

Despliegue de WordPress usando contenedores Docker y Docker Compose

Despliegue de Aplicaciones Web



Daniel Godoy Galindo
2º DAW Vespertino

Índice

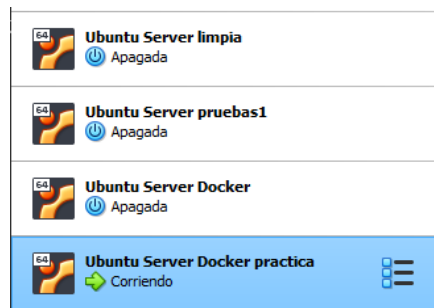
Índice.....	2
Previo a la configuración de SSH.....	3
Conexión por SSH en Visual Studio Code.....	4
Instalación de Docker y Docker Compose.....	6
Descripción de la configuración del archivo docker-compose.yml.....	7
Descripción de las acciones que se han realizado durante la puesta en producción..	14
Capturas del acceso a wordpress, phpMyAdmin y el no acceso a la base de datos mysql.....	16
Repositorio GitHub.....	20
Extra: Políticas de reinicio.....	22

Previo a la configuración de SSH

Creación de una máquina virtual Ubuntu Server:

He clonado una máquina virtual Ubuntu Server que tengo para las prácticas y la he configurado con conexión de Adaptador Puente. He instalado también ssh y añadida una clave ssh:

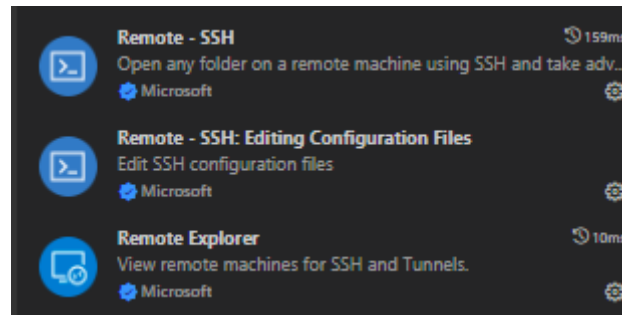
```
apt-get install ssh      ssh-keygen
```



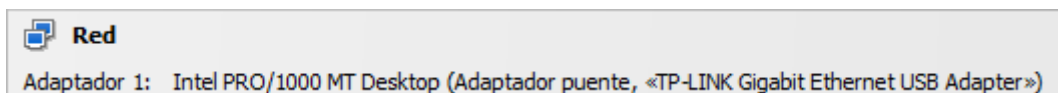
```
alumno@alumno:~$ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/alumno/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alumno/.ssh/id_ed25519
Your public key has been saved in /home/alumno/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:zQNckg3JT5IaL9L755fHPFxQbtphBFxnfaE/MVI3s3o alumno@alumno
The key's randomart image is:
+--[ED25519 256]--+
|      .0*. ..0**|
|      ..*00  .00X|
|      .+0+   0.*.|
|      . + .+   =0=|
|      . OS +  ..E.|
|      .      . 0 +|
|      .      = .|
|      . . 0 *|
|      0.. . .|
+-----[SHA256]-----+
```

Conexión por SSH en Visual Studio Code

Para poder conectarme a mi servidor de Ubuntu Server desde Visual Studio Code, primero he instalado las extensiones de Microsoft de SSH en Visual Studio Code:



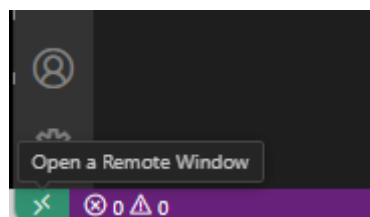
He configurado Ubuntu Server con su conexión en **Adaptador Puente**.



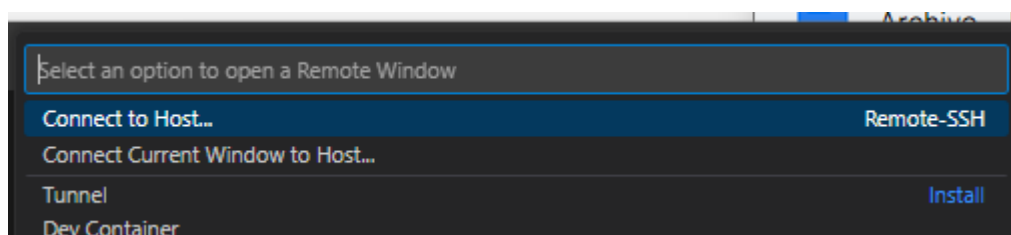
Y he comprobado cual es mi dirección IP:

```
alumno@alumno:~$ ifconfig  
enp0s3: flags=4163<UP,BROADCAST,SMART  
    inet 192.168.1.78
```

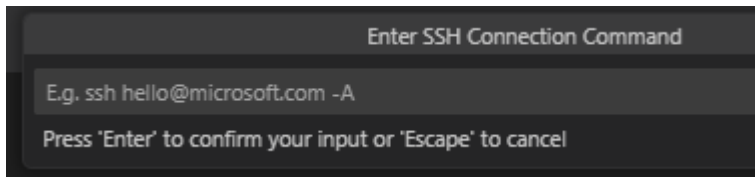
Ahora configuro la conexión SSH desde Visual Studio Code. En la parte inferior izquierda hay un botón verde para abrir una nueva ventana remota:



Le doy a connect to host:



Y después a “Add New SSH Host”:

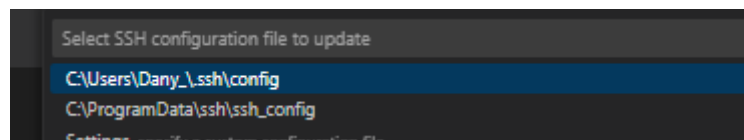


Aparece una línea de comandos para indicar la orden ssh para conectarse

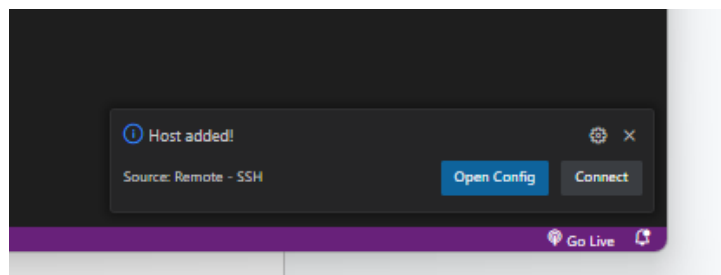
Y se pone la instrucción ssh → ssh alumno@ip_del_servidor_ubuntu

En mi caso → ssh alumno@192.168.1.78

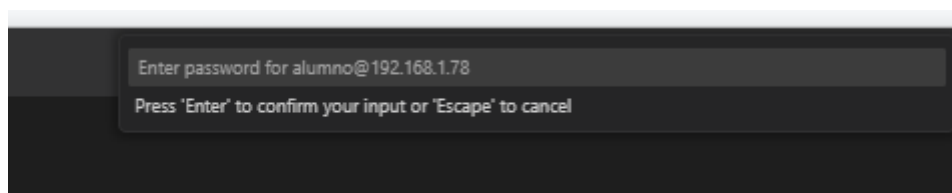
Se selecciona el fichero de configuración:



Aparece abajo la notificación para conectarse y se le da a connect:



Pide la contraseña del servidor al que te quieres conectar, y listo:



Instalación de Docker y Docker Compose

Estos son todos los comandos que he usado para instalar docker y docker-compose:

```
sudo apt update
sudo apt install -y docker.io
sudo apt install -y docker-compose
sudo apt install -y docker-compose-v2
```

Tras instalar todo, compruebo con unos comandos que todo esté correcto:

```
alumno@alumno:~/practicaDocker$ sudo docker version
Client:
 Version:           26.1.3
 API version:       1.45
 Go version:        go1.22.2
 Git commit:        26.1.3-0ubuntu1~24.04.1
 Built:             Mon Oct 14 14:29:26 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          26.1.3
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.22.2
  Git commit:       26.1.3-0ubuntu1~24.04.1
  Built:            Mon Oct 14 14:29:26 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.24
  GitCommit:
 runc:
  Version:          1.1.12-0ubuntu3.1
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
```

```
alumno@alumno:~/practicaDocker$ sudo docker-compose version
docker-compose version 1.29.2, build unknown
docker-py version: 5.0.3
CPython version: 3.12.3
OpenSSL version: OpenSSL 3.0.13 30 Jan 2024
```

Lo siguiente es crear un fichero de configuración para composer. Creo una carpeta llamada practicaDocker y dentro creo el fichero:

```
alumno@alumno:~$ mkdir practicaDocker
alumno@alumno:~$ cd practicaDocker/
alumno@alumno:~/practicaDocker$ touch docker-compose.yml
```

Descripción de la configuración del archivo docker-compose.yml

Partiendo de un fichero de muestra voy a configurar mi fichero .yml

```
services:
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_NAME: exampledb
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
    depends_on:
      - db
    restart: always

  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: examplepass
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
    restart: always
```

Un fichero de configuración de docker composer define servicios. El que voy a crear va a definir tres servicios: la base de datos MySQL, phpMyAdmin para conectarse a la BDD y Wordpress. En el fichero también se indican cuales son los volúmenes para la persistencia de datos y las redes usadas para interconectar los servicios.

El fichero comienza por la directiva "Services" y a partir de ahí se indica cual es la configuración de cada servicio. La mayoría de directivas son comunes a todos los servicios como por ejemplo "image", "container-name", "volumes" y "networks".

Como primera **configuración de MySQL** indico estos datos en el fichero decompose:

```
docker-compose.yml X
practicaDocker > docker-compose.yml
1  services:
2    db:
3      image: mysql:5.7
4      environment:
5        MYSQL_ROOT_PASSWORD: examplepass
6        MYSQL_DATABASE: exampledb
7        MYSQL_USER: exampleuser
8        MYSQL_PASSWORD: examplepass
9      restart: always
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2
• alumno@alumno:~/practicaDocker$ sudo docker compose up -d
[+] Running 12/12
✓ db Pulled
✓ 20e4dcae4c69 Pull complete
✓ 1c56c3d4ce74 Pull complete
✓ e9f03a1c24ce Pull complete
✓ 68c3898c2015 Pull complete
✓ 6b95a940e7b6 Pull complete
✓ 90986bb8de6e Pull complete
✓ ae71319cb779 Pull complete
✓ ffc89e9dfd88 Pull complete
✓ 43d05e938198 Pull complete
✓ 064b2d298fba Pull complete
✓ df9a4d85569b Pull complete
[+] Running 2/2
✓ Network practicaDocker_default Created
✓ Container practicaDocker-db-1 Started
o alumno@alumno:~/practicaDocker$
```

Compruebo que se crea una “network” por defecto ya que no la he indicado y se inicia el contenedor sin problema. El siguiente paso es indicar cual es la imagen que se usa para la práctica. En este caso dejo la que está ya que he leído en documentaciones que es la que menos problemas da (la versión 5.7 de mysql):

```
image: mysql:5.7
```

Se le da nombre al contenedor con la directiva container_name:

```
container_name: mysql_db
```

Indico que se reinicie a menos que se detenga manualmente:

```
restart: always
```

Las variables de entorno sirven para configurar los servicios de manera flexible sin modificar en el mismo archivo de configuración y permiten definirse en otro fichero externo .env

Configuro las de mi fichero así:

```
environment:
  MYSQL_ROOT_PASSWORD: bitnami
  MYSQL_DATABASE: bitnami_wordpress
  MYSQL_USER: bn_wordpress
  MYSQL_PASSWORD: bitnami
```

El usuario raíz es MYSQL_ROOT_PASSWORD y la base de datos es MYSQL_DATABASE. El usuario y contraseña para wordpress serán MYSQL_USER y MYSQL_PASSWORD.

Se monta en un volumen para la persistencia de los datos del servicio:

```
volumes:
  - db_data:/var/lib/mysql
```

La red a la que se conectará es a la de backend-network ya que este servicio no será accesible por el host:

```
networks:
  - backend-network
```

Lo siguiente es la **configuración del servicio phpMyAdmin**, para ello indico lo siguiente:

Se usa la imagen phpmyadmin:

```
image: phpmyadmin/phpmyadmin
```

El nombre del contenedor es phpmyadmin

```
container_name: phpmyadmin
```

El reinicio es automático:

```
restart: always
```

Le indico que debe de depender del servicio de base de datos (en este caso mySQL):

```
depends_on:
  - db
```

El puerto que usará es el 8080:

```
ports:
  - "8080:80"
```

Esta asignación de puertos en compose indica cómo se conectará el host al contenedor de Docker. “8080:80” indica que el puerto 8080 es el puerto en el host (mi ordenador con Windows) y “80” es el puerto del contenedor, por lo que tendré que acceder a la IP del servidor con el puerto 8080 para ir a phpMyAdmin.

La variable de entorno define el servidor de base de datos al que se conectará phpMyAdmin, en este caso es “db” que es el nombre que le he dado al servicio de base de datos:

```
environment:
    PMA_HOST: db
```

Las redes a las que se conecta son la de frontend y la de backend:

```
networks:
    - frontend-network
    - backend-network
```

El último servicio en configurar es **Wordpress:**

Le indico que utilice un fichero para las variables de entorno, con ello se mejora la flexibilidad y organización del código. También sirve para encapsular credenciales para que no estén directamente en el fichero docker-compose.yml y facilita los cambios sin modificar el archivo principal:

```
env_file: ".env"
```

La imagen a usar es la última versión de bitnami/wordpress:

```
image: bitnami/wordpress:latest
```

El nombre del contenedor es:

```
container_name: wordpress_site
```

El reinicio se configura como always, esto significa que docker reiniciará el contenedor si hay un fallo o error inesperado y cuando docker se reinicie, también lo hará el contenedor:

```
restart: always
```

La siguiente directiva es la de indicar que dependa de otros servicios, en este caso que dependa de la base de datos mysql. Con esto Wordpress esperará a que mysql esté “saludable” antes de iniciarse, por lo que depende completamente de que el contenedor de la base de datos se haya iniciado correctamente:

```
depends_on:
    db:
        condition: service_healthy
```

La directiva `depends_on` controla el orden de arranque de los servicios e indica quién tiene dependencia de quien. He consultado este gist de github donde se comparte una configuración del fichero docker-compose.yml:

<https://gist.github.com/bradtraversy/faa8de544c62eef3f31de406982f1d42>

Nota: para que la directiva `condition: service_healthy` funcione se tiene que configurar otra directiva en mysql llamada "healthcheck" que define un mecanismo de verificación de estado en Docker Compose para comprobar si el servicio MySQL está funcionando correctamente. Se usa para comprobar si un sistema está funcionando adecuadamente al ejecutar un comando específico y comprobando la respuesta. He consultado este issue de github donde se indica cuál puede ser una configuración para mysql:

<https://github.com/docker-library/mysql/issues/930>

Los puertos que tiene configurados el servicio de Wordpress son:

```
ports:
  - "80:8080"
  - "443:8080"

# "puerto del host : puerto del contenedor"
```

Estos puertos indican que cuando visite la dirección del servidor con el puerto 80, Docker enviará el tráfico al puerto 8080 del contenedor y si visito el servidor con el protocolo seguro https, este tráfico también se redirigirá al puerto 8080 del contenedor.

