Docker

Autor:

Kevin Cárdenas.

Febrero de 2023

Índice

1. Introducción

Docker es una plataforma de software que permite la creación, distribución y ejecución de aplicaciones en contenedores. Estos contenedores son una forma de virtualización ligera que permite la ejecución de aplicaciones en entornos aislados y portables. Esto significa que las aplicaciones pueden ser empaquetadas junto con todas sus dependencias en un contenedor, y ser ejecutadas en cualquier máquina que tenga Docker instalado.

La tecnología de contenedores es similar a la virtualización de máquinas, pero con una diferencia clave: en lugar de virtualizar todo el hardware de una máquina física, los contenedores virtualizan solo el sistema operativo y sus recursos, compartiendo el kernel del sistema operativo anfitrión. Esto hace que los contenedores sean mucho más ligeros que las máquinas virtuales, lo que permite una mayor eficiencia en el uso de recursos y una mayor portabilidad.

En términos más simples, los contenedores son como paquetes que contienen todo lo necesario para que una aplicación funcione correctamente, y Docker es una plataforma que permite crear, almacenar, transportar y ejecutar estos paquetes de manera eficiente y aislada del sistema anfitrión.

1.1. ¿Qué son los contenedores y para qué sirven?

Los contenedores son una tecnología de virtualización a nivel de sistema operativo que permite empaquetar una aplicación con todas sus dependencias en un único contenedor, lo que lo hace muy portable y fácil de distribuir. Un contenedor es una unidad de software que incluye todo lo necesario para que una aplicación se ejecute, como librerías, archivos de configuración, dependencias, etc.

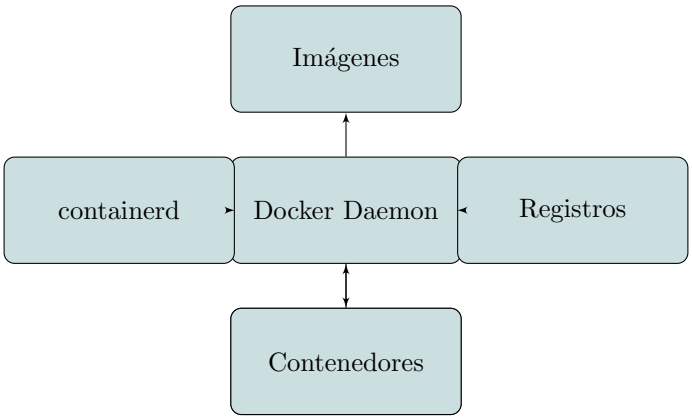
En comparación con las máquinas virtuales, los contenedores son más livianos y consumen menos recursos, ya que no necesitan un sistema operativo completo para funcionar. Además, los contenedores pueden ser implementados en cualquier sistema operativo que soporte la tecnología de contenedores, lo que los hace muy versátiles.

Para entender la idea de los contenedores, se puede hacer la analogía de un contenedor de envío de carga. Un contenedor de envío de carga es un contenedor estándar que puede ser utilizado para transportar cualquier tipo de carga, ya sea ropa, alimentos, maquinaria, etc. El contenedor asegura que la carga se mantenga segura durante el viaje y se puede mover de un lugar a otro con facilidad. De manera similar, un contenedor de software encapsula todo lo necesario para que una aplicación se ejecute y lo hace portátil y fácil de mover de un lugar a otro.

En resumen, los contenedores son una tecnología de virtualización de sistema operativo que permite empaquetar una aplicación con todas sus dependencias en un único contenedor. Los contenedores son más livianos y consumen menos recursos que las máquinas virtuales y son muy portables y versátiles. La analogía de un contenedor de envío de carga puede ayudar a entender cómo los contenedores encapsulan una aplicación y la hacen fácil de mover de un lugar a otro.

1.2. Arquitectura de Docker

La arquitectura de Docker se basa en el modelo cliente-servidor, donde el cliente es la herramienta de línea de comandos de Docker y el servidor es el daemon de Docker. El daemon de Docker es responsable de realizar la mayoría de las tareas de Docker, como crear, ejecutar y administrar contenedores. A continuación, se describen los principales componentes de la arquitectura de Docker:



- **Docker daemon**: es el servidor de Docker que se ejecuta en el sistema operativo host. Es responsable de crear, ejecutar y administrar contenedores, y se comunica con el cliente de Docker a través de una API REST.
- **Docker client**: es la herramienta de línea de comandos que se utiliza para interactuar con el daemon de Docker. El cliente de Docker envía solicitudes al daemon a través de la API REST, y el daemon realiza las tareas solicitadas.
- **Docker registries**: son repositorios de imágenes de Docker que se utilizan para almacenar y distribuir imágenes de contenedores. El registro público de Docker es el Docker Hub, pero también se pueden configurar registros privados.
- **Docker images**: son plantillas de solo lectura que se utilizan para crear contenedores. Las imágenes se almacenan en los registros de Docker y contienen todos los componentes necesarios para ejecutar una aplicación, como bibliotecas, dependencias y código fuente.
- **Docker containers**: son instancias en ejecución de una imagen de Docker. Los contenedores están aislados del sistema operativo host y de otros contenedores, pero comparten el mismo kernel del sistema operativo. Esto permite que los contenedores sean ligeros y portátiles.

En resumen, Docker es una herramienta cliente-servidor que utiliza una arquitectura modular basada en contenedores para proporcionar un entorno de ejecución aislado y portátil para aplicaciones. El daemon de Docker es el componente clave de la arquitectura, ya que es el encargado de realizar las tareas de Docker, mientras que el cliente de Docker se utiliza para enviar solicitudes al daemon a través de una API REST. Los registros, imágenes y contenedores son otros componentes importantes de la arquitectura de Docker que se utilizan para almacenar, distribuir y ejecutar aplicaciones en contenedores.

2. Fundamentos

Docker es una herramienta de virtualización de aplicaciones que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones de forma eficiente y consistente en diferentes entornos. Fue creado por la empresa Docker Inc. en 2013 y se ha convertido en una herramienta muy popular en el mundo de la informática, especialmente en la industria del software.

