Docker Compose

Antonio Espín Herranz

Contenidos

- Introducción
- Fichero de configuración
- YAML
- services
- volumes
- network
- configs
- secret
- Comandos docker compose
- Archivos con Variables de entorno

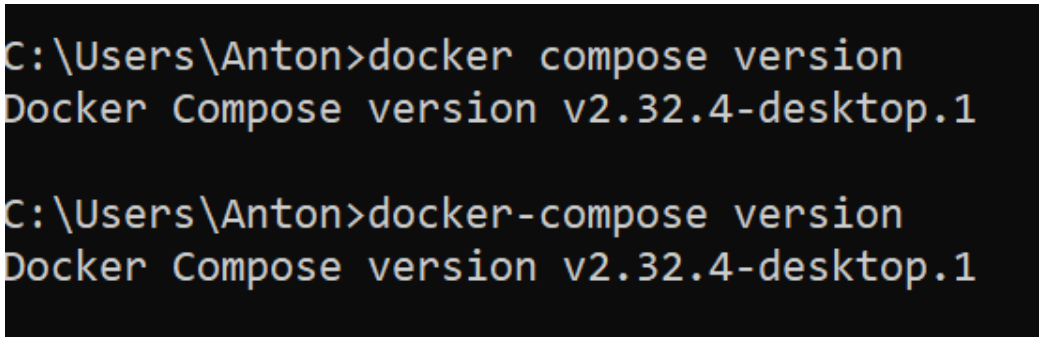
Introducción

- Docker Compose es una **herramienta** que se utiliza para definir y manejar aplicaciones Docker con múltiples contenedores de manera sencilla.
- Sirve para **automatizar** el **despliegue** y la configuración de estos **contenedores**, lo cual es muy útil cuando tienes varios servicios que necesitan comunicarse entre sí, como una base de datos, un backend y un frontend.

- Con Docker Compose podemos:
  - Definir la infraestructura en un archivo YAML (docker-compose.yml), especificando los contenedores, redes, volúmenes, etc.
  - **Levantar** todos los **servicios** con un solo comando (docker-compose up).
  - **Simplificar** la **gestión de servicios complejos**, eliminando la necesidad de configurar manualmente cada contenedor.

docker compose

- El **comando ya se instala en Windows al instalar Docker desktop**.
  - La versión 1 se hizo en Python
  - La versión 2 en go y que se integre como un comando de Docker.
- Se puede comprobar desde una consola lanzando el comando:
  - **docker compose** (*tiende a utilizarse esta*)
  - **docker-compose**
  - Uno u otro funcionan desde la consola de Windows



Fichero de configuración

- El fichero que busca por defecto se llama: **docker-compose.yml**
- También **compose.yml**

- Se puede cambiar el nombre con la opción –f
  - docker compose **–f** otro_fichero.yml

- Para iniciar y ejecutar los servicios se utiliza la opción: **up**
  - **docker compose up**
    - Busca el fichero por defecto e inicia y ejecuta servicios

docker compose up

- Comportamiento de docker-compose **up**:
  - **Construcción de imágenes**: Si no existen imágenes para los servicios definidos, las construirá usando las instrucciones del Dockerfile.
  - **Creación de contenedores**: Inicia los contenedores necesarios para los servicios especificados.
  - **Conexión de redes**: Configura las redes entre los contenedores según lo indicado en el archivo.
  - **Persistencia de volúmenes**: Crea y conecta los volúmenes definidos para almacenar datos.

Comandos relacionados

- **docker-compose up**: Inicia los servicios y muestra los logs directamente en la terminal.

- **docker-compose up -d**: Ejecuta los servicios en segundo plano (modo "**detached**").

- **docker-compose up --build**: Fuerza la reconstrucción de las imágenes antes de iniciar los servicios.

Apartados del fichero

- Sólo es obligatorio el apartado de services:
- version → version: **“3.9”**
- **services** → equivale a los contenedores
- volumes
- networks
- configs
- secret

- Comentarios con #

YAML

- Formato de serialización
- Se utiliza para definir listas, propiedades, etc.
- https://es.wikipedia.org/wiki/YAML

- Trabaja con indentaciones como en Python

Ejemplo

- El directorio donde especificamos la estructura y se encuentra el fichero compose.yml se trata como un proyecto.

- Mantener carpetas separadas con cada proyecto.
  - pr_nginx
    - compose.yml

  - version: ‘3.9’
  - services:
    - nginx:
      - image: nginx (aquí también se puede indicar build para construir la imagen a partir de un Dockerfile
    - ports:
      - “80:80”

- Se recomienda utilizar nuestras propias redes (bridge personalizadas)
  - Si no se indica la red construye una por defecto.

