Docker Básico

Arturo Silvelo

Try New Roads

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
```