Antes de la llegada de Docker, la virtualización solía realizarse a través de máquinas virtuales. Las máquinas virtuales se crean a partir de un sistema operativo completo y separado, que se ejecuta en un hipervisor en la máquina anfitriona. Aunque esto proporciona una gran flexibilidad, también puede ser costoso en términos de recursos y tiempo de configuración.

Docker, por otro lado, utiliza la virtualización a nivel de sistema operativo, lo que significa que puede ejecutar múltiples aplicaciones en un solo sistema operativo anfitrión sin la sobrecarga asociada con las máquinas virtuales. Además, Docker utiliza la tecnología de contenedores para encapsular las aplicaciones y sus dependencias en unidades lógicas, lo que facilita la gestión de aplicaciones y garantiza que se ejecuten de manera consistente en diferentes entornos.

En resumen, Docker ofrece una solución flexible y rentable para la virtualización de aplicaciones, lo que la convierte en una herramienta imprescindible para los desarrolladores y profesionales de la informática.

2.1. Instalación

Para instalar Docker, sigue estos pasos:

1. Descarga e instala Docker Desktop desde la página oficial de Docker: https://www.docker.com/products/docker-desktop.
2. Abre una terminal y verifica que Docker esté instalado correctamente ejecutando el siguiente comando:
   `docker version`
3. Si el comando anterior devuelve información sobre la versión de Docker instalada, entonces la instalación se ha completado correctamente.

2.2. Primeros pasos: hola mundo

En una terminal camos a ejecutar el comando `docker fun hello-world`

Obtendremos como salida:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:6e8b6f026e0b9c419ea0fd02d3905dd0952ad1feea67543f525c73a0a790fefb
```

```
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
      1. The Docker client contacted the Docker daemon.
      2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
      (amd64)
      3. The Docker daemon created a new container from that image which runs the
      executable that produces the output you are currently reading.
      4. The Docker daemon streamed that output to the Docker client, which sent it
      to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
      $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
      https://hub.docker.com/

For more examples and ideas, visit:
      https://docs.docker.com/get-started/
```

Docker está buscando la imagen "hello-world" localmente en tu máquina, pero no la encuentra, por lo que intenta descargarla de Docker Hub, que es el registro público de imágenes de Docker. La descarga se muestra en el segundo y tercer paso de la salida. Una vez que la imagen se descarga, Docker crea un nuevo contenedor a partir de esa imagen y lo ejecuta para mostrar el mensaje "Hello from Docker!" en tu terminal. El cuarto paso de la salida muestra cómo se transmitió la salida del contenedor al cliente de Docker y, por lo tanto, a tu terminal. Finalmente, se proporcionan algunos enlaces para obtener más información y recursos relacionados con Docker.

2.3. Comandos

A continuación te presento algunos de los comandos básicos más útiles en Docker:

- `docker info`: es utilizado para obtener información del sistema donde se está ejecutando Docker, como por ejemplo la versión de Docker instalada, el número de contenedores que se están ejecutando, la cantidad de imágenes descargadas, entre otras características del sistema.

  Al ejecutar el comando "docker info" en una terminal, se muestra una salida con información detallada acerca de la configuración del sistema Docker. Esta información puede ser útil para solucionar problemas de configuración, entender el rendimiento del sistema y planificar la capacidad de almacenamiento.

  Algunos de los elementos que se pueden encontrar en la salida del comando "docker info" incluyen:

1. Nombre y versión del sistema operativo.
2. Versión de Docker instalada.
3. Tipo de driver de almacenamiento utilizado por Docker.
4. Información sobre el número de contenedores en ejecución y el número total de imágenes descargadas.
5. Configuración de red, como la dirección IP predeterminada para el demonio Docker.
6. Información sobre el hardware y recursos disponibles en el sistema, como la cantidad de CPU y memoria RAM disponibles.

En resumen, el comando "docker info" es útil para obtener una visión general del sistema Docker y de sus recursos.

- `docker run`: se utiliza para crear y ejecutar contenedores. La sintaxis básica del comando es la siguiente:

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Donde:

- **OPTIONS**: Son las opciones que se pueden pasar al comando, como por ejemplo el puerto que se desea abrir en el contenedor, el nombre que se le quiere dar al contenedor, etc.
- **IMAGE**: Es la imagen que se utilizará para crear el contenedor. Si la imagen no existe localmente, Docker la buscará en el registro de imágenes predeterminado (Docker Hub).
- **COMMAND** (opcional): Es el comando que se desea ejecutar dentro del contenedor.
- **ARG** (opcional): Son los argumentos que se pasan al comando que se va a ejecutar dentro del contenedor.

El comando docker run puede tomar muchas opciones diferentes para configurar el contenedor y su entorno de ejecución. Por ejemplo, se puede especificar qué puertos abrir en el contenedor, cómo conectar volúmenes, establecer variables de entorno, entre otras opciones.

- `docker ps` es utilizado para listar todos los contenedores Docker que están en ejecución en tu sistema. Al ejecutar este comando, se mostrará una tabla con información sobre cada contenedor, como su ID, la imagen que se está utilizando, el comando que se está ejecutando dentro del contenedor, el estado actual del contenedor, los puertos que están siendo expuestos y el nombre del contenedor.
  Además, el comando "docker ps.ªcepta varias opciones para personalizar su salida. Algunas de las opciones más comunes son:
  - `docker ps -a`: muestra todos los contenedores, no solo los que están en ejecución.
  - `docker ps -q`: muestra solo los IDs de los contenedores en lugar de la información detallada.
  - `docker ps -f`: permite filtrar los contenedores según ciertas características, como el estado o la imagen utilizada.

Por ejemplo cuando hacemos `docker ps -a` la salida es:

```
CONTAINER ID  IMAGE        COMMAND    CREATED              STATUS                       PORTS   NAMES
dcfaffa82f0e  hello-world  "/hello"   4 seconds ago        Exited (0) 3 seconds ago             dazzling_saha
cc1505d68ad9  hello-world  "/hello"   About a minute ago   Exited (0) About a minute ago        magical_khorana
```

Aqui podemos hacer `docker inspect dcfaffa82f0e` que despliega la descripción del contenedor. Nos concentraremos en la columna "NAMES". Que nos sirve para referirnos al contenedor en vez del "CONTAINER ID"

En el comando `docker run`, podemos colocar un nombre especifico, y **único** al contenedor por ejemplo:: `docker run --name hello-bby hello-world` haremos que el nombre, que podemos usar para llamar en diferentes momentos al contenedor sea "hello-bby" También podríamos renombrar un contenedor haciendo:

```
docker rename hello-bby hola-bb
```

- `docker build`: es utilizado para construir una imagen de Docker a partir de un Dockerfile. Un Dockerfile es un archivo de texto que contiene todas las instrucciones necesarias para construir una imagen de Docker.

  El siguiente es un ejemplo básico de un Dockerfile que se utilizaría para construir una imagen de Docker para una aplicación de Python:

  ```
  FROM python:3.8
  COPY requirements.txt .
  RUN pip install --no-cache-dir -r requirements.txt
  COPY . .
  CMD ["python", "app.py"]
  ```

  Para construir una imagen de Docker a partir de este Dockerfile, se utilizaría el siguiente comando:

  ```
  1 docker build -t my-python-app .
  ```

  El argumento -t se utiliza para establecer una etiqueta o nombre para la imagen, y el punto al final del comando indica que se debe buscar el Dockerfile en el directorio actual.

  Una vez que se ha construido la imagen, se puede ejecutar un contenedor utilizando el comando docker run y especificando el nombre de la imagen:

  ```
  1 docker run -p 5000:5000 my-python-app
  ```

  Este comando ejecutará un contenedor utilizando la imagen my-python-app, y mapeará el puerto interno 5000 del contenedor al puerto 5000 del host para que la aplicación sea accesible desde el navegador web en http://localhost:5000.

- `docker container prune`: para eliminar todos los cotenedores que no se están utilizando.

- `docker stop`: Este comando se utiliza para detener un contenedor en ejecución. Por ejemplo, si tienes un contenedor con el ID `mycontainerid`, puedes detenerlo usando el siguiente comando:

  ```
  1 docker stop mycontainerid
  ```

Este comando detendrá el contenedor con el ID `mycontainerid`.

- `docker rm`: Este comando se utiliza para eliminar un contenedor. Por ejemplo, si tienes un contenedor con el ID `mycontainerid`, puedes eliminarlo usando el siguiente comando:

```
docker rm mycontainerid
```

Este comando eliminará el contenedor con el ID `mycontainerid`.

- `docker rmi`: Este comando se utiliza para eliminar una imagen de Docker. Por ejemplo, si tienes una imagen con el nombre `myimage`, puedes eliminarla usando el siguiente comando:

```
docker rmi myimage
```

Este comando eliminará la imagen con el nombre `myimage`.

Estos son solo algunos de los comandos básicos más útiles en Docker. Hay muchos otros comandos disponibles que puedes utilizar según tus necesidades. Para obtener más información sobre los comandos de Docker, te recomiendo consultar la documentación oficial de Docker.

2.3.1. Docker run: Modo interactivo

En la terminal corremos `docker run ubuntu`. Pero está apagado, entonces debemos correr `docker run -it ubuntu`, el cual lo "enciende", con resultado:

```
$ docker run -it ubuntu
root@66a6b450b2ae:/#
```

Estamos corriendo un "Ubuntu" en efecto:

```
root@66a6b450b2ae:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"
```

y ahí tenemos la información del «Ubuntu» que estamos corriendo.

Podemos Abrir otra terminal y hacer `docker ps`, con resultado:

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
66a6b450b2ae   ubuntu    "/bin/bash"   9 minutes ago   Up 9 minutes             optimistic_leavitt
```

Y por último con `docker ps -a`, vemos que el otro contenedor tambñien existe pero está "apagado":

```
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
66a6b450b2ae   ubuntu    "/bin/bash"   11 minutes ago   Up 11 minutes                         optimistic_leavitt
527a7bf18557   ubuntu    "/bin/bash"   15 minutes ago   Exited (0) 15 minutes ago             ubuntu_docker
```

para salir solo hacemos

```
root@66a6b450b2ae:/# exit
exit
```

Un contenedor se “apaga” cuando su proceso principal se “apaga”.
y si quiero volver a “iniciar” basta hacer `docker start <ID o nombre del contenedor>`, y podemos entrar a la terminal de este contenedor con `docker exec -it ubuntu_docker bash`

- `docker exec`: el comando principal que permite ejecutar un comando dentro de un contenedor Docker.
- `-it`: dos opciones combinadas en una sola, donde “-i” significa “modo interactivo” y “-t” significa “terminal”, que se usan para que podamos interactuar con la terminal del contenedor como si estuviéramos trabajando en una sesión de terminal local.
- `ubuntu_docker`: el nombre o ID del contenedor donde se ejecutará el comando. En este ejemplo, el nombre del contenedor es “ubuntu_docker”.
- `bash`: el comando que se ejecutará dentro del contenedor. En este caso, se está ejecutando el shell “bash” dentro del contenedor.

2.4. ¿Cómo exponer un contenedor?

Usaremos un servidor que se llama **nginx**, con el comando `docker run -d --name proxy nginx`, Este comando ejecuta una instancia de un contenedor de la imagen `nginx` en segundo plano (`-d`) con el nombre de contenedor `proxy` (`--name proxy`). La imagen `nginx` es una imagen de un servidor web que se utiliza comúnmente como proxy inverso en arquitecturas de aplicaciones web. Una vez que se ejecuta este comando, el contenedor estará en ejecución en segundo plano y listo para servir solicitudes a través del puerto 80.
Resultado:

```
$ docker run -d --name proxy nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
bb263680fed1: Pull complete
258f176fd226: Pull complete
a0bc35e70773: Pull complete
077b9569ff86: Pull complete
3082a16f3b61: Pull complete
7e9b29976cce: Pull complete
Digest: sha256:6650513efd1d27c1f8a5351cbd33edf85cc7e0d9d0fcb4ffb23d8fa89b601ba8
Status: Downloaded newer image for nginx:latest
a6a61e461a20cfa4c459520369ab574b633dfde0d57e438ab7fdeab2f9702080
```

El resultado indica que Docker no encontró la imagen de Nginx en el sistema local y por lo tanto, comenzó a descargar la última versión de Nginx desde el repositorio de Docker Hub utilizando la etiqueta "latest". A continuación, se muestra la salida de Docker que indica que se están descargando los diferentes componentes de la imagen ("Pull complete") y una vez que se descargan todos los componentes, se muestra el hash SHA256 de la imagen descargada y el estado de la descarga ("Downloaded newer image for nginx:latest"). Finalmente, se muestra el ID del contenedor en ejecución. Si hacemos `docker ps`, tendremos:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
c63e2d259bed   nginx     "/docker-entrypoint..."  6 seconds ago   Up 6 seconds   80/tcp    proxy
```

La salida está organizada en una tabla con las siguientes columnas:

- CONTAINER ID: El identificador único del contenedor.
- IMAGE: La imagen Docker que se usó para crear el contenedor.
- COMMAND: El comando que se está ejecutando dentro del contenedor.
- CREATED: La fecha y hora en la que se creó el contenedor.
- STATUS: El estado actual del contenedor (en este caso, Üp 6 seconds", lo que indica que el contenedor está en ejecución y lleva 6 segundos corriendo).
- PORTS: Los puertos del contenedor que están expuestos al host.
- NAMES: El nombre del contenedor, que puede ser asignado por el usuario o generado automáticamente por Docker. En este caso, la salida indica que hay un solo contenedor en ejecución, con un ID de c63e2d259bed, creado a partir de la imagen nginx, que se está ejecutando un comando definido en docker-entrypoint.sh y lleva 6 segundos corriendo. Además, el contenedor expone el puerto 80 y se le asignó el nombre proxy.

pero para ver el contenedor en el puerto "local", debemos hacer `docker run --name proxy -p 8080:80 nginx` y la salida en la terminal es:

```
$ docker run --name proxy -p 8080:80 nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

y lo podemos ver en el puerto localhost:8080, y revisando con `docker ps`, vemos:

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
09cf0f92a8d0   nginx     "/docker-entrypoint..."  5 minutes ago    Up 5 minutes    0.0.0.0:8080->80/tcp, :::8080->80/tcp   proxy
```

En este caso, hay un contenedor que fue creado a partir de la imagen de "nginx" y se le asignó el nombre "proxy". Además, el contenedor se encuentra en ejecución y está exponiendo el puerto 80 del contenedor en el puerto 8080 del host.
El comando `docker logs --tail 10 -f proxy` permite visualizar los últimos 10 registros del log del contenedor "proxy" en tiempo real.

- La opción "–tail 10" especifica que solo se deben mostrar los últimos 10 registros del log.
- La opción "-f" permite seguir el log en tiempo real (como un "tail -f" en Linux).
- "proxy" es el nombre del contenedor cuyo log queremos visualizar.

De esta manera, al ejecutar este comando en la terminal, se mostrarán en tiempo real los últimos 10 registros del log del contenedor "proxy". Si se quiere visualizar más registros, se puede cambiar el valor de "–tail" por un número mayor.

3. Ejemplos prácticos

A continuación, se presentan algunos ejemplos prácticos de cómo utilizar Docker.

3.1. Crear una imagen de Docker a partir de un Dockerfile

Te muestro un ejemplo básico de cómo crear una imagen de Docker a partir de un Dockerfile:

1. Crea un directorio para el proyecto y entra en él:

```
mkdir mi-proyecto
cd mi-proyecto
```

2. Crea un archivo llamado "Dockerfile" en el directorio:

```
touch Dockerfile
```

3. Abre el archivo Dockerfile en tu editor de texto favorito y agrega el siguiente contenido:

```
FROM nginx:latest


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

Este Dockerfile utiliza la imagen base de Nginx y luego copia el archivo "index.html" desde el directorio actual al directorio predeterminado de Nginx para archivos HTML.

4. Crea el archivo "index.html" en el mismo directorio que el "Dockerfile" y agrega el siguiente contenido:

```
<!DOCTYPE html>
<html>
<head>
      <title>Mi sitio web</title>
</head>
<body>
      <h1>Bienvenido a mi sitio web!</h1>
</body>
</html>
```

Este es el archivo HTML que se copiará al directorio de Nginx.

5. Ejecuta el siguiente comando para construir la imagen de Docker:

```
docker build -t mi-sitio-web .
```

Este comando construirá la imagen de Docker utilizando el archivo "Dockerfile" y la etiquetará con el nombre "mi-sitio-web".

6. Una vez que la imagen se haya construido, puedes ejecutar el siguiente comando para crear un contenedor a partir de la imagen:

```
docker run -d -p 80:80 mi-sitio-web
```

   Este comando crea un contenedor a partir de la imagen mi-“sitio-web” y lo ejecuta en segundo plano (“-d”). También mapea el puerto 80 del contenedor al puerto 80 del host (“-p 80:80”).

7. Abre tu navegador y visita localhost para ver el sitio web en acción.

Ahora tienes una imagen de Docker que puede ser utilizada para ejecutar tu sitio web en cualquier servidor de Docker.

3.2. Crear una imagen de Docker para una aplicación web Flask

En este ejemplo, vamos a crear una imagen de Docker para una aplicación web Flask simple que mostrará un mensaje en la página principal.

1. Crea un archivo llamado `app.py` en el directorio de trabajo actual y agrega el siguiente código:

```
from flask import Flask

app = Flask(name)

@app.route('/')
def hello_world():
return 'Hola, mundo!'

if name == 'main':
app.run(host='0.0.0.0')
```

   Este es un ejemplo básico de una aplicación web Flask que muestra un mensaje en la página principal.

2. Crea un archivo llamado `requirements.txt` en el directorio de trabajo actual y agrega la siguiente línea:

```
Flask==1.1.2
```

   Este archivo especifica las dependencias de Python que se necesitan para ejecutar la aplicación web.

