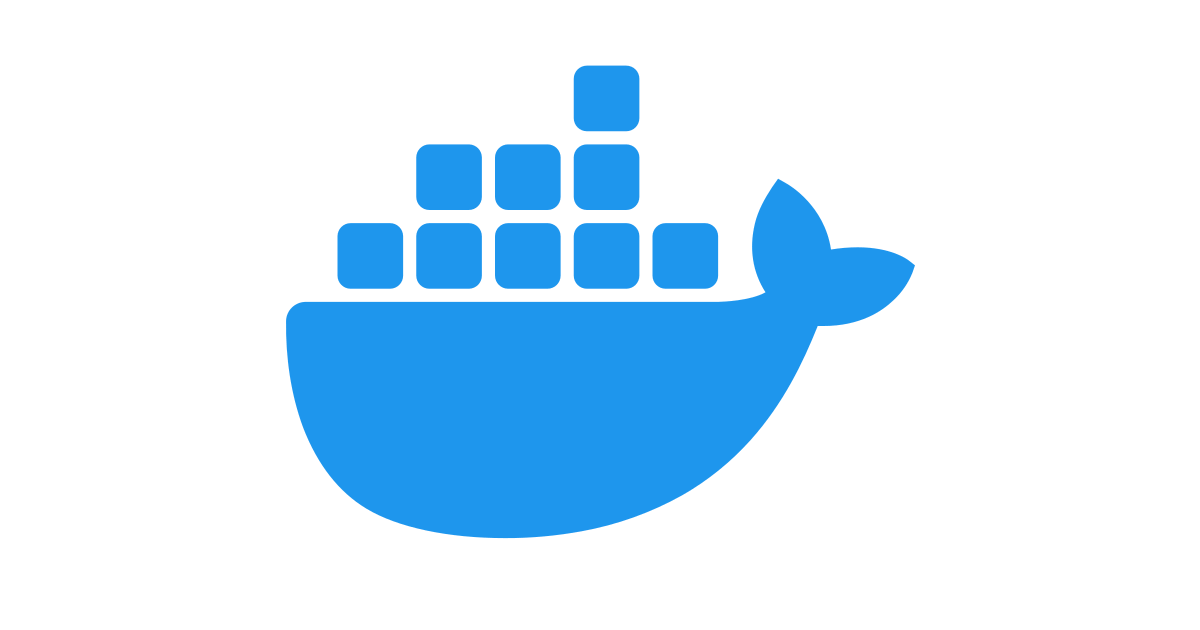
🐳 APUNTES DE DOCKER



***APUNTES DE DOCKER***

**Descargar una imagen publicada en Docker-Hub:**

**docker pull [nombre Imagen o Tag]** **🡪** Normal

**docker pull [nombre Imagen o Tag]:latest 🡪** Última Versión

**docker pull [nombre Imagen o Tag]:3.7** **🡪** Versión específica

**docker search [nombre Imagen]** **🡪** Busca resultados acerca de la imagen impuesta.

**docker container** **🡪** Muestra una lista de todos los subcomandos disponibles y sus opciones para gestionar contenedores en Docker.

**docker image** **🡪** Muestra las imágenes presentes en Docker Desktop.

**1- CREACIÓN Y EJECUCIÓN DE CONTENEDORES**

**docker --help** **🡪** **AYUDA**

**Listar contenedores (y sus datos):**

**docker ps 🡪** En ejecución

**docker ps -a** **🡪** Detenidos y en ejecución

**docker run [nombre Imagen]** **🡪** Crea y ejecuta un contenedor basado en la imagen impuesta.

// Si por ejemplo utilizáramos el siguiente comando: **docker run alpine**, se creará y ejecutará un contenedor basado en la imagen alpine.

También podemos ejecutar comandos en un contenedor a la vez que lo arrancamos de la siguiente manera:

**docker run [nombre Imagen] [comando que se quiera ejecutar en el contenedor]**

// Si por ejemplo utilizáramos: **docker run alpine ls**, se creará y ejecutará un contenedor basado en la imagen alpine y seguido se listarán todos los archivos y directorios que se encuentren en su interior (eso es lo que hace el comando **“ls”**).

**2- CONTENEDORES EN MODO INTERACTIVO**

**docker run -it --name alpinec alpine**

Es básicamente: **docker run -it --name [nombre Contenedor] [nombre Imagen]**

**-i** **🡪** Este parámetro nos permite interaccionar con el contenedor a través de la entrada estándar STDIN.

**-t** **🡪** Este parámetro nos asigna un terminal dentro del contenedor.

**-it** **🡪** Los dos parámetros nos permiten usar un contenedor como si fuese una máquina virtual tradicional.

**--name** **🡪** Este parámetro nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre, Docker nos asignará uno automáticamente.

Una vez ejecutado el comando anterior nos aparece un prompt (el terminal del contenedor) para poder interaccionar con el contenedor que acabamos de crear.

# ***2.1 -* Eliminar un contenedor**

**docker rm [nombre Contenedor o ID Contenedor]** **🡪** Elimina un contenedor QUE ESTÁ DETENIDO.

**docker rm [nombre Contenedor o ID Contenedor]** **-f 🡪** Elimina un contenedor QUE ESTÁ EN EJECUCIÓN.

**docker rm (docker ps -aq) 🡪** Elimina TODOS los contenedores QUE ESTÁN DETENIDOS.

**docker rm -f (docker ps -aq) 🡪** Elimina TODOS los contenedores (EN EJECUCIÓN Y DETENIDOS).

**-f** **🡪** Este comando fuerza el borrado de contenedores en ejecución.

# ***2.2 -* Salir de un contenedor**

Para salir del contenedor y detenerlo:

* Escribir **exit**
* **Ctrl** + **C**

Para salir del contenedor SIN detenerlo:

* **Ctrl** + **P** + **Q**

**--rm** **🡪** Este parámetro hace que cuando se detenga el contenedor o nos salgamos de este, se elimine automáticamente.

# ***2.3 -* attach y exec**

**docker attach [nombre Contenedor o ID Contenedor] 🡪** Permite acceder al terminal de un contenedor QUE ESTÁ EN EJECUCIÓN indicando su nombre o su ID.

Hay que tener en cuenta que no crea un nuevo terminal, sino que usa el terminal original que ha estado en ejecución, de modo que si salimos del terminal con **exit** el contenedor se detendrá.

**-d** **🡪** Este parámetro hace que cuando ejecutemos un contenedor lo haga en modo *“detached”* o *“background”*, o sea, nos permite ejecutar un contenedor con la terminal en segundo plano (no nos sale la terminal en la cara como antes).

**-dit** **🡪** Estos parámetros permiten ejecutar un contenedor en modo “detached” y que podamos interaccionar con él a través de un terminal.

**-b** **🡪** Este parámetro se escribe TRAS UN COMANDO, y permite que este se ejecute en segundo plano (modo *“batch”*).

Al contrario que si lo hiciéramos en primer plano, si no utilizáramos el modo batch el contenedor se detendría una vez que finaliza la ejecución del comando.

// El siguiente comando:

**docker run -dit --rm --name topdemo ubuntu /usr/bin/top -b**

Crea y ejecuta un contenedor llamado *“topdemo”* (basado en la imagen de ubuntu) en segundo plano (por lo que no nos saca la terminal en la cara al ejecutarlo) y con el cual podemos interactuar mediante dicha terminal. Al mismo tiempo ejecuta el comando ***“/usr/bin/top”*** (un simple comando de ruta)en segundo plano (modo *“batch”*, por lo que no lo vemos).

**docker exec -it [nombre Contenedor o ID Contenedor] bin/sh 🡪** Crea un nuevo terminal a un contenedor QUE ESTÁ EN EJECUCIÓN indicando su nombre o su ID, machacando el anterior.

Una vez ejecutado el comando anterior nos aparece un prompt para poder interactuar con el contenedor que acabamos de crear.

# ***2.4 -* start y stop**

**docker start [nombre Contenedor] [nombre Contenedor] 🡪** Ejecuta uno o más contenedores detenidos.

**docker stop [nombre Contenedor] [nombre Contenedor] 🡪** Detiene uno o más contenedores en ejecución.

# ***2.5 -* logs e inspect**

**docker logs [nombre Contenedor] 🡪** Permite ver los registros de salida del contenedor.

**docker logs -f [nombre Contenedor] 🡪** Permite ver los registros de salida del contenedor y se irán actualizando automáticamente.

**docker inspect [nombre Contenedor] 🡪** Permite obtener información sobre el contenedor nombrado.

El comando ***“docker inspect”*** nos permite inspeccionar objetos Docker como:

contenedores, imágenes y redes.

# ***2.6 -* Exponer puertos**

Consiste en reservar un puerto del servidor de Docker con el objetivo de redirigir las peticiones de un contenedor a un puerto local específico.

Se puede hacer de 2 maneras:

**-p [puerto Contenedor] 🡪** Indica el puerto al que se va a conectar un contenedor.

**-p [puerto Local]:[puerto Contenedor] 🡪** Indica el puerto local al que se van a redirigir las peticiones del puerto al que se va a conectar el contenedor.

\\ En el primer caso, como no se indica el puerto local al que queremos redirigir el puerto de contenedor, se asigna uno aleatorio.

\\ En el segundo caso, si el puerto local que indicamos ya está en uso, obtendremos un error y el contenedor no se creará.

Comprobamos que podemos acceder al contenido del contenedor de 2 maneras:

* accediendo a la URL **http://localhost:[puerto Local]** en el buscador.
* Con el comando **curl http://localhost:[puerto Local]**

**LO HAS DEJADO EN LA PÁGINA 47**

**APUNTES DE DOCKERFILE**

**crear una imagen**

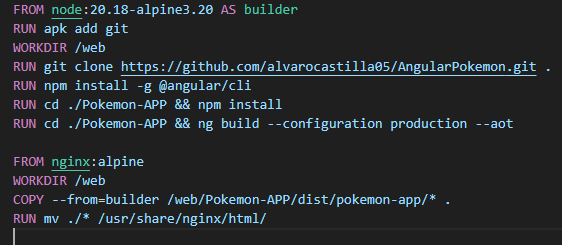
**docker build -t [rutaImagen:versionImagen] . 🡪** Crea una imagen a partir de un archivo dockerfile. El -t es para escribir a continuación el tag con la ruta y la versión (si no se pone la versión se machacará con lastest todo el rato en vez de guardar sus versiones).

**docker logs -f [nombre Contenedor] 🡪** Permite ver los registros de salida del contenedor y se irán actualizando automáticamente.

**docker inspect [nombre Contenedor] 🡪** Permite obtener información sobre el contenedor nombrado.

****

**ng build --configuration production --aot 🡪**

****

**\* LISTA DE PARÁMETROS**

**-i** **🡪** Este parámetro nos permite interaccionar con el contenedor a través de la entrada estándar STDIN.

**-t** **🡪** Este parámetro nos asigna un terminal dentro del contenedor.

**-it** **🡪** Los dos parámetros nos permiten usar un contenedor como si fuese una máquina virtual tradicional.

**--name** **🡪** Este parámetro nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre, Docker nos asignará uno automáticamente.

**--rm** **🡪** Este parámetro hace que cuando se detenga el contenedor o nos salgamos de este, se elimine automáticamente.

**-d** **🡪** Este parámetro hace que cuando ejecutemos un contenedor lo haga en modo *“detached”* o *“background”*, o sea, nos permite ejecutar un contenedor con la terminal en segundo plano (no nos sale la terminal en la cara como antes).

**-dit** **🡪** Estos parámetros permiten ejecutar un contenedor en modo “detached” y que podamos interaccionar con él a través de un terminal.

