

Introducción a Docker

# UD 06. Docker Compose

---



Fons Social Europeu

L'FSE inverteix en el teu futur

UNIÓ EUROPEA

Autor: Sergi García Barea

Actualizado Abril 2021

## Licencia



**Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA):** No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

<b>Introducción</b>	<b>3</b>
<b>Docker Compose</b>	<b>3</b>
<b>¿Qué es Docker Compose?</b>	<b>3</b>
<b>Instalando Docker Compose</b>	<b>3</b>
<b>Formato YAML y Fichero "docker-compose.yml"</b>	<b>4</b>
<b>¿Qué es YAML?</b>	<b>4</b>
<b>¿Qué es el fichero "docker-compose.yml"?</b>	<b>4</b>
<b>¿Qué información podemos especificar en nuestro fichero "docker-compose.yml"?</b>	<b>5</b>
<b>Entendiendo un fichero "docker-compose.yml" con ejemplo Wordpress</b>	<b>6</b>
<b>Palabra clave "version"</b>	<b>6</b>
<b>Palabra clave "services" y contenedor "db"</b>	<b>7</b>
<b>Palabra clave "services" y contenedor "wordpress"</b>	<b>8</b>
<b>Palabra clave "volumes" fuera de las plantillas de contenedores</b>	<b>9</b>
<b>Lanzando nuestro ejemplo Wordpress "docker-compose.yml"</b>	<b>9</b>
<b>Comandos de "Docker Compose"</b>	<b>10</b>
<b>Comando "up"</b>	<b>10</b>
<b>Comando "down"</b>	<b>10</b>
<b>Comando "build"</b>	<b>11</b>
<b>Comando "pause"/"unpause"</b>	<b>11</b>
<b>Comando "start"/"stop"</b>	<b>11</b>
<b>Comando "ps"</b>	<b>11</b>
<b>Comando "exec"</b>	<b>11</b>
<b>Bibliografía</b>	<b>11</b>

## UD06. DOCKER COMPOSE

### 1. INTRODUCCIÓN

En esta unidad hablaremos de “**Docker Compose**”, una utilidad para definir y poner en marcha múltiples contenedores Docker, mediante un fichero de configuración en formato **YAML**.

### 2. DOCKER COMPOSE

#### 2.1 ¿QUÉ ES DOCKER COMPOSE?

A lo largo de anteriores unidades hemos visto aplicaciones alojadas en contenedores Docker que para ponerse en marcha no utilizaban un único contenedor, sino que requerían el uso de varios contenedores. El proceso de poner en marcha estas aplicaciones era tedioso, ya que usualmente debíamos levantar “a mano” esos contenedores, respetando el orden de levantado, etc.

“**Docker Compose**” es una aplicación para simplificar la tarea de lanzar múltiples contenedores con una configuración específica y enlazarlos entre sí. Además, “**Docker Compose**” actúa como un orquestador simple, permitiendo cierto escalado local (aunque con bastantes limitaciones en comparación con otros orquestadores, como Kubernetes).

Para definir el comportamiento que realizará Docker Compose, utilizaremos un fichero de configuración escrito en formato **YAML**.

Para más información <https://docs.docker.com/compose/>

#### 2.2 Instalando Docker Compose

Para instalar/actualizar “**Docker Compose**”, seguiremos los pasos indicados para nuestro sistema operativo en <https://docs.docker.com/compose/install/>.

Básicamente la instalación consiste en obtener el binario adecuado para nuestro sistema de la cuenta GitHub de “**Docker Compose**” <https://github.com/docker/compose/releases> manualmente o con los comandos que proponemos y tras ello darle permisos de ejecución.

**Instalación Linux:** una forma de obtenerla es mediante el comando “**curl**” es:

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.0/docker-comp
ose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Una vez obtenido, le dotamos de permisos de ejecución:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Si lo deseamos, podemos crear también un enlace simbólico a otros sitios, como aquí:

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

**Instalación Windows:** en una PowerShell, habilita la versión de TLS que necesita GitHub para transferir paquetes con el comando:

```
[Net.ServicePointManager]::SecurityProtocol =  
[Net.SecurityProtocolType]::Tls12
```

y descargamos el binario con el comando:

```
Invoke-WebRequest  
"https://github.com/docker/compose/releases/download/1.29.0/docker-comp  
ose-Windows-x86_64.exe" -UseBasicParsing -OutFile  
$Env:ProgramFiles\docker\docker-compose.exe
```

Una vez instalado **“Docker Compose”**, podemos comprobar si la instalación es correcta con:

```
docker-compose --version
```

```
sergi@ubuntu:~$ docker-compose --version  
docker-compose version 1.29.0, build 07737305
```

En el momento de la redacción de este documento, la última versión es la **1.29.0**.

### 3. FORMATO YAML Y FICHERO “DOCKER-COMPOSE.YML”


#### 3.1 ¿Qué es YAML?

YAML es el acrónimo recursivo de **“YAML Ain't Markup Language”**, que en castellano significa, **“YAML no es un lenguaje de marcas”**. Mas información en <https://es.wikipedia.org/wiki/YAML>

**YAML** es a efectos prácticos una forma de definir información utilizando formato texto. Funciona de forma similar a **XML** o **JSON**. Si tenéis experiencia con **XML** o **JSON**, en este blog plantean y explican un ejemplo representando la misma información en los 3 formatos:

<https://journey2theccie.wordpress.com/2020/02/28/devnet-associate-1-1-compare-data-formats-xml-json-yaml/>

El uso de **YAML** dentro de **“Docker Compose”** es sencillo y fácilmente entendible con cada uno de los ejemplos. Podéis hacer un repaso previo a los principales elementos de **YAML** utilizados en ficheros **“Docker Compose”** revisando los ejemplos de **YAML** en la Wikipedia, disponibles en: <https://es.wikipedia.org/wiki/YAML#Ejemplos>.

 **Interesante:** Utilidades como <https://codeutility.org/yaml-to-json-xml/> nos permiten convertir **YAML** a **XML** o **JSON**.

#### 3.2 ¿Qué es el fichero “docker-compose.yml”?

El fichero **“docker-compose.yml”** es un fichero en formato **YAML** que definirá el comportamiento de cada configuración de **“Docker Compose”**. Lo habitual, es tener ese fichero en la raíz de nuestro proyecto. En este el próximo punto veremos que es el formato **YAML** y posteriormente presentaremos un ejemplo básico y comprensible de un fichero **“docker-compose.yml”**.

### 3.3 ¿Qué información podemos especificar en nuestro fichero “docker-compose.yml”?

La especificación completa para la versión 3 del fichero “**docker-compose.yml**” está definida en <https://docs.docker.com/compose/compose-file/compose-file-v3/>

A continuación, vamos a definir los más importantes:

- **version:** permite indicar la versión de la especificación del fichero “docker-compose.yml” no es necesario desde la versión 1.27.0 de “**Docker Compose**”.
- **services:** array asociativo con las diferentes plantillas de cada contenedor.
- **build:** usado en las plantillas de contenedores. Sirve para indicar si debemos construir la imagen a partir de un “**Dockerfile**”. Aunque tiene varias formas de uso detalladas en <https://docs.docker.com/compose/compose-file/compose-file-v3/#build>, la forma más habitual de usarla es indicar en qué directorio está nuestro “**Dockerfile**” de la forma “**build: ./directorio**”.
- **command:** sobrescribe el comando por defecto a la imagen. Se usa de la forma “**command: /bin/bash**”.
- **container\_name:** especifica un nombre de contenedor (si no, se generará automáticamente). Se usa de la forma “**container\_name: micontenedor**”
- **depends\_on:** indica que esta plantilla de contenedor, depende de que se haya creado un contenedor previo de la/s plantilla/s especificada/s. Los servicios también son detenidos en orden inverso a la dependencia. Se pueden ver ejemplos en [https://docs.docker.com/compose/compose-file/compose-file-v3/#depends\\_on](https://docs.docker.com/compose/compose-file/compose-file-v3/#depends_on)
- **env\_file y environment:** permite definir variables de entorno en la plantilla del contenedor.
  - **env\_file** especifica un fichero o lista de ficheros donde están definidas las variables de entorno, similar a “**env\_file: .env**”
  - **environment** especifica una lista de variables de entorno con su valor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#environment>
- **expose / ports:** permite definir un conjunto de puertos que se exportarán en el contenedor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#expose> y en <https://docs.docker.com/compose/compose-file/compose-file-v3/#ports>.
- **image:** especifica la imagen en la que se basa el contenedor. No es necesario cuando se especifica a partir de un “**Dockerfile**”.
- **network\_mode:** especifica el modo de red, de forma similar al parámetro --network de Docker. Los modos soportados se detallan en el siguiente enlace [https://docs.docker.com/compose/compose-file/compose-file-v3/#network\\_mode](https://docs.docker.com/compose/compose-file/compose-file-v3/#network_mode)
- **networks:** define las redes a crear para poner en marcha nuestros contenedores. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#networks>
- **restart:** indica cuando debe reiniciarse el contenedor. El valor por defecto es “**no**” (no se reinicia). Otros valores soportados son “**always**” (se reinicia cuando el contenedor se para) y “**on-failure**” (se reinicia si el contenedor se para y devuelve un valor de salida distinto de cero) y “**unless-stopped**” (se reinicia siempre, excepto si el contenedor es parado manualmente con “**docker stop**”).
  - Ejemplos <https://docs.docker.com/compose/compose-file/compose-file-v3/#restart>
- **tmpfs:** establece una lista de directorios a montar en formato “**tmpfs**” en el contenedor. Ejemplos en <https://docs.docker.com/compose/compose-file/compose-file-v3/#tmpfs>
- **volumes:** establece una lista de volúmenes, ya sea en formato “**bind mount**” o volumen Docker. Si quieres reutilizar un volumen entre distintas plantillas, además debes definir la variable “**volumes**” fuera de las plantillas de contenedores. Ejemplos <https://docs.docker.com/compose/compose-file/compose-file-v3/#volumes>

