Programación de Servicios y Procesos: Práctica de Docker Local

Álvaro Del Valle Fernández

December 7, 2025

1 Introducción

El objetivo de esta práctica era crear una base de datos local, en este documento mostraré como cree los contenedores, la API y la base de datos, usando Docker, Docker Hub,GitHub y notificaciones en Discord.
Tambien incluí un elemento .java para realizar pruebas y comprobar el correcto funcionamiento del sistema.

2 Requisitos Funcionales

2.1 Comunicación entre Servicios

- **RF-01**: Debe conectarse de forma automatica una vez iniciada
- **RF-02**: Debe notificar de posibles errores
- **RF-03**: Comunicacion mediante Docker

3 Requisitos No Funcionales

- **RNF-01**: Debe de funcionar de forma fluida y sin esperas demasiado largas.
- **RNF-02**: Debe de estar organizado de forma modular permitiendo mejoras de forma sencilla.

4 Arquitectura

El sistema se compone de los siguientes elementos:

- **Backend**: Api con Node.js y Express
- **Contenedorización**: Docker

- **Registro**: Docker Hub
- **CI/CD**: GitHub
- **Notificaciones**: Discord con Webhooks

4.1 Flujo de Datos

1. Al iniciar se conecta al puerto adecuado automaticamente.
2. La API gestiona la petición
3. La API se conecta a la base de datos mendiante Docker
4. Devuelve los resultados
5. La API devuelve la respuesta al usuario mediante Discord

5 Clases Principales

5.1 docker-push.yml

```
    name: Build and Push Docker Image

on:
  push:
    branches:
      - main
      - master
    paths:
      - 'api/**'
      - '.github/workflows/docker-push.yml'
  workflow_dispatch: # Permite ejecutar manualmente

env:
  DOCKER_IMAGE_NAME: ${{ secrets.DOCKER_USERNAME }}/ejemplo-api-docker
  DOCKER_TAG: latest

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

    steps:
    - name: Checkout codigo
      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.DOCKER_USERNAME }}
      password: ${{ secrets.DOCKER_PASSWORD }}

  - name: Construir y subir imagen Docker
    uses: docker/build-push-action@v5
    with:
      context: ./api
      file: ./api/Dockerfile
      push: true
      tags: |
        ${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}
        ${{ env.DOCKER_IMAGE_NAME }}:v1.0.0
      cache-from: type=registry,ref=${{ env.DOCKER_IMAGE_NAME }}:buildcache
      cache-to: type=inline

  - name: Notify Discord - SUCCESS
    if: success()
    uses: sarisia/actions-status-discord@v1
    with:
      webhook: ${{ secrets.DISCORD_WEBHOOK }}
      title: "DEPLOY EXITOSO - Docker Image"
      description: |
        **Imagen construida y subida correctamente**

        **Imagen:** `${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}`
        **Tag:** v1.0.0
        **Estado:** Success
        **URL:** https://hub.docker.com/r/${{ secrets.DOCKER_USERNAME }}/api-
      username: "Docker Deploy Bot"
      avatar_url: "https://cdn-icons-png.flaticon.com/512/919/919853.png"

  - name: Notify Discord - FAILURE
    if: failure()
    uses: sarisia/actions-status-discord@v1
    with:
      webhook: ${{ secrets.DISCORD_WEBHOOK }}
      title: "DEPLOY FALLIDO - Docker Image"
      description: |
        **Fallo al construir o subir la imagen Docker**

        **Imagen:** `${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}`
```

```
            **Estado:** Failed
            **Workflow:** ${{ github.workflow }}

            *Revisa los logs de GitHub Actions*
          username: "Docker Deploy Bot"
```

5.2 Archivo Dockerfile

```
    FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

5.3 Archivo package.json

```
  {
  "name": "ejemplo-api",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "pg": "^8.11.3",
    "dotenv": "^16.3.1",
    "cors": "^2.8.5",
    "express-validator": "^7.0.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

