Práctica 6.1 – Dockerización de una aplicación Nodejs

1. Introducción

En este caso vamos a Dockerizar una aplicación escrita en Node.js que hace uso de una API para gestionar personas en un libro de direcciones que guardará en una base de datos PostgreSQL.

¿Por qué *dockerizar*?

Si uno trata de informarse, encontrará múltiples y variadas razones para dockerizar nuestras aplicaciones y servicios.

Por citar sólo algunas:

1. **Configuración rápida del entorno en local para el equipo de desarrollo:** si todos los servicios están implementados con contenedores, es muy rápida la configuración de dicho entorno.
2. **Evita el clásico "en mi máquina funciona":** gran parte de los problemas de desarrollo provienen de la propia configuración que los integrantes del equipo de desarrollo tienen de su entorno. Con los servicios en contenedores, esto queda solucionado en gran medida.
3. **Despliegues más rápidos.** Con docker compose se puede levantar con un solo comando.
4. **Mejor control de versiones:** como ya sabéis, se puede etiquetar (tags), lo que ayuda en el **control de versiones.**
5. **Rollbacks más fáciles:** puesto que se tienen las cosas más controladas por la versión, es más fácil revertir el código. A veces, simplemente apuntando a su versión de trabajo anterior.
6. **Fácil configuración de múltiples entornos:** como hacen la mayoría de los equipos de desarrollo, se establece un entorno local, de integración, de puesta en escena (preprod) y de producción. Esto se hace más fácil cuando los servicios están en contenedores y, la mayoría de las veces, con sólo un cambio de VARIABLES DE ENTORNO.
7. **Apoyo de la comunidad:** existe una fuerte comunidad de ingenieros de software que continuamente contribuyen con grandes imágenes que pueden ser reutilizadas para desarrollar un gran software. ¿Por qué reinventar la rueda?

2. Despliegue con Docker

En primer lugar, si eliminastéis el repositorio en su momento, debéis volver a clonarlo en vuestra Debian, en caso contrario obviad este paso:

```
$ git clone https://github.com/raul-profesor/DAW_practica_6.1_2024.git
```

Ahora, puesto que la aplicación ya viene con el Dockerfile necesario dentro del directorio para construir la imagen y correr el contenedor, vamos a estudiar su contenido.

Así pues, tener nuestra aplicación corriendo es cuestión de un par de comandos.

Hacemos un build de la imagen de Docker. Le indicamos que ésta se llama librodirecciones y que haga el build con el contexto del directorio actual de trabajo, así como del Dockerfile que hay en él:

```
$ docker build -t librodirecciones .
```

Y, por último, iniciamos el contenedor con nuestra aplicación. Ahora sí, con la opción -p, le indicamos que escuche conexiones entrantes de cualquier máquina en el puerto 3000 de nuestra máquina anfitrión que haremos coincidir con el puerto 3000 del contenedor (-p 3000:3000). Y con la opción -d lo haremos correr en modo demonio, en background:

```
$ docker run -p 3000:3000 -d librodirecciones
```

Tras esto sólo queda comprobar que al intentar acceder desde nuestra máquina a la aplicación**: http://IP_Maq_Virtual:3000** se produce un error de conexión.

Esto sirve para ilustrar un punto importante de los contenedores: poseen su propia red. La aplicación, por defecto, intenta buscar la base de datos en nuestro localhost, pero, técnicamente, está en otro host (su contenedor).

A pesar de que todos los contenedores corren en la misma máquina, cada uno es considerado un host diferente y por eso la aplicación falla al conectar.

Podríamos utilizar los comandos network de Docker para solucionar el asunto. En lugar de eso, introduciremos el concepto de Docker Compose para administrar contenedores.

Actividades

1. Completa este Dockerfile con las opciones/directivas adecuadas, leed los comentarios y podéis apoyaros en la teoría, en este cheatsheet, en este otro o en cualquiera que encontréis.

```
#Indica que se utilizará una imagen de Docker Hub oficial de Node, en su versión 18.16.0 junto con Alpine, una diminuta distribución de Linux

_____ node:18.16.0-alpine3.17

# Ejecuta un comando en una nueva capa de la imagen

_____  mkdir -p /opt/app

# Define el Directorio sobre el que se ejecutarán las subsiguientes instrucciones del Dockerfile

_____ /opt/app

# Copia los archivos de fuera del contenedor dentro, en este caso package.json

_____ src/package.json src/package-lock.json .

# Permite ejecutar “npm install” para instalar las dependencias en la imagen

_____ npm install
```

```
# Copiamos todos los archivos de nusetro directorio /src al contenedor

_____ src/ .

# Documenta qué puertos estarán expuestos o a la esucha en el contenedor

_____ 3000

# Permite ejecutar un comando dentro del contenedor. Inicia la aplicación

_____ ["npm", "run", "start:dev"] #
```

2. Documenta, incluyendo capturas de pantallas, el proceso que has seguido para realizar el despliegue de esta nueva aplicación, así como el resultado final.

3. Docker Compose

Docker Compose es una herramienta para gestionar aplicaciones multicontenedor. En Linux tiene que ser instalado por separado, consultad su documentación para ello.

Docker Compose puede:

- Iniciar y detener múltiples contenedores en secuencia.
- Conectar contenedores utilizando una red virtual.
- Manejar la persistencia de datos usando Docker Volumes.
- Establecer variables de entorno.
- Construir o descargar imágenes de contenedores según sea necesario.

Docker Compose utiliza un archivo de definición YAML para describir toda la aplicación. En nuestro caso:

```
services:
  postgres:
    image: postgres:latest
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - '5432:5432'
    volumes:
      - addressbook-db:/var/lib/postgresql/data
```

```
  addressbook:
    build:
      context: .
    environment:
      DB_SCHEMA: postgres
      DB_USER: postgres
      DB_PASSWORD: postgres
      DB_HOST: postgres
    depends_on:
      - postgres
    ports:
      - '3000:3000'


volumes:
  addressbook-db:
```

Así las cosas, para levantar nuestra infraestructura basada en contenedores no tenemos más que hacer:

$ docker compose run addressbook npm run migrate

Esto creará las tablas necesarias en la base de datos.

Y construiremos nuestros contenedores a partir de las imágenes:

$ docker compose up --build -d

Una vez construidas las imágenes, podemos levantar los contenedores:

Podéis correr unos tests para comprobar que la aplicación funciona correctamente con:

$ docker compose run addressbook npm test

Tarea 3. Probad que la aplicación junto con la BBDD funciona correctamente. El funcionamiento de la API es:

- GET /persons/all muestra todas las personas en el libro de direcciones
- GET /persons/1 muestra la persona con el id 1
- PUT /persons/ añade una persona al libro de direcciones
- DELETE /persons/1 elimina a la persona con el id 1

Ejemplos:

```
curl -X PUT http://IP_APLICACION:3000/persons -H 'Content-Type: application/json' -
d '{"id": 1, "firstName": "Raúl", "lastName": "Profesor"}'
```

```
curl -X GET http://localhost:3000/persons -H 'Content-Type: application/json'
```