

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de EPS
Ingeniería en Ciencias y Sistemas
Prácticas iniciales
Ing. Igor Veliz
Sección F-



Manual básico de Docker

Grupo No. 7

INTEGRANTES:

- 202402587 - Angel Raúl Herrera Chilel
- 202402473 - Kimberly Samantha Gómez Chávez
- 202106435 - María Hercilia Flores Alvarez
- 202200013 - Marcos Aarón Toledo Alvarez
- 202403929 - Emiliana Elizabeth Pú Lara

TUTORES:

- Julio Ruano
- Gabriel Melgar

Índice

1.	Docker	Pág. 3-15
1.1.	¿Qué es Docker?	Pág. 3
1.2.	¿Cómo funciona?	Pág. 3
1.3.	Ventajas	Pág. 3
1.4.	Comandos de Docker	Pág. 3-15
2.	DockerFile	Pág. 15-16
2.1.	¿Qué es DockerFile?	Pág. 15
2.2.	Instrucciones comunes	Pág. 15-16
2.3.	Estructura típica de un DockerFile	Pág. 16
2.4.	Ejemplo	Pág. 16
3.	DockerHub	Pág. 17-18
3.1.	¿Qué es DockerHub?	Pág. 17
3.2.	Ventajas	Pág. 17
3.3.	Como iniciar en DockerHub	Pág. 17-18
4.	Ejemplo de DockerFile y DockerHub	Pág. 18
5.	Cómo correr las imágenes de Docker	Pág. 19
5.1.	Mostrar imagenes corriendo	Pág. 19
5.2.	Aplicación Cliente	Pág. 20
5.3.	Aplicación Servidor	Pág. 20
6.	Foto grupal	Pág. 21
7.	Bibliografia	Pág. 21

Docker

¿Qué es Docker?

Docker es una plataforma de código abierto que permite crear, ejecutar y gestionar contenedores. Fue lanzada en 2013 y facilita el desarrollo y despliegue de aplicaciones en distintos entornos. Aunque fue diseñada para Linux, también funciona en Windows y macOS mediante una capa de virtualización.

¿Cómo funciona?

Docker utiliza funciones del kernel de Linux como espacios de nombres y grupos de control para ejecutar aplicaciones de forma aislada en contenedores. Cada contenedor incluye todo lo necesario para que una aplicación funcione (código, dependencias, configuración), lo que garantiza portabilidad, eficiencia y seguridad.

Ventajas

- Modularidad: Permite separar partes de una aplicación para actualizarlas o repararlas sin afectar a todo el sistema.
- Capas y control de versiones: Cada imagen está compuesta por capas. Los cambios crean nuevas capas, lo que permite reutilizarlas entre imágenes, mejorando eficiencia y facilitando el versionado.
- Restauración: Puedes volver fácilmente a una versión anterior de una imagen si algo falla, lo cual apoya el desarrollo ágil y el uso de CI/CD.
- Implementación rápida: Los contenedores se levantan en segundos, sin necesidad de instalar sistemas operativos completos, lo que reduce tiempo, costo y esfuerzo.

Comandos de Docker

1. docker system

El comando `docker system` ayuda a inspeccionar y gestionar el entorno Docker. Admite argumentos de línea de comandos para realizar tareas específicas relacionadas con la gestión, entre las que se incluyen:

- **df** muestra cómo estás utilizando el disco
- **prune** elimina redes, contenedores, imágenes o volúmenes no utilizados
- **info** muestra información relacionada con el sistema
- **events** muestra un registro de eventos del sistema en tiempo real

A continuación se muestra el mensaje inicial tras ejecutar **docker system prune**.

```
$ docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N]
```

2. docker context

Este comando te ayuda a navegar y configurar diferentes contextos. En Docker, los contextos incluyen nombres, información de seguridad de la capa de transporte, configuraciones de endpoints y orquestadores.

Algunos de los argumentos `docker context`:

- **ls** muestra los detalles del contexto por defecto
- **inspect [CONTEXT]** inspecciona un contexto especificado
- **create [CONTEXT]** crea un nuevo contexto
- **use [CONTEXT]** cambia entre contextos

A continuación se muestra un ejemplo de la salida de **docker context ls**:

```
$ docker context ls
NAME                TYPE                DESCRIPTION                DOCKER ENDPOINT
default             moby                Current DOCKER_HOST based configuration  unix:///var/run/docker.sock
desktop-linux *     moby                Docker Desktop              unix:///Users/b/.docker/run/docker.sock
$
```

3. docker pause and unpause

El comando **docker pause** congela los procesos activos de un contenedor. Para ejecutarlo, debes especificar el nombre del contenedor, como se muestra a continuación:

docker pause [CONTAINER]

A continuación se muestra un ejemplo de la salida que puedes esperar tras pausar un contenedor.

```
$ docker pause devkinsta_db
devkinsta_db
$ █
```

El comando **docker unpause** reanuda los procesos pausados de un contenedor. Al igual que el comando anterior, debes especificar el nombre del contenedor, como se muestra a continuación:

docker unpause [CONTAINER]

A continuación se muestra un ejemplo de la salida que puedes esperar tras desbloquear un contenedor.

```
$ docker unpause devkinsta_db
devkinsta_db
$ █
```

4. docker rm

Este comando elimina contenedores, volúmenes y redes. Permite seleccionar el componente a eliminar en función de sus atributos. Por ejemplo, puedes forzar la eliminación de contenedores en ejecución o de todos los contenedores especificados:

docker rm [CONTAINER] elimina el contenedor cuyo nombre se especifique. La salida de este comando está en la captura de pantalla siguiente.

```
$ docker rm demo_redis
demo_redis
$ █
```

5. docker rmi

Utiliza este comando para eliminar imágenes. Puedes eliminar una sola imagen o varias a la vez. Puedes describir la imagen a eliminar utilizando el ID corto o el ID largo. Este comando es importante para mantener el nodo host limpio y eficiente.

El comando para eliminar imágenes utiliza esta estructura:

docker rmi [IMAGE ID]

A continuación se muestra un ejemplo de su salida.

```
$ docker rmi b089993fa569
Deleted: sha256:b089993fa569d729ff13760d0f4f71640f1795b323100112fe64f6bb439ff258
$
```

6. docker volume

Este comando te permite gestionar volúmenes en Docker. Puedes utilizarlo para crear, eliminar, listar e inspeccionar volúmenes.

Algunos de los argumentos de **docker volume** son:

- **create [OPTIONAL NAME]** crea un nuevo volumen. Puedes especificar el nombre del volumen o dejar que Docker genere un nombre aleatorio.
- **ls** lista los volúmenes disponibles
- **inspect [NAME]** muestra información detallada del volumen.
- **rm [NAME]** elimina un volumen de Docker.

A continuación se muestra un ejemplo de la salida tras crear un volumen.

```
$ docker volume create
28a8bf53a7f1ed9235d5df3e255f75a68674f0d0cbab375e63464c8e75d52798
$
```

7. docker search

Utiliza este comando para buscar imágenes de Docker Hub, que luego podrás ejecutar como contenedores en tu máquina. Te permite acceder a las imágenes del registro de Docker Hub sin visitar el sitio web.

El comando sigue esta estructura: **docker search**. Puedes especificar los nombres de las imágenes que buscas o crear un filtro.

A continuación se muestra un ejemplo de la salida de la siguiente consulta:

docker search --filter is-official=true --filter stars=500 mysql

```
$ docker search --filter is-official=true --filter stars=500 mysql
NAME                DESCRIPTION                                STARS     OFFICIAL   AUTOMATED
mysql               MySQL is a widely used, open-source relation... 14651     [OK]
mariadb             MariaDB Server is a high performing open sou... 5589      [OK]
percona             Percona Server is a fork of the MySQL relati... 622       [OK]
phpmyadmin          phpMyAdmin - A web interface for MySQL and M... 902       [OK]
```

8. docker push

El comando `docker push` te permite compartir tus imágenes en el registro Docker Hub o en un repositorio privado.