5.4 Archivo server.js

```
    const express = require('express');
  const app = express();
  app.use(express.json());
```

```
let usuarios = [
  { id: 1, username: 'admin', password: '123' },
  { id: 2, username: 'user1', password: '456' }
];
//webhook
//rutas http://localhost:3000/
app.get('/', (req, res) => {
  res.json({
    mensaje: 'API funcionando',
    endpoints: [
      'GET /users',
      'POST /users',
      'POST /login'
    ]
  });
});

//GET USUARIO
app.get('/users', (req, res) => {
  res.json({
    usuarios: usuarios.map(u => ({ id: u.id, username: u.username }))
  });
});

//CREAR USUARIO
app.post('/users', (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) {
    return res.status(400).json({ error: 'Faltan datos' });
  }

  const nuevoId = usuarios.length > 0 ? Math.max(...usuarios.map(u => u.id)) + 1
  const nuevoUsuario = { id: nuevoId, username, password };

  usuarios.push(nuevoUsuario);

  res.status(201).json({
    mensaje: 'Usuario creado',
    usuario: { id: nuevoId, username }
  });
});

app.post('/login', (req, res) => {
  const { username, password } = req.body;
```

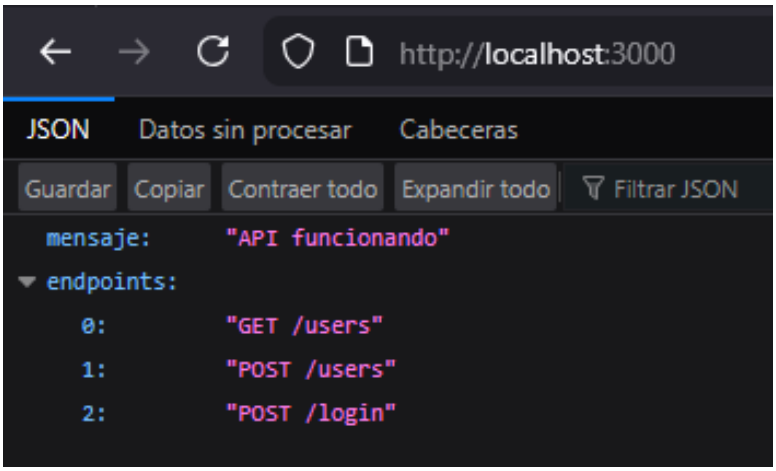
```
  const usuario = usuarios.find(u =>
    u.username === username && u.password === password
  );

  if (usuario) {
    res.json({
      mensaje: 'Login correcto',
      usuario: { id: usuario.id, username: usuario.username }
    });
  } else {
    res.status(401).json({ error: 'Credenciales incorrectas' });
  }
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor en http://localhost:${PORT}`);
});
```

