Documentación de la api Docker

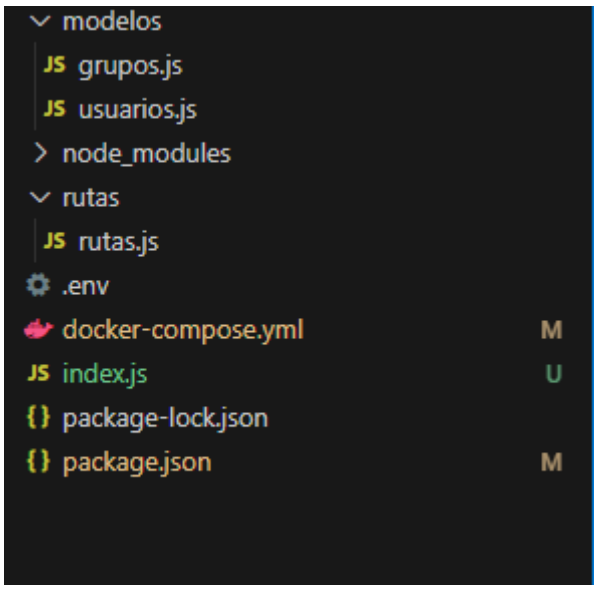
Andrea Sofía Pais Dos Santos

November 28, 2025

1 Índice

2 Creación de la API con base de datos local

Para construir una API con MongoDB Local usando docker, primero vamos a inicializar el proyecto "npm init -y" y crear, por ahora, la siguiente estructura de carpetas y archivos. Luego para la creación de una imagen vamos a tener que crear otros documentos extras.



Desglose de los archivos necesarios junto a su contenido.

- Archivo index.js, contiene las importaciones y la conexión a la base de datos:

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import mongoose from "mongoose";
import rutas from "./rutas/rutas.js";

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

app.use(rutas);
const uri = process.env.MONGO_URI;
export async function conectarBD(){
    try{
        await mongoose.connect(uri,{});
        console.log("Base de datos conectada");
    }catch(error){
        console.log("Error conectandose a la base de datos", error);
    }
}

const PORT = process.env.PORT || 3000;

conectarBD().then(async () => {
    app.listen(PORT, () => console.log(`API en ${PORT}`));
});
```

- Archivo rutas.js, está compuesto por los EndPoints (GET,POST). GET para obtener los usuarios y los grupos y POST para crear esos usuarios y grupos.

```
import express from "express";
import Usuario from "../modelos/usuarios.js";
import Grupo from "../modelos/grupos.js";

const router = express.Router();

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

router.get("/grupos", async (req, res) => {
  const grupos = await Grupo.find();
  res.json(grupos);
});

router.post("/usuarios", async (req, res) => {
  const nuevoUsuario = new Usuario(req.body);
  await nuevoUsuario.save();
  res.json(nuevoUsuario);
});

router.post("/grupos", async (req, res) => {
  const nuevoGrupo = new Grupo(req.body);
  await nuevoGrupo.save();
  res.json(nuevoGrupo);
});

router.get("/", (req, res) => {
  res.json({ mensaje: "API funcionando correctamente" });
});

export default router;
```

- Carpeta "modelos" con grupos.js y usuarios.js, que contienen el esquema con las propiedades de ambos elementos.

```
\\\\\\\\\\\\\\\\\\\\ Archivo grupos \\\\\\\\\\\\\\\\\\\\\\\\\\\\
import mongoose from "mongoose";

const GrupoEschema = new mongoose.Schema({
    nombre_grupo: {
        type: String,
        required: true
    },
    participantes: {
        type: Array,
        required: true
    }
});

export default mongoose.model("Grupo", GrupoEschema);


```

```
\\\\\\\\\\\\\\\\\\\\\\ Archivo usuarios \\\\\\\\\\\\\\\\\\\\\\\\\\\\

import mongoose from "mongoose";

const UsuarioEschema = new mongoose.Schema({
    nombre_user: {
        type: String,
        required: true
    },
    apellido_user: {
        type: String,
        required: true
    },
    edad_user: {
        type: Number,
        required: true
    },
    dni_user : {
        type: String,
        required: true
    }
});

export default mongoose.model("Usuario", UsuarioEschema);
```

- Archivo .env es el archivo de configuración que almacena datos delicados y el puerto que usaremos en este caso el 3000.

```
MONGO_URI=mongodb://<nombre>:<contrasena>@localhost:27017/mi_base_datos?admin
PORT=3000
```

- Archivo docker-compose.yml contiene unos servicios definidos, servicio "api"(nuestra aplicación) y servicio "db"(la base de datos MongoDB).

