**¿Qué es docker?**

Una razón por la que Docker es tan popular es que cumple la promesa de “desarrollar una vez, ejecutar en cualquier lugar”. Docker ofrece una forma sencilla de empaquetar una aplicación y sus dependencias de tiempo de ejecución en un solo contenedor; también proporciona una abstracción en tiempo de ejecución que permite que el contenedor se ejecute en diferentes versiones del kernel de Linux.

Usando Docker, un desarrollador puede crear una aplicación en contenedor en su estación de trabajo, y luego implementar fácilmente el contenedor en cualquier servidor habilitado para Docker. No es necesario volver a probar o volver a sintonizar el contenedor para el entorno del servidor, ya sea en la nube o en las instalaciones

**Problemas del Desarrollo de software profesional**

**Problemas al construir:**

* Dependencias de desarrollo (paquetes)
* Versiones de tiempo de ejecución
* Equivalencia de los entornos de desarrollo (código compartido)
* Equivalencia de entornos de producción (pasar a producción)
* Versiones / compatibilidad (integración de otros servicios, por ejemplo: bases de datos)

**Problemas en la distribución:**

* Diferentes generaciones de compilación
* Acceso a los servidores de producción
* Ejecución nativa vs. Distribuida
* Sin servidor

**Problemas al ejecutar:**

* Dependencias de la aplicación
* Compatibilidad del sistema operativo
* Disponibilidad de servicios externos
* Recursos de hardware

**Docker permite:**

Construir, distribuir y ejecutar su código en cualquier lugar sin preocuparse.

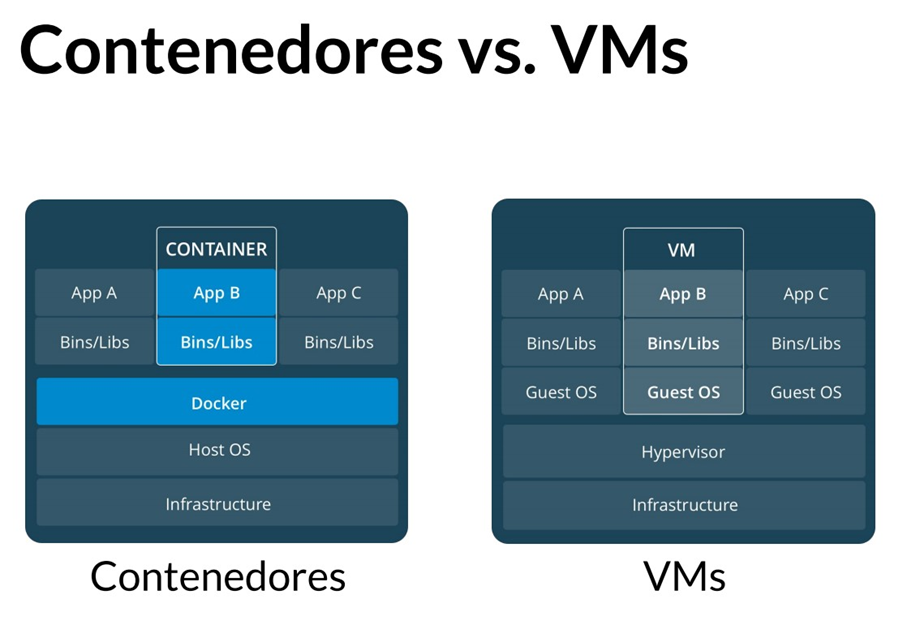
**Contenedores**

Un contenedor la unidad lógica más importante de docker que permite encapsular las dependencias de un proyecto en un entorno aislado, con esto puedes conseguir resolver el problema de “But it works in my machine” ya que te permite crear una especie de máquina virtual y pasársela a tus compañeros o colocarlo en un servidor para el despliegue de manera fácil.

¡✋Alerta! Puedes ver a un contendor como una máquina virtual, pero eso no significa que lo sea, una máquina virtual puede llegar a ser muy similar debido a sus funcionalidades como el aislamiento de procesos.