### 3.4 Entendiendo un fichero “docker-compose.yml” con ejemplo Wordpress

En este apartado vamos a ver un ejemplo de “**docker-compose.yml**” para poner en marcha Wordpress. En primer lugar, crearemos un directorio “**miwordpress**” o similar, nos situaremos dentro del directorio y allí crearemos el fichero “**docker-compose.yml**”.

El contenido completo del fichero sería el siguiente:

```
version: "3.9"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data:
```

Procedemos a analizar el contenido del fichero:

#### 3.4.1 Palabra clave “version”

```
version: "3.9"
```

Aquí tenemos definido en formato YAML un par variable/valor (la variable es “**version**” y el valor es “**3.9**”). “**Docker Compose**” ha ido ampliándose con el tiempo, así que aquí indicamos indicando la versión del formato de fichero. Este dato es opcional desde la versión “**1.27.0**” de “**Docker Compose**”, ya que es capaz de auto detectarlo.

### 3.4.2 Palabra clave “services” y contenedor “db”

```
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

En primer lugar, establece la variable “**services**”. Siguiendo el formato **YAML**, lo que almacena “**services**” es un array asociativo con las variables “**db**” y posteriormente “**wordpress**” (no se ve en este fragmento). Para saber que un elemento está contenido en otro, lo detecta mediante el nivel de los espacios (de forma similar a lenguajes como **Python**).

Cada una de esas variables, define un contenedor. En este caso, la plantilla de contenedor “**db**”, es también un array asociativo en formato **YAML** y define las siguientes propiedades:

```
image: mysql:5.7
```

Define la imagen que utilizará el contenedor.

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

Define qué volúmenes contendrá esta imagen. En este caso, los volúmenes en sí, se definen como una lista, de un solo elemento. Sabemos que es una lista porque en **YAML** empiezan con “-”.

En este caso, se define el volumen “db\_data” que se mapea con el directorio del contenedor “/var/lib/mysql”. Si quisiéramos tener varios volúmenes, podríamos hacer algo similar a:

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

Con esto ya hemos visto los 3 principales tipos elementos de **YAML** (par variable/valor, array asociativo, lista).

```
restart: always
```

Este fragmento, usando la sintaxis **YAML** de par variable/valor, indica que si el servidor se detiene, “**Docker Compose**” lo re-lance automáticamente (útil para contenedores que puedan caer por un fallo, pero sean necesarios para que la aplicación funcione).

```
environment:
  MYSQL_ROOT_PASSWORD: somewordpress
  MYSQL_DATABASE: wordpress
```

```
MYSQL_USER: wordpress
MYSQL_PASSWORD: wordpress
```

En este último fragmento, se define la variable “**environment**” del contenedor. Estas contendrán mediante un vector asociativo, el par variable/valor de cada variable de entorno definida en el contenedor. A efectos prácticos, está definiendo el password de root de **MySQL**, el nombre de una base de datos “wordpress”, un usuario con permisos de root para **MySQL** llamado “wordpress” (necesario para conexiones remotas) y su password como “wordpress”.

### 3.4.3 Palabra clave “services” y contenedor “wordpress”

```
wordpress:
  depends_on:
    - db
  image: wordpress:latest
  ports:
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
    WORDPRESS_DB_NAME: wordpress
```

En este caso, se está definiendo dentro de services, la plantilla del contenedor “**wordpress**”.

Pasamos a revisar sus fragmentos:

```
depends_on:
  - db
```

Indica que este contenedor depende del contenedor “**db**”, por lo cual el contenedor “**db**” deberá ponerse en marcha antes de iniciar nuestro contenedor “**wordpress**”.