La estructura del comando es

docker push [OPTIONS] NAME[:TAG]

- **[OPTIONS]** te permite establecer **-disable-content-trust**.

Por defecto, este valor es verdadero, y no es obligatorio incluirlo.

- **NAME[:TAG]** requiere utilizar el nombre del registro, el repositorio y la etiqueta de imagen.

A continuación se muestra un ejemplo de la salida de **docker push**

```
$ docker push kinsta/longchain:latest
The push refers to repository [docker.io/kinsta/longchain]
42d0f5564b32: Preparing
1e7903969aad: Preparing
9d45042c9b8d: Preparing
e4b78ed727f5: Preparing
d8815e8a268d: Preparing
8655910e6b5f: Waiting
355bb094feb8: Waiting
ed123c9f1a56: Waiting
```

9. docker pull

Este comando descarga una imagen Docker de un repositorio en un registro privado o público.

El comando funciona así

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Este comando te permite utilizar imágenes existentes en lugar de crear otras nuevas siempre que debas crear una aplicación en contenedores.

El ejemplo siguiente muestra la salida de un comando **docker pull**:

```
$ docker pull kuria/spring-mvc-app:0.1
0.1: Pulling from kuria/spring-mvc-app
08c01a0ec47e: Pull complete
fbc7fc430c8c: Pull complete
f4cbdc672596: Pull complete
Digest: sha256:e734fef44c2fefbf7a746a605db691cf174cf8db0b08d4aa2aea1f708097a14c
Status: Downloaded newer image for kuria/spring-mvc-app:0.1
docker.io/kuria/spring-mvc-app:0.1
```

10. docker ps

Por defecto, este comando muestra la lista de todos los contenedores en ejecución. Sin embargo, puedes añadir una bandera para listar los contenedores en función de atributos como el tamaño de uso del disco, los contenedores enlazados y las etiquetas.

El comando sigue la siguiente estructura:

docker ps [OPTIONS]

Algunos de sus argumentos son

- **-a** muestra una lista de los contenedores en ejecución y de los que han finalizado
- **-s** muestra el tamaño en disco y el tamaño virtual de cada contenedor

Puedes utilizar los dos juntos así:

docker ps -as

A continuación se muestra un ejemplo de la salida de un comando **docker ps**.

```
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
db5a5d9c8ab6   kinsta/devkinsta_nginx:1.3.2       "/docker-entrypoint..." 2 months ago   Up 26 hours
46312a354270   kinsta/devkinsta_adminer:1.3.2     "docker-php-entrypoi..." 2 months ago   Up 26 hours
45b0643d1243   kinsta/devkinsta_fpm:1.3.2         "/usr/local/sbin/kin..." 2 months ago   Up 26 hours
4944fd541e48   kinsta/devkinsta_mailhog:1.3.2     "MailHog"                2 months ago   Up 26 hours
8a3d6a075ca7   kinsta/devkinsta_db:1.3.2          "docker-entrypoint.s..." 2 months ago   Up About an hour
$
```

11. docker tag

Utiliza esta etiqueta para añadir metadatos, como la versión, a tu imagen. Las etiquetas suelen crearse cuando se construye una imagen, pero el comando `docker tag` te permite añadir una etiqueta más tarde, creando esencialmente un alias para la imagen de destino.

Este comando sigue la siguiente estructura:

docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

En el ejemplo siguiente, listamos imágenes con el nombre «redis». Tras etiquetar la imagen con un número de versión («2.0»), aparecen en la lista el nuevo alias y la imagen etiquetada originalmente.

```
$ docker images redis
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
redis         latest   da63666bbe9a   2 months ago   138MB

$ docker tag redis:latest redis:2.0 ←

$ docker images redis
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
redis         2.0       da63666bbe9a   2 months ago   138MB
redis         latest   da63666bbe9a   2 months ago   138MB
$
```

12. docker rename

Utiliza este comando para renombrar un contenedor. Es útil cuando tienes varios contenedores y quieres diferenciarlos en función de su finalidad.

Este comando sigue la siguiente estructura:

docker rename [OLD_NAME] [NEW_NAME]

A continuación se muestra un ejemplo de salida antes y después de un comando **docker rename**.

```
[ $ docker ps -a -f name=feed
CONTAINER ID   IMAGE                COMMAND                  NAMES
b9d58a4bad03   feedback-node:volumes "docker-entrypoint.s..." feedback-app

[ $ docker rename feedback-app feedback-demo ←
[ $ docker ps -a -f name=feed
CONTAINER ID   IMAGE                COMMAND                  NAMES
b9d58a4bad03   feedback-node:volumes "docker-entrypoint.s..." feedback-demo
$
```

13. docker commit

Este comando te permite crear nuevas imágenes después de realizar cambios en los archivos de un contenedor. Esto es importante porque te permite depurar un contenedor utilizando un shell interactivo.

Este comando sigue la siguiente estructura

docker commit [CONTAINER_ID] [name-of-new-image]

A continuación se muestra un ejemplo y una salida del comando **docker commit**.

```
$ docker commit 8a3d6a075ca7 kinsta/longchain:latest
sha256:ad424e1f5ccb2d8b337f3ac5c86372f75ee931fd2247f8c99b6bf98aeac681db
```

14. docker network

Es un comando de gestión de red que te permite crear potentes aplicaciones conectando servicios y contenedores.

El comando tiene la siguiente estructura:

docker network [OPTIONS]

Sus argumentos incluyen:

- **connect** para conectar contenedores a redes
- **create** para crear nuevas redes
- **disconnect** para desconectar de las redes los contenedores en ejecución
- **rm** para eliminar una o varias redes

A continuación se muestra la salida de un comando **docker network create**.

```
$ docker network create my-network  
c3ddf22c75d0d07709ffff876ecfacea25c415b22507f244fe1885206c9b03bb  
$
```

15. docker history

Este comando proporciona el historial de una imagen especificada, ayudándote a comprender cómo se creó y mostrando el tamaño de la imagen.

El comando tiene la siguiente estructura:

docker history [IMAGE]

A continuación, vemos el historial asociado a la imagen **redis:latest**.

```
$ docker history redis:latest
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
da6366bbe9a    2 months ago    /bin/sh -c #(nop)  CMD ["redis-server"]      0B
<missing>      2 months ago    /bin/sh -c #(nop)  EXPOSE 6379                  0B
<missing>      2 months ago    /bin/sh -c #(nop)  ENTRYPOINT ["docker-entry...             0B
<missing>      2 months ago    /bin/sh -c #(nop)  COPY file:e873a0e3c13001b5...             661B
<missing>      2 months ago    /bin/sh -c #(nop)  WORKDIR /data                                0B
<missing>      2 months ago    /bin/sh -c #(nop)  VOLUME [/data]                              0B
<missing>      2 months ago    /bin/sh -c mkdir /data && chown redis:redis ... 0B
<missing>      2 months ago    /bin/sh -c set -eux; savedAptMark="$(apt-m... 58.8MB
<missing>      2 months ago    /bin/sh -c #(nop)  ENV REDIS_DOWNLOAD_SHA=5c...      0B
<missing>      2 months ago    /bin/sh -c #(nop)  ENV REDIS_DOWNLOAD_URL=ht...      0B
<missing>      2 months ago    /bin/sh -c #(nop)  ENV REDIS_VERSION=7.2.1                0B
<missing>      2 months ago    /bin/sh -c set -eux; savedAptMark="$(apt-ma... 4.12MB
<missing>      2 months ago    /bin/sh -c #(nop)  ENV GOSU_VERSION=1.16                0B
<missing>      2 months ago    /bin/sh -c groupadd -r -g 999 redis && usera... 4.3kB
<missing>      2 months ago    /bin/sh -c #(nop)  CMD ["bash"]                  0B
<missing>      2 months ago    /bin/sh -c #(nop)  ADD file:a1398394375faab8d...             74.8MB
$
```

