Fuente: https://raul-profesor.github.io/Despliegue/data/

# Sumario



## Persistiendo datos

Por defecto ya hemos indicado que un contenedor está aislado de todo. Hemos visto como podemos conectar el contenedor a un puerto de red para poder acceder a él. Eso incluye al sistema de archivos que contiene. De tal manera que si se elimina el contenedor, se eliminan también sus archivos.

Si queremos **almacenar datos** (una web, una base de datos, etc.) dentro de un contenedor necesitamos una manera de almacenarlos sin perderlos. Docker ofrece tres maneras:

- A través de volúmenes, que son objetos de Docker como las imágenes y los contenedores.
- Montando un directorio de la máquina anfitrión dentro del contenedor.
- Almacenándolo en la memoria del sistema (aunque también se perderían al reiniciar el servidor).

Lo normal es usar volúmenes, pero habrá ocasiones en que es preferible montar directamente un directorio de nuestro espacio de trabajo. Por ejemplo, para guardar los datos de una base de datos usaremos volúmenes, pero para guardar el código de una aplicación o de una página web montaremos el directorio. Así tanto nuestro entorno de desarrollo como el contenedor tendrán acceso a los archivos del código fuente.

Los volúmenes, al contrario que los directorios montados, no deben accederse desde la máquina anfitrión.

### Crear un volumen

Como necesitamos crear una base de datos para nuestro blog con WordPress vamos a crear un volumen donde guardar la información:

```
$ docker volume create wordpress-db
wordpress-db
```

### Listar volúmenes

Con docker volume ls podemos visualizar todos los volumenes disponibles.

```
$ docker volume ls
DRIVER              VOLUME NAME
local               wordpress-db
```

Visualizar volúmenes

Los volúmenes se crean en un directorio del sistema y no es recomendable acceder, no mientras haya un contenedor usándolo. En cualquier caso, si queremos ver los metadatos de un volumen podemos usar:

```
$ docker volume inspect wordpress-db
[
    {
        "CreatedAt": "yyyy-mm-ddThh:ii:ss+Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/wordpress-db/_data",
        "Name": "wordpress-db",
        "Options": {},
        "Scope": "local"
    }
]
```

Borrar volúmenes

Como todos los objetos de Docker, los volúmenes también pueden ser borrados, pero solo si no están en uso. Mucha precaución al borrar los volúmenes, porque perderíamos todos los datos que contenga.

Para borrar un contenedor usaremos:

```
$ docker volume rm nombre_contenedor
```

## Levantar un WordPress con Docker Compose

El cliente de Docker es engorroso para crear contenedores, así como para crear el resto de objetos y vincularlos entre sí. Para automatizar la creación, inicio y parada de un contenedor o un conjunto de ellos, Docker proporciona una herramiento llamada Docker Compose.

Compose es una herramienta para definir y ejecutar aplicaciones multi-contenedor. Con un solo comando podremos crear e iniciar todos los servicios que necesitamos para nuestra aplicación.

Los casos de uso más habituales para docker-compose son:

- Entornos de desarrollo
- Entornos de testeo automáticos (integración contínua)
- Despliegue en host individuales (no clusters)

Compose tiene comandos para manejar todo el ciclo de vida de nuestra aplicación:

- Iniciar, detener y rehacer servicios.
- Ver el estado de los servicios.
- Visualizar los logs.
- Ejecutar un comando en un servicio.

Creación de contenedores automatizada

En el directorio ~/Sites/wordpress (si no lo tenemos lo creamos) vamos a crear un fichero llamado *docker-compose.yaml* con el siguiente contenido:

```
version: '3'
services:
  db:
    image: mariadb:10.3.9
    volumes:
      - data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=manager
      - MYSQL_PASSWORD=secret
```

```
  web:
    image: wordpress:4.9.8
    depends_on:
      - db
    volumes:
      - ./target:/var/www/html
    environment:
      - WORDPRESS_DB_USER=manager
      - WORDPRESS_DB_PASSWORD=secret
      - WORDPRESS_DB_HOST=db
    ports:
      - 8080:80
volumes:
  data:
```

Los ficheros de Compose están divididos en tres secciones: services, volumes y networks; y deben indicar un número de versión. Nos permite realizar practicamente lo mismo que podemos hacer con el cliente de docker, pero de forma automática. (No vamos a utilizar la sección networks).

Iniciar servicios

Vamos a ejecutar esta aplicación y luego procederemos a explicarla. Arranca la aplicación con Compose:

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

El parámetro -d es similar al que hemos visto en docker run: nos permite levantar los servicios en segundo plano.

Cuando arrancamos la aplicación, Compose nos informa de los servicios que ha ido levantando:

```
Creating network "wordpress_default" with the default driver
Creating volume "wordpress_data" with local driver
Creating wordpress_db_1 ...
Creating wordpress_db_1 ... done
Creating wordpress_web_1 ...
Creating wordpress_web_1 ... done
```

Veamos los contenedores activos:

```
$ docker container ls
CONTAINER ID   IMAGE                 COMMAND         CREATED             STATUS             PORTS
NAMES
a07b5d4d3982   wordpress:4.9.8   "docker.s…"   10 seconds ago   Up 8 seconds    0.0.0.0:8080->80/tcp
wordpress_web_1
d9204884cec5    mariadb:10.3.9      "docker.s…"    11  seconds  ago    Up  10  seconds    3306/tcp
wordpress_db_1
```

También podemos ver los contenedores con Compose:

```
$ docker-compose ps
    Name                   Command               State          Ports
-------------------------------------------------------------------------------
wordpress_db_1    docker-entrypoint.sh mysqld      Up      3306/tcp
wordpress_web_1   docker-entrypoint.sh apach ...   Up      0.0.0.0:8080->80/tcp
```

Lo que tenemos que tener en cuenta es lo siguiente:

- *docker-compose ps* solo muestra información de los servicios que se define en docker-compose.yaml, mientras que docker muestra todos.
- Cuando creamos contenedores con docker sin indicar un nombre, por defecto asigna uno aleatorio; mientras que en Compose el prefijo es el nombre del directorio y el sufijo el nombre del servicio: wordpress_db_1. El número indica el número de instancia. Es posible levantar más de una instancia de un mismo servicio.

Si accedemos a la dirección http://localhost:8080/ veremos de nuevo la instalación de WordPress.

Detener servicios

Podemos detener servicios con:

```
$ docker-compose stop
```

Borrar servicios

Podemos borrar servicios con:

```
$ docker-compose down
```

Esto borra los contenedores, pero no los volúmenes. Así que si hemos creado bien la aplicación nuestros datos están a salvo.

Si queremos borrar también los volúmenes:

```
$ docker-compose down -v
```

Estructura de la configuración

Veamos la configuración por partes:

```
version: '3'
```

Compose se actualiza a menudo, con lo que el archivo de configuración va adquiriendo nuevas funcionalidades. La versión '3' (importante poner comillas) es la última y para conocer todas sus características mira la página de referencia de la versión 3 de Compose.

```
volumes:
  data:
```

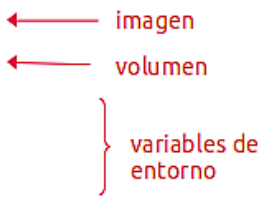
Ya hemos indicado que es importante guardar los datos volátiles de las aplicaciones en volúmenes. En este caso hemos creado un volumen llamado data. Recordemos que Compose siempre añade como prefijo el nombre del directorio, con lo que el nombre real del volumen es wordpress_data. Podemos comprobarlo con el cliente de docker como hicimos en el capítulo de volúmenes:

```
$ docker volume ls
DRIVER              VOLUME NAME
local               wordpress_data
```

Vamos a la sección de servicios, que son los contenedores que precisa o componen nuestra aplicación.

Primero la base de datos mariadb que necesita Wordpress para funcionar:

```
services:
    db:
        image: mariadb:10.3.9
        volumes:
            - data:/var/lib/mysql
        environment:
            - MYSQL_ROOT_PASSWORD=secret
            - MYSQL_DATABASE=wordpress
            - MYSQL_USER=manager
            - MYSQL_PASSWORD=secret
```




Después de abrir la parte de servicios, el primer nivel indica el nombre del servicio db, que genera el contenedor wordpress_db con el servidor mariadb.

Lo anterior es equivalente, excepto por el nombre, a ejecutarlo por línea de comando, solo que esta vez está predefinido en un fichero de configuración que utilizará esta configuración para lanzar el comando con todos estos parámetros:

```
$ docker run -d --name wordpress-db \
        --mount source=wordpress-db,target=/var/lib/mysql \
        -e MYSQL_ROOT_PASSWORD=secret \
        -e MYSQL_DATABASE=wordpress \
        -e MYSQL_USER=manager \
        -e MYSQL_PASSWORD=secret mariadb:10.3.9
```

Y después la configuración para el programa WordPress:

```
services:
    web:
        image: wordpress:4.9.8
        depends_on:
            - db
        volumes:
            - ./target:/var/www/html
        environment:
            - WORDPRESS_DB_USER=manager
            - WORDPRESS_DB_PASSWORD=secret
            - WORDPRESS_DB_HOST=db
        ports:
            - 8080:80
```

En este caso equivaldría al comando de consola:

```
$ docker run -d --name wordpress \
    --link wordpress-db:mysql \
    --mount type=bind,source="$(pwd)"/target,target=/var/www/html \
    -e WORDPRESS_DB_USER=manager \
    -e WORDPRESS_DB_PASSWORD=secret \
    -p 8080:80 \
    wordpress:4.9.8
```

Los parámetros en docker y en composer se relacionan de la siguiente manera:

| parámetro Docker | parámetro Composer |
|---|---|
| --link | depends_on |
| --mount | volumes |
| -e | environment |
| -p, --publish | ports |
| | image |

Si reiniciamos el ordenador, los contenedores estarán detenidos (stop), podremos reiniciarlos con docker start o docker-compose start. Este es el comportamiento predeterminado que nos interesa en un entorno de desarrollo.

Sin embargo, en otros entornos, o para casos concretos, igual queremos que un contenedor tenga el mismo estado en el que estaba antes de reiniciar la máquina (iniciado o parado).

Para eso usaremos el parámetro restart. En el caso de la base de datos de nuestro ejemplo, la configuración quedaría como:

```
services:
  db:
    image: mariadb:10.3.9
    restart: unless-stopped
    volumes:
      - data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=manager
      - MYSQL_PASSWORD=secret
```

El equivalente en la consola sería:

```
$ docker run -d --name wordpress-db \
    --restart unless-stopped
    --mount source=wordpress-db,target=/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=secret \
    -e MYSQL_DATABASE=wordpress \
    -e MYSQL_USER=manager \
    -e MYSQL_PASSWORD=secret mariadb:10.3.9
```

Otros valores para “restart” son: no (por defecto), always y on-failure.

### Portainer

Portainer es una gestor de contenedores a través de una interfaz web. Para usarlo creamos un directorio donde guardar nuestro docker-compose.yaml.

```
mkdir -p ~/Sites/portainer
cd ~/Sites/portainer
```

Guardamos el siguiente fichero como *docker-compose.yaml* en nuestro directorio:

```
version: '2'

services:
    portainer:
    image: portainer/portainer
    command: -H unix:///var/run/docker.sock
    volumes:
        - /var/run/docker.sock:/var/run/docker.sock
        - portainer_data:/data
    ports:
        - 127.0.0.1:9000:9000
volumes:
    portainer_data:
```

Y ejecutamos el contenedor:

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

### Limpieza

Para borrar objetos que no están en uso:

```
$ docker system prune
```

Para borrar volúmenes que no están asociados a ningún contenedor:

```
$ docker volume rm $(docker volume ls -q -f "dangling=true")
```

Para borrar contenedores que han terminado su ejecución:

```
$ docker rm $(docker ps -q -f "status=exited")
```

Para borrar imágenes que no están etiquetadas:

```
$ docker rmi $(docker images -q -f "dangling=true")
```