# **Documentación: API de Gestión de Usuarios y Tareas**

**Resumen** Este documento explica paso a paso cómo construir, ejecutar y desplegar la API de gestión de usuarios y sus tareas usando **Spring Boot**, **H2** como base de datos embebida y **Docker / Docker Compose** para contenerización. También incluye ejemplos de endpoints, configuración recomendada, ficheros necesarios y soluciones a problemas comunes.

## **1. Requisitos previos**

* Java JDK 17 (o la versión que use tu proyecto). Verificar con java -version.
* Maven instalado (mvn) o usar el wrapper mvnw incluido en el proyecto.
* Docker Desktop instalado y en funcionamiento.
* Código fuente del proyecto (por ejemplo, proyecto Spring Boot con pom.xml).

## **2. Estructura del proyecto (ejemplo)**

Demo/

├─ src/

│ ├─ main/

│ │ ├─ java/.../controller/

│ │ ├─ java/.../service/

│ │ ├─ java/.../entities/

│ │ ├─ java/.../repositories/

│ │ ├─ resources/

│ │ │ └─ application.properties

├─ pom.xml

├─ Dockerfile

├─ docker-compose.yml

├─ .dockerignore

└─ README.md

## **3. Configuración de la base de datos H2**

La base de datos utilizada por la aplicación es **H2**, una base de datos embebida en memoria ideal para desarrollo y pruebas. El proyecto define dos tablas principales:

* **User**: representa a los usuarios del sistema.
* **Task**: representa las tareas asociadas a cada usuario.

La relación entre ambas entidades es **uno a muchos (1:N)**, donde un **usuario puede tener múltiples tareas**, pero **cada tarea pertenece a un único usuario**.

Esta configuración se maneja automáticamente mediante las anotaciones JPA dentro de las entidades del proyecto (@OneToMany en User y @ManyToOne en Task), y H2 genera las tablas correspondientes en memoria al iniciar la aplicación.

La configuración mínima en src/main/resources/application.properties es:

* spring.datasource.url=jdbc:h2:mem:testdb
* spring.datasource.username=sa
* spring.datasource.password=
* spring.datasource.driver-class-name=org.h2.Driver
* spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
* spring.h2.console.enabled=true
* spring.h2.console.path=/h2-console

spring.jpa.hibernate.ddl-auto=update

## **4. Dockerfile (imagen para la API)**

Crea un Dockerfile en la raíz del proyecto. Ejemplo para una aplicación Spring Boot empaquetada en un jar:

# Dockerfile

FROM openjdk:17-jdk-slim

WORKDIR /app

# Copiamos el jar ya generado

COPY app.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

Asegúrate de que el nombre del .jar coincide con el real generado por Maven.

### **.dockerignore**

target/

.vscode/

.gradle/

.idea/

\*.iml

.git

## **5. Pasos para construir y ejecutar (localmente)**

### **5.1 Generar el JAR con Maven**

Si usas Maven localmente:

mvn clean package -DskipTests

Verifica que target/\*.jar existe.

### **5.2 Construir la imagen Docker**

docker build -t api-gestion-usuarios:latest .

### **5.3 Ejecutar con docker-compose**

docker-compose up -d --build

Comprueba logs:

docker-compose logs -f api

# o

docker logs -f api\_gestion\_usuarios

Probar la API (curl):

curl http://localhost:8080/api/users

Acceder a la consola H2:

* URL: http://localhost:8080/h2-console
* JDBC URL (por defecto): jdbc:h2:file:./data/demo-db
* Usuario: sa
* Contraseña: (vacía si no la pones)

## **6. Endpoints sugeridos (convención REST)**

A continuación una lista de endpoints típicos para una API de usuarios y tareas. Ajusta rutas y payloads según tu implementación real.

**Usuarios**

* GET /api/users — Listar usuarios
* GET /api/users/{id} — Obtener usuario por id
* POST /api/users — Crear usuario
* PUT /api/users/{id} — Actualizar usuario
* DELETE /api/users/{id} — Eliminar usuario
* GET /api/users/login - Comprueba que el user y el password sean correctos

**Tareas**

* GET /api/tasks — Listar tareas
* GET /api/tasks/{id} — Obtener tarea
* POST /api/tasks — Crear tarea
* PUT /api/tasks/{id} — Actualizar tarea
* DELETE /api/tasks/{id} — Eliminar tarea

**Ejemplo para crear usuario**

http://localhost:8080/api/users

'{"name":"Juan","email":"juan@example.com",”password”:”pass+w0rd}

## **7. Comandos útiles Docker/Maven**

* mvn clean package
* docker build -t springboot-api-h2 .
* docker run -d --name springboot-api -p 8080:8080 springboot-api-h2
* docker ps / docker logs -f <container>

## **8. Anexos: ejemplos de payloads**

**Usuario (POST /api/users)**

{

"name": "Juan Pérez",

"email": "juan@example.com",

"password": "secret123"

}

**Tarea (POST /api/users/{userId}/tasks)**

{

"title": "Comprar material",

"description": "Comprar tornillos y piezas",

"completed": false

}