**-b** **🡪** Este parámetro se escribe TRAS UN COMANDO, y permite que este se ejecute en segundo plano (modo *“batch”*).

**-f** **🡪** Este comando fuerza el borrado de contenedores en ejecución.

**COMANDOS —————————————————**

**CREAR UN CONTENEDOR EN MODO INTERACTIVO (CREAR NUEVA TERMINAL):**

docker run -it --name nombre imagen-

**DESCARGAR UNA IMAGEN PUBLICADA EN DOCKER-HUB:**

docker pull-nombreDeLaImagenOTag- **→** Normal

docker pull-nombreDeLaImagenOTag:latest- **→** Última Versión

docker pull-nombreDeLaImagenOTag:3.7- **→** Versión específica

**-AYUDA:-**

docker --help-

docker comandoDelQueQuierasAyuda --help-

**BUSCA IMÁGENES EN DOCKER-HUB:**

-docker search-palabraRelacionadaConLaBúsqueda-

**LISTAR CONTENEDORES (Y SUS DATOS):**

-docker ps- .**→** En ejecución

-docker ps -a- **→** Detenidos

**BORRAR CONTENEDORES:**

-docker rm nombreContendor- **→** Uno

-docker rm nombreCont1 nombreConte2- **→** Varios

**CONECTAR UN PUERTO LOCAL : PUERTO CONTENEDOR:**

docker run -it --name nombre -p 81:80 imagen- **→**  Puerto concreto

docker run -it --name nombre -p imagen- **→**  Elige cualquier puerto libre

echo “<h1>Hola Mundo</h1>” >> index.html

-e NOMBREVARIABLEDEENTORNO\_ROOT\_PASSWORD=root

**MUESTRA UNA LISTA DE TODOS LOS SUBCOMANDOS DISPONIBLES Y SUS OPCIONES PARA GESTIONAR CONTENEDORES EN DOCKER:**

-docker container-

**MUESTRA LAS IMÁGENES PRESENTES EN DOCKER DESKTOP:**

-docker image-

**EJECUTA UN CONTENEDOR CON EL NOMBRE “alpinec” USANDO LA IMAGEN “alpine”:**

-docker run -it --name-alpinec alpine-

docker run -dit --rm --name topdemo ubuntu /usr/bin/top -b

NOMBRE IMAGEN

docker attach topdemo

docker run -d --rm -p 81:80 --name httpdc httpd

VINCULAR EL PUERTO DE ORD AL DEL CONTENEDOR -> -p ORD/CONT

docker run -d --rm -P --name httpdc httpd

**PARA LA EJECUCIÓN DE UN CONTENEDOR:**

-docker stop-nombreDelContenedor-

docker stop $(docker ps -q)

**SALIR (DENTRO DE UN TERMINAL DE MV):**

-exit-

**——————————————————————————————————**

**— INFORMACIÓN ÚTIL —————————————————————**

**DOCKER CONTAINER:**

A todos los comandos -docker container- se les puede omitir el “container” como:

* -docker container run  **----->** -docker run-
* -docker container ps . **----->** -docker ps-
* -docker container rm  **----->** -docker rm-

**CUANDO UN CONTENEDOR ESTÁ EJECUTÁNDOSE:**

1. Si salimos pulsando **CTRL + C** el contenedor se detendrá y finalizará su ejecución.
2. Si salimos pulsando **CTRL + P + Q** el contenedor seguirá ejecutándose en background.

**¿QUÉ ES DOCKER?**

Docker es una plataforma para que desarrolladores y administradores puedan desarrollar, desplegar y ejecutar aplicaciones en un entorno aislado denominado **contenedor**.

Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute (librerías, código, archivos de configuración, etc).

**¿QUÉ ES UNA MÁQUINA VIRTUAL?**

Es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real.

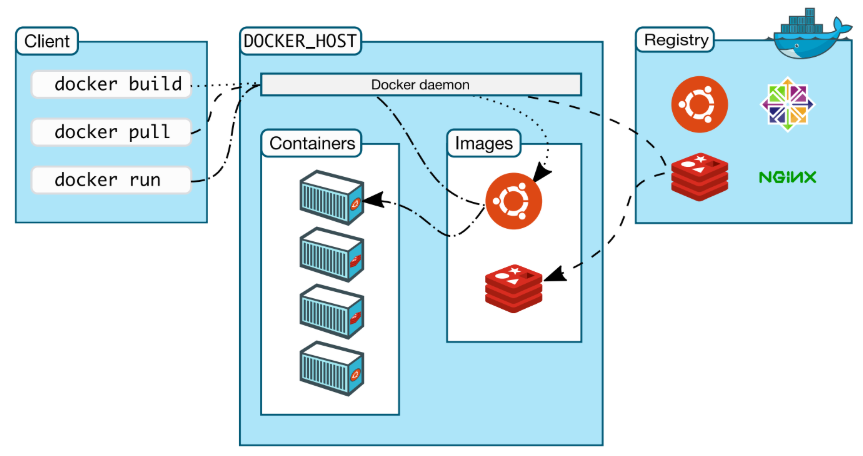
Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellas.

**¿QUÉ ES UN CONTENEDOR?**

Un contenedor es un proceso que ha sido aislado de todos los demás procesos en la máquina anfitriona (máquina host). Ese aislamiento aprovecha características de Linux como los **namespaces del kernel y cgroups.**

Aunque es posible tener más de un proceso en un contenedor las buenas prácticas nos recomiendan ejecutar sólo un proceso por contenedor (PID 1).

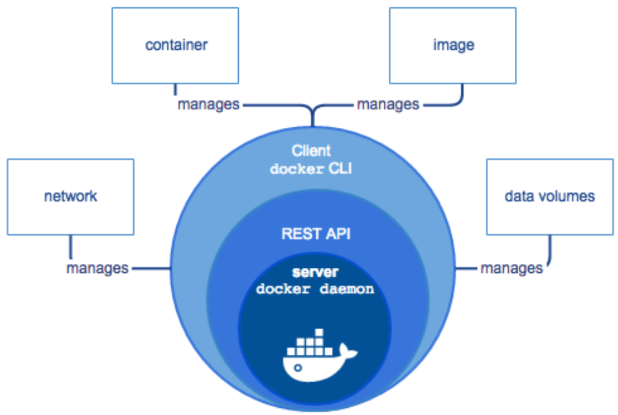
**ARQUITECTURA DE DOCKER**

****

**DOCKER ENGINE**

El Docker Engine es una aplicación cliente-servidor formada por los siguientes componentes:

* Docker daemon
* Docker REST API
* Docker CLI

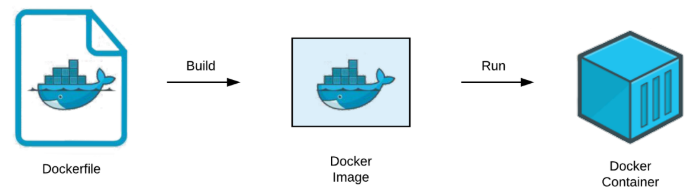


**DOCKERFILES, IMÁGENES Y CONTENEDORES**

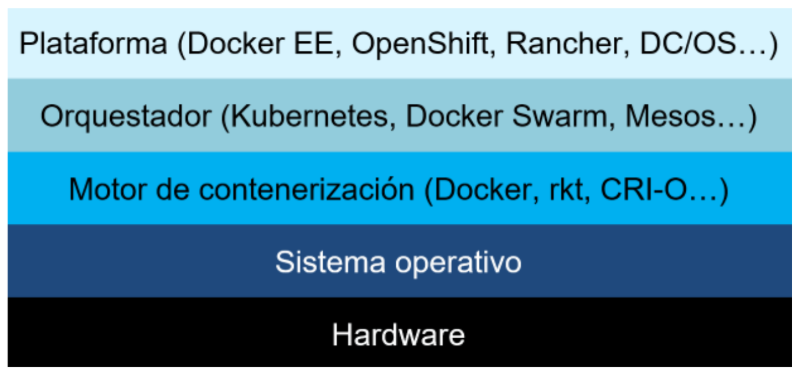
Un **Dockerfile** es un archivo de texto que contiene los comandos necesarios para crear una imagen.

Una **imagen** se crea a partir de un **archivo Dockerfile**. Contienen la unión de sistemas de archivos apilados en capas, donde cada capa representa una modificación de la imagen y equivale a una instrucción en el archivo Dockerfile.

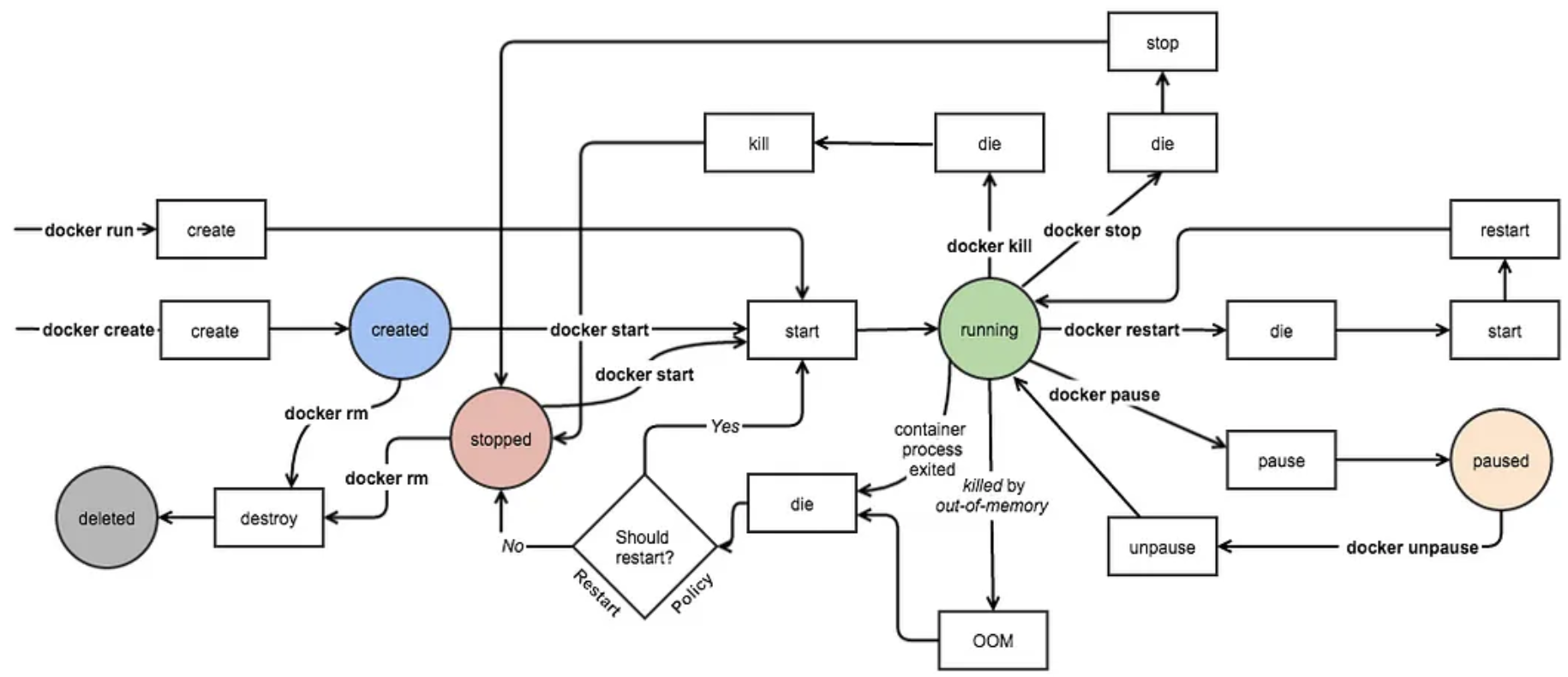
Un **contenedor** es una instancia en ejecución de una imagen.

****

**STACK DE CONTENERIZACIÓN**

****

**CICLO DE VIDA DE UN CONTENEDOR DOCKER**

****

**BUSCAR IMÁGENES EN DOCKER HUB**

**——————————————————————————————————**

**— INSTALACIÓN ————————————————————————**

**PARA COMPROBAR QUE DOCKER ESTÁ INSTALADO CORRECTAMENTE**

$ docker version

**ERRORES COMUNES DESPUÉS DE LA INSTALACIÓN**

Si nos aparece el siguiente error después de la instalación en Linux es porque el usuario con el que estamos ejecutando docker no tiene privilegios de root o no está en el grupo docker.

**$** docker search ubuntu

Got permission denied while trying to connect to the Docker daemon socket at\_\_.….

unix:///var/run/docker.sock: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_…\_.

Get http://%2Fvar%2Frun%2Fdocker.sock/v1.39/images/search?limit=25&term=ubu

ntu: dial unix/var/run/docker.sock: connect: permission denied\_\_\_\_\_\_\_\_\_\_\_\_\_\_..

**CONFIGURACIÓN DEL USUARIO**

El daemon de Docker utiliza un socket Unix y por defecto, el socket Unix es propiedad del usuario **root**, de modo que los demás usuarios sólo pueden acceder a él usando sudo.

Para evitar tener que escribir sudo cada vez que vayamos a ejecutar un comando de docker tenemos que añadir el usuario con el que vamos a trabajar al grupo docker.

$ sudo usermod -aG docker $USER

Para activar los cambios en los grupos sin tener que cerrar la sesión podemos ejecutar lo siguiente.

$ newgrp docker

**——————————————————————————————————**

**COMANDOS DE DOCKER CLI —————————————————**

**COMANDOS COMUNES:**

* run Crea y ejecuta un nuevo contenedor a partir de una imagen
* exec Ejecuta un comando en un contenedor que está ejecutándose
* ps Muestra contenedores
* build Crea una imagen a partir de un archivo Dockerfile
* pull Descarga una imagen de un registro
* push Sube una imagen a un registro
* images Muestra imágenes
* login Autenticarse en un registro
* logout Cerrar sesión en un registro
* search Busca imágenes en Docker Hub
* version Muestra la versión de Docker
* info Muestra la información de todo el sistema

**COMANDOS DE ADMINISTRACIÓN:**

* ai\* Pregunta a Gordon - Agente de Docker
* builder Gestiona la administración de creación (docker build)
* buildx\* Extensión de Docker que proporciona capacidades avanzadas

para crear imágenes de Docker a través de BuildKit

* checkpoint Gestiona los checkpoints
* compose\* Docker Compose
* container Gestiona los contenedores
* context Gestiona contextos
* debug\* Te permite depurar una imagen de un contenedor en ejecución
* desktop\* Comandos de Docker Desktop (Beta)
* dev\* Docker Dev Environments
* extension\* Gestiona las extensiones de Docker
* feedback\* Provide feedback, right in your terminal!
* image Gestiona las imágenes
* init\* Creates Docker-related starter files for your project
* manifest Manage Docker image manifests and manifest lists
* network Manage networks
* plugin Gestiona los plugins
* sbom\* View the packaged-based Software Bill Of Materials (SBOM) for

an image

* scout\* Docker Scout
* system Gestiona Docker
* trust Manage trust on Docker images
* volume Gestiona volúmenes

**COMANDOS DE ENJAMBRE / SWARM:**

* config Manage Swarm configs
* node Manage Swarm nodes
* secret Manage Swarm secrets
* service Manage Swarm services
* stack Manage Swarm stacks
* swarm Manage Swarm

**COMANDOS:**

* 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

* events Get real time events from the server
* export Export a container's filesystem as a tar archive
* history Show the history of an image
* import Import the contents from a tarball to create a filesystem image
* inspect Return low-level information on Docker objects
* kill Kill one or more running containers
* load Load an image from a tar archive or STDIN
* logs Fetch the logs of a container
* pause Pause all processes within one or more containers
* port List port mappings or a specific mapping for the container
* rename Rename a container
* restart Restart one or more containers
* rm Remove one or more containers
* rmi Remove one or more images
* save Save one or more images to a tar archive (streamed to STDOUT

by default)

* 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
* tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE
* 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