Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	

Despliegue del aplicativo para gestión financiera en multi contenedor con Docker Compose contenedores en la plataforma Docker

El objetivo principal del presente proyecto es llevar a cabo **despligue de un aplicativo multicapa** de carácter sencillo **sobre un sistema multicontenedor**, 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) {
```

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;
        }
    }
}
```

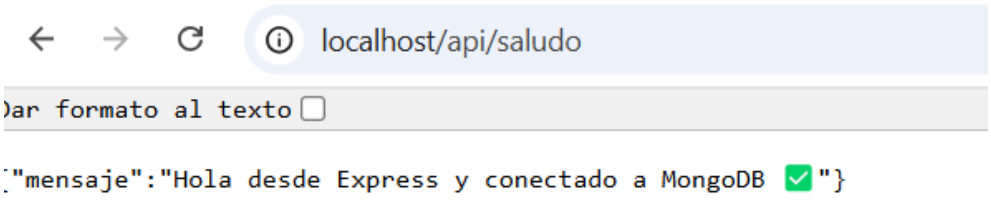
Esta petición es redirigida hacia la capa de backend, donde se encuentra desplegada la aplicación Express.

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

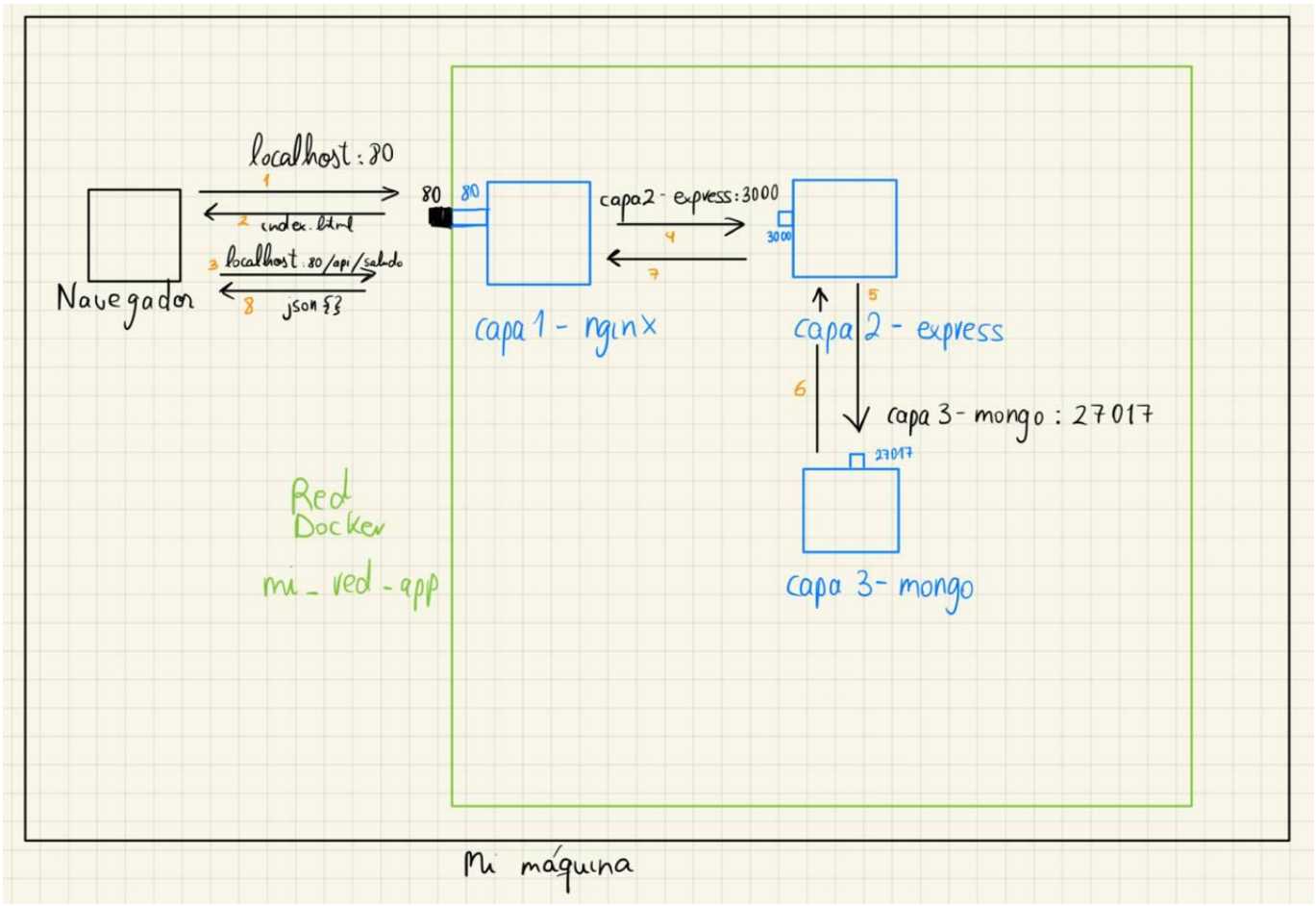
Desde allí, la aplicación trata de establecer conexión con la base de datos MongoDB.

```
