Introducción a Docker

UD 06. Docker Compose

Autor: Sergi García Barea

Actualizado Marzo 2023

Licencia

Nomenclatura

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

📖 **Importante**

❗ **Atención**

💬 **Interesante**

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

En versiones antiguas de ***"Docker Compose",*** *este se instalaba con un ejecutable aparte (por este motivo en muchos manuales verás el comando de "**Docker Compose**" escrito como "**docker-compose**".* Aquí una captura antigua de este curso donde se observa este nombre:

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

Sin embargo, en las versiones actuales, dado su importancia, "Docker Compose" se ha integrado como un plugin de Docker y si hemos realizado la instalación de Docker como se propone en este curso y en la documentación oficial, ya lo tendremos.

En el momento de la redacción de este documento, la última versión es la **2.16.0**. Podemos ver una captura con la forma de llamarlo actual, como un comando más de Docker "***docker compose***".

```
sergi@casa:~$ docker compose version
Docker Compose version v2.16.0
```

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***". Más 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://onlineyamltools.com/convert-yaml-to-json 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**: sobreescribe 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 puden ver ejemplos en 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**: específica 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
- **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-stoped" (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: mariadb:10.11.2
     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: mariadb:10.11.2
      volumes:
         - db_data:/var/lib/mysql
      restart: always
      environment:
         MARIADB_ROOT_PASSWORD: somewordpress
         MARIADB_DATABASE: wordpress
         MARIADB_USER: wordpress
         MARIADB_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: mariadb:10.11.2
```

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 relance automáticamente (útil para contenedores que puedan caer por un fallo, pero sean necesarios para que la aplicación funcione).

```
    environment:
       MARIADB_ROOT_PASSWORD: somewordpress
       MARIADB_DATABASE: wordpress
       MARIADB_USER: wordpress
       MARIADB_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 imágen 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 (Nota: si nos da problemas, probemos sin usar "-d" para poder recabar mayor información de lo que está ocurriendo en los contenedores). Al acabar observaremos algo similar a:

```
sergi@casa:~/Desktop/miwordpress$ docker compose up -d
[+] Running 31/2
 ⠿ db Pulled
 ⠿ wordpress Pulled
[+] Running 3/3
 ⠿ Network miwordpress_default        Created
 ⠿ Container miwordpress-db-1         Started
 ⠿ Container miwordpress-wordpress-1  Started
```

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 imagenes 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
```

Pone en marcha 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 mariadb --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/