6 Comandos Utilizados

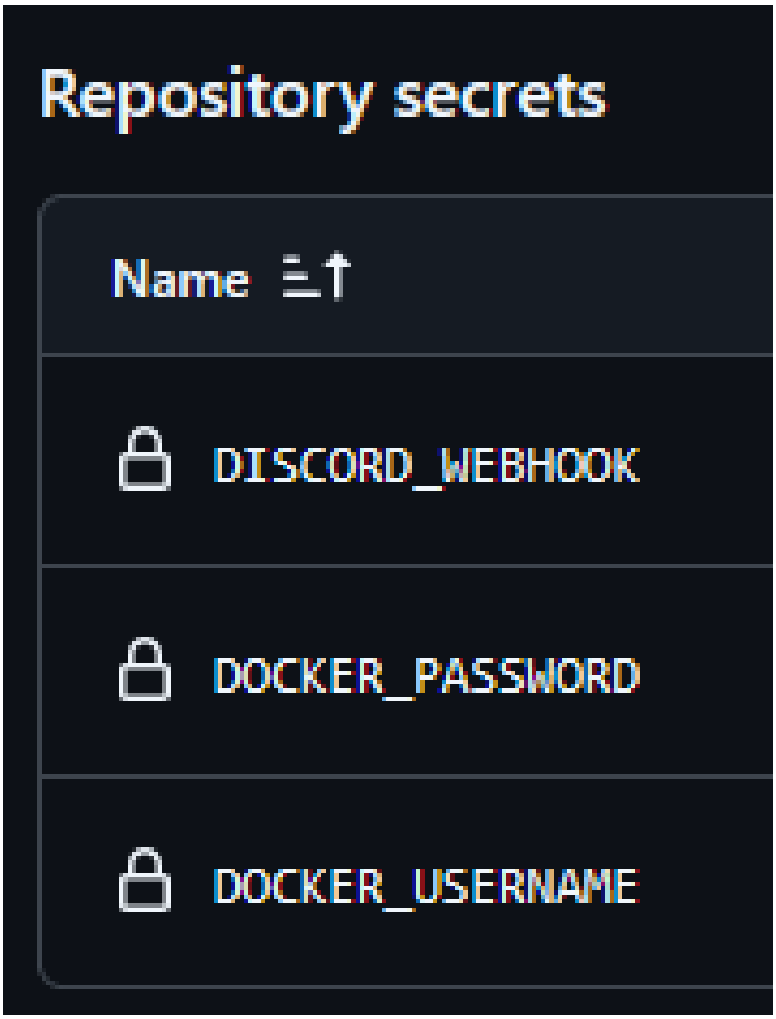
Para realizar esta practica utilize numerosos comandos, entre ellos:
Lanzar el server:
npm installnode server.js
Crear y correr el contenedor:
docker build -t prueba ./api
docker run -p 3000:3000 prueba
docker-compose up
docker-compose down

7 Capturas de Pantalla

Test de comprobacion del funcionamiento de la API:



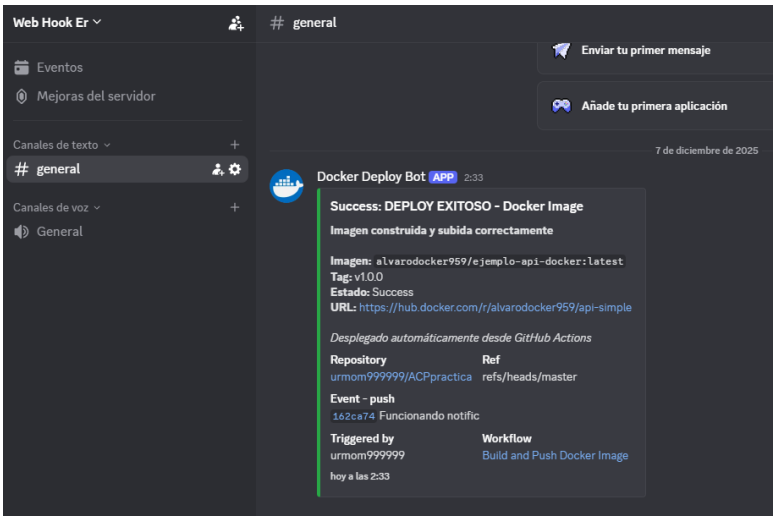
comprobacion de los secrets de GitHub:



Server corriendo localmente:

```
Node.js v22.12.0
PS C:\Users\delva\Desktop\Clases\SEGUNDO AÑO\segundo cuatrimestre\ACPpractica\api> node server.js
Servidor en http://localhost:3000
```

Notificación de comunicación correcta en Discord:



8 Tecnologías Usadas

- **Node** - Entorno de ejecucion JS
- **Express** - Framework
- **Docker** - Para crear contenedores
- **Docker Hub** - Registro de imágenes Docker
- **GitHub** - Para automatización CI/CD
- **Discord Webhooks** - Notificaciones