El primero requiere el nombre exacto del contenedor, mientras que el segundo permite usar el nombre del servicio definido en el archivo de composición, lo cual es más práctico. Además, vi que si se usa “`docker compose down -v`”, también se borran los volúmenes y por tanto los datos de la base de datos.

En cuanto a los logs, vi que puedo verlos directamente con “`docker logs web`” porque Docker redirige la salida estándar de Apache y PHP, lo que hace que no sea necesario entrar al contenedor para revisar los archivos de log.

Finalmente, entendí que el comando “`docker run`” sirve para lanzar un contenedor individual, mientras que “`docker compose up`” levanta un entorno completo definido con varios servicios, redes y volúmenes. En mi caso, este entorno está claramente orientado a desarrollo porque utilizo volúmenes que reflejan los cambios de mi código en tiempo real y tengo activadas opciones que no serían seguras en producción.