16. docker update

Este comando te permite actualizar la configuración de un contenedor. Ayuda a evitar que los contenedores consuman demasiados recursos del host Docker. El formato del comando es

docker update [OPTIONS] [CONTAINER]

Algunas de sus opciones son:

- **--restart** actualiza la política de reinicio de un contenedor
- **--memory** establece el límite de memoria de un contenedor
- **--cpus** establece el número de CPUs asignadas

A continuación se muestra un ejemplo de salida de un comando **docker update**.

```
$ docker update --restart=on-failure:3 devkinsta_db
devkinsta_db
$
```

7. docker plugin install

Este comando te permite gestionar plugins. Es esencial porque permite añadir nuevas funcionalidades sin alterar las configuraciones del host Docker.

Los argumentos de **docker plugin** incluyen:

- **create** para crear nuevos plugins
- **enable** para activar plugins instalados
- **install** para instalar nuevos plugins
- **rm** para eliminar uno o varios plugins
- **ls** mostrar una lista de plugins

A continuación, utilizamos **docker plugin install** para añadir un plugin a nuestro entorno. A continuación, utilizamos **docker plugin ls** para mostrar su estado.

```
$ docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
- network: [host]
- mount: [/var/lib/docker/plugins/]
- mount: []
- device: [/dev/fuse]
- capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
Digest: sha256:1d3c3e42c12138da5ef7873b97f7f32cf99fb6edde75fa4f0bcf9ed277855811
52d435ada6a4: Complete
Installed plugin vieux/sshfs

$ docker plugin ls
ID                NAME                DESCRIPTION                ENABLED
752179f97d15     vieux/sshfs:latest  sshFS plugin for Docker    true
$
```

18. docker container

Este comando te permite gestionar contenedores. Lo utilizas para realizar acciones como crear, detener y eliminar contenedores, entre otras.

Las opciones de **docker container** incluyen:

- **create** para crear un contenedor
- **commit** para crear una nueva imagen después de realizar cambios en un contenedor
- **exec** para ejecutar comandos dentro de un contenedor en ejecución
- **kill** para detener un contenedor en ejecución
- **ls** para mostrar una lista de contenedores dentro de un host Docker
- **restart** reiniciar un contenedor
- **run** crear un contenedor a partir de una imagen y ejecutarlo

- **rm** eliminar un contenedor de un servidor Docker

A continuación se muestra un ejemplo de la salida de un comando **docker container**.

```
$ docker container restart devkinsta_db
devkinsta_db
$ █
```

19. docker logs

Este comando recupera los registros de un contenedor. Proporciona información sobre las operaciones de un contenedor, que puede ser esencial a la hora de depurar.

A continuación se muestra un ejemplo de la salida de un comando **docker logs**.

```
$ docker logs devkinsta_db
2023-09-25 15:54:30+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.17+maria~ubu2004 started.
2023-09-25 15:54:30+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-09-25 15:54:30+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.17+maria~ubu2004 started.
2023-09-25 15:54:31+00:00 [Note] [Entrypoint]: Initializing database files
```

20. docker swarm

Este comando ayuda a gestionar un enjambre Docker (Docker swarm) — un grupo de máquinas (virtuales o físicas) que ejecutan Docker y están configuradas para trabajar juntas en un clúster. Este comando facilita la configuración de un enjambre y el disfrute de sus ventajas.

A continuación se indican algunas de las opciones de **docker swarm**:

- **init** para iniciar un enjambre
- **join** para unirse a un enjambre
- **leave** para abandonar un enjambre
- **update** para actualizar un enjambre

A continuación se muestra un ejemplo y la salida de un comando **docker swarm init**.

```
$ docker swarm init
Swarm initialized: current node (oq1owxske37kzscqee2decxzv) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-64phenho97jxzxa6hu3dc692tjcethrw5dcxk0l13k19sqnrner-e8mpbv2u
    zm6qtcgro5stwajtb 192.168.65.4:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

$
```

DockerFile

¿Qué es un DockerFile?

Un DockerFile es un documento basado en texto que se usa para crear una imagen de contenedor. Proporciona instrucciones al creador de imágenes sobre los comandos que se deben ejecutar, los archivos que se deben copiar, el comando de inicio y más.

Instrucciones comunes:

Algunas de las instrucciones más comunes en un DockerFile incluyen:

- **FROM <image>** - Esto especifica la imagen base que extenderá la compilación.
- **WORKDIR <path>** - Esta instrucción especifica el "directorio de trabajo" o la ruta en la imagen donde se copiarán los archivos y se ejecutarán los comandos.
- **COPY <host-path> <image-path>** - Esta instrucción le dice al compilador que copie archivos del host y los coloque en la imagen del contenedor.
- **RUN <command>** - Esta instrucción le dice al compilador que ejecute el comando especificado.
- **ENV <name> <value>** - Esta instrucción establece una variable de entorno que usará un contenedor en ejecución.
- **EXPOSE <port-number>** - Esta instrucción establece la configuración en la imagen que indica un puerto que la imagen desea exponer.
- **USER <user-or-uid>** - Esta instrucción establece el usuario predeterminado para todas las instrucciones posteriores.

- **CMD** ["<command>", "<arg1>"] - Esta instrucción establece el comando predeterminado que ejecutará un contenedor que use esta imagen.

Estructura típica de un DockerFile:

Un DockerFile común suele seguir estos pasos:

1. Definir la imagen base (FROM)
2. Establecer el directorio de trabajo (WORKDIR)
3. Instalar dependencias (RUN)
4. Copiar archivos necesarios (COPY)
5. Configurar puertos, variables y usuarios (EXPOSE, ENV, USER)
6. Definir el comando de inicio (CMD)

Ejemplo: El siguiente DockerFile produciría una aplicación Python lista para ejecutar:

```
FROM python:3.12
```

```
WORKDIR /usr/local/app
```

```
# Instalar las dependencias de la aplicación
```

```
COPY requirements.txt ./
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Copiar el código fuente
```

```
COPY src ./src
```

```
EXPOSE 5000
```

```
# Configurar un usuario de la aplicación para que el contenedor no se ejecute como usuario root
```

```
RUN useradd app
```

```
USER app
```

```
# Comando de inicio
```

```
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```


DockerHub

¿Qué es DockerHub?

Docker Hub es el registro público predeterminado para almacenar, compartir y obtener imágenes de contenedores. Funciona como una biblioteca global donde puedes:

- Buscar y usar imágenes preconstruidas, como Nginx, MySQL, Node.js, entre otras. Estas imágenes suelen estar verificadas (Docker Official Images), lo que garantiza calidad y seguridad.
- Subir tus propias imágenes para compartirlas con tu equipo o con millones de desarrolladores.
- Acelerar flujos de trabajo al evitar configuraciones complejas: simplemente obtienes y ejecutas contenedores listos.

Ventajas:

- Biblioteca enorme de imágenes listas para usar.
- Integración directa con Docker CLI y Docker Desktop.
- Ideal para compartir proyectos o configurar entornos rápidamente.

Como iniciar en DockerHub:

1. Crear una cuenta

Visita <https://hub.docker.com> y regístrate gratuitamente.

2. Conectar con Docker desde la terminal

Usa el siguiente comando: **docker login**

Esto te pedirá tu usuario y contraseña de Docker Hub.

3. Buscar imágenes oficiales

- En la web: ve a la sección "Explore" en Docker Hub.
- En la terminal:
docker search nginx --filter "is-official=true"

4. Subir tu propia imagen (docker push)

Primero crea una imagen local y etiquétala con tu nombre de usuario:

docker build -t TU_USUARIO/mi-imagen .

Luego, súbela al repositorio:

docker push TU_USUARIO/mi-imagen

Si el repositorio no existe, Docker lo crea automáticamente al hacer el push.

Ejemplo DockerFile y Dockerhub:

1. Crear un Dockerfile básico

