Ejercicio 1. Volumes

Vamos a trabajar con la persistencia de datos en los contenedores.

1. En la siguiente guía del sitio web oficial de Docker tienes una explicación detallada sobre los volúmenes: https://docs.docker.com/storage/. Explica la diferencia entre los volumes, bind mounts y tmpfs.

Los volumes son creados y gestionados por Docker.

La principal diferencia entre los dos tipos de volúmenes(bind mounts y tmpfs) es el almacenamiento de la información, en el tipo de volumen bind mounts, la información se almacena en el disco duro del host o anfitrión. En cambio, utilizando un volumen tipo tmpfs los datos se guardan en memoria siendo estos volátiles, esta opción se utiliza en los casos donde no se quiera conservar los datos, ya sea por razones de seguridad o para proteger el rendimiento del contenedor.

2. En el siguiente tutorial se explican cuatro formas de gestionar volúmenes con contenedores. Compruébalas.

Creamos un nuevo volumen llamado DataVolume1

```
ivan@debian-ivan:~$ docker volume create --name DataVolume1
DataVolume1
ivan@debian-ivan:~$
```

Usamos el nuevo volumen creando un nuevo contenedor a partir de la imagen de Ubuntu.

```
ivan@debian-ivan:~$ docker run -ti --rm -v DataVolume1:/datavolume1 ubuntu
root@16b04990bd6a:/#
```

Una vez dentro del contenedor creamos un nuevo archivo que se almacenará en el volumen. Seguido este paso salimos del contenedor, esto provocará la eliminación del mismo, pero el archivo ya estará en el volumen.

```
root@16b04990bd6a:/# echo "Example1" > /datavolume1/Example1.txt
root@16b04990bd6a:/# exit
exit
ivan@debian-ivan:~$ docker volume inspect DataVolume1
[
    {
        "CreatedAt": "2019-02-20T17:44:27Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/DataVolume1/_data",
        "Name": "DataVolume1",
        "Options": {},
        "Scope": "local"
    }
]
ivan@debian-ivan:~$
```

Visualizamos que el volumen se encuentra en el sistema

Creamos un nuevo contenedor, asignando él volumen. Y dentro del contenedor comprobamos como existe el archivo.

```
ivan@debian-ivan:~$ docker run -ti --rm -v DataVolume1:/datavolume1 ubuntu
root@328369c0d117:/# ls
bin   datavolume1  etc   lib    media  opt   root  sbin  sys  usr
boot  dev          home  lib64  mnt    proc  run   srv   tmp  var
root@328369c0d117:/# cat datavolume1/Example1.txt
Example1
root@328369c0d117:/#
```

Paso 2 Crear un volumen que persiste cuando se quita el contenedor.

Creamos un nuevo contenedor, asignándole un nuevo volumen

```
ivan@debian-ivan:~$ docker run -ti --name=Container2 -v DataVolume2:/datavolume2
 ubuntu
root@1fac76456196:/#
```

A continuación, creamos un nuevo archivo dentro del mismo.

```
root@1fac76456196:/# echo "Example2" > /datavolume2/Example2.txt
root@1fac76456196:/# cat /datavolume2/Example2.txt
Example2
root@1fac76456196:/#
```

Procederemos a reiniciar el contenedor y comprobaremos como el volumen se monta automáticamente en el contenedor.

```
ivan@debian-ivan:~$ docker start -ai Container2
root@1fac76456196:/# cat /datavolume2/Example2.txt
Example2
root@1fac76456196:/#
```

Para eliminar un volumen se utiliza la opción rm , aunque si el volumen esta actualmente referenciado a un contenedor Docker no nos permite eliminarlo, deberemos de borrar él contenedor(ID).

```
ivan@debian-ivan:~$ docker rm 1fac764561960010dc1127223b6bad9f71ae65b99a05727f79
474b1771203765
1fac764561960010dc1127223b6bad9f71ae65b99a05727f79474b1771203765
ivan@debian-ivan:~$ docker volume ls
DRIVER              VOLUME NAME
local               DataVolume1
local               DataVolume2
ivan@debian-ivan:~$
```

