Docker Básico

Arturo Silvelo

Try New Roads

Ejercicios Contenedores

Objetivo de la práctica

Aprender a gestionar contenedores Docker mediante la creación, modificación y manipulación de un contenedor NGINX.

Pasos a seguir

1. Limpiar cualquier recurso previo (contenedores, imágenes, volúmenes, redes).

```
docker container prune
docker image prune
docker volume prune
docker network prune
```

2. Crear un contenedor NGINX que sirva contenido web en un puerto aleatorio.

Este contenedor se ejecutará en un puerto aleatorio y permitirá acceder al contenido web que servirá.

```
docker run -d --name nginx-practica -P nginx
```

Utiliza el comando `docker ps` para ver el contenedor en ejecución y el puerto asignado.

```
docker ps
```

3. Acceder al contenedor para modificar el archivo `index.html` y personalizar el contenido.

Una vez que el contenedor esté en ejecución, accede al contenedor y edita el archivo `index.html` para modificar el contenido, como los elementos `h1` o `p`. Puedes hacerlo con los siguientes comandos:

```
docker exec -it nginx-practica /bin/bash
cd /usr/share/nginx/html
nano index.html
```

Estos cambios se verán reflejados inmediatamente en el contenedor de `nginx-practica` y serán accesibles en el navegador si la página está en ejecución.

Instala `nano` para editar el archivo `index.html`

```
apt update
apt install -y nano
```

4. Copiar el contenido de la carpeta HTML del contenedor al sistema host.

Una vez que hayas realizado los cambios, copia el contenido del directorio HTML del contenedor a tu máquina host con el siguiente comando:

```
docker cp nginx-practica:/usr/share/nginx/html .
```

Este comando copia el contenido de la carpeta `/usr/share/nginx/html` del contenedor `nginx-practica` a la ubicación actual del host.
Al ejecutar este comando, se creará una carpeta llamada `html` en el directorio actual del host, que contendrá todos los archivos HTML del contenedor.

5. Crear un nuevo contenedor NGINX utilizando la carpeta copiada como su contenido web.

Monta la carpeta copiada anteriormente como su contenido web

```
docker run --name nginx-practica -v $(pwd):/usr/share/nginx/html -P nginx
```

ó

```
cd /ruta/a/mi/carpeta
docker run --name nginx-practica -v .:/usr/share/nginx/html -P nginx
```

La carpeta del host queda sincronizada con el contenedor, reflejando cualquier cambio en tiempo real (como las ediciones en `index.html`).

En sistemas Unix, `$(pwd)` representa el directorio actual. En PowerShell de Windows, usa `${PWD}` en su lugar.

Soluciones Redes

Descargar imágenes

```
docker pull ghcr.io/trynewroads/course-frontend:1.0.0
docker pull ghcr.io/trynewroads/course-backend:1.0.0
```

Verificar que las imágenes se han descargado

```
docker image ls
```

1. Crea una red personalizada llamada `course-network` que permita la comunicación entre los contenedores.

```
docker network create course-network
```

Verifica que se ha creado y el rango de la subred creada

```
docker network ls
docker inspect course-network
```

```
{
    "Subnet": "172.22.0.0/16",
    "Gateway": "172.22.0.1"
}
```

2. Inicia un contenedor basado en la imagen `course-backend`, conéctalo a la red `course-network` ,dale un nombre de host y al contenedor.

```
docker run
-d  # Segundo plano
-p 3000:3000 # Conectar el puerto 3000 del host con el 3000 del contenedor
--network course-network # Asignar la red
--hostname course-backend # Nombre en la red
--name cb # Nombre del contendor
ghcr.io/trynewroads/course-backend:1.0.0 # imagen usada
```