Las variables de entorno son:

```
environment:
  WORDPRESS_DATABASE_HOST: ${WORDPRESS_DATABASE_HOST}
  WORDPRESS_DATABASE_USER: ${WORDPRESS_DATABASE_USER}
  WORDPRESS_DATABASE_PASSWORD: ${WORDPRESS_DATABASE_PASSWORD}
  WORDPRESS_DATABASE_NAME: ${WORDPRESS_DATABASE_NAME}
  WORDPRESS_BLOG_NAME: ${WORDPRESS_BLOG_NAME}
  WORDPRESS_USERNAME: ${WORDPRESS_USERNAME}
  WORDPRESS_PASSWORD: ${WORDPRESS_PASSWORD}
  WORDPRESS_EMAIL: ${WORDPRESS_EMAIL}
```

Estas variables son llamadas dinámicamente a las del fichero .env que configuraré.

EL volumen para este contenedor está configurado con la siguiente directiva:

```
volumes:
  - wordpress_data:/bitnami/wordpress
```

Y sus redes son las siguientes:

```
networks:
  - frontend-network
  - backend-network
```

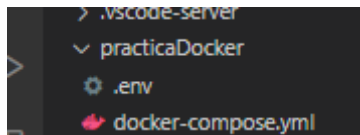
En este caso y al igual que con phpMyAdmin, el contenedor de Wordpress debe conectarse a estas dos redes para acceder a la base de datos por medio de la red backend y a la red frontend para que se comuniquen con el host.

Por último en el fichero de docker-compose.yml se deben indicar cuáles son las redes a usar por los contenedores y cuáles con los volúmenes para la persistencia de datos:

```
networks:
  frontend-network:
  backend-network:
volumes:
  db_data:
  wordpress_data:
```

Fichero de variables de entorno

Creo un fichero .env y lo guardo dentro del mismo directorio que el fichero de configuración:



Tal como se indica en la documentación de DockerDocs, incluyo el atributo env_file en la configuración de Wordpress para indicar el nombre del fichero con las variables de entorno:

<https://docs.docker.com/compose/how-tos/environment-variables/set-environment-variables/>

Este es el fichero .env configurado:

```
WORDPRESS_DATABASE_HOST=db
WORDPRESS_DATABASE_USER=bn_wordpress
WORDPRESS_DATABASE_PASSWORD=bitnami
WORDPRESS_DATABASE_NAME=bitnami_wordpress
WORDPRESS_BLOG_NAME="User's blog"
WORDPRESS_USERNAME=user
WORDPRESS_PASSWORD=bitnami
WORDPRESS_EMAIL=user@example.com
```

Como he indicado antes, le añado la directiva de healthcheck a la configuración de MySQL y a Wordpress le digo que dependa de la base de datos con “depends_on” y además que tenga la condición de que el servicio esté sano.

Esto lo añado a la configuración de mysql:

```
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "127.0.0.1", "--silent"]
  interval: 5s
  timeout: 3s
  retries: 2
  start_period: 0s
```

El parámetro test indica el comando que se ejecutará para verificar la salud del contenedor: se ejecuta el comando `mysqladmin ping -h 127.0.0.1 --silent` que verifica si MySQL está respondiendo en la dirección de loopback, y silent que sólo devuelve el resultado sin ningún mensaje más. El resto de parámetros son tiempos que se le indica a la orden: interval es para indicar que el chequeo de salud sea ejecutado cada 5 segundos, timeout es para indicar cuándo será dado como fallido el comando si éste no responde, retries es el número de veces que se intentará el comando antes de dar el contenedor como no saludable y start_period indica el tiempo de espera inicial antes de comenzar a realizar las comprobaciones de salud.

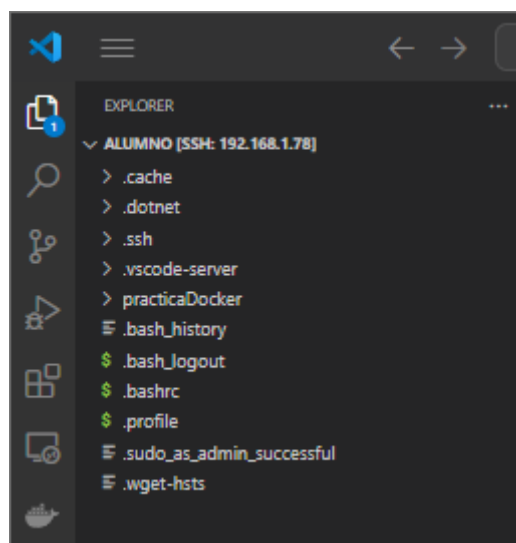
Y para terminar de configurar la directiva healthcheck de MySQL, le añado esto a la de Wordpress, que indica que para que el servicio pueda depender del contenedor de la base de datos, éste debe de ser un servicio “saludable”:

```
depends_on:
  db:
    condition: service_healthy
```

Descripción de las acciones que se han realizado durante la puesta en producción

Para poder configurar el fichero docker-compose.yml me he conectado por SSH al servidor Ubuntu Server, usando la extensión SSH de Microsoft en Visual Studio Code. De esta manera puedo editar los ficheros del servidor directamente desde el editor y es mucho más rápido y sencillo que hacerlo en una máquina virtual usando el editor Nano de Ubuntu.