Borrando el contenedor, Docker no borrara el volumen

Por último, procedemos a borrar el contenedor

```
ivan@debian-ivan:~$ docker volume rm DataVolume2
DataVolume2
ivan@debian-ivan:~$ docker volume ls
DRIVER              VOLUME NAME
local               DataVolume1
ivan@debian-ivan:~$
```

Paso 3 Crear un volumen desde un directorio existente con datos.

```
ivan@debian-ivan:~$ docker volume ls
DRIVER              VOLUME NAME
local               DataVolume1
local               DataVolume3
```

```
ivan@debian-ivan:~$ docker run --rm -v DataVolume3:/datavolume3 ubuntu ls d
atavolume3
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
ivan@debian-ivan:~$
```

Paso 4 Compartir datos entre múltiples contenedores Docker.

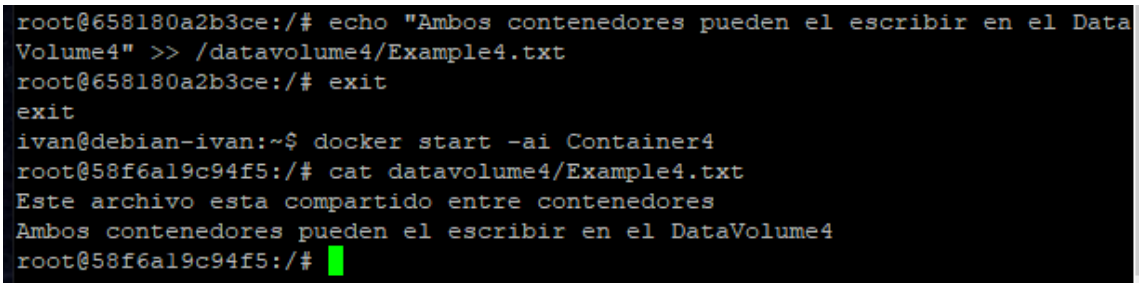
Creamos un nuevo contenedor con un nuevo volumen, seguidamente creamos un archivo y salimos.

```
ivan@debian-ivan:~$ docker run -ti --name=Container4 -v DataVolume4 -v Data
Volume4:/datavolume4 ubuntu
root@58f6a19c94f5:/# echo "Este archivo esta compartido entre contenedores"
 > /datavolume4/Example4.txt
root@58f6a19c94f5:/# exit
exit
ivan@debian-ivan:~$
```

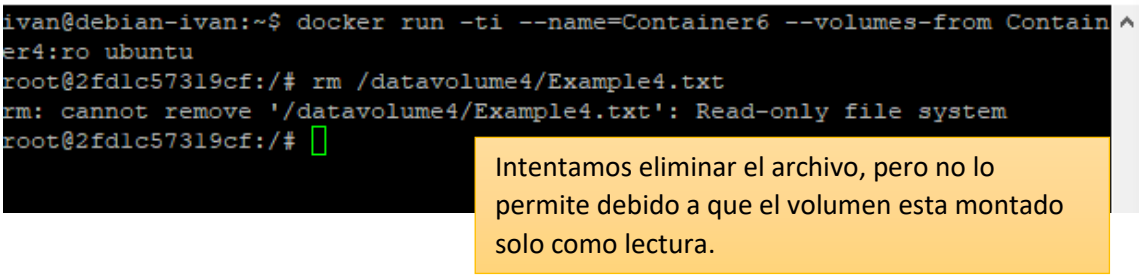
Creamos un nuevo contenedor asignándole el volumen del contenedor 4, y comprobamos que el archivo existe.

```
ivan@debian-ivan:~$ docker run -ti --name=Container5 --volumes-from Contain
er4 ubuntu
root@658180a2b3ce:/# cat datavolume4/Example4.txt
Este archivo esta compartido entre contenedores
root@658180a2b3ce:/#
```

Editamos el archivo creado en el contenedor4 y comprobamos conectándonos al contenedor4 como podemos ver el archivo previamente modificado desde el contenedor 5.



Por último, podemos hacer que un volumen montado en un contenedor sea de solo lectura con la opción ":ro"



3. **Explica qué has hecho en cada una de las cuatro aproximaciones.**

   En el paso 1 creamos un volumen independiente, y se lo montamos a un contenedor, comprobando así la persistencia de datos.

   En el paso 2 Creamos un volumen que tiene persistencia, cuando el contenedor es eliminado.

   En el paso 3 creamos un volumen partiendo de un directorio con datos.

   En el paso 4 procedimos a compartir un volumen entre múltiples contenedores, probando como se realizan las modificaciones sobre los archivos, tanto en un contenedor como en otro, por último también aprendimos como montar un volumen de solo lectura sobre un contenedor.

4. **Indica una ventaja y un inconveniente de cada una.**

Ventaja donde nos permite que un volumen sea compartido por varios contenedores.

5. **Comprueba también que puedes compartir volúmenes entre contenedores en modo sólo lectura (ro).**

   Apartado comprobado anteriormente.

Ejercicio 2. Bind mounts

En este otro tutorial se comparte una carpeta entre el host local y el contenedor. En concreto se comparten los logs de un Nginx ejecutándose en un contenedor. Haz el tutorial y explica las opciones del comando que has utilizado.

```
ivan@debian-ivan:~$ docker run --name=nginx -d -v ~/nginxlogs:/var/log/ngin
x -p 5000:80 nginx
88679da304626d187c201fcf11c382a81689fdd4ba9a0943c31f059119c4bc25
ivan@debian-ivan:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
          STATUS              PORTS                    NAMES
88679da30462        nginx               "nginx -g 'daemon of…"   About a mi
nute ago    Up About a minute   0.0.0.0:5000->80/tcp     nginx
ivan@debian-ivan:~$
```

Las opciones utilizadas para crear el contenedor han sido:

- **--name = nginx** Permite establecer un nombre al contenedor.
- **-d** Separa el proceso y lo ejecuta en segundo plano.
- -v ~/nginxlogs:/var/log/nginx configura un volumen bindmount que vincula el /Var/log/nginx desde el directorio del contenedor , al ~/nginxlogs directorio de la máquina host.
- -p 5000:80 Establece el puerto 5000 del host al puerto que escucha por defecto nginx(80)
- Nginx especifica que el contenedor debe de generarse desde una imagen de nginx

Ahora tenemos una imagen de nginx corriendo en un contenedor de Docker, accedemos desde el navegador al puerto 5000 que mapeara al 80 del contenedor.

Podemos también mirar los logs de nginx desde el host

```
ivan@debian-ivan:~$ cat ~/nginxlogs/access.log
192.168.0.147 - - [20/Feb/2019:18:55:09 +0000] "GET / HTTP/1.1" 200 612 "-"
 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/72.0.3626.109 Safari/537.36" "-"
192.168.0.147 - - [20/Feb/2019:18:55:09 +0000] "GET /favicon.ico HTTP/1.1"
404 555 "http://192.168.0.155:5000/" "Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.109 Safari/537
.36" "-"
ivan@debian-ivan:~$
```

Ejercicio 3. Networking

1. Averigua cuál es la diferencia entre un puerto expuesto y un puerto público en un contenedor.

Un puerto público es puerto que nos permite acceder desde fuera del contenedor, por ejemplo 5000:80, y un puerto expuesto es cuando permitimos conexiones entrantes a los contendores de Docker.

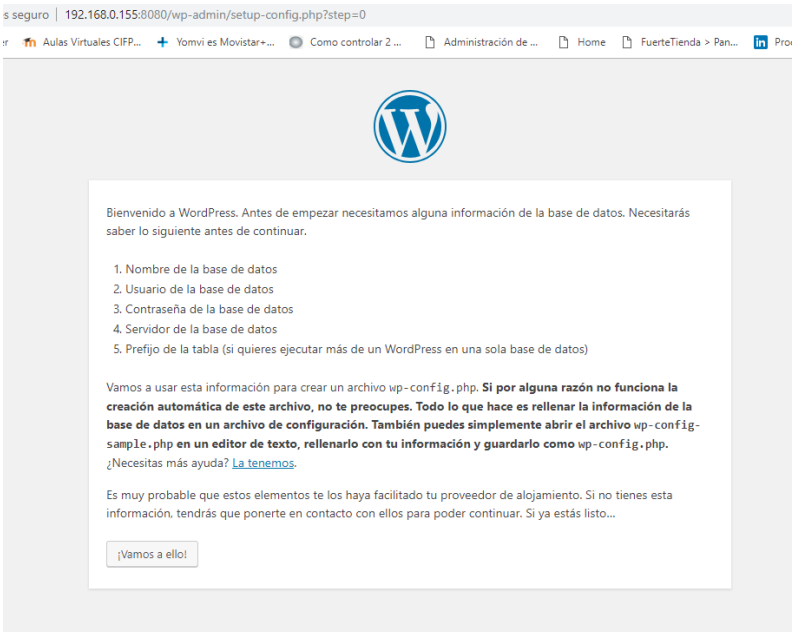
2. ¿Cuál es la subred que utiliza docker en tu sistema? **172.17.0.0**
3. Ejecuta el siguiente comando: docker run --name mydb -e MYSQL_ROOT_PASSWORD=123 -d mysql. Explica detalladamente qué es lo que hace.

   Arranca un contenedor estableciendo un nombre al contenedor "mydb" y lo ejecuta en background "-d", establece también una contraseña para root, y el contenedor se crea a partir de una imagen de "mysql".

4. Ejecuta el siguiente comando: docker run -d -p 8080:80 --name mywordpress --link mydb wordpress. Explica detalladamente qué es lo que hace.

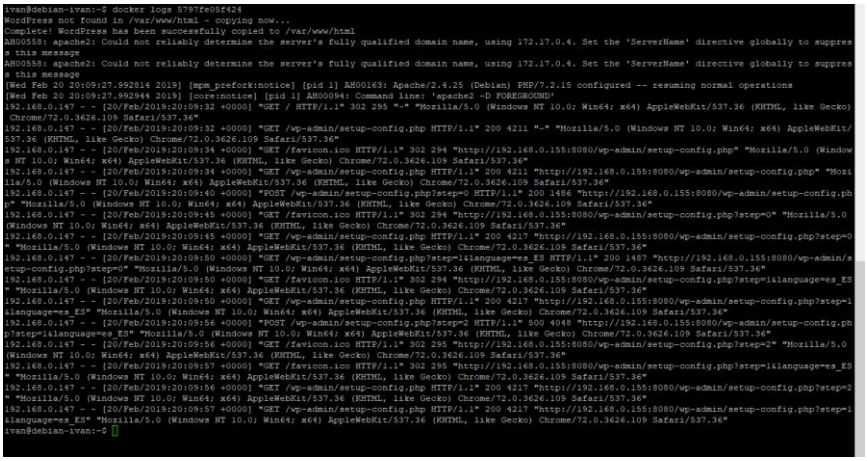
   Ejecuta en segundo plano el contenedor "-d", establece un mapeo desde el 8080 al puerto 80, establece un nombre al contenedor "—name" (mywordpress), --link permite que los contenedores transfieran de manera segura información sobre un contenedor a otro, para que wordpress acceda a mydb.

5. Conéctate a tu máquina virtual con un navegador en el puerto 8080 y comprueba que puedes ver la pantalla de instalación de wordpress.



6. ¿Cómo puedes ver los logs de un contenedor?

   Mediante Docker logs y el id del contendor



Puedes consultar la documentación de la imagen de wordpress en hubdock: https://hub.docker.com/_/wordpress/

Puedes consultar la documentación de la imagen de mysql en hubdock: https://hub.docker.com/_/mysql/