```
FROM nginx
```

```
RUN echo "<h1>Hello world from Docker!</h1>" > /usr/share/nginx/html/index.html
```

- Descripción: Esta instrucción parte desde la imagen base nginx de Docker Hub y crea una página web simple que se mostrará automáticamente al ejecutar el contenedor.

2. Construir la imagen localmente

```
docker build -t TU_USUARIO/nginx-custom .
```

- Aclaración: Asegúrate de reemplazar TU_USUARIO con tu nombre de usuario de Docker Hub.

3. Ejecutar la imagen construida

```
docker run -p 8080:80 --rm TU_USUARIO/nginx-custom
```

Luego, abre un navegador en: <http://localhost:8080>

Deberías ver el mensaje: "Hello world from Docker!"

4. Subir la imagen a Docker Hub

- Inicia sesión desde la terminal:

```
docker login
```
- Empuja la imagen hacia tu repositorio en Docker Hub:

```
docker push TU_USUARIO/nginx-custom
```

Docker creará el repositorio automáticamente si no existe.

Mostrar las imágenes corriendo actualmente y una manera de consumirlas.

Ver las imágenes corriendo actualmente en Docker:

Para ver todas las imágenes que tienes descargadas en tu sistema, usa:

```
bash
docker images
```

Este comando te mostrará una tabla similar a esta:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cliente-app	latest	a1b2c3d4	2 days ago	350MB
servidor-api	v1.0	e5f6g7h8	3 days ago	420MB
mysql	8.0	x9y1z2w3	1 week ago	500MB

Para ver los contenedores que están ejecutándose actualmente, usa:

```
bash
docker ps
```

Ejemplo de salida.

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS	NAMES
1a2b3c4d	cliente-app	"npm start"	Up 10 minutes	0.0.0.0:8080->3000	cliente_web
5e6f7g8h	servidor-api	"node server.js"	Up 10 minutes	0.0.0.0:5000->5000	servidor_backend

Una vez que los contenedores están corriendo, puedes acceder a sus servicios según los puertos expuestos.

Consumir la aplicación cliente

Si tu contenedor de cliente mapea el puerto 3000 del contenedor al 8080 de tu máquina:

```
bash

http://localhost:8080
```

Consumir la aplicación servidor (API)

Para probar que la API funciona, puedes hacer una petición GET usando `curl` o desde tu navegador.

Por ejemplo, si el servidor expone el puerto 5000:

```
bash

curl http://localhost:5000
```

O bien abrir en el navegador:

```
bash

http://localhost:5000
```

Si la API tiene un endpoint, por ejemplo `/usuarios`, sería:

```
bash

curl http://localhost:5000/usuarios
```

Foto grupal



Bibliografía

- **Red Hat.** (s.f.). *What is Docker?* Red Hat. Recuperado de <https://www.redhat.com/es/topics/containers/what-is-docker>
- **DataScientest.** (9 de agosto, s.f.). *Docker: ¿qué es y cómo se usa?* DataScientest. Recuperado de <https://datascientest.com/es/docker-todo-que-saber>
- **Docker.** (s.f.). *Writing a Dockerfile.* Documentación oficial de Docker. Recuperado de <https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/>
- **Docker.** (s.f.). *Quickstart (Docker Hub).* Documentación oficial de Docker. Recuperado de <https://docs.docker.com/docker-hub/quickstart/>