```
image: wordpress:latest
```

Como se ha comentado anteriormente, indica la imagen en que se basa el contenedor.

```
ports:
  - "8000:80"
```

Indica que el puerto “**80**” de ese contenedor se mapea con el puerto “**8000**” del anfitrión.

```
restart: always
```

Como se ha comentado anteriormente, indica que si el contenedor se para, se reinicie automáticamente.

```
environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_USER: wordpress
```



```
WORDPRESS_DB_PASSWORD: wordpress
WORDPRESS_DB_NAME: wordpress
```

De forma similar al anterior punto, definimos el valor de distintas “variables de entorno” del contenedor “**wordpress**”.

### 3.4.4 Palabra clave “volumes” fuera de las plantillas de contenedores

```
volumes:
  db_data:
```

Simplemente indica que el volumen “**db\_data**” está disponible para otros contenedores puestos en marcha con “**Docker Compose**”.

### 3.5 Lanzando nuestro ejemplo Wordpress “docker-compose.yml”

Una vez creado y entendido nuestro fichero “**docker-compose.yml**”, podemos poner en marcha nuestro servicio situándonos en el directorio donde está este fichero y escribiendo:

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

Con la opción “**up**” indicamos que se interprete la plantilla definida en “**docker-compose.yml**” y con “**-d**” indicamos la ejecución en segundo plano. Observaremos algo similar a:

```
sergi@ubuntu:~/miwordpress$ docker-compose up
Creating network "miwordpress_default" with the default driver
Creating volume "miwordpress_db_data" with default driver
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
75646c2fb410: Pull complete
878f3d947b10: Pull complete
1a2dd2f75b04: Pull complete
8faaceef2b94: Pull complete
b77c8c445ec2: Pull complete
074029aeaa5f: Pull complete
5a1122545c6c: Pull complete
0539e5d96e0a: Pull complete
5bf3befab801: Pull complete
a8ba39ab191f: Pull complete
d472d441f7e9: Pull complete
```

Y continúa con texto como el siguiente

```
Digest: sha256:3e74612c64a39131ab2a567e91d4277c0a169ca4ef7f43504583a91890a439d4
Status: Downloaded newer image for wordpress:latest
Creating miwordpress_db_1 ... done
Creating miwordpress_wordpress_1 ... done
Attaching to miwordpress_db_1, miwordpress_wordpress_1
db_1 | 2021-04-09 17:21:12+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.33-1debian10 started.
wordpress_1 | WordPress not found in /var/www/html - copying now...
db_1 | 2021-04-09 17:21:12+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
wordpress_1 | Complete! WordPress has been successfully copied to /var/www/html
db_1 | 2021-04-09 17:21:12+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.33-1debian10 started.
wordpress_1 | No wp-config.php found in /var/www/html, but 'WORDPRESS_...' variables supplied; copying 'wp-config-docker.php' (WORDPRESS_
HOST WORDPRESS_DB_NAME WORDPRESS_DB_PASSWORD WORDPRESS_DB_USER)
db_1 | 2021-04-09 17:21:12+00:00 [Note] [Entrypoint]: Initializing database files
db_1 | 2021-04-09T17:21:12.691921Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_f
timestamp server option (see documentation for more details).
db_1 | 2021-04-09T17:21:14.927210Z 0 [Warning] InnoDB: New log files created, LSN=45790
db_1 | 2021-04-09T17:21:15.411983Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
db_1 | 2021-04-09T17:21:15.624677Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time that this s
ver has been started. Generating a new UUID: fd53881d-9957-11eb-8e66-0242ac120002.
db_1 | 2021-04-09T17:21:15.648253Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
db_1 | 2021-04-09T17:21:15.981954Z 0 [Warning] CA certificate ca.pem is self signed.
db_1 | 2021-04-09T17:21:16.199263Z 1 [Warning] root@localhost is created with an empty password ! Please consider switching off the
initialize-insecure option.
wordpress_1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerNam
directive globally to suppress this message
wordpress_1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerNam
directive globally to suppress this message
wordpress_1 | [Fri Apr 09 17:21:16.927983 2021] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.38 (Debian) PHP/7.4.16 configured -- resum
g normal operations
wordpress_1 | [Fri Apr 09 17:21:16.928025 2021] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
```

Observamos, que tal como hemos definido, primero se descarga y pone en marcha el contenedor basado en la plantilla “**db**”, al cual se le da nombre “**miwordpress\_db\_1**”, ya que establece el nombre del directorio como base “**miwordpress**”, luego el de la plantilla, y luego un número según el número de contenedor creado.

Posteriormente, se crea de igual forma el contenedor basado en la plantilla “**wordpress**”.

Si accedemos a <http://localhost:8000> observaremos que todo funciona correctamente.

Podemos detener todos los contenedores con el comando:

```
docker-compose down
```

## 4. COMANDOS DE “DOCKER COMPOSE”

En este apartado vamos a definir los principales comandos de “Docker Compose”.

### 4.1 Comando “up”

Con el comando “**up**” se interpretará la plantilla “**docker-compose.yml**” y se lanzarán los contenedores necesarios. El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/up/>

**Algunos de los usos más típicos son:**

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

Interpreta la plantilla y crea la aplicación, pero con el parámetro “**-d**” actúa en segundo plano.

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

El parámetro “**-f**” permite indicar un nombre de fichero **YAML** para “**Docker Compose**”.

```
docker-compose up -d --force-recreate
```

El parámetro “**--force-recreate**” fuerza a reconstruir contenedores incluso si ya existen en tu máquina.

```
docker-compose up -d --scale db=3
```

El parámetro “**--scale**” indica que del servicio “**db**” se creen tres contenedores, realizando un escalado local. El propio “**Docker Compose**” se encargará mediante algoritmo “round robin” de distribuir las peticiones al host “db” (como el nombre del servicio) a cada uno de los contenedores escalados. Esto se verá más claramente en uno de los casos prácticos propuestos.

### 4.2 Comando “down”

Con el comando “**down**” se interpretará la plantilla y se paran los contenedores necesarios.

El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/down/>

**Algunos de los usos más típicos son:**

```
docker-compose down
```

Detiene todos los contenedores.

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

Elimina los volúmenes creados en la plantilla

```
docker-compose down -t 5
```

Establece un “timeout” para la parada de contenedores de 5 segundos. El valor por defecto si no se indica nada es de 10 segundos.

### 4.3 Comando “build”

Comando que simplemente crea las imágenes definidas en la plantilla “**docker-compose.yml**”. Su funcionamiento se detalla en <https://docs.docker.com/compose/reference/build/>

### 4.4 Comando “pull”

Comando descarga las imágenes de contenedores necesarias para “**docker-compose.yml**”. Su funcionamiento se detalla en <https://docs.docker.com/compose/reference/pull/>

### 4.5 Comando “run”

Comando que permite lanzar un comando contra un servicio concreto definido en la plantilla “**docker-compose.yml**”. El detalle de su funcionamiento está disponible en <https://docs.docker.com/compose/reference/run/>

### 4.6 Comando “pause”/“unpause”

Con el comando “**pause**”/“**unpause**” se pausan/retoman los contenedores de la plantilla. El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/pause/>

### 4.7 Comando “start”/“stop”

Con el comando “**start**”/“**stop**” podemos iniciar/parar sin parámetros todos los servicios de la plantilla o con parámetros, el servicio que deseemos de esa plantilla. También inicia/para sus dependencias, si las tiene.

El detalle de su funcionamiento y parámetros está definido en:

- <https://docs.docker.com/compose/reference/start/>
- <https://docs.docker.com/compose/reference/stop/>

#### **Ejemplo:**

```
docker-compose start db
```

Inicia el servicio “**db**” de la plantilla.

```
docker-compose stop db
```

Detiene el servicio “**db**” de la plantilla.

### 4.8 Comando “ps”

Similar al comando “**docker ps**”, solo que aplicado a los contenedores de la plantilla. El detalle de su funcionamiento y parámetros está definido en la siguiente dirección <https://docs.docker.com/compose/reference/ps/>

### 4.9 Comando “exec”

El comando “**exec**” es similar a “**docker exec**”. Nos permite ejecutar un comando los contenedores con una plantilla determinada. El detalle de su funcionamiento está definido en <https://docs.docker.com/compose/reference/exec/>

#### **Ejemplo:**

```
docker-compose exec db mysql --version
```

#### 4.10 Comando “rm”

El comando “**rm**” elimina todos los contenedores parados según la plantilla determinada en “**docker-compose.yml**”. El detalle de su funcionamiento está definido en <https://docs.docker.com/compose/reference/rm/>

### 5. BIBLIOGRAFÍA

- [1] Docker Docs <https://docs.docker.com/>
- [2] Docker Compose <https://docs.docker.com/compose/>