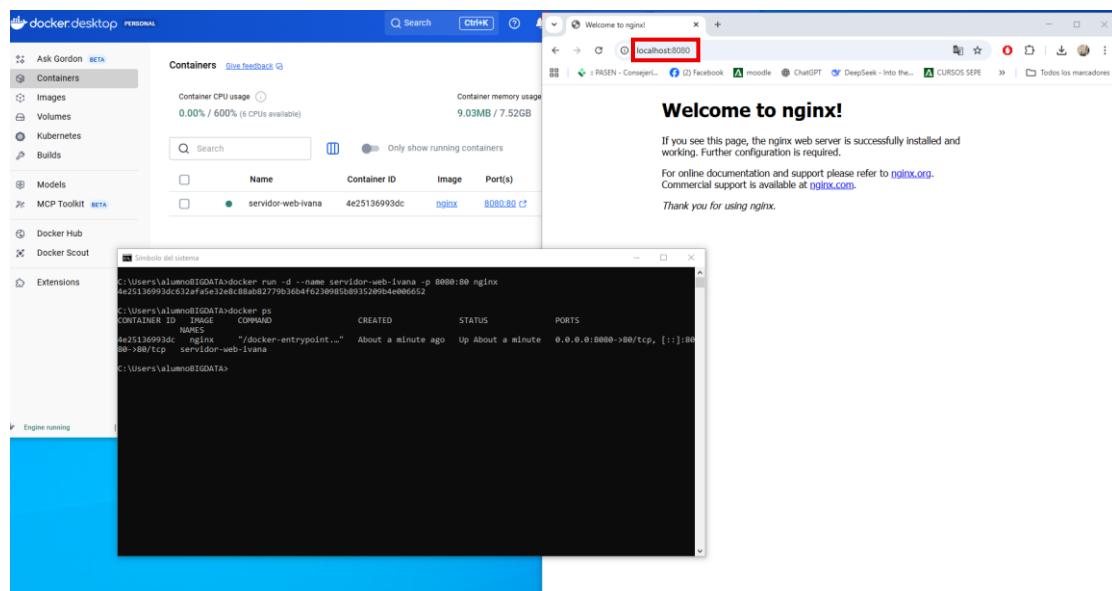


- 1- Crear un contenedor de Docker que tenga la capacidad de ejecutar un servidor web, utilizando una de las imágenes existentes en Docker Hub. Para ello, se debe acceder a Docker Hub, (repositorio en línea de imágenes para contenedores Docker), y buscar una imagen de servidor web que se aadecue a las necesidades del proyecto. Luego, se debe utilizar el comando docker run junto con el nombre de la imagen seleccionada para crear y ejecutar el contenedor. Por ejemplo, se podría utilizar la imagen "nginx" para crear el contenedor de servidor web.

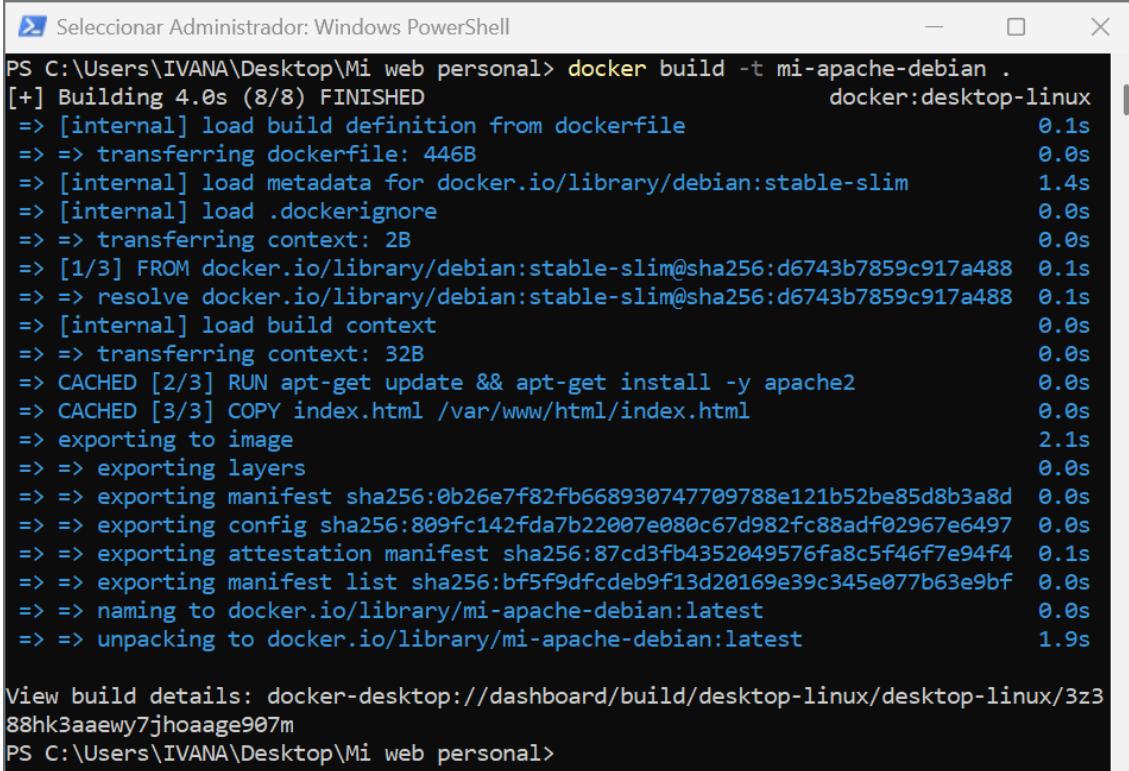
```
docker run -d --name servidor-web-ivana -p 8080:80 nginx
```



