

Práctica 2B - Despliegue mediante contenedores Docker



Miembros: Yijun Wang, Geer Wang Liu y Jimena de Prado

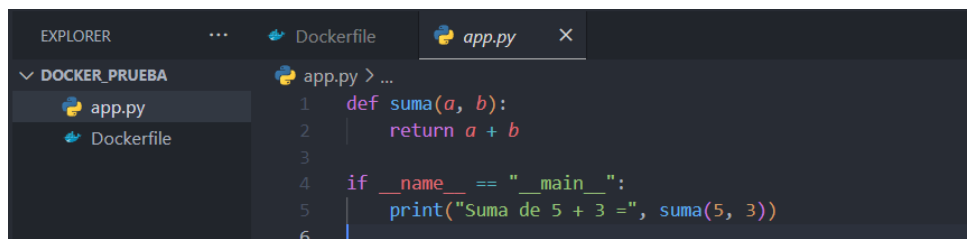
Grupo: 03

Apartado 1: Familiarización don dockers.....	3
Aplicación de calculadora	3
Contenido del Dockerfile	3
Documentación de las instrucciones del Dockerfile:	3
1. FROM	3
1. WORKDIR	4
2. COPY	4
3. RUN	4
4. CMD	4
5. EXPOSE	4
Construcción y salida del ejemplo:	4
Utilizar Almacenamiento Persistente con Docker.....	5
1. Volúmenes.....	5
2. Bind Mounts.....	5
3. tmpfs.....	6
Apartado 2: Contenedor para backend.....	6
Creación de Dockerfile en el backend (archivo ubicado en la carpeta backend)	6
Construcción y salida:	8
Verificación del funcionamiento	8
Comando docker ps	8
comando curl	9
Apartado 3: Contenedor para el Frontend	9
Creación de Dockerfile en el frontend (archivo ubicado en carpeta frontend)	10
Etapa 1: Construcción de la aplicación Vue.js	10
Etapa 2: Servir la aplicación con Nginx	10
Construcción y salida:	11
Verificación del funcionamiento	11
Apartado 4: Unificación mediante Docker Compose.....	12
Creación del Fichero Docker-compose (ubicado dentro de la carpeta Docker_compose).....	12
Construcción y salida:	13
Comando:	14
Creación:	14
CTRL+C:.....	14
Acceso a la página:.....	14

Apartado 1: Familiarización con dockers

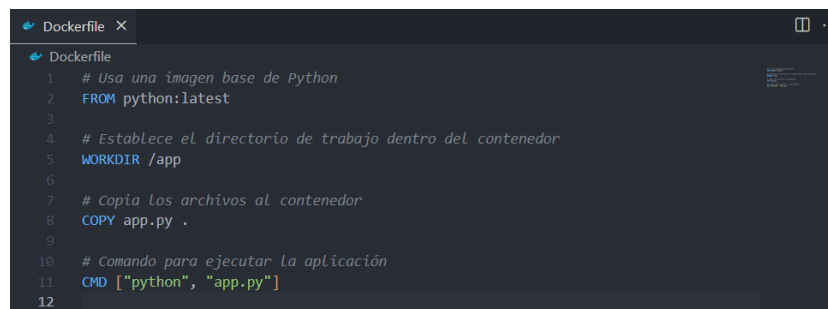
- Aplicación Python sencilla y explicaciones de instrucciones utilizadas en el Dockerfile.
- Investigar y documentar cómo se puede utilizar almacenamiento persistente.

Aplicación de calculadora (Dicho archivo se encuentra en la carpeta Apartado_1):



```
1 def suma(a, b):
2     return a + b
3
4 if __name__ == "__main__":
5     print("Suma de 5 + 3 =", suma(5, 3))
6
```

Contenido del Dockerfile



```
1 # Usa una imagen base de Python
2 FROM python:latest
3
4 # Establece el directorio de trabajo dentro del contenedor
5 WORKDIR /app
6
7 # Copia los archivos al contenedor
8 COPY app.py .
9
10 # Comando para ejecutar la aplicación
11 CMD ["python", "app.py"]
12
```

Documentación de las instrucciones del Dockerfile:

1. FROM

- **Descripción:** Especifica la imagen base desde la cual se construirá la nueva imagen.
- **FROM python:latest**

Se usa la imagen base más reciente de Python como punto de partida. Esto incluye un entorno preconfigurado con Python instalado.

1. 2. WORKDIR

- **Descripción:** Establece el directorio de trabajo para las instrucciones que siguen (RUN, CMD, ENTRYPOINT, COPY, ADD).
- **WORKDIR** /app
Establece /app como el directorio de trabajo dentro del contenedor

2. 3. COPY

- **Descripción:** Copia archivos o directorios desde el sistema host al sistema de archivos de la imagen en construcción.
- **COPY** app.py .
Copia el archivo app.py desde el host al directorio actual (.) dentro del contenedor.

3. 4. RUN

- **Descripción:** Ejecuta un comando durante el proceso de construcción de la imagen (scripts de configuración, o cualquier preparación necesaria)
- No se ha tenido que usar en el ejemplo, pero se verá en la integración con la aplicación web.

4. 5. CMD

- **Descripción:** Especifica el comando predeterminado que se ejecutará cuando se inicie un contenedor a partir de la imagen.
- **CMD** ["python", "app.py"]
Ejecuta python app.py. al ejecutar la aplicación a la hora de iniciar el contenedor.

5. 6. EXPOSE

- **Descripción:** Declara que el contenedor escucha en un puerto de red específico.
- No se ha tenido que usar en el ejemplo, pero se verá en la integración con la aplicación web.

Construcción y salida del ejemplo:

```
PS D:\AAAUNIVERSIDAD\TERCERO_CARRERA\DISTRIBUIDA\Docker-prueba> docker build -t python-calculadora .
[+] Building 1.3s (8/8) FINISHED
=> [internal] load build definition from Dockerfile                                docker:desktop-linux 0.0s
=> => transferring dockerfile: 289B                                              0.0s
=> [internal] load metadata for docker.io/library/python:latest                 0.9s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                    0.0s
=> CACHED [1/3] FROM docker.io/library/python:latest@sha256:bc78d3c007f86dbb87d711b8b082d9d564b8025487e780d24ccb 0.0s
=> => resolve docker.io/library/python:latest@sha256:bc78d3c007f86dbb87d711b8b082d9d564b8025487e780d24ccb8581d83 0.0s
=> [internal] load build context                                                 0.0s
=> => transferring context: 140B                                                  0.0s
=> [2/3] WORKDIR /app                                                            0.0s
=> [3/3] COPY app.py .                                                            0.0s
=> exporting to image                                                            0.2s
=> => exporting layers                                                            0.1s
=> => exporting manifest sha256:95dee11916c04806f783407f383d699b00cc05855d7b7c6a0a5f9898cfcaa07d 0.0s
=> => exporting config sha256:7d2e250d6de4e6804910e8f5362f6d9e8140e2680e100ee860fe8ab0544cc153 0.0s
=> => exporting attestation manifest sha256:bedca870d7b35dea699becf51464fc3a641514b0fec4d900f8c36da3f4415 0.0s
=> => exporting manifest list sha256:6f047b29925c36a0e24a923d08647726352b79756178d7fe9fcc4385400e9014 0.0s
=> => naming to docker.io/library/python-calculadora:latest                    0.0s
=> => unpacking to docker.io/library/python-calculadora:latest                  0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/0jh5h7ztfit16kdcf5843nm08
PS D:\AAAUNIVERSIDAD\TERCERO_CARRERA\DISTRIBUIDA\Docker-prueba> docker run python-calculadora
Suma de 5 + 3 = 8
```

Con ***docker build -t python-calculadora*** . se construye la nueva imagen con el archivo DockerFile, al cual le asignamos el nombre de Python-calculadora y se crea y ejecuta el contenedor basado en la imagen creada con ***docker run python-calculadora***

Utilizar Almacenamiento Persistente con Docker

El almacenamiento persistente en Docker permite que los datos generados por un contenedor no se pierdan al detenerlo o eliminarlo.

Opciones de Almacenamiento Persistente

Almacenamiento persistente en Docker

Los contenedores son efímeros por diseño, pero hay formas de mantener datos persistentes:

1. Volúmenes

- Un volumen es un espacio de almacenamiento gestionado completamente por Docker.
- Los volúmenes se almacenan fuera del sistema de archivos del contenedor y son independientes del ciclo de vida de este.