Services

- Es el apartado clave donde defines los servicios que compondrán tu aplicación.
- **Cada servicio corresponde a un contenedor Docker.**

- Dentro de **cada servicio** puedes **definir**:
  - **image**: Imagen base a usar.
  - **build**: Configuración para construir la imagen desde un Dockerfile.
  - **ports**: Mapeo de puertos entre el host y el contenedor.
  - **volumes**: Volúmenes para persistir o compartir datos.
  - **environment**: Variables de entorno.
  - **depends_on**: Dependencias de otros servicios.

Services

- Podemos tener varios:
  - services:
    - servicio_1:
      - # configuración s1
    - servicio_2:
      - # configuración s2
    - servicio_3:
      - # configuración s3

Ejemplo

```yaml
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  db:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=example
```

Ejemplo 2

```yaml
version: "3.9"

services:
  database:
    image: mysql:8.0
    container_name: mi_base_datos
    environment:
      MYSQL_ROOT_PASSWORD: supersegura
      MYSQL_DATABASE: mi_app
    ports:
      - "3306:3306"
    volumes:
      - datos_db:/var/lib/mysql

volumes:
  datos_db:
```

- **Servicio database:**
  - Se utiliza la imagen oficial de MySQL, versión 8.0.
  - Se configura con las variables de entorno para establecer la contraseña del usuario root y crear una base de datos llamada mi_app.
  - Se mapea el puerto 3306 del contenedor al host para que sea accesible.

- **Volumen datos_db:**
  - Está definido en la sección **volumes** del archivo.
  - Este volumen se monta en **/var/lib/mysql**, que es donde MySQL almacena los datos en el contenedor. Esto asegura que los datos persistan incluso si el contenedor se detiene o elimina.

- **Persistencia**:
  - Los datos almacenados en **datos_db** permanecerán disponibles, ya que Docker gestiona el volumen fuera del ciclo de vida del contenedor.

Volumes

- Define volúmenes que pueden ser utilizados por los servicios para persistir datos o compartir información entre contenedores.

```
volumes:
  datos_app:
```

Networks

- Especifica redes personalizadas que permiten la comunicación entre los servicios o con el host.

```
networks:
  red_interna:
```

Configs

- Define configuraciones que los servicios pueden consumir, como archivos de configuración.
- Se suele utilizar con Docker Swarm (similar a Kubernetes)

```
configs:
  app_config:
    file: ./config/app.conf
```

Secret

- Permite manejar información sensible, como contraseñas, de manera segura.
- Se suele utilizar con Docker Swarm (similar a Kubernetes)

```
secrets:
  db_password:
    file: ./password.txt
```

Fichero entero

- Tiene 3 servicios
  - web, app y db

- Cada servicio va en un contenedor distinto.

```yaml
version: "3.9"

services:
  web:
    image: nginx
    ports:
      - "8080:80"
    networks:
      - red_interna

  app:
    build:
      context: ./app
    depends_on:
      - db
    environment:
      - DATABASE_URL=mysql://db:3306
    networks:
      - red_interna

  db:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=example
    volumes:
      - datos_db:/var/lib/mysql
    networks:
      - red_interna

volumes:
  datos_db:

networks:
  red_interna:
```

Comandos docker compose

- Los comandos permiten administrar el ciclo completo de una aplicación:
  - docker compose **up**
    - Crea e inicia los contenedores. Con –d se ejecutan en 2 plano
  - docker compose **down**
    - Detiene los contenedores que están en ejecución y elimina las redes.
    - Con –v elimina todos los volúmenes (de la sección de volumes) y los creados por los contenedores.
  - docker compose **ps**
    - Estado de los contenedores de la aplicación
  - docker compose **logs**
    - Muestra stdout (salida estándar) y stderr (errores) de los contenedores
    - Con –f actualizar el tiempo real.
  - docker compose **exec**
    - Ejecuta un comando en un contenedor que está en ejecución

Comandos docker compose II

- docker compose **top**
  - Los procesos que se están ejecutando en cada uno de los contenedores de la aplicación.
- docker compose **start**
  - Inicia los servicios (contenedores) de la aplicación
- docker compose **stop**
  - Detiene los servicios (contenedores) de la aplicación
- docker compose **build**
  - Crea las imágenes de los servicios que utilizan ficheros dockerFile, en lugar de utilizar la imagen de un repositorio (Docker hub)

Archivos con variables de entorno

- Hay veces que necesitamos utilizar variables de entorno dentro del archivo de configuración y las podemos pasar a través de otro fichero: **.env**

- Estos ficheros **tienen que estar en el mismo directorio** que el fichero: docker-compose.yml

- Formato:
  - VARIABLE=valor
  - Líneas que empiezan con # están comentadas

Archivos con variables de entorno

- Fichero:
  - MYSQL_ROOT_PASSWORD=root
  - MYSQL_DATABASE=database
  - …

- Para utilizarlo:
  - services
    - mysql:
      - environment:
        - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}