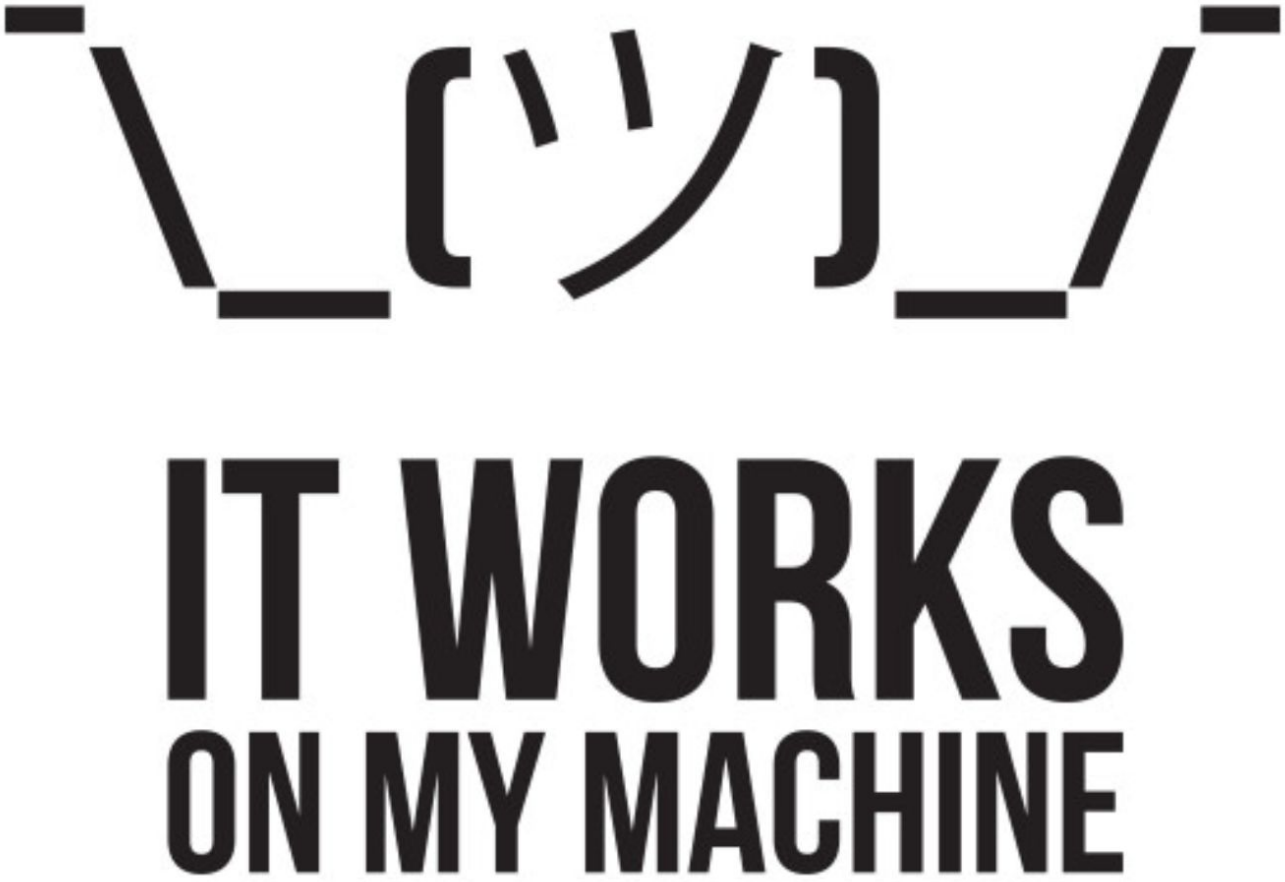
Introducción a Docker

MC. Jesus Humberto Abundis Patiño
jesus.abundis@info.uas.edu.mx

UNIVERSIDAD AUTÓNOMA DE SINALOA
Facultad de Informática Culiacán

¿será cierto?



¿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).

Antecedentes

- Antes de Docker ya existían implementaciones de aislamiento de recursos como:
  - Chroot, en el año 1982.
  - FreeBSD Jails, en el año 2000.
  - Linux Containers (LXC), en el año 2008.
- Docker empezó a ganar popularidad en el año 2013 permitiendo a los desarrolladores crear, ejecutar y escalar rápidamente sus aplicaciones creando contenedores.

