

Programación Multimedia y Dispositivos Móviles

Documentación API dockerizada

Servidor Backend con Express y MongoDB

Autor:

Santi Martínez

19 de noviembre de 2025

Índice

1	Introducción	4
1.1	Contexto del proyecto	4
1.2	Objetivos de la dockerización	4
1.3	Tecnologías utilizadas	4
2	Fundamentos teóricos	5
2.1	¿Qué es Docker?	5
2.2	Contenedores vs Máquinas Virtuales	5
2.3	Conceptos clave	5
2.4	Docker Compose	5
3	Arquitectura	6
3.1	Descripción de la API	6
3.2	Componentes del sistema	6
3.3	Diagrama de arquitectura	6
3.4	Flujo de comunicación	6
4	Análisis del código fuente	7
4.1	Estructura del proyecto	7
4.2	Servidor Express (app.js)	7
4.3	Dependencias y paquetes	7
4.4	Configuración de MongoDB	7
5	Dockerfile: Construcción de la imagen	8
5.1	Análisis del Dockerfile	8
5.2	Imagen base (FROM)	8
5.3	Directorio de trabajo (WORKDIR)	8
5.4	Instalación de dependencias	8
5.5	Copia de archivos	8
5.6	Exposición de puertos	8
5.7	Comando de inicio (CMD)	8
5.8	Buenas prácticas aplicadas	8
6	Docker Compose: Orquestación de servicios	9
6.1	Estructura del archivo docker-compose.yml	9
6.2	Servicio API	9
6.3	Servicio MongoDB	9
6.4	Configuración de volúmenes	9
6.5	Variables de entorno	9
6.6	Dependencias entre servicios	9
7	Gestión de variables de entorno	10
7.1	Archivo .env	10
7.2	Variables de configuración de MongoDB	10
7.3	URI de conexión	10
7.4	Seguridad y buenas prácticas	10
8	Proceso de dockerización	11
8.1	Preparación del entorno	11
8.2	Construcción de la imagen	11

8.3 Creación y ejecución de contenedores	11
8.4 Verificación del funcionamiento.....	11
9 Comandos Docker útiles	12
9.1 Comandos básicos	12
9.2 Gestión de imágenes.....	12
9.3 Gestión de contenedores	12
9.4 Docker Compose CLI.....	12
9.5 Comandos de depuración	12
10 Redes y comunicación	13
10.1 Red por defecto de Docker Compose.....	13
10.2 Resolución de nombres DNS	13
10.3 Comunicación entre contenedores	13
11 Persistencia de datos	14
11.1 Volúmenes en Docker.....	14
11.2 Volumen mongo_data.....	14
11.3 Backup y recuperación	14
12 Pruebas y validación	15
12.1 Verificación de contenedores activos	15
12.2 Pruebas de conectividad	15
12.3 Pruebas de endpoints.....	15
12.4 Logs y monitorización	15
13 Optimizaciones y mejoras	16
13.1 Optimización del Dockerfile.....	16
13.2 Multi-stage builds	16
13.3 Reducción del tamaño de la imagen.....	16
13.4 Cache de capas.....	16
14 Seguridad	17
14.1 Gestión de secretos	17
14.2 Usuarios no privilegiados.....	17
14.3 Escaneo de vulnerabilidades	17
14.4 Actualización de imágenes base	17
15 Despliegue en producción	18
15.1 Diferencias con desarrollo	18
15.2 Configuración para producción	18
15.3 Escalabilidad	18
15.4 Herramientas de orquestación	18
16 Troubleshooting	19
16.1 Problemas comunes.....	19
16.2 Errores de conexión	19
16.3 Problemas de permisos	19
16.4 Debugging de contenedores	19
17 Conclusiones	20
17.1 Ventajas de la dockerización	20
17.2 Resultados obtenidos	20

17.3 Trabajo futuro	20
18 Referencias	21
18.1 Documentación oficial	21
18.2 Recursos adicionales.....	21

1. Introducción

1.1. Contexto del proyecto

Este proyecto aborda la dockerización de una API REST desarrollada con Node.js y Express, que utiliza MongoDB como sistema de gestión de base de datos. La API implementa funcionalidades de gestión de usuarios y grupos.

La aplicación se compone de dos servicios principales que deben funcionar de manera coordinada: el servidor de aplicación que expone los endpoints REST y la base de datos MongoDB que persiste la información.

La dockerización de estos componentes permite encapsular cada servicio en contenedores independientes, facilitando su gestión, despliegue y mantenimiento.

1.2. Objetivos de la dockerización

Los principales objetivos para realizar la dockerización de la API son los siguientes:

- **Consistencia:** Permite la ejecución del entorno en cualquier ordenador preparado para ejecutar Docker; es decir, funciona de manera autónoma, llevando todo lo necesario en su interior y operando desde ahí, de forma similar a un caballo de Troya.
- **Aislamiento de componentes:** Cada servicio (API y MongoDB) se ejecuta en su propio contenedor con su propio sistema de archivos, procesos y red, evitando conflictos de dependencias.
- **Portabilidad:** Permitir que la aplicación se ejecute de manera consistente en cualquier sistema operativo (Windows, macOS, Linux).
- **Simplificación del despliegue:** Reducir el proceso de instalación y configuración.
- **Escalabilidad:** Permite crear varias copias de un servicio para que se pueda ejecutar en varios lugares al mismo tiempo.
- **Gestión de dependencias:** Encapsular todas las dependencias que usa la API (mongoose, express, dotenv, etc ...) de la aplicación dentro de la imagen Docker, garantizando que siempre se mantengan correctas.

1.3. Tecnologías utilizadas

El conjunto de tecnologías utilizado es el siguiente:

- **Node.js 20:** Entorno de ejecución de JavaScript del lado del servidor.
- **Express.js:** Framework de Node para la realización de servidores.
- **MongoDB 6:** Base de datos NoSQL orientada a documentos.
- **Mongoose:** ODM (Object Document Mapper) que permite que Node.js se comunique y gestione datos en MongoDB.
- **Docker:** Plataforma de contenedorización que permite empaquetar aplicaciones con todas sus dependencias en contenedores estandarizados. Proporciona un aislamiento ligero y eficiente.
- **Docker Compose:** Herramienta de docker para orquestar varios contenedores a la vez. Permite configurar todos los servicios, redes y volúmenes de la aplicación en un único archivo YML (docker-compose.yml).
- **Variables de entorno:** Variables de configuración del programa, guardadas de forma privada en un .env.

La combinación de estas tecnologías crea un ecosistema robusto, escalable y fácil de mantener.

2. Fundamentos teóricos

2.1. ¿Qué es Docker?

Es una plataforma que ejecuta aplicaciones en contenedores, asegurando que su funcionamiento sea el correcto en cualquier sistema.

Docker corre principalmente sobre Linux, porque utiliza características del kernel de Linux para los contenedores.

- En **Linux**, se ejecuta de forma nativa.
- En **Windows o Mac**, usa una máquina virtual ligera con Linux para poder correr contenedores.

2.2. Contenedores vs Máquinas Virtuales

Características	Contenedor	Máquina Virtual (VM)
Sistema operativo	Comparte el SO del host	Cada VM tiene su propio SO completo
Peso	Ligero, rápido de iniciar	Pesado, tarda más en iniciar
Recursos	Usa solo lo necesario	Consumo más recursos, reserva CPU/RAM
Aislamiento	Aislado a nivel de procesos	Aislado a nivel de hardware virtual
Portabilidad	Muy portable	Menos portable

Tabla 1: Comparación entre contenedores y máquinas virtuales

2.3. Conceptos clave

Para una mayor comprensión de la utilización de Docker, hay que analizar también unos conceptos principales; estos son de vital importancia:

- **Imágenes**: Plantilla que contiene todo lo necesario para ejecutar una aplicación perfectamente (Mongo, Ubuntu, Odoo, ...)
- **Contenedores**: Instancia de una imagen, la cual se encuentra aislada.
- **Volúmenes**: Almacenamiento persistente que conserva datos fuera del contenedor. Básicamente, en el caso de eliminar el contenedor por lo que sea, los datos guardados en el volumen siguen existiendo en el host. Se utiliza para datos importantes.
- **Redes**: Medio que permite la comunicación entre contenedores y el exterior. Por ejemplo a la hora de realizar manualmente ejecutarse a un contenedor, la parte del comando `-p 3000:3000`, el primer puerto es del host y el segundo del contenedor, eso hace que ambos se comuniquen.

2.4. Docker Compose

Docker Compose es una herramienta que permite orquestar varios contenedores mediante un archivo de configuración (**docker-compose.yml**). Facilita levantar, detener y administrar varios contenedores juntos, incluyendo servicios, redes y volúmenes.

3. Arquitectura

- 3.1. Descripción de la API**
- 3.2. Componentes del sistema**
- 3.3. Diagrama de arquitectura**
- 3.4. Flujo de comunicación**

4. Análisis del código fuente

- 4.1. Estructura del proyecto**
- 4.2. Servidor Express (app.js)**
- 4.3. Dependencias y paquetes**
- 4.4. Configuración de MongoDB**

5. Dockerfile: Construcción de la imagen

- 5.1. Análisis del Dockerfile**
- 5.2. Imagen base (FROM)**
- 5.3. Directorio de trabajo (WORKDIR)**
- 5.4. Instalación de dependencias**
- 5.5. Copia de archivos**
- 5.6. Exposición de puertos**
- 5.7. Comando de inicio (CMD)**
- 5.8. Buenas prácticas aplicadas**

6. Docker Compose: Orquestación de servicios

- 6.1. Estructura del archivo docker-compose.yml**
- 6.2. Servicio API**
- 6.3. Servicio MongoDB**
- 6.4. Configuración de volúmenes**
- 6.5. Variables de entorno**
- 6.6. Dependencias entre servicios**

7. Gestión de variables de entorno

- 7.1. Archivo .env**
- 7.2. Variables de configuración de MongoDB**
- 7.3. URI de conexión**
- 7.4. Seguridad y buenas prácticas**

8. Proceso de dockerización

- 8.1. Preparación del entorno**
- 8.2. Construcción de la imagen**
- 8.3. Creación y ejecución de contenedores**
- 8.4. Verificación del funcionamiento**

9. Comandos Docker útiles

- 9.1. Comandos básicos**
- 9.2. Gestión de imágenes**
- 9.3. Gestión de contenedores**
- 9.4. Docker Compose CLI**
- 9.5. Comandos de depuración**

10. Redes y comunicación

- 10.1. Red por defecto de Docker Compose**
- 10.2. Resolución de nombres DNS**
- 10.3. Comunicación entre contenedores**

11. Persistencia de datos

- 11.1. Volúmenes en Docker**
- 11.2. Volumen mongo_data**
- 11.3. Backup y recuperación**

12. Pruebas y validación

- 12.1. Verificación de contenedores activos**
- 12.2. Pruebas de conectividad**
- 12.3. Pruebas de endpoints**
- 12.4. Logs y monitorización**

13. Optimizaciones y mejoras

- 13.1. Optimización del Dockerfile**
- 13.2. Multi-stage builds**
- 13.3. Reducción del tamaño de la imagen**
- 13.4. Cache de capas**

14. Seguridad

- 14.1. Gestión de secretos**
- 14.2. Usuarios no privilegiados**
- 14.3. Escaneo de vulnerabilidades**
- 14.4. Actualización de imágenes base**

15. Despliegue en producción

- 15.1. Diferencias con desarrollo**
- 15.2. Configuración para producción**
- 15.3. Escalabilidad**
- 15.4. Herramientas de orquestación**

16. Troubleshooting

- 16.1. Problemas comunes**
- 16.2. Errores de conexión**
- 16.3. Problemas de permisos**
- 16.4. Debugging de contenedores**

17. Conclusiones

- 17.1. Ventajas de la dockerización**
- 17.2. Resultados obtenidos**
- 17.3. Trabajo futuro**

18. Referencias

- 18.1. Documentación oficial**
- 18.2. Recursos adicionales**