// 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.



Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	



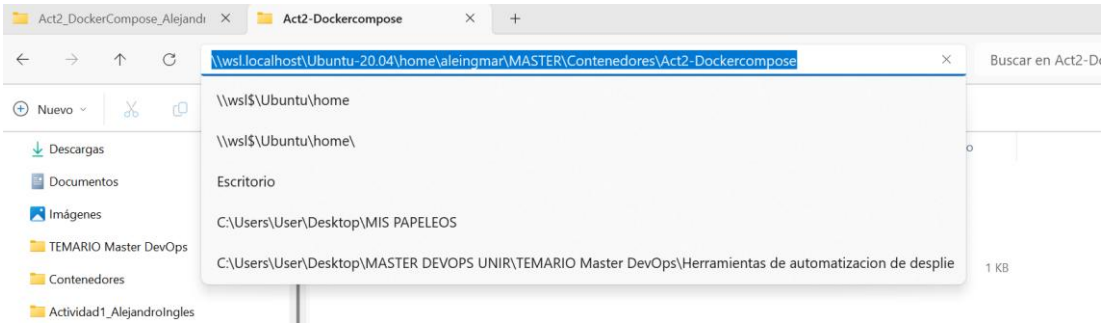
Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	

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.

Proceso de desarrollo

En primer lugar, codifico el **fichero docker-compose.yml** en el sistema de archivos normal de windows y despues lo paso al sistema WSL para probarlo y usar docker desde ahí.



Una vez copiado y dentro de WSL

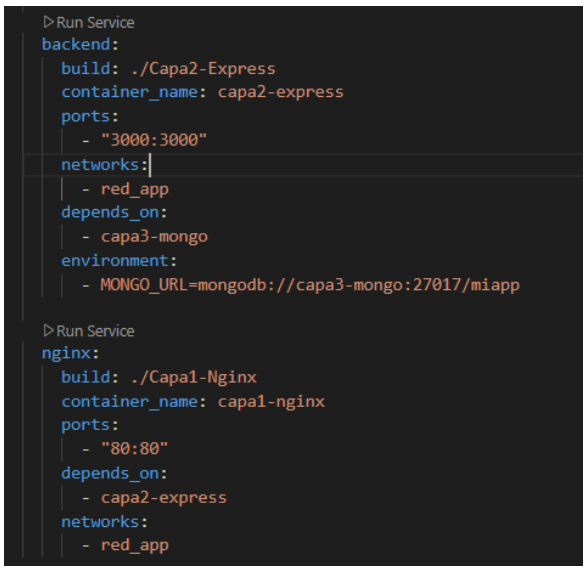
```
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act2-Dockercompose$ ls
Capa1-Nginx  Capa2-Express  Capa3-mongo  docker-compose.yml
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act2-Dockercompose$
```

Hago el primer intento de lanzamiento con la primera versión del docker-compose.yml

ERROR 1

