

## Contenedores Docker. Docker-compose

### 1. Apuntes previos:

#### a) Ejemplo de estructura del fichero **docker-compose.yml**

```
version: '3.1'
services:
  app:
    container_name: guestbook
    image: iesgn/guestbook
    restart: always
    ports:
      - 8080:5000
  db:
    container_name: redis
    image: redis
    restart: always
    volumes:
      - redis:/data
volumes:
  redis:
```

En este fichero YAML se definen los datos relativos a los contenedores de la aplicación como **servicios** (**app** y **db**) y dentro de cada servicio los datos del nombre del **contenedor**, la **imagen** a descargar, la opción de **reiniciarse** si se apaga, las **variables de entorno**, los **puertos** que se exponen y los **volúmenes** de datos persistentes que se van a emplear

Este fichero debe guardarse en una carpeta de nuestro sistema y a continuación desde la consola del sistema, se debe acceder a dicha carpeta para ejecutar el comando que crea los contenedores definidos en el archivo:

```
docker-compose up -d
```

#### b) Opciones habituales de docker-compose (en **negrita** las más típicas):

- `docker-compose up`: Crea los contenedores (servicios) que están descritos en el `docker-compose.yml`.
- **`docker-compose up -d`**: Crea en modo detach los contenedores (servicios) que están descritos en el
- **`docker-compose stop`**: Detiene los contenedores que previamente se han lanzado con `docker-compose up`.
- **`docker-compose run`**: Inicia los contenedores descritos en el `docker-compose.yml` que estén parados.
- `docker-compose rm`: Borra los contenedores parados del escenario. Con la opción `-f` elimina también los contenedores en ejecución.

- **docker-compose restart:** Reinicia los contenedores. Orden ideal para reiniciar servicios con nuevas configuraciones.
- **docker-compose down:** Para los contenedores, los borra y también borra las redes que se han creado con docker-compose up (en caso de haberse creado).
- **docker-compose down -v:** Para los contenedores y borra contenedores, redes y volúmenes.
- **docker-compose exec servicio1 /bin/bash:** Ejecuta una orden, en este caso /bin/bash en un contenedor llamado servicio1 que estaba descrito en el docker-compose.yml
- **docker-compose build:** Ejecuta, si está indicado, el proceso de construcción de una imagen que va a ser usado en el docker-compose.yml a partir de los ficheros Dockerfile que se indican.

c) Volúmenes con docker-compose → Ejemplo de volumen para mariadb:

```
version: '3.1'

services:
  db:
    container_name: contenedor_mariadb
    image: mariadb
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: asdasd
    volumes:
      - mariadb_data:/var/lib/mysql

volumes:
  mariadb_data:
```

Se definen en el bloque ultimo volumes: pero además hay que indicar que volumen usara cada contenedor como se indica que se va a crear un volumen mariadb\_data apuntando a la carpeta /var/lib/mysql del contenedor

También es posible crear volúmenes en carpetas concretas dentro de nuestro sistema como se muestra:

```
volumes:
  - c:\users\avelino\mariadb_data:/var/lib/mysql
```

2. Se pide ahora desplegar la aplicación Guestbook a partir del fichero docker-compose.yml que se muestra en el apartado 1. a). Este fichero lo guardaremos en una carpeta de nuestro usuario de Windows 10 y una vez en la consola accedemos a la carpeta para desplegar la aplicación con el comando `docker compose up -d`. Verificar el acceso a la aplicación con <http://localhost:8080>

Comprobar también desde la vista de los volúmenes de Docker Desktop que el almacenamiento persistente para la base de datos está creado

3. Crear ahora un fichero **docker-compose.yml** para el despliegue de la aplicación de Temperaturas que se realizó en la práctica anterior, con los siguientes datos:

- a) versión 3.1
- b) los servicios serán frontend y backend
- c) para el servicio frontend:
  - a. el contenedor se llamará temperaturas-frontend
  - b. la imagen para usar es avelinopef/temperaturas\_front
  - c. se debe reiniciar automáticamente
  - d. El mapeo de puertos es 80:3000
  - e. Este servicio depende de backend (usaremos depends-on)
- d) Para el servicio backend:
  - a. el contenedor se llamará temperaturas-backend
  - b. la imagen para usar es avelinopef/temperaturas\_back
  - c. se debe reiniciar automáticamente

Una vez creado, como en el caso anterior debemos guardarlo en una carpeta particular y desplegarlo con `docker-compose up -d`

Verificar desde un navegador que se accede a la aplicación con <http://localhost>

4. Se pide ahora desplegar la aplicación de Wordpress con docker-compose a partir del fichero wp\_mariadb.yml que se entrega.

