Arquitectura Aplicacion Movil

Manuel Añel García

7 de diciembre de 2025

1. Introduction

En este documento se va a documentar la dockerizacion de una API de gestion de usuarios y grupos y su subida a docker hub.

2. Tecnologías utilizadas

- NodeJS con Express: Para el backend de la aplicación (Servidor)
- Mongo: Para la base de datos
- Moongose: Es un paquete para la conexion entre el backend y la base de datos
- Coors: Para la seguridad de las peticiones HTTP
- Docker: La plataforma con la que vamos a trabajar para subir la API

3. Arquitectura de la aplicacion

Arquitectura Backend (API REST) en contenedores

- Servidor (Node.js): Maneja las rutas, realiza operaciones en la base de datos.
- Base de datos (MongoDB): Almacena los datos de los usuarios y grupos.

4. Backend

Ahora se va a explicar el funcionamiento de la API en cada caso

4.1. Conexion a Mongo

El backend se conecta mediante el paquete moongose en el backend haciendo un moongose.connect() y dentro de los parentesis la direccion del servidor que tenemos creado

```
let direccion = direccion_mongo;

mongoose.connect(direccion)
.then(() => console.log('Conectado a la base de datos'))
.catch(err => console.error('Error de conexion a la base de datos:', err));
```

4.2. Servidor Express

En este caso se ve como se crea el servidor mediante express con app = express() y se crean los endpoints que son las peticiones que se van a hacer desde la aplicacion al servidor para que se ejecuten, luego para desplegarlo se usa app.listen(), esto es igual para los grupos

- GET: Es para recojer todos los usuarios que hay en la base de datos
- POST: Mediante el cuerpo de un usuarios (sus parametros: nombre, apellido etc..) se recoje y se guarda el usuario en la base de datos con esos parametros
- PUT: Basicamente mediante un id (el de mongo por defecto) se busca el usuario y mediante un cuerpo como en el anterior que se especifica en la aplicacion de modifica el usuario ya existente
- DELETE: Mediante un id pasado por la aplicacion se elimina el usuario con el que concuerda

```
app.get('/usuarios', async (req, res) => {
  const usuarios = await Usuario.find();
  res.json(usuarios);
});

app.post('/usuarios', async (req, res) => {
  const nueva = new Usuario(req.body);
  await nueva.save();
  res.json(nueva);
});

app.put('/usuarios/:id', async (req, res) => {
  const actualizada = await Usuario.findOneAndUpdate({ id: Number(req.params.id) },
      req.body, { new: true });
  res.json(actualizada);
});

app.delete('/usuarios/:id', async (req, res) => {
  await Usuario.findOneAndDelete({ id: Number(req.params.id) });
});
```

4.3. Variables de entorno

Para la seguridad de mi API creé variables de entorno que básicamente la informacion delicada esta en un archivo .env y en donde deberia estar tiene una variable definida en el .env y eso no se sube a docker y luego cuando lo inicies sin eso se puede poner los valores que quieras por consola

Esto es lo que tenemos en el .env

```
MONGO_URI='mongodb://localhost:27017/GestionUG'
```

```
require('dotenv').config();
const direccion_mongo = process.env.MONGO_URI;
```

5. Docker

Vamos ahora a explicar como dockerizar la API

5.1. Crear Dockerfile

Sirve para que siga los pasos para construir la imagen despues

```
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "server.js"]
```

5.2. Crear la imagen

En la carpeta donde está el Dockerfile hay que ejecutar este comando

```
docker build -t manu8944/api-docker:v1.0
```

5.3. Subir la imagen a Dockerhub

Ponemos los siguientes comandos para subirla imagen que creamos antes a docker hub

```
docker login
docker push manu8944/api-docker:v1.0
```

5.4. Docker compose

El docker compose es un archivo para crear el contenedor en nuestro dockerdesktop

```
version: "3.9"
services:
  backend:
    image: ejemplo-api:v1.0
    ports:
      - "3000:3000"
    environment:
      - MONGO_URI=mongodb://localhost:27017/GestionUG
    depends_on:
      - db

  db:
    image: mongo
    ports:
      -"27017:27017"
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
```

Lo que se menciona en el código son las imágenes para subir, los puertos usados, url de conexion a la bd etc..

5.5. Construir contenedor con DockerCompose

Es para crear el contenedor localmente

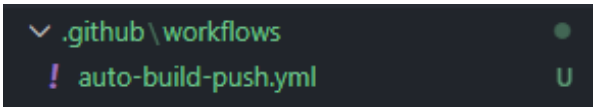
```
docker compose up -d
```

6. Automatizar subida

Ahora vamos a ver como se hace para que al hacer push en nuestro repositorio de github automáticamente se haga un push de la imagen en docker hub

6.1. Crear archivo

Creamos esta estructura de carpetas con el archivo



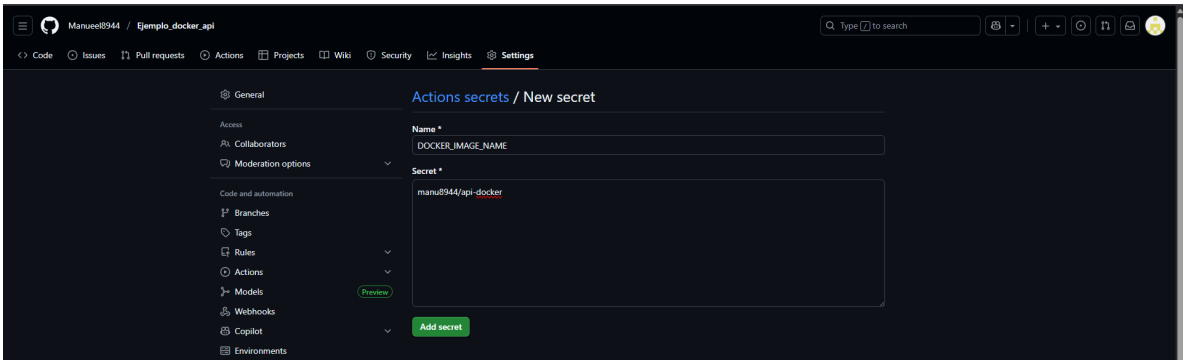
Ponemos esto para que se haga el push a DH cuando se haga un commit

```yaml
name: auto-build-push

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout código
        uses: actions/checkout@v4

      - name: Configurar Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login a Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Construir y subir imagen Docker
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: |
            ${{ secrets.DOCKER_IMAGE_NAME }}:latest
```

6.2. Secrets de github

Ahora en el repositorio hay que poner los secrets que son básicamente las variables de entorno que vimos en el código anterior.

Asi se hacen los secrets:



Asi es como quedan todos los secrets: