

Las imágenes de Docker son plantillas o plantillas de solo lectura que contienen todo lo necesario para ejecutar una aplicación o servicio en un contenedor de Docker. En términos simples, una imagen de Docker es un paquete autónomo que incluye todo lo necesario para ejecutar una pieza de software, como el código fuente, las bibliotecas, las dependencias, las variables de entorno y las configuraciones.

Repositorios de Imágenes:

Existen diferentes repositorios públicos de donde descargar nuestras imágenes.

- Docker Hub

```
docker pull nginx
```

- ECR (public.ecr.aws)

```
docker pull public.ecr.aws/amazonlinux/amazonlinux:2.0.20240620.0-arm64v8
```

- GC (<https://console.cloud.google.com/gcr/images/google-containers/global/busybox>)

```
docker pull gcr.io/google-containers/busybox:1.27
```

Repositorios privados

Docker Registry

<https://docs.docker.com/registry/deploying/>

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

Harbor

<https://goharbor.io/>

<https://medium.com/kocsistem/harbor-installation-on-docker-65949ebb8530>

Imágenes Oficiales

Docker proporciona una serie de imágenes oficiales en su Docker Hub (<https://hub.docker.com/>) que son mantenidas y respaldadas por el equipo de Docker. Estas imágenes oficiales están diseñadas para ser utilizadas como base para construir tus propias imágenes de contenedor o para ejecutar aplicaciones en contenedores.

Algunas de las imágenes oficiales populares de Docker son:

- **Ubuntu:** Imagen oficial de Ubuntu, un sistema operativo basado en Linux ampliamente utilizado.
- **Alpine:** Imagen oficial de Alpine Linux, una distribución Linux minimalista y ligera.
- **Nginx:** Imagen oficial del servidor web Nginx.
- **MySQL:** Imagen oficial de MySQL, una base de datos relacional.
- **PostgreSQL:** Imagen oficial de PostgreSQL, otra base de datos relacional de código abierto.
- **Node.js:** Imagen oficial de Node.js, un entorno de ejecución de JavaScript.
- **Python:** Imagen oficial de Python, un lenguaje de programación popular.
- **Redis:** Imagen oficial de Redis, una base de datos en memoria de código abierto.
- **MongoDB:** Imagen oficial de MongoDB, una base de datos NoSQL orientada a documentos.
- **WordPress:** Imagen oficial de WordPress, una plataforma de gestión de contenido.

Estas son solo algunas de las imágenes oficiales disponibles en el Docker Hub. Puedes explorar el Docker Hub para encontrar más imágenes oficiales y buscar imágenes de otros proveedores o comunidades que también estén disponibles para su uso en Docker.

Imagen oficial de Nginx

https://hub.docker.com/_/nginx

```
docker pull nginx
```

Tags

En Docker, los tags (etiquetas) se utilizan para identificar versiones específicas o variantes de una imagen. Proporcionan una forma de diferenciar entre diferentes versiones de la misma imagen, lo que te permite administrar e implementar versiones específicas de tus contenedores.

Un tag de Docker está compuesto típicamente por dos partes: el repositorio y el tag en sí. El repositorio identifica la imagen de Docker, mientras que el tag especifica una versión o variante particular de esa imagen.

Aquí tienes algunas prácticas comunes para usar los tags de Docker:

1. **Tag "latest"**: De forma predeterminada, si no se especifica un tag, Docker asume el tag "latest" (último). Representa la versión más reciente de la imagen. Por ejemplo, `ubuntu:latest` se refiere a la imagen más reciente de Ubuntu.
2. **Versionamiento semántico**: Muchas imágenes de Docker siguen las convenciones de versionamiento semántico. Esto te permite especificar una versión particular de la imagen basada en los tags proporcionados. Por ejemplo, `nginx:1.16.5` se refiere a la versión 1.16.5 de la imagen de Nginx.
3. **Aliases**: Los tags de Docker también pueden usar aliases, que son etiquetas legibles por humanos que representan versiones específicas de la imagen. Los aliases se utilizan a menudo para denotar versiones estables, beta o de candidatos a lanzamiento. Por ejemplo, `python:3.9-alpine` se refiere a la versión 3.9 de la imagen de Python basada en Alpine Linux.
4. **Tags personalizados**: Además de las prácticas estándar, también puedes crear tags personalizados para adaptarse a tus necesidades. Estos se pueden utilizar para diferenciar entre diferentes variantes de compilación, configuraciones o características específicas.

Al realizar la descarga (pull) o ejecución (run) de imágenes de Docker, puedes especificar el tag utilizando la sintaxis `imagen:tag`.

Es importante tener en cuenta que las imágenes de Docker pueden tener múltiples tags asociados. Esto te permite acceder a diferentes versiones o variantes de la misma imagen utilizando tags diferentes.

En resumen, los tags de Docker proporcionan una forma conveniente de administrar y controlar las versiones de tus aplicaciones en contenedores. Ayudan a garantizar la consistencia y la reproducibilidad en las implementaciones y simplifican el proceso de actualización o reversiones a versiones específicas de imágenes.

hub.docker.com/_/nginx/tags

dockerhub nginx

Explore Pricing Sign In Register

Explore Official Images nginx

NGINX **nginx** DOCKER OFFICIAL IMAGE · 1B+ · 10K+
Official build of Nginx.

docker pull nginx

Overview **Tags**

Sort by Newest Filter Tags

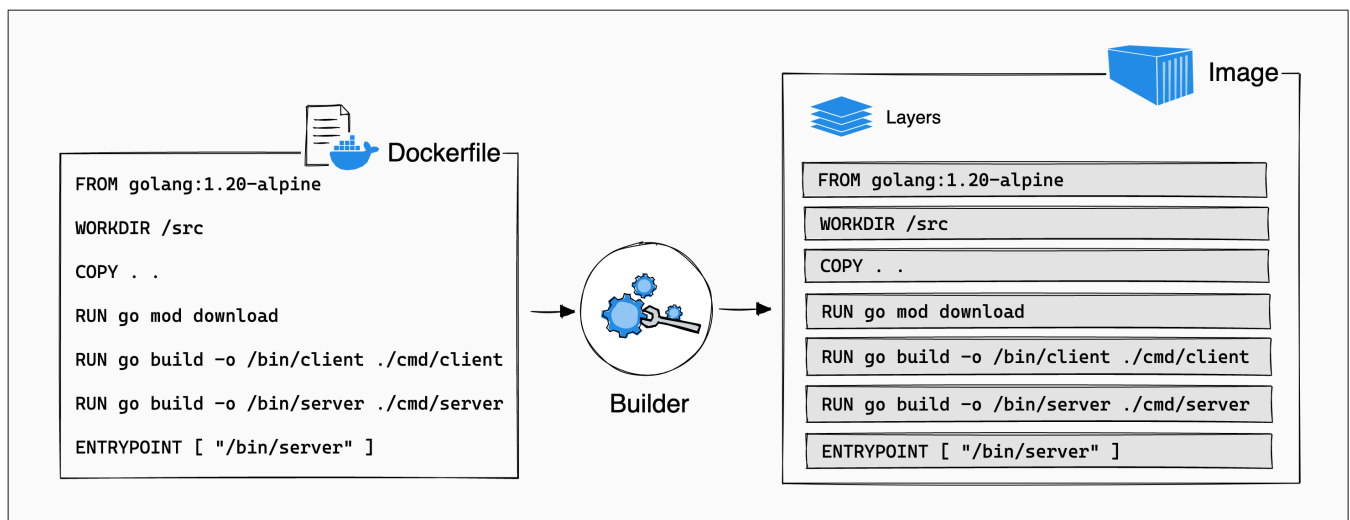
| TAG | DIGEST | OS/ARCH | VULNERABILITIES | COMPRESSED SIZE |
|--|--|--|--|-------------------------------|
| stable-alpine3.17-slim Last pushed 17 days ago by doijanky | cf8f11464780 da86ecb516d8 da5314a345a8 +4 more... | linux/386 linux/amd64 linux/arm/v6 | None found None found None found | 5.21 MB 4.94 MB 4.78 MB |

docker pull nginx:stable-alpine3...

```
docker pull ubuntu:latest
docker pull nginx:stable-alpine
or
docker run nginx:stable-alpine-slim
```

Docker Layers


El orden de las instrucciones de Dockerfile es importante. Una compilación de Docker consta de una serie de instrucciones de compilación ordenadas. Cada declaración en un Dockerfile se traduce aproximadamente en una capa de imagen. El siguiente diagrama ilustra cómo un Dockerfile se traduce en una pila de capas en una imagen de contenedor.



Capas en caché

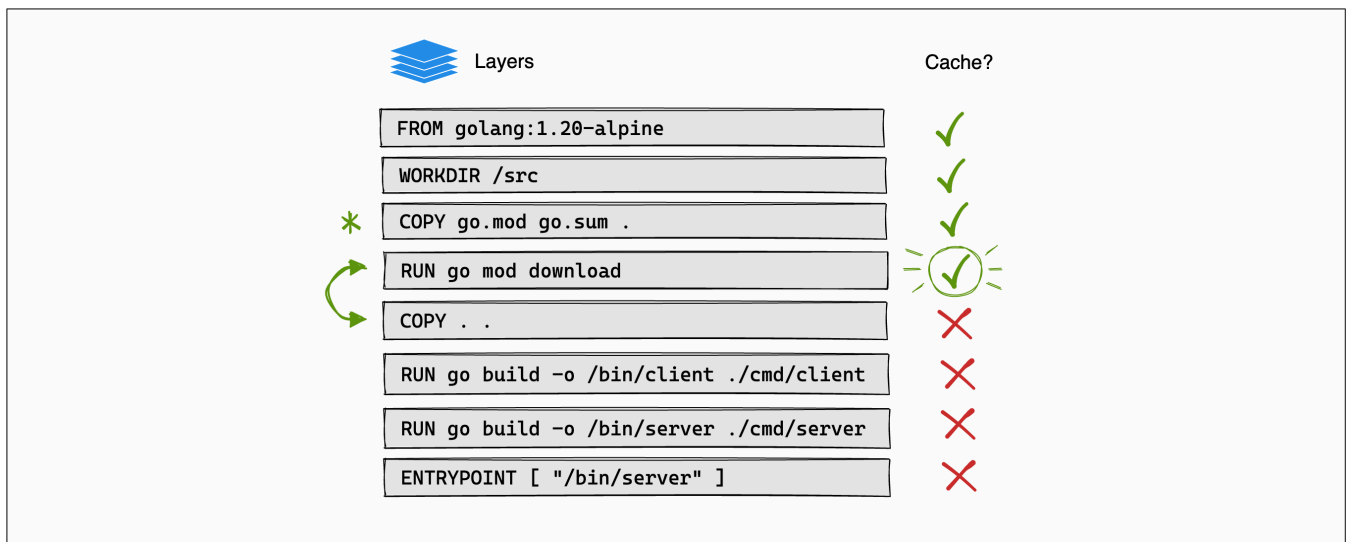
Cuando ejecuta una compilación, el constructor intenta reutilizar capas de compilaciones anteriores. Si una capa de una imagen no ha cambiado, el constructor la recoge de la memoria caché de construcción. Si una capa ha cambiado desde la última compilación, esa capa y todas las capas siguientes deben reconstruirse.

El Dockerfile de la sección anterior copia todos los archivos del proyecto en el contenedor (COPY . .) y luego descarga las dependencias de la aplicación en el siguiente paso (RUN go mod download). Si cambiara alguno de los archivos del proyecto, eso invalidaría el caché para la capa COPY. También invalida el caché para todas las capas que siguen.

|  Layers | Cache? |
|--|--------|
| FROM golang:1.20-alpine | ✓ |
| WORKDIR /src | ✓ |
| COPY . . | ✗ |
| RUN go mod download | ✗ |
| RUN go build -o /bin/client ./cmd/client | ✗ |
| RUN go build -o /bin/server ./cmd/server | ✗ |
| ENTRYPOINT ["/bin/server"] | ✗ |

Actualiza el orden de las instrucciones

```
# syntax=docker/dockerfile:1
FROM golang:1.20-alpine
WORKDIR /src
- COPY . .
+ COPY go.mod go.sum .
  RUN go mod download
+ COPY . .
  RUN go build -o /bin/client ./cmd/client
  RUN go build -o /bin/server ./cmd/server
  ENTRYPOINT [ "/bin/server" ]
```



```
Esteban (-zsh)
esteban ~ % docker pull nginx:stable-alpine-slim
stable-alpine-slim: Pulling from library/nginx
ed6b6dbacee9: Already exists
4c7f12338fe3: Already exists
002a136ea5c5: Already exists
6d407d2ad632: Already exists
d1543f6e84d3: Already exists
ad428fb17e98: Already exists
Digest: sha256:31491c883c5e56aeb96d62e663256a7359a1937803be4269b775dda244cc051
Status: Downloaded newer image for nginx:stable-alpine-slim
docker.io/library/nginx:stable-alpine-slim
esteban ~ %
```

Dangling Images

Las imágenes **Dangling** son imágenes de Docker sin etiquetar que no utiliza un contenedor ni depende de un descendiente. Por lo general, no sirven para nada, pero aún consumen espacio en disco. Acumulará imágenes colgantes cuando reemplace una etiqueta existente al comenzar una nueva compilación.

Todas las imágenes **Dangling** se muestran como **none:none** en la CLI de Docker. Tener demasiados de ellos puede ser abrumador cuando tienes docenas de imágenes sin información sobre su verdadera identidad. La ejecución periódica de

```
docker image prune
or
docker system prune
```

Primer Imagen Docker

```
FROM Ubuntu:latest
RUN apt update
RUN apt install nginx
```

```
esteban ~/Projects/Personal/DockerCurs/nginx % docker ps -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------------|---------|---------------|--------------------------|-------|----------------|
| 409505cb9378 | nginx_custom | "sh" | 3 seconds ago | Exited (0) 3 seconds ago | | dreamy_galileo |

Run nginx server in foreground

```
FROM Ubuntu:latest
RUN apt update
RUN apt install -y nginx
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
esteban ~/Projects/Personal/DockerCurs/nginx % docker ps -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|--------------|---------------------------|--------------------|-------------------------------|-------|-----------------|
| 29ee45892ef9 | nginx_custom | "nginx -g 'daemon of...'" | 5 seconds ago | Exited (0) 2 seconds ago | | laughing_golick |
| 409505cb9378 | 95a44b2b18ac | sh | About a minute ago | Exited (0) About a minute ago | | dreamy_galileo |

```
docker history nginx_custom
```

```
esteban ~/Projects/Personal/DockerCurs/nginx % docker history nginx_custom
```

| IMAGE | CREATED | CREATED BY | SIZE | COMMENT |
|-------------|----------------|---|--------|------------------------|
| 2dde505d533 | 11 minutes ago | CMD ["nginx" "-g" "daemon off;"] | 0B | buildkit.dockerfile.v0 |
| <missing> | 11 minutes ago | EXPOSE map[80/tcp:{}] | 0B | buildkit.dockerfile.v0 |
| <missing> | 11 minutes ago | RUN /bin/sh -c apt install -y nginx # buildk... | 44.3MB | buildkit.dockerfile.v0 |
| <missing> | 12 minutes ago | RUN /bin/sh -c apt update # buildkit | 7.53MB | buildkit.dockerfile.v0 |
| <missing> | 4 days ago | CMD ["sh"] | 0B | buildkit.dockerfile.v0 |
| <missing> | 4 days ago | RUN /bin/sh -c apt-get -y install build-esse... | 276MB | buildkit.dockerfile.v0 |
| <missing> | 4 days ago | RUN /bin/sh -c apt-get update # buildkit | 38.4MB | buildkit.dockerfile.v0 |
| <missing> | 4 days ago | USER root | 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:1043594b482384e96... | 69.2MB | |
| <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 | |

Imagen Docker repositorio oficial:

<https://github.com/nginxinc/docker->

Dockerfile

<https://docs.docker.com/engine/reference/builder/>

<https://docs.docker.com/build/buildkit/>

Un Dockerfile es un archivo de texto que contiene instrucciones para construir una imagen de Docker. Estas instrucciones describen los pasos necesarios para configurar el entorno de ejecución dentro de un contenedor Docker.

El Dockerfile es utilizado por el comando `docker build` para construir la imagen a partir del contexto actual. El contexto es el directorio local que contiene el Dockerfile y los archivos relacionados necesarios para la construcción de la imagen.

A continuación, se muestra una descripción básica de algunas instrucciones comunes que se pueden utilizar en un Dockerfile:

- `FROM` : Especifica la imagen base a partir de la cual se construirá la nueva imagen. Debe ser la primera instrucción en el Dockerfile.
- `RUN` : Ejecuta un comando en el sistema de archivos de la imagen durante el proceso de construcción. Se utiliza para instalar paquetes, ejecutar comandos de configuración, entre otros.
- `COPY` o `ADD` : Copia archivos o directorios desde el contexto local a la imagen.
- `WORKDIR` : Establece el directorio de trabajo dentro de la imagen para las instrucciones siguientes.
- `ENV` : Establece variables de entorno en la imagen.
- `EXPOSE` : Especifica los puertos en los que el contenedor estará escuchando en tiempo de ejecución.
- `CMD` o `ENTRYPOINT` : Define el comando que se ejecutará cuando se inicie un contenedor basado en la imagen. Puede haber solo una instrucción `CMD` o `ENTRYPOINT` en un Dockerfile.

Estas son solo algunas de las instrucciones más utilizadas en un Dockerfile. Hay muchas otras instrucciones disponibles para realizar tareas más avanzadas, como configuración de red, configuración de usuarios, configuración de volúmenes, entre otras.

Una vez que se ha creado el Dockerfile, se puede ejecutar el comando `docker build` para construir la imagen. Por ejemplo:

```
docker build -t nombre_de_la_imagen .
```


La opción `-t` permite especificar el nombre y la etiqueta de la imagen resultante.

El uso de un Dockerfile permite automatizar el proceso de construcción de imágenes de Docker y garantizar la reproducibilidad de los entornos de contenedor. También facilita la colaboración y la distribución de aplicaciones en contenedores de Docker.

Las variables de entorno son compatibles con la siguiente lista de instrucciones en Dockerfile:

- `ADD` ✓
- `COPY` ✓
- `ENV` ✓
- `EXPOSE` ✓
- `FROM` ✓
- `LABEL` ✓
- `STOPSIGNAL` ✓
- `USER` ✓
- `VOLUME` ✓
- `WORKDIR` ✓
- `ONBUILD` (cuando se combina con una de las instrucciones compatibles anteriores) ✓

FROM

```
FROM ubuntu:latest
```

RUN

```
FROM ubuntu:latest
RUN apt update
```

COPY/ADD

COPY solo admite la copia básica de archivos locales en el contenedor, mientras que **ADD** tiene algunas características (como la extracción local de archivos tar y la compatibilidad con URL remotas) que no son obvias de inmediato. En consecuencia, el mejor uso para ADD es la extracción automática del archivo tar local en la imagen, como en `ADD rootfs.tar.xz /`.

```
index.html
<H1> LLEIDA </H1>
```

```
FROM ubuntu:latest
RUN apt update
RUN apt install -y nginx
COPY . /var/www/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

← → ↻ ⓘ localhost

LLEIDA

ADD

```
index.html
```

```
<H1>LLEIDA TAR </H1>
```

```
tar cvf index.tar index.html
```

```
FROM ubuntu:latest
RUN apt update
RUN apt install -y nginx
ADD index.tar /var/www/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

LLEIDA TAR

ENV

```
FROM ubuntu:latest #imagen a utilizar
RUN  ls # comando de linux a ejecutar en el build
COPY readme.md /etc/opt
ENV DOCKER=FIRSTENV
```

```
docker build -t ubuntu .
```

```
docker run  ubuntu env
```

```
Esteban (-zsh)
esteban ~/Projects/Personal/DockerCurs/nginx % docker build . -t ubuntu
[+] Building 0.1s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 156B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/2] FROM docker.io/library/ubuntu:latest
=> CACHED [2/2] RUN ls # comando de linux a ejecutar en el build
=> exporting to image
=> => exporting layers
=> => writing image sha256:eaab978bf89f25319db99572a9afc27e2ec8fae89c8e579caabd98dde9a068c1
=> => naming to docker.io/library/ubuntu
esteban ~/Projects/Personal/DockerCurs/nginx % docker run ubuntu env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=7c96b2f98786
DOCKER=FIRSTENV
HOME=/root
esteban ~/Projects/Personal/DockerCurs/nginx %
```

WORKDIR

La instrucción **WORKDIR** establece el directorio de trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT, COPY y ADD que le sigue en el Dockerfile. Si **WORKDIR** no existe, se creará incluso si no se usa en ninguna instrucción posterior de Dockerfile.

La instrucción **WORKDIR** se puede usar varias veces en un Dockerfile. Si se proporciona una ruta relativa, será relativa a la ruta de la instrucción **WORKDIR** anterior. Por ejemplo:

```
FROM ubuntu:latest
RUN apt update
RUN apt install -y nginx
WORKDIR /var/www/html
ADD index.tar .
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

EXPOSE

La instrucción **EXPOSE** informa a Docker que el contenedor escucha en los puertos de red especificados en tiempo de ejecución. Puede especificar si el puerto escucha en TCP o UDP, y el valor predeterminado es TCP si no se especifica el protocolo.

```
FROM ubuntu:latest
RUN apt update
RUN apt install -y nginx
WORKDIR /var/www/html
```

```
ADD index.tar .
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
docker run -p 81:80 ubuntu
```

← → ↻ ⓘ localhost:81

LLEIDA TAR

LABEL

```
FROM ubuntu:latest
LABEL author="Esteban Martin Gimenez"
LABEL version="0.0.1"
RUN apt update
RUN apt install -y nginx
WORKDIR /var/www/html
ADD index.tar .
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
esteban ~/Projects/Personal/DockerCurs/nginx % docker image inspect --format='{{json .Config.Labels}}' ubuntu
{"author":"Esteban Martin Gimenez","org.opencontainers.image.ref.name":"ubuntu","org.opencontainers.image.version":"22.04","version":"0.0.1"}
esteban ~/Projects/Personal/DockerCurs/nginx %
```

```
esteban ~/Projects/Personal/DockerCurs/nginx % docker image inspect --format='{{json .Config.Labels}}' ubuntu | jq
{
  "author": "Esteban Martin Gimenez",
  "org.opencontainers.image.ref.name": "ubuntu",
  "org.opencontainers.image.version": "22.04",
  "version": "0.0.1"
}
esteban ~/Projects/Personal/DockerCurs/nginx % docker image inspect --format='{{json .Config.Labels}}' ubuntu | jq .author
"Esteban Martin Gimenez"
esteban ~/Projects/Personal/DockerCurs/nginx %
```

USER

La instrucción **USER** establece el nombre de usuario (o UID) y, opcionalmente, el grupo de usuarios (o GID) para usar como usuario y grupo predeterminados durante el resto de la etapa actual. El usuario especificado se utiliza para las instrucciones RUN y, en tiempo de ejecución, ejecuta los comandos ENTRYPOINT y CMD relevantes.

```
FROM nginx:stable-alpine-slim
LABEL author="Esteban Martin Gimenez"
LABEL version="0.0.1"
WORKDIR /var/www/html
ADD index.tar .
USER nginx
```

```

esteban ~/Projects/Personal/DockerCurs/nginx % docker inspect ce7 | jq '.[].Config'
{
  "Hostname": "ce742ff63c27",
  "Domainname": "",
  "User": "nginx",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "ExposedPorts": {
    "80/tcp": {}
  },
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "NGINX_VERSION=1.24.0",
    "PKG_RELEASE=1"
  ],
  "Cmd": [
    "nginx",
    "-g",
    "daemon off;"
  ],
  "Image": "nginx2",
  "Volumes": null,
  "WorkingDir": "/var/www/html",
  "Entrypoint": [
    "/docker-entrypoint.sh"
  ],
  "OnBuild": null,
  "Labels": {
    "author": "Esteban Martin Gimenez",
    "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>",
    "version": "0.0.1"
  },
  "StopSignal": "SIGQUIT"
}
esteban ~/Projects/Personal/DockerCurs/nginx % docker inspect ce7 | jq '.[].Config.User'
"nginx"
esteban ~/Projects/Personal/DockerCurs/nginx %

```

VOLUME

La instrucción **VOLUME** crea un punto de montaje con el nombre especificado y lo marca como que contiene volúmenes montados externamente desde un host nativo u otros contenedores. El valor puede ser una matriz JSON, VOLUMEN ["/var/log/"] o una cadena sin formato con varios argumentos, como VOLUMEN /var/log o VOLUMEN /var/log /var/db.

```

FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol

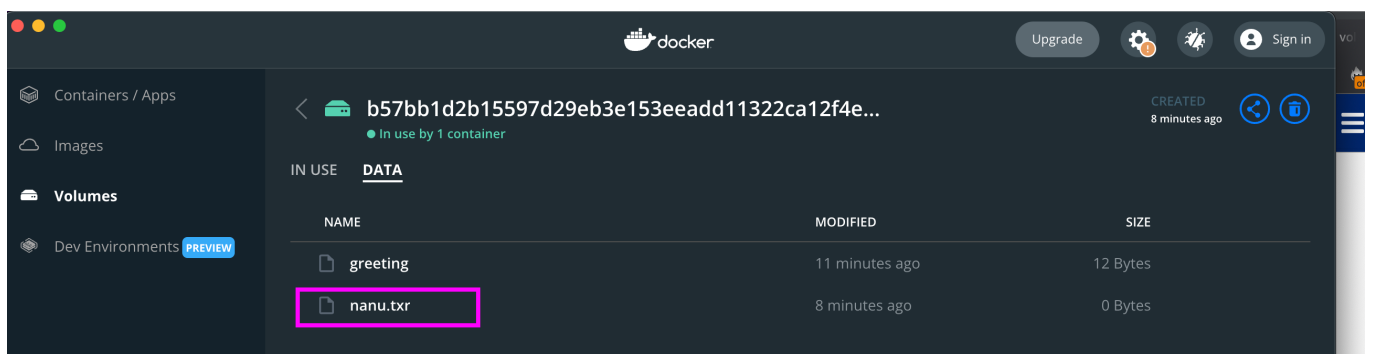
```

```
Esteban (docker)
root@c387a4f5aca5:/# ls
bin boot dev etc home lib media mnt myvol opt proc root run sbin srv sys tmp usr var
```

Cuando se construye la imagen, también se crea el volumen en Docker.

```
Esteban (-zsh)
esteban ~/Projects/Personal/DockerCurs/nginx % docker volume ls
DRIVER      VOLUME NAME
esteban ~/Projects/Personal/DockerCurs/nginx % docker build -t volume .
[+] Building 0.6s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 136B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/3] FROM docker.io/library/ubuntu@sha256:6120be6a2b7ce665d0cbddc3ce6eae60fe94637c6a66985312d1f02f63cc0bcd
=> CACHED [2/3] RUN mkdir /myvol
=> CACHED [3/3] RUN echo "hello world" > /myvol/greeting
=> exporting to image
=> => exporting layers
=> => writing image sha256:34c0275041fabe1a72a1890b0d76e6f2481e44577435690cad6583c90b6beda6
=> => naming to docker.io/library/volume
esteban ~/Projects/Personal/DockerCurs/nginx % docker run -it volume
root@4cdb195a322d:/# exit
exit
esteban ~/Projects/Personal/DockerCurs/nginx % docker volume ls
DRIVER      VOLUME NAME
local       efd0b0276dd64a7cf374778409ab00a6fd26a5dbedf5425606fe7cf1c1e9ade3
esteban ~/Projects/Personal/DockerCurs/nginx %
```

```
docker run -it volume
# cd myvol
#touch readme.md
#exit
```




```
Esteban (-zsh)
esteban ~ % docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
b57c792fe6dc   volume    "/bin/bash"            9 minutes ago Up 7 minutes   naughty_goldstine
esteban ~ % docker stop b57c792fe6dc
b57c792fe6dc
esteban ~ % docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
esteban ~ % docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
b57c792fe6dc   volume    "/bin/bash"            9 minutes ago Exited (137) 20 seconds ago   naughty_goldstine
esteban ~ % docker start b57c792fe6dc
b57c792fe6dc
esteban ~ % docker exec -it b57c792fe6dc bash
root@b57c792fe6dc:/# ls
bin  boot  dev  etc  home  lib  media  mnt  myvol  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@b57c792fe6dc:/# cd myvol/
root@b57c792fe6dc:/myvol# ls
greeting  nanu.txt
root@b57c792fe6dc:/myvol# exit
exit
esteban ~ %
```

STOPSIGNAL

La instrucción **STOPSIGNAL** establece la señal de llamada al sistema que se enviará al contenedor para salir. Esta señal puede ser un nombre de señal con el formato **SIG NAME**, por ejemplo SIGKILL, o un número sin signo que coincida con una posición en la tabla de llamadas al sistema del kernel, por ejemplo 9. El valor predeterminado es SIGTERM si no está definido.

La señal de parada predeterminada de la imagen se puede anular por contenedor, usando el indicador `--stop-signal` en `docker run` y `docker create`.

CMD

En Docker, la instrucción **CMD** se usa para especificar el comando predeterminado que debe ejecutarse cuando se inicia un contenedor Docker. Es similar a la instrucción **ENTRYPOINT**, pero también le permite especificar los argumentos predeterminados.

Shell (interprete de comandos):

```
CMD command arg1 arg2
```

```
CMD echo "Hello, world!"
```

*Según el shell, los comandos se ejecutarán como procesos secundarios del shell, lo que tiene algunas consecuencias potencialmente negativas.

Exec:

Los comandos se escriben entre corchetes [] y se ejecutan directamente, no a través de un shell.

```
CMD ["executable", "arg1", "arg2"]
```

```
CMD ["echo", "Hello, world!"]
```

```
FROM ubuntu:latest  
CMD echo "Hello, Docker!"
```

Cuando crea una imagen con una instrucción **CMD**, define el comando predeterminado que se ejecutará cuando inicie un contenedor desde esa imagen. Sin embargo, puede anular la instrucción **CMD** especificando un comando diferente al ejecutar el contenedor:

```
docker run command.
```

Ejemplo:
Dockerfile

```
FROM python:3.9  
WORKDIR /app  
COPY script.py .  
COPY script2.py .  
CMD ["python", "script2.py"]
```

script.py

```
print("¡Hola desde el script de Python!")
```

```
docker build -t python .
```

```
docker run python2 python script2.py juan
```

ENTRYPOINT

En Docker, la instrucción **ENTRYPOINT** se utiliza para configurar el comando base o el programa que se ejecutará dentro de un contenedor cuando se inicie. A diferencia de la instrucción `CMD`, el `ENTRYPOINT` no se puede sobrescribir fácilmente al ejecutar un contenedor con un comando diferente.

La instrucción `ENTRYPOINT` se puede escribir en dos formatos diferentes:

Shell:

```
ENTRYPOINT command param1 param2
```

```
ENTRYPOINT echo "Hola, mundo!"
```

Exec:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

```
ENTRYPOINT ["echo", "Hola, mundo!"]
```

Pasando argumentos a nuestros script:

```
FROM python:3.9
WORKDIR /app
COPY script.py .
ENTRYPOINT ["python", "script.py"]
```

```
import sys

if len(sys.argv) < 2:
    print("Debe proporcionar al menos un argumento.")
    sys.exit(1)

arg = sys.argv[1]

print(f"¡Hola desde el script de Python! Hola: {arg}")
```

```
docker run python argumento
```

Shell Vs Exec:

```
FROM ubuntu
WORKDIR /app
```

ENTRYPOINT top -b

```
esteban ~/Projects/Personal/Clase1/Python % docker build -t python .
[+] Building 1.2s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 36B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/2] FROM docker.io/library/ubuntu@sha256:6120be6a2b7ce665d0cbddc3ce6ae60fe94637c6a66985312d1f02f63cc0bcd
=> CACHED [2/2] WORKDIR /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:e754fb49a1c6acacce5226a48f054fe02d983a397cfdfe769b13e5358fae97a3
=> => naming to docker.io/library/python

esteban ~/Projects/Personal/Clase1/Python % docker run python
top - 13:32:27 up 4:48, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1988.7 total, 92.6 free, 290.2 used, 1605.9 buff/cache
MiB Swap: 1024.0 total, 1017.4 free, 6.6 used. 1281.5 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   6580   2404   2084 R   0.0   0.1   0:00.01 top

top - 13:32:31 up 4:49, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.3 sy, 0.0 ni, 99.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1988.7 total, 92.4 free, 290.2 used, 1606.1 buff/cache
MiB Swap: 1024.0 total, 1017.4 free, 6.6 used. 1281.5 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   6580   2404   2084 R   0.0   0.1   0:00.01 top
```

FROM ubuntu

WORKDIR /app

ENTRYPOINT ["top", "-b"]

```
Esteban (docker)
esteban ~/Projects/Personal/Clase1/Python % docker build -t python .
[+] Building 0.6s (6/6) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 84B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 0.5s
=> [1/2] FROM docker.io/library/ubuntu@sha256:6120be6a2b7ce665d0cbddc3ce6ae60fe94637c6a66985312d1f02f63cc0bcd 0.0s
=> CACHED [2/2] WORKDIR /app 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:0b14885d765bc2c6ed22ba99050103e30c7788c5a392aec49bcd7df474b5199d 0.0s
=> => naming to docker.io/library/python 0.0s

esteban ~/Projects/Personal/Clase1/Python % docker run python
top - 13:33:13 up 4:49, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 1.7 sy, 0.0 ni, 96.7 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
MiB Mem : 1988.7 total, 93.1 free, 289.4 used, 1606.2 buff/cache
MiB Swap: 1024.0 total, 1017.4 free, 6.6 used. 1282.4 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   2308    836    752 S   0.0   0.0   0:00.02 sh
    7 root        20   0   6580   2420   2100 R   0.0   0.1   0:00.00 top

top - 13:33:16 up 4:49, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.3 sy, 0.0 ni, 99.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1988.7 total, 93.1 free, 289.1 used, 1606.5 buff/cache
MiB Swap: 1024.0 total, 1017.4 free, 6.6 used. 1282.6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   2308    836    752 S   0.0   0.0   0:00.02 sh
    7 root        20   0   6580   2420   2100 R   0.0   0.1   0:00.00 top
```

CMD vs ENTRYPOINT

CMD: establece parámetros predeterminados que se pueden anular desde la interfaz de línea de comandos (CLI) de Docker mientras se ejecuta un contenedor docker.

ENTRYPOINT: establece parámetros predeterminados que no se pueden anular al ejecutar contenedores Docker con parámetros CLI.

.dockerignore file

En Docker, el archivo ".dockerignore" se utiliza para especificar qué archivos y directorios deben ser excluidos del contexto de construcción de la imagen Docker. Es similar al archivo ".gitignore" utilizado en Git.

El archivo ".dockerignore" es opcional, pero puede resultar útil para evitar la inclusión accidental de archivos innecesarios en la imagen Docker, reduciendo así su tamaño y mejorando el rendimiento de la construcción.

Aquí hay un ejemplo básico de cómo se puede utilizar un archivo ".dockerignore":

```
# Comentarios comienzan con el símbolo de numeral (#)

# Ignorar todos los archivos con la extensión .txt
*.txt

# Ignorar el directorio "logs" y todo su contenido
logs/

# Ignorar el archivo llamado "secreto.txt"
secreto.txt

# Ignorar todos los archivos y subdirectorios dentro de "temp" pero no el
directorio en sí
temp/
```

En este ejemplo, los patrones en el archivo ".dockerignore" indican qué archivos o directorios deben ser ignorados durante el proceso de construcción de la imagen Docker. Los patrones pueden incluir caracteres comodín, como "*" para representar cualquier conjunto de caracteres y "/" para denotar un directorio.

Para utilizar el archivo ".dockerignore", colócalo en el mismo directorio que el archivo Dockerfile y asegúrate de que se encuentre en el contexto de construcción antes de ejecutar el comando "docker build". De esta manera, Docker respetará las exclusiones especificadas en el archivo ".dockerignore" y no incluirá los archivos o directorios ignorados en la imagen resultante.

Docker MultiStage

<https://docs.docker.com/build/building/multi-stage/>

<https://github.com/TechPrimers/multi-stage-example/tree/master>

```
FROM openjdk:8-jdk as builder
RUN mkdir -p /app/source
COPY . /app/source
WORKDIR /app/source
RUN ./mvnw clean package
RUN cp target/*.jar /app/app.jar
EXPOSE 8080
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom", "-jar",
"/app/app.jar"]
```

```
FROM openjdk:8-jdk as builder
RUN mkdir -p /app/source
COPY . /app/source
WORKDIR /app/source
RUN ./mvnw clean package

FROM openjdk:8-jdk-alpine
COPY --from=builder /app/source/target/*.jar /app/app.jar
EXPOSE 8080
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom", "-jar",
"/app/app.jar"]
```

```
esteban ~/Projects/Personal/Clase1/Multistage/multi-stage-example % docker image ls
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---------------------|---------------|--------------|--------------------|--------|
| java_big | latest | 471cfe8fb15c | About a minute ago | 633MB |
| java_small | latest | 25aafb73c20d | 31 minutes ago | 122MB |
| java | latest | ebc869cfd5b5 | 48 minutes ago | 216MB |
| obamarama/lleida | 0.0.1 | 0471804c0bfa | 4 hours ago | 284MB |
| nanusefue/lleida | 0.0.2 | 94b8ad8d1b8b | 4 hours ago | 284MB |
| aniol0012/lleida | 0.0.1 | c25b791d40dc | 5 hours ago | 284MB |
| nanusefue/lleida | 0.0.3 | c25b791d40dc | 5 hours ago | 284MB |
| la33iscoming/jamsha | 0.0.1 | 480779988f77 | 5 hours ago | 252MB |
| nanusefue/lleida | 0.0.1 | fefb3417389e | 5 hours ago | 284MB |
| wordpress | latest | 89df9eb0296d | 11 days ago | 615MB |
| arm64v8/mysql | latest | 1732fe3340d5 | 2 weeks ago | 587MB |
| registry | 2 | daace2c8ce4c | 2 weeks ago | 23.8MB |
| nginx | stable-alpine | ba1fca8fb480 | 2 weeks ago | 40.7MB |

```
docker build --target builder -t java_small2 .
```

```
esteban ~/Projects/Personal/Clase1/Multistage/multi-stage-example % docker build --target builder -t java_small2 .
[+] Building 39.7s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 345B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:8-jdk
=> [builder 1/5] FROM docker.io/library/openjdk:8-jdk@sha256:86e863cc57215cfb181bd319736d0baf625fe8f150577f9eb58bd937f5452cb8
=> [internal] load build context
=> => transferring context: 164.45kB
=> CACHED [builder 2/5] RUN mkdir -p /app/source
=> [builder 3/5] COPY . /app/source
=> [builder 4/5] WORKDIR /app/source
=> [builder 5/5] RUN ./mvnw clean package
=> exporting to image
=> => exporting layers
=> => writing image sha256:e0b19779cfe570279f2b1ec4856c01fed08a190db8d49cca115d32888f67931c
=> => naming to docker.io/library/java_small2
esteban ~/Projects/Personal/Clase1/Multistage/multi-stage-example %
```

Ejercicios:

1 - Crea una imagen Docker base para ser reutilizada, es necesario que crees un Dockerfile con las siguientes especificaciones:

Sistema Operativo Base: Debian, Ubuntu (A tu elección)

Herramientas a instalar:

- Nginx
- PHP 8.3

...

<?php

```
phpinfo();
```

```
?>
```

creando un index.php con la función de phpinfo.

2 – Ahora intenta te piden que el tamaño de tu imagen creada pese lo menos posible

3 – Sube tu imagen a un repositorio publico y añade el tag slim y stable para cada una de las imágenes que generaste.

2 – Crea una imagen Docker base para ser reutilizada, es necesario que crees un Dockerfile con las siguientes especificaciones:

Aplicación desarrollada en java :

<https://github.com/jenkins-docs/simple-java-maven-app>

Herramientas a instalar:

- maven
- openjdk

2 – Ahora intenta te piden que el tamaño de tu imagen creada pese lo menos posible

3 – Sube tu imagen a un repositorio publico y añade el tag slim y stable para cada una de las imágenes que generaste.