Comprobar la creación

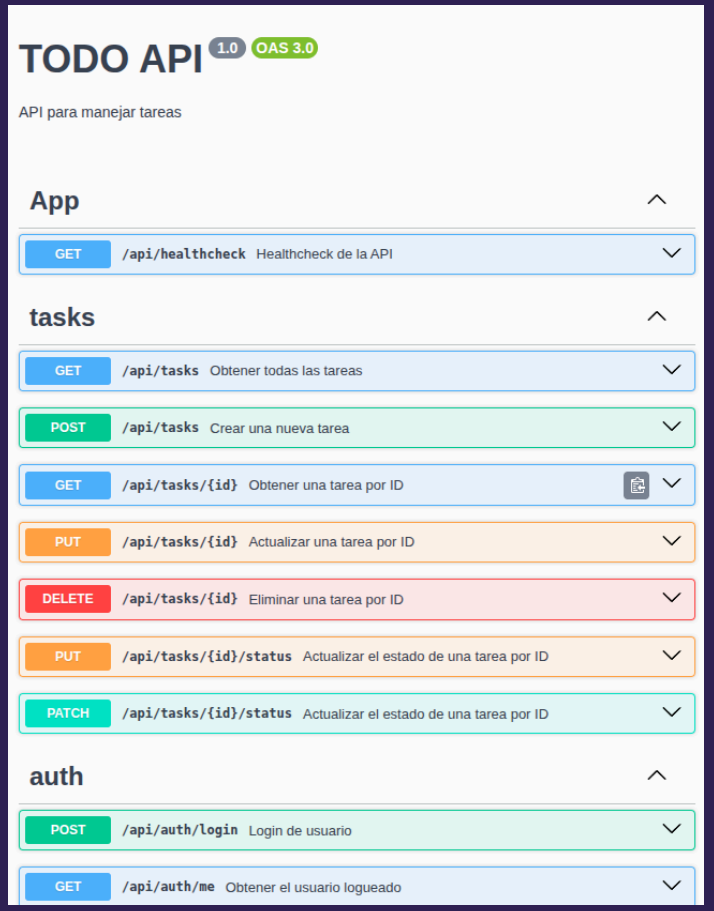
```
docker ps
```

Comprobar la red

```
docker inspect course-network
```

```
"Networks": {
  "course-network": {
    Gateway": "172.22.0.1",
    "IPAddress": "172.22.0.2",
    "DNSNames": [
      "cb", # Resolución por el nombre del contenedor
      "43bf280a0797", # Resolución por el id del contenedor
      "course-backend" # Resolución por el hostname
    ]
```

Acceder servidor **http://localhost:3000/docs**



3. Inicia un contenedor basado en la imagen `course-frontend` y conéctalo también a la red `course-network`.

Crear fichero de configuración ningx `default.conf.template`

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://course-backend:3000/api;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Headers para CORS si es necesario
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS";
        add_header Access-Control-Allow-Headers "Content-Type, Authorization";
    }
}
```

```
docker run
-d  # Segundo plano
-p 80:80 # Conectar el puerto 3000 del host con el 3000 del contenedor
-v ./default.conf.template:/etc/nginx/templates/default.conf.template # Configuración de ningx
--network course-network # Asignar la red
--hostname course-frontend # Nombre en la red
--name cf # Nombre del contendor
ghcr.io/trynewroads/course-frontend:1.0.0 # imagen usada
```

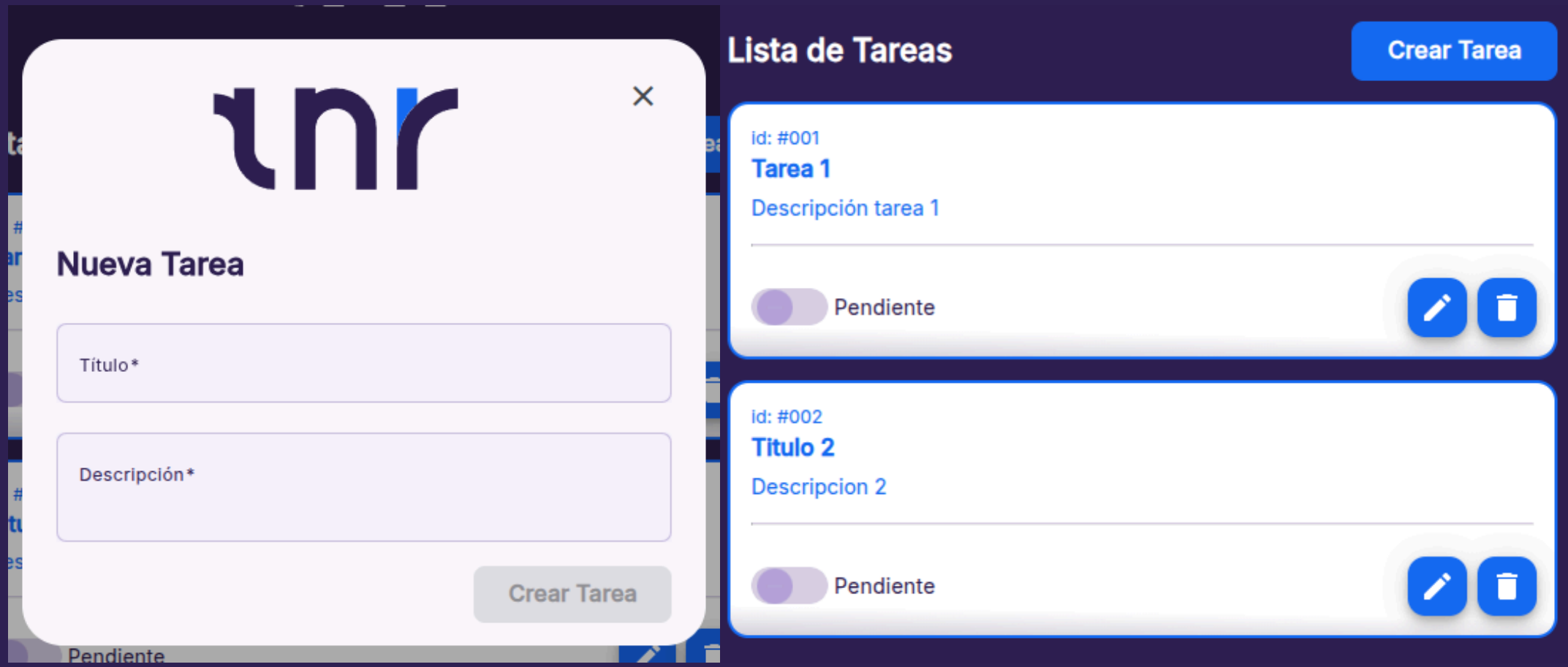
Comprobar la creación