Observa los valores de las variables de entorno y ajusta los valores de modo que:

- a) El nombre del contenedor de wordpress será contenedor\_wp
- b) El nombre del contenedor de mariadb será contenedor\_mariadb
- c) El usuario para wp en mariadb será admin
- d) La contraseña del usuario anterior será naranco23
- e) La base de datos de wp será bbdd\_wp
- f) El volumen de datos para wordpress será datos\_wp
- g) El volumen de datos para mariadb será datos\_mariadb

Ya que el fichero no cuenta con el nombre habitual por defecto (docker-compose.yml), la composición debe ser lanzada con la orden:

```
docker-compose -f wp_mariadb.yml up -d
```

Accede al <http://localhost> y completa la instalación de wp

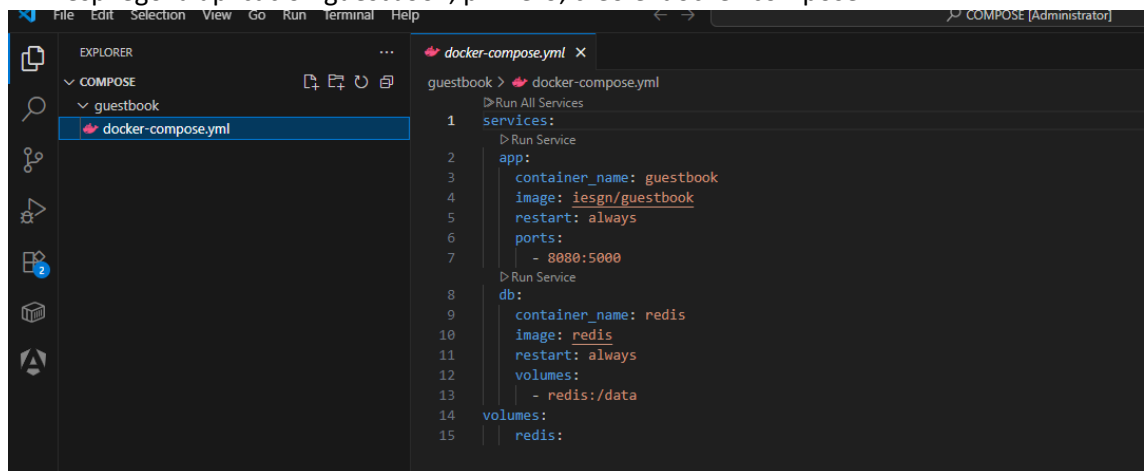
5. Ahora desplegaremos una aplicación java sobre un servidor Tomcat y con proxy inverso nginx que permita el acceso a la web de la aplicación. Para ello:
  - a) Vamos a utilizar los volúmenes para copiar el archivo sample.war al directorio de publicación de tomcat y el fichero default.conf para copiar una versión modificada

del fichero por defecto de nginx. Estos dos archivos deben copiarse una carpeta particular de nuestro sistema (por ejemplo c:\users\usuario\tomcat)

- b) Copiamos a la misma carpeta el fichero tomcat-nginx.yml
  - c) Desplegamos como siempre con docker-compose
  - d) Verificamos el acceso a la aplicación con <http://localhost>
6. Se pide ahora realizar el despliegue con docker-compose de la aplicación phpmyadmin que hará uso de un servidor de bases de datos mariadb. Para ello:
- a) Debe crearse un fichero Docker-compose.yml que permita el desplegar los dos servicios
  - b) Se crearán también volúmenes de persistencia de datos para los dos contenedores
  - c) Se verificará el acceso a phpmyadmin desde la URL cargada en un navegador
7. Se pide ahora desplegar la tienda virtual Prestashop. Para ello se entrega el fichero prestashop.yml y se deben realizar los siguientes cambios:
- a) El usuario de la base de datos para prestashop será user\_prestashop
  - b) El nombre de la base de datos será mitienda