Antecedentes

- El uso de contenedores es actualmente uno de los mecanismos más comunes para desplegar software.

- Empresas como Google, Microsoft, Amazon, Oracle, WMware, IBM y RedHat están apostando fuertemente por las tecnologías de contenerización.

- El pasado 13 de noviembre de 2019, la empresa desarrolladora de Docker, Docker Inc, fue adquirida por Mirantis por 35 millones de dólares.

Analogía de contenedores

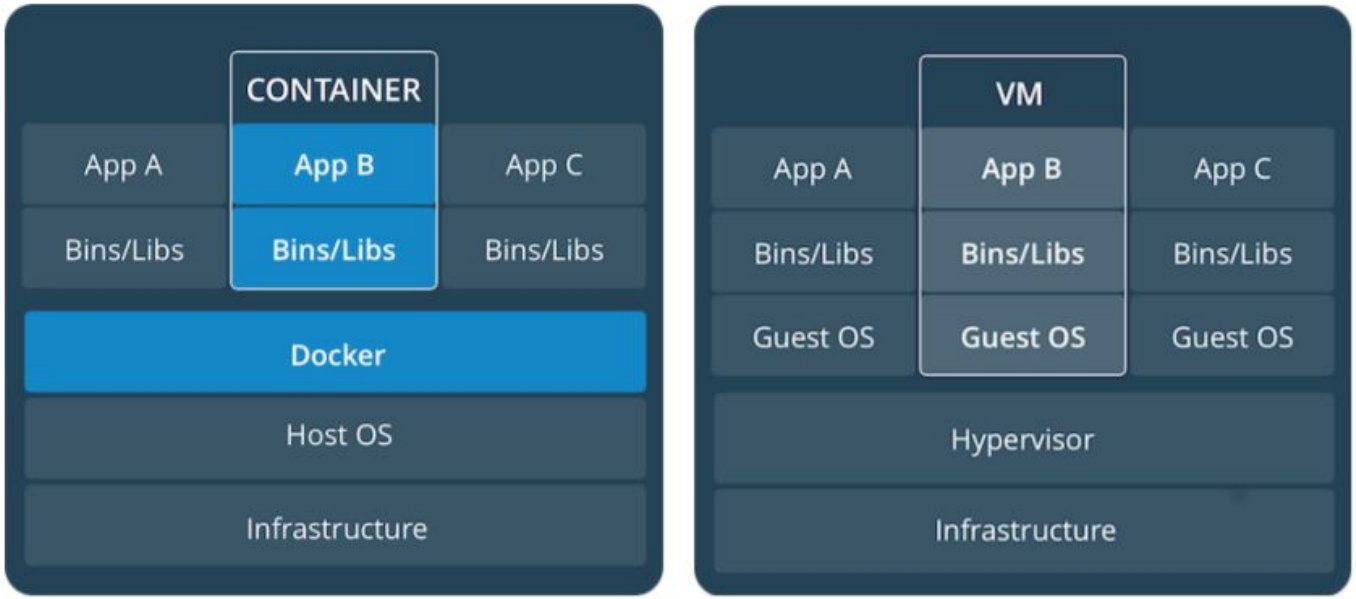
- Los contenedores de transporte marítimo:
  - Cumplen un estándar para enviar mercancías.
  - No nos importa el contenido sino que su forma sea estándar.
  - Pueden ser transportados en cualquier embarcación que cumpla el estándar.
- Los contenedores software:
  - Cumplen un estándar para empaquetar software.
  - No nos importa el contenido sino que su "forma" sea estándar.
  - Pueden ser ejecutados en cualquier servidor que "cumpla el estándar"

docker

Máquinas virtuales vs Contenedores

- 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.
- 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.

Máquinas virtuales vs Contenedores



1. Los contenedores son más ligeros que las máquinas virtuales porque comparten el kernel del host.
2. Con el mismo hardware, es posible tener un mayor número de contenedores que de máquinas virtuales.
3. Los contenedores se pueden ejecutar en hosts que sean máquinas virtuales.

Ventajas del uso de Contenedores

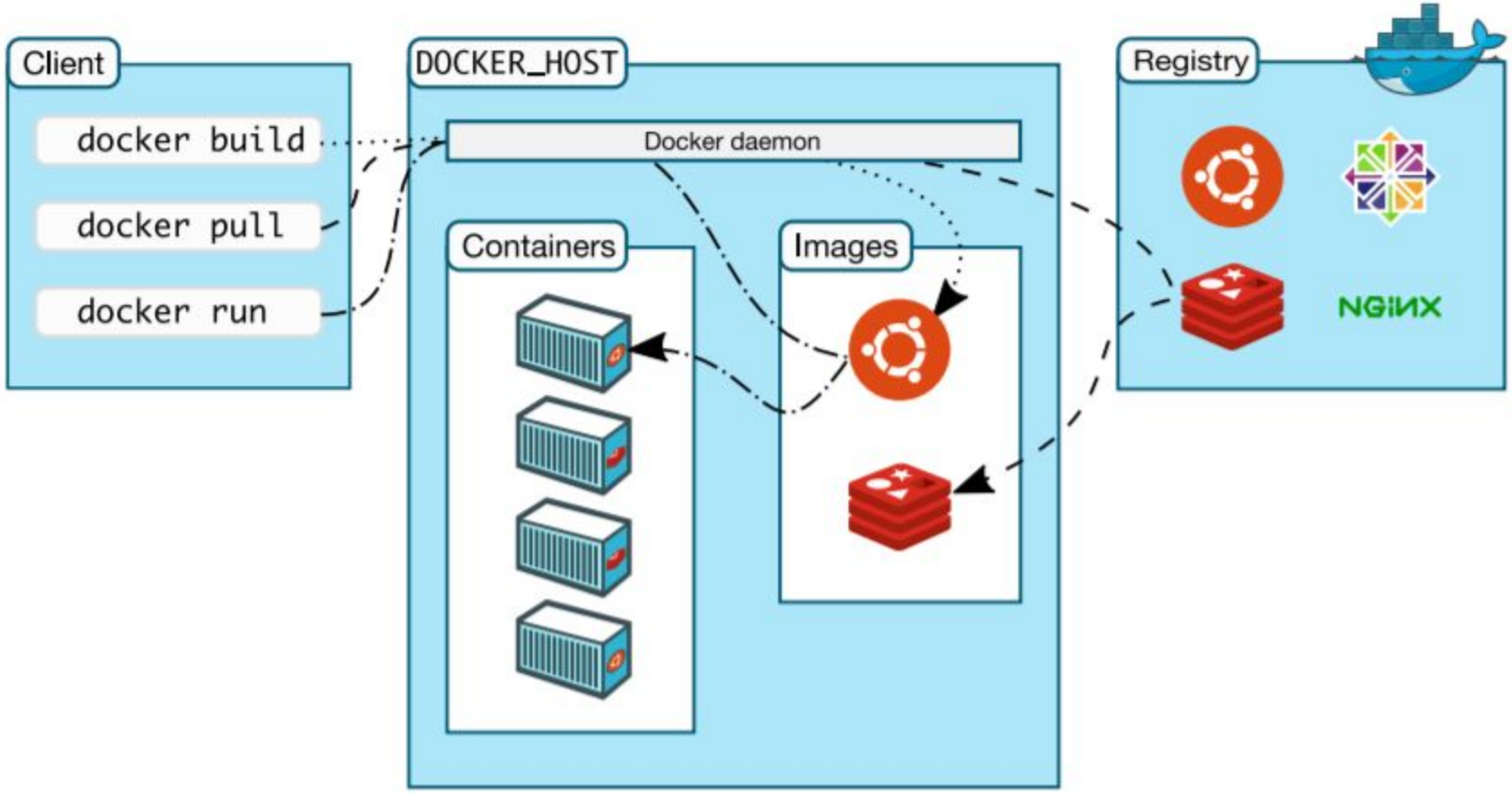
Ventajas para los desarrolladores

1. Soluciona el problema "It works on my machine".
2. Permite tener un entorno de desarrollo limpio, seguro y portátil.
3. Permite la automatización de pruebas, integración y empaquetado.
4. Permite empaquetar una aplicación con todas las dependencias que necesita (código fuente, librerías, configuración, etc.) para ser ejecutada en cualquier plataforma.

Ventajas para administradores

1. Se eliminan inconsistencias entre los entornos de desarrollo, pruebas y producción.
2. El proceso de despliegue es rápido y repetible.

Arquitectura de Docker



Arquitectura de Docker. Imagen de Docker.com

Conceptos

Docker Daemon: Es el servicio en segundo plano que se ejecuta en el host que gestiona la construcción, ejecución y distribución de contenedores Docker.

Docker Client: la herramienta de línea de comandos que permite al usuario interactuar con el demonio.

Docker Hub: Es un repositorio de imágenes de Docker en la nube. Si es necesario, uno puede alojar sus propios registros Docker y puede usarlos para extraer imágenes.

Dockerfiles vs Imágenes vs 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.

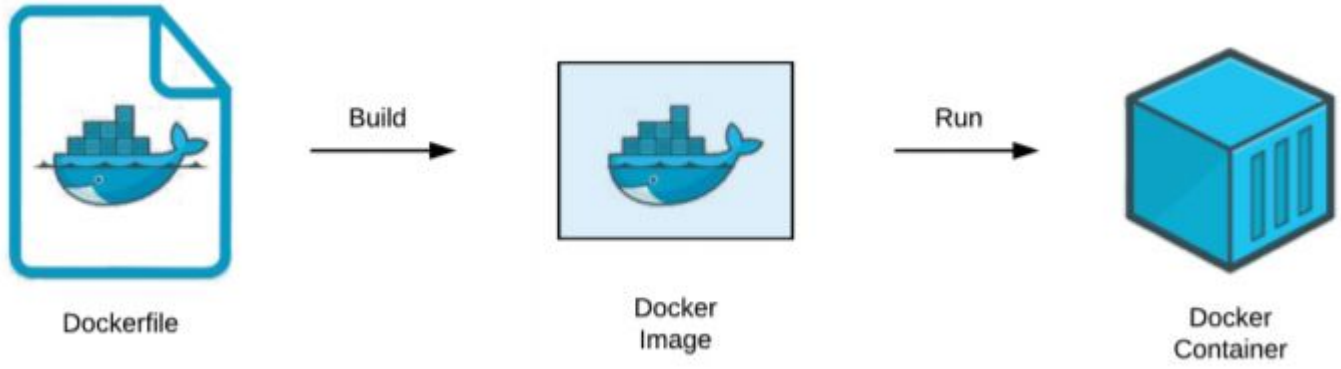


Figura: Dockerfiles vs Imágenes vs Contenedores. Imagen de Ekaba Bisong

¿Qué tecnología hay detrás de Docker?

Espacio de nombres

Es una característica de aislamiento de recursos del kernel de Linux. Nos permiten realizar visualizaciones restringidas de los recursos.

Cuando ejecutamos un contenedor, Docker crea un conjunto de *namespaces* para ese contenedor.

1. Process trees (PID namespace)
2. Mounts (MNT namespace)
3. Network (NET namespace)
4. Unix Timesharing System (UTS namespace)
5. Inter Process Communication (IPC Namespace)

¿Qué tecnología hay detrás de Docker?

Cgroups (Control Groups)

Es una característica del kernel de Linux que permite limitar y aislar recursos (CPU, memoria, disco I/O, red, etc.) utilizados por un grupo de procesos.

Sistemas de archivos en capas (Union File Systems)

Estos sistemas de archivos que funcionan creando capas, haciéndolos muy ligeros y rápidos. Docker Engine utiliza UnionFS para proporcionar los bloques de construcción para contenedores.

Productos de Docker

Docker Enterprise Edition (EE):

Es la versión empresarial y es de pago.

Docker Community Edition (CE):

Es la versión de uso gratuito, open source y se puede usar en Windows, Mac y Linux.

