Desafío 10 - Bootcamp Devops Engineer

Alumno: Ignacio Peretti

Objetivo:

El siguiente desafío tiene como objetivo desarrollar un build en docker y configurar un entorno local para que corra una aplicación con docker-compose.

Escenario:

Durante el sprint celebrado recientemente nuestro equipo nos asignó una tarea para desarrollar el archivo de build de una aplicación NestJS, esperan que para finalizar el sprint entreguemos el archivo Dockerfile funcional y un manifiesto de docker-compose que levante la aplicación y una base de datos MongoDB.

Nuestro aporte al equipo va a permitir que todos los desarrolladores que trabajen en el proyecto puedan poder levantar el mismo entorno en su entorno local.

La aplicación que va a ser manejada por este proceso se encuentra en el siguiente enlace:

https://github.com/edgaregonzalez/devops-bootcamp/tree/main/Desafios/Fase3/educacionit-app

Requisitos:

1. Elaborar el archivo Dockerfile con todas las instrucciones necesarias para utilizar la aplicación.

2. Entregar un archivo docker-compose.yaml que permita al desarrollador levantar un entorno de trabajo local con un simple comando.

3. Elaborar toda la documentación necesaria.

Lo primero que haremos será levantar un proyecto en NESTJS para poder trabajar con él.

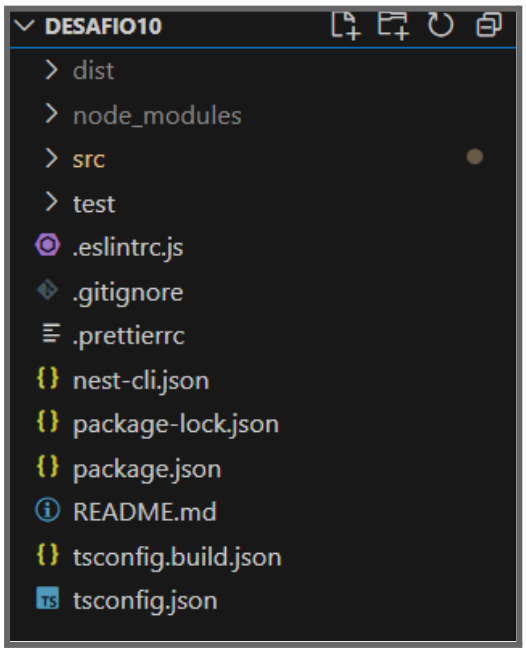
Configurar un nuevo proyecto es bastante sencillo con la CLI de Nest .

Con npm instalado, podemos crear un nuevo proyecto de Nest con los siguientes comandos en la terminal de tu sistema operativo:

- npm i -g @nestjs/cli
- nest new project-name (reemplazamos name por el nombre del proyecto)

Creamos el proyecto y se instalarán los módulos de nodo y algunos otros archivos repetitivos, y se creará un directorio que se completará con varios archivos principales.

Podremos ver algo como esto.



Podemos correr nuestra aplicación con cualquiera de los siguientes comandos.

$ npm start

 # development
$ npm run start

 # watch mode
$ npm run start:dev

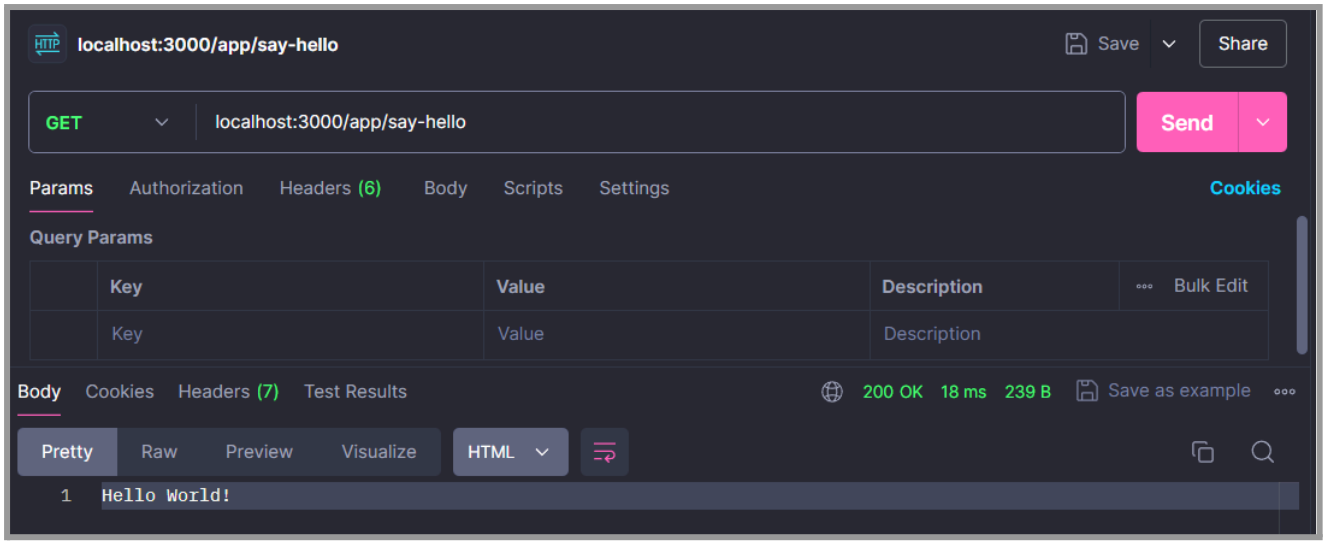
 # production mode
$ npm run start:prod

Una vez que tenemos nuestra aplicación corriendo podemos ver este mensaje en la terminal.

```
PS C:\Users\CPU\Desktop\devops\devops-bootcamp\educacionitdevops\desafio10> npm start

> nestjs-desafio10@0.0.1 start
> nest start

[Nest] 13092  - 27/08/2024, 06:46:48     LOG [NestFactory] Starting Nest application...
[Nest] 13092  - 27/08/2024, 06:46:48     LOG [InstanceLoader] AppModule dependencies initialized +10ms
[Nest] 13092  - 27/08/2024, 06:46:48     LOG [RoutesResolver] AppController {/app}: +6ms
[Nest] 13092  - 27/08/2024, 06:46:48     LOG [RouterExplorer] Mapped {/app/say-hello, GET} route +3ms
[Nest] 13092  - 27/08/2024, 06:46:48     LOG [NestApplication] Nest application successfully started +2ms
```

Para comprobarlo genero una petición en postman.

Podemos ver cómo nos devuelve un “ Hello World! “.



Los archivos generados contienen la siguiente información.

La carpeta **src/**: Contiene el código fuente de la aplicación.

La carpeta **dist/:** Contiene los archivos compilados de la aplicación. Al usar TypeScript, el código fuente en src se transpila a JavaScript y se coloca en dist.

La carpeta **node_modules/:** Contiene todas las dependencias del proyecto.

La carpeta **test/**: Contiene los archivos de prueba que se utilizan para verificar que la aplicación funcione correctamente.

A nivel raíz tenemos:

package.json: Define las dependencias del proyecto y scripts de npm.

tsconfig.json: Configuración del compilador TypeScript.

init-mongo.js: Script de inicialización, crea una base de datos y una colección con un único documento.

A continuación crearemos el archivo Dockerfile y Docker-compose para completar la tarea que se nos asignó.

Nuestro Dockerfile contiene los siguientes pasos.

```
# Usamos la imagen base de Node.js
FROM node:18

# Creamos un directorio de trabajo en el contenedor
WORKDIR /usr/src/app

# Copiamos los archivos de configuración de dependencias
COPY package*.json ./

# Instalamos las dependencias de la aplicación
RUN npm install

# Copiamos el código fuente de la aplicación al contenedor
COPY . .

# Construir la aplicación
RUN npm run build

# Exponemos el puerto en el que la aplicación escuchará
EXPOSE 3000

# Comandos para iniciar la aplicación
CMD ["npm", "run", "start:prod"]
```

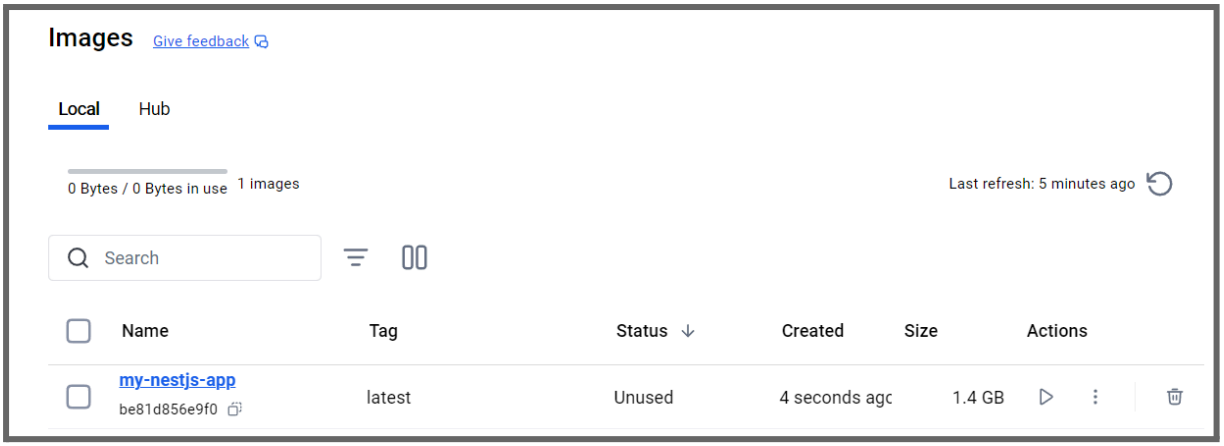
Dockerfile > ...

```dockerfile
# Usar la imagen base de Node.js
FROM node:18

# Crear un directorio de trabajo en el contenedor
WORKDIR /usr/src/app

# Copiar los archivos de configuración de dependencias
COPY package*.json ./

# Instalar las dependencias de la aplicación
RUN npm install

# Copiar el código fuente de la aplicación al contenedor
COPY . .

# Construir la aplicación
RUN npm run build

# Exponer el puerto en el que la aplicación escuchará
EXPOSE 3000

# Comando para iniciar la aplicación
CMD ["npm", "run", "start:dev"]
```

Creamos nuestra imagen usando el comando **Docker build -t my-nestjs-app .**

(Reemplazar **my-nestjs-app** por el nombre que le pongan a la imagen)

Vamos a nuestro docker desktop a verificar que la imagen se creó correctamente.



A continuación crearemos el archivo **Docker-compose**.

Nuestro Docker-compose contiene los siguientes pasos.

docker-compose.yml

```yaml
version: '3'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - mongo
    environment:
      MONGO_URI: mongodb://ignacio:desafio10@mongo:27017/educacionit?authSource=admin
    networks:
      - app-network

  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    networks:
      - app-network
    environment:
      MONGO_INITDB_ROOT_USERNAME: ignacio
      MONGO_INITDB_ROOT_PASSWORD: desafio10
    volumes:
      - ./init-mongo.js:/docker-entrypoint-initdb.d/init-mongo.js

networks:
  app-network:
    driver: bridge
```

version: “3”: Especifica la versión del formato de archivo de Docker Compose.

services: Define los servicios que Docker Compose debe ejecutar. Cada servicio se ejecutará en su propio contenedor Docker.

app: Nombre del servicio para tu aplicación NestJS.

build

context: Define el contexto de construcción, que es el directorio donde Docker buscará el Dockerfile. En este caso, es el directorio actual (.).

dockerfile: Especifica el archivo Dockerfile que Docker debe usar para construir la imagen. Si el nombre del archivo es diferente, cámbialo aquí.

ports:"3000:3000": Mapea el puerto 3000 del contenedor al puerto 3000 del host. Esto significa que puedes acceder a la aplicación NestJS en http://localhost:3000.

depends_on:

mongo: Indica que el servicio app depende del servicio mongo. Docker Compose asegura que el contenedor de MongoDB se inicie antes que el contenedor de la aplicación.

environment:

MONGO_URI: Define una variable de entorno que la aplicación utilizará para conectarse a MongoDB. La URI incluye las credenciales (root:example) y especifica que se conecte a la base de datos educacionit usando la autenticación contra la base de datos admin.

networks:
app-network: Conecta el contenedor de la aplicación a una red personalizada, esto permite que los servicios se comuniquen entre sí.

mongo: Nombre del servicio para MongoDB.

image: mongo:latest - Usa la última versión de la imagen oficial de MongoDB desde Docker Hub.

ports: "27017:27017": Mapea el puerto 27017 del contenedor al puerto 27017 del host. Esto te permite conectarte a MongoDB en mongodb://localhost:27017.

networks:
app-network: Conecta el contenedor de MongoDB a la misma red app-network que la aplicación, permitiendo la comunicación entre ellos.

environment:

MONGO_INITDB_ROOT_USERNAME: root: Define el nombre de usuario para el usuario root de MongoDB.

MONGO_INITDB_ROOT_PASSWORD: example: Define la contraseña para el usuario root de MongoDB. Estas variables son utilizadas por MongoDB para crear el usuario root durante la inicialización del contenedor.

volumes:

- ./init-mongo.js:/docker-entrypoint-initdb.d/init-mongo.js:

Monta el archivo init-mongo.js.

Desde el host al contenedor en /docker-entrypoint-initdb.d/.

MongoDB ejecutará automáticamente todos los scripts .js en este directorio durante el arranque, permitiendo la inicialización de la base de datos con datos predeterminados.

networks:

Define redes personalizadas para los servicios.

app-network:

driver: bridge: Especifica el controlador de red bridge, que es el tipo de red predeterminado en Docker. Permite que los contenedores se comuniquen entre sí dentro de la misma red.

Una vez configurado nuestro **docker-compose** y el **Dockerfile** lo siguiente es levantar el docker compose.

Para eso utilizamos el comando **docker-compose up --build.**

Docker-compose up levantará todos los servicios definidos en tu archivo docker-compose.yml.

Docker Compose utilizará la imagen construida para el servicio de la aplicación NestJS y configurará MongoDB con las variables de entorno proporcionadas.

Esto levantara la app en http://localhost:3000 y la base de datos en mongodb://localhost:27017

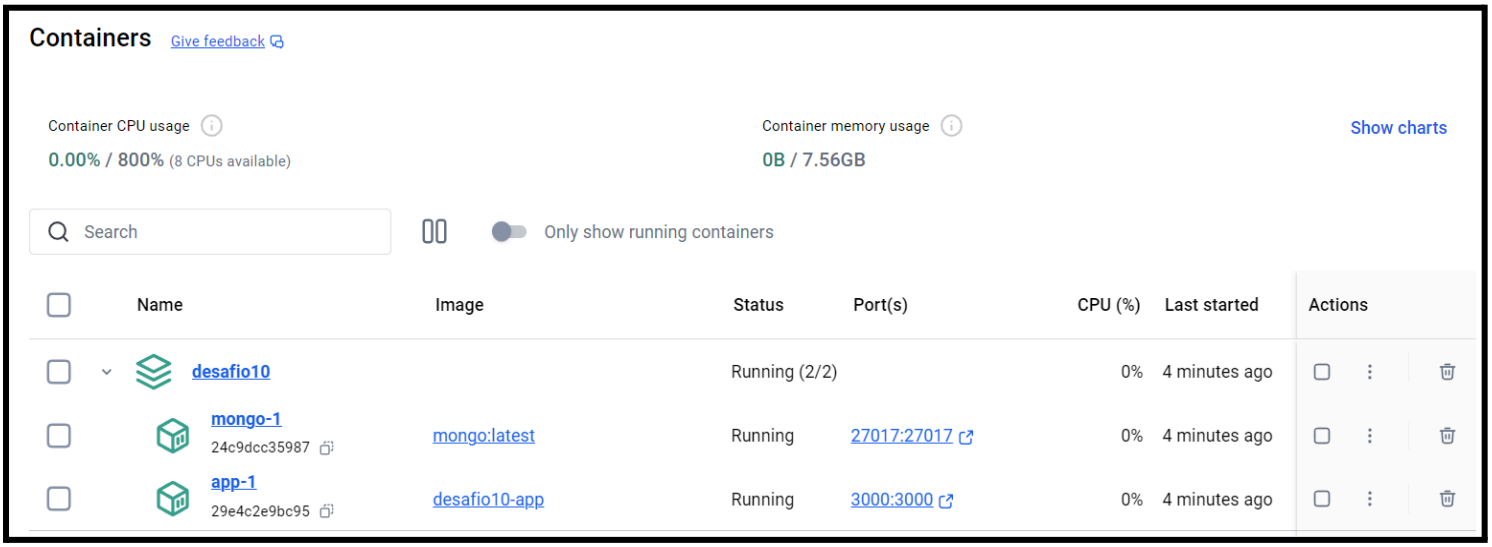
Podemos verificar que los contenedores están funcionando correctamente con:

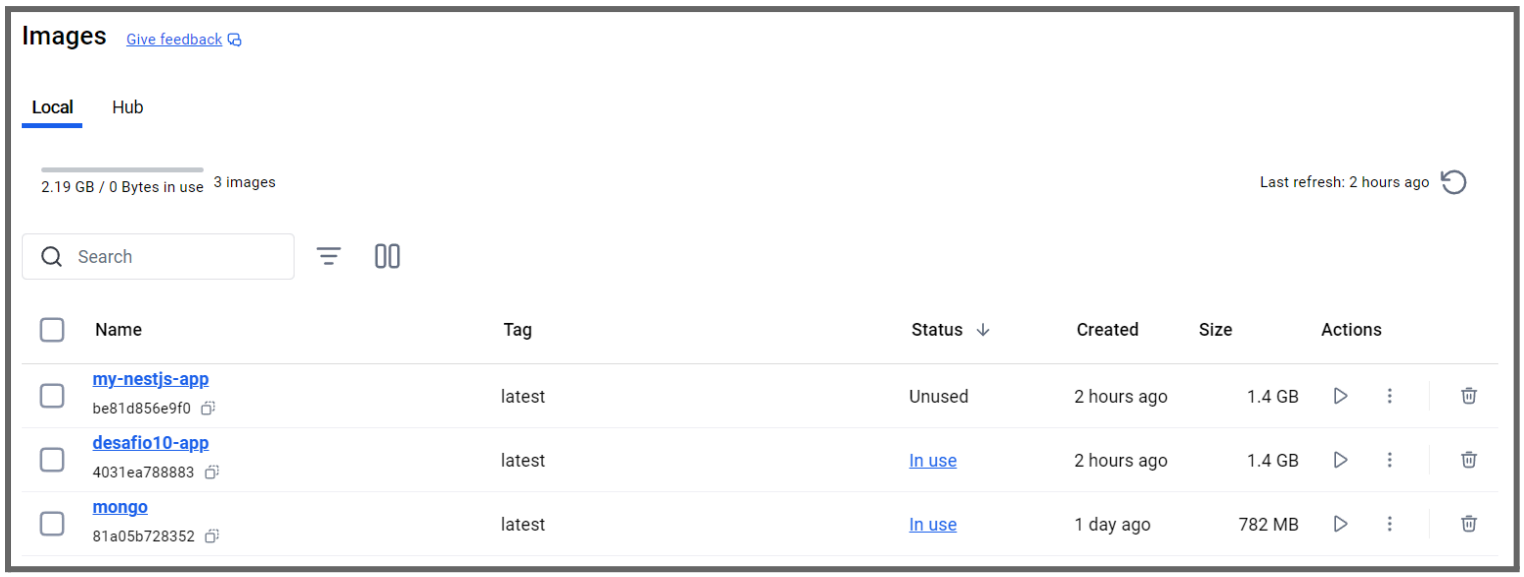
docker-compose ps.

Una vez terminado esto vamos a tener a nuestra app conectada con nuestra base de datos

Podemos verificar la creación del contenedor en Docker Desktop

Este lo mostrará como un stack





Inicio de la aplicación.

app-1 | [1:21:16 AM] Found 0 errors. Watching for file changes.
app-1 |
app-1 | [Nest] 36 - 08/28/2024, 1:21:17 AM LOG [NestFactory] Starting Nest application...
app-1 | [Nest] 36 - 08/28/2024, 1:21:17 AM LOG [InstanceLoader] AppModule dependencies initialized +31ms
app-1 | [Nest] 36 - 08/28/2024, 1:21:17 AM LOG [RoutesResolver] AppController {/app}: +8ms
app-1 | [Nest] 36 - 08/28/2024, 1:21:17 AM LOG [RouterExplorer] Mapped {/app/say-hello, GET} route +7ms
app-1 | [Nest] 36 - 08/28/2024, 1:21:17 AM LOG [NestApplication] Nest application successfully started +3ms

