# What is Docker?

“Build once, run anywhere”

After we install Docker Desktop (Docker Daemon).

# Beginning with docker

## Check if docker is running

we are going to execute the next command:

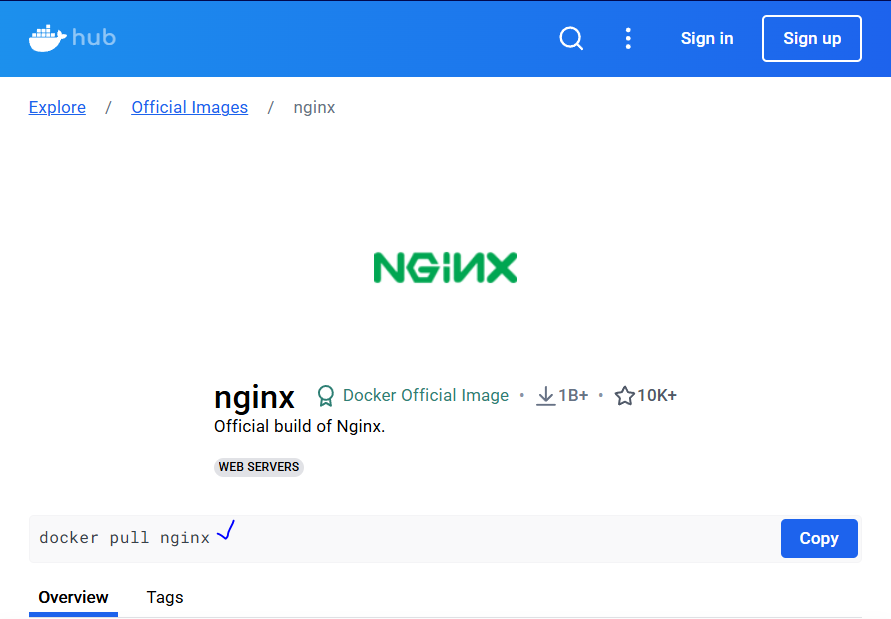
Texto

Descripción generada automáticamente con confianza media

If Docker is running, it will display detailed system information.

Now, we can create a container using the downloaded image, for example using the next command:

## Download an image from docker hub



Execute the command will download the image.

## Create a container from that image

Texto

Descripción generada automáticamente

#### Docker run -p 8080:80 -dt nginx

**Explanation:**

1. **docker run**
   * This command is used to create and start a new container from a specified image.
2. **-p 8080:80**
   * This flag maps the container's port **80** (where Nginx runs by default) to your host machine's port **8080**.
   * This means you can access Nginx by visiting http://localhost:8080 on your web browser.
3. **-d**
   * This flag runs the container in **detached mode**, meaning it runs in the background instead of keeping the terminal occupied.
4. **-t**
   * Allocates a **pseudo-TTY** (terminal) inside the container. It's useful for interactive processes but not always necessary for Nginx.
5. **nginx**
   * This is the Docker image name. It tells Docker to pull and run an **Nginx** web server container.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**What Happens When You Run This?**

* Docker checks if the **nginx** image is available locally; if not, it pulls it from Docker Hub.
* A new container is created and started in **detached mode**.
* Nginx inside the container listens on **port 80**, which is mapped to **port 8080** on your local machine.
* You can access the running Nginx server via http://localhost:8080.

How can we get the List of the images available in our local:

#### Docker images

Texto

Descripción generada automáticamente

## “Docker container” commands

To deal with the available containers we had the next commands:

#### Docker container

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Command | Description |
| attach | Attach local standard input, output, and error streams to a running container |
| commit | Create a new image from a container's changes |
| cp | Copy files/folders between a container and the local filesystem |
| create | Create a new container |
| diff | Inspect changes to files or directories on a container's filesystem |
| exec | Execute a command in a running container |
| export | Export a container's filesystem as a tar archive |
| inspect | Display detailed information on one or more containers |
| kill | Kill one or more running containers |
| logs | Fetch the logs of a container |
| ls | List containers |
| pause | Pause all processes within one or more containers |
| port | List port mappings or a specific mapping for the container |
| prune | Remove all stopped containers |
| rename | Rename a container |
| restart | Restart one or more containers |
| rm | Remove one or more containers |
| run | Create and run a new container from an image |
| start | Start one or more stopped containers |
| stats | Display a live stream of container(s) resource usage statistics |
| stop | Stop one or more running containers |
| top | Display the running processes of a container |
| unpause | Unpause all processes within one or more containers |
| update | Update configuration of one or more containers |
| wait | Block until one or more containers stop, then print their exit codes |

## “Docker ps” commands

The docker ps command is used to **list running containers**.

**Usage:**

#### docker ps

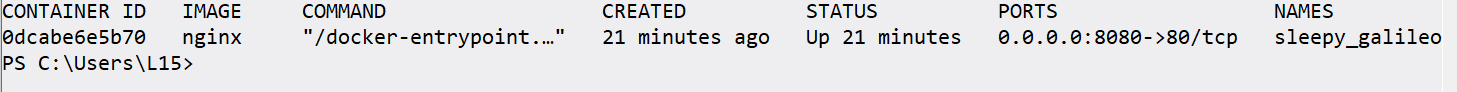
**Explanation:**

* Shows details of **running** containers by default.
* Displays **Container ID, Image, Command, Created time, Status, Ports, and Names**.

**Common Variations:**

|  |  |
| --- | --- |
| Command | Description |
| docker ps | List only **running** containers. |
| docker ps -a | List **all** containers (including stopped ones). |
| docker ps -q | Show only **container IDs** (useful for scripting). |
| docker ps --format "{{.Names}}" | Display only container names. |

For example, we had the next information about containers:



So, in order to get the ids of all containers we can use:

#### Docker ps -q

Forma

Descripción generada automáticamente con confianza baja

Which can be useful for scripting.

## About Client – Server Authentication

For these practices we’ll use Digital Ocean Droplets (Similar to EC2). And to Establish the connection there are to options.

**1. Password-Based Authentication 🔑**

* You log in using a **username** and **password**.
* **Pros:**
  + Simple to set up.
  + No extra keys needed.
* **Cons:**
  + Less secure (vulnerable to brute force attacks).
  + You have to remember/manage passwords.

**2. SSH Key-Pair Authentication (Recommended) 🗝️**

* Uses a **public-private key pair** for authentication.
* The **private key** stays on your local machine, while the **public key** is stored on the server.
* **Pros:**
  + More secure (no passwords to brute force).
  + Easier for automation (no need to type passwords).
  + Prevents unauthorized access if passwords are leaked.
* **Cons:**
  + Requires generating and managing SSH keys.
  + If you lose the private key, you may lose access.

The select one it’ll be SSH Key-pair Authentication.

# Register an account in Digital Ocean.

1. Create an account
2. Access to link: <https://www.digitalocean.com/>

# Creating Ubuntu Server on Digital Ocean and communication thought SSH.

When we are inside the main page, we’ll see this.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

A prior step we need to create SSH key pair.

## SSH Key Pair

### Check if ssh utility is installed

Texto

Descripción generada automáticamente

### Create a ssh pair key

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Then copy the info about the public key and paste it into the creation of the server in Digital Ocean.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Key pair added.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Selecting a name (Any name) and assigning it to a project.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Now, we have the droplet created.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

We got a public IP, so we can use it with ssh connection.

### Connect to the server using the private key

Texto

Descripción generada automáticamente

* First, we need to go to the path where the private key is.
* Then execute the command.

#### ssh -i .\docker\_rsa [root@137.184.220.95](mailto:root@137.184.220.95)

“root” is the main user in the Ubuntu Server.

# Installing Docker on Ubuntu Server (On Digital Ocean)

<https://docs.docker.com/engine/install/ubuntu/>

Execute the next steps

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

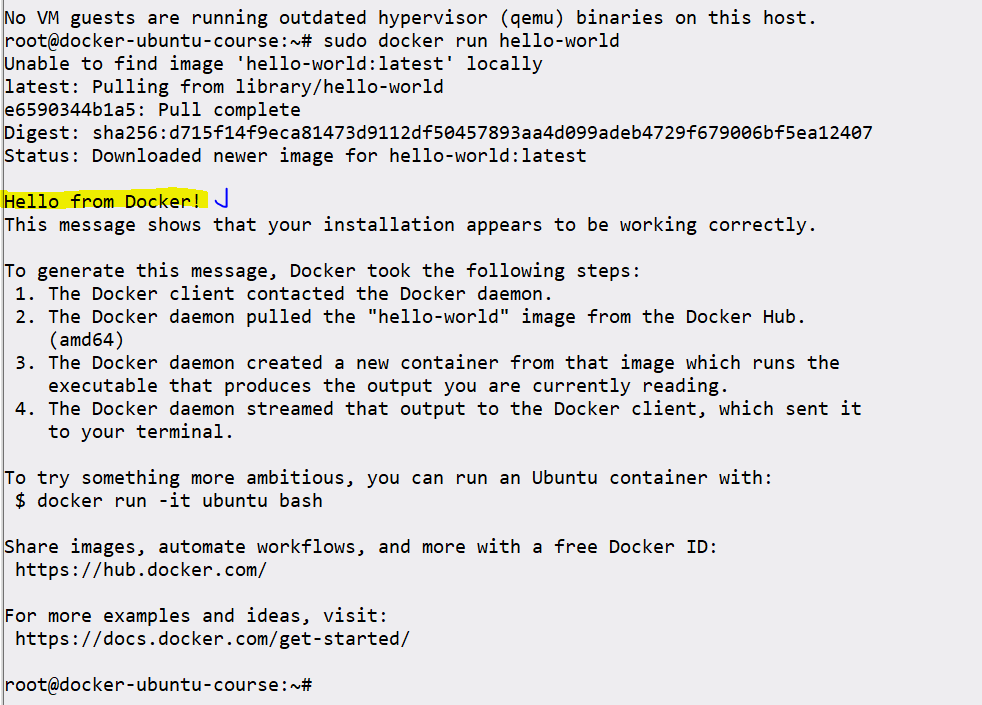
## Update ubuntu packages

Texto

Descripción generada automáticamente

Execute the other commands.

We finally get



Check “docker info”

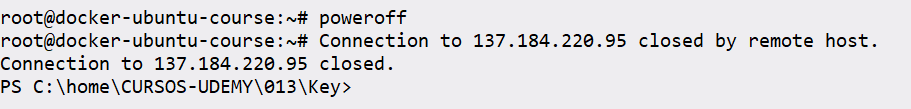
Texto

Descripción generada automáticamente

## Power off Droplet

Connected through SSH we execute:

#### poweroff



This will “power-off”

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

With the aim of reducing costs.

## Turn on Droplet

This option is available in:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# Work with the Droplet from ssh

Turn on the Droplet

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

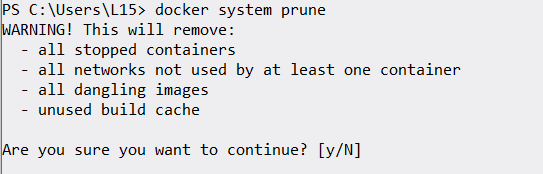
Execute the command where de public key is:

#### ssh -i .\docker\_rsa [root@137.184.220.95](mailto:root@137.184.220.95)

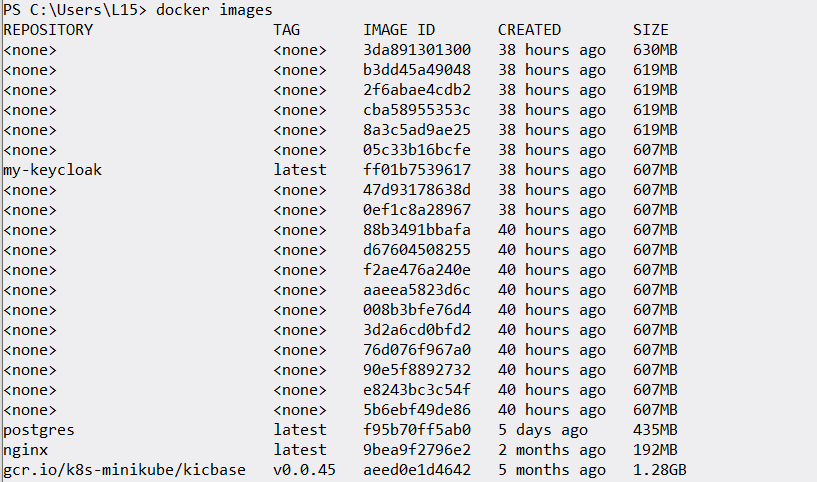
Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

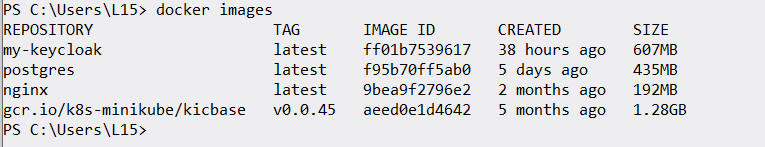
### Docker system prune



Check for images



Executing Docker system prune, the result is:



# Commands new approach

The new approach to commands is, for example, to stop containers we use:

Docker stop <container\_id>, but now we can also use docker container stop <container\_id>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Getting started with Dockers

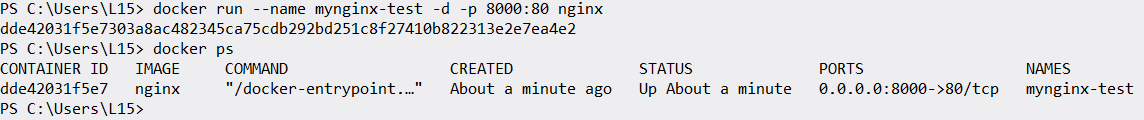
## Container identification

By adding “–-name=meaningful\_name” argument during docker run command, we can specify our own name to the containers.

Texto

El contenido generado por IA puede ser incorrecto.

Check in the name assigned to the new container:



Now with can manage the container using the name assigned to the container.

## Port Binding

In the command:

#### Docker run –name mynginx-test -d -p 8000:80 nginx

**-p 8080:80**

1. This flag maps the container's port **80** (where Nginx runs by default) to your host machine's port **8080**.
2. This means you can access Nginx by visiting http://localhost:8080 on your web browser.

Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

## Attached and Detached modes

When we start a docker container, we need to decide if we want to run in a default foreground mode or the detached mode.

### Attached mode

In the attached mode we’ll see the log’s container of the app running

Texto

El contenido generado por IA puede ser incorrecto.

### Detached mode

To run a container in a detached mode we can use the -d flag

#### Docker run -d 8080:80 nginx

## Removing docker containers

To stop containers, we can use the “stop” reserved word followed by the id’s containers.

### Imagen que contiene Gráfico de dispersión El contenido generado por IA puede ser incorrecto.

List Id’s container

#### Docker container ls -qa

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Now to stop all the containers

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

### Remove and specific container

#### Docker container rm <container\_id>

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

### Remove more than one stopped container in one go

Use this command to delete all stopped containers **in one go**

#### Docker ps -a

Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

The difference between **docker container ls -a** and **docker container ps -a** is that both commands do the same thing, but **docker container ls** is the more modern and recommended way, while **docker ps** is the classic form.

**Key Differences**

|  |  |  |
| --- | --- | --- |
| Command | Equivalent | Description |
| docker container ls -a | docker ps -a | Lists all containers (running, stopped, exited, etc.). |
| docker container ls | docker ps | Lists only running containers. |

## Docker container exec

In a Docker command, the -i and -t flags are used together to provide an interactive terminal session inside a container.

* -i (interactive): Keeps the STDIN (standard input) open, even if you are not attached to the container.
* -t (tty): Allocates a pseudo-TTY (a terminal session), making it behave like an actual terminal.

**Common Usage:**

When used together as -it, they allow you to interact with the container's shell

**Example:**

docker run -it ubuntu bash

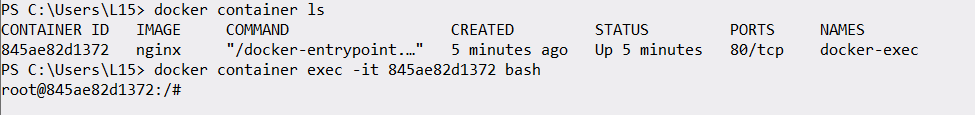
* -it: Enables interaction with the container.
* ubuntu: The base image used.
* bash: The command to run inside the container.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.



We can execute commands inside the container while not logically logged in to the container, for example:

Texto

El contenido generado por IA puede ser incorrecto.

From outside the container.

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

## Default Container command

How this command.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

How did my container automatically start nginx and it did not really start other things like bash or netstat or any other binary. The answer is the “Default Container command”

Texto

El contenido generado por IA puede ser incorrecto.

The nginx status in the container is “Running”

Texto

El contenido generado por IA puede ser incorrecto.

The default container command is in every image.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

## Overriding default container command

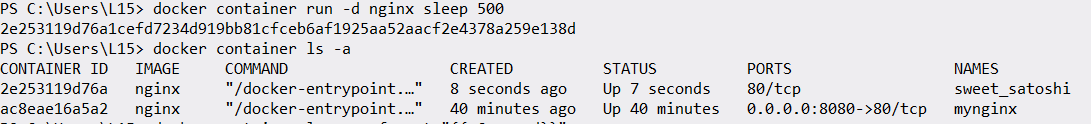
By default, for nginx container “nginx -g daemon off” is executed. But what if we want to change that default command.

There are two ways to change that default command.

### In the Dockerfile

In this way you can get the Dockerfile modifying it and create the image from the Dockerfile afterwards run the container.

The next command will override the default command:



So Args are changed.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

The default command has changed “CMD”

Texto

El contenido generado por IA puede ser incorrecto.

On the other hand, the other container.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

.Config.Cmd

* The Cmd property represents the **default command** that gets executed inside the container **if no additional command is provided at runtime**.
* It is defined in the Dockerfile using the CMD instruction.
* If an ENTRYPOINT is also defined, Cmd serves as **arguments** to ENTRYPOINT.

Example in a Dockerfile:

**FROM ubuntu**

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

If you start the container normally:

#### docker run ubuntu

It will execute:

echo "Hello, world!"

.Args

* The Args property shows the **actual arguments that were passed when the container was run**.
* This includes any command provided **when starting the container**.
* If the container uses an ENTRYPOINT, Args shows what was passed **after the ENTRYPOINT**.

For example, consider a container with:

**FROM ubuntu**

**ENTRYPOINT ["echo"]**

**CMD ["Hello, world!"]**

If you run:

#### docker run ubuntu

* .Config.Cmd: ["Hello, world!"]
* .Args: ["Hello, world!"]

But if you override the command:

**docker run ubuntu "Goodbye”**

* .Config.Cmd: ["Hello, world!"] (default, but overridden)
* .Args: ["Goodbye"] (actual argument passed)

**Summary**

|  |  |
| --- | --- |
| Property | Purpose |
| .Config.Cmd | Default command set in Dockerfile (overridable at runtime) |
| .Args | Actual arguments passed when running the container |

If you see the **same values** in .Args and .Cmd, it's likely because:

1. You didn't override the command when running the container.
2. The container doesn't have an ENTRYPOINT, so Cmd directly executes.

### How ENTRYPOINT and CMD Work Together

When both ENTRYPOINT and CMD are defined in a Dockerfile:

* ENTRYPOINT defines **the command that will always run** when the container starts.
* CMD provides **default arguments** to the ENTRYPOINT command.
* If you specify additional arguments when running the container, they **override CMD**, but **ENTRYPOINT remains unchanged**.

**Example 1: Using Both ENTRYPOINT and CMD**

FROM ubuntu

ENTRYPOINT ["echo"]

CMD ["Hello, world!"]

**What happens:**

1. ENTRYPOINT ["echo"] ensures that **every time the container runs, echo will execute**.
2. CMD ["Hello, world!"] serves as **default arguments** for echo.

**Running the container:**

#### docker run my-image

🔹 Output:

The command executed inside the container is:

echo "Hello, world!"

**Example 2: Overriding CMD**

Now, if we run the container with additional arguments:

#### docker run my-image "Goodbye!"

🔹 Output:

The command executed inside the container is:

echo "Goodbye!"

🔹 **What happened?**

* ENTRYPOINT (echo) **stayed the same**.
* CMD ("Hello, world!") was **replaced by** "Goodbye!".

**Example 3: Overriding Both ENTRYPOINT and CMD**

If you want to completely override ENTRYPOINT, you must use the --entrypoint flag:

#### docker run --entrypoint ls my-image -l

🔹 This run:

#### ls -l

Instead of echo "Hello, world!".

**Final Summary**

|  |  |
| --- | --- |
| Definition | Behavior |
| ENTRYPOINT | **Command that always runs** when the container starts. |
| CMD | **Default arguments for ENTRYPOINT** (can be overridden at runtime). |

If you define both:

ENTRYPOINT ["some-command"]

CMD ["default-arg"]

Running docker run my-image is like running:

**some-command default-arg**

But if you specify the arguments:

**docker run my-image custom-arg**

It executes:

**some-command custom-arg**

### Order of commands

**Order of CMD and ENTRYPOINT in the Dockerfile matters**, but in a specific way:

* CMD **must go after** ENTRYPOINT if both are used together.
* The reason: CMD provides **default arguments** to ENTRYPOINT, so if ENTRYPOINT is defined after CMD, it will ignore CMD completely.

### Running with custom arguments:

#### docker run my-image "Goodbye" "Docker!"

Commands should go after image name.

### ****Final Summary****

* 🖼️ **For images**, use docker inspect my-image to check **original CMD/ENTRYPOINT**.
* 📦 **For containers**, use docker inspect my-container to check **CMD, ENTRYPOINT, and runtime ARGS**.
* ⏳ **Passing arguments when running a container does not change the image**, but affects Args in the container.

# Restart Policies in Docker

These policies are used to restart the container if the docker daemon is restarted.

Tabla

El contenido generado por IA puede ser incorrecto.

Unless stopped is used in “PRODUCTION”.

The unless-stopped option in Docker is used to define the **restart policy** of a container. Here’s how it works:

1. **The container will automatically restart if it stops unexpectedly** (e.g., due to an internal failure).
2. The **container will restart when Docker or the system is restarted**.
3. **The container will NOT restart if it was manually stopped** using the docker stop <container\_id> command.

**Example usage:**

#### docker run -d --restart unless-stopped my-container-image

**Behavior in different scenarios:**

|  |  |
| --- | --- |
| Scenario | Will the container restart? |
| Unexpected container failure | ✅ Yes |
| Docker or system restart | ✅ Yes |
| Manually stopped with docker stop | ❌ No |
| Manually started again with docker start <container\_id> | ✅ Yes |

This mode is useful when you want the container to **run persistently**, but without restarting automatically after a **manual stop**.

### Restart docker

#### Systemctl restart docker

## Removing docker Container Images

We can use two commands:

#### Docker container rm <container\_id>

#### Docker rm <container\_id>

Texto

El contenido generado por IA puede ser incorrecto.

## Disk usage metrics for Docker components

Monitor for disk usage on the host. The usage of disk of one container can affect performance or are not able to run effectively.

Texto

El contenido generado por IA puede ser incorrecto.

Gráfico

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

More about this in:

<https://www.udemy.com/course/docker-certified-associate/learn/lecture/14546148#overview>

Texto

El contenido generado por IA puede ser incorrecto.

**How to Free Up Space?**

If docker system df -v shows **a lot of reclaimable space**, you can clean it up using:

**🗑 Remove all unused data (images, containers, volumes, networks)**

#### docker system prune -a

🔹 **Warning:** This deletes **all unused** images, stopped containers, and networks!

**6️⃣ Summary**

|  |  |
| --- | --- |
| Command | Description |
| docker system df | Show disk usage summary for Docker. |
| docker system df -v | Show **detailed** disk usage (per image/container/volume). |
| docker system prune -a | Remove **all unused** Docker resources to free space. |

The space used by a **container** is **different** from the space used by the **image** itself. Here's how **Docker disk usage works**:

**Image vs. Container Space Usage**

|  |  |  |
| --- | --- | --- |
| Component | Description | Stored In |
| Image | The base template (OS + app) is used to create containers. | /var/lib/docker/images |
| Container | A running instance of an image, with additional changes (logs, temp files, writes). | /var/lib/docker/containers |
| Volumes | Persistent data storage for containers. | /var/lib/docker/volumes |
| Build Cache | Cached layers from Docker builds. | /var/lib/docker/buildkit |

Texto

El contenido generado por IA puede ser incorrecto.

**🔹 Containers create a writable layer on top of the image**

* **The base image remains unchanged**.
* **Any changes inside the container (logs, new files, updates) go into a writable layer**.
* When a container is deleted, **its writable layer is removed**, but the image remains.

**Example of Space Usage**

Let's say we have an image:

#### docker images

nginx

REPOSITORY TAG IMAGE ID SIZE

ubuntu latest 123456789abc 100MB

The **image size is 100MB**.

Now, we run a container and modify it:

#### docker run -dit --name test-container ubuntu

#### docker exec test-container apt update && apt install -y nano

This **adds new data inside the container**, increasing its size.

Checking space usage:

#### docker system df -v

TYPE TOTAL ACTIVE SIZE RECLAIMABLE

Images 1 1 100MB 0B

Containers 1 1 50MB 10MB (20%)

* The **container** is using **50MB** (because we installed nano).
* The **original image** still remains **100MB**.
* The **container’s extra 50MB is in its writable layer**.

**How Does This Affect Storage?**

* **Stopping or deleting a container does NOT delete the image**.
* **Removing an image does NOT remove containers using it**.
* **Deleting a container removes its writable layer** but **keeps the image**.

**How to Clean Up Space?**

**Remove a container and its space**

#### docker rm -v test-container

🗑️ Removes the **container** and its **extra data**.

**Remove unused images**

#### docker image prune -a

🗑️ Deletes **unused images** that are **not linked to a container**.

**Remove all unused containers, images, and volumes**

#### docker system prune -a

🗑️ Cleans **everything unused** (except active containers).

**Final Summary**

|  |  |  |
| --- | --- | --- |
| Component | Uses Space? | When? |
| Image | ✅ Yes | Always (stored in Docker). |
| Container | ✅ Yes | When running or stop (has a writable layer). |
| Volumes | ✅ Yes | When containers store persistent data. |

💡 **Containers add space on top of images** but don’t modify the image itself. **Cleaning up stopped containers and unused images helps free disk space.** 🚀

## Automatically delete Container on Exit

Execute this comman:

#### docker container run -dt --name testcontainer busybox ping -c10 google.com

When the job completes the container will stop it but not delete it.

Texto

El contenido generado por IA puede ser incorrecto.

To achieve deletion of the container we need to pass -rm

#### docker container run -dt --rm --name testcontainer busybox ping -c10 google.com

Código QR

El contenido generado por IA puede ser incorrecto.

In this way the container will be deleted after it is stopped.

# Image creation, management, and registry

## Overview of Dockerfile

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

The format of Dockerfile is like the below syntax:

**# Comment**

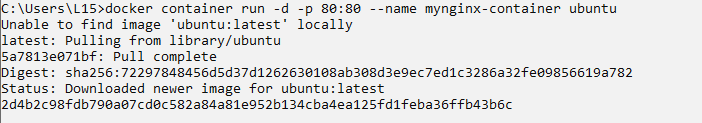
**INSTRUCTION arguments**

A Dockerfile must start with a **FROM** instruction. The **FROM** instruction specifies the Base Image from which you are building.

## Creating our first Dockerfile

Let’s see the steps we need to execute for creations of the container:

#### docker container run -d -p 80:80 --name mynginx-container ubuntu



But when we executed this command, we got this in the list of containers:

STEP 1:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

This happens because Docker containers run as long as their main process is active. The official Ubuntu image is a minimal base image and doesn’t have a long-running process by default. When you run:

**docker container run -d -p 80:80 --name mynginx-container ubuntu**

Docker starts the container in detached mode, but the default command (if none is specified) immediately finishes, so the container exits shortly afterward.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

But if we execute the same commands adding -t:

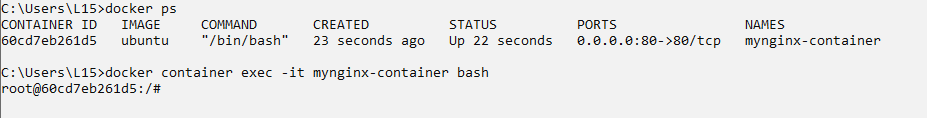
**docker container run -d -p 80:80 -t --name mynginx-container ubuntu**

You’re allocating a pseudo-TTY for the container’s main process. In the official Ubuntu image, the default command is typically something like a shell (e.g., /bin/bash). Without a TTY, the shell may detect it’s not in an interactive environment and immediately exit. However, with the -t flag, the shell “thinks” it’s running interactively, so it stays open waiting for input—even though no interactive input is coming in detached mode.

**What’s Happening Under the Hood**

* **Without -t:**  
  The default shell in the Ubuntu image sees that it isn’t attached to a terminal and may finish immediately, causing the container to exit.
* **With -t:**  
  Allocating a TTY makes the shell behave as if it’s in an interactive session, keeping it running. This is why your container remains alive even though you haven’t provided a long-running service or command.

Step 2: Get into the bash



Step 3: Update repositories

Texto

El contenido generado por IA puede ser incorrecto.

Step 4: Install NGINX

Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Step 5: Check for NGINX status:



Here is the index:



Overriding content:



Step 6: Starting NGINX



Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

I we are using Droplet you should use the IP of your server.

### Dockerfile

We’ll put in the same folder the index and the Dockerfile:

Imagen que contiene nombre de la empresa

El contenido generado por IA puede ser incorrecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto.

Texto

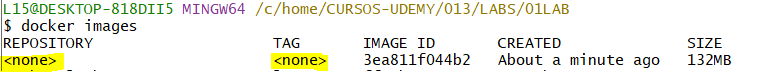
El contenido generado por IA puede ser incorrecto.

#### Docker build .

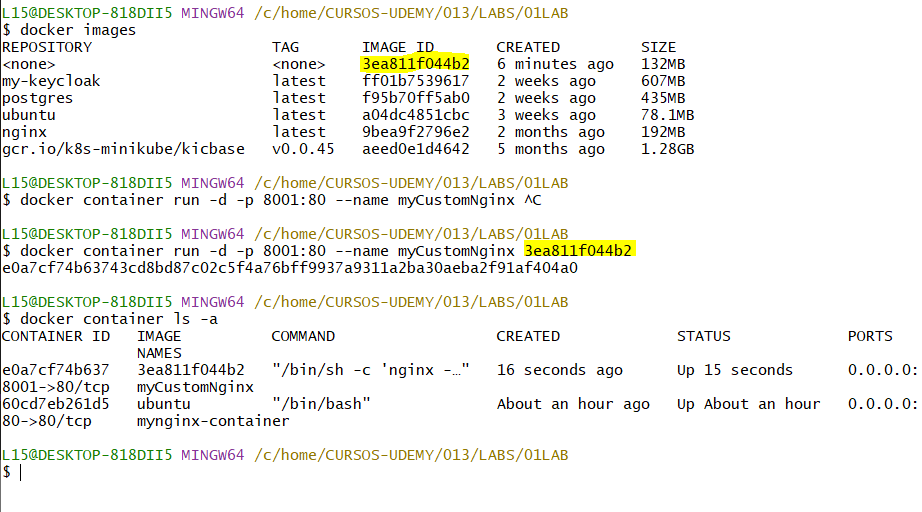
Texto

El contenido generado por IA puede ser incorrecto.

Now we have this new image:



New container created from our custom image:



Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

## COPY VS ADD

In a directory we have this:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Dockerfile instructions:

Texto

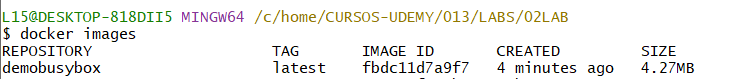
El contenido generado por IA puede ser incorrecto.

Build the image:

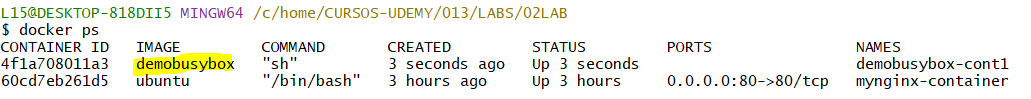


Texto

El contenido generado por IA puede ser incorrecto.



Now with the image we can create a container.



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Every Command: ADD, RUN… Create a new layer, this increases the size of the image.

In Docker, both ADD and COPY are used to copy files and directories into a Docker image, but they have some key differences:

**1. COPY (Recommended in Most Cases)**

* COPY simply copies **files and directories** from the host machine into the Docker image.
* It **does not** handle compressed files (e.g., .tar.gz).
* It has a straightforward syntax:

#### COPY source destination

* Example:

#### COPY myfile.txt /app/myfile.txt

#### COPY myfolder /app/myfolder

**2. ADD (More Powerful but Less Predictable)**

* ADD does everything COPY does **but with additional features**:
  1. It can **extract compressed files (tar archives)** automatically.

#### ADD myarchive.tar.gz /app/

* + - This will **extract** the contents of myarchive.tar.gz into /app/ automatically.
  1. It **allows remote URLs**, downloading files from the internet directly.

#### ADD https://example.com/file.zip /app/file.zip

* + - This will download file.zip from the URL and store it in /app/file.zip.

**Which One Should You Use?**

* **Use COPY** whenever you simply need to copy files/directories **without automatic extraction**.
* **Use ADD** only if you specifically need:
  + **Auto-extraction** of .tar.gz files.
  + **Downloading from a URL** (though curl or wget is usually preferred in a RUN command for better control).

**Best Practice**

The Docker documentation **recommends using COPY over ADD** unless you need the extra functionality of ADD, as it makes the behavior more predictable.

## EXPOSE command

The **EXPOSE** instructions inform Docker that the container listens on the specified network ports at runtime.

The **EXPOSE** instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.

#### Docker run -d –name mynginx nginx

Texto

El contenido generado por IA puede ser incorrecto.

In this command the column “PORTS” show 80/tcp that information comes from the expose instruction which the image creator had added within the Dockerfile.

Executing

#### Docker inspect mynginx

Texto

El contenido generado por IA puede ser incorrecto.

Then we can publish that port to be accessible from the exterior.

Texto

El contenido generado por IA puede ser incorrecto.

Now any traffic which comes on the host port of 80 will automatically be forwarded to the container port of 80.

In other words, the port of the server on 80 will be forwarded to the port 80 of the nginx container.

## HEALTCHECK INSTRUCTIONS

#### Docker container run -dt –-name busybox-container busybox sh

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Let’s check the IP of the container:

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

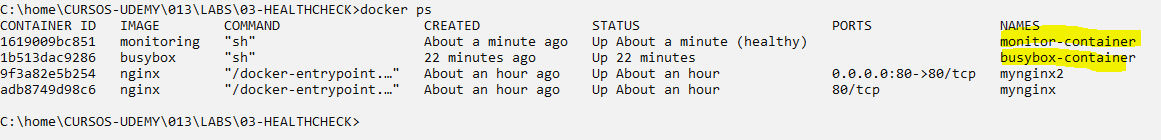
El contenido generado por IA puede ser incorrecto.

Creating the container that makes ping to the other container using the Image created by previously.

Texto

El contenido generado por IA puede ser incorrecto.

Here the “monitor-container” is a container that was built to make a ping to the “busybox-container”.



Let’s inspect “monitor-container”.

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto.

Exit code 0 means, execution has been done successfully. Exit code 1 means unhealthy.

Now we’ll stop the target container:

Imagen que contiene Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

Waiting for state container, STATUS of monitoring container will change to “unhealthy”

Calendario

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

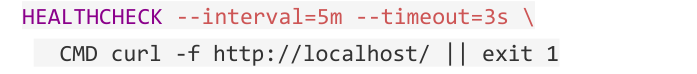
El contenido generado por IA puede ser incorrecto.

## HEALTHCHECK OPTIONS

Tabla

El contenido generado por IA puede ser incorrecto.

Simple User Case



It will check every five minutes whether the curl on the localhost is working or not and the timeout is 3 seconds which means the result will be displayed in that time.

## SHELL commands

El comando **SHELL** en un Dockerfile permite cambiar el shell predeterminado que se usa cuando ejecutas comandos en la forma de **shell** (como con RUN, CMD, y ENTRYPOINT cuando no se especifican en forma de array).

Por defecto:

* En **Linux**, el shell predeterminado es ["/bin/sh", "-c"], lo que significa que los comandos se ejecutan con **sh**.
* En **Windows**, el shell predeterminado es ["cmd", "/S", "/C"], lo que significa que los comandos se ejecutan con **cmd.exe**.

**¿Para qué sirve SHELL?**

Sirve para cambiar el shell con el que se ejecutan los comandos en el Dockerfile. Esto es especialmente útil en **Windows**, donde puedes elegir entre cmd.exe, PowerShell, o sh de Git Bash, por ejemplo.

**Ejemplo en Windows:**

Si quieres que los comandos RUN se ejecuten en **PowerShell** en lugar de cmd.exe, puedes hacer esto:

# Cambia el shell predeterminado a PowerShell

SHELL ["powershell", "-Command"]

RUN Write-Host "Este comando se ejecuta con PowerShell"

En este caso, en lugar de ejecutar con cmd.exe, los comandos usarán PowerShell.

**Ejemplo en Linux:**

Aunque en Linux normalmente no es necesario cambiar el shell, podrías hacerlo para usar bash en lugar de sh:

SHELL ["/bin/bash", "-c"]

RUN echo "Ahora los comandos se ejecutan con Bash"

Esto es útil si necesitas ejecutar comandos que requieren características específicas de **Bash** que no están disponibles en **sh**.

**Resumen:**

* SHELL cambia el shell que Docker usa para ejecutar comandos en el Dockerfile.
* Es más útil en Windows, donde hay varias opciones de shell (cmd.exe, PowerShell, sh).
* Puede usarse en Linux para cambiar de sh a bash, si es necesario.
* Se usa en forma de **JSON** (SHELL ["executable", "parameters"]).
* Cada SHELL afecta solo a los comandos que aparecen después de él en el Dockerfile.

## Testing the HEALCHECK Commands

We have the following commands:

HEALTHCHECK --interval=5m --timeout=3s \

[CMD](https://docs.docker.com/reference/dockerfile/#cmd) curl -f http://localhost/ || exit 1

This command will check every five minutes whether the curl on localhost is working or not in the timeout established of three seconds.

The options can be used as part of the Dockerfile but also in the Docker run command. These are the flags options:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

With the following options we will use the default values.

Texto

El contenido generado por IA puede ser incorrecto.

Then check the container state:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

After more than 30 seconds show “unhealthy” state:

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

That’s because there’s nothing running in the specified URL.

Let’s inspect “tmp” container

Imagen que contiene Esquemático

El contenido generado por IA puede ser incorrecto.

The result of the curl:

Texto, Carta

El contenido generado por IA puede ser incorrecto.

## “ENTRYPOINT” and “CMD” COMMAND

It is used normally to set the image’s main command.

ENTRYPOINT doesn’t allow you to override the command.

For example, we have this Dockerfile:

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Building image:

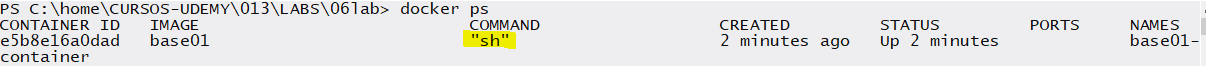
Texto

El contenido generado por IA puede ser incorrecto.

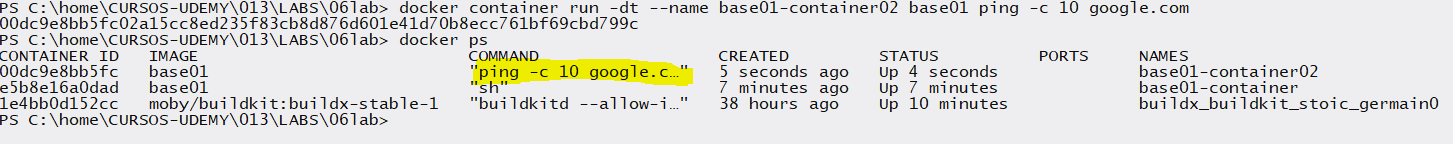
Create a container from the image:



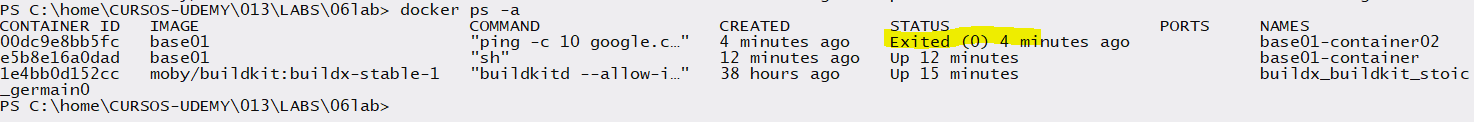
We’ll see the which the exec command is connected to “sh”



This default command could be overridden using parameters after the image name:



After 10 times ping being executed and the container will exit.



The ENTRYPOINT cannot be overridden so if we have this in our Dockerfile:

Interfaz de usuario gráfica, Word

El contenido generado por IA puede ser incorrecto.

Let’s build a new image:

Texto

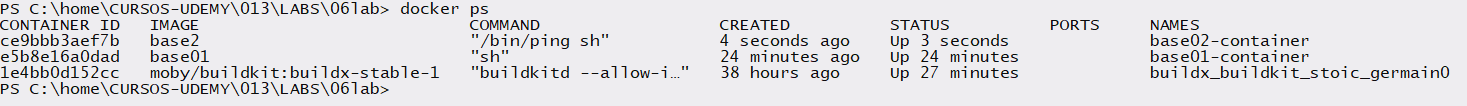
El contenido generado por IA puede ser incorrecto.

Now run a container with the next command:

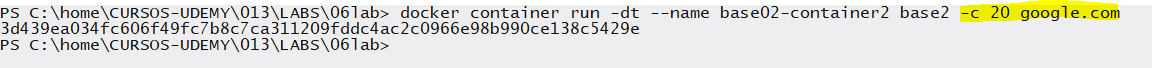


This will try to do something like:

/bin/ping sh



Which is a not valid command then will exit immediately. On the other hand, we can pass:



Which is a valid command:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Exited (0) means succeeded.

Exited (1) means failed.

## WORKDIR INSTRUCTION

The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow int the Dockerfile.

Texto, Carta

El contenido generado por IA puede ser incorrecto.

Build an image:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Creating a container from the image:

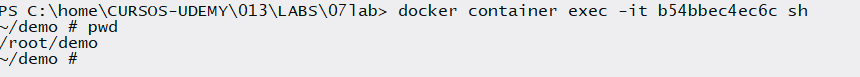
Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Get into the container:



If we get into the container:



Automatically I will get into the path where the WORKDIR was set.

Now if we do not create the directory with the RUN MKDIR:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Building an image from Dockerfile:

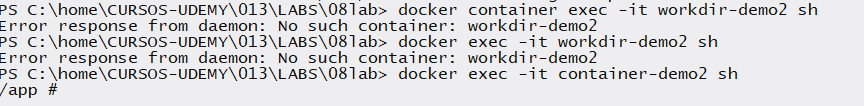
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Running a container:



Ingress into the container:



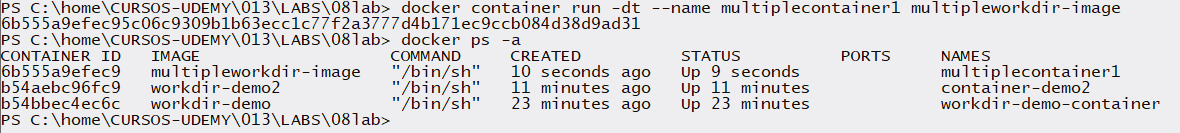
In the same way the WORKDIR will be created.

Third case:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Listing containers:

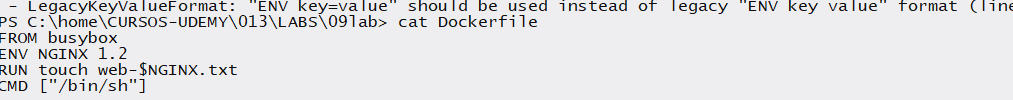


The last WORKDIR set will be the default path when you get into the container, but all the other paths are created as well:

Imagen que contiene Escala de tiempo

El contenido generado por IA puede ser incorrecto.

## “ENV” INSTRUCTION



Build an image a run container from it.

Texto

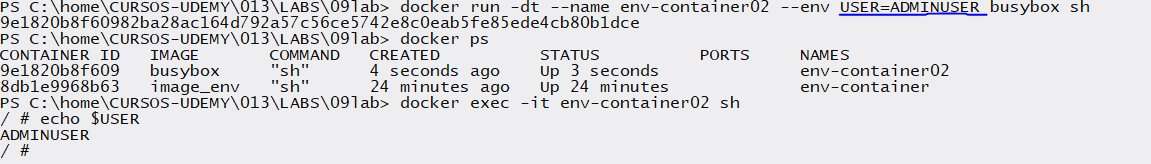
El contenido generado por IA puede ser incorrecto.

The changes can be seen as follows:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

We can pass that env arguments:



when you run:

#### docker run -dt --name env-container02 --env USER=ADMINUSER busybox sh

You are setting the USER environment variable inside the running container. If the busybox image already has a predefined USER environment variable, your --env USER=ADMINUSER option **will override** it.

**How Environment Variable Overriding Works in Docker:**

1. **Base Image Defaults**: The image may define environment variables in its Dockerfile using the ENV instruction (e.g., ENV USER=root).
2. **Run-time Overrides**: When you specify --env or -e in docker run, it **overrides** the predefined value inside the container for that session.

**Verifying the Value:**

After starting the container, you can check the value by running:

#### docker exec -it env-container02 sh -c 'echo $USER'

This should output:

**ADMINUSER**

If you need to retain the default value but also allow overrides, you can check the existing value inside your Dockerfile:

ENV USER=${USER:-default\_value}

This way, it only changes if not explicitly set at runtime.

Then, the behavior will be:

* If **no** USER variable is passed when running the container, it defaults to **default\_value.**
* If USER **is set** at runtime using --env USER=ADMINUSER, then USER will take the value ADMINUSER.

**1. Difference Between ARG and ENV**

|  |  |  |
| --- | --- | --- |
| Feature | ARG | ENV |
| Scope | Only available **at build time** | Available **at runtime** inside the container |
| Accessibility | Cannot be accessed inside a running container | Accessible inside the running container |
| Default Value | Can be set using ARG VAR=value | Can be set using ENV VAR=value |
| Overridable | Can be overridden using --build-arg at build time | Can be overridden using --env or ENV in the Dockerfile |
| Persistence | **Not** preserved in the final image | **Preserved** in the final image and used by containers |

# Define an ARG (only available at build time)

ARG BUILD\_VERSION=1.0

# Define an ENV variable (available at both build and runtime)

ENV APP\_VERSION=${BUILD\_VERSION}

# Set another ENV variable directly

ENV USER=defaultuser

**3. How They Behave in Different Scenarios**

**Case 1: Overriding ARG at Build Time**

You can override ARG while building the image:

#### docker build --build-arg BUILD\_VERSION=2.0 -t myimage .

* Inside the build process, BUILD\_VERSION is 2.0.
* APP\_VERSION is set to 2.0 because ENV APP\_VERSION=${BUILD\_VERSION}.

However, since ARG **does not persist** in the image, if you inspect or run a container from this image, BUILD\_VERSION will be **gone**, but APP\_VERSION will still exist.

**Case 2: Overriding ENV at Runtime**

You can override ENV when running a container:

#### docker run -dt --name mycontainer --env APP\_VERSION=3.0 --env USER=adminuser myimage sh

* Inside the container:

**echo $APP\_VERSION # Output: 3.0 (overridden)**

**echo $USER # Output: adminuser (overridden)**

**4. Summary: Can Both ARG and ENV Be Passed and Replaced?**

✅ **Yes, but with different rules**:

* **ARG is for build-time** and is not available at runtime.
* **ENV is for runtime** and can be overridden when running a container.
* **You can pass ARG at build-time and use it to set an ENV variable** for runtime.

**ENV** variables defined in a Dockerfile can be overridden **only when running the container**, using the --env (-e) flag in the docker run command.

**How ENV Works:**

1. If you define an ENV variable in a **Dockerfile**, it is stored in the image and available **by default** inside any container created from that image.
2. If you **override** it with --env in docker run, the new value is used **only for that container**.
3. The **original ENV value in the image does not change**—only the running container sees the new value.

**Example of ENV Behavior**

**1. Define ENV in Dockerfile**

FROM busybox

ENV USER=defaultuser

CMD ["sh"]

**2. Build the Image**

#### docker build -t myimage .

**3. Run a Container Without Overriding ENV**

#### docker run --rm myimage sh -c "echo \$USER"

🟢 **Output:** defaultuser

**4. Run a Container and Override ENV**

#### docker run --rm --env USER=adminuser myimage sh -c "echo \$USER"

🟢 **Output:** adminuser (Overridden value)

**Key Takeaways**

✅ **Default ENV values persist in the image.**  
✅ **You can override ENV when running a container using --env.**  
❌ **Changing ENV in a running container does not update the image.**  
❌ **ENV variables cannot be changed after the container has started unless explicitly set.**

**ENV variables cannot be directly set using the docker build command**. You can only set **ARG** variables during the build process using --build-arg. However, you can use ARG to pass values and **then assign them to ENV inside the Dockerfile.**

**🚀 How to Pass Variables in docker build**

Since ENV is meant for runtime, **you must use ARG to pass values at build time** and then set ENV using ARG values.

**Example: Using ARG to Set ENV in a Dockerfile**

# Define an ARG variable (available only at build time)

ARG BUILD\_USER=defaultbuilduser

# Assign ARG to an ENV variable (available at runtime)

ENV USER=${BUILD\_USER}

# Use CMD to test the variable

CMD ["sh", "-c", "echo USER is $USER"]

**🏗️ Build the Image with --build-arg**

#### docker build --build-arg BUILD\_USER=custombuilduser -t myimage .

* Here, BUILD\_USER=custombuilduser is passed **at build time**.
* Since ENV USER=${BUILD\_USER}, USER will now be custombuilduser **inside the image**.

**Run the Container Without Overriding ENV**

#### docker run --rm myimage

🟢 **Output:**

#### USER is custombuilduser

✅ The ENV USER took the value from ARG BUILD\_USER at build time.

**Run the Container and Override ENV**

#### docker run --rm --env USER=adminuser myimage

🟢 **Output:**

#### USER is adminuser

✅ The USER variable was overridden **at runtime**.

**🔑 Key Takeaways**

1. **ARG can be set using --build-arg in docker build**
2. **ARG exists only during build time and is not available at runtime**
3. **You can assign ARG to ENV in the Dockerfile so that it persists in the container**
4. **At runtime, ENV can still be overridden using --env in docker run**

## Tagging Docker images

When creating an image:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

The image is untagged and has no associated repository.

Second way:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Will be tagged with latest but if we need to specify a tag:

Texto

El contenido generado por IA puede ser incorrecto.

Remove tagged images

Texto, Carta

El contenido generado por IA puede ser incorrecto.

We can tag an untagged image:

Texto

El contenido generado por IA puede ser incorrecto.

### Create a tag associated with an already existing image

An image with the same Id can have multiple tags:

Imagen que contiene Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

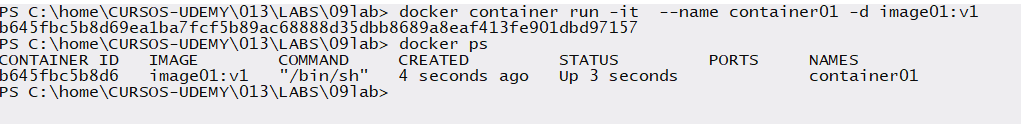
Or using the id:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

## Docker commit

Let’s create a container:



Let’s enter an create a file:

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

* + 1. We’ve created an image from the container that has been changed.
    2. Let’s create a container from the modified image.
    3. Let’s check in the new container the file created in the first step.

We can change the values of the container in a commit for example:

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

We notice that the CMD command that is trying to be changed Its being concatenated:

Calendario

El contenido generado por IA puede ser incorrecto.

## Layers of Docker Image

A docker image is built up from a series of layers, each layer represents an instruction in the image’s Dockerfile.

Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.

The major difference between a container and an image is the top writeable layer.

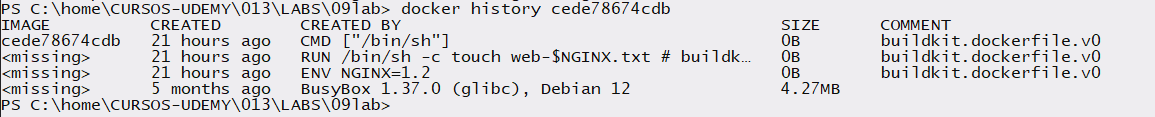
When you run a Docker container, it is based on an **image**. Docker images are **read-only** and consist of multiple layers (built using the Dockerfile instructions like FROM, RUN, COPY, etc.).

However, when a container starts, **Docker adds a writable layer on top of the read-only image layers**. This writable layer allows changes to be made inside the running container.

* Any **new files created**, **existing files modified**, or **deleted files** are stored in this **writable layer**.
* If you **stop the container**, the writable layer is preserved.
* If you **delete the container**, the writable layer is lost.

This allows us create “N” containers from the same base image.

Check the layers of an image:



For example, this ubuntu image has 6 layers:

Texto

El contenido generado por IA puede ser incorrecto.

Building an image with 2 extra commands:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Now checking layers again:

Captura de pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto.

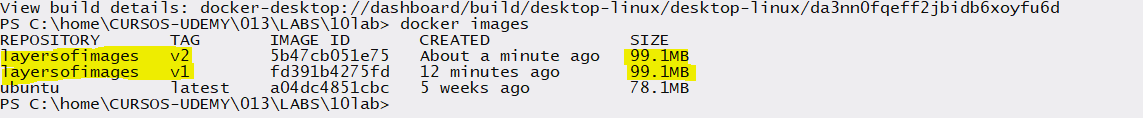
We notice there are 2 more layers.

What happens if we add 2 more instructions to remove this file, what would be the size at the end.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Let’s check the image’s size:



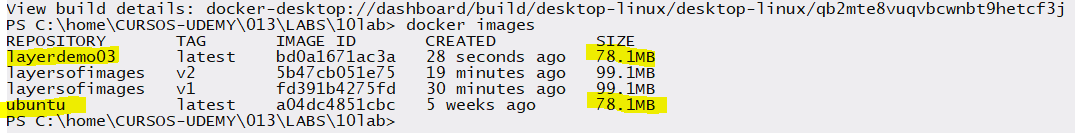
Even though we remove the file size is the same because the previous layers persist, it means that that files are removed in the two last layers and the previous ones are not modified. THE FILE STILL EXITS IN THE LAYERS THAT WERE CREATED.

To ensure that the final image retains the same size as the images where the files were added, the deletion of those files must occur in the same layer where they were created.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Now, we can see that the image sizes are the same.



That’s why in that layer 0B was added to the final image.

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

Less layers will help us to have better performance, if we can reduce layers when you install utilities or update packages we should try reducing the number of layers.

## Inspecting Docker Images

Texto

El contenido generado por IA puede ser incorrecto.

To check specific information

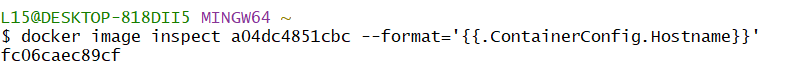
Texto

El contenido generado por IA puede ser incorrecto.

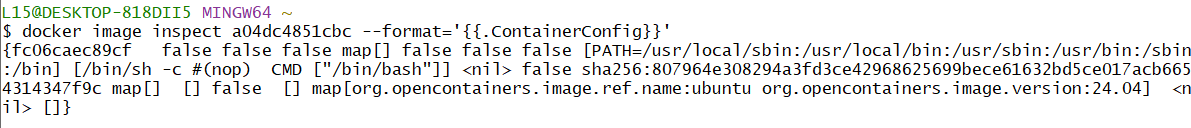
Using the format parameter:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.



If we need to see tag with more information:



To see JSON format, key-vault format:

Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

## Pruning Docker Images

Docker image prune, by default command will only clean up dangling images.

Dangling Images: Image without Tags and Image not referenced by any container.

Texto

El contenido generado por IA puede ser incorrecto.

#### Docker image prune -a

Will remove all the images which are not being referenced by any container.

Texto

El contenido generado por IA puede ser incorrecto.

This command is related to images there exits prune for networks, volumes, containers and so forth.

## Modifying Image to Single Layer

Let’s check the layers on ubuntu layers.

Imagen que contiene Texto

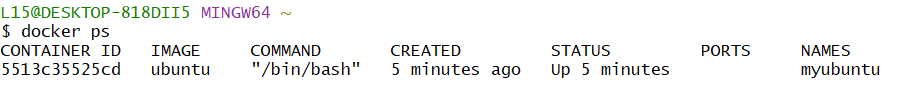
El contenido generado por IA puede ser incorrecto.

Let’s create a container:

Texto

El contenido generado por IA puede ser incorrecto.

To merge the layers of the image there exists some approaches, but the approach we’ll use is **export/import container.**



Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

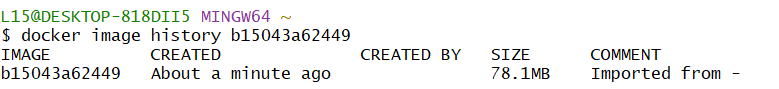
Now our container has been exported to us .tar file.

Next step let’s import from tar file into a new tagged image.

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

Let’s check for the layers’ new image.



Now just one layer is present.

## Docker registries

**1st Command:**

#### $ docker run -d -p 5000:5000 --restart always --name registry registry:2

**Breakdown:**

* **docker run** → Runs a new container.
* **-d** → Runs the container in detached mode (in the background).
* **-p 5000:5000** → Maps port **5000** on the host to port **5000** in the container.
* **--restart always** → Ensures the container always restarts if it stops or when Docker restarts.
* **--name registry** → Assigns the container the name registry.
* **registry:2** → Uses the image registry with the tag 2 (which refers to **Docker Registry v2**).

**What does this command do?**

It runs a **Docker private registry** on port 5000, allowing you to store and distribute Docker images privately.

#### $ docker pull ubuntu

**Breakdown:**

* **docker pull** → Downloads an image from Docker Hub.
* **ubuntu** → The name of the image (defaults to the latest version if no tag is specified).

**What does this command do?**

It downloads the latest ubuntu image from Docker Hub onto your local machine.

**3rd Command:**

#### $ docker tag ubuntu localhost:5000/ubuntu

**Breakdown:**

* **docker tag** → Creates a new tag (alias) for an existing image.
* **ubuntu** → The source image to tag.
* **localhost:5000/ubuntu** → The new name of the image, where:
  + **localhost:5000** → Specifies the private Docker registry running on port 5000.
  + **ubuntu** → The new image name inside the registry.

**What does this command do?**

It **renames** the ubuntu image so that it points to the private registry at localhost:5000, preparing it for upload.

**4th Command:**

#### $ docker push localhost:5000/ubuntu

**Breakdown:**

* **docker push** → Uploads an image to a registry.
* **localhost:5000/ubuntu** → The image to upload to the private registry.

**What does this command do?**

It **pushes** the ubuntu image to the private Docker registry running on localhost:5000.

**Final Summary of What These Commands Do**

1. **Start a private Docker registry** (docker run ... registry).
2. **Download an Ubuntu image** from Docker Hub (docker pull ubuntu).
3. **Tag the Ubuntu image** for the private registry (docker tag ubuntu localhost:5000/ubuntu).
4. **Push the Ubuntu image** to the private registry (docker push localhost:5000/ubuntu).

Now, the ubuntu image is stored in your private registry and can be pulled from localhost:5000/ubuntu instead of Docker Hub.

So, this:

#### $ docker tag ubuntu localhost:5000/ubuntu

What does “tag”? And generally, what does the entire command create a new image or create a reference?

“TAG” is nothing but the Alias.

The command:

#### $ docker tag ubuntu localhost:5000/ubuntu

**does NOT create a new image**. Instead, it creates a **reference (alias)** to the existing image.

**What does "tag" mean in Docker?**

Tagging in Docker is like assigning a new **name** or **label** to an existing image. It does **not** duplicate or modify the image but simply provides another way to reference it.

**What does command do?**

* It creates a **new reference** (localhost:5000/ubuntu) for the **existing** ubuntu image.
* The actual image **remains the same**, but now it can be pushed to a registry or used under a different name.

**Example:**

After running:

#### $ docker tag ubuntu localhost:5000/ubuntu

If you check your local images with:

#### $ docker images

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
| ubuntu | latest | abc123def456 | 2 weeks ago | 29MB |
| localhost:5000/ubuntu | latest | abc123def456 | 2 weeks ago | 29MB |

This shows that both **ubuntu** and **localhost:5000/ubuntu** reference the same image (same IMAGE ID),

Notice that both ubuntu and localhost:5000/ubuntu share the **same IMAGE ID**, meaning they are the same image, just with different names.

**Why use tagging?**

1. **Preparing for Push** → You need to tag an image before pushing it to a private registry.
2. **Versioning Images** → You can tag an image with different versions (e.g., myapp:v1.0, myapp:v2.0).
3. **Organizing Images** → Tags help manage images in different environments (e.g., myapp:dev, myapp:prod).

**Final Answer:**

Tagging **does not create a new image**; it simply **creates another reference (alias)** for an existing image.

Now to pull the image from the local registry:

#### $ docker pull localhost:5000/ubuntu