

# Docker Images y Dockerfile

Fase	Semana	Día	Lección
6 - Bases de Datos	3	2	4-7

# Docker Images

Una imagen de docker es plantilla de solo lectura que define al contenedor.


La imagen contiene el código necesario por nuestro contenedor (como la definición para cualquier biblioteca o dependencias).

Las imágenes de docker se crean a partir de un archivo llamado **Dockerfile**

Las imágenes se pueden crear local u obtenerlas de un repositorio de imágenes como Docker Hub  
<https://hub.docker.com/>

# Búsqueda de Imágenes Docker Hub

Sitio: <https://hub.docker.com/search?q=>

 Search Docker Hub

Explore Pricing Sign In [Sign up](#)

### Filters




1 - 25 of 10,000 available results.

Suggested ▾

#### Products


- ☐ Images
- ☐ Extensions
- ☐ Plugins




#### Trusted Content

- ☐  Docker Official Image ⓘ
- ☐  Verified Publisher ⓘ
- ☐  Sponsored OSS ⓘ

#### Operating Systems

- ☐ Linux




**alpine**  Docker Official Image ·  1B+ ·  10K+


Updated 16 days ago


A minimal Docker image based on Alpine Linux with a complete package index and only 5 ...




Linux riscv64 x86-64 ARM ARM 64 386 PowerPC 64 LE IBM Z

Pulls: 11,207,022  
Last week



[Learn more](#) 




**nginx**  Docker Official Image ·  1B+ ·  10K+


Updated 10 days ago

Official build of Nginx.

Linux ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z x86-64

Pulls: 14,336,719  
Last week



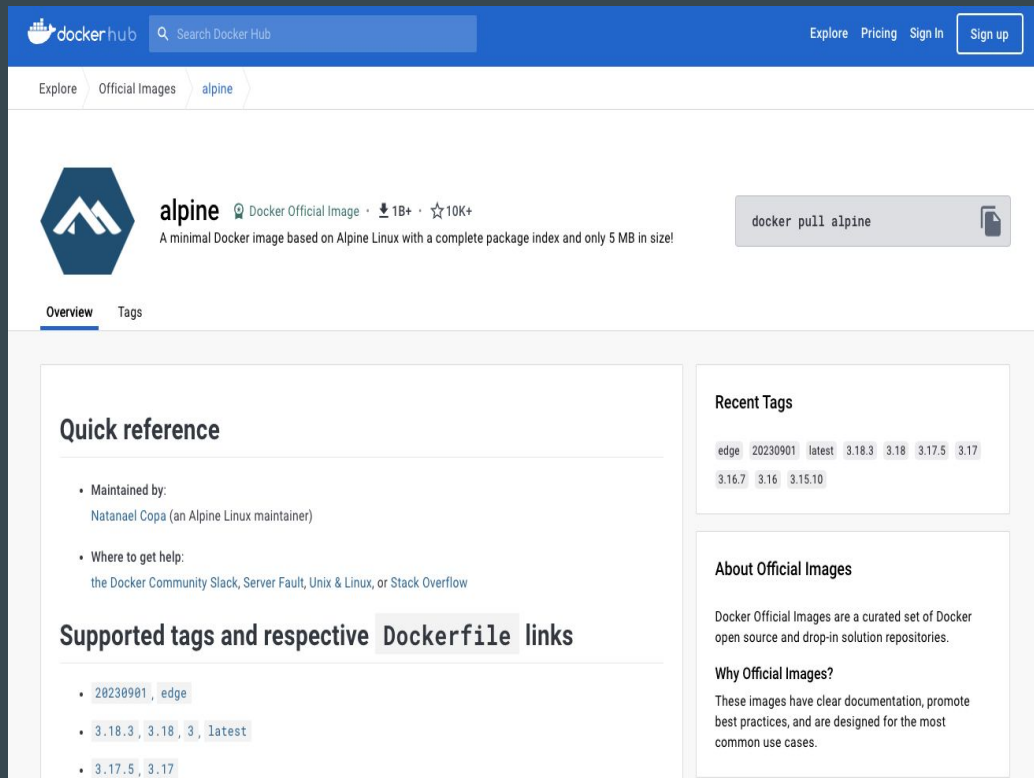
[Learn more](#) 

# Obtener Imagen desde Docker Hub

Al seleccionar una de los resultados de la página anterior nos llevará a la página con la información de la imagen seleccionada. Acá se verán datos como las versiones, como utilizarla, ...

Una vez sepamos cuál imagen queramos utilizar/descargar en nuestro sistema lo único que tenemos que hacer es correr en la terminal el comando que nos indica en la esquina superior derecha


`docker pull alpine`




The screenshot shows the Docker Hub interface for the 'alpine' image. The header includes the Docker Hub logo, a search bar, and links for Explore, Pricing, Sign In, and Sign up. The main content area features the 'alpine' logo, a description of the image as a minimal Docker image based on Alpine Linux, and a 'docker pull alpine' button. Below this, there are tabs for Overview and Tags. The Overview tab is active, showing a 'Quick reference' section with links to the maintainer (Natanael Copa) and where to get help (Docker Community Slack, Server Fault, etc.). A 'Supported tags and respective Dockerfile links' section lists various tags like '20230901', 'edge', '3.18.3', '3.18', '3', 'latest', '3.17.5', and '3.17'. On the right side, there are sections for 'Recent Tags' and 'About Official Images'.

dockerhub Search Docker Hub Explore Pricing Sign In Sign up

Explore Official Images alpine

 **alpine** Docker Official Image · 1B+ · 10K+  
A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

docker pull alpine 

Overview Tags

**Quick reference**

- Maintained by:  
Natanael Copa (an Alpine Linux maintainer)
- Where to get help:  
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

**Supported tags and respective Dockerfile links**

- 20230901, edge
- 3.18.3, 3.18, 3, latest
- 3.17.5, 3.17

**Recent Tags**

edge 20230901 latest 3.18.3 3.18 3.17.5 3.17 3.16.7 3.16 3.15.10

**About Official Images**

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

**Why Official Images?**

These images have clear documentation, promote best practices, and are designed for the most common use cases.

# Interactuar con Imagen de Docker Hub

Una vez se descarga la imagen de Docker Hub podemos iniciar un contenedor e interactuar dentro del mismo

```
~|⇒ docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
7264a8db6415: Already exists
Digest: sha256:7144f7bab3d4c2648d7e59409f15ec52a18006a128c733fcff20d3a4a54ba44a
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

```
~|⇒ docker run -it alpine
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root
/ # echo "Hello World!"
Hello World!
/ # █
```

# Dockerfile

# Qué es un Dockerfile?

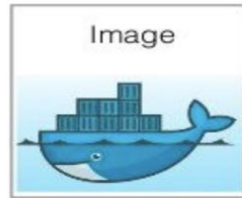
Dockerfile es un archivo de texto que contiene las instrucciones necesarias para crear una nueva imagen del contenedor. Estas instrucciones incluyen la identificación de una imagen existente que se usará como base, los comandos que se ejecutarán durante el proceso de creación de la imagen y un comando que se ejecutará cuando se implementen instancias nuevas de la imagen del contenedor.

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

```
FROM ubuntu:16.04
MAINTAINER John Doe <john.doe@example.com>
RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip3 install Flask
EXPOSE 5000
CMD ["python3", "app.py"]
```

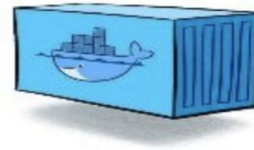
Dockerfile

build



Docker Image

run



Docker Container

# Entendiendo el Dockerfile: Sintaxis

El flujo típico de un Dockerfile es:

- Seleccionar la imagen base
- Información general de la imagen
- Descargar e instalar dependencias
- Comandos a ejecutar al iniciar el contenedor

Una imagen consiste de capas de lectura donde cada capa es representada por cada una de las instrucciones del Dockerfile.

Las capas se crean una sobre otra y cada una se compone de la diferencia de los cambios realizados entre la capa anterior y nuevos cambios.

```
1  # Imagen base
2  FROM node:alpine
3
4  # Metadatos que indican el encargado de mantener la imagen.
5  LABEL maintainer="william@test.com"
6
7  # Carpeta de trabajo
8  WORKDIR /usr/app
9
10 # Copiado de archivos
11 COPY ./ ./
12
13 # Instalación de dependencias
14 RUN npm install
15
16 # Comando a ejecutar defecto dentro del contenedor
17 CMD ["npm", "start"]
```



# Entendiendo el Dockerfile: Instrucciones

**FROM:** Define la imagen que se utilizará como base para la nuestra.

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

**RUN:** Ejecuta cualquier comando en una nueva capa sobre la imagen actual. El resultado se utilizará en el siguiente paso/instrucción del Dockerfile

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

**CMD:** Su propósito principal es proporcionar valores predeterminados para un contenedor en ejecución. Sólo puede haber una instrucción CMD en un Dockerfile

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

**LABEL:** Agrega metadatos a la imagen que se va a generar/crear/construir

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

# Entendiendo el Dockerfile: Instrucciones

**EXPOSE:** Informa a Docker que el contenedor estará escuchando sobre un puerto específico en la red

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

**ENV:** Define variables de ambiente que se pueden utilizar dentro del contenedor o para las siguientes instrucciones.

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

**ADD:** Copia archivos (locales o remotos por medio de URLs), y directorios, y los agrega al sistema de archivos del contexto y ruta que se indique dentro del contenedor

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

**COPY:** Copia archivos o directorios desde la ruta donde se encuentra el Dockerfile a la ruta que se indique dentro del contenedor

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

# Entendiendo el Dockerfile: Instrucciones

**ENTRYPOINT:** Nos permite configurar un contenedor que correrá como ejecutable

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

**VOLUME:** Se utiliza para indicar el directorio que será utilizado como directorio que contiene volúmenes montados externamente desde el host nativo y otros contenedores

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

**WORKDIR:** Indica el directorio dentro del contenedor sobre el que se ejecutarán las siguientes instrucciones/pasos del Dockerfile

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

**ARG:** Define variables que podemos pasar al momento de construir nuestra imagen

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

# Ejemplo de Dockerfile



```
# Imagen base
FROM node:alpine

# Metadatos que indican el encargado de mantener la imagen.
LABEL maintainer="william@test.com"

# Carpeta de trabajo
WORKDIR /usr/app

# Copiado de archivos
COPY ./ ./

# Instalación de dependencias
RUN npm install

# Comando a ejecutar por defecto dentro del contenedor
CMD ["npm", "start"]
```

# Como crear/generar una imagen desde un Dockerfile

Una vez se tiene el Dockerfile se debe ejecutar el comando `docker build` para generar / construir la imagen

Este comando tiene varias opciones, entre estas se encuentra

`--file`: se utiliza para indicar en donde se encuentra el archivo con las instrucciones para generar la imagen.  
Por defecto busca por el Dockerfile en la ruta actual

`--tag`: se usa para indicar dar un nombre y tag a nuestra imagen

Docs:

<https://learn.microsoft.com/es-es/virtualization/windowscontainers/manage-docker/manage-windows-dockerfile#docker-build>

<https://docs.docker.com/engine/reference/commandline/build/>

# Ejemplo Creación/Construcción imagen personalizada

```
~|➔ cat Dockerfile
FROM alpine
CMD echo 'Hello world'
~|➔ docker build -t "my-first-image:v1" .
[+] Building 0.1s (5/5) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile             0.0s
=> => transferring dockerfile: 72B                               0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load metadata for docker.io/library/alpine:latest 0.0s
=> CACHED [1/1] FROM docker.io/library/alpine                  0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:05c9da00b60c0d047ee340166590f01756ceb242e1745f8cf22789092b360cbb 0.0s
=> => naming to docker.io/library/my-first-image:v1            0.0s
```

## What's Next?

View a summary of image vulnerabilities and recommendations → `docker scout quickview`

```
~|➔ docker run my-first-image:v1
```

```
Hello world
```

# Capas de una imagen de Docker

## Capa Base (Base Layer):

Esta es la primera capa de una imagen de Docker. Generalmente, se basa en una imagen existente del registro de Docker, como una imagen oficial de una distribución de Linux, como Ubuntu o Alpine. Esta capa contiene el sistema operativo base y las herramientas esenciales.

## Capas Intermedias:

Cada instrucción en un Dockerfile crea una nueva capa. Por ejemplo, cuando se ejecuta el comando RUN para instalar paquetes o configurar software, se crea una capa separada. Estas capas son de solo lectura y se pueden compartir entre múltiples imágenes.

## Capa Superior (Top Layer):

La capa superior es la capa final que se agrega a la imagen. Contiene los cambios específicos realizados en el contenedor, como la adición de archivos de aplicación y configuraciones. Esta capa es de lectura/escritura, lo que permite que los contenedores basados en la imagen realicen cambios temporales.

# Ejemplos de capas

```
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM ubuntu:latest
---> d6c8736f0e9f
Step 2/3 : WORKDIR /root
---> Running in 3068d7075458
Removing intermediate container 3068d7075458
---> d80343d04808
Step 3/3 : ENTRYPOINT ["/run.sh"]
---> Running in ae6480ddac50
Removing intermediate container 44cf42bcd050
---> 8c2b84201d65
Successfully built 8c2b84201d65
Successfully tagged myimage:latest
```

Se puede observar que cada step genera un nuevo hash que es el identificador de la nueva capa. Este se genera a partir del hash anterior junto a los nuevos cambios.



# Caché al generar imágenes

Docker utiliza una caché para optimizar el proceso de construcción de imágenes. Cuando se ejecuta una instrucción en un Dockerfile, Docker verifica si esa instrucción y sus dependencias ya se han ejecutado antes.

La forma en que se sabe si se ha ejecutado antes es calculado el hash del step y comparando con el generado la vez pasada, si este no ha cambiado, se utiliza la misma capa que se había generado anteriormente y se encuentra en caché.

Esto hará que el build sea más rápido la próxima vez

```
Sending build context to Docker daemon 1.584kB
Step 1/3 : FROM ubuntu:latest
----> Using cache
----> d6c8736f0e9f
Step 2/3 : WORKDIR /root
----> Using cache
----> d80343d04808
Step 3/3 : ENTRYPOINT ["/run2.sh"]
----> Running in ae6480ddac70
Removing intermediate container 44cf42bcd070
----> 8c2b84201d70
Successfully built 8c2b84201d70
Successfully tagged myimage:latest
```

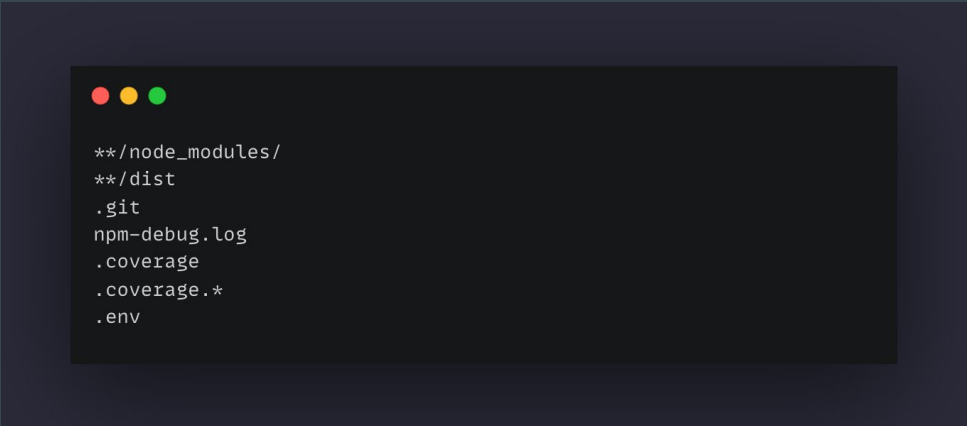
# Buenas prácticas en un Dockerfile

# Uso del archivo .dockerignore

El archivo `.dockerignore` es útil para excluir archivos o directorios grandes del build de la imagen de docker. Este archivo se debe de crear en el root del proyecto

Esto ayuda a disminuir el tamaño de la imagen de docker que se va a construir

También ayuda en términos de seguridad ya que nos facilita el ignorar/excluir archivos con datos confidenciales como lo son los archivos `.env`

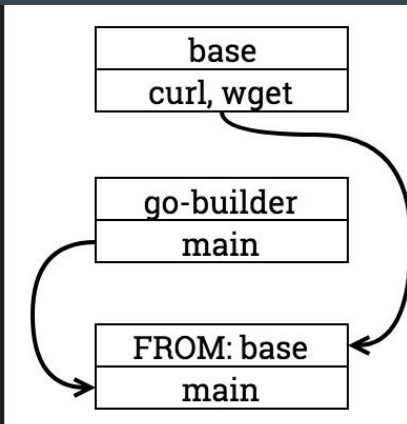
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the content of a .dockerignore file.