MongoDB logs. (Para ver los logs podemos ejecutar el comando **docker-compose logs mongo**)

mongo-1 | {"t":{"$date":"2024-08-28T01:21:09.522+00:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":29,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"24c9dcc35987"}}

mongo-1 | {"t":{"$date":"2024-08-28T02:41:44.281+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"/tmp/mongodb-27017.sock"}}

mongo-1 | {"t":{"$date":"2024-08-28T02:41:44.281+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"0.0.0.0"}}
mongo-1 | {"t":{"$date":"2024-08-28T02:41:44.281+00:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"listener","msg":"Waiting for connections","attr":{"port":27017,"ssl":"off"}}

mongo-1 | {"t":{"$date":"2024-08-28T02:41:44.281+00:00"},"s":"I", "c":"CONTROL", "id":8423403, "ctx":"initandlisten","msg":"mongod startup complete","attr":{"Summary of time elapsed":{"Startup from clean shutdown?":true,"Statistics":{"Transport layer setup":"0 ms","Run initial syncer crash recovery":"0 ms","Create storage engine lock file in the data directory":"0 ms","Get metadata describing storage engine":"0 ms","Validate options in metadata against current startup options":"0 ms","Create storage engine":"831 ms","Write current PID to file":"10 ms","Initialize FCV before rebuilding indexes":"6 ms","Drop abandoned idents and get back indexes that need to be rebuilt or builds that need to be restarted":"0 ms","Rebuild indexes for collections":"0 ms","Load cluster parameters from disk for a standalone":"0 ms","Build user and roles graph":"0 ms","Verify indexes for admin.system.users collection":"0 ms","Set up the background thread pool responsible for waiting for opTimes to be majority committed":"0 ms","Initialize information needed to make a mongod instance shard aware":"0 ms","Start up the replication coordinator":"3 ms","Start transport layer":"0 ms","_initAndListen
total elapsed time":"868 ms"}}}}

mongo-1 | {"t":{"$date":"2024-08-28T02:42:24.196+00:00"},"s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"Connection accepted","attr":{"remote":"172.18.0.1:38702","uuid":{"uuid":{"$uuid":"107be965-7dcc-4265-8a65-7026706a8102"}},"connectionId":1,"connectionCount":1}}

mongo-1 | {"t":{"$date":"2024-08-28T02:42:24.197+00:00"},"s":"I", "c":"NETWORK", "id":22943, "ctx":"listener","msg":"Connection accepted","attr"

```
PS C:\Users\CPU\Desktop\devops\devops-bootcamp\educacionitdevops\desafio10> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                     NAMES
29e4c2e9bc95   desafio10-app  "docker-entrypoint.s…"   10 minutes ago  Up 10 minutes  0.0.0.0:3000->3000/tcp    desafio10-app-1
24c9dcc35987   mongo:latest   "docker-entrypoint.s…"   10 minutes ago  Up 10 minutes  0.0.0.0:27017->27017/tcp  desafio10-mongo-1
[1:21:12 AM] Starting compilation in watch mode...
app-1    |
app-1    | [1:21:16 AM] Found 0 errors. Watching for file changes.
app-1    |
app-1    | [Nest] 36  - 08/28/2024, 1:21:17 AM     LOG [NestFactory] Starting Nest application...
app-1    | [Nest] 36  - 08/28/2024, 1:21:17 AM     LOG [InstanceLoader] AppModule dependencies initialized +31ms
app-1    | [Nest] 36  - 08/28/2024, 1:21:17 AM     LOG [RoutesResolver] AppController {/app}: +8ms
app-1    | [Nest] 36  - 08/28/2024, 1:21:17 AM     LOG [RouterExplorer] Mapped {/app/say-hello, GET} route +7ms
app-1    | [Nest] 36  - 08/28/2024, 1:21:17 AM     LOG [NestApplication] Nest application successfully started +3ms
mongo-1  | about to fork child process, waiting until server is ready for connections.
mongo-1  | forked process: 29
mongo-1  |
mongo-1  | {"t":{"$date":"2024-08-28T01:21:09.512+00:00"},"s":"I",  "c":"CONTROL",  "id":20698,   "ctx":"main","msg":"***** SERVER RESTARTED ****
```

Con esto corriendo ya tenemos desarrollado un build en docker que nos configura un entorno local para correr una aplicación con docker-compose.