- Docker for Linux
- Docker Desktop for MacOS
- Docker Desktop for Windows

Instalación de Docker

El paquete de instalación de Docker se encuentra disponible en el repositorio oficial de Ubuntu.

```
$ sudo apt install docker.io
```

Configuración de usuario de Docker

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 solo 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
```

```
$ newgrp docker
```

Configurar el servicio de Docker

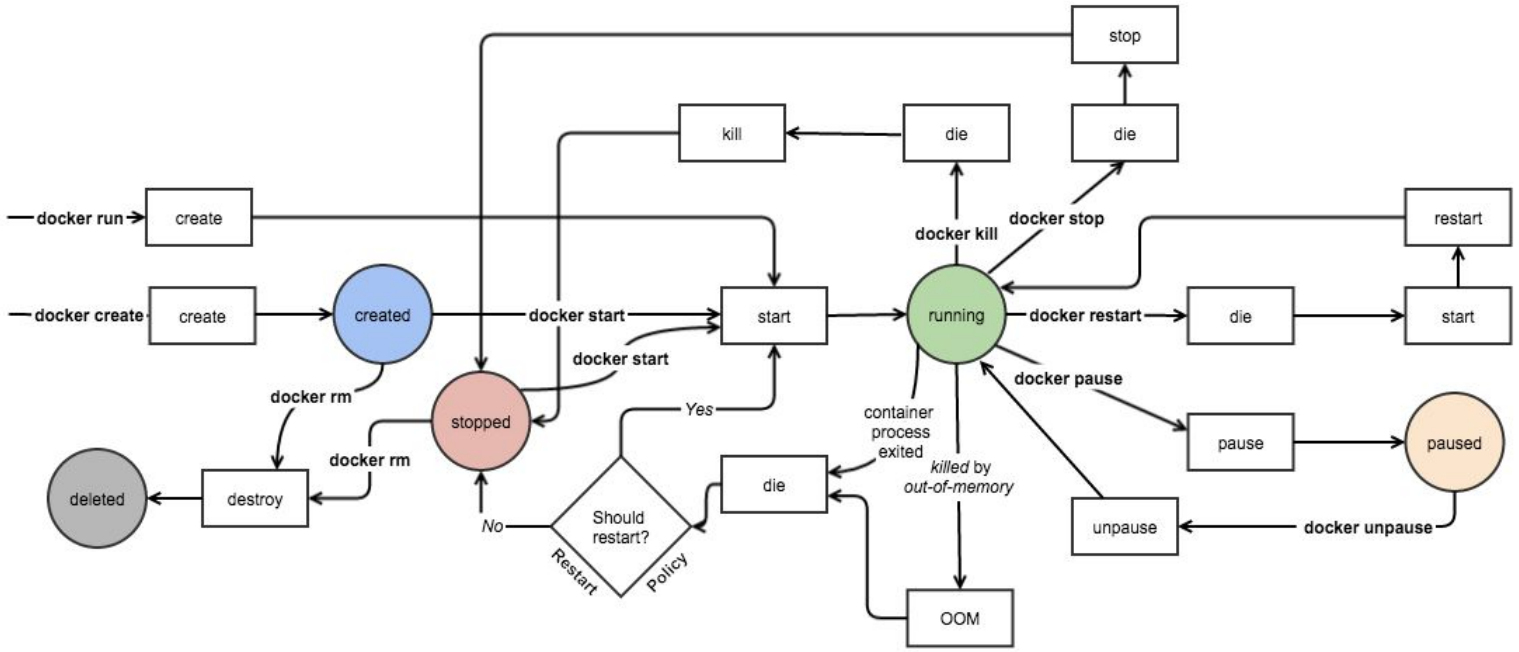
Inicie automáticamente