Entonces… ¿cuál es el beneficio de usar contenedores en lugar de máquinas virtuales?

El mayor beneficio es que los contenedores de Docker están en el nivel de los MB eso nos da ventajas en el consumo de recursos, ya que estos corren compartiendo el host del kernel de Linux, por otro lado las máquinas virtuales son un sistema operativo (O.S) con sus propias apps que corre sobre el tuyo usando virtualización y que consume muchos recursos en el nivel de los GB. Una mejor forma de ver esto es observando la arquitectura de los contendores y de las máquinas virtuales.



**Componentes DENTRO del círculo de Docker:**

Docker daemon: Es el centro de docker, el corazón que gracias a el, podemos comunicarnos con los servicios de docker.

REST API: Como cualquier otra API, es la que nos permite visualizar docker de forma “gráfica”.

Cliente de docker: Gracias a este componente, podemos comunicarnos con el corazón de docker (Docker Daemon) que por defecto es la línea de comandos.

Dentro de la arquitectura de Docker encontramos:

Contenedores: Es la razón de ser de Docker, es donde podemos encapsular nuestras imágenes para llevarlas a otra computadora, o servidor, etc.

Imágenes: Son las encapsulaciones de x contenedor. Podemos correr nuestra aplicación en Java por medio de una imagen, podemos utilizar Ubuntu para correr nuestro proyecto, etc.

Volúmenes de datos: Podemos acceder con seguridad al sistema de archivos de nuestra máquina.

Redes: Son las que permiten la comunicación entre contenedores.

**Conceptos básicos e importantes sobre Docker**

**¿Que es un contenedor?**

Es una agrupación de procesos.

Es una entidad lógica, no tiene el límite estricto de las máquinas virtuales, emulación del sistema operativo simulado por otra más abajo.

Ejecuta sus procesos de forma nativa.

Los procesos que se ejecutan adentro de los contenedores ven su universo como el contenedor lo define, no pueden ver más allá del contenedor, a pesar de estar corriendo en una maquina más grande.

No tienen forma de consumir más recursos que los que se les permite. Si esta restringido en memoria ram por ejemplo, es la única que pueden usar.

A fines prácticos los podemos imaginar cómo maquinas virtuales, pero NO lo son. Máquinas virtuales livianas.

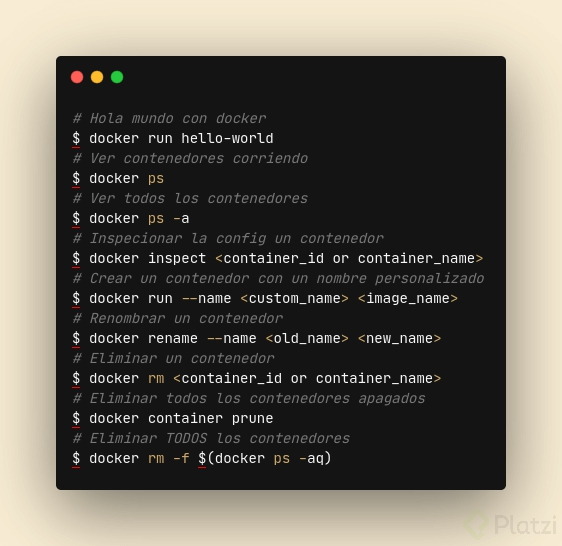
Docker corre de forma nativa solo en Linux.

Sector del disco: Cuando un contenedor es ejecutado, el daemon de docker le dice, a partir de acá para arriba este disco es tuyo, pero no puedes subir mas arriba.

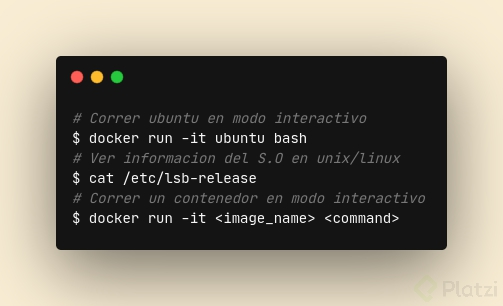
Docker hace que los procesos adentro de un contenedor este aislados del resto del sistema, no le permite ver más allá.