- 2- Crear una imagen de docker mediante un fichero Dockerfile que tenga las siguientes características:

- 3- Usar una plantilla de Debian.
- 4- Instalar Apache2. (`apt update && apt install apache2 -y`)
- 5- Establecer el directorio por defecto en `/var/www/html`
- 6- Mover una página web que os descarguéis de Internet a dicho directorio.
- 7- El Puerto 80 debe de estar abierto.
- 8- Añadir al final del fichero la siguiente instrucción para que apache2 funcione en segundo plano:
`CMD ["/usr/sbin/apache2ctl","-DFOREGROUND"]`

Lo haremos con el comando: **docker build -t mi-apache-debian**.



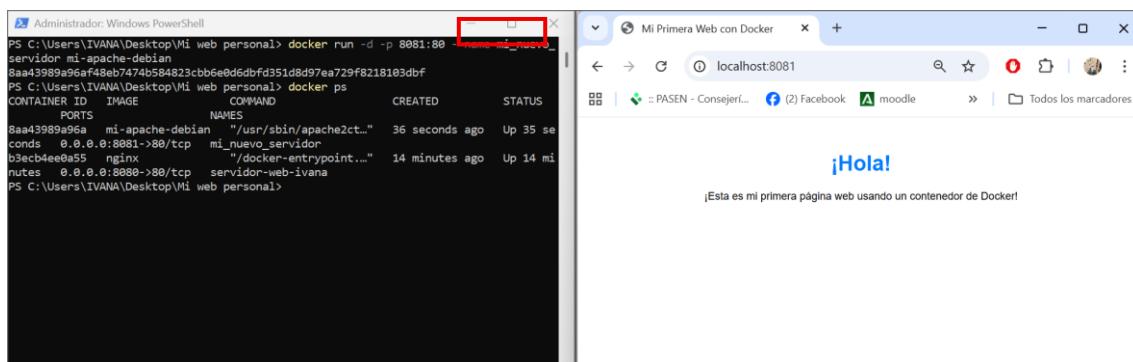
```
PS C:\Users\IVANA\Desktop\Mi web personal> docker build -t mi-apache-debian .
[+] Building 4.0s (8/8) FINISHED
          docker:desktop-linux
=> [internal] load build definition from dockerfile                               0.1s
=> => transferring dockerfile: 446B                                            0.0s
=> [internal] load metadata for docker.io/library/debian:stable-slim           1.4s
=> [internal] load .dockerrcignore                                              0.0s
=> => transferring context: 2B                                                0.0s
=> [1/3] FROM docker.io/library/debian:stable-slim@sha256:d6743b7859c917a488  0.1s
=> => resolve docker.io/library/debian:stable-slim@sha256:d6743b7859c917a488  0.1s
=> [internal] load build context                                               0.0s
=> => transferring context: 32B                                              0.0s
=> CACHED [2/3] RUN apt-get update && apt-get install -y apache2            0.0s
=> CACHED [3/3] COPY index.html /var/www/html/index.html                         0.0s
=> exporting to image                                                       2.1s
=> => exporting layers                                                 0.0s
=> => exporting manifest sha256:0b26e7f82fb668930747709788e121b52be85d8b3a8d  0.0s
=> => exporting config sha256:809fc142fda7b22007e080c67d982fc88adf02967e6497  0.0s
=> => exporting attestation manifest sha256:87cd3fb4352049576fa8c5f46f7e94f4  0.1s
=> => exporting manifest list sha256:bf5f9dfcdeb9f13d20169e39c345e077b63e9bf  0.0s
=> => naming to docker.io/library/mi-apache-debian:latest                      0.0s
=> => unpacking to docker.io/library/mi-apache-debian:latest                     1.9s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/3z388hk3aaewy7jhoaage907m
PS C:\Users\IVANA\Desktop\Mi web personal>
```

- 3- Crear un contenedor a partir del fichero anterior y arrancarlo. Comprobar que la web que has descargado se visualiza en localhost. ¿Qué harías para que la página web pudiera ser visible en localhost:8081?