Se debe acceder a la URL de la tienda y mostrar la aplicación online en el navegador

1.- Despliegue la aplicación guestbook, primero, creo el docker-compose:



2.- Tras esto lo ejecuto, esta vez desde el visual studio, después serán todas desde la consola.

```
uestbook > docker-compose.yml
  ▸ Run All Services
1  services:
  ▸ Run Service
2    app:
3      container_name: guestbook
4      image: iesgn/guestbook
5      restart: always
6      ports:
7        - 8000:5000
  ▸ Run Service
8    db:
9      container_name: redis
10     image: redis
11     restart: always
12     volumes:
13       - redis:/data
14 volumes:
15   redis:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Unable to get image 'iesgn/guestbook': error during connect: Get "http://%20
contrar el archivo especificado.

* The terminal process "C:\Program Files\Git\bin\bash.exe '--login', '-i'
* Terminal will be reused by tasks, press any key to close it.

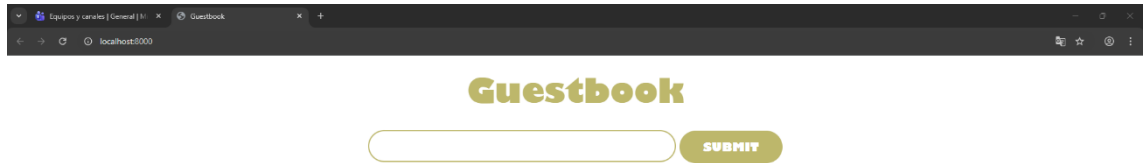
* Executing task: docker compose -f 'guestbook\docker-compose.yml' up -d

[+] Running 14/20
- app [██████████] 342MB / 344.5MB Pulling
- db [██████████] 52.98MB / 52.98MB Pulling
```

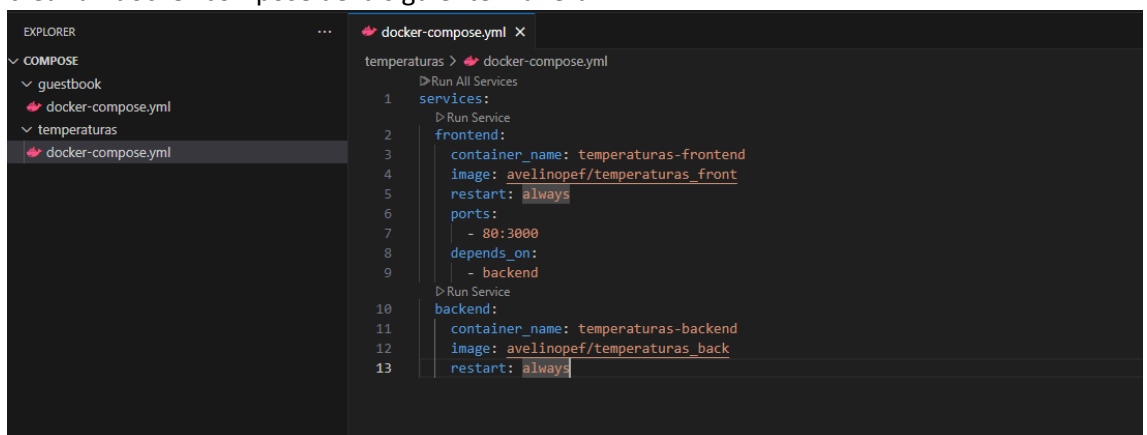
3.- Como se puede observar ya están funcionando los dos contenedores.

<input type="checkbox"/>	guestbook	-	-	-	0.51%	36 seconds ago			
<input type="checkbox"/>	redis	a24872914830	redis		0.26%	36 seconds ago			
<input type="checkbox"/>	guestbook	74f5ffad5760	iesgn/guestbook	8000:5000 ↗	0.25%	36 seconds ago			

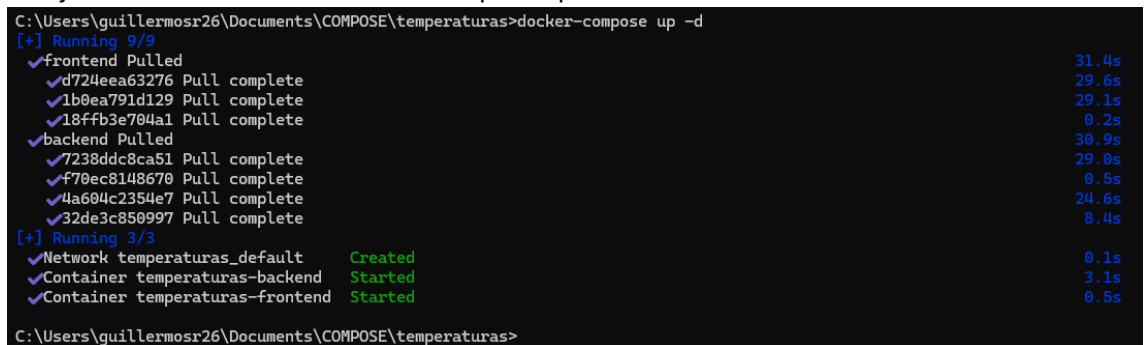
4.- Ahora si vamos a localhost:8000, podremos ver la página funcionando.



5.- Ahora vamos a desplegar la página de temperaturas de avelino, asique volvemos a crear un docker-compose de la siguiente manera.



6.- Ejecutamos el comando docker-compose up -d



Y ya la tendríamos funcionando



7.- Ahora vamos a hacer lo mismo para la página de wordpress, para ello descargamos el docker-compose que nos dan y lo editamos y cambiamos por lo que se nos pide:

```
EXPLORER  ...  Extension: Container Tools  ! wp_mariadb.yml  Extension: Dev Containers

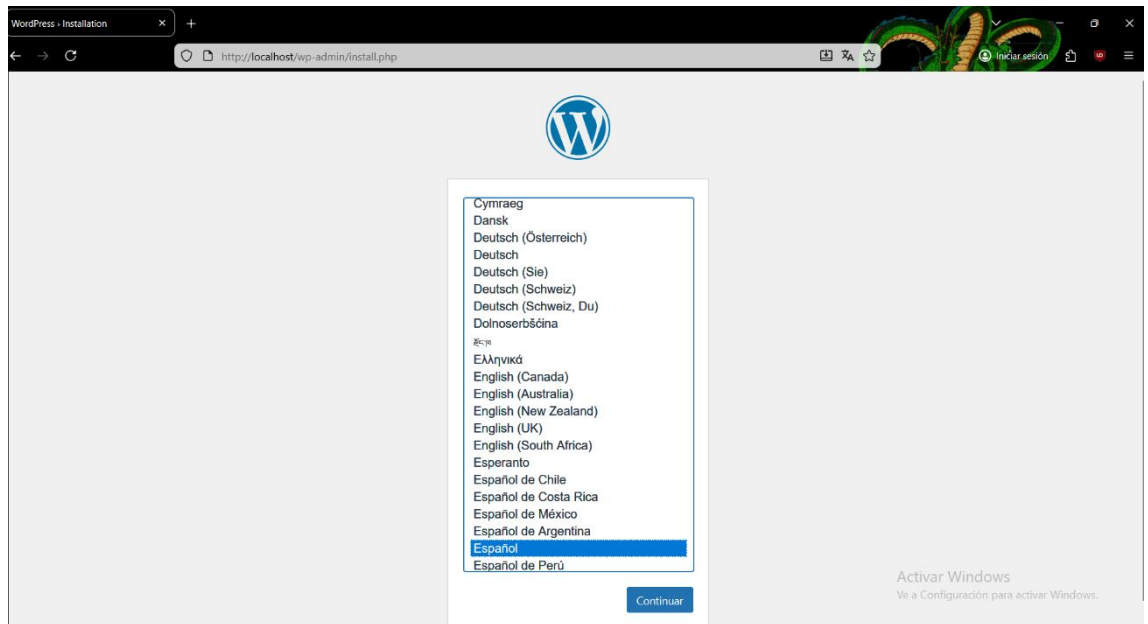
COMPOSE
! wp_mariadb.yml

! wp_mariadb.yml
1  version: '3.1'
2  services:
3    wordpress:
4      container_name: contenedor_wp
5      image: wordpress
6      restart: always
7      environment:
8        WORDPRESS_DB_HOST: db
9        WORDPRESS_DB_USER: admin
10       WORDPRESS_DB_PASSWORD: naranco23
11       WORDPRESS_DB_NAME: bbdd_wp
12     ports:
13       - 80:80
14     volumes:
15       - datos_wp:/var/www/html/wp-content
16   db:
17     container_name: contenedor_mariadb
18     image: mariadb
19     restart: always
20     environment:
21       MYSQL_DATABASE: bbdd_wp
22       MYSQL_USER: admin
23       MYSQL_PASSWORD: naranco23
24       MYSQL_ROOT_PASSWORD: naranco23
25     volumes:
26       - datos_mariadb:/var/lib/mysql
27   volumes:
28     datos_wp:
29     datos_mariadb:
```

