Backend Sprint 0 PBIO

Generado por Doxygen 1.9.4

Capítulo 1

Sprint0

1.1. V0.1

1.1.1. ¿Cómo funciona esta primera versión?

He seguido las instrucciones de Jordi Bataller, creando una carpeta propia para MariaDB y Node.js, con sus respectivos Dockerfiles. Después, he creado un archivo `docker-compose.yml` que relaciona ambos contenedores y automatiza todo aquello que deberíamos hacer a mano.

1.1.1.1. MariaDB

El Dockerfile de MariaDB funciona de la siguiente manera:

- **Imagen Base**: Se utiliza la última imagen de MariaDB de la web oficial de Docker Hub (`mariadb`). Para crear esta imagen se ejecuta el comando `docker pull mariadb`.
- **Contraseña**: Se crea una variable que establece una contraseña (en esta primera versión es `1234`).
- **Inicialización**: Se ejecuta el comando para crear la base de datos (ubicada en el directorio `/sql` dentro de la misma carpeta de MariaDB) y se define la ruta del entrypoint que inicializará la base de datos en el contenedor de MariaDB.

1.1.1.1.1. SQL El archivo `ejemploBBDD.sql` contiene la creación de la base de datos y la tabla que se utilizará en el proyecto. Aquí se guardará la información que se reciba de los sensores de gas. En este instante, contiene los siguientes campos:

- `id`: Es la Primary Key y es un valor que se autoincrementa.
- `hora`: Está definida por el tipo de variable `TIME` y debe ser `NOT NULL`, debido a que no puede dar un valor 0. Tendrá la siguiente estructura: "00:00", siendo el primer par de ceros las horas (de 1 a 24) y el segundo par los minutos.
- `lugar`: Está definido por el tipo de variable `VARCHAR` que puede contener hasta 255 caracteres, tampoco puede ser `NULL`.
- `id_sensor`: Es un valor entero `INT` y será el encargado de identificar a cada sensor individualmente.

- `valorGas`: Es la medición que dará el sensor de gas y está definida por un número `DECIMAL(5,2)`, que tampoco puede ser `NULL`.
- `valorTemperatura`: Es la medición que dará el sensor de temperatura. Sigue estando definida por un número `DECIMAL(5,2)` y su implementación se realizará en un futuro.

Un ejemplo de inserción de datos podría ser el siguiente:

```
INSERT INTO mediciones (id, hora, lugar, idSensor, valorGas, valorTemperatura)
VALUES (333, '12:00', 'Cartagena', 130, 55, 35);
```

Mediante la consola de MariaDB desde dentro de su contenedor, se ha comprobado de manera exitosa que los datos se insertan correctamente y que la tabla se actualiza recargando la página en tu propia máquina en local: http://localhost:8080

1.1.1.2. Node.js

Dentro de la carpeta de Node.js podrás encontrar todos los módulos que se crean al iniciar por primera vez npm con Node.js (el comando es: `npm init -y`).

El Dockerfile de Node.js funciona de la siguiente manera:

- **Imagen Base**: Se utiliza la última imagen de Node.js de la web oficial de Docker Hub (`node`). Para crear esta imagen se ejecuta el comando `docker pull node`.
- **Directorio de Trabajo**: Se crea el directorio donde se ejecutará nuestra lógica de negocio ubicada en el archivo `main.js`.
- **Copiar Archivos de Dependencias**: Se copian los archivos generados por el comando `npm init -y` (`package.json` y `package-lock.json`) desde nuestro host al sistema de archivos del contenedor Docker.
- **Instalación de Dependencias**: Para comunicarse con la base de datos de MariaDB, es necesario que se ejecute la instalación con el comando `npm install mariadb`.
- **Copiar Código**: Con `COPY . .` copiamos el resto del código de nuestra máquina host al contenedor Docker en el directorio correspondiente.
- **Exponer Puerto**: Abrimos el puerto 8080 de nuestra máquina local con el comando `EXPOSE 8080`. De esta manera podremos acceder al JSON de la base de datos con la siguiente URL: http://localhost:8080.
- **Ejecutar Aplicación**: Por último, ejecutamos en la consola nuestro `main.js` con el comando `CMD ["node", "main.js"]`.

1.1.1.2.1. ServidorREST En el archivo `servidorREST.js` se ha creado un pequeño servidor REST donde se ha configurado la base de datos de MariaDB, pasándole los parámetros para poder acceder a la misma. También, se ha implementado una API con Swagger, que es una herramienta gráfica que ayuda a probar la API creada. Esta API se encarga de hacer consultas a la base de datos de manera de prueba. Puede realizar tanto GET como POST, apuntando a la tabla de mediciones (que contiene los campos: `id`, `hora`, `lugar`, `idSensor`, `valorGas`, `valorTemperatura`).

El funcionamiento de la API se puede ver de manera muy visual yendo a la siguiente dirección URL: http://localhost:3000/api-docs/.

1.1.1.3. docker-compose.yml

Este es un archivo que se encarga de unir tanto el Dockerfile de MariaDB como el Dockerfile de Node.js. Con este archivo, se ejecutan automáticamente ambos sin la necesidad de teclear nada por la terminal.

- **MariaDB**: Se abre un puerto en 3306 y un puerto de Docker (también el 3306). Después, crea una red interna que es esencial para que ambos contenedores funcionen correctamente sincronizados. En mi caso se llama "redsprint0". He implementado también una verificación de salud del contenedor de MariaDB. El comando `healthcheck` se encarga de pasar unos tests internos que aseguran el correcto funcionamiento del contenedor.
- **Node.js**: En el módulo de Node.js se abre también un puerto (en este caso el 8080 para que no se solape con el 3306 de MariaDB) y se crea también la misma red que hará de puente entre los dos contenedores.

1.2. V0.2

1.2.1. ¿Cómo funciona esta segunda versión?

1.2.1.1. MariaDB

El Dockerfile de MariaDB funciona de la siguiente manera:

- **Imagen Base**: Se utiliza la última imagen de MariaDB de la web oficial de Docker Hub (`mariadb`). Para crear esta imagen se ejecuta el comando `docker pull mariadb`.
- **Contraseña**: Se crea una variable que establece una contraseña (en esta primera versión es `1234`).
- **Inicialización**: Se ejecuta el comando para crear la base de datos (ubicada en el directorio `/sql` dentro de la misma carpeta de MariaDB) y se define la ruta del entrypoint que inicializará la base de datos en el contenedor de MariaDB.

1.2.1.1.1. SQL El archivo `ejemploBBDD.sql` contiene la creación de la base de datos y la tabla que se utilizará en el proyecto. Aquí se guardará la información que se reciba de los sensores de gas. En este instante, contiene los siguientes campos:

- `hora`: Está definida por el tipo de variable `TIME` y debe ser `NOT NULL`, debido a que no puede dar un valor 0. Tendrá la siguiente estructura: "00:00", siendo el primer par de ceros las horas (de 1 a 24) y el segundo par los minutos.
- `lugar`: Está definido por el tipo de variable `VARCHAR` que puede contener hasta 255 caracteres, tampoco puede ser `NULL`.
- `id_sensor`: Es un valor entero `INT` y será el encargado de identificar a cada sensor individualmente.
- `valorGas`: Es la medición que dará el sensor de gas y está definida por un número `DECIMAL(5,2)`, que tampoco puede ser `NULL`.
- `valorTemperatura`: Es la medición que dará el sensor de temperatura. Sigue estando definida por un número `DECIMAL(5,2)` y su implementación se realizará en un futuro.

Se ha eliminado el campo 'id' debido a que es autoincremental y se trata de una Primary Key, para así evitar errores al hacer 'POST' en la Tabla 'mediciones'.

Un ejemplo de inserción de datos podría ser el siguiente:

```
INSERT INTO mediciones (hora, lugar, id_sensor, valorGas, valorTemperatura) VALUES
('12:00', 'Cartagena', 130, 55, 35);
```

Ahora ya no hace falta acceder a la consola de MariaDB para comprobar que los archivos se han insertado correctamente. Únicamente, deberemos de acceder a la API de 'Swagger' `http://localhost:8080`) y comprobar a través de su Interfaz Gráfica lo siguiente:

- Al pulsar en el desplegable de 'POST /mediciones Agrega una nueva medición' y después sobre el botón 'Try it out', podremos realizar una petición 'POST' directamente a la base de datos de mariadb. Insertemos por tanto el ejemplo anteriormente mencionado
- Para comprobar que se ha enviado correctamente, nos dirigiremos a la parte de abajo, en la sección de 'Responses', si nos aparece el código 201, significará que la Medición se ha creado exitosamente
- Si quieres comprobarlo visualmente, te invito a que pulses ahora en el desplegable de 'GET /mediciones Obtiene todas las mediciones', y después en el botón 'Try it out'. Te deberán de salir los datos de ejemplo añadidos automáticamente y el que acabas de crear

1.2.1.2. Node.js

Dentro de la carpeta de Node.js podrás encontrar todos los módulos que se crean al iniciar por primera vez npm con Node.js (el comando es: npm init -y).