Simplemente a la hora de crear el contenedor le especifico el puerto 8081 en localhost:

docker run -d -p 8081:80 --name mi_nuevo_servidor mi-apache-debian



4. ¿Le has dado algún nombre a la imagen? ¿Qué nombre tiene la imagen que has creado? ¿Qué nombre tiene el contenedor?

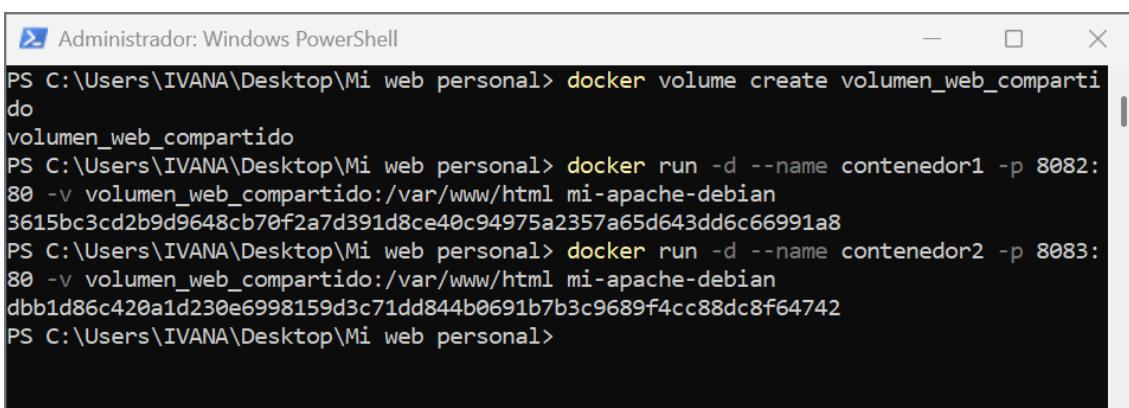
A la imagen le he dado el nombre de “mi_apache_debian” y al contenedor le he dado el de “mi-nuevo-servidor”

5. Crear dos nuevos contenedores a partir de la imagen anterior con las siguientes características:
- Un volumen que será compartido entre los dos contenedores y que conectará dentro del contenedor con la ruta /var/www/html
 - Los puertos de acceso a los contenedores deben ser 8082 y 8083

docker volume create volumen_web_compartido

**docker run -d --name contenedor1 -p 8082:80 -v
volumen_web_compartido:/var/www/html mi-apache-debian**

**docker run -d --name contenedor2 -p 8083:80 -v
volumen_web_compartido:/var/www/html mi-apache-debian**

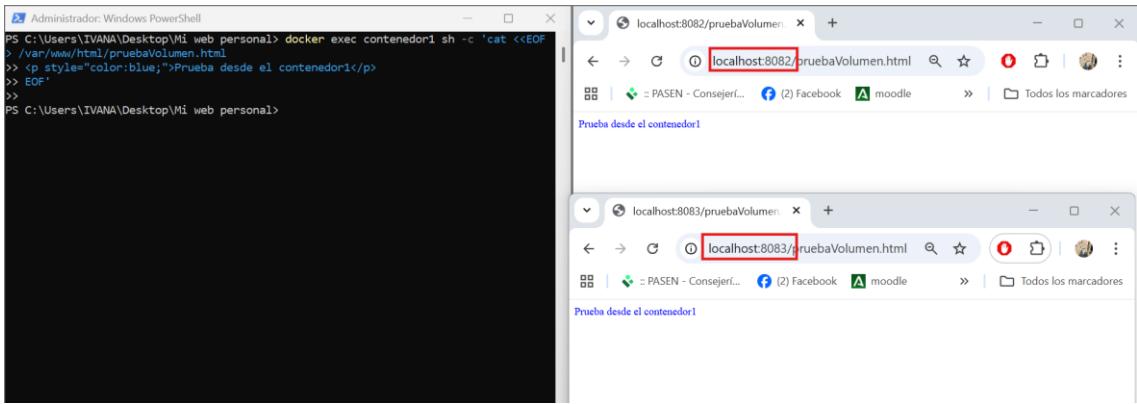


```
PS C:\Users\IVANA\Desktop\Mi web personal> docker volume create volumen_web_compartido
PS C:\Users\IVANA\Desktop\Mi web personal> docker run -d --name contenedor1 -p 8082:80 -v volumen_web_compartido:/var/www/html mi-apache-debian
3615bc3cd2b9d9648cb70f2a7d391d8ce40c94975a2357a65d643dd6c66991a8
PS C:\Users\IVANA\Desktop\Mi web personal> docker run -d --name contenedor2 -p 8083:80 -v volumen_web_compartido:/var/www/html mi-apache-debian
dbb1d86c420a1d230e6998159d3c71dd844b0691b7b3c9689f4cc88dc8f64742
PS C:\Users\IVANA\Desktop\Mi web personal>
```

Conéctate al primero de los contenedores y crea un fichero pruebaVolumen.html escribe dentro lo siguiente: <p>Prueba desde el contenedor 1</p> .