```
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act2-Dockercompose$ docker compose up --build
service "nginx" depends on undefined service capa2-express: invalid compose project
```

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	



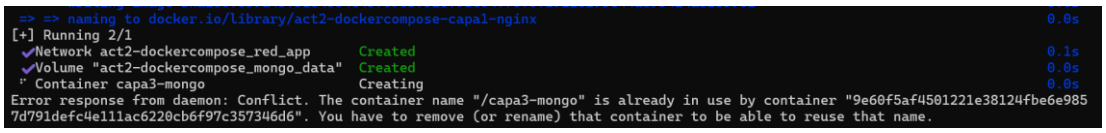
```
▷Run Service
backend:
  build: ./Capa2-Express
  container_name: capa2-express
  ports:
    - "3000:3000"
  networks:
    - red_app
  depends_on:
    - capa3-mongo
  environment:
    - MONGO_URL=mongodb://capa3-mongo:27017/miapp

▷Run Service
nginx:
  build: ./Capa1-Nginx
  container_name: capa1-nginx
  ports:
    - "80:80"
  depends_on:
    - capa2-express
  networks:
    - red_app
```

El error se debía a que en el sistema compose, nombraba a las distintos **services** de una forma (backend, nginx) pero después les referenciaba en el parámetro **depends of** con los nombres del propio contenedor.

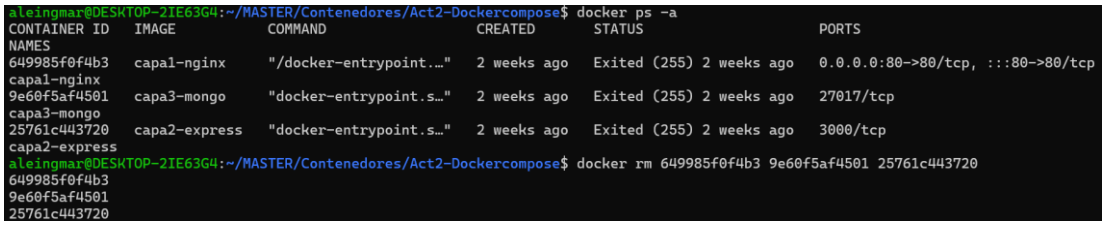
Nombré los distintos services del compose con los mismo nombres que los contenedores y listo.

ERROR 2



```
 => => naming to docker.io/library/act2-dockercompose-capa1-nginx                                                    0.0s
[+] Running 2/1
 ✔Network act2-dockercompose_red_app      Created                                                                    0.1s
 ✔Volume "act2-dockercompose_mongo_data"  Created                                                                    0.0s
 ⠼ Container capa3-mongo                  Creating                                                                   0.0s
Error response from daemon: Conflict. The container name "/capa3-mongo" is already in use by container "9e60f5af4501221e38124fbe6e985
7d791defc4e111ac6220cb6f97c357346d6". You have to remove (or rename) that container to be able to reuse that name.
```

El error se debía a que tenía contenedores creados aunque en deshuso de la primera actividad, los borro y lo vuelvo a levantar.

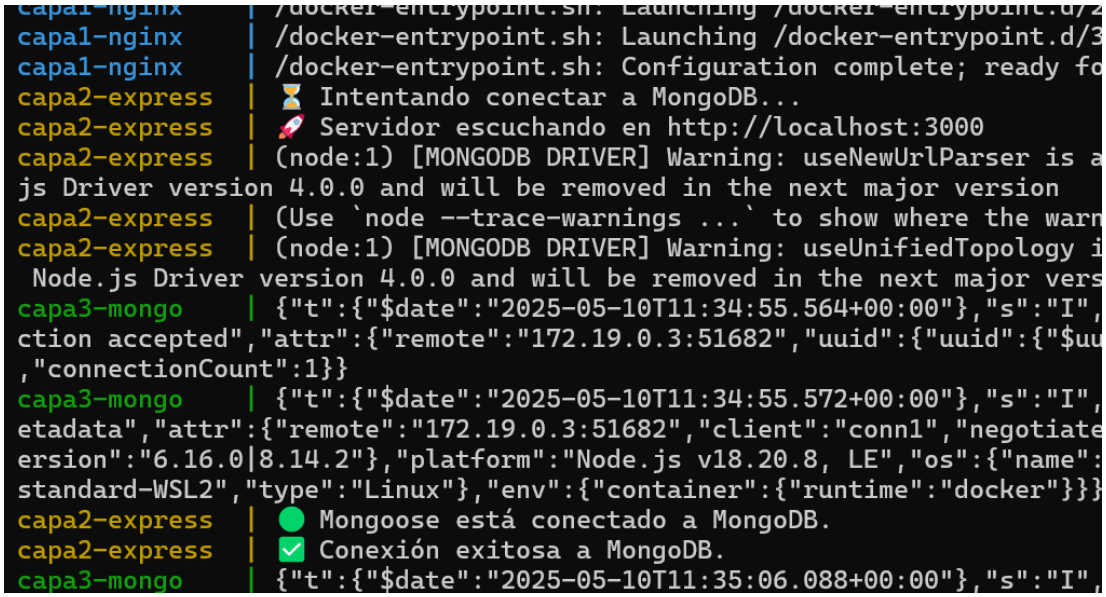