El Dockerfile de Node.js funciona de la siguiente manera:

- **Imagen Base**: Se utiliza una imagen fija de Node.js de la web oficial de Docker Hub (`node`). En este caso, es la 18, la más estable para el proyecto
- **Directorio de Trabajo**: Se crea el directorio donde se ejecutará nuestra lógica de negocio en el contenedor de nodejs, en este caso se ubicará en '/home/nodejs'
- **Copiar Archivos de Dependencias**: Se copian los archivos generados por el comando npm init -y (package.json y package-lock.json) desde nuestro host al sistema de archivos del contenedor Docker.
- **Instalación de Dependencias**: Para comunicarse con la base de datos de MariaDB, es necesario que se ejecute la instalación con el comando npm install mariadb y para evitar errores, se usará el comamdp RUN apt update.
- **Copiar Código**: Con COPY . . copiamos el resto del código de nuestra máquina host al contenedor Docker en el directorio correspondiente.
- **Exponer Puerto**: Abrimos el puerto 8080 de nuestra máquina local con el comando EXPOSE 8080. De esta manera podremos acceder al JSON de la base de datos con la siguiente URL: `http://localhost:8080/mediciones`.
- **Ejecutar Aplicación**: Por último, ejecutamos en la consola nuestro main.js con el comando CMD ["node", "main.js"].

1.2.1.2.1. ServidorREST En el archivo servidorREST.js se ha creado un pequeño servidor REST donde se ha configurado la lógica de la base de datos de MariaDB con una API de Swagger. Esta API se encarga de hacer consultas a la base de datos de mariadb a través del puerto 3306. Puede realizar tanto GET como POST, apuntando a la tabla de 'mediciones' (que contiene los campos: id, hora, lugar, idSensor, valorGas, valorTemperatura).

El funcionamiento de la API se puede ver de manera muy visual yendo a la siguiente dirección URL: http://localhost:8080/api-docs/.

Aquí hay un esquema básico de lo que realiza esta herramienta visual:

```
/**
 * @swagger
 * components:
 *   schemas:
 *     Medicion:
 *       type: object
 *       required:
 *         - id
 *         - hora
 *         - lugar
 *         - id_sensor
 *         - valorGas
 *         - valorTemperatura
 *       properties:
 *         id:
 *           type: integer
 *           description: ID de la medición
 *         hora:
 *           type: string
 *           description: Hora de la medición
 *         lugar:
 *           type: string
 *           description: Lugar de la medición
 *         id_sensor:
 *           type: integer
 *           description: ID del sensor
 *         valorGas:
 *           type: number
 *           description: Valor de la medición de Gas
 *         valorTemperatura:
 *           type: number
 *           description: Valor de la medición de Temperatura
 *       example:
 *         hora: '10:00'
 *         lugar: 'Madrid'
 *         id_sensor: 101
 *         valorGas: 40
 *         valorTemperatura: 32
 */
```

1.2.1.3. docker-compose.yml

Este es un archivo que se encarga de unir tanto el Dockerfile de MariaDB como el Dockerfile de Node.js. Con este archivo, se ejecutan automáticamente ambos sin la necesidad de teclear nada por la terminal.

- **Node.js**: En el módulo de Node.js se abre el puerto 8080 de ambas máquinas y se crea una dependencia con la base de datos de mariadb.
- **MariaDB**: Se abre un puerto en 3306 y un puerto de Docker (también el 3306). He prescindido de usar una red interna entre ambos contenedores debido a que los 2 ya pertenecen a la misma red local. También, he prescindido de hacer una comprobación de salud del contenedor, debido a que no hace falta a tales niveles en el proyecto. Por último, he considerado que es necesario guardar los datos nuevos introducidos en un volumen

1.3. V0.3

1.3.1. ¿Cómo funciona esta tercera versión?

1.3.1.1. Node.js

Ningún cambio en el código de Node.js. Se sigue utilizando el mismo código que en la versión anterior.

1.3.1.1.1. ServidorREST Mismo código que en la versión anterior

1.3.2. Novedades de esta versión

1.3.2.1. docker-compose.yml

He implementado un nuevo contenedor en esta versión: el contenedor de nginx. Éste se encarga básicamente de hacer de servidor para la app_web. Por lo tanto, he creado un nuevo archivo de Dockerfile en la carpeta de app_web. En este archivo, se instala nginx y se configura para que redirija el tráfico hacia el puerto 8080 de la app_web.

1.3.2.2. Variables de Entorno

He creado el archivo variables.env con la intención de guardar ahí las credenciales sensibles del servidor y de las credenciales de acceso a la base de datos. De esta manera, los datos no están expuestos a posibles ataques.

1.3.2.3. .gitignore

He creado el archivo para que git ignore ciertos archivos y carpetas que no son necesarios ser subidos a GitHub. De esta manera, no los trackea y se quedan en local