Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

Universidad Internacional de La Rioja

Maestría en Desarrollo y Operaciones de Software

Creación de aplicativos en contendores con diferentes lenguajes de programación

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

Index

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

1 Introducción

1.1 Breve descripción del proyecto

Este proyecto tiene como objetivo aplicar los conocimientos adquiridos en la asignatura de Contenedores, a través de la creación y ejecución de contenedores Docker utilizando aplicaciones de distintos lenguajes de programación. La actividad consiste en construir y ejecutar aplicaciones en contenedores Docker utilizando diferentes lenguajes de programación populares. Además, se integrarán aplicaciones con bases de datos en contenedores y se orquestarán con Docker Compose.

El proceso incluye la creación de archivos Dockerfile para cada aplicación, la configuración de un archivo docker-compose.yml para ejecutar múltiples contenedores y la publicación de las imágenes resultantes en un repositorio de Docker Hub. Se requiere la documentación completa de los pasos seguidos, incluyendo la explicación de los Dockerfiles y los comandos utilizados para subir las imágenes.

El entregable final será un archivo ZIP con los ejemplos de los contenedores creados y un informe en PDF que explique detalladamente cada paso del proceso, incluyendo los archivos utilizados y los resultados obtenidos.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

2 Repositorios y Dockerfiles

2.1 Repositorios Utilizados

En esta subsección, se describen los repositorios GitHub utilizados para obtener el código de las aplicaciones. Explica qué aplicación o framework fue utilizado, cómo se integró con Docker y si se realizó alguna modificación al código original.

Repositorio Node.js (Angular):

URL: https://github.com/wkrzywiec/aston-villa-app

Modificaciones realizadas: Se descargó el código base y se configuró el Dockerfile para adaptar el build de Angular y el servidor Nginx.

Repositorio Python (FastAPI):

URL: https://github.com/asdkant/fastapi-hello-world

Modificaciones realizadas: Se descargó el código base y se configuró el Dockerfile para incluir dependencias específicas de Python y configuraciones.

Repositorio Java (Springboot):

URL: https://github.com/BuntyRaghani/spring-boot-3-hello-world

Modificaciones realizadas: Se descargó el código base y se configuró el Dockerfile para incluir dependencias específicas de Java y configuraciones.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

2.2 Dockerfiles

En esta subsección, se explican los Dockerfiles utilizados para cada uno de los proyectos, pero con más detalle el de un proyecto en específico. Cada Dockerfile debe ser presentado con una breve explicación de su propósito, la base utilizada y las instrucciones clave.

Repositorio Java (Springboot)

Dockerfile

```
# Etapa 1: Construcción
FROM maven:3.9.5-eclipse-temurin-21 AS build
WORKDIR /app

COPY pom.xml ./
COPY src ./src

RUN --mount=type=cache,target=/root/.m2 mvn clean package -DskipTests

# Etapa 2: Ejecución
FROM eclipse-temurin:21-jre-alpine
WORKDIR /app

COPY --from=build /app/target/*.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]
```

Para la API del repositorio de Springboot se utilizaron imágenes livianas solo con las dependencias necesarias para poder levantar el aplicativo de manera óptima. Estas se pueden observar junto al comando FROM del Dockerfile.

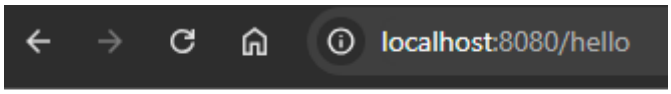
El Dockerfile se divide en dos capas. La primer capa o etapa es la de construcción, donde se compila la API de springboot usando Maven para obtener todas sus dependencias y se limpia la cache. La segunda etapa es la de ejecución, donde solo se toma la aplicación ya compilada del paso anterior para evitar tener ficheros innecesarios que solo gastarían almacenamiento, se expone el puerto 8080 de la API y se ejecuta el backend con el comando “java -jar app.jar”.

Los comandos utilizados para construir la imagen a partir de este Dockerfile y levantar un contenedor a partir de esa imagen son los siguientes:

Crea una imagen con el nombre “springboot-api-unir” y el tag “v1.0” desde la misma carpeta donde se encuentra el Dockerfile, esto se indica con un punto(.) al final del comando:
docker build -t springboot-api-unir:v1.0 .

Se crea un contenedor a partir de la imagen anteriormente creada el cual está funcionando como un proceso de fondo o backgroud process gracias a -d, se definen los puertos del contenedor y del host así como la imagen con la cual se creara el contenedor:
docker run -d -p 8080:8080 springboot-api-unir:v1.0

Al final es posible acceder al endpoint de tipo GET mediante la siguiente URL en el navegador.
http://localhost:8080/hello



Hello, World!

Para subir la imagen creada a un repositorio, elegí usar Docker Hub y cree el siguiente repositorio público: https://hub.docker.com/repository/docker/lordbears117/unir-cesar-castro/general.

Después de crear el repositorio procedí a ejecutar los siguientes comandos desde mi propia computadora:

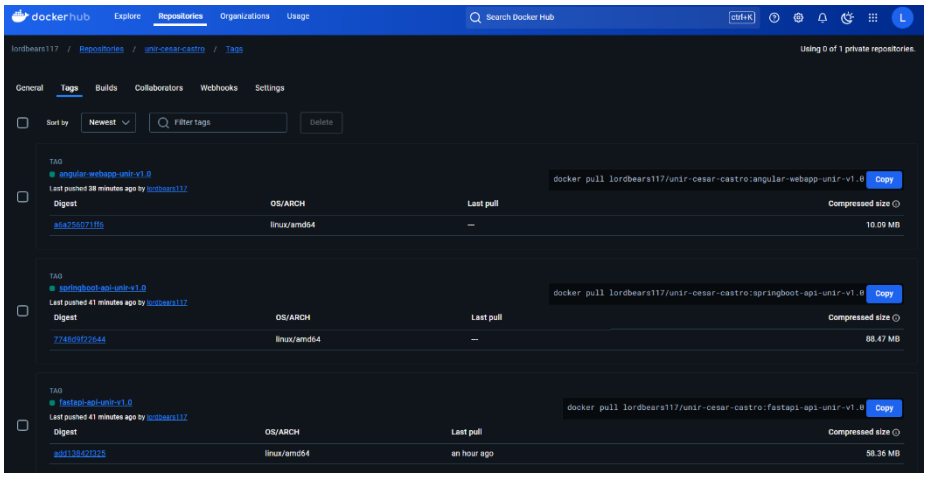
Creo un tag para subir mi imagen:
docker tag springboot-api-unir:v1.0 lordbears117/unir-cesar-castro:springboot-api-unir-v1.0

Realizo un push de mi imagen local a mi repositorio público:
docker push lordbears117/unir-cesar-castro:springboot-api-unir-v1.0

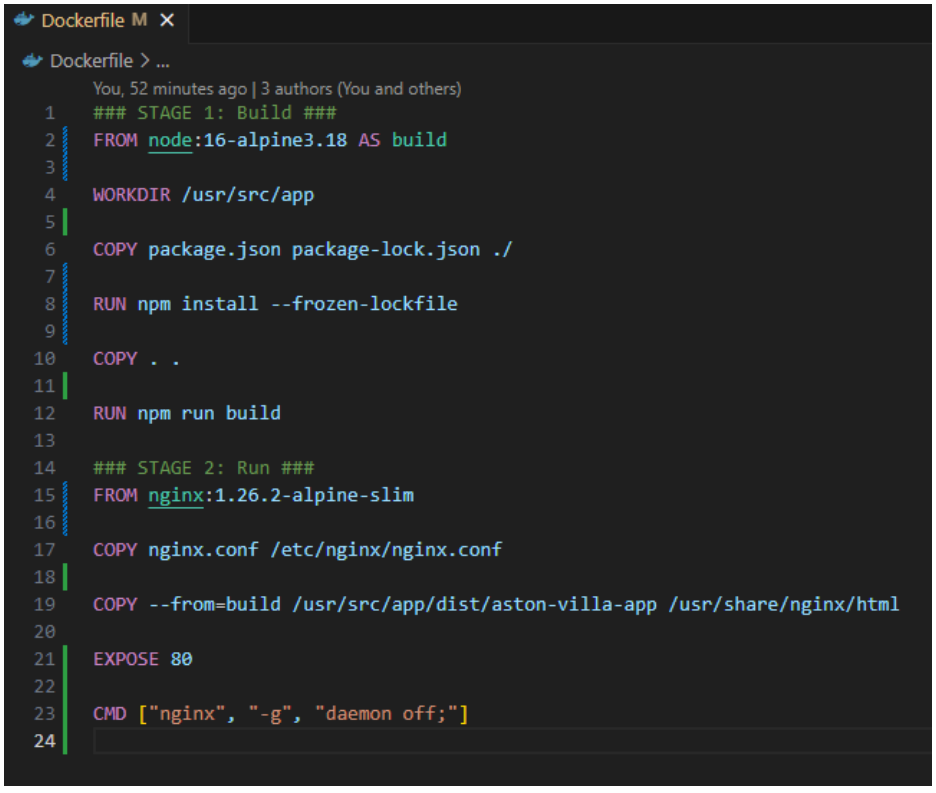
Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

Repositorio con las imágenes creadas para este proyecto.

https://hub.docker.com/repository/docker/lordbears117/unir-cesar-castro/general



Repositorio Node.js (Angular)



Comandos para crear imagen, contenedor y URL de acceso:

docker build -t angular-webapp-unir:v1.0 .

docker run -d -p 80:80 angular-webapp-unir:v1.0

http://localhost:80

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

Comandos para subir imagen a repositorio de Docker Hub:

docker tag angular-webapp-unir:v1.0 lordbears117/unir-cesar-castro:angular-webapp-unir-v1.0

docker push lordbears117/unir-cesar-castro:angular-webapp-unir-v1.0

Repositorio Python (FastAPI)

Dockerfile

```
# Imagen base ligera
FROM python:3.7-slim-buster

ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1

WORKDIR /app

COPY requirements.txt .
RUN apt-get update && apt-get install -y --no-install-recommends gcc \
    && pip install --upgrade pip \
    && pip install --no-cache-dir -r requirements.txt \
    && apt-get purge -y gcc \
    && apt-get autoremove -y \
    && rm -rf /var/lib/apt/lists/*

COPY . .

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Comandos para crear imagen, contenedor y URL de acceso:

docker build -t fastapi-api-unir:v1.0 .

docker run -d -p 8000:8000 fastapi-api-unir:v1.0

http://localhost:8000/hello

Comandos para subir imagen a repositorio de Docker Hub:

docker tag fastapi-api-unir:v1.0 lordbears117/unir-cesar-castro:fastapi-api-unir-v1.0

docker push lordbears117/unir-cesar-castro:fastapi-api-unir-v1.0

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

3 Docker Compose

Implementación de la aplicación Express con base de datos PostgreSQL usando Docker Compose

En esta sección se describirá el proceso de creación de la aplicación backend, desarrollada con Express en Node.js, y la integración con una base de datos PostgreSQL, utilizando Docker Compose para orquestar ambos contenedores. Docker Compose permite gestionar múltiples contenedores de manera sencilla, lo que facilita la creación y gestión de aplicaciones en un entorno aislado.

Estructura del Proyecto

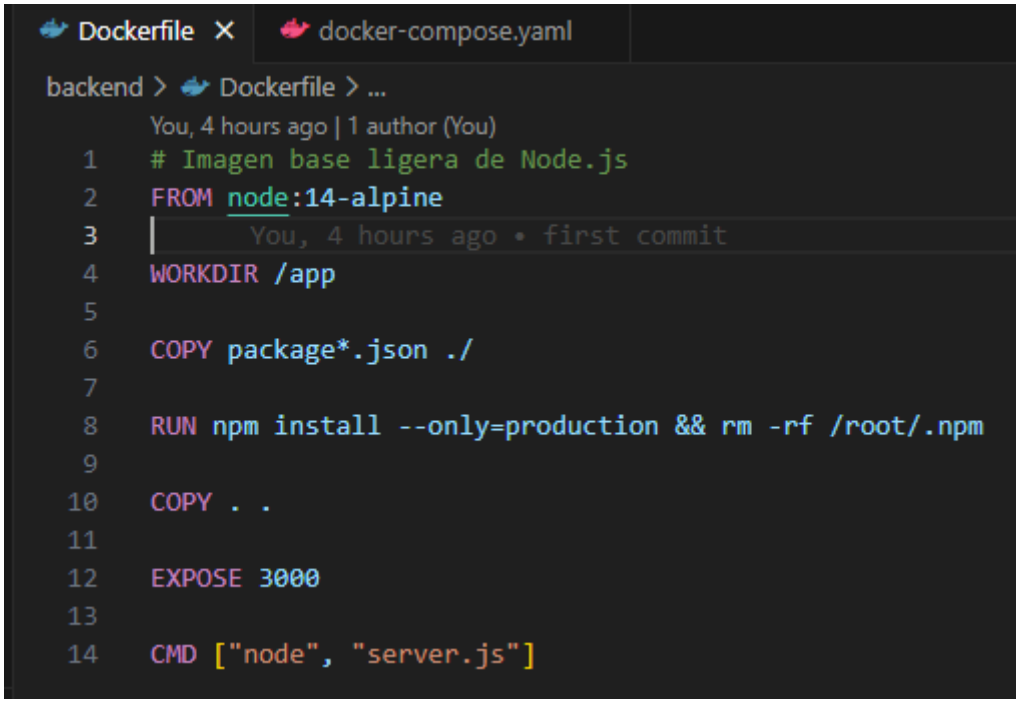
La estructura del proyecto se organiza de la siguiente manera:

```
express-api-docker-compose/
|
├── backend/            # Carpeta para la aplicación Express
|   ├── Dockerfile      # Dockerfile para la app Express
|   ├── package.json    # Dependencias de la app Express
|   ├── server.js       # Archivo principal de la app Express
|
├── docker-compose.yml  # Archivo para orquestar los contenedores
└── init.sql            # Script SQL para inicializar la base de datos
```

Nota: Se decidió no usar un archivo .env para mayor practicidad del proyecto. La mejor practica es tener todas las credenciales almacenadas de manera segura y no directo en la aplicación.

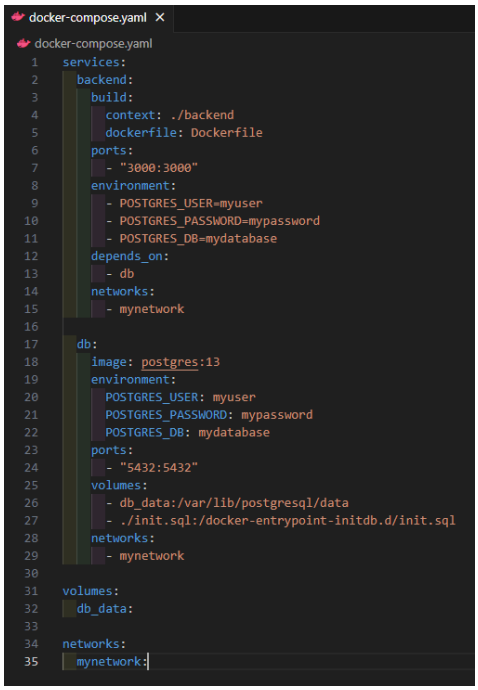
Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

Dockerfile del backend de Express



Se descargan solo las dependencias necesarias para el proyecto, se expone el puerto 3000 para el backend y se levanta con el comando ”node server.js”.

Docker Compose



Se crean dos servicios, backend y db. Backend se crea a partir del Dockerfile definido en la ruta ./backend, donde está el código de la API en Express. Se definen las credenciales necesarias para la base de datos, el puerto, sus dependencias y la red. En este caso el backend tiene como dependencia el servicio de db.

Se crea el servicio de db, el cual se crea a partir de una imagen oficial de postgres 13. Se definen las credenciales de la base de datos, el puerto por defecto de postgres, los volumes, el fichero con la sentencia de sql para la creación de una tabla de usuarios y la misma red que usa el servicio de backend.

Al final se define el volumen y la red para la ejecución de Docker Compose.

Comandos:

Se ejecuta el siguiente comando en la raíz del proyecto donde tenemos nuestro yaml de Docker Compose para construir las imágenes, su configuración general y para levantar el proyecto completo.

docker-compose up –build

```
[+] Running 3/2
 ✓ Network express-api-docker-compose_mynetwork    Created                    0.1s
 ✓ Container express-api-docker-compose-db-1       Created                    0.2s
 ✓ Container express-api-docker-compose-backend-1  Created                    0.0s
Attaching to backend-1, db-1
db-1       |
db-1       | PostgreSQL Database directory appears to contain a database; Skipping in
itialization
db-1       |
db-1       | 2024-12-16 04:41:46.472 UTC [1] LOG:  starting PostgreSQL 13.18 (Debian
13.18-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0,
 64-bit
db-1       | 2024-12-16 04:41:46.472 UTC [1] LOG:  listening on IPv4 address "0.0.0.0
", port 5432
db-1       | 2024-12-16 04:41:46.472 UTC [1] LOG:  listening on IPv6 address "::", po
rt 5432
db-1       | 2024-12-16 04:41:46.474 UTC [1] LOG:  listening on Unix socket "/var/run
/postgresql/.s.PGSQL.5432"
db-1       | 2024-12-16 04:41:46.478 UTC [27] LOG:  database system was shut down at
2024-12-16 00:26:31 UTC
db-1       | 2024-12-16 04:41:46.485 UTC [1] LOG:  database system is ready to accept
 connections
backend-1  | Servidor ejecutándose en http://localhost:3000

v View in Docker Desktop   o View Config   w Enable Watch
```

Para terminar todos los procesos podemos usar el siguiente comando.

docker-compose down

Para limpiar y volver a construir la imagen podemos usar los siguientes comandos.

docker-compose down --volumes --remove-orphans

docker-compose build

docker-compose up

Desde la terminal es posible acceder a la base de datos con el siguiente comando, en este caso express-api-docker-compose-db-1 hace referencia al contendor de la base de datos en especifico y toma las credenciales que necesita la base de datos, en nuestro caso es Postgres.

docker exec -it express-api-docker-compose-db-1 psql -U myuser -d mydatabase

La API consta de dos endpoints:

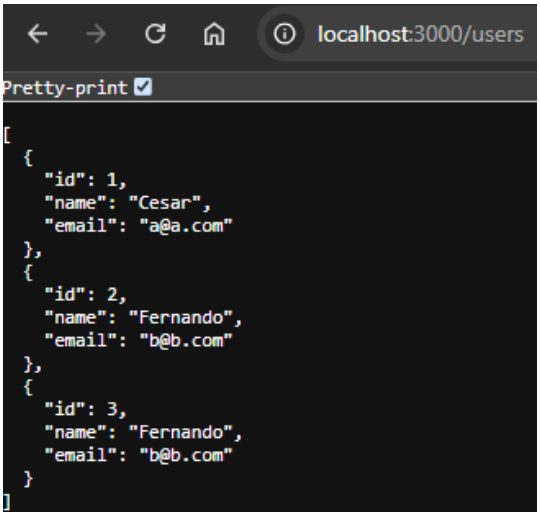
GET http://localhost:3000/users

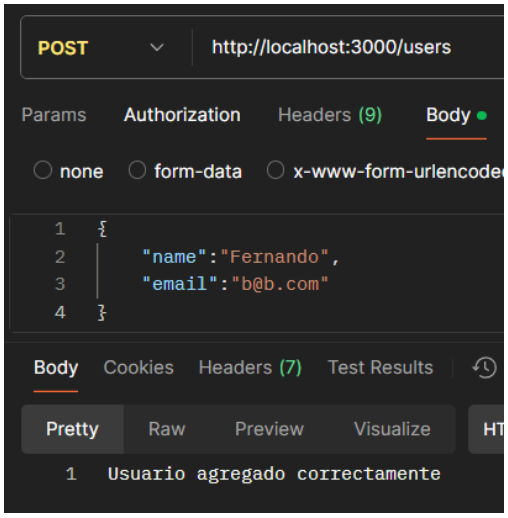
Retorna todos los usuarios existentes.

POST http://localhost:3000/users

Recibe un JSON en el body con el siguiente formato.

```
{
   "name":"Cesar",
   "email":"cesar.castroparada@gmail.com"
}
```





4 Conclusiones

La creación de imágenes Docker y su configuración permiten construir entornos portables y reproducibles para aplicaciones de distintos lenguajes de programación. Durante este proyecto, se desarrollaron imágenes Docker para aplicaciones en Node.js, Python, Java y Angular, mostrando la versatilidad de la creación de contenedores en diferentes stacks tecnológicos.

Además, se implementó un entorno más complejo usando Docker Compose, integrando una API desarrollada con Express.js y una base de datos PostgreSQL. Esto demostró cómo se puede gestionar una arquitectura de múltiples servicios, simplificando la administración y el despliegue de aplicaciones. Docker Compose permitió definir servicios, volúmenes y redes de manera declarativa, mejorando la organización y el control del entorno.

El proceso incluyó desde la creación de archivos Dockerfile siguiendo buenas prácticas, hasta la publicación de imágenes en Docker Hub, asegurando la reutilización y portabilidad del proyecto. La gestión adecuada de variables de entorno, configuraciones y volúmenes garantizó seguridad y persistencia de datos.

En resumen, este proyecto destacó la importancia de la creación de contenedores para el desarrollo y despliegue de aplicaciones modernas. Docker y Docker Compose son herramientas esenciales que facilitan el escalado, despliegue continuo y la alta disponibilidad en entornos de desarrollo y producción.

Es claro que tecnologías como Docker y Kubernetes han ganado mucho auge en los últimos años y han demostrado que agregan mucho valor a las empresas que utilizan estas tecnologías tan versátiles.

Asignatura	Datos del alumno	Fecha
Contenedores	Apellidos: CASTRO PARADA	14/12/2024
	Nombre: CESAR FERNANDO	

5 Referencias

Lordbears117. (2024, 15 de diciembre). **Repositorio de Docker Hub: unir-cesar-castro.** Docker Hub.
https://hub.docker.com/r/lordbears117/unir-cesar-castro

Krzywiec, W. (2024, 15 de diciembre). **Aston Villa App - GitHub Repository.** GitHub.
https://github.com/wkrzywiec/aston-villa-app

Raghani, B. (2024, 15 de diciembre). **Spring Boot 3 Hello World - GitHub Repository.** GitHub.
https://github.com/BuntyRaghani/spring-boot-3-hello-world

Kant, A. (2024, 15 de diciembre). **FastAPI Hello World - GitHub Repository.** GitHub.
https://github.com/asdkant/fastapi-hello-world

Lordbear117. (2024, 15 de diciembre). **Express API Docker Compose - GitHub Repository.** GitHub.
https://github.com/Lordbear117/express-api-docker-compose

Docker. (2024, 15 de diciembre). **Docker CLI Reference.** Docker Documentation.
https://docs.docker.com/reference/cli/docker/

Docker. (2024, 15 de diciembre). **Docker Compose CLI Reference.** Docker Documentation.
https://docs.docker.com/reference/cli/docker/compose/