```
version: '3.8'

services:
  api:
    image: ejemplo-api:v1.0.0
    ports:
      - "3000:3000"
    environment:
      - MONGO_URI=mongodb://nombre:contrasena@db:27017/admin
    depends_on:
      - db
  db:
    image: mongo:latest
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=nombre
      - MONGO_INITDB_ROOT_PASSWORD=contrasena
    volumes:
      - mongo_data:/data/db
    restart: unless-stopped

volumes:
  mongo_data:
```

Tras tener la API estructurada, la construimos con el siguiente comando:

```
docker-compoose up
```

Ahora en Docker deberíamos de ver al API creada.

Hay que acordarse de que para utilizarlos hay que importar todo lo necesario, con el siguiente comando:

```
npm install express mongoose .... "todo lo que necesitemos"
```

3 Creación de la imagen

Para la construcción de la nueva imagen creamos los siguientes archivos y modificamos el docker-compose.yml.

- Archivo Dockerfile va a contener las instrucciones que sirven para crear la imagen.

```
FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 3000
CMD ["node","index.js"]
```

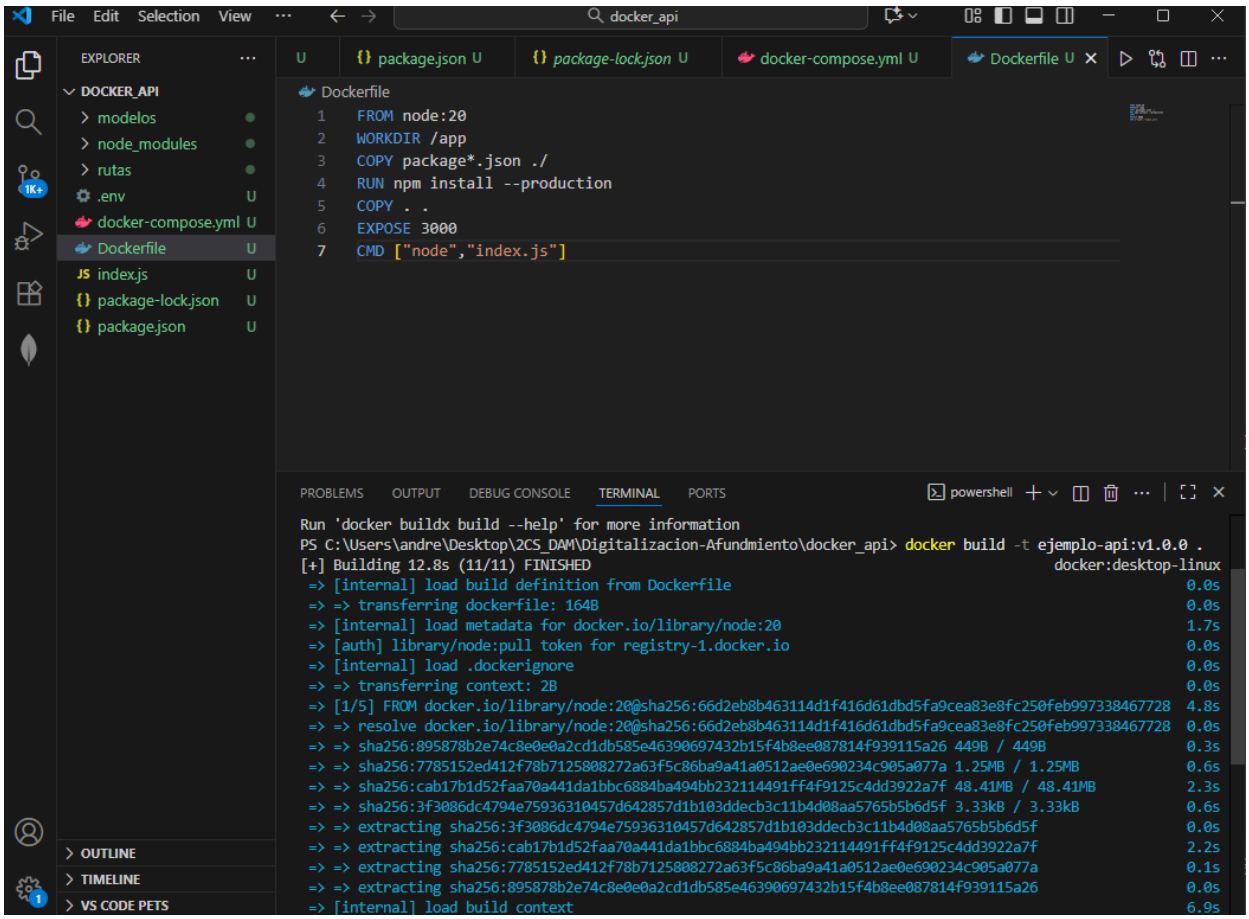
  Lo que nos quiere cedir esto es que desde node vamos a crear "app" que es donde vamos a trabajar desde ahora, se copiará el package*.json, va a ejecutar esa instalación compiando todo en el puerto 300.

- Archivo .dockerignore va a ignorar archivos pesados e información sensible.

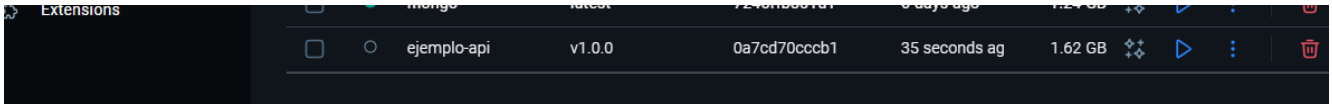
```
node_modules
npm-debug.log
.git
.gitignore
.env
docker-compose.yml
```

Tras tener esos archivos y hayamos modificado el docker-compose cambiando el nombre de la imagen, vamos a construir la imagen, para ellos usamos el siguiente comando:

```
docker build -t ejemplo-api:v1.0.0 .
```

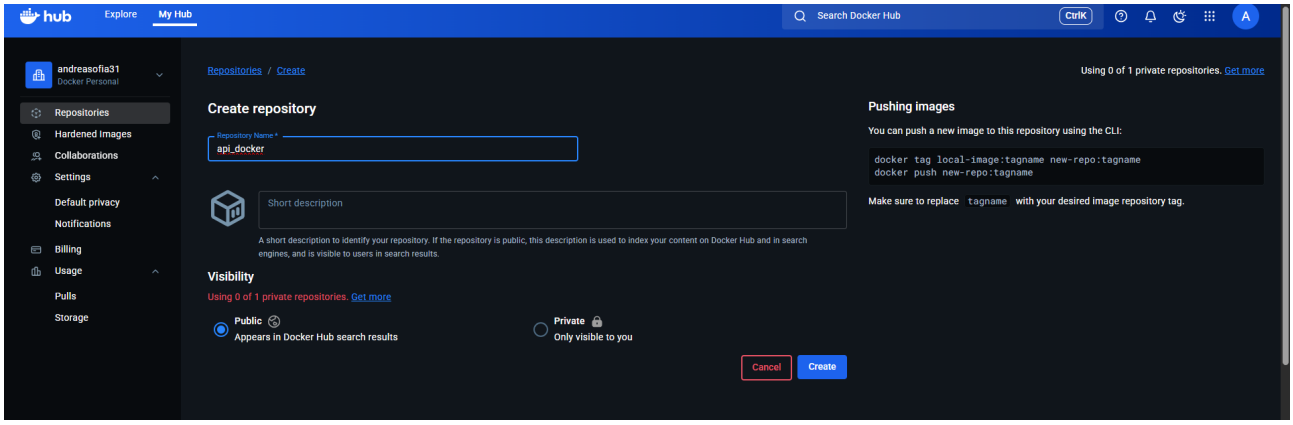


Ahora al acceder a docker, en images tendremos lo siguiente:



4 Subir la imagen a DockerHub

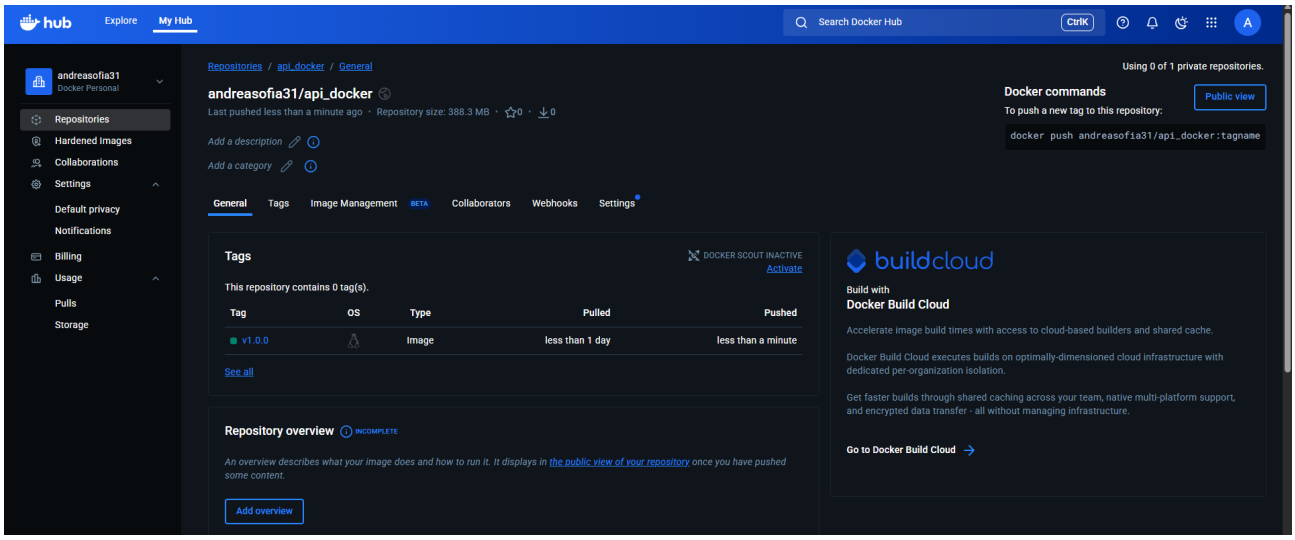
Para subir la imagen a DockerHub, lo primero es tener una cuenta e iniciar sesión. Tras iniciar sesión, creamos un repositorio con el nombre que queramos.



Y desde la terminal, ejecutamos el siguiente comando para que suba la imagen:

```
1 docker push andreasofia31/api_docker:v1.0.0
```

Si está correctamente subida, en el repositorio de DockerHub debería salir la imagen que hemos creada:



Automaticaciín y interaccion continua CI/CD