

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Creación de una aplicación multicapa en contenedores en la plataforma Docker

El objetivo principal del presente proyecto es llevar a cabo la **dockerización de un aplicativo multicapa** de carácter sencillo, diseñado con un enfoque pedagógico para ilustrar la arquitectura por capas y su despliegue mediante contenedores. Esta aplicación se ha concebido con tres capas diferenciadas: **presentación web, lógica de negocio y persistencia de datos**.

A pesar de que inicialmente se planteó un stack de tipo **MEAN** (Mongo - Express – Angular - Node), la capa de presentación basada en Angular y Nginx no ha sido implementada en su totalidad. Por ello, la estructura final del proyecto queda configurada de la siguiente manera:

- **Primera capa, capa de presentación:** Nginx + Sitio web
- **Segunda capa, capa de lógica de negocio:** Aplicación app.js desarrollada utilizando **Express** sobre Node.js.
- **Capa de persistencia:** Implementada mediante **MongoDB**.



La funcionalidad del aplicativo es deliberadamente simple, con el objetivo de centrarse en la estructura y el despliegue. Se trata de una especie de "*Hola Mundo*", en la que el cliente se conecta inicialmente a la capa de presentación (Nginx), que le entrega un fichero index.html. Este fichero contiene una llamada que provoca una segunda petición HTTP al mismo servidor Nginx, pero que es procesada de forma distinta en función del PATH de la solicitud.

Index.html

```

<script>
  fetch('/api/saludo')
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      console.log(data);
    })
    .catch(error => {
      console.error(error);
    });
</script>

```

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez Nombre: Alejandro	19/04/25

Nginx.conf

```
http {
    server {
        listen 80;

        location /api/ {
            proxy_pass http://capa2-express:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        location / {
            root /usr/share/nginx/html;
            index index.html;
            try_files $uri $uri/ =404;
        }
    }
}
```

Esta petición es redirigida hacia la capa de backend, donde se encuentra desplegada la aplicación Express.

```
// const PORT = 3000;
// Inicio del servidor
app.listen(PORT, () => {
  console.log(`⚡ Servidor escuchando en http://localhost:\${PORT}`);
});
```

Desde allí, la aplicación trata de establecer conexión con la base de datos MongoDB.

```
// URL usando el nombre del contenedor Docker como hostname
const MONGO_URL = "mongodb://capa3-mongo:27017/miapp";
```

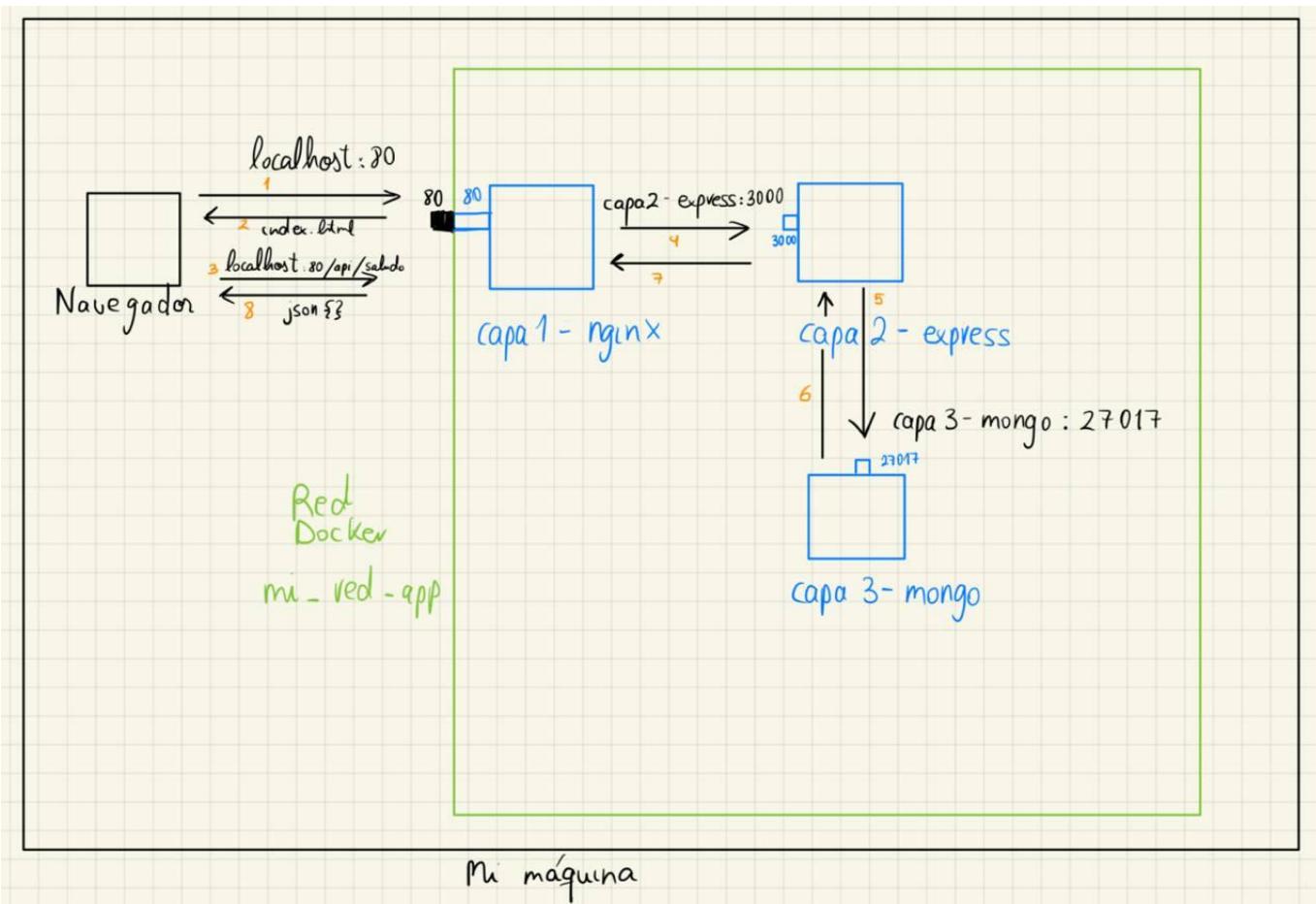
En función del resultado de dicha conexión, el backend responde al cliente con un mensaje en formato JSON, indicando si la conexión ha sido exitosa ("Hola mundo, conexión establecida correctamente") o si ha ocurrido algún error.

← → ⚙️ localhost/api/saludo

Dar formato al texto

```
{"mensaje": "Hola desde Express y conectado a MongoDB ✅"}
```

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez Nombre: Alejandro	19/04/25



Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Entorno de desarrollo

Todo el desarrollo y las pruebas del proyecto se han realizado en mi **ordenador personal**, concretamente utilizando **WSL (Windows Subsystem for Linux)**. La distribución elegida para el entorno Linux ha sido **Ubuntu 20.04 LTS (Long Term Support)**, una versión con la que ya tenía experiencia previa en otros proyectos. Esto me ha permitido aprovechar un entorno ya preparado, con todas las herramientas necesarias previamente instaladas: Docker, Visual Studio Code, y demás utilidades.

Justificación técnica

Si bien se trata de un aplicativo multicapa, y en un entorno profesional lo habitual sería utilizar **Docker Compose** para orquestar la comunicación entre contenedores, en este proyecto se ha optado por **no emplearlo**. Esta decisión responde a que Docker Compose será tratado en mayor profundidad en una actividad posterior del curso, he preferido no adelantar su uso en esta primera fase.

Comandos:

`docker network create mi_red_app`

```
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act1$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
36d1eff58bba   bridge    bridge      local
a9032005ffa1   host      host      local
babfd8145ae4   mi_red_app    bridge      local
e30f5694f466   none      null      local
```

`cd Capa2-Express/`

`docker build -t capa2-express .`

`docker run -d --name capa2-express --network mi_red_app capa2-express`

No expongo el servicio de express en un puerto del host, ya que será accedido desde el servidor web. Así me aseguro a que todo el tráfico del servicio tenga que pasar obligado por nginx.

Introduzco al contenedor en la red docker llamada **mi_red_app**

`cd ..Capa1-Nginx/`

`docker build -t capa1-nginx .`

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

docker run -d --name capa1-nginx --network mi_red_app -p 80:80 capa1-nginx

Expongo el servicio de nginx que escucha el pto 80 del contenedor en el puerto 80 del host.

cd ..//Capa3-mongo /

docker build -t capa3-mongo .

docker run -d --name capa3-mongo --network mi_red_app capa3-mongo

Resultado:

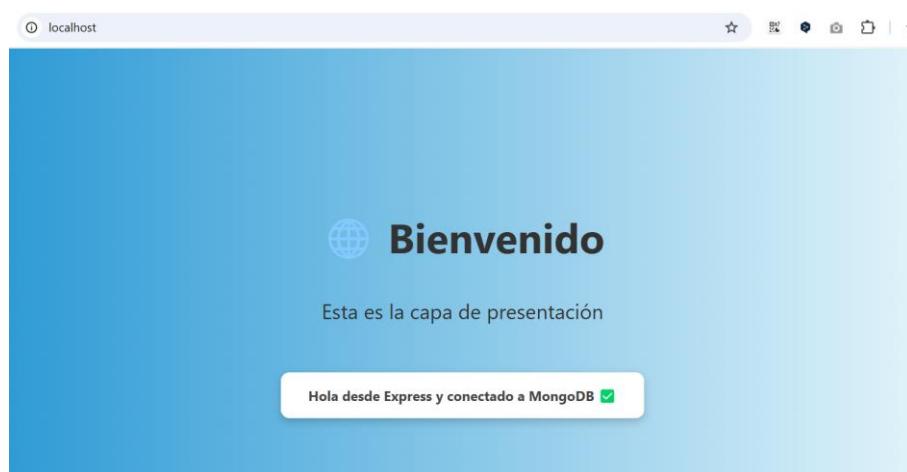
Imágenes creadas

```
● aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act1/Capa3-mongo$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
capa3-mongo    latest    3c6d05633a0a  14 minutes ago  814MB
capa1-nginx    latest    5caf04ad2cdf  16 minutes ago  192MB
capa2-express  latest    90fa588370c2  17 minutes ago  1.12GB
mongo          7.0      b61aac0fa893  4 days ago    814MB
node           18       de20d623379f  3 weeks ago   1.09GB
nginx          stable   64e5ac93d424  2 months ago   192MB
```

Contenedores activos

```
● aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act1/Capa3-mongo$ docker ps
CONTAINER ID  IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
9e60ff5af4501  capa3-mongo  "docker-entrypoint.s..."  3 seconds ago  Up 2 seconds  27017/tcp  capa3-mongo
e84ad12ef1ce  capa1-nginx  "/docker-entrypoint.s..."  2 minutes ago  Up 2 minutes  0.0.0.0:80->80/tcp, :::80->80/tcp  capa1-nginx
25761c443720  capa2-express  "docker-entrypoint.s..."  3 minutes ago  Up 3 minutes  3000/tcp  capa2-express
```

Interfaz de la aplicación

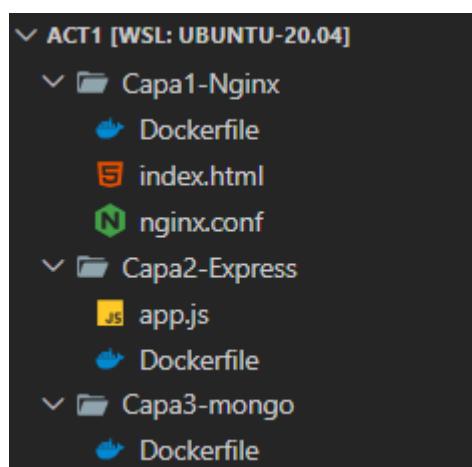


Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Jerarquía de directorios

La arquitectura del proyecto se ha organizado de forma modular siguiendo una estructura jerárquica de directorios, basada en una división por contenedor y capa del servicio. Esta organización responde al patrón de arquitectura en **tres capas**: presentación, lógica de negocio y persistencia, siendo cada una de ellas desplegada en su propio contenedor Docker.

Tal como puede observarse en la imagen adjunta, en la raíz del proyecto se encuentran tres carpetas principales.



Capa1-Nginx

Esta carpeta contiene todo lo necesario para el despliegue del sitio web estático y la configuración del servidor web Nginx, que actúa como **reverse proxy**. En su interior se encuentran los siguientes archivos:

- Dockerfile: define la construcción del contenedor que alojará Nginx y los archivos estáticos del sitio web.
- index.html: archivo principal del sitio web, que representa la interfaz de usuario (cliente).
- nginx.conf: archivo de configuración del servidor Nginx, encargado de gestionar el enrutamiento de las peticiones HTTP entrantes.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

En la configuración del Nginx, se establecen dos rutas relevantes:

- `/`: redirige al archivo `index.html`, sirviendo la interfaz estática al usuario.
- `/api/`: reenvía la petición a la capa de backend (contenedor de la capa 2), mediante una petición HTTP a través del nombre del servicio definido en la red de Docker. Esta red compartida entre contenedores permite la resolución de nombres entre ellos, facilitando la comunicación interna.

El servidor Nginx se mantiene escuchando en el **puerto 80** dentro del contenedor, lo cual permite la recepción de peticiones HTTP externas.

```
events {}

http {
    server {
        listen 80;

        location /api/ {
            proxy_pass http://capa2-express:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        location / {
            root /usr/share/nginx/html;
            index index.html;
            try_files $uri $uri/ =404;
        }
    }
}
```

Capa2-Express

Esta carpeta corresponde al backend de la aplicación, donde se implementa la lógica de negocio utilizando **Node.js** con el framework **Express**. Contiene los siguientes elementos:

- `Dockerfile`: instruye cómo construir el contenedor que ejecutará la aplicación backend.
- `app.js`: archivo JavaScript donde se define la aplicación Express, incluyendo las rutas API, la conexión con la base de datos y la lógica necesaria para responder a las peticiones del cliente.

Este servicio permanece escuchando en el **puerto 3000** y expone una serie de endpoints bajo el path `/api`, que son accedidos desde la capa 1. El backend está configurado para conectarse a la base de datos que se encuentra en la capa 3

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

mediante el driver Mongoose (ODM de MongoDB), utilizando también el nombre del servicio correspondiente como host de conexión, aprovechando nuevamente la red compartida de Docker.

```
// URL usando el nombre del contenedor Docker como hostname
const MONGO_URL = "mongodb://capa3-mongo:27017/miapp";
```

Capa3-mongo

Esta tercera carpeta representa la capa de persistencia de datos, y contiene únicamente el Dockerfile necesario para crear un contenedor con el sistema gestor de bases de datos MongoDB.

No se requiere lógica adicional en esta capa, ya que se trata de una base de datos funcional que será gestionada completamente por el servicio de backend de la capa 2.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Explicación Dockerfiles

Capa 1

El Dockerfile correspondiente a la **capa de presentación** define las instrucciones necesarias para construir la imagen Docker del contenedor que actuará como servidor web. Este contenedor ejecuta **Nginx** y se encarga de servir los archivos estáticos del sitio web, así como de redirigir las peticiones hacia el backend cuando sea necesario. El contenido del Dockerfile consta de cinco líneas, cada una de las cuales representa una **nueva capa** en la imagen construida.

FROM nginx:alpine

Esta línea indica que la imagen base del contenedor será nginx:alpine. Se trata de una imagen oficial de Nginx que incluye un sistema operativo Linux Alpine, conocido por su ligereza y eficiencia. A partir de esta base, se construirán nuevas capas que modificarán y adaptarán la imagen para nuestro caso concreto.

COPY nginx.conf /etc/nginx/nginx.conf

Aquí se copia el archivo nginx.conf (ubicado en el mismo nivel que el Dockerfile) hacia el directorio donde Nginx espera encontrar su archivo de configuración, que es /etc/nginx/nginx.conf. Con esta instrucción se reemplaza la configuración por defecto que viene incluida en la imagen base, asegurando así un comportamiento personalizado del servidor web, tanto en lo referente al enrutamiento como a las redirecciones internas.

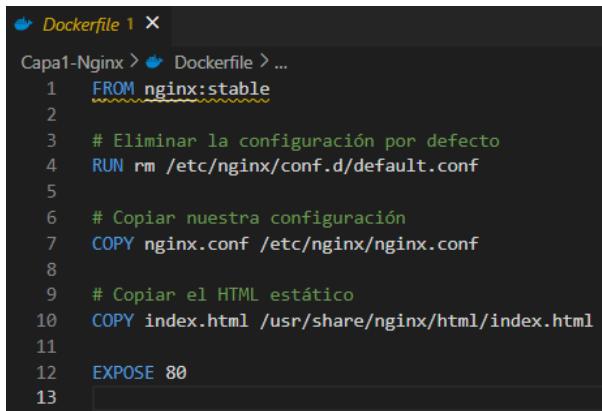
COPY index.html /usr/share/nginx/html/index.html

Esta línea se encarga de copiar el archivo index.html (el código fuente del sitio web) al directorio que Nginx utiliza por defecto para servir contenido estático, que es /usr/share/nginx/html/. De esta manera, cuando un usuario accede al contenedor a través de un navegador, Nginx responde directamente con este archivo HTML.

EXPOSE 80

Finalmente, se declara que el contenedor expondrá su puerto 80 para que sea accesible, que es el puerto estándar para el tráfico HTTP.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez Nombre: Alejandro	19/04/25



```

Dockerfile 1 X
Capa1-Nginx > Dockerfile > ...
1 FROM nginx:stable
2
3 # Eliminar la configuración por defecto
4 RUN rm /etc/nginx/conf.d/default.conf
5
6 # Copiar nuestra configuración
7 COPY nginx.conf /etc/nginx/nginx.conf
8
9 # Copiar el HTML estático
10 COPY index.html /usr/share/nginx/html/index.html
11
12 EXPOSE 80
13

```

Capa 2

La segunda capa de la arquitectura corresponde a la **lógica de negocio**, implementada mediante una aplicación en **Node.js** que hace uso del framework **Express** para gestionar las peticiones HTTP y de la librería **Mongoose** para conectarse a la base de datos MongoDB (ubicada en la capa 3). A través del Dockerfile, se define el entorno necesario para desplegar esta aplicación en un contenedor independiente.

Este Dockerfile consta de **seis instrucciones**, cada una de las cuales genera una nueva capa en la imagen Docker construida. A continuación, se describe detalladamente el propósito y funcionalidad de cada una:

FROM node:18

Se utiliza la imagen oficial node:18, que incluye el entorno de ejecución de Node.js versión 18, así como un sistema operativo Linux base y todas las dependencias necesarias para ejecutar aplicaciones JavaScript en servidor. Esta imagen proporciona un entorno completo y preparado para ejecutar el backend sin necesidad de instalar manualmente Node.js.

WORKDIR /app

Esta instrucción crea (si no existe) y define /app como el directorio de trabajo dentro del contenedor. A partir de esta línea, todas las instrucciones que se ejecuten, como COPY o RUN, se harán relativas a este directorio, lo cual permite mantener un entorno organizado y predecible.

COPY ..

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Se copian todos los archivos del directorio actual (donde se encuentra el Dockerfile, junto al app.js) al directorio de trabajo dentro del contenedor. El primer punto (.) representa el contexto de construcción local, y el segundo punto representa el destino relativo al WORKDIR. Esto asegura que tanto el archivo app.js como cualquier otro fichero necesario queden disponibles en el contenedor.

RUN npm install express mongoose cors

Con esta instrucción se instalan las dependencias necesarias para el funcionamiento de la aplicación. npm es el gestor de paquetes oficial de Node.js, y mediante este comando se descargan y registran las librerías:

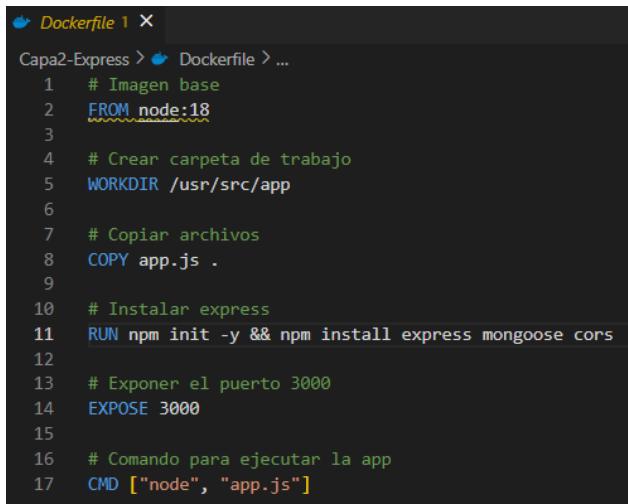
- express: framework web minimalista y rápido para Node.js.
- mongoose: ODM para MongoDB, que permite interactuar con la base de datos de forma sencilla.
- cors: middleware que permite controlar y configurar el intercambio de recursos entre diferentes orígenes, importante cuando se accede al backend desde un frontend alojado en otro dominio o puerto.

EXPOSE 3000

CMD ["node", "app.js"]

Por último, se define el comando por defecto que se ejecutará cuando se inicie el contenedor. En este caso, se lanza la aplicación con Node.js ejecutando el archivo app.js. Esta es la entrada principal del backend, donde se define el servidor Express, las rutas y la conexión con la base de datos.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez Nombre: Alejandro	19/04/25



```

Dockerfile 1 ×
Capa2-Express > Dockerfile > ...
1 # Imagen base
2 FROM node:18
3
4 # Crear carpeta de trabajo
5 WORKDIR /usr/src/app
6
7 # Copiar archivos
8 COPY app.js .
9
10 # Instalar express
11 RUN npm init -y && npm install express mongoose cors
12
13 # Exponer el puerto 3000
14 EXPOSE 3000
15
16 # Comando para ejecutar la app
17 CMD ["node", "app.js"]

```

Capa 3

La tercera y última capa del sistema corresponde a la **persistencia de datos**, que se lleva a cabo mediante una base de datos **MongoDB**, ejecutada dentro de su propio contenedor Docker. Esta capa actúa como almacenamiento para la lógica de negocio de la capa anterior, y es accedida por ésta mediante la red interna de Docker.

El Dockerfile de esta capa es muy sencillo, ya que no requiere configuraciones adicionales ni archivos personalizados. Se limita a utilizar una imagen oficial y exponer el puerto adecuado para la comunicación.

FROM mongo:7

Se utiliza la imagen oficial mongo:7, que incluye el servidor MongoDB versión 7, además de un sistema operativo base y todas las dependencias necesarias. Esta imagen está preparada para ejecutar automáticamente el proceso principal de MongoDB al arrancar el contenedor, lo que facilita enormemente su uso.

No se necesita instalar manualmente Mongo ni configurarlo, ya que esta imagen está pensada para funcionar "out of the box" (lista para usarse). Aun así, si fuera necesario, en el futuro se podrían añadir volúmenes para persistencia de datos o archivos de configuración específicos.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez Nombre: Alejandro	19/04/25

👉 Dockerfile 1 X

Capa3-mongo > 📁 Dockerfile > ...

```

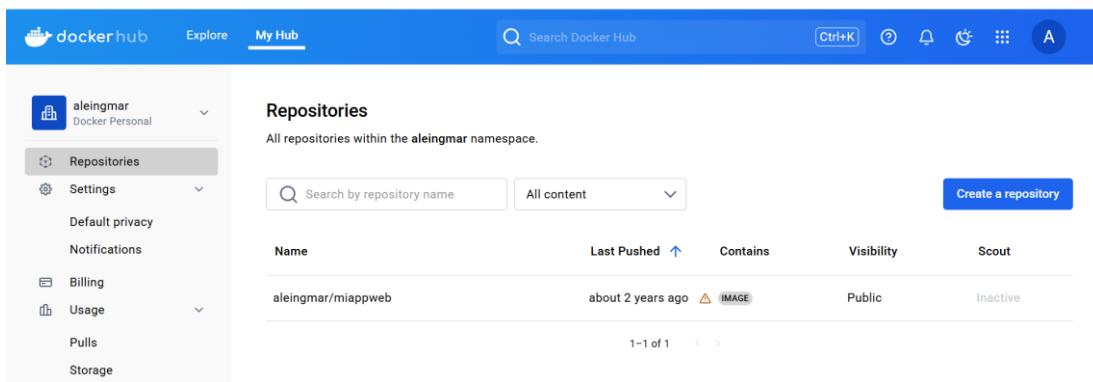
1  # Usamos la imagen oficial de MongoDB como base
2  FROM mongo:7.0
3
4  # Puedes opcionalmente añadir scripts de inicialización
5  # por ejemplo:
6  # COPY ./init-db.js /docker-entrypoint-initdb.d/
7
8  # Puerto por defecto de MongoDB
9  EXPOSE 27017
10

```

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	19/04/25
	Nombre: Alejandro	

Publicar las imágenes en Docker Hub

Iniciar sesión



```
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act1/Capa1-Nginx$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, click here to create one.
Username: aleingmar
Password:
Login Succeeded
```

Renombrar imágenes

docker tag capa3-mongo aleingmar/capa3-mongo:v1 (asi con los tres)

aleingmar/capa1-nginx	v1	d0da2c1e03c6	13 minutes ago	192MB
aleingmar/capa3-mongo	v1	3c6d05633a0a	2 hours ago	814MB
capa3-mongo	latest	3c6d05633a0a	2 hours ago	814MB
aleingmar/capa2-express	v1	90fa588370c2	2 hours ago	1.12GB

Subir las imágenes:

docker push aleingmar/capa1-nginx:v1

Repositories

All repositories within the aleingmar namespace.

Name	Last Pushed	Contains	Visibility	Scout
aleingmar/capa1-nginx	less than a minute ago	IMAGE	Public	Inactive
aleingmar/capa2-express	1 minute ago	IMAGE	Public	Inactive
aleingmar/capa3-mongo	2 minutes ago	IMAGE	Public	Inactive