Accede a la ruta <http://localhost:8082/pruebaVolumen.html>.

```
docker exec contenedor1 sh -c 'cat <<EOF >
/var/www/html/pruebaVolumen.html
<p style="color:blue;">Prueba desde el contenedor1</p>
EOF'
```



¿Qué aparece en el otro contenedor? ¿Existe en la siguiente ruta <http://localhost:8081/pruebaVolumen.html>?

Ambas rutas (contenedor1 y contenedor2) están leyendo los datos del mismo volumen: volumen-compartido, el mensaje que he creado en el contenedor1 → Prueba desde el contenedor 1.

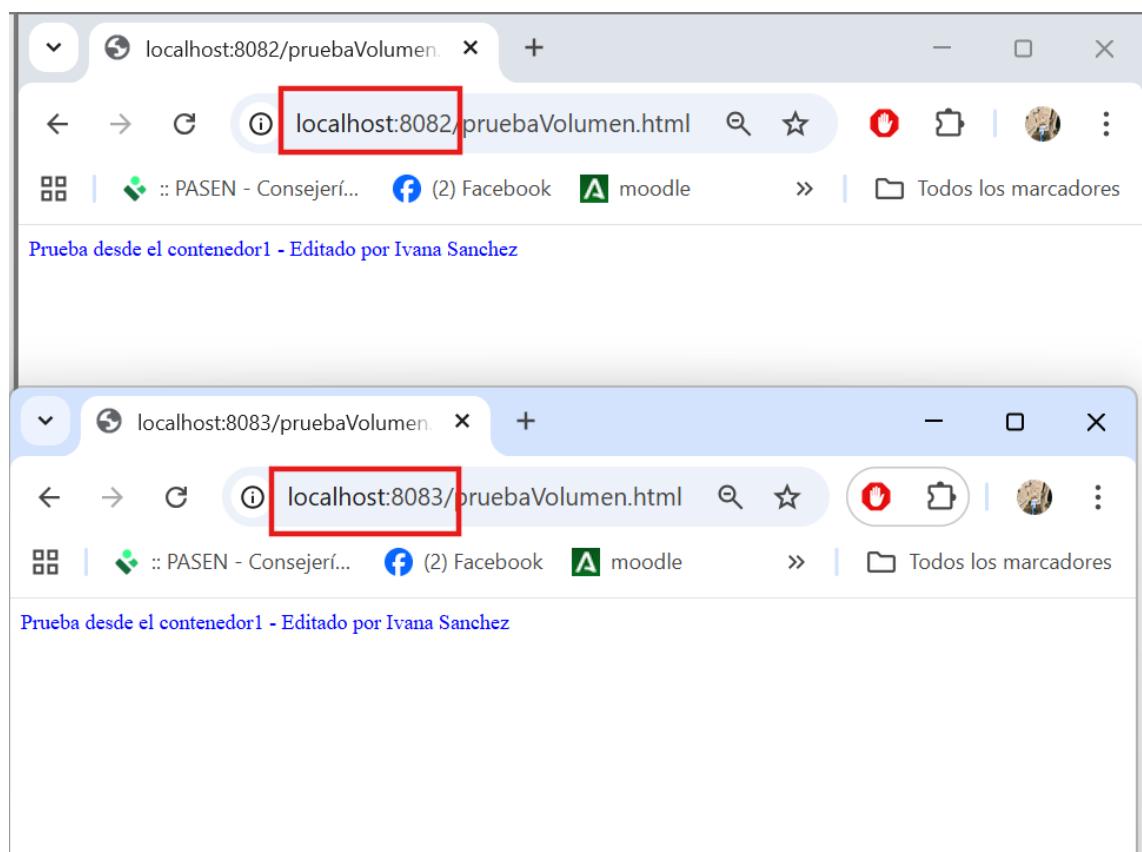
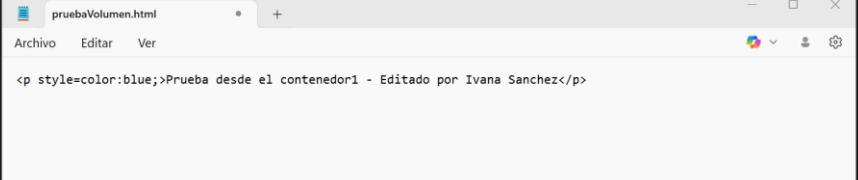
Edita el fichero desde el host y pon tu nombre, observa los cambios en ambos contenedores

Como utilizamos WSL y no se puede acceder directamente desde el explorador del archivos al html, copiaré el archivo **pruebaVolumen.html** desde el contenedor al escritorio de mi proyecto. Desde ahí lo abrimos y modificamos con Notepad, guardamos y lo copiamos de nuevo al contenedor.

Desde la ubicación de mi carpeta de la tarea:

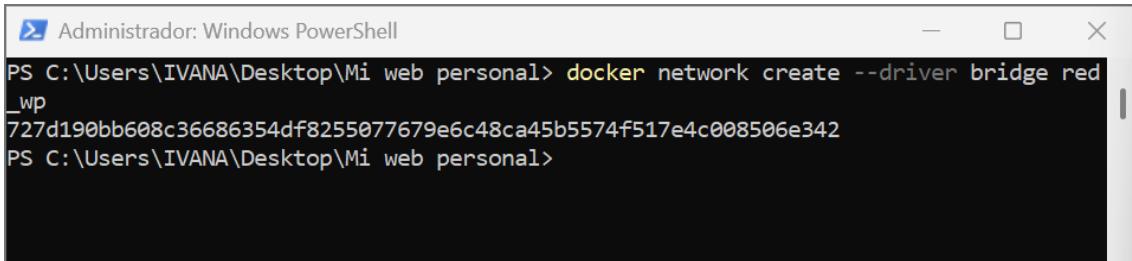
- 1- docker volume inspect volumen_web_compartido
- 2- wsl -l -v
- 3- docker cp contenedor1:/var/www/html/pruebaVolumen.html pruebaVolumen.html
- 4- notepad .\pruebaVolumen.html
- 5- docker cp pruebaVolumen.html contenedor1:/var/www/html/pruebaVolumen.html

```
Administrator: Windows PowerShell
PS C:\Users\IVANA\Desktop\Mi web personal> docker volume inspect volumen_web_compartido
[{"Name": "volumen_web_compartido", "Driver": "local", "Scope": "local", "Labels": null, "Mountpoint": "/var/lib/docker/volumes/volumen_web_compartido/_data", "Options": null, "CreatedAt": "2025-10-19T17:00:10Z", "State": "Running", "Version": 2}
]
PS C:\Users\IVANA\Desktop\Mi web personal> wsl -l -v
NAME           STATE    VERSION
* docker-desktop   Running      2
Ubuntu          Stopped      2
PS C:\Users\IVANA\Desktop\Mi web personal> docker cp contenedori:/var/www/html/pruebaVolumen.html pruebaVolumen.html
Successfully copied 2.05kB to C:\Users\IVANA\Desktop\Mi web personal\pruebaVolumen.html
PS C:\Users\IVANA\Desktop\Mi web personal> notepad .\pruebaVolumen.html
PS C:\Users\IVANA\Desktop\Mi web personal>
```



- Vamos a conectar dos contenedores en red, para ello, crea una red de tipo bridge y a continuación crea los siguientes dos contenedores utilizando la red creada anteriormente.

```
docker network create --driver bridge red_wp
```



```
PS C:\Users\IVANA\Desktop\Mi web personal> docker network create --driver bridge red_wp
727d190bb608c36686354df8255077679e6c48ca45b5574f517e4c008506e342
PS C:\Users\IVANA\Desktop\Mi web personal>
```

El primer contenedor será una base de datos mariadb y lo llamaremos servidor_mysql:

```
docker run -d --name servidor_mysql --network XXXX -e
MYSQL_DATABASE=bd_wp -e MYSQL_USER=user_wp -e
MYSQL_PASSWORD=test -e MYSQL_ROOT_PASSWORD=test mariadb
```

```
docker run -d --name servidor_mysql --network red_wp -e
MYSQL_DATABASE=db_wp -e MYSQL_USER=user_wp -e
MYSQL_PASSWORD=test -e MYSQL_ROOT_PASSWORD=test mariadb
```

```

PS C:\Users\IVANA\Desktop\Mi web personal> docker run -d --name servidor_mysql --net
work red_wordpress -e MYSQL_DATABASE=bd_wp -e MYSQL_USER=user_wp -e MYSQL_PASSWORD=t
est -e MYSQL_ROOT_PASSWORD=test mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
481da1b848c7: Pull complete
2d279465a633: Pull complete
a1a21c96bc16: Pull complete
7e655d170aaaf: Pull complete
a703234cef5c: Pull complete
9a2a3f7e7f07: Pull complete
69f1e38330ba: Pull complete
181016d47749: Pull complete
Digest: sha256:03a03a6817bb9eaa21e5aed1b734d432ec3f80021f5a2de1795475f158217545
Status: Downloaded newer image for mariadb:latest
debd0a2534997b5583900d37997de60ca9826f56bacb19acb603e715fbb0ca19
PS C:\Users\IVANA\Desktop\Mi web personal> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
ATUS PORTS NAMES
debd0a253499 mariadb "docker-entrypoint.s..." About a minute ago Up
About a minute 3306/tcp servidormysql
a65eefc358a5 servidor-web-ivana "/usr/sbin/apache2ct..." 58 minutes ago Up
58 minutes 0.0.0.0:8081->80/tcp contenedor2
cad9621ac6bb servidor-web-ivana "/usr/sbin/apache2ct..." 58 minutes ago Up
58 minutes 0.0.0.0:8080->80/tcp contenedor1
ac7a607d065d mysql:8 "docker-entrypoint.s..." 29 hours ago Up
About an hour 3306/tcp, 33060/tcp pettracer-mysql-1
PS C:\Users\IVANA\Desktop\Mi web personal>

```

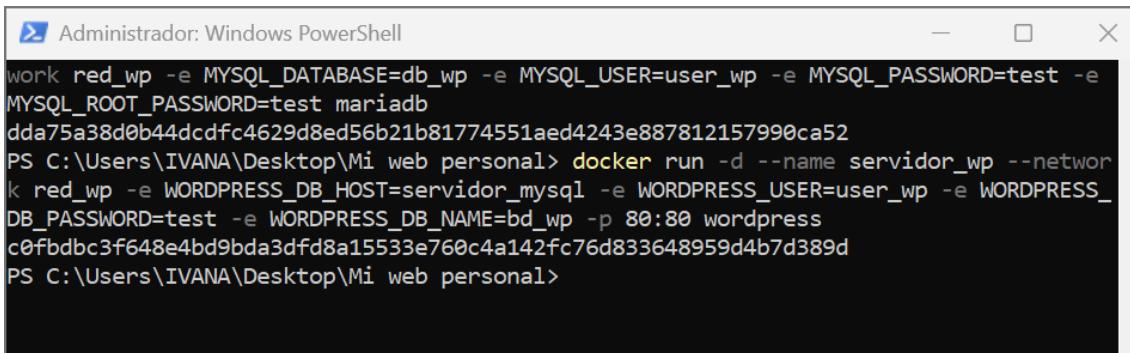
A continuación vamos a crear un contenedor con el nombre servidor_wp, a partir de la imagen wordpress, conectada a la misma red y con las variables de entorno necesarias:

```

docker run -d --name servidor_wp --network XXXX -e
WORDPRESS_DB_HOST=servidor_mysql -e
WORDPRESS_DB_USER=user_wp
-e WORDPRESS_DB_PASSWORD=test -e WORDPRESS_DB_NAME=bd_wp -p
80:80 wordpress

```

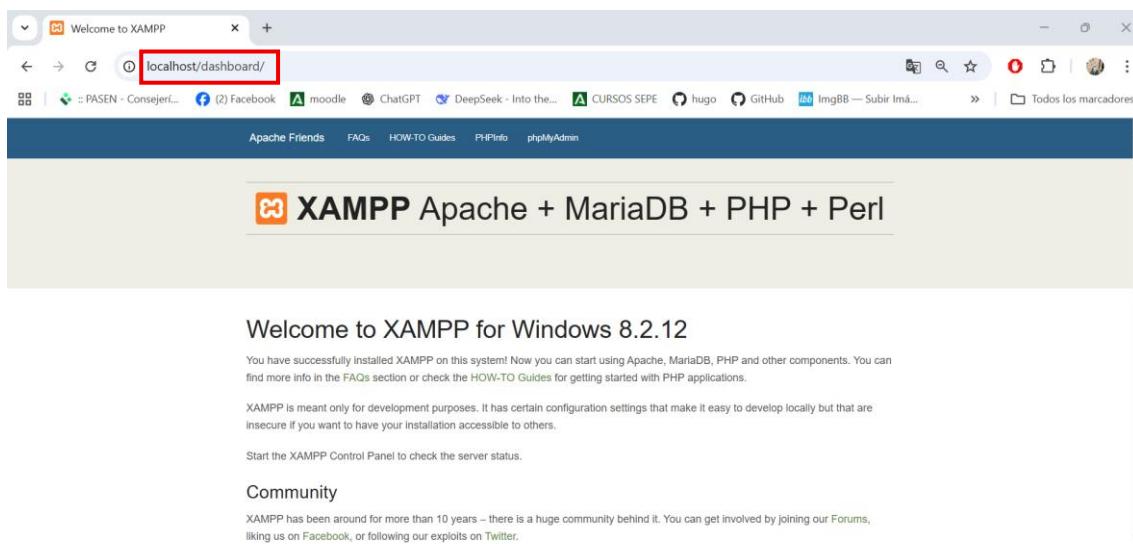
***docker run -d --name servidor_wp --network red_w -e
WORDPRESS_DB_HOST=servidor_mysql -e
WORDPRESS_DB_USER=user_wp -e WORDPRESS_DB_PASSWORD=test -
e WORDPRESS_DB_NAME=bd_wp -p 80:80 wordpress***



```
Administrator: Windows PowerShell
work red_wp -e MYSQL_DATABASE=db_wp -e MYSQL_USER=user_wp -e MYSQL_PASSWORD=test -e
MYSQL_ROOT_PASSWORD=test mariadb
dd75a38d0b44dcdfc4629d8ed56b21b81774551aed4243e887812157990ca52
PS C:\Users\IVANA\Desktop\Mi web personal> docker run -d --name servidor_wp --networ
k red_wp -e WORDPRESS_DB_HOST=servidor_mysql -e WORDPRESS_USER=user_wp -e WORDPRESS_
DB_PASSWORD=test -e WORDPRESS_DB_NAME=bd_wp -p 80:80 wordpress
c0fbdbc3f648e4bd9bda3dfd8a15533e760c4a142fc76d833648959d4b7d389d
PS C:\Users\IVANA\Desktop\Mi web personal>
```

Accede a la ip del servidor docker y comprueba la instalación de wordpress. La variable WORDPRESS_DB_HOST hace referencia a la IP donde está la base de datos mysql ¿Por qué crees que hemos podido poner “servidor_mysql” en la variable WORDPRESS_DB_HOST?

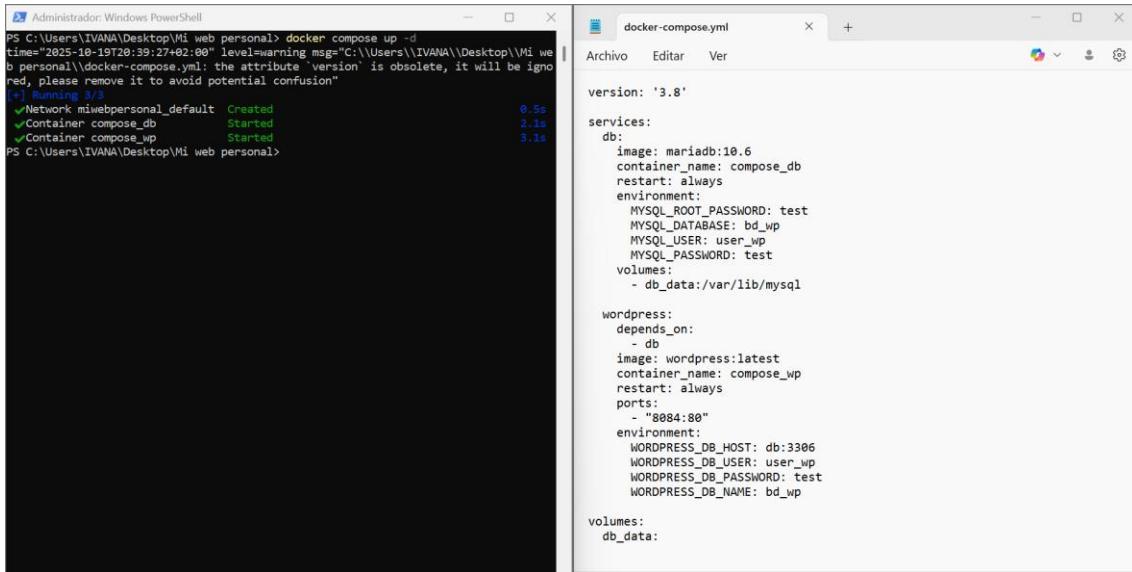
Se puede usar el nombre del contenedor (servidor_mysql) en vez de su IP porque, al conectar ambos contenedores (servidor_mysql y servidor_wp) a la misma red de tipo puente (red_wordpress), Docker configura automáticamente un servicio interno de DNS. Esto permite que los contenedores dentro de esa red se comuniquen usando el nombre que le he dado. Esto hace que la configuración sea más flexible y fácil de usar, ya que las IP internas de Docker son dinámicas.



7. Buscar en docker hub la imagen de wordpress y cómo se instala con docker-compose. Generar el fichero correspondiente para arrancarlo en vuestras máquinas.

notepad docker-compose.yml

Para levantar el contendor utilizaremos el comando: **docker compose up -d**



```
PS C:\Users\IVANA\Desktop\Mi web personal> docker compose up -d
time='2025-10-19T20:39:27+02:00' level=warning msg="C:\\Users\\IVANA\\Desktop\\Mi web personal\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
  ✓ Network mivebpersonal_default Created          0.5s
  ✓ Container compose_db Started                  2.1s
  ✓ Container compose_wp Started                  3.1s
PS C:\Users\IVANA\Desktop\Mi web personal>
```

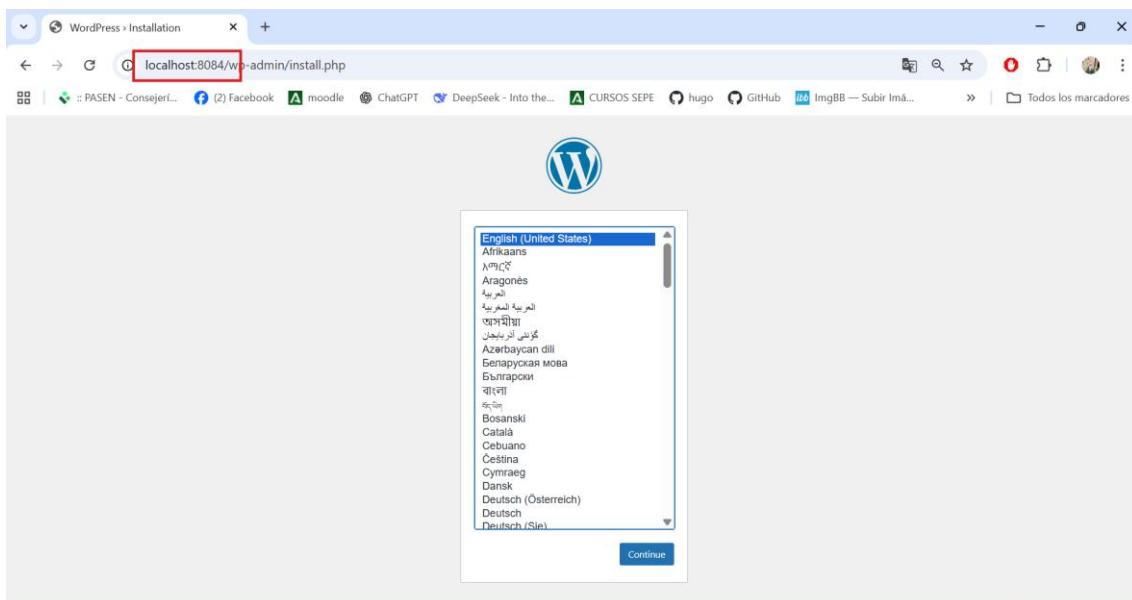
```
version: '3.8'

services:
  db:
    image: mariadb:10.6
    container_name: compose_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: test
      MYSQL_DATABASE: bd_wp
      MYSQL_USER: user_wp
      MYSQL_PASSWORD: test
    volumes:
      - db_data:/var/lib/mysql

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    container_name: compose_wp
    restart: always
    ports:
      - "8084:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: user_wp
      WORDPRESS_DB_PASSWORD: test
      WORDPRESS_DB_NAME: bd_wp

volumes:
  db_data:
```

En el navegador ponemos el puerto que le hemos dado a WordPress en el fichero de docker-compose.yml → 8084



¿Qué servicios están siendo definidos en este archivo de configuración?

Se definen 2 servicios: db para la base de datos y worpress para la aplicación web

¿Qué puertos están siendo mapeados para el servicio de Wordpress?

El puerto interno del contenedor (80) se está mapeando al puerto 8082 de mi pc para no interferir con el puerto 80 que ya se está usando.

¿Qué significa el atributo "restart: always" en la definición de ambos servicios?

Que si un contenedor se detiene por cualquier razón, Docker Compose intentará reiniciarlo automáticamente.

¿Qué imagen está siendo utilizada para el servicio de base de datos MySQL?

La imagen de mariadb:10.6

8. Buscar en docker hub la aplicación Portainer y levantar un contenedor. Investigar sobre su funcionamiento, ¿Para que se utiliza esta aplicación?

Portainer es una herramienta de gestión gráfica para Docker. Su principal función es simplificar la administración del entorno Docker. Actúa como un panel de control web que permite a los usuario gestionar, monitorear y desplegar contenedores, imágenes, volúmenes y redes mediante una interfaz visual, eliminando el uso de los comandos.

```
docker run -d --name portainer --restart always -p 9443:9443 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