```
docker ps
```

Comprobar la red

```
docker inspect course-network
```

Acceder web **http://localhost** con el usuario `admin` y contraseña `12345678`



4. Asegúrate de que la aplicación frontend pueda acceder al backend a través del nombre del servicio (`course-backend`) en lugar de una dirección IP.

- Usuario `admin`, contraseña: `12345678`
- Acceder servidor **http://localhost:3000/docs**
- Acceder cliente **http://localhost:80**
- Crear una tarea
- Ver el listado de tareas

Ejercicio: Volúmenes

Ejercicio 1: Anonymous

1. Crea una nueva instancia de postgres con un volumen anónimo e inspecciona la información de los volúmenes.

Creamos el contenedor de postgres y lo asignamos a la red

```
docker run --name cdb -e POSTGRES_PASSWORD=12345678 --hostname course-database --network course-network -d postgres
```

Verificar la creación

```
docker ps
docker inspect course-network
```

2. Inserta algunas entradas de ejemplo en la base de datos de postgres.

Comprobar que los datos se han guardado en la base de datos

```
docker exec -it cdb psql -U postgres -d postgres -c "SELECT * FROM task;"
```

3. Elimina el contenedor de postgres y crea uno nuevo.

Eliminamos la máquina en caliente, este paso puede hacer que el backend deje de funcionar, por lo que tenemos que reiniciar.

```
docker rm -f cdb

docker restart cb
```

Creamos de nuevo la máquina

```
docker run --name cdb -e POSTGRES_PASSWORD=12345678 --hostname course-database --network course-network -d postgres
```

Repetimos el punto 2 para verificar que están los datos.

Ejercicio 2: Named Volumes

1. Crea un nuevo contenedor de postgres usando un volumen de nombrado.

Creamos el volume

```
docker volume create postgress-data
```

Comprobar que los datos se han guardado en la base de datos

```
docker run --name cdb -e POSTGRES_PASSWORD=12345678 --hostname course-database --network course-network -d  -v postgress-data:/var/lib/postgresql/data postgres
```

2. Inserta algunas entradas de ejemplo en la base de datos de postgres.

Comprobar que los datos se han guardado en la base de datos

```
docker exec -it cdb psql -U postgres -d postgres -c "SELECT * FROM task;"
```

3. Elimina el contenedor de postgres y crea uno nuevo.

Eliminamos el contenedor

```
docker rm -f cdb
```

Creamos otro de nuevo pero usando el mismo volume

```
docker run --name cdb -e POSTGRES_PASSWORD=12345678 --hostname course-database --network course-network -d  -v postgress-data:/var/lib/postgresql/data postgres
```

Comprobar que los datos siguen

```
docker exec -it cdb psql -U postgres -d postgres -c "SELECT * FROM task;"
```

Ejercicio 3: Bind Mounts

1. Haz una copia del volumen en un directorio local.

Copiamos el contenido de la base de datos a local.