Una vez conectado por SSH he solicitado acceso a la carpeta del usuario “alumno” para poder ver el sistema de archivos en Visual Studio:



Una vez tenido acceso he creado el directorio practicaDocker y he configurado los dos ficheros, el .yml y el .env como ya he explicado. Para probar si mi configuración es correcta ejecuto los siguientes comandos:

`docker-compose up -d`

Con este comando se inician los servicios definidos en el archivo docker-compose.yml.

```
alumno@alumno:~/practicaDocker$ sudo docker compose up -d
[+] Running 3/3
 ✓ Container mysql_db      Healthy
    8.9s
 ✓ Container wordpress_site Started
    7.4s
 ✓ Container phpmyadmin    Started
    2.0s
```

docker-compose ps

Muestra los contenedores y su estado bajo la configuración de docker-compose

```
* alumno@alumno:~/practicaDocker$ sudo docker-compose ps
[sudo] password for alumno:

```

Name	Command	State	Ports
mysql_db	docker-entrypoint.sh mysqld	Up (healthy)	3306/tcp, 33060/tcp
phpmyadmin	/docker-entrypoint.sh apac ...	Up	0.0.0.0:8080->80/tcp, :::8080->80/tcp
wordpress_site	/opt/bitnami/scripts/wordp ...	Up	0.0.0.0:443->8080/tcp, :::443->8080/tcp, 0.0.0.0:80->8080/tcp, :::80->8080/tcp, 8443/tcp

docker network ls

Muestra las redes de Docker disponibles.

```
* alumno@alumno:~/practicaDocker$ sudo docker network ls

```

NETWORK ID	NAME	DRIVER	SCOPE
d343a2fed87d	bridge	bridge	local
c3087da14a8a	host	host	local
578c5719b426	none	null	local
546869b70cb7	practicadocker_backend-network	bridge	local
36b5d2f1c484	practicadocker_default	bridge	local
2848bf564173	practicadocker_frontend-network	bridge	local

docker volume ls

Muestra los volúmenes de Docker disponibles.

```
* alumno@alumno:~/practicaDocker$ sudo docker volume ls

```

DRIVER	VOLUME NAME
local	0d1a42a91983595e3a68f187bd409d881dca8759c065f72aaf3c29db34e42d9a
local	practicadocker_db_data
local	practicadocker_wordpress_data

Ahora para comprobar si las conexiones son correctas, voy a acceder desde mi navegador en Windows a las direcciones de cada contenedor:

- phpMyAdmin → 192.168.1.78:8080 ya que en el fichero docker-compose.yml tengo para el contenedor esta directiva:

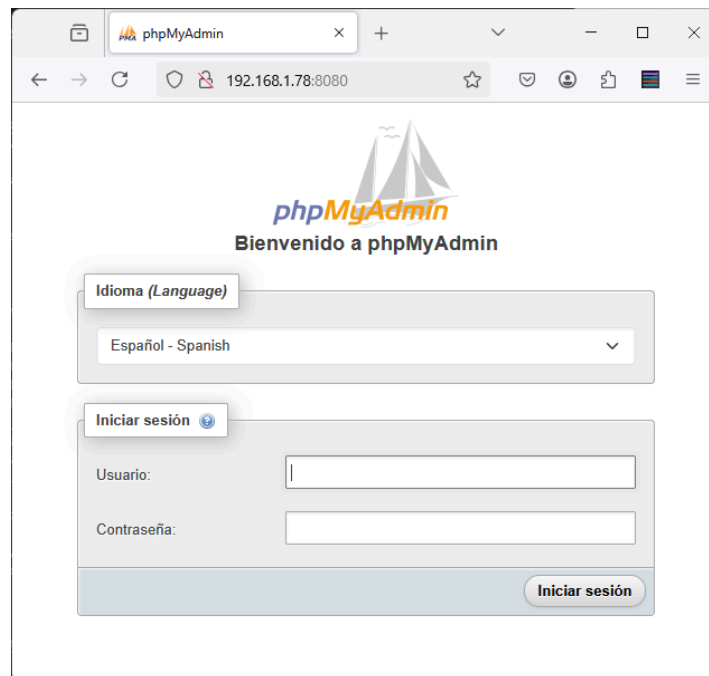
```
ports:
  - "8080:80"
```

- WordPress → 192.168.1.78:80

```
ports:
  - "80:8080"
  - "443:8080"
```

Capturas del acceso a wordpress, phpMyAdmin y el no acceso a la base de datos mysql

Acceso a phpMyAdmin:



phpMyAdmin

Bienvenido a phpMyAdmin

Idioma (Language)

Español - Spanish

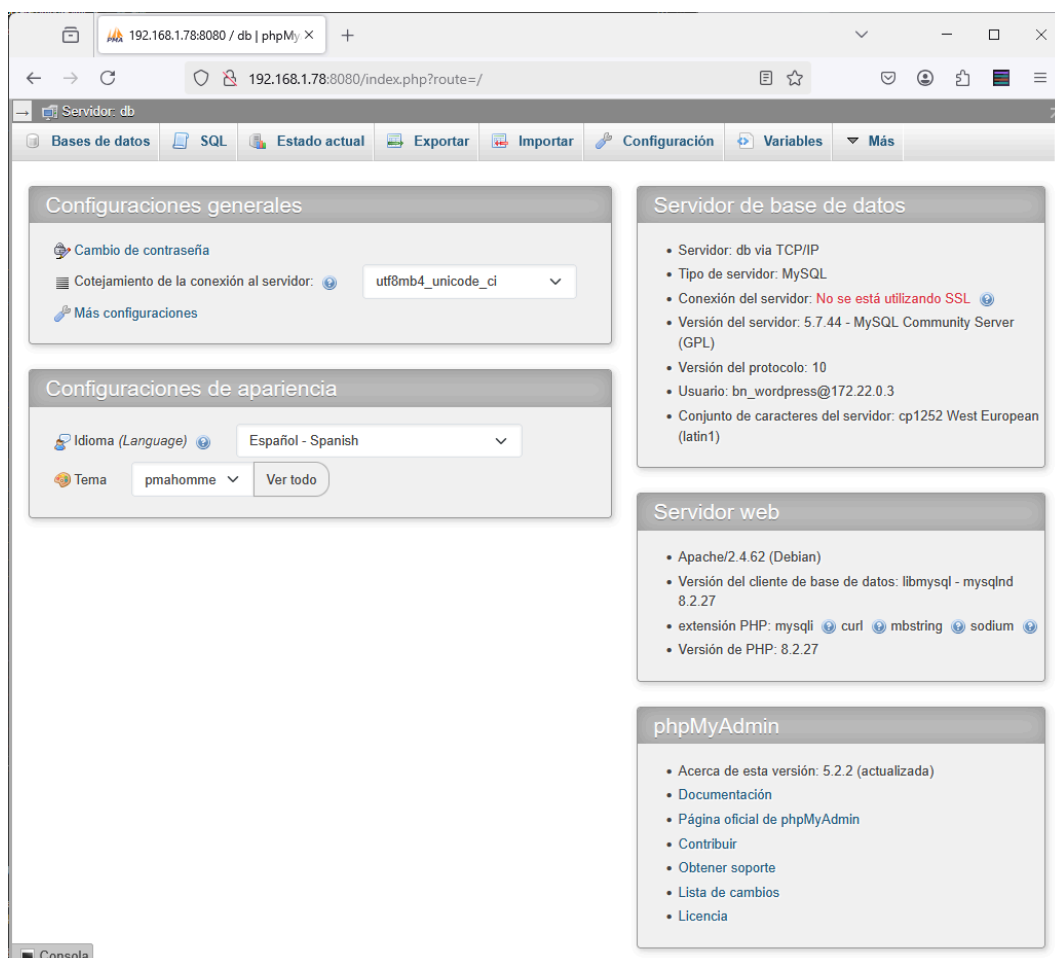
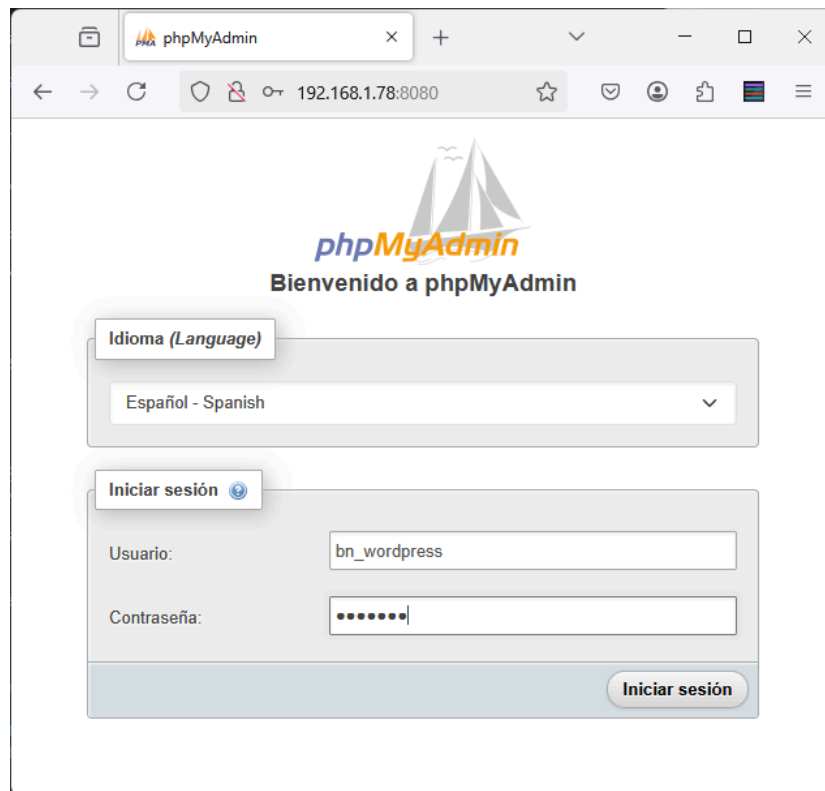
Iniciar sesión

Usuario:

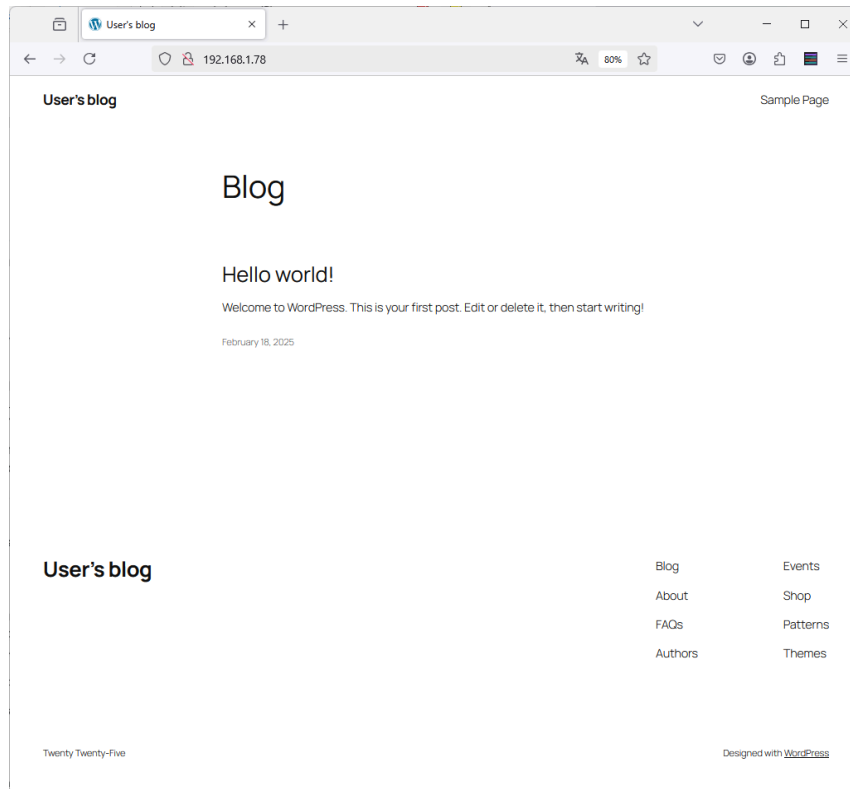
Contraseña:

Iniciar sesión

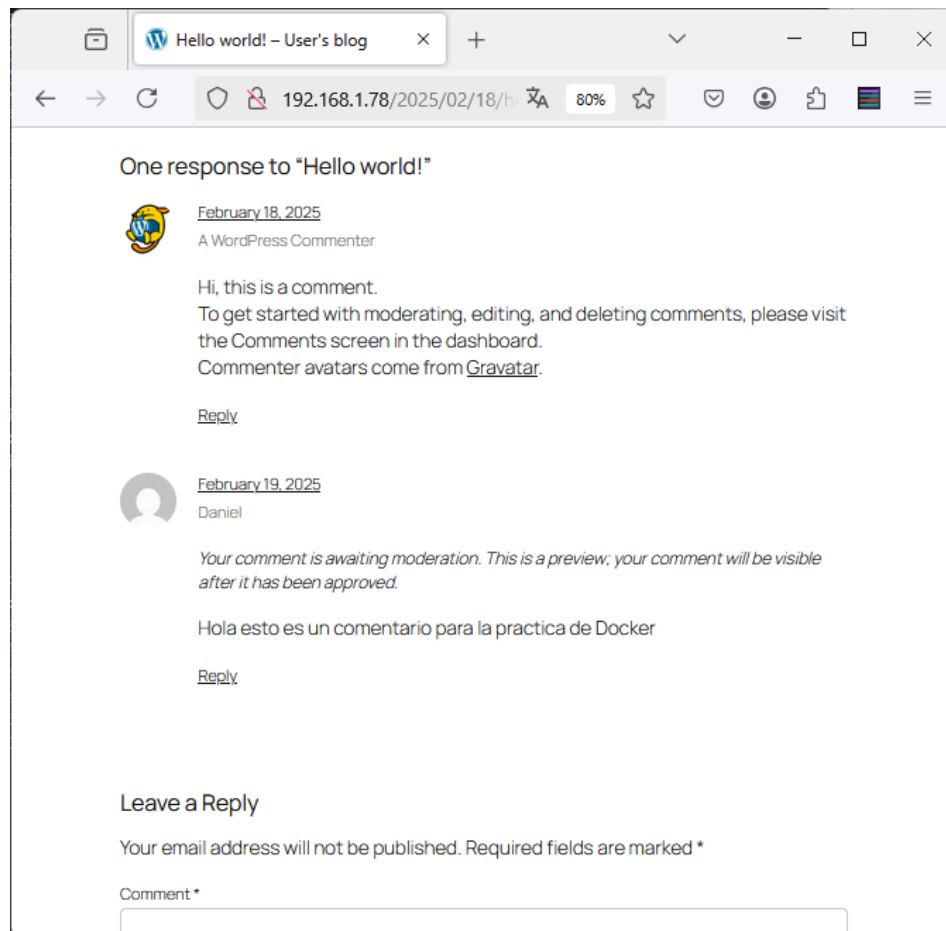
```
MYSQL_USER: bn_wordpress #usuario para phpmyadmin
MYSQL_PASSWORD: bitnami #password para phpmyadmin
```

Acceso a WordPress:



He puesto un comentario en la sección de Hello World para comprobar si funciona:



No acceso a mySQL:

Para probar el acceso a mySQL voy a ingresar un comando para intentar conectar a la base de datos indicando unos parámetros como la dirección del host, el puerto a usar y con qué usuario conectarse. Primero instalo el cliente mySQL:

```
sudo apt update
sudo apt install mysql-client -y
```

Y ahora pruebo a conectar indicando que se intente conectar a localhost, por el puerto 3306, como usuario bn_wordpress y que me pregunte la contraseña:

```
mysql -h localhost -P 3306 -u bn_wordpress -p
```

```
alumno@alumno:~/practicaDocker$ mysql -h 127.0.0.1 -P 3306 -u bn_wordpress -p
Enter password:
ERROR 2003 (HY000): Can't connect to MySQL server on '127.0.0.1:3306' (111)
```

Compruebo cómo están configurados los puertos con “docker ps”:

```
alumno@alumno:~/practicaDocker$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
1e42e73f0ae7	phpmyadmin/phpmyadmin	"/docker-entrypoint..."	59 minutes ago	Up 59 minutes	0.0.0.0:8080->80/tcp
ce6cd941e897	bitnami/wordpress:latest	"/opt/bitnami/script..."	59 minutes ago	Up 59 minutes	8443/tcp, 0.0.0.0:80->80/tcp
bc5a14631ed7	mysql:5.7	"docker-entrypoint.s..."	59 minutes ago	Up 59 minutes (healthy)	3306/tcp, 33060/tcp

Se comprueba que el puerto 3306 de mysql no está expuesto al host, ya que no tiene un mapeo como lo tienen los contenedores de phpmyadmin y WordPress:

```
phpmyadmin 0.0.0.0:8080->80/tcp, :::8080->80/tcp
wordpress 0.0.0.0:80->8080/tcp, 0.0.0.0:443->8080/tcp
```

```
1e42e73f0ae7 phpmyadmin/phpmyadmin "/docker-entrypoint..." 59 minutes ago Up 59 minutes 0.0.0.0:8080->80/tcp, :::8080->80/tcp
ce6cd941e897 bitnami/wordpress:latest "/opt/bitnami/script..." 59 minutes ago Up 59 minutes 8443/tcp, 0.0.0.0:80->8080/tcp, 0.0.0.0:443->8080/tcp, :::80->8080/tcp, :::443->8080/tcp
```

Si el puerto 3306 de MySQL estuviese expuesto aparecería esto:

```
0.0.0.0:3306->3306/tcp
```

y en el fichero de docker-compose.yml estaría configurado así:

```
# ports:
# - "3306:3306" #Si quieres habilitar el acceso a MySQL desde el host
```

Otra prueba de que no hay conexión desde el host hacia la base de datos es que no puedo conectarme desde el navegador indicando el puerto 3306:



Desde phpMyAdmin puedo ingresar con el usuario "bn_wordpress" y su contraseña "bitnami" y veo que tengo conexión con la base de datos mysql, ya que usan la misma red backend configurada en docker-compose.yml:

Servidor de base de datos

- Servidor: db via TCP/IP
- Tipo de servidor: MySQL
- Conexión del servidor: No se está utilizando SSL ⓘ
- Versión del servidor: 5.7.44 - MySQL Community Server (GPL)
- Versión del protocolo: 10
- Usuario: bn_wordpress@172.22.0.3
- Conjunto de caracteres del servidor: cp1252 West European (latin1)

Repositorio GitHub

A continuación indico mi repositorio en GitHub llamado "PracticaDAW_Docker-Compose", donde incluyo un documento técnico con todos los pasos que he realizado en esta práctica, así como los ficheros de configuración docker-compose.yml y el fichero de variables de entorno .env

https://github.com/DanielGodoyGalindo/PracticaDAW_Docker-Compose

Extra: Políticas de reinicio

En el fichero de docker-compose.yml se pueden definir unas políticas de reinicio para indicarle al contenedor que hacer cuando éste se detiene de forma inesperada. Con la directiva `restart` se puede configurar la orden de reinicio del contenedor. Existen varias políticas:

- `no`: que no se reinicie el contenedor automáticamente, es la opción por defecto.
- `always`: siempre reiniciar el contenedor si éste se para, sin importar la razón por la que se detuvo.
- `on-failure`: reiniciar el contenedor sólo si termina con un error que es un código de salida distinto de cero. Se indica así `on-failure[:N]` siendo N un número distinto de cero.
- `unless-stopped`: reiniciar siempre el contenedor a no ser que se pare explícitamente con una orden a docker, o que Docker se pare o se reinicie entonces que el contenedor se reinicie con él.

Para configurar los contenedores para que se reinicien cada vez que se detengan de forma inesperada, he elegido la opción de `restart: unless-stopped`, ya que el contenedor se reiniciará automáticamente si falla o si el sistema se reinicia, y no se reiniciará si lo intento detener con la orden `docker-compose stop`.

```
restart: unless-stopped
```