```
*/node_modules/  
*/dist  
.git  
npm-debug.log  
.coverage  
.coverage.*  
.env
```

# Imágenes de múltiples etapas (multi-stage build)

Un Dockerfile de múltiples etapas (multi-stage Dockerfile) es una técnica avanzada que se utiliza para construir imágenes de Docker de manera más eficiente, especialmente cuando se trabaja con aplicaciones complejas que requieren la compilación de código fuente, pero no se desea incluir todas las herramientas de compilación en la imagen final.

```
1 FROM alpine AS base
2 RUN apk add --no-cache curl wget
3
4 FROM golang:1.9.2 AS go-builder
5 WORKDIR /go
6 COPY *.go /go/
7 RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o main .
8
9 FROM base
10 COPY --from=go-builder /go/main /main
11 CMD ["/main"]
```



# Ejemplo de Dockerfile multi-stage para aplicación Node



```
# Etapa 1: Construcción de la aplicación
FROM node:14 as builder

WORKDIR /app

COPY package*.json ./
COPY tsconfig.json ./

RUN npm install
COPY src/ ./src/
RUN npm run build

# Etapa 2: Creación de la imagen final
FROM node:14
WORKDIR /app

# Copia solo los archivos necesarios de la etapa de construcción
COPY --from=builder /app/package*.json ./
COPY --from=builder /app/dist/ ./dist/

RUN npm install --only=production
CMD ["npm", "start"]
```

# Ejercicio #1 - Crear imagen para aplicación Node existente

Descargue / Clone el siguiente proyecto:

<https://github.com/docker/getting-started-app/tree/main>

Agregue un archivo **Dockerfile** en el root del directorio

Usando un editor de texto agregue las instrucciones necesarias en el Dockerfile para hacer correr la aplicación.

Notas:

- Es una aplicación que requiere de node
- Se deben instalar dependencias en modo producción con yarn
  - yarn install --production
- El archivo de inicio es el src/index.js
- La aplicación debe exponerse en el puerto 3001

# Ejercicio #1 - Crear imagen para aplicación Node existente

Una vez se tenga listo el Dockerfile, crear una imagen bajo el nombre de **getting-started**

Revisar que la imagen se haya creado en nuestro sistema (**docker images**)

Correr un contenedor usando nuestra nueva imagen

```
docker run -dp 127.0.0.1:3001:3001 getting-started
```

Revisar que el contenedor esté corriendo (**docker ps**)

Abrir en el navegador la siguiente dirección: <http://localhost:3001/>

Interactuar con el sitio y revisar que funcione bien

## Ejercicio #2 - Subir imagen a Docker Hub

Objetivo: Tomar la imagen creada en el ejercicio anterior y subirla a Docker Hub

Paso 1: Crear cuenta en <https://hub.docker.com/>

Paso 3: Crear un repositorio público (puede usar el nombre de getting-started)

Paso 4: Seleccionar **Create**

Paso 5: Loguearse desde la terminal: `docker login -u your_docker_hub_username`

Paso 6: Crear tag para nuestra imagen:

```
docker tag local_image_name_or_id your_docker_hub_username/image_name  
docker tag getting-started your_docker_hub_username/getting-started
```



## Ejercicio #2 - Subir imagen a Docker Hub

Paso 7: Hacer push al repo en Docker Hub

```
docker push your_docker_hub_username/getting-started
```

Paso 8: Correr contenedor usando imagen de Docker Hub

```
docker run -dp 0.0.0.0:3001:3001 your_docker_hub_username/getting-started
```

Paso 9: Abrir en el navegador la siguiente dirección: <http://localhost:3000/>

Paso 10: Interactuar con el sitio y revisar que funcione bien