Cada contenedor tiene un ID único, también tiene un nombre.

**Comandos Útiles en Docker**



**Comandos para correr Linux con Docker**



**Nota**

**Cada vez que un contendor se ejecuta, en realidad lo que ejecuta es un proceso del sistema operativo. Este proceso se le conoce como Main process.**

**Main process**

**Determina la vida del contenedor, un contendor corre siempre y cuando su proceso principal este corriendo.**

**Sub process**

**Un contenedor puede tener o lanzar procesos alternos al main process, si estos fallan el contenedor va a seguir encedido a menos que falle el main.**

**Ejemplos manejados en el video**

**Batch como Main process**

**Agujero negro (/dev/null) como Main process**

**docker run --name alwaysup -d ubuntu tail -f /dev/null**

**\_el ouput que te regresa es el ID del contentedor \_**

**Te puedes conectar al contenedor y hacer cosas dentro del él con el siguiente comando (subproceso)**

**docker exec -it alwaysup bash**

**Se puede matar un Main process desde afuera del contenedor, esto se logra conociendo el id del proceso principal del contenedor que se tiene en la máquina. Para saberlo se ejecuta los siguientes comandos;**

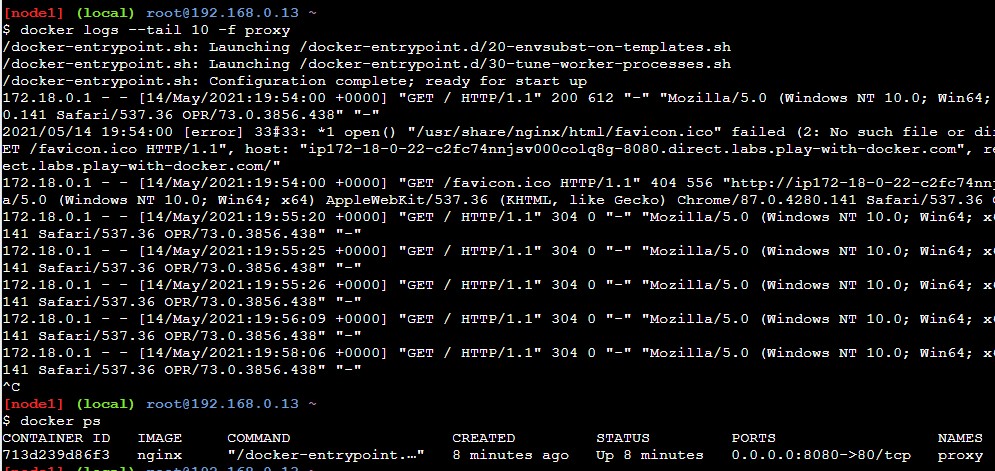
**docker inspect --format '{{.State.Pid}}' alwaysup**

