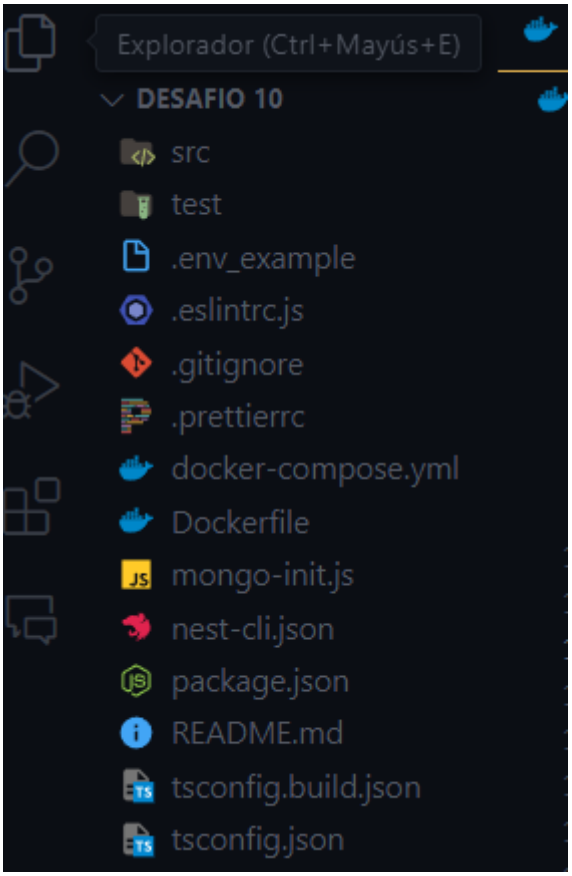
Desafío 10

1- Comenzamos el desafío 10 creando un **Dockerfile** específico para la aplicación. Luego, integraremos esa imagen en un archivo **Docker Compose**, junto con la imagen de nuestra base de datos. La estructura que utilizaremos es la siguiente



src/: Directorio que contiene el código fuente de la aplicación.
test/: Directorio que contiene los archivos de pruebas automatizadas.
.env_example: Archivo de ejemplo que muestra las variables de entorno necesarias para el proyecto.
docker-compose.yaml: Archivo que define los servicios **Docker** (aplicación y base de datos).
Dockerfile: Archivo que da instrucciones para construir la imagen, **Docker** de la aplicación.
mongo-init.js: Script que se utiliza para inicializar **MongoDB** con datos iniciales.
package.json: Archivo que define las dependencias del proyecto y scripts de npm.
tsconfig.json: Archivo de configuración del compilador **TypeScript.**

2- Seguidamente seguimos con la creación de nuestro archivo **Dockerfile**, al cual le daremos las instrucciones para la creación de nuestra aplicación.

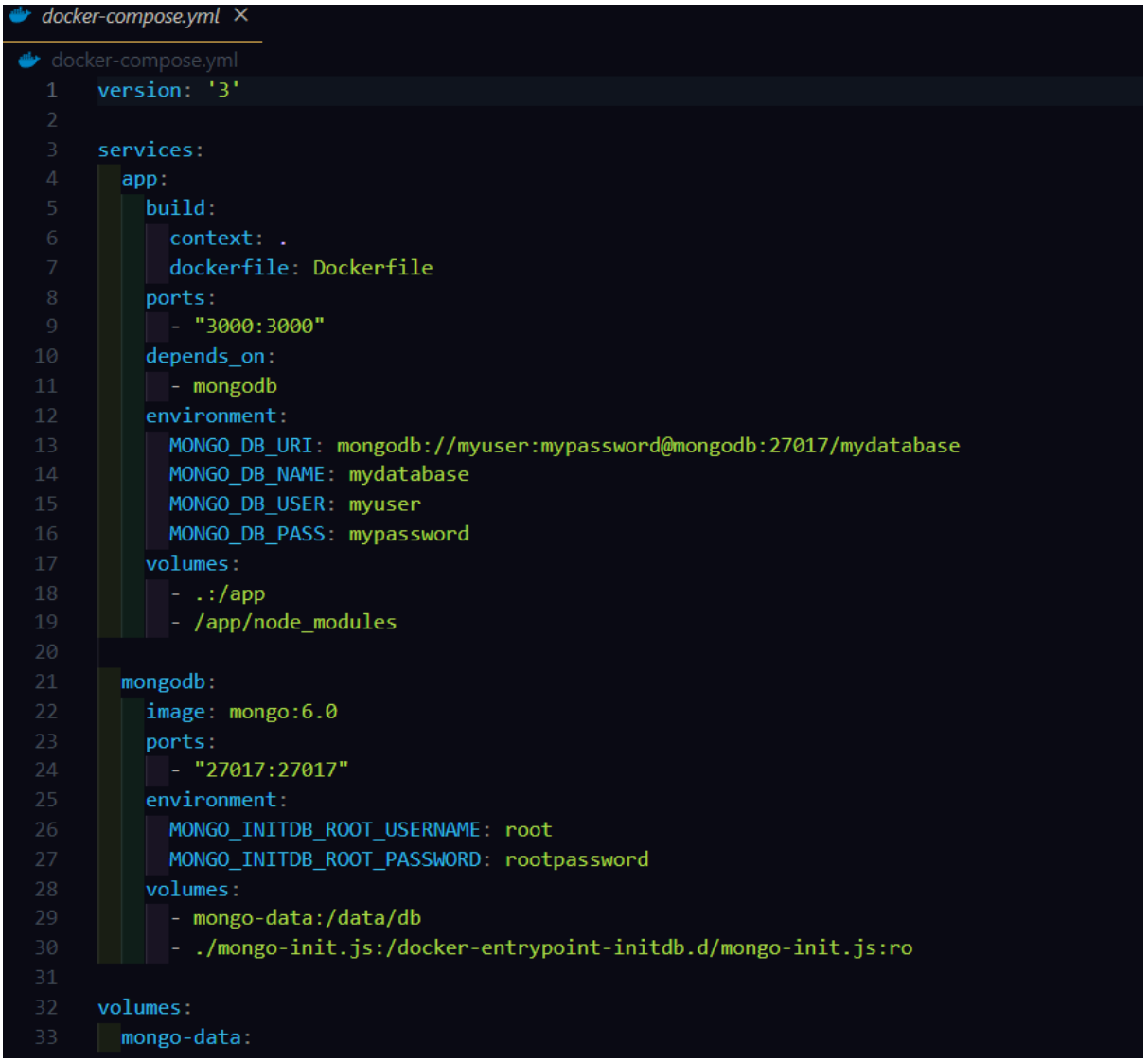
Dockerfile

```dockerfile
# imagen Node.js
FROM node:18-alpine AS builder

# directorio de trabajo
WORKDIR /app

# Copia archivos package.json  package-lock.json
COPY package*.json ./

# Instala las dependencias
RUN npm install

#Instala typescript globalmente en la etapa de construccion
RUN npm install -g typescript

# Copia archivos de la aplicación
COPY . .

#compila typescript a javascript
RUN tsc

#Etapa 2: imagen final
FROM node:18-alpine

#Establece directorio de trabajo
WORKDIR /app

#Copia los archivos necesarios en etapa de construccion
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package*json ./

# Expone el puerto en el que corre la aplicación
EXPOSE 3000

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

Etapas del Dockerfile

1. **builder:** instala Node.js, configura las dependencias y compila el código **TypeScript.**
2. **runtime:** Utiliza una imagen ligera con **Node.js** y los archivos que necesita para ejecutarla.

3- El siguiente paso es la creación de nuestro archivo **docker-compose.yml**



Version '3': Version del formato del archivo de Docker compose
app: Servicio que define la aplicación Node.js.

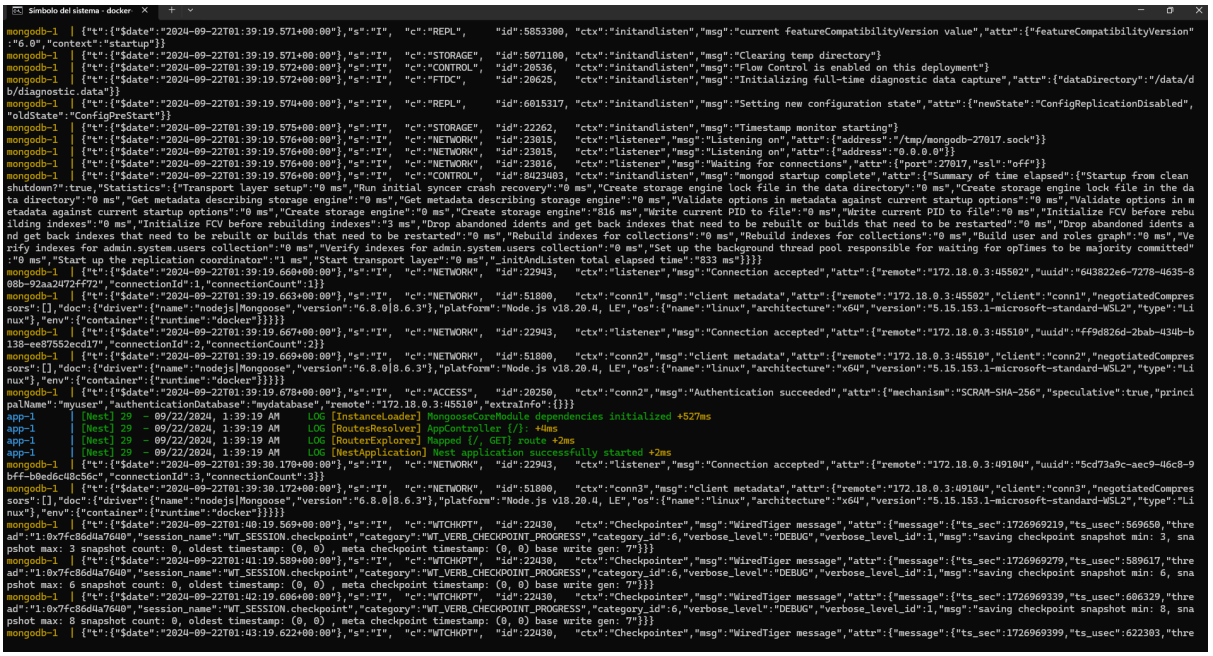
- **depends_on:** Garantiza que MongoDB se ejecute antes de iniciar la aplicación.
- **MONGO_URI:** La URI de conexión para MongoDB.

mongodb: Servicio que define la base de datos MongoDB.

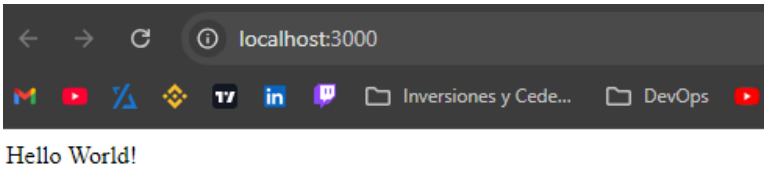
- **volumes:** Persisten los datos de MongoDB y permiten la inicialización mediante mongo-init.js.

4- Dentro del docker compose agregamos un archivo js para iniciar la base de datos con un usuario y pass, llamado **mongo-init.js**

5- Al tener nuestro **docker.compose.yml** finalmente configurado, junto con **Dockerfile** de nuestra app y el archivo para iniciar nuestra base de datos, seguimos con el levantamiento de nuestra aplicación usando docker compose y lo hacemos con el comando: **docker-compose up --build.** Al ejecutar ese comando en la terminal de Powershell crea las imágenes e inicia los contenedores.



6- cuando termine de cargar todo en nuestra terminal de powershell nos dirigimos hacia nuestra dirección ip seguida del puerto 3000 http://localhost:3000



Hello World!

7- Dentro de nuestra pc local en la aplicación de Docker Vemos que los contenedores están en estado de Running.