3. Crea un archivo llamado `Dockerfile` en el directorio de trabajo actual y agrega el siguiente contenido:

```
FROM python:3.9-slim-buster
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

Este archivo Dockerfile especifica una imagen base de Python 3.9 y agrega el código y las dependencias necesarias para ejecutar la aplicación web Flask. También expone el puerto 5000 y especifica el comando que se ejecutará cuando se ejecute el contenedor.

4. Crea una imagen de Docker ejecutando el siguiente comando:

```
docker build -t myflaskapp .
```

Este comando construye una imagen de Docker con el nombre `myflaskapp` utilizando el Dockerfile y los archivos en el directorio actual.

5. Ejecuta un contenedor de la imagen que acabas de crear:

```
docker run -p 5000:5000 myflaskapp
```

Este comando ejecuta un contenedor de la imagen que acabas de crear y mapea el puerto 5000 del contenedor al puerto 5000 del host.

6. Abre un navegador web y navega a `http://localhost:5000`. Verás la aplicación web Flask que acabas de crear, que muestra el mensaje "Hola, mundo" en la página principal.

Este es un ejemplo básico de cómo crear una imagen de Docker para una aplicación web Flask y ejecutarla en un contenedor. Puedes personalizar este ejemplo para adaptarlo a tus necesidades y crear tus propias imágenes de Docker para tus aplicaciones web.

4. Manejo de datos

En Docker, puedes manejar datos de varias maneras.

4.1. Bind Mount

En Docker, un "bind mount" (montaje de unión) es un mecanismo para montar un directorio o archivo desde el sistema de archivos del host dentro de un contenedor Docker. De esta manera, los datos dentro del contenedor pueden persistir incluso si el contenedor se elimina, lo que hace que los bind mounts sean una forma de persistir datos en Docker.
Cuando se crea un contenedor Docker, se pueden especificar uno o varios bind mounts mediante el uso de la opción "-v" (o "–mount").
Por ejemplo, la siguiente opción de línea de comando crea un contenedor y especifica un bind mount para un directorio en el sistema de archivos del host:

```
docker run -v /ruta/al/directorio/host:/ruta/dentro/del/contenedor imagen-de-docker
```

Vamos a una terminal en un directorio de trabajo y creemos un directorio nuevo con:
`mkdir docker_data` y nos movemos allí `cd docker_data`.
crearemos un directorio desde el cual vamos a copiar lo que pase dentro del contenedor, para eso haremos:

```
1 mkdir mongodata
```

Guardamos la ubicació de este así:

```
1 # pwd
2 /home/quind/Documentos/Clases/Docker/Ejemplos/docker_data
```

Usaremos una base de datos llamada **MongoDB**, para eso correremos un contenedor de Mongo, con:
`docker run -d --name db mongo`, pero en este caso haremos:

```
1 docker run -d --name db -v /home/quind/Documentos/Clases/Docker/Ejemplos/docker_data/mongodata:/data/db mongo
```

Este comando ejecuta un nuevo contenedor a partir de la imagen "mongo" y lo nombra "db". La opción "-d" indica que el contenedor se ejecutará en segundo plano (en modo detached), es decir, no bloqueará la terminal. La opción "-v" crea un volumen llamado "mongodata" en el host de la máquina y lo asigna al directorio "/data/db" dentro del contenedor. Esto permite persistir los datos de MongoDB más allá del ciclo de vida del contenedor, ya que los datos se almacenan en el volumen del host. De esta manera, cuando se destruye el contenedor, los datos se mantienen en el host.
Ahora ejecutamos el bash en este contenedor con: `docker exec -it db bash`,
y procedemos a ejecutar el binario con: `mongosh`. y podemos mirar que bases de datos tenemos con `show dbs`:

```
test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   40.00 KiB
```

A partir de ahí creamos una nueva base de datos con: `use new_data`, la salida es: `switched to db new_data`.

Insertamos un dato con:

```
db.users.insert({"nombre": "Kevin"})
```

que agrega un nuevo usuario a la colección de usuarios en la base de datos, lo podemos ver con: `db.users.find()`, la salida es:

```
new_data> db.users.find()
[ { _id: ObjectId("63f3a86d68b0191ec379a63f"), nombre: 'Kevin' } ]
```

Y podemos ver en el directorio **mongodata** que hay unos archivos que representan lo que paso en el contenedor, aunque borremos el contenedor. Vamos a borrar el contenedor, primero revisamos los prosesos que tiene "vivos" y los "matamos", eso haciendo: `docker top db`, lo que muestra algo así:

```
UID        PID          PPID         C          STIME          TTY            TIME           CMD
999        12582        12561        0          11:01          ?              00:00:19       mongod --bind_ip_all
```

ahora hacemos: `sudo kill 12582`, luego: `docker rm db`

Y podemos construit un nuevo contenedor para utilizar estos datos, a esto se le llama **bind mounts**. hacemos el comando:

```
1 docker run -d --name db -v /home/quind/Documentos/Clases/Docker/Ejemplos/docker_data/mongodata:/data/db mongo
```

y entramos, con: `docker exec -it db bash`.

Si nos conectamos, con: `mongosh` y hacemos: `show dbs`, vemos:

```
test> show dbs
admin     40.00 KiB
config    60.00 KiB
local     72.00 KiB
new_data  40.00 KiB
```

Vemos que tenemos la base de datos que creamos en el contenedor que anteriormente habíamos borrado. Tambien podemos ver estos datos, por ejemplo:

```
test> use new_data
switched to db new_data
new_data> db.users.find()
[ { _id: ObjectId("63f3a86d68b0191ec379a63f"), nombre: 'Kevin' } ]
```

Hay que tener cuidado con lo directorios que estamos usando, pues el contenedor tiene acceso a este directorio.

4.2. Volumenes

En Docker, los volúmenes son una forma de persistir datos en un contenedor y compartirlos con otros contenedores. Los volúmenes son una abstracción de la capa de almacenamiento que se encuentra debajo del contenedor, y permiten separar los datos de la aplicación de la infraestructura del contenedor.

Un volumen en Docker es un directorio o archivo en el sistema de archivos del host que se monta en el sistema de archivos del contenedor. Los volúmenes pueden ser creados y administrados a través de la línea de comando de Docker o usando un archivo Docker Compose.

Los volúmenes se pueden utilizar para persistir datos de la aplicación, como bases de datos, archivos de configuración, y registros. También pueden ser compartidos entre varios contenedores, lo que permite la creación de una arquitectura de microservicios en la que los contenedores se comunican entre sí y comparten datos a través de volúmenes.

Con el comando `docker volume ls` Podemos ver los volumenes creados, la salida es algo como esto:

```
$ docker volume ls
DRIVER    VOLUME NAME
local     3a9e5ae8100955d1c839ef4c22d9033dae8600eb77f6514022fae9c4a573e6e2
local     4a2d845bd6a1cdebba7a89702c060efbc9d82bc2bf78c800be087d55ca6768b4
local     6dd23903c51c0ad8dbb5e787ca03a134003f74bd95d56e30025b7edc299d61ac
local     67ad0169d8a733154e7e09d24950a73c58e9187b537e66c150ee6cff0606ced9
local     4956f1024f70ad123f577c6b8a2e865678e571471cc3200d915af340014e8288
local     dbfa437e2081f6ae9e1d5dc1f0bd8e5e19645aa659e9a9d7e0cf477a8acc5115
local     ef4cf5d5e84a098d1a1ddc1e45245d82a7a38a67213b43961ca5597bcd76b00f
local     ffef37cbdb5f1e7c550525205af6ddcdb918da977c11657ac4081da6ecba257d
```

El comando "docker volume ls"se utiliza para listar los volúmenes que existen en Docker. En la salida que has proporcionado, los volúmenes se encuentran en la columna "VOLUME NAME" y están representados por una serie de números y letras aleatorios que corresponden a su identificador único. En este caso, los volúmenes son de tipo "local", lo que significa que se encuentran almacenados localmente en la máquina donde se ejecuta Docker. También podemos crear un volumen con el comando: `docker volume create dbdata`

Ahora que tenemos un volumen hagamos el mismo ejemplo anterior:

1. hacemos el comando:

   ```
   docker run -d --name db --mount src=dbdata,dst=/data/db mongo
   ```

   La opción "–mount" se utiliza para crear un volumen de montaje en el contenedor. La sintaxis para la opción es `--mount src=<origen>,dst=<destino>`. En este caso, src=dbdata especifica que se está utilizando un volumen llamado "dbdata" y `dst=/data/db` especifica la ruta en la que se va a montar el volumen dentro del contenedor.

Podemos borrar el contenedor y crear uno con el mismo comando:

```
docker run -d --name db --mount src=dbdata,dst=/data/db mongo
```

Y ahí podemos encontrar los datos que construimos en el contenedor que borramos anteriormente, es decir que los datos persisten aunque el contenedor sea borrado.

4.3. Tmpfs mount

tmpfs es un sistema de archivos temporal en memoria que se puede usar para almacenar datos de manera temporal y efímera en sistemas Linux.

En Docker, los "tmpfs mounts" se pueden usar para crear un sistema de archivos temporal en memoria para un contenedor. La ventaja de usar tmpfs mounts es que los datos se almacenan en la memoria del host, lo que proporciona una alta velocidad de lectura y escritura, y se eliminan cuando se detiene o se borra el contenedor.

Un ejemplo de uso de tmpfs mounts en Docker podría ser para almacenar archivos temporales del sistema de una aplicación que requiere un alto rendimiento en la lectura y escritura de archivos temporales. La sintaxis para crear un tmpfs mount en Docker es similar a la de los bind mounts y volumes, utilizando la opción –mount con el driver tmpfs y especificando la ruta de montaje y opciones adicionales si es necesario.

Un ejemplo de comando de Docker para crear un contenedor que use un tmpfs mount sería el siguiente:

```
docker run -d --name mycontainer --mount type=tmpfs,destination=/app/tmpfs,readonly=false,size=100m myimage
```

En este ejemplo, se está creando un contenedor con el nombre mycontainer, que utiliza un tmpfs mount con un tamaño máximo de 100 MB. El directorio /app/tmpfs se monta en el tmpfs mount, que se usa para almacenar archivos temporales.

4.4. Insertar y extraer datos de un contenedor

Para insertar y extraer datos en un contenedor de Docker, puedes utilizar el siguiente comando:
`docker cp`: te permite copiar archivos o directorios desde o hacia un contenedor Docker en ejecución.
Te muestro cómo:

1. creemos un directorio de trabajo con `mkdir Insertar-extraer`, nos movemos allí, `cd Insertar-extraer` y creamos un archivo en este directorio con `touch pruebas.txt`

2. Ahora creamos un contenedor por ejemplo `docker run -d --name copytest ubuntu tail -f /dev/null`
   Este comando crea un contenedor a partir de la imagen ubuntu, le asigna el nombre copytest y lo ejecuta en segundo plano (-d). El comando "tail -f /dev/null" se utiliza para mantener el contenedor en ejecución sin realizar ninguna otra acción. En lugar de usar este comando, a veces se utiliza el comando "sleep infinity", que tiene el mismo efecto. Este comando es útil cuando se necesita un contenedor en ejecución, pero no es necesario que haga nada activamente, por ejemplo, como un contenedor temporal que se utiliza para copiar datos dentro o fuera del contenedor.

3. Accedemos al contenedor anterior en modo interactivo y abrimos un shell de bash, con `docker exec -it copytest bash`

4. creamos una carpeta en el contendor con `mkdir testing` y nos salimos de este simplemente con el comando `exit`

5. **Insertamos el archivo** “prueba.txt” en el contenedor con el comando `docker cp prueba.txt copytest:/testing/test.txt`, que copia el archivo “prueba.txt” desde la máquina host al contenedor de Docker copytest, en la ruta “/testing/test.txt”.

6. **Extraemos el directorio** “testing” del contenedor con el comando `docker cp copytest:/testing localtesting`, que copia un directorio desde el contenedor “copytest” al host local y lo guarda en el directorio local, llamandolo “localtesting”. En este caso, se está copiando directorio en el directorio “/testing” del contenedor “copytest”.

5. Imagenes

En Docker, una imagen es una plantilla o modelo a partir del cual se crea un contenedor. Es un paquete de software ligero, portátil y autónomo que contiene todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas, las dependencias y las configuraciones.

Las imágenes se construyen a partir de un archivo de configuración llamado Dockerfile, que contiene instrucciones para crear una imagen a partir de un sistema de archivos base. A medida que se ejecutan estas instrucciones, Docker crea capas de imágenes que se acumulan para formar la imagen final. Las imágenes se almacenan en un registro de imágenes de Docker, que puede ser local o remoto. Los registros de imágenes remotos incluyen Docker Hub, el registro de imágenes predeterminado de Docker, y otros registros de imágenes públicos y privados.

Al utilizar imágenes de Docker, los desarrolladores pueden garantizar que sus aplicaciones se ejecuten de manera consistente en diferentes entornos y sistemas, lo que facilita la implementación y la administración de aplicaciones.

5.1. Construcción de una imagen

La construcción de una imagen en Docker se lleva a cabo mediante el uso de un archivo de configuración llamado Dockerfile. Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones que Docker utiliza para crear una imagen de contenedor.
La construcción de una imagen implica seguir los siguientes pasos:

1. Crear un archivo Dockerfile: se debe definir el sistema operativo base y las dependencias necesarias para el contenedor.
2. Ejecutar el comando docker build: una vez que se tiene el Dockerfile, se debe usar el comando "docker build" para construir la imagen del contenedor. Este comando leerá el Dockerfile y creará una imagen de contenedor a partir de las instrucciones que contiene.
3. Enviar la imagen a un registro de contenedores (opcional): si se desea utilizar la imagen en diferentes máquinas, se puede enviar la imagen a un registro de contenedores como Docker Hub para que esté disponible para su descarga.

En el proceso de construcción, Docker crea una serie de capas para la imagen de contenedor. Cada instrucción en el Dockerfile crea una nueva capa en la imagen. Cada capa se puede reutilizar en imágenes de contenedor futuras, lo que permite la creación de imágenes más pequeñas y rápidas.
Te muestro como construir una imagen.

1. Empecemos creando un nuevo directorio y moviendonos allí. `mkdir imagenes` y `cd imagenes/`
2. Creemos un nuevo "Dockerfile", con `touch Dockerfile` y lo abrimos "vs code", con `code .`

3. Lo que escribiremos en el Dockerfile será:

```
FROM ubuntu:latest


RUN touch /usr/src/hola-mundo.txt
```

   - La primera línea es una instrucción "FROM" que indica que se utilizará la imagen "ubuntu:latest" como la imagen base para construir la nueva imagen. La imagen base es la imagen en la que se basa la nueva imagen y sobre la cual se construirá.
   - La segunda línea es una instrucción "RUN" que ejecutará el comando "touch /usr/src/hola-mundo.txt" dentro del contenedor. La instrucción RUN se utiliza para ejecutar comandos durante el proceso de construcción de la imagen. En este caso, el comando crea un archivo llamado "hola-mundo.txt" en la ruta "/usr/src" dentro del contenedor.

4. Ahora vamos a una terminal en este directorio y ejecutamos

```
docker build -t ubuntu:mi_version .
```

   que construye una nueva imagen Docker con la etiqueta "mi_version" a partir del archivo Dockerfile en el directorio actual (".").

   Si hacemos `docker image ls`, vemos la imagen que acabamos de construir:

```
$docker image ls
REPOSITORY     TAG          IMAGE ID       CREATED          SIZE
ubuntu         mi_version   0f7a46bfc059   18 minutes ago   77.8MB
mi-sitio-web   latest       59155fb8a06a   22 hours ago     142MB
nginx          latest       3f8a00f137a0   12 days ago      142MB
mongo          latest       a440572ac3c1   2 weeks ago      639MB
ubuntu         20.04        e40cf56b4be3   2 weeks ago      72.8MB
ubuntu         latest       58db3edaf2be   3 weeks ago      77.8MB
hello-world    latest       feb5d9fea6a5   17 months ago    13.3kB
```

Podemos correr un nuevo contenedor a partir de la imagen anterior:

1. Hagamos `docker run -it ubuntu:mi_version`, necesitamos sber si el archivo "hola-mundo.txt" existe en el contenedor, para eso sólo hacemos `ll /usr/src`, que nos da como respuesta:

```
drwxr-xr-x 1 root root 4096 Feb 21 13:11 ./
drwxr-xr-x 1 root root 4096 Jan 26 02:03 ../
-rw-r--r-- 1 root root    0 Feb 21 13:11 hola-mundo.txt
```

   efectivamente existe.

Podemos publicar esta imagen con `docker login`, logueandote con las credenciales en docher hub,

luego hacemos `docker tag ubuntu:mi_version kevincardenas/ubuntu:mi_version`, "kevincardenas" es mi nombre de usuario, debes usar

el tuyo.

Podemos ver con `docker image ls`:

```
$ docker image ls
REPOSITORY              TAG           IMAGE ID       CREATED              SIZE
kevincardenas/ubuntu    mi_version    0f7a46bfc059   About an hour ago    77.8MB
ubuntu                  mi_version    0f7a46bfc059   About an hour ago    77.8MB
mi-sitio-web            latest        59155fb8a06a   23 hours ago         142MB
nginx                   latest        3f8a00f137a0   12 days ago          142MB
mongo                   latest        a440572ac3c1   2 weeks ago          639MB
ubuntu                  20.04         e40cf56b4be3   2 weeks ago          72.8MB
ubuntu                  latest        58db3edaf2be   3 weeks ago          77.8MB
hello-world             latest        feb5d9fea6a5   17 months ago        13.3kB
```

que hay una "nueva" imagen, en realidad es solo un nuevo tag para poder publicarlo.

Al final simplemente `docker push kevincardenas/ubuntu:mi_version` con resultado:

```
$ docker push kevincardenas/ubuntu:mi_version
The push refers to repository [docker.io/kevincardenas/ubuntu]
3b68e1ecf7cd: Pushed
c5ff2d88f679: Mounted from library/mongo
mi_version: digest: sha256:25916b7347413f131f8d983f8b041795417e92b7526bd58cc1f6af4ec34752ab size: 736
```

Ten cuidado con las imagenes que publicas, y ten en cuenta que después de cierto tiempo de no uso de una imagen, esta puede ser borrada automaticamente.

5.2. Dive

Dive es una herramienta de línea de comandos que permite analizar y explorar las capas de una imagen de Docker. Con Dive, se puede ver una vista previa de los cambios que se realizan en una imagen a medida que se construye, se modifican o se actualizan. La herramienta proporciona una interfaz de usuario interactiva que permite explorar las capas de la imagen y ver cómo se relacionan.

Es muy útil para depurar problemas en las imágenes, identificar archivos innecesarios o redundantes que se agregan a la imagen, y para optimizar el tamaño de la imagen en general. También es útil para detectar posibles vulnerabilidades y problemas de seguridad en la imagen.

Para usar Dive, primero debe instalar la herramienta en su sistema. Luego, puede ejecutar el comando "dive" seguido del nombre de la imagen que desea analizar. La herramienta mostrará una vista previa de la imagen y se pueden explorar las diferentes capas y sus cambios

También ofrece una variedad de opciones y argumentos de línea de comandos para personalizar su análisis de la imagen, como excluir archivos o directorios específicos de la imagen. En general, Dive es una herramienta muy útil para cualquier persona que trabaje con imágenes de Docker y quiera comprender mejor cómo se construyen y cómo se relacionan sus diferentes capas.

Te muestro Cómo instalar esta herramienta en Linux:

1. Agrega el repositorio de dive e instálalo:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E04F0923
sudo add-apt-repository "deb http://ppa.launchpad.net/nazunalika/dive/ubuntu bionic main"
sudo apt update
sudo apt install dive
```

2. Agrega el repositorio de dive a tu sistema:

```
sudo add-apt-repository "deb http://ppa.launchpad.net/nvbn-rm/ppa/ubuntu bionic main"
```

3. Actualiza la lista de paquetes e instala dive:

```
sudo apt update
sudo apt install dive
```

Si los pasos anteriores no funcionan sigue estos pasos:

1. Visite la página de releases de Dive en GitHub: github.com/wagoodman/dive/releases
2. Descargue el paquete correspondiente a su arquitectura y sistema operativo. Si está en Ubuntu, es posible que desee buscar la versión para Debian o para Linux en general.
3. Una vez descargado, abra una terminal y navegue hasta el directorio donde se encuentra el archivo descargado.
4. Ejecute el siguiente comando para instalar Dive:

```
sudo dpkg -i dive_<version>_<arch>.deb
```

Si instalamos la versión "`dive_0.10.0_linux_amd64.deb`" el comando sería

```
sudo dpkg -i dive_0.10.0_linux_amd64.deb
```

5.2.1. docker history

"docker history" es un comando de Docker que te permite ver el historial de una imagen de Docker. Con este comando, puedes ver los distintos cambios que se han realizado en una imagen a lo largo de su desarrollo.
Al ejecutar "docker history", se muestra una lista de todas las capas que componen la imagen, con información adicional para cada capa, como su ID, su tamaño, el comando utilizado para crearla y el autor de la capa. La salida se presenta en orden

cronológico inverso, lo que significa que las capas más recientes aparecen primero en la lista.

El comando "docker history" es útil para entender cómo se construyó una imagen y cómo se crearon las diferentes capas que la componen. También es una herramienta útil para identificar posibles problemas en una imagen, como la inclusión de paquetes o archivos innecesarios, que pueden hacer que la imagen sea más grande de lo necesario. Con esta información, puedes optimizar la imagen y reducir su tamaño para que sea más fácil de transferir y almacenar.

Por ejemplo:

```
$ docker history ubuntu:mi_version
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
0f7a46bfc059   2 hours ago   RUN /bin/sh -c touch /usr/src/hola-mundo.txt... 0B        buildkit.dockerfile.v0
<missing>      3 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      3 weeks ago   /bin/sh -c #(nop) ADD file:18e71f049606f6339... 77.8MB
<missing>      3 weeks ago   /bin/sh -c #(nop)  LABEL org.opencontainers...  0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  LABEL org.opencontainers...  0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH...  0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ARG RELEASE...               0B
```

Se muestra el historial de la imagen "ubuntu:mi_version". La primera columna muestra el ID de la capa, la segunda columna muestra cuándo se creó la capa y la tercera columna muestra el comando que se utilizó para crear esa capa. En este caso, la imagen tiene seis capas. La capa superior, creada hace 2 horas, muestra que se agregó un archivo "hola-mundo.txt" en el directorio "/usr/src" utilizando el comando "touch" en el archivo Dockerfile. La capa inferior no tiene ID y representa la capa base de la imagen, que contiene el sistema de archivos inicial de la imagen "ubuntu:mi_version".

5.2.2. Hagamos lo mismo con dive

El comando dive `ubuntu:mi\_version` utiliza la herramienta "dive" para analizar la capas de la imagen "ubuntu:mi_version" y proporcionar una visualización interactiva y detallada de su contenido.

Por ejemplo:

```
$ dive ubuntu:mi_version
Image Source: docker://ubuntu:mi_version
Fetching image... (this can take a while for large images)
Analyzing image...
Building cache...
```

```
Cmp   Size  Command
78 MB  FROM 7c1cd5e64627ede
0 B  RUN /bin/sh -c touch /usr/src/hola-mundo.txt # buildkit
```

Dive es una herramienta de línea de comandos de código abierto que se utiliza para explorar y analizar imágenes de contenedores Docker. Permite a los usuarios examinar y explorar cada capa de la imagen y proporciona una descripción detallada de los cambios que se producen en cada capa. También proporciona una descripción detallada de las capas que se han agregado o

eliminado en una nueva versión de la imagen.

Al ejecutar el comando `dive ubuntu:mi\_version`, Dive analiza la imagen "ubuntu:mi_version" y muestra una visualización interactiva de las capas de la imagen, lo que permite al usuario navegar por cada capa y ver los cambios que se han producido en ella. Esta herramienta puede ser útil para optimizar y reducir el tamaño de una imagen de contenedor, ya que permite identificar archivos o capas que pueden eliminarse o comprimirse para reducir el tamaño final de la imagen.

6. Bibliografía

Referencias

[1] Docker. *Página oficial de Docker*. Recuperado el 15 de febrero de 2023, de https://www.docker.com/.

[2] Platzi. *Curso de Docker*. Recuperado el 15 de febrero de 2023, de https://platzi.com/cursos/docker/.