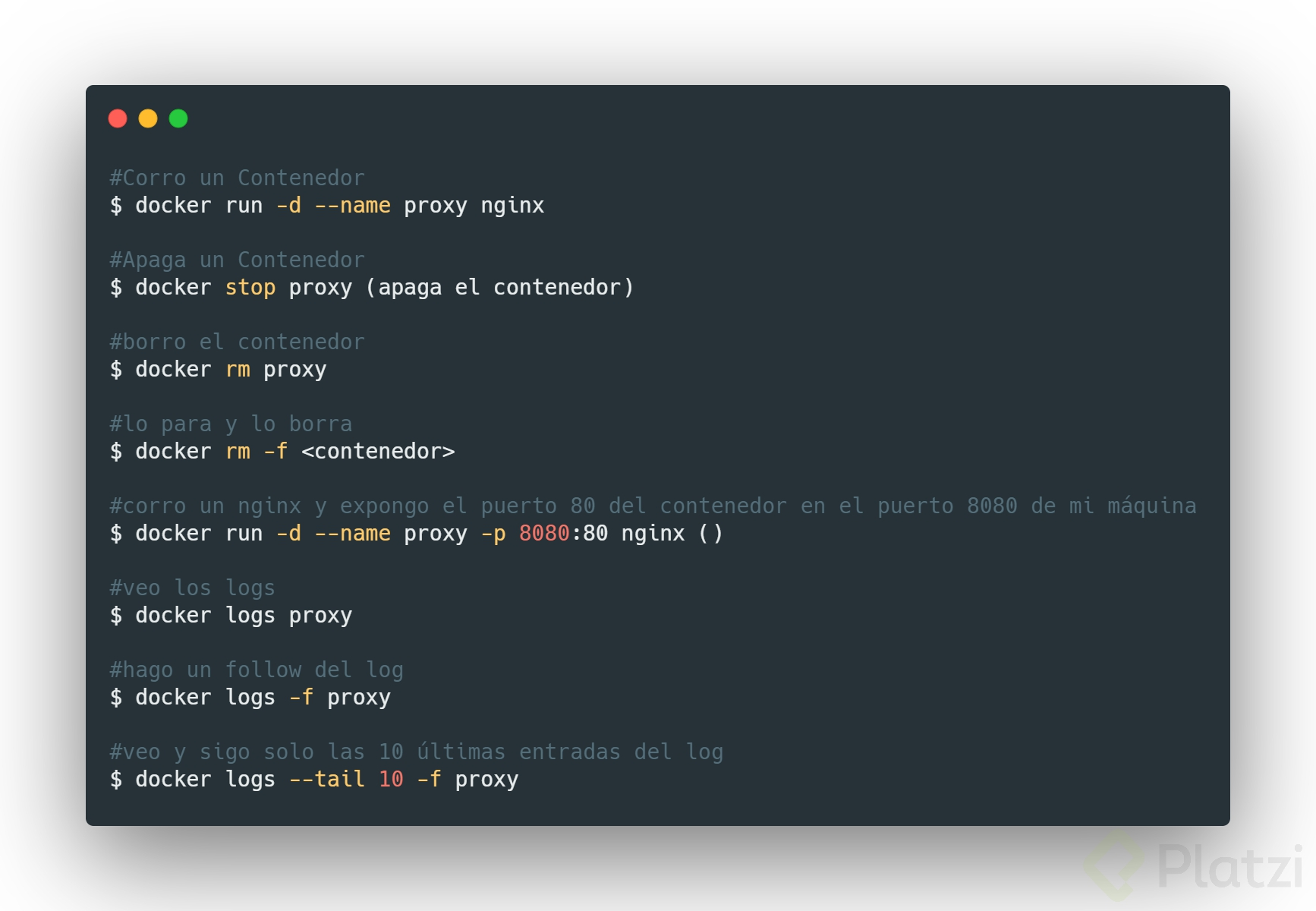
**\_El output del comando es el process ID (2474) \_**

**Para matar el proceso principal del contenedor desde afuera se ejecuta el siguiente comando (solo funciona en linux)**

**Kill -9 2474**

**Exponiendo Contenedores**

****



**Manejo de Los Datos en Docker**

Los contenedores son autocontenidos y no pueden acceder a la información de la máquina que los hostea.

A veces tenemos aplicaciones que generan datos y que queremos acceder a ellos.

Crearemos una carpeta nueva para que interactúe con nuestro contenedor

$ mkdir dockerdata

Vamos a crear una base de datos en mongo db y hacer que los datos no mueran con el contenedor cuando se cierre

$ docker run -d --name db mongo

Ahora entraremos al contenedor para generar datos en él:

$ docker exec -it db bash

Ahora estoy dentro del contenedor y accederé a mongo:

$ mongo

ya estoy conectado en la base de datos local y crearé una base de dato llamada “Platzi”

use platzi

crearé un nuevo usuario llamado “guido”

db.users.insert({“nombre”:“guido”})

verifico que esté creado:

db.users.find()

Saldré de mongo y del contenedor con el comando exit dos veces

y eliminaré el contenedor por completo con docker rm -f db

Pero al volver de acceder no estará disponible, dado que eliminé el contenedor.

Para que no se elimine el dato, necesito generar un directorio espejo en mi máquina local, para esto, utilizaremos bind mounts.

Lo primero que necesitamos es crear un nuevo directorio

mkdir mongodata

y obtendremos el path en el que está montado:

pwd

Y copiaremos en el portapapeles la ruta del directorio:

docker run -d --name db -v /miruta/mongodata:/data/db mongo

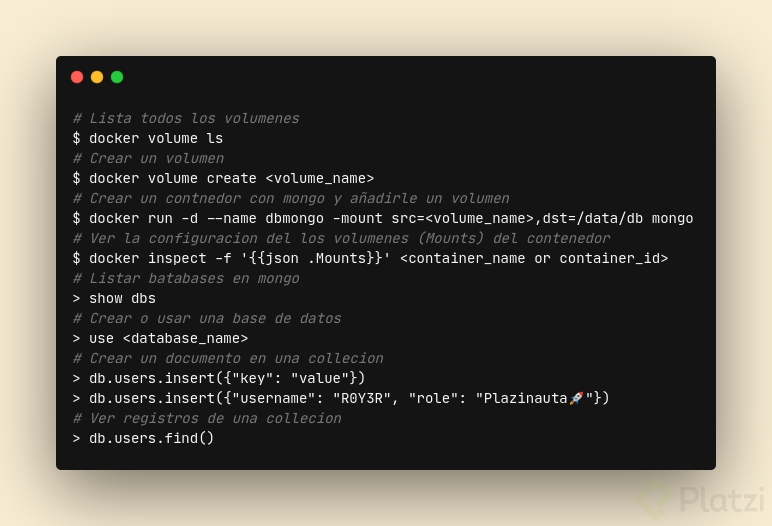
donde /miruta/mongodata es la ruta a mi carpeta local y /data/db es la carpeta donde mongo guarda sus datos

ahora, si repetimos todo el procedimiento anterior, todo quedará espejado en el directorio local, por lo tanto, el dato que ingresamos en la base de datos está disponibles en mongo

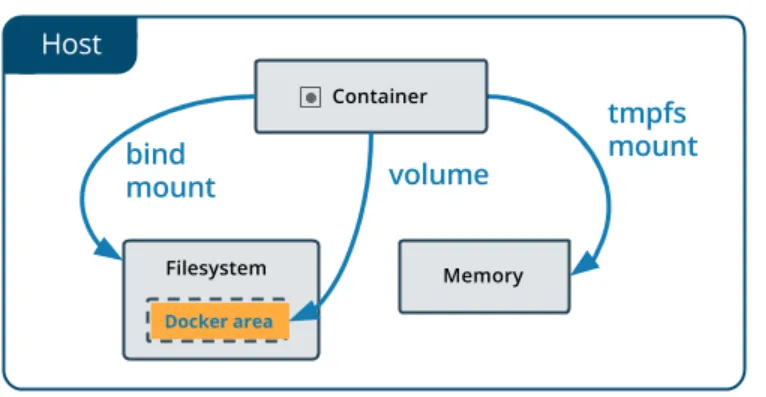
**¿Qué son los volume?**

Los volúmenes son el mecanismo preferido para conservar los datos generados y utilizados por los contenedores de Docker. Si bien los montajes de enlace dependen de la estructura del directorio y el sistema operativo de la máquina host, Docker administra completamente los volúmenes. Los volúmenes tienen varias ventajas sobre los montajes vinculantes:

**Comandos**



**Insertar y extraer archivos de un contenedor**



**Host: Donde Docker esta instalado.**

**Bind Mount: Guarda los archivos en la maquina local persistiendo y visualizando estos datos (No seguro).**

**Volume: Guarda los archivos en el area de Docker donde Docker los administra (Seguro).**

**TMPFS Mount: Guarda los archivos temporalmente y persiste los datos en la memoria del contenedor, cuando muera sus datos mueren con el contenedor.**

**Comandos**

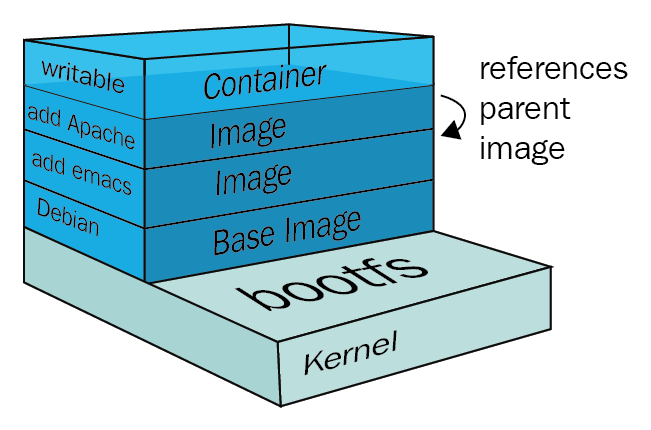


**Concepto fundamental de Docker: imágenes**

**Una imagen contiene distintas capas de datos (distribución, diferente software, librerías y personalización).**

**Podemos llegar a la conclusión, que una imagen se conforma de distintas capas de personalización, en base a una capa inicial (base image), la dicha capa, es el más puro estado del SO.**

**La siguiente ilustración nos mostraría la representación gráfica, del concepto de una imágen en Docker.**



**Si observamos, partimos desde la base del SO, y vamos agregando capas de personalización hasta obtener la imagen que necesitamos:**

* **distribución debian**
* **se agrega el editor Emacs**
* **se agrega el servidor Apache**
* **se agregan los permisos de escritura para la carpeta /var/www de Apache**

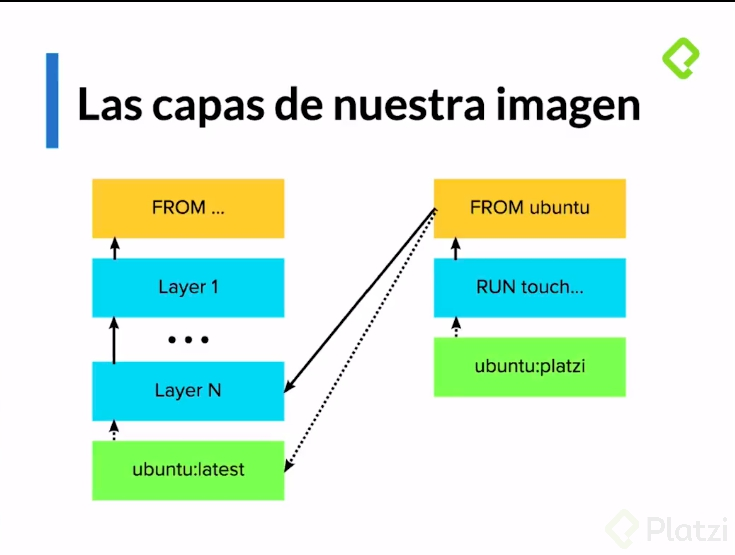
**Comandos**



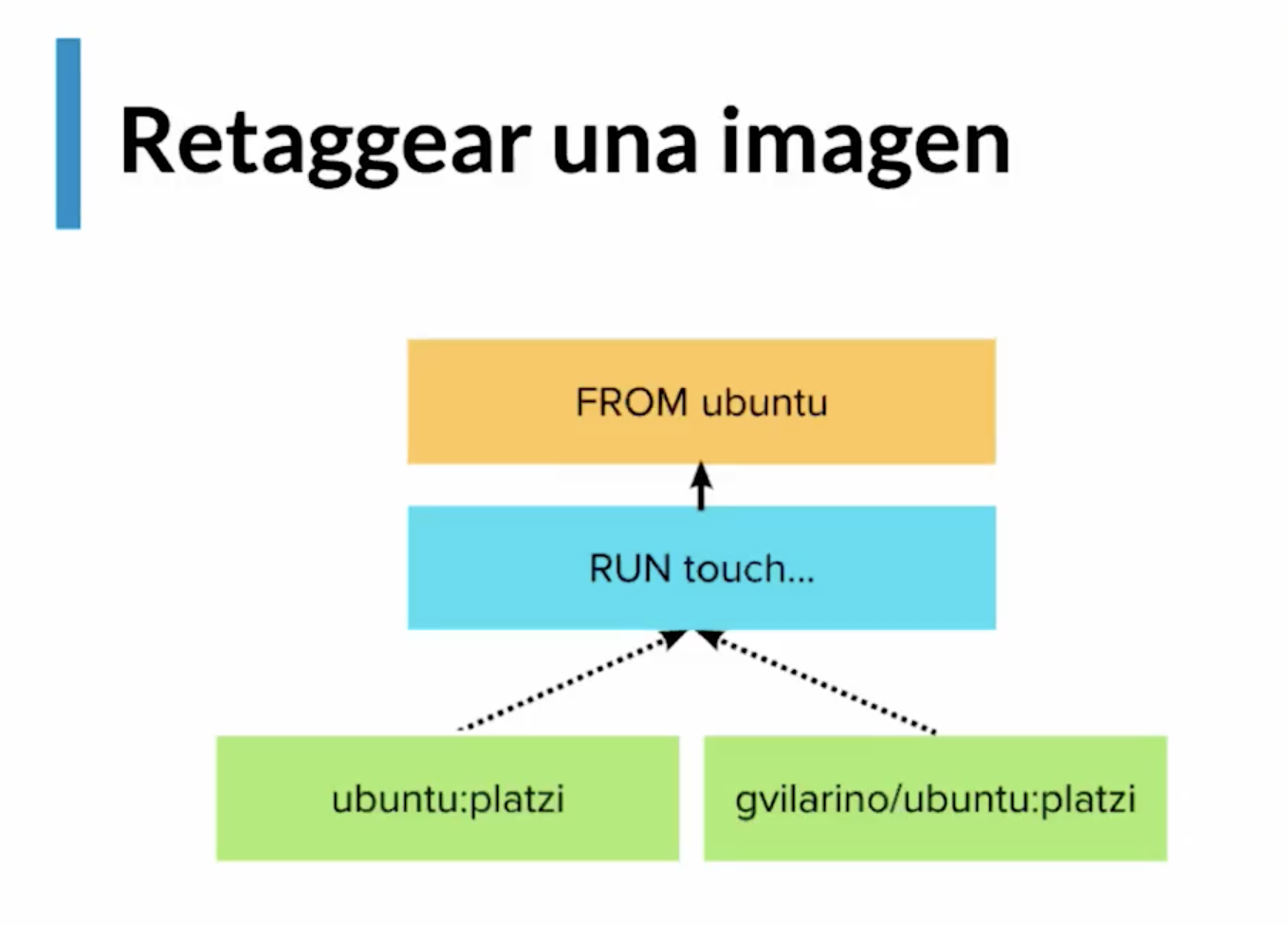
**Construyendo una imagen propia**

Las imágenes son un conjunto de capas y cada capa nueva es una diferencia con la capa anterior

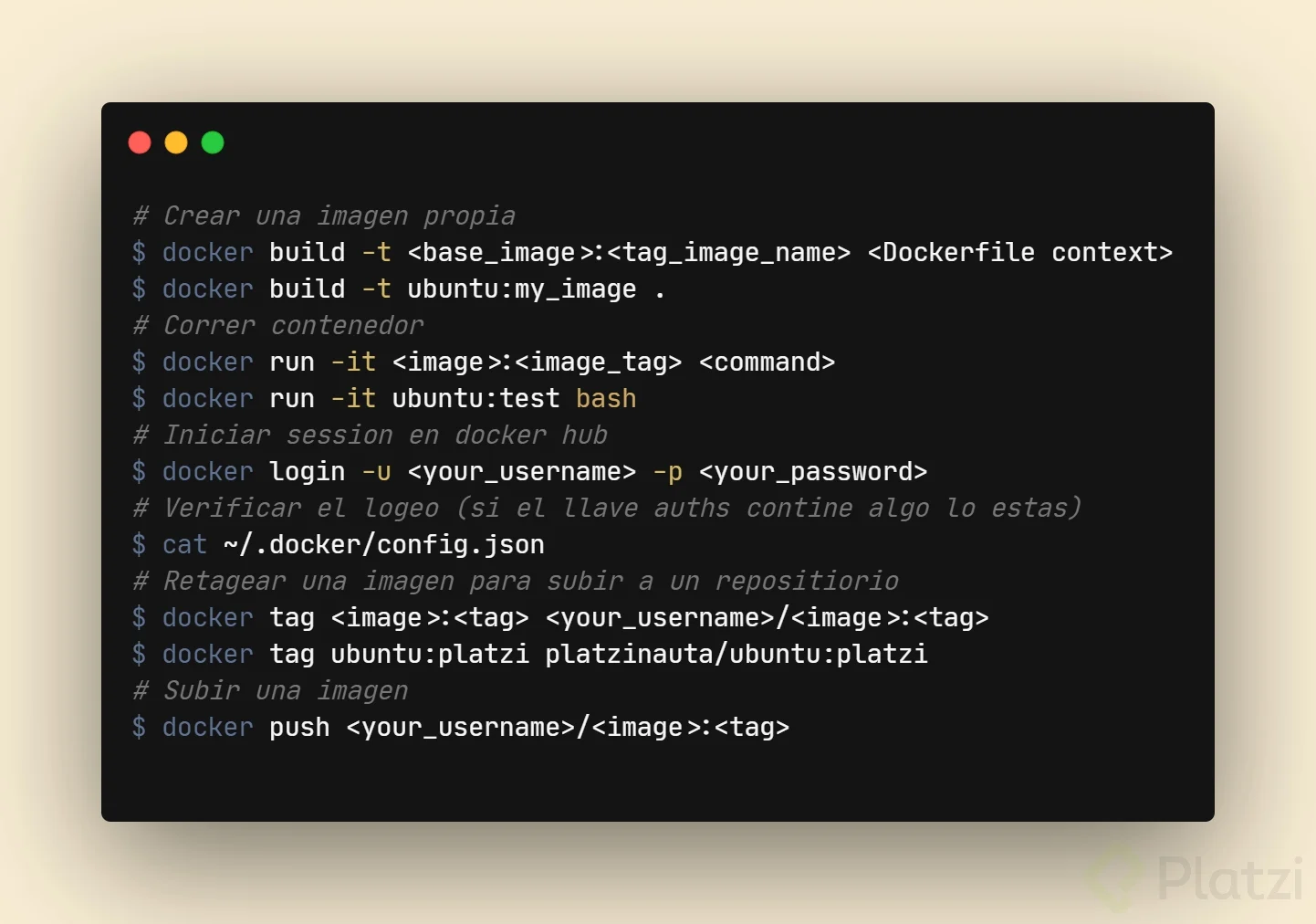
**Capas de nuestra imagen**



**Retaggear una imagen**



**Comandos**



**El sistema de capas**

Las imágenes son un conjunto de capas, y a partir del Dockerfile se puede saber como está construido una imagen. Existen distintas maneras de ver las capas que conforman una imagen:

Si la imagen es pública se puede visitar Dockerhub y buscar una imagen para ver su Dockerfile. Una vez encontrar la imagen buscamos en los tag y haciendo clic en los tag se mostrará el docker file para construir dicha imagen.

Se puede hacer a través de la línea, esta opción no es muy cómoda. Es importante destaca que los cambios se presenta por filas y las filas más inferiores son los cambios más viejos y las que están más arriba son los cambios más recientes. el comando es:

$ sudo docker history <nombre imagen: nombre tag>

Ejemplo:

$ sudo docker history ubuntu:platzi