```
docker cp cdb:/var/lib/postgresql/data .
```

Esto copia la carpeta de data del contenedor a una carpeta data en el host

2. Creamos una nueva máquina de postgres con un volumen de tipo `bind`.

Eliminamos el contenedor de la base de datos.

```
docker rm -f cdb
```

Creamos uno nuevo pero usando `bind` para montar el directorio copiado en el paso anterior

```
docker run --name cdb -e POSTGRES_PASSWORD=12345678 --hostname course-database --network course-network -d -v ./data:/var/lib/postgresql/data postgres
```

3. Verifica que ambas bases de datos tengan los mismos datos.

Es posible que necesitemos reiniciar el backend

```
docker restart cb
```

Comprobamos que los datos de la nueva base de datos sean los mismos.

```
docker exec -it cdb psql -U postgres -d postgres -c "SELECT \* FROM task;"
```

También podemos crear nuevas tareas y comprobar que se añaden

Ejercicios Imágenes

Ejercicio 1: Crear una Imagen Docker

Crear una imagen Docker para la aplicación **backend**

1. Descargar el repositorio

```
git clone git@github.com:trynewroads/course-backend.git
```

2. Crear un archivo `Dockerfile`

3. Usar `node:20` como imagen base.

```
FROM node:20
```

4. Establecer un directorio de trabajo (`/app`).

```
WORKDIR /app
```

5. Copiar fichero de dependencias (`package.json`).

```
COPY package*.json ./
```

6. Instalar las dependencias (`npm install`).

```
RUN npm install
```

7. Definir **variables** y configurar puerto (3000).

```
ENV NODE_ENV=production \
PORT=3000 \
DEBUG_REQUEST=false \
ENABLE_AUTH=true \
DB_HOST=localhost \
DB_PORT=5432 \
USE_DB=false

EXPOSE 3000
```

8. Copiar todo el contenido.

```
COPY . .
```

9. Compilar la aplicación (`npm run build`)

```
RUN npm run build
```

10. Comando para iniciar la aplicación (`node dist/main.js`)

```
CMD [ "node", "dist/main.js" ]
```

11. Crear la imagen

```
docker build -t course-test-backend -f 05-images/soluciones/Dockerfile ../course-backend/

docker image ls course-test-backend
```

12. Probar la imagen

Para evitar conflictos con el contenedor que esta corriendo, vamos cambiar los nombres y el puerto.

```
docker run \
-d \
-p 3001:3000 \
--network course-network \
--hostname course-test-backend \
--name ctb \
-e USE_DB=true \
-e DB_HOST=course-database \
-e DB_PORT=5432 \
-e DB_USER=postgres \
-e DB_PASS=12345678 \
-e DB_NAME=postgres \
course-test-backend
```

Ahora podremos acceder al backend en **Contenedor 1** y **Contenedor 2**

Si creamos una tarea usando el Swagger podremos ver que se añade en el frontend.

```
docker ps
```

Ejercicio 2: Crea una imagen de docker para desarrollo

Crear una imagen Docker para la aplicación **backend**

1. Descargar el repositorio

```
git clone git@github.com:trynewroads/course-backend.git
```

2. Crear un archivo `Dockerfile.dev`

3. Usar `node:20` como imagen base

```
FROM node:20
```

4. Establecer un directorio de trabajo (`/app`)

```
WORKDIR /app
```

5. Copiar fichero de dependencias (`package.json`)

```
COPY package*.json ./
```

6. Instalar las dependencias (`npm install`)

```
RUN npm install
```

7. Definir **variables** y configurar puerto (3000)

```
ENV NODE_ENV=production \
PORT=3000 \
DEBUG_REQUEST=false \
ENABLE_AUTH=true \
DB_HOST=localhost \
DB_PORT=5432 \
USE_DB=false

EXPOSE 3000
```

8. Copiar todo el contenido

```
COPY . .
```

9. Definir volume (en la carpeta `app`)

```
VOLUME /app
```

10. Comando para iniciar la aplicación (`npm run start:dev`)

```
CMD [ "npm", "run", "start:dev" ]
```

11. Crear la imagen (`docker build`)

```bash
docker build -t course-dev-backend -f 05-images/soluciones/Dockerfile.dev ../course-backend/

docker image ls course-dev-backend
```

12. Probar la imagen (`docker run`)

Con esta imagen tendremos algo parecido a la anterior, pero si modificamos el código dentro del contenedor los cambios se verán reflejados.

