Curso Docker

Arturo Silvelo

Try New Roads

Tabla de Contenido

Introducción

Contenedores

Redes

Volúmenes

Imágenes

Docker Compose

¿Qué es Docker Compose?

- **Definición**: Docker Compose es una herramienta para definir y ejecutar aplicaciones de múltiples contenedores a partir de un único archivo de configuración YAML.
- **Ventajas**: Simplifica la gestión de servicios, redes y volúmenes en un solo archivo, facilitando el despliegue de tu stack de aplicaciones de forma eficiente.
- **Uso**: Con un solo comando, puedes crear y arrancar todos los servicios configurados en el archivo `docker-compose.yml`, ideal para entornos de producción, desarrollo, testing y CI/CD.
- **Comandos Disponibles**:
  - Iniciar, detener y reconstruir servicios.
  - Ver el estado de los servicios en ejecución.
  - Visualizar logs de los servicios.
  - Ejecutar comandos puntuales en un servicio.

¿Cómo Funciona Docker Compose?

- Docker Compose utiliza un archivo YAML (`compose.yaml`) para definir la configuración de tu aplicación y sus servicios.
- Sigue las reglas establecidas por la *Compose Specification*[1].
- También se soportan archivos `docker-compose.yml` para compatibilidad con versiones anteriores.
- Permite combinar múltiples archivos Compose para definir aplicaciones más complejas.
- Puedes utilizar fragmentos y extensiones para mantener un archivo eficiente y modular.

[1] Consulta la especificación completa en:
`https://docs.docker.com/reference/compose-file/`

Comandos Comunes de Docker Compose

`docker compose`[2] para gestionar aplicaciones multicontenedor.

- `docker compose up`: Inicia todos los servicios definidos en el archivo.
- `docker compose down`: Detiene y elimina todos los contenedores, redes y volúmenes creados por `up`.
- `docker compose logs`: Muestra los logs de todos los servicios.
- `docker compose ps`: Lista todos los contenedores que se están ejecutando.

Puedes invocar contenedores de manera individual usando `docker compose <command> <nombre del servicio>`.

[2] `https://docs.docker.com/reference/cli/docker/compose/`

Creando un archivo `compose.yaml` básico

Crear un contenedor de mongo:

```
docker run -d \
  --name my-mongo-container \
  --hostname mongodb \
  -p 27017:27017 \
  -e MONGO_INITDB_ROOT_USERNAME=admin \
  -e MONGO_INITDB_ROOT_PASSWORD=secret \
  -v $(pwd)/mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro \
  -v mongo-data:/data/db \
  --network my-network \
  --restart always \
  mongo:latest
```

Creando un archivo `compose.yaml` básico

Un archivo `compose.yaml` define cómo se deben ejecutar los servicios en Docker Compose.

```yaml
version: '3.8'  # Versión de Docker Compose

services:
  database:  # Definición del servicio "database"
    image: mongo:latest  # Imagen de Docker para MongoDB
    container_name: my-mongo-container  # Nombre del contenedor
    hostname: mongodb  # Nombre del host dentro del contenedor
    volumes:  # Volúmenes para persistencia de datos
      - mongo-data:/data/db  # Monta un volumen para la base de datos
      - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro  # Script de inicialización (
          opcional)
    networks:  # Redes a las que se conecta el servicio
      - my-network
    ports:  # Mapea los puertos del contenedor al host
      - "27017:27017"  # Mapea el puerto 27017 del contenedor al host
    environment:  # Variables de entorno para la configuración de MongoDB
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=secret
    restart: always  # Política de reinicio del contenedor

networks:  # Esta sección crea la red si no existe
  my-network:
    driver: bridge

volumes:  # Define el volumen para la persistencia
  mongo-data:
    driver: local
```

Ejercicio 1: Crear un Docker Compose para las Aplicaciones Back-end y Frontend

Objetivo: En este ejercicio, deberás crear un archivo `docker-compose.yml` para levantar tanto la aplicación de **Backend** como la de **Frontend**. La aplicación de **Backend** utilizará una base de datos en memoria.

1. Configuración básica del compose (version, services, networks, volumes)
2. Añadir Backend: Configuración para usar una base de datos en memoria
3. Añadir Frontend

Ejercicio 2: Usar una base de datos

Objetivo: En este ejercicio, deberás editar el archivo `docker-compose.yml` para levantar tanto la aplicación de **Backend** como la de **Frontend**, la **Base de datos** y una **interfaz** para la Base de datos **(mongo-express)**.

1. Editar Backend para usar mongo
2. Añadir Mongo
3. Añadir Mongo Express

Ejercicio 3: ¿Y la seguridad?

Objetivo: En este ejercicio, deberás editar el archivo `docker-compose.yml` para securizar la base de datos y las conexiones.

- Mongo: Añadir usuario y contraseña a la base de datos para evitar entrar sin autenticación:

  MONGO_INITDB_ROOT_USERNAME
  MONGO_INITDB_ROOT_PASSWORD

- Backend: Configurar el backend para acceder con autenticación.

  `mongodb://<user>:<password>@<host>:<port>/<db>?authSource=admin`

- Mongo Express: Configurar la interfaz de mongo para acceder con autenticación.

  `mongodb://<user>:<password>@<host>:<port>`

Ejercicio 4: Más seguridad

Objetivo: En este ejercicio, deberás editar el archivo `docker-compose.yml` para permitir solo conexiones a la interfaz mongo desde el host y eliminar los puertos no necesarios

- Mongo: Eliminar puerto
- Mongo Express

Acceso restringido al host:

```
ports:
  - 127.0.0.1:8081:8081
```

Acceso restringido a la red local:

```
ports:
  - <ip_host>:8081:8081
```

Sin restricción:

```
ports:
  - 8081:8081
```