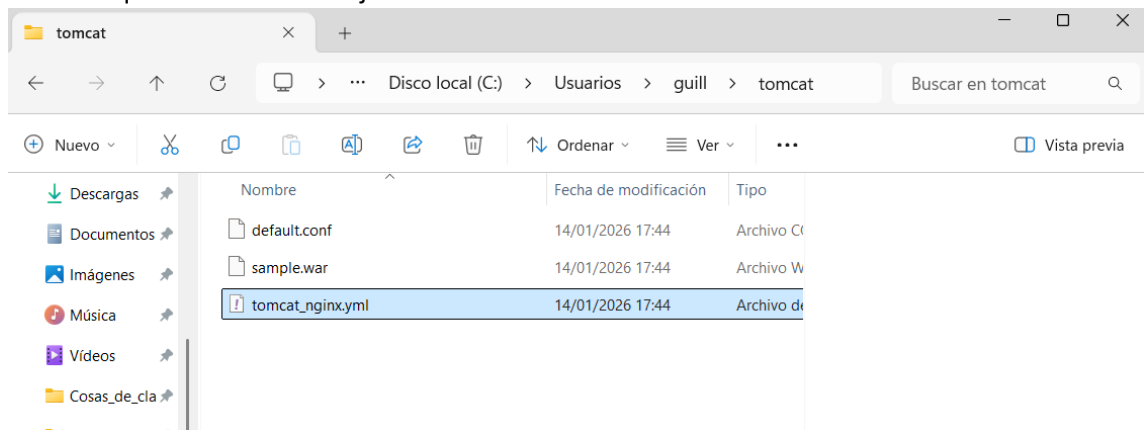
8.- después de esto debemos ejecutar el comando, en este caso es con -f ya que el archivo adjunto no se llama docker-compose.

```
C:\Users\guill\Documents\2DAWB\COMPOSE>docker-compose -f wp_mariadb.yml up -d
```

9.- Ahora si vamos al localhost ya tenemos la página de wordpress funcionando.



10.- Para continuar, creamos una carpeta en c:\users\tomcat y metemos ahí los archivos que nos dan con el ejercicio.