```
docker run \
-d \
-p 3002:3000 \
--network course-network \
--hostname course-dev-backend \
--name cvb \
-e USE_DB=false \
course-dev-backend
```

Lo que podemos hacer es montar el volumen del host de forma directa y así las modificaciones que se hacen en el host se actualizarán con el contenedor y se verán reflejados de forma inmediata.

```
docker run \
-d \
-p 3003:3000 \
--network course-network \
--hostname course-dev-backend \
--name cvvb \
-v ../course-backend:/app \
--user $(id -u):$(id -g) \
-e USE_DB=false \
course-dev-backend
```

Ejercicio Docker Compose

Crear la estructura que llevamos usando hasta ahora pero en formato `compose`.

Ejercicio 1: Crear Compose de backend

1. Crear un fichero `docker-compose.yml` ó `compose.yml`

2. Configurar el servicio `backend`

```
docker run -d \
-p 3005:3000 \
--network course-network \
--hostname course-compose-backend \
--name ccb \
-e USE_DB=false \
ghcr.io/trynewroads/course-backend:1.0.0
```

3. Levantar el `compose`

```
docker compose -f 06-compose/soluciones/compose.yml up -d

docker compose -f ./06-compose/soluciones/compose.yml ps
```

4. Comprobar que el servicio.

Acceder al **servidor**

Ejercicio 2: Crear Compose del frontend

1. Editar el fichero `docker-compose.yml` ó `compose.yml`

2. Configurar el servicio `frontend`

```
docker run \
-d
-p 8080:80 \
--network course-network \
--hostname course-compose-frontend \
-v ./nginx/default.conf.template:/etc/nginx/templates/default.conf.template:ro \
--name ccf \
ghcr.io/trynewroads/course-frontend:1.0.0
```

3. Levantar el `compose`

```
docker compose -f 06-compose/soluciones/compose.yml up -d

docker compose -f ./06-compose/soluciones/compose.yml ps
```

4. Comprobar que el servicio.

Acceder al **Cliente**

Ejercicio 3: Crear Compose de la base de datos

1. Editar el fichero `docker-compose.yml` ó `compose.yml`

2. Configurar el servicio `postgres`

```
docker run
-d
--name ccdb
-e POSTGRES_PASSWORD=12345678
--hostname course-database
--network course-network
-v postgres_data:/var/lib/postgresql/data postgres
```

3. Levantar el `compose`

```
docker compose -f 06-compose/soluciones/compose.yml up -d

docker compose -f ./06-compose/soluciones/compose.yml ps
```

4. Comprobar que el servicio.

```
docker exec -it ccdb psql -U postgres -d postgres -c "SELECT * FROM task;"
```

Ejercicio 4: Mejorar compose

1. Establecer redes independientes para aislar los servicios que se comuniquen entre ellos.

```
course-compose-backend:
  networks:
    - course-compose-back
    - course-compose-front

course-compose-frontend:
  networks:
    - course-compose-front

course-compose-database:
  networks:
    - course-compose-back

networks:
  course-compose-back:
    driver: bridge

  course-compose-front:
    driver: bridge
```

2. Establecer dependencia entres los servicios para que se inicien en orden (`depends_on`)

```
course-compose-backend:
  depends_on:
    - course-compose-database

course-compose-frontend:
  depends_on:
    - course-compose-backend
```

3. Eliminar las exposición de puertos no necesarios (`puerto de backend`)

```
course-compose-backend:
  #port:
  #  - 3005:3000
```

4. Limpiar sistema

Probamos el compose que funciona con los cambios

```
docker compose -f ./06-compose/soluciones/4.compose.yml up -d
docker compose -f ./06-compose/soluciones/4.compose.yml ps
```

Comprobamos si podemos acceder al **Cliente**, que no podemos acceder al **Servidor**.

```
docker compose -f ./06-compose/soluciones/4.compose.yml down --volumes
```