07 NEST BONUS DOCKERIZAR APP

El DockerFile

- Elimino la db pokedex del container de DockerDesktop
- Una vez tengo la imagen solo necesito este comando.
    - Especifico el archivo con -f y le asigno un nuevo .env de producción

```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up
```

- Creo el Dockerfile en la raíz
- Para construir estas imágenes (la de mongo, por ejemplo) es este procedimiento
- Este Dockerfile es una **versión básica**. En la siguiente lección hay una versión mas **PRO**
- Usualmente vamos a construir imággenes basadas en otras imágenes
    - De aquí el **FROM node:18**, alpine es una imagen super liviana de linux (6MB) con características esenciales
        - Es como tener un linux con node instalado
    - Creo un working directory con **RUN mkdir** (ejecuto el comando mkdir) /var/www/pokedex
    - Le digo que trabaje en el directorio que he creado con **WORKDIR**
- **NOTA**: es cómo tener un SO con el que te comunicas a través de comandos, sin interfaz visual
    - Le digo que copie todo lo que hay con **COPY .** (origen) ./var/www/pokedex (destino)
    - **COPY** los archivos package.json,ts-config, etc. **el último es el path**
        - En teoría no sería necesario pero lo hago por si acaso
    - **RUN** (ejecuta) npm i --prod y npm run build. El build construye la carpeta dist/
        - Debo asegurarme que no tengo la carpeta dist para que no se copie con el comando **COPY .**
            - El dist no lo quiero copiar porque lo voy a ejecutar yo
            - No quiero la data de mongo porque voy a crear la imagen sin el volumen (la data de la db guardada en el directorio)
        - Tampoco quiero los node_modules ya que son especificos del SO
        - Para ignorar estos archivos y carpetas voy a crear el .dockerignore
- .dockerignore
-

```
dist/
node_modules/
.gitignore
.git/
mongo/
```

- **RUN** añadir usuario pokeuser sin password. Conviene crear un nuevo usuario para no usar el root por default
- **RUN** establezco el acceso de pokeuser a solo /var/www/pokedex
    - Si entrara un admin solo podría hacer lo que pokeuser puede hacer, que es en el directorio /var/www/pokedex

- **USER** pokeuser, hago uso del usuario que acabo de crear
- Limpio la caché
- Expongo el puerto 3000
-
- Dockerfile

```
FROM node:18-alpine3.15

# Set working directory
RUN mkdir -p /var/www/pokedex
WORKDIR /var/www/pokedex

# Copiar el directorio y su contenido
COPY . ./var/www/pokedex
COPY package.json package-lock.json tsconfig.json tsconfig.build.json
/var/www/pokedex/
RUN npm install --prod
RUN npm run build


# Dar permiso para ejecutar la applicación
RUN adduser --disabled-password pokeuser
RUN chown -R pokeuser:pokeuser /var/www/pokedex
USER pokeuser

# Limpiar el caché
RUN npm cache clean --force

EXPOSE 3000

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

Definir la construcción de la imagen

- Borro lo que hay en el Dockerfile y pego esto
- Nombro la imagen de node como deps
- Instalo la librería libc6 (necesaria)
- Trabajo en el directorio /app
- Copio el package.json
- Uso el npm ci que equivale a yarn install --frozen-lockfile
    - Todo esto crea la imagen únicamente con las dependencias de mi aplicación (el package.json y el package-lock)
    - Muevo las dependencias y hago las instalaciones de los módulos de node ahi
    - De esta manera puedo mantener en caché todas esas dependencias. Y solo si cambian van a instalarse
    - Por lo que hacer diferentes builds, si no han cambiado las dependencias va a ser super rápido porque todo está en caché

    - Si no hiciera este paso, cada vez haría el npm i para instalar todas las dependencias
- Hago el build, es otra imagen de node que llamo builder
    - Trabajo en /app
    - Va a copiar de las dependencias (deps, así llamé la imagen anterior) de /app/node_modules (origen) a node_modules (destino)
    - Si no cambiaron es un paso que está todo en caché e instantáneamente los va a mover
    - Copio todo al WORKINDIRECTORY con **COPY .** a **.** (al WORKINGDIRECTORY)
    - Hago el build de producción
- El siguiente paso es el runner. Es quien va a correr la app
    - El WORKDIR es apuntar a app igual
    - Copio el package.json y el package-lock. El último path es dónde quiero que caigan los archivos (./)
    - Hago la instalación de las dependencias de producción
    - Copio del builder (el paso anterior) la carpeta app/dist a ./dist(destino)
    - Ejecuto node dist/main
- El EXPOSE 3000 no va a hacer falta porque por defecto lo vamos a manejar desde afuera

```
# Install dependencies only when needed
FROM node:18-alpine3.15 AS deps
# Check https://github.com/nodejs/docker-
node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why
libc6-compat might be needed.
RUN apk add --no-cache libc6-compat
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci

# Build the app with cache dependencies
FROM node:18-alpine3.15 AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npm run build


# Production image, copy all the files and run next
FROM node:18-alpine3.15 AS runner

# Set working directory
WORKDIR /usr/src/app

COPY package.json package-lock.json ./

RUN npm install --prod

COPY --from=builder /app/dist ./dist

# # Copiar el directorio y su contenido
# RUN mkdir -p ./pokedex
```

```
# COPY --from=builder ./app/dist/ ./app
# COPY ./.env ./app/.env

# # Dar permiso para ejecutar la applicación
# RUN adduser --disabled-password pokeuser
# RUN chown -R pokeuser:pokeuser ./pokedex
# USER pokeuser

# EXPOSE 3000

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

- Creo el docker-compose.prod.yaml en la raíz de mi proyecto

```
version: '3'  # es un standard

services:
  pokedexapp:     # creo un nuevo servicio
    depends_on:
      - db        # depende de la db. Si la db no se levanta no se levantará
pokedex
    build:        # construcción
      context: .  # que se base en la posición física de mi docker-
compose.prod.yaml
      dockerfile: Dockerfile      # Dockerfile, solo tengo uno
    image: pokedex-docker         # Así se llamará la imagen
    container_name: pokedexapp    # El nombre del container
    restart: always # reiniciar el contenedor si se detiene
    ports:
      - "${PORT}:${PORT}"         # Conecto mi variable de entorno PORT con la del
contenedor
    # working_dir: /var/www/pokedex
    environment:                  # Defino mis variables de entorno. Las uso de mi
.env.prod
      MONGODB: ${MONGODB}
      PORT: ${PORT}
      DEFAULT_LIMIT: ${DEFAULT_LIMIT}
    # volumes:
    #   - ./:/var/www/pokedex     # Por si quisiera montar con la data existente
en el fs, pero no es lo habitual en un build

  db:
    image: mongo:5              # la imagen que tengo de la db
    container_name: mongo-poke  # el nombre, importante!
    restart: always             # en caso de que se caiga va a volver a qintentar
levantarla
    ports:
      - 27017:27017             # establezco los puertos de comunicación entre la
db y el contenedor
    environment:
      MONGODB_DATABASE: nest-pokemon  # la variable de entorno
    # volumes:
```

```
    #   - ./mongo:/data/db  //En lugar de usar un volumen con data vamos a
levantar la imagen de la db desde 0
```

- Antes de ejecutar el comando quiero crear un archivo de variables de entorno de producción

---

Construir la imagen

- Copio el .env en .env.prod
- Con la variable de entorno MONGODB necesito apuntar al container de la db que tengo en el docker-compose.prod.yaml
- Uso el container_name y el puerto que expongo en el docker-compose.prod.yaml para apuntar a él, seguido del nombre de la db
- El container_name podría verse como un DNS de la dirección del container (192.169....)

```
MONGODB=mongodb://mongo-poke:27017/nest-pokemon
```

- No tengo nada corriendo
- Para hacer el build es

```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up --build
```

- Uso -d para que no ver toda la info de docker en consola
- Cuando quiera ejecutarlo será este comando (sin el --build)

```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up
```

- docker-compose usa por defecto .env por lo que si no lo especificamos lo tomará por defecto
- Para ejecutar el build debo tener DockerDesktop corriendo y conexión (obvio)
- Ahora puedo usar el endpoint http://localhost:3000/api/v2/seed para ejecutar el seed
- Con ctrl+C cancelo el proceso de docker en la terminal
- Ahora ya no necesito usar el --build, solo docker-compose -f docker-compose.prod.yaml --env-file .env.prod up

---

Conservar la db y analizar la imagen

- Para habilitar la persistencia de la data en la db descomento las dos últimas lineas del docker-compose.prod.yaml

```
version: '3'

services:
  pokedexapp:
    depends_on:
      - db
    build:
      context: .
```

```
      dockerfile: Dockerfile
    image: pokedex-docker
    container_name: pokedexapp
    restart: always # reiniciar el contenedor si se detiene
    ports:
      - "${PORT}:${PORT}"
    # working_dir: /var/www/pokedex
    environment:
      MONGODB: ${MONGODB}
      PORT: ${PORT}
      DEFAULT_LIMIT: ${DEFAULT_LIMIT}
    # volumes:
    #   - ./:/var/www/pokedex

  db:
    image: mongo:5
    container_name: mongo-poke
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGODB_DATABASE: nest-pokemon
    #AQUI!!!!
    volumes:
      - ./mongo:/data/db
```

- Estoy apuntando al mismo lugar que docker-compose.yaml
- Quito mongo de .dockerignore
- Si cambiaramos algo de la app habría que hacer de nuevo el --build
- Me puedo conectar directamente a la imagen a través de la terminal
- Puedo abrir el contenedor con la accion (los tres puntitos) de Open in Terminal
- Desde ahi puedo acceder a la carpeta dist, listar con ls, visualizar archivos con cat, podría instalar apt-get e instalar un editor para editar los archivos como nano, etc

---

Actualizar README

- README

````
<p align="center">
  <a href="http://nestjs.com/" target="blank"><img
src="https://nestjs.com/img/logo-small.svg" width="200" alt="Nest Logo" /></a>
</p>

# Ejecutar en desarrollo

1. Clonar el repositorio
2. Ejecutar
```
npm i
````
````

```
```

3. Tener el Nest CLI instalado
```
npm i -g @nestjs/cli
```
4. Levantar la base de datos
```
docker-compose up -d
```
5. Clonar el archivo .env.template y renombrar la copia a .env

6. LLenar las variables de entorno definidas en el .env

7. Ejecutar la app en dev:
```
npm run start:dev
```

8. Reconstruir la base de datos con la semilla
```
http://localhost:3000/api/v2/seed
```
Build de produccion

1. Crear env.prod
2. Llenar las variables de entorno de producción
3. Crear la imagen con docker-compose
```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up --build
```

Stack Usado

- MongoDB
- Nest