Ventajas:

- Totalmente gestionados por Docker, lo que facilita el mantenimiento y la portabilidad.
- Permiten compartir datos entre múltiples contenedores.
- No están ligados al sistema operativo del host, por lo que pueden ser utilizados incluso si el host cambia.
- Protegen los datos del acceso directo del host, lo que puede ser útil para mayor seguridad.

Casos de uso:

Bases de datos (como PostgreSQL o MySQL), donde los datos deben persistir incluso si el contenedor se reinicia.

2. Bind Mounts

- Los bind mounts montan un directorio o archivo del sistema de archivos del host directamente en el contenedor.
- Los cambios realizados en los archivos del contenedor se reflejan directamente en el host, y viceversa.

Ventajas:

- Ofrece control total sobre la ubicación y los datos, ya que se trabaja directamente con el sistema de archivos del host.
- Útil durante el desarrollo, ya que permite sincronizar cambios entre el host y el contenedor en tiempo real.

Casos de uso:

- Entornos de desarrollo donde los archivos fuente se editan en el host pero se ejecutan en el contenedor.

3. tmpfs

- Almacena los datos en la memoria RAM del host, en lugar de en el sistema de archivos.
- Los datos son **temporales**: se eliminan cuando el contenedor se detiene o se reinicia.

Ventajas:

- Muy rápido, ya que utiliza directamente la RAM.
- Ideal para datos sensibles o temporales que no deben persistir en el disco.
- Mejora el rendimiento al evitar operaciones de disco.

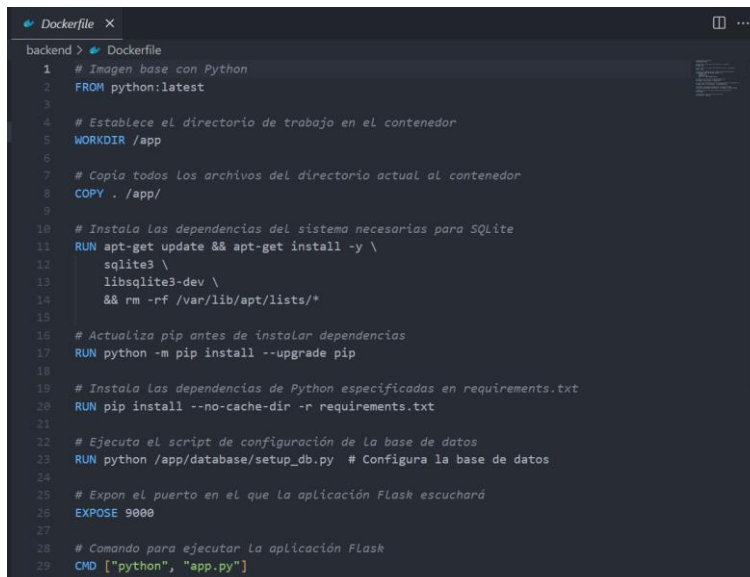
Casos de uso:

- Almacenar credenciales sensibles o datos temporales como cachés.

Apartado 2: Contenedor para backend

- Creación de contenedor para nuestra API REST desarrollada en Flask.
- Verificar que la API responde correctamente al ejecutarla desde el contenedor.

Creación de Dockerfile en el backend (archivo ubicado en la carpeta backend)



```
1 # Imagen base con Python
2 FROM python:latest
3
4 # Establece el directorio de trabajo en el contenedor
5 WORKDIR /app
6
7 # Copia todos los archivos del directorio actual al contenedor
8 COPY . /app/
9
10 # Instala las dependencias del sistema necesarias para SQLite
11 RUN apt-get update && apt-get install -y \
12     sqlite3 \
13     libsqlite3-dev \
14     && rm -rf /var/lib/apt/lists/*
15
16 # Actualiza pip antes de instalar dependencias
17 RUN python -m pip install --upgrade pip
18
19 # Instala las dependencias de Python especificadas en requirements.txt
20 RUN pip install --no-cache-dir -r requirements.txt
21
22 # Ejecuta el script de configuración de la base de datos
23 RUN python /app/database/setup_db.py # Configura la base de datos
24
25 # Expon el puerto en el que la aplicación Flask escuchará
26 EXPOSE 9000
27
28 # Comando para ejecutar la aplicación Flask
29 CMD ["python", "app.py"]
```

Explicación del código:

1. FROM python:latest:

- Utiliza la imagen base más reciente de Python.
- Proporciona un entorno preconfigurado para ejecutar aplicaciones de Python.

4. WORKDIR /app:

- Establece /app como el directorio de trabajo dentro del contenedor.

5. COPY ./app/:

- Copia todo el contenido del directorio actual al directorio de trabajo del contenedor (/app).

6. RUN apt-get update && apt-get install -y sqlite3 libsqlite3-dev && rm -rf /var/lib/apt/lists/:

- Actualiza el gestor de paquetes apt e instala SQLite y sus dependencias necesarias para la base de datos.
- Limpia la caché de apt para reducir el tamaño de la imagen.

7. RUN python -m pip install --upgrade pip:

- Actualiza el gestor de paquetes de Python (pip) a la última versión.

8. RUN pip install --no-cache-dir -r requirements.txt:

- Instala las dependencias de Python listadas en requirements.txt necesarias para la aplicación Flask.

9. RUN python /app/database/setup_db.py:

- Ejecuta un script de configuración para inicializar la base de datos.

10. EXPOSE 9000:

- Declara el puerto **9000** como el puerto que se usará para servir la aplicación Flask.

11. CMD ["python", "app.py"]:

- Especifica el comando para iniciar la aplicación Flask, ejecutando app.py.

Construcción y salida:

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\backend> docker build -t vue-backend .
[+] Building 55.1s (12/12) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 914B
=> [internal] load metadata for docker.io/library/python:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/7] FROM docker.io/library/python:latest@sha256:2d9f338cf7598aae8110f4eef1f3a6d2f8342d0c0b820879b2d79838ed 33.4s
=> => resolve docker.io/library/python:latest@sha256:2d9f338cf7598aae8110f4eef1f3a6d2f8342d0c0b820879b2d79838edf 0.0s
=> => sha256:ff09d2a676dd13e56baacd30cbe0749bd58228cbf515cdd3da24006e6d3e8beb 249B / 249B
=> => sha256:7fd26ac3f7b8d4a63de47857fc746620b36cb34fe84f214f0a757024ce7fc18f 27.20MB / 27.20MB
=> => sha256:07db14c4e2abc8dfd0101b363383e7c150e26bc5a8fca952c5ef7b15e458b5b9 6.16MB / 6.16MB
=> => sha256:ce82e98d553dd62ca6a12bebf83992ae9f9ae2748275e74b66a68cc094f868b 211.31MB / 211.31MB
=> => sha256:551df7f94f9c131f2fec0e8063142411365f0a1c88b935b9fac22be91af227e0 64.39MB / 64.39MB
=> => sha256:5bd71677db44bb63b94de61b6f1f95d5540b4ba2d6a8a6bc4d19f422b25e0c2b 23.87MB / 23.87MB
=> => sha256:fd894e782a221820acf469d425b802be26aedb5e5d26ea80a650ff6a974d488 48.50MB / 48.50MB
=> => extracting sha256:fd894e782a221820acf469d425b802be26aedb5e5d26ea80a650ff6a974d488 2.1s
=> => extracting sha256:5bd71677db44bb63b94de61b6f1f95d5540b4ba2d6a8a6bc4d19f422b25e0c2b 0.7s
=> => extracting sha256:551df7f94f9c131f2fec0e8063142411365f0a1c88b935b9fac22be91af227e0 2.2s
=> => extracting sha256:ce82e98d553dd62ca6a12bebf83992ae9f9ae2748275e74b66a68cc094f868b 5.9s
=> => extracting sha256:07db14c4e2abc8dfd0101b363383e7c150e26bc5a8fca952c5ef7b15e458b5b9 0.2s
=> => extracting sha256:7fd26ac3f7b8d4a63de47857fc746620b36cb34fe84f214f0a757024ce7fc18f 0.8s
=> => extracting sha256:ff09d2a676dd13e56baacd30cbe0749bd58228cbf515cdd3da24006e6d3e8beb 0.0s
=> [internal] load build context
=> => transferring context: 37.66kB
=> [2/7] WORKDIR /app
=> [3/7] COPY . /app/
=> [4/7] RUN apt-get update && apt-get install -y sqlite3 libsqlite3-dev && rm -rf /var/lib/apt/list 6.9s
=> [5/7] RUN python -m pip install --upgrade pip 5.9s
=> [6/7] RUN pip install --no-cache-dir -r requirements.txt 3.3s
=> [7/7] RUN python /app/database/setup_db.py # Configura la base de datos 0.5s
=> => exporting to image
=> => exporting layers
=> => exporting manifest sha256:37d854048b983dbc7cb403799c809a6e10b9172390362f035c411da96e8c13d6 0.0s
=> => exporting config sha256:46d4eb78e61e2a6f2bc3fe61837241873927c3bd89c2d7ba47c1a6a3cd1cc67 0.0s
=> => exporting attestation manifest sha256:c73570bc552c2c90e1fd40189df88d9bc5ca897a5810f24a66782a4714a78c80 0.0s
=> => exporting manifest list sha256:1969f712bae6473d73940c0e428d18f75e6ed59f7fe5d9db8b5b0244803e8c0b 0.0s
=> => naming to docker.io/library/vue-backend:latest 0.0s
=> => unpacking to docker.io/library/vue-backend:latest 0.8s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/Lfivzbv83q9vj9wnhnoun2j7
```

Explicación del comando:

- `-t vue-backend`: Etiqueta la imagen que se va a construir con el nombre `vue-backend`.
- `.`: Indica el contexto de construcción (en este caso, el directorio actual).

Verificación del funcionamiento

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\backend> docker run -d -p 9000:9000 --name flask-api vue-backend 27f461ccdfab105789d587d3e9efae0ae949f3ae40644b762f8292edda4a9158
```

Explicación:

1. **docker run**: Lanza un nuevo contenedor desde una imagen existente.
2. **-d**: Ejecuta el contenedor en segundo plano (modo "detached").
3. **-p 5000:5000**: Mapea el puerto 5000 del contenedor al puerto 5000 del host. Esto permite acceder al servicio desde el host, por ejemplo, a través de `http://localhost:5000`.
4. **--name flask-api**: Asigna el nombre `flask-api` al contenedor, lo que facilita su identificación.
5. **vue-backend**: Es el nombre de la imagen Docker que usas para crear el contenedor
6. La salida representa el **ID único del contenedor** recién creado.

Comando docker ps: Muestra los contenedores que están actualmente en ejecución.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
27f461ccdfab	vue-backend	"python app.py"	31 seconds ago	Up 24 seconds	0.0.0.0:9000->9000/tcp	flask-api

1. **CONTAINER ID:** Identificador único del contenedor.
2. **IMAGE:** La imagen usada para crear el contenedor (vue-backend).
3. **STATUS:** El contenedor está corriendo (Up 26 seconds).
4. **PORTS:** El puerto 5000 del host está mapeado al puerto 5000 del contenedor (0.0.0.0:5000->5000/tcp).
5. **NAMES:** El contenedor se llama flask-api.

comando curl :

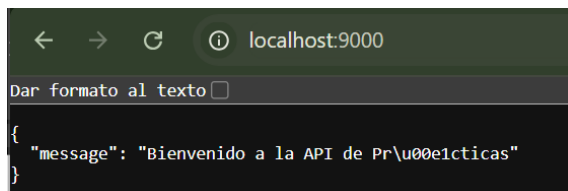
```
PS C:\Users\xwl19\Desktop\7311-01-P2B\backend> curl http://localhost:9000

StatusCode      : 200
StatusDescription : OK
Content         : {
                  "message": "Bienvenido a la API de Pr\u00e1cticas"
                }

RawContent      : HTTP/1.1 200 OK
                  Connection: close
                  Content-Length: 57
                  Content-Type: application/json
                  Date: Tue, 03 Dec 2024 11:51:54 GMT
                  Server: Werkzeug/3.1.3 Python/3.13.0
                  {
                    "message": "Bienvenido a la API..."
                  }
Forms           : {}
Headers        : {[Connection, close], [Content-Length, 57], [Content-Type, application/json], [Date, Tue, 03 Dec 2024 11:51:54 GMT]...}
Images         : {}
InputFields    : {}
Links          : {}
ParsedHtml     : mshtml.HTMLDocumentClass
RawContentLength : 57
```

Con el **comando curl** sabemos que el servidor está ejecutando correctamente porque:

1. **StatusCode: 200:** Indica que la solicitud fue exitosa.
2. **StatusDescription: OK:** Confirma que no hubo errores.
3. **Content: Bienvenido a la API de Prácticas:** Muestra el mensaje esperado, lo que significa que el servicio responde correctamente.



Apartado 3: Contenedor para el Frontend

1. Creación de Dockerfile multi-etapa: la primera etapa compilará la aplicación Vue.js y la segunda etapa servirá los archivos estáticos mediante un servidor web ligero.

Creación de Dockerfile en el frontend (archivo ubicado en carpeta frontend)

```
1  # Etapa 1: Construcción de la aplicación Vue.js
2  FROM node:18 AS build-stage
3
4  # Establece el directorio de trabajo en el contenedor
5  WORKDIR /app
6
7  # Copia package.json y package-lock.json al contenedor
8  COPY package.json package-lock.json ./
9
10 # Instala las dependencias
11 RUN npm install
12
13 # Copia todo el código fuente al contenedor
14 COPY . ./
15
16 # Construye la aplicación Vue.js
17 RUN npm run build
18
19 # Etapa 2: Servir la aplicación con Nginx
20 FROM nginx:alpine AS production-stage
21
22 # Copia los archivos de la construcción al contenedor de Nginx
23 COPY --from=build-stage /app/dist /usr/share/nginx/html
24
25 # Expone el puerto 80
26 EXPOSE 80
27
28 # Comando por defecto para ejecutar Nginx
29 CMD ["nginx", "-g", "daemon off;"]
30
```

Etapa 1: Construcción de la aplicación Vue.js

1. **FROM node:18 AS build-stage:**
 - Utiliza la imagen base de Node.js para construir la aplicación Vue.js.
 - Se etiqueta como build-stage para referencia en la siguiente etapa.
2. **WORKDIR /app:**
 - Establece el directorio de trabajo dentro del contenedor en /app.
3. **COPY package*.json package-lock.json.:**
 - Copia los archivos package.json y package-lock.json al contenedor (necesarios para instalar dependencias).
4. **RUN npm install:**
 - Instala todas las dependencias necesarias para el proyecto desde package.json.
5. **COPY . .:**
 - Copia el código fuente dentro del contenedor.
6. **RUN npm run build:**
 - Ejecuta el script de construcción definido en package.json. Esto genera los archivos optimizados de producción en la carpeta dist.

Etapa 2: Servir la aplicación con Nginx

1. **FROM nginx:alpine AS production-stage:**
 - Usa una imagen base ligera de Nginx para servir los archivos de la aplicación.
2. **COPY --from=build-stage /app/dist /usr/share/nginx/html:**
 - Copia los archivos generados en la carpeta dist durante la etapa de construcción al directorio predeterminado de Nginx (/usr/share/nginx/html).
3. **EXPOSE 80:**
 - Expone el puerto 80, que es el puerto predeterminado de Nginx, para que la aplicación sea accesible desde el host.
4. **CMD ["nginx", "-g", "daemon off;"]:**
 - Configura el comando por defecto para iniciar Nginx en modo no demonio, lo que mantiene el contenedor en ejecución.

Construcción y salida:

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\frontend> docker build -t vue-frontend .
[+] Building 1.0s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 747B
=> [internal] load metadata for docker.io/library/nginx:alpine 0.6s
=> [internal] load metadata for docker.io/library/node:18 0.6s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 64.80kB 0.1s
=> [build-stage 1/6] FROM docker.io/library/node:18@sha256:cef95c42cf26eeefa0a557f30967b9acf58bc206a7c2bba5dd0fd61237ee7a5c 0.0s
=> => resolve docker.io/library/node:18@sha256:cef95c42cf26eeefa0a557f30967b9acf58bc206a7c2bba5dd0fd61237ee7a5c 0.0s
=> [production-stage 1/2] FROM docker.io/library/nginx:alpine@sha256:41523187cf7d7a2f2677a80609d9caal4388b5c1fbca9c410ba3de602aaab4 0.1s
=> => resolve docker.io/library/nginx:alpine@sha256:41523187cf7d7a2f2677a80609d9caal4388b5c1fbca9c410ba3de602aaab4 0.0s
=> CACHED [build-stage 2/6] WORKDIR /app 0.0s
=> CACHED [build-stage 3/6] COPY package-lock.json ./ 0.0s
=> CACHED [build-stage 4/6] RUN npm install 0.0s
=> CACHED [build-stage 5/6] COPY ./ 0.0s
=> CACHED [build-stage 6/6] RUN npm run build 0.0s
=> CACHED [production-stage 2/2] COPY --from=build-stage /app/dist /usr/share/nginx/html 0.0s
=> exporting to image 0.2s
=> => exporting layers 0.0s
=> => exporting manifest sha256:a96ef850f4c1029404e047de402168e47e8516adff53ec456db1b59bc63ea 0.0s
=> => exporting config sha256:b1ca9037f240791dc8b1b17c585bb040d6bfde21dd426b69fbd55913ce531148 0.0s
=> => exporting attestation manifest sha256:486d460e9e7efc934ffa374360af7cfc1b1812633078c48c65d8e8f89b2839af 0.0s
=> => exporting manifest list sha256:16fbae68370a63ceb6ffdd881770a85f873a004ce9032c490f3dd37b3a5c1aaf8 0.0s
=> => naming to docker.io/library/vue-frontend:latest 0.0s
=> => unpacking to docker.io/library/vue-frontend:latest 0.1s
```

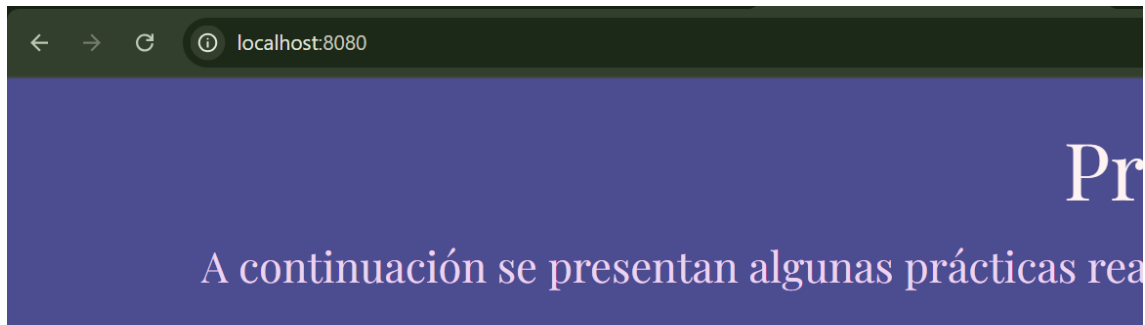
1. **docker build:** Construye la imagen Docker basada en el archivo Dockerfile.frontend.
2. **-t vue-frontend:** Asigna el nombre vue-frontend a la imagen.
3. **-f Dockerfile.frontend:** Especifica el Dockerfile a utilizar.

Verificación del funcionamiento

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\frontend> docker run -d -p 8080:80 --name frontend vue-frontend
a318681f4379725086a622f421c582a55364c0a9593ba846870067fd173d27d4
```

- **docker run:** Ejecuta un contenedor basado en la imagen vue-frontend.
- **-d:** Corre el contenedor en segundo plano (modo "detached").
- **-p 8080:80:** Mapea el puerto **80** del contenedor (usado por Nginx) al puerto **8080** del host.
- **vue-frontend:** Especifica la imagen desde la que se crea el contenedor.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a318681f4379	vue-frontend	"/docker-entrypoint..."	32 seconds ago	Up 25 seconds	0.0.0.0:8080->80/tcp	frontend



Apartado 4: Unificación mediante Docker Compose

- Unificación mediante Docker Compose para simplificar el despliegue completo.

Creación del Fichero Docker-compose (ubicado dentro de la carpeta Docker_compose)

```
C: > Users > xwl19 > Desktop > 7311-01-P2B > docker_compose > docker-compose.yml
1  services:
2      backend:
3          build:
4              context: ../backend
5              dockerfile: Dockerfile
6          ports:
7              - "9000:9000"
8          networks:
9              - app-network
10
11     frontend:
12         build:
13             context: ../frontend
14             dockerfile: Dockerfile # Ruta al Dockerfile dentro de src/plugins
15         ports:
16             - "8080:80" # Mapea el puerto 80 del contenedor frontend al puerto 8080
17         depends_on:
18             - backend
19         networks:
20             - app-network
21
22     networks:
23         app-network:
24             driver: bridge
```

1. Backend:

- Se construye a partir del Dockerfile.backend y expone el puerto **9000**.

- Está conectado a una red llamada app-network para comunicarse con el frontend.
- Creamos un **volumen para lograr datos persistentes**.

2. Frontend:

- Se construye a partir del Dockerfile.frontend y expone el puerto **8080** (en el host) al puerto **80** del contenedor (servido por Nginx).
- Depende del backend, asegurando que este se inicie primero.

3. Red app-network:

- Permite que el frontend y el backend se comuniquen fácilmente por nombre de servicio, gracias al controlador bridge.

Construcción y salida:

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\docker_compose> docker-compose up --build
[+] Building 11.3s (28/28) FINISHED
=> [backend internal] load build definition from Dockerfile
=> => transferring dockerfile: 914B
=> [backend internal] load metadata for docker.io/library/python:latest
=> [backend internal] load .dockerignore
=> => transferring context: 2B
=> [backend internal] load build context
=> => transferring context: 266B
=> [backend 1/7] FROM docker.io/library/python:latest@sha256:2d9f338cf7598aae8110f4eeef1f3a6d2f8342d0c0b820879b2d79838edf6f565
=> => resolve docker.io/library/python:latest@sha256:2d9f338cf7598aae8110f4eeef1f3a6d2f8342d0c0b820879b2d79838edf6f565
=> CACHED [backend 2/7] WORKDIR /app
=> CACHED [backend 3/7] COPY . /app/
=> CACHED [backend 4/7] RUN apt-get update && apt-get install -y sqlite3 libsqlite3-dev && rm -rf /var/lib/apt/lists/*
=> CACHED [backend 5/7] RUN python -m pip install --upgrade pip
=> CACHED [backend 6/7] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [backend 7/7] RUN python /app/database/setup_db.py # Configura la base de datos
=> [backend] exporting to image
```

```
[+] Running 4/4
✔ Network docker_compose_app-network Created 0.0s
✔ Volume "docker_compose_backend_data" Created 0.0s
✔ Container docker_compose-backend-1 Created 0.1s
✔ Container docker_compose-frontend-1 Created 0.1s
Attaching to backend-1, frontend-1
frontend-1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
frontend-1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
frontend-1 | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
frontend-1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
frontend-1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
frontend-1 | /docker-entrypoint.sh: Configuration complete; ready for start up
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: using the "epoll" event method
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: nginx/1.27.3
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: OS: Linux 5.15.167.4-microsoft-standard-WSL2
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker processes
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 30
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 31
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 32
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 33
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 34
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 35
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 36
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 37
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 38
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 39
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 40
frontend-1 | 2024/12/03 15:57:05 [notice] 1#1: start worker process 41
backend-1 | * Serving Flask app 'app'
backend-1 | * Debug mode: on
backend-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
backend-1 | * Running on all addresses (0.0.0.0)
backend-1 | * Running on http://127.0.0.1:9000
backend-1 | * Running on http://172.18.0.2:9000
backend-1 | Press CTRL+C to quit
backend-1 | * Restarting with stat
backend-1 | * Debugger is active!
backend-1 | * Debugger PIN: 138-957-523
```

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\docker_compose> docker volume ls
DRIVER      VOLUME NAME
local       docker_compose_backend_data
```

Comando:

- **docker-compose up --build:** Construye imágenes y levanta los contenedores definidos en docker-compose.yml.

Creación:

- Primero se construye el backend (docker_compose-backend-1) y luego el frontend (docker_compose-frontend-1).
- Crea una red Docker (docker_compose_app-network) para que los contenedores se comuniquen.

CTRL+C:

- Si se usa CTRL+C, detienes ambos contenedores

Acceso a la página:

- Para acceder a la página solo tendríamos de nuevo solo se tendría que hacer **docker-compose up**

Nota:

Verificar que los puertos no están en usos antes de hacer --build:

```
PS C:\Users\xwl19\Desktop\7311-01-P2B\7311-01-P1C> netstat -ano | findstr :8080
PS C:\Users\xwl19\Desktop\7311-01-P2B\7311-01-P1C> netstat -ano | findstr :9000
```

Si no hay salido es que dichos puertos están libres.