```
$ sudo systemctl enable docker
```

```
$ docker version
```

Administración básica de contenedores Docker

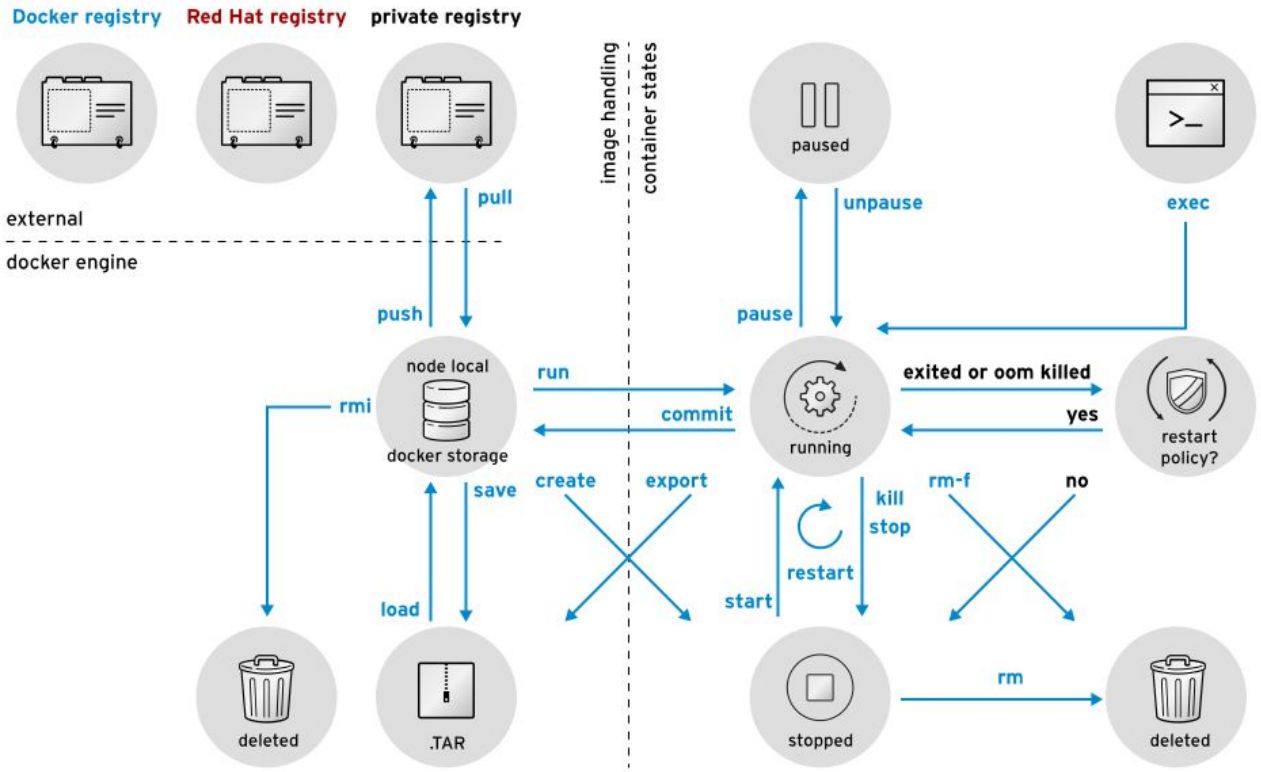
Ciclo de vida de un contenedor Docker



Ciclo de vida de un contenedor Docker. Imagen de Nitin Agarwal

Administración básica de contenedores Docker

Docker client query actions



Docker client query actions. Imagen de edX.

Comandos de Docker

```
$ docker --help

Usage: docker [OPTIONS] COMMAND
A self-sufficient runtime for containers
Options:
      --config string Location of client config files (default
    "/Users/josejuansanchez/.docker")
      -c, --context string Name of the context to use to connect to the daemon
    (overrides DOCKER_HOST env var and default context set with "docker context
    use")
      -D, --debug Enable debug mode
    …
    Management Commands:
    builder Manage builds
    config Manage Docker configs
    container Manage containers
    context Manage contexts
    image Manage images
    network Manage networks
    node Manage Swarm nodes
    plugin Manage plugins
    secret Manage Docker secrets
    service Manage services
    stack Manage Docker stack
```

Comandos de Docker

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  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
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  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
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  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
```

Comandos de Docker

```
ps          List containers
pull        Pull an image or a repository from a registry
push        Push an image or a repository to a registry
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
rmi         Remove one or more images
run         Run a command in a new container
save        Save one or more images to a tar archive (streamed to STDOUT by default)
search      Search the Docker Hub for images
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
version     Show the Docker version information
wait        Block until one or more containers stop, then print their exit codes
```

Imágenes Docker

```
$ docker search --help
```

```
Usage:  docker search [OPTIONS] TERM

Search the Docker Hub for images

Options:
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print search using a Go template
      --limit int       Max number of search results (default 25)
      --no-trunc        Don't truncate output
```

Ejemplo de búsqueda de Imágenes Docker

Vamos a buscar la imagen de Alpine Linux, que es una distribución Linