Imágenes

Ver Imágenes:

```
docker images
```

Descargar imagen:

```
docker pull nombreImagen:versionImagen
```

Ver configuración de docker:

```
docker info
```

Eliminar imágenes:

```
docker image nombreImagen:versionImagen
```

Contenedores

Crear contenedor:

`docker create nombreImagen` -> Esto devolverá un hash.

Iniciar contenedor:

`docker start hashContenedor` -> Esto devolverá otro hash

Ver contenedores en ejecución:

```
docker ps
```

Detener contenedor:

`docker stop hashContendor` -> Esto devuelve un hash con el mismo hash que le hemos proporcionado.

Ver contenedores creados:

```
docker ps -a
```

Eliminar contenedor:

```
docker rm hashContendor/nombreContenedor
```

Crear contendor con alias propio:

`docker create --name nombrePropio nombreImagenes` -> Podemos iniciar el contendor mediante su alias (`docker start nombrePropio`).

Exponer un puerto para conectarse desde la maquina real:

```
docker create -p puertoExterno:puertoInterno
```

Ver logs de un contenedor:

```
docker logs nombreContenedor
```

Monitorizar logs de un contenedor:

```
docker logs --follow nombreContendor
```

Comando rápido para levantar un contenedor EN PRIMER PLANO con sus imágenes rápido (si no la encuentra, la descargará):

`docker run nombreImagen` -> Esto presenta un problema porque ejecuta la imagen en segundo plano.

Comando rápido para levantar un contenedor EN SEGUNDO PLANO con sus imágenes rápido (si no la encuentra, la descargará):

```
docker run -d nombreImagen
```

Ejecutar todo de golpe con parámetros personalizados:

```
docker run --name nombreContenedor -p puertoExterno:puertoInterno -d nombreImagen
```

NOTA: EL COMANDO RUN CREA UN CONTENEDOR CADA VEZ QUE SE EJECUTA

Ejemplo MongoDB con variables de entorno:

1. ```
   docker create -p 27017:27017 –name monguito -e MONGO_INITDB_ROOT_USERNAME=usuario -e
   MONGO_INITDB_ROOT_PASSWORD=contraseña mongo
   ```

2. ```
   docker start monguito
   ```

Crear contenedor

Un ejemplo de creación de contenedor para node y mongo. Para crear un contenedor tenemos que crear el archivo Dockerfile (en la raíz de la carpeta del proyecto) con lo siguiente:

```
FROM node:18 #imagen en la que está basado el container

RUN mkdir -p /home/app #ruta interna del contenedor donde estará alojada la app

COPY . /home/app #copia la app del SO real a la ruta del contenedor

EXPOSE 3000

CMD ["node", "/home/app/index.js"] #comando que inicializará la app. PD: se pasa el comando y cada parámetro en una posición del array
```

NOTA: LO ANTERIOR NO FUNCIONARÁ PORQUE LOS SERVICIOS DE MONGO Y NODE ESTÁN EN CONTENEDORES DISTINTOS

Para crear una red, utilizamos el siguiente comando:

```
docker network create nombreRed
```

Ver las redes:

```
docker network ls
```

Borrar una red:

```
docker network rm nombreRed
```

NOTA: DOCKER UTILIZA EL NOMBRE DEL CONTENEDOR COMO HOSTNAME

Construir una imagen con un Dockerfile:

```
docker build nombreImagen rutaProyecto
```

Ejemplo. Crear contendor con una red y variables de entorno:

```
docker create -p puertoExterno:puertoInterno --name nombreContendor --network nombreRed -e
MONGO_INITDB_ROOT_USERNAME=usuario -e MONGO_INITDB_ROOT_PASSWORD=contraseña
```

Docker Compose

Para hacer los pasos anteriores con docker compose:

docker-compose.yaml

```yaml
version: "3.9" #version de docker Compose

services:
  node: #nombre del contenedor
    build: . #ruta del Dockerfile
    ports:
      - "3000:3000" #puertoExterno:puertoInterno
    links:
      - mongo #mapea el contendor con el que interactuará

  mongo:
    image: mongo #nombre de la imagen
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=usuario
      - MONGO_INITDB_ROOT_PASSWORD=contraseña
```

Para levantar/construir el contenedor ejecutamos en la ruta donde está el fichero docker-compose.yml lo siguiente:

```
docker compose up -d
```

Para eliminar todo lo asociado a un docker compose ejecutamos el siguiente comando en el directorio del docker-compose.yml:

```
docker compose down
```

NOTA: SI ESPECIFICAMOS LOS CONTENDORES DENTRO DE UN ARCHIVO docker-compose.yml LOS INCLUIRÁ AUTOMÁTICAMENTE DENTRO DE UNA MISMA RED

VOLÚMENES

Para persistir los datos de la bd de mongo dejamos el archivo docker-compose.yml de la siguiente manera:

docker-compose.yaml

```yaml
version: "3.9" #version de docker Compose

services:
  node: #nombre del contenedor
    build: . #ruta del Dockerfile
    ports:
      - "3000:3000" #puertoExterno:puertoInterno
    links:
      - mongo #mapea el contendor con el que interactuará

  mongo:
    image: mongo #nombre de la imagen
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=usuario
      - MONGO_INITDB_ROOT_PASSWORD=contraseña
    volumes:
      - mongo-data: /data/db #Ruta donde se guarda la data de la bd por defecto
      # mysql -> /var/lib/mysql
      # postgres -> /var/lib/postgresql/data

volumes:
  mongo-data:
```