```
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act2-Dockercompose$ docker ps -a
CONTAINER ID   IMAGE           COMMAND                  CREATED       STATUS                     PORTS
NAMES
649985f0f4b3   capa1-nginx     "/docker-entrypoint.…"   2 weeks ago   Exited (255) 2 weeks ago   0.0.0.0:80->80/tcp, :::80->80/tcp
capa1-nginx
9e60f5af4501   capa3-mongo     "docker-entrypoint.s…"   2 weeks ago   Exited (255) 2 weeks ago   27017/tcp
capa3-mongo
25761c443720   capa2-express   "docker-entrypoint.s…"   2 weeks ago   Exited (255) 2 weeks ago   3000/tcp
capa2-express
aleingmar@DESKTOP-2IE63G4:~/MASTER/Contenedores/Act2-Dockercompose$ docker rm 649985f0f4b3 9e60f5af4501 25761c443720
649985f0f4b3
9e60f5af4501
25761c443720
```

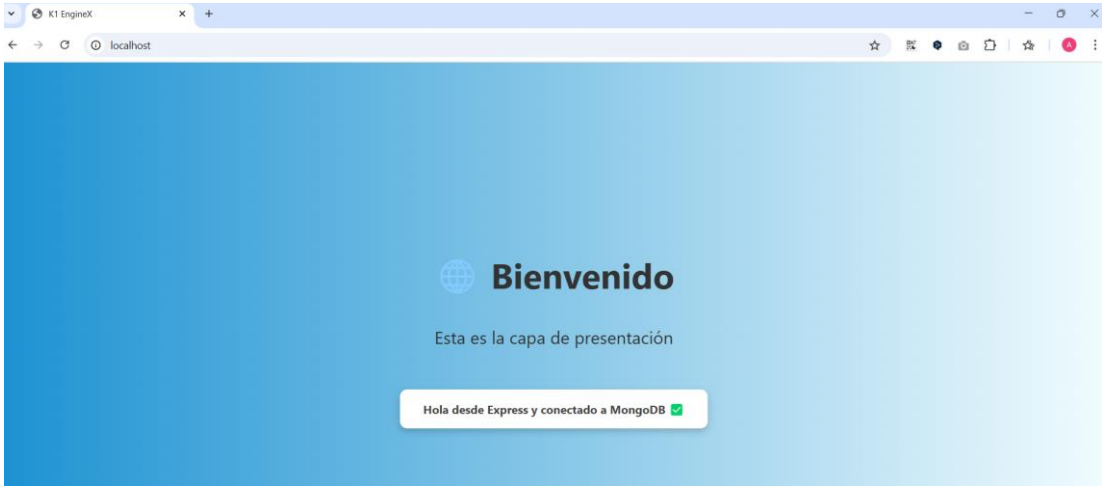
Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	

LAZAMIENTO EXISTOSO

Logs de lanzamiento



Aplicativo accedido correctamente desde la interfaz del navegador

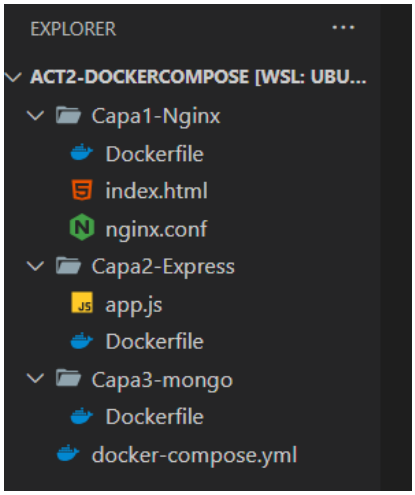


Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	

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.

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

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.

```
// Usar la variable de entorno definida en docker-compose
const MONGO_URL = process.env.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	10/05/2025
	Nombre: Alejandro	

Explicación de docker-compose.yml

```
version: '3.8'
services:
  capa3-mongo:
    build: ./Capa3-mongo
    container_name: capa3-mongo
    ports:
      - "27017:27017"
    environment:
      - BACKEND_URL=http://capa2-express:3000
    networks:
      - red_app
    volumes:
      - mongo_data:/data/db
  capa2-express:
    build: ./Capa2-Express
    container_name: capa2-express
    ports:
      - "3000:3000"
    networks:
      - red_app
    depends_on:
      - capa3-mongo
    environment:
      - MONGO_URL=mongodb://capa3-mongo:27017/miapp
  capa1-nginx:
    build: ./Capa1-Nginx
    container_name: capa1-nginx
    ports:
      - "80:80"
    depends_on:
      - capa2-express
    networks:
      - red_app
volumes:
  mongo_data:
networks:
  red_app:
    driver: bridge
```

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: Inglés Martínez	10/05/2025
	Nombre: Alejandro	

Este archivo **docker-compose.yml** define un sistema de tres capas que se comunican entre sí a través de una **red compartida llamada red**_app. Está basado en la versión 3.8 del esquema de Compose y organiza los servicios como sigue:

Servicio capa3-mongo

- Construye su imagen desde la carpeta ./Capa3-mongo.
- El contenedor se llama explícitamente capa3-mongo.
- Expone el puerto **27017**, que es el estándar de MongoDB, para poder acceder desde fuera si es necesario. (el expose del dockerfile solo lo documenta, este sí que lo expone realmente)
- Usa un volumen llamado mongo_data para que la base de datos persista aunque el contenedor se destruya.
- Pertenece a la red red_app.

Servicio capa2-express

- Se construye desde ./Capa2-Express.
- El contenedor se llama capa2-express.
- Expone el puerto **3000**, típico para backends en Node.js/Express.
- Está conectado también a la red red_app, lo que le permite comunicarse directamente con Mongo.
- Declara una dependencia con capa3-mongo, asegurando que MongoDB arranque antes que este servicio.
- Usa una variable de entorno MONGO_URL para indicar la URL de conexión a la base de datos, aprovechando que los servicios en la red se resuelven por nombre (capa3-mongo).

Servicio capa1-nginx

- Construido desde ./Capa1-Nginx.

- Su contenedor se llama capa1-nginx.
- Expone el puerto **80**, que es el habitual para NGINX y accesible desde el navegador.
- Depende de capa2-express, por lo que se asegura de que el backend esté disponible antes de arrancar.
- También está conectado a la red red_app, permitiendo que NGINX actúe como proxy hacia el backend Express.

Red red_app (tipo bridge): todos los servicios están en la misma red para que puedan llamarse por nombre de servicio, sin necesidad de IPs fijas.