11.- Volvemos a hacer lo mismo con el docker-compose -f

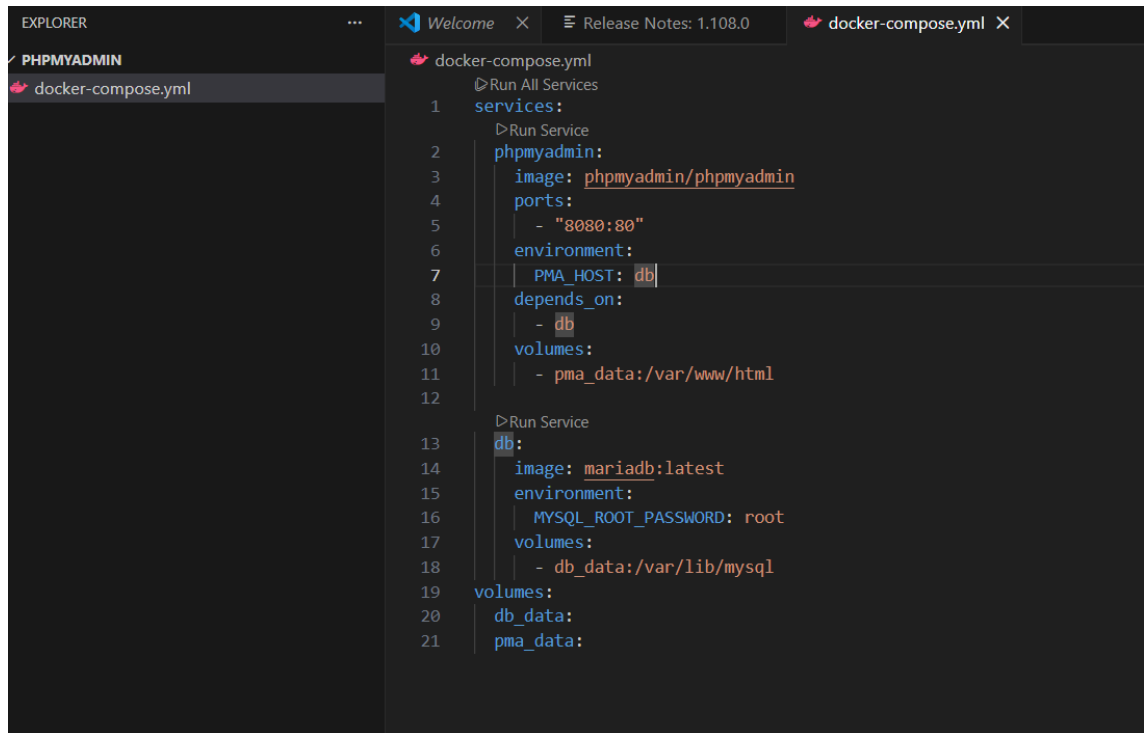
```
C:\Users\guill\tomcat>docker-compose -f tomcat_nginx.yml up -d
```

12.- Ahora si entramos a localhost nos saldrá el sample.



13.- Ahora crearemos un contenedor para phpmyadmin con mariadb. Primero creamos el docker-compose de la siguiente manera.

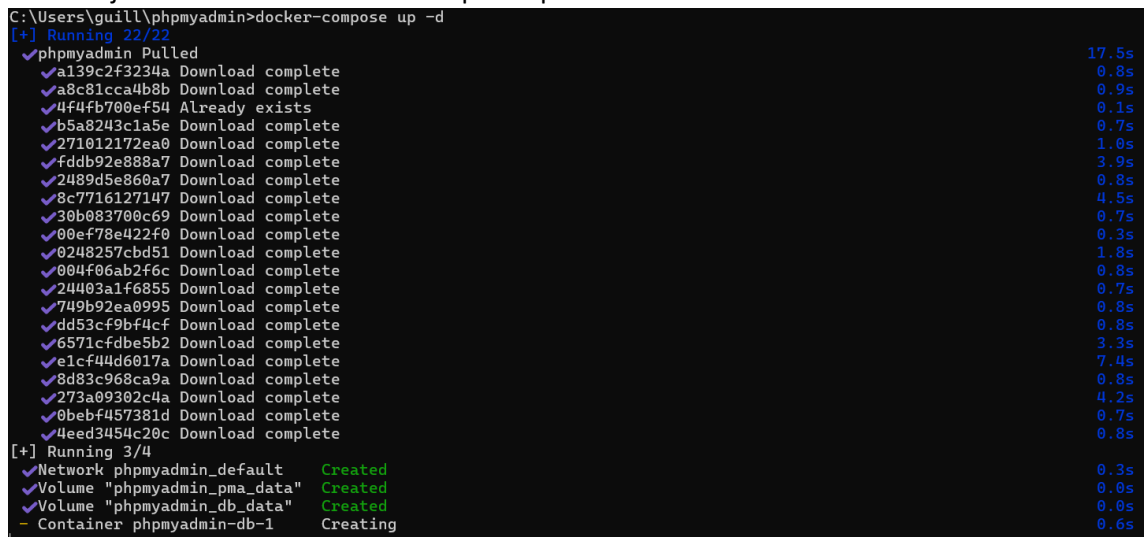




```
EXPLORER
PHPMYADMIN
docker-compose.yml

docker-compose.yml
1  Run All Services
2  services:
3    >Run Service
4    phpmyadmin:
5      image: phpmyadmin/phpmyadmin
6      ports:
7        - "8080:80"
8      environment:
9        PMA_HOST: db
10     depends_on:
11       - db
12     volumes:
13       - pma_data:/var/www/html
14
15   >Run Service
16   db:
17     image: mariadb:latest
18     environment:
19       MYSQL_ROOT_PASSWORD: root
20     volumes:
21       - db_data:/var/lib/mysql
22
23 volumes:
24   db_data:
25   pma_data:
```

14.- Lo ejecutamos con el docker-compose up -d



```
C:\Users\guill\phpmyadmin>docker-compose up -d
[+] Running 22/22
  ✓ phpmyadmin Pulled                                17.5s
  ✓ a139c2f3234a Download complete                   0.8s
  ✓ a8c81cca4b8b Download complete                   0.9s
  ✓ 4f4fb700ef54 Already exists                      0.1s
  ✓ b5a8243c1a5e Download complete                   0.7s
  ✓ 271012172ea0 Download complete                   1.0s
  ✓ fddb92e888a7 Download complete                   3.9s
  ✓ 2489d5e860a7 Download complete                   0.8s
  ✓ 8c7716127147 Download complete                   4.5s
  ✓ 30b083700c69 Download complete                   0.7s
  ✓ 00ef78e422f0 Download complete                   0.3s
  ✓ 0248257cbd51 Download complete                   1.8s
  ✓ 004f06ab2f6c Download complete                   0.8s
  ✓ 24403a1f6855 Download complete                   0.7s
  ✓ 749b92ea0995 Download complete                   0.8s
  ✓ dd53cf9bf4cf Download complete                   0.8s
  ✓ 6571cfdb5b2 Download complete                   3.3s
  ✓ e1cf44d6017a Download complete                   7.4s
  ✓ 8d83c968ca9a Download complete                   0.8s
  ✓ 273a09302c4a Download complete                   4.2s
  ✓ 0beb457381d Download complete                   0.7s
  ✓ 4eed3454c20c Download complete                   0.8s
[+] Running 3/4
  ✓ Network phpmyadmin_default                      0.3s
  ✓ Volume "phpmyadmin_pma_data" Created             0.0s
  ✓ Volume "phpmyadmin_db_data" Created              0.0s
  - Container phpmyadmin-db-1 Creating               0.6s
```

15.- Y si ahora vamos a localhost:8080 nos aparecerá la interfaz de phpMyAdmin, donde usuario será root y contraseña root.



  
Bienvenido a phpMyAdmin

Idioma (Language)

Español - Spanish ▼

Iniciar sesión ⓘ

Usuario:

Contraseña:

Iniciar sesión

16.- Este es el .yaml del último ejercicio, no sé si he hecho algo mal, pero a la hora de hacer el docker-compose up me da un error, estuve buscando en internet y parece que han puesto algunos repositorios privados.

```
EXPLORER
└─ PRESTASHOP
  └─ prestashop.yaml

! prestashop.yaml
1 version: '2'
2 services:
3   mariadb:
4     image: docker.io/bitnami/mariadb:10.6
5     environment:
6       - ALLOW_EMPTY_PASSWORD=yes
7       - MARIADB_USER=user_prestashop
8       - MARIADB_DATABASE=mitienda
9     volumes:
10      - 'mariadb_data:/bitnami/mariadb'
11   prestashop:
12     image: docker.io/bitnami/prestashop:8
13     ports:
14       - '80:8080'
15       - '443:8443'
16     environment:
17       - PRESTASHOP_HOST=localhost
18       - PRESTASHOP_DATABASE_HOST=mariadb
19       - PRESTASHOP_DATABASE_PORT_NUMBER=3306
20       - PRESTASHOP_DATABASE_USER=user_prestashop
21       - PRESTASHOP_DATABASE_NAME=mitienda
22       - ALLOW_EMPTY_PASSWORD=yes
23     volumes:
24       - 'prestashop_data:/bitnami/prestashop'
25     depends_on:
26       - mariadb
27   volumes:
28     mariadb_data:
29       driver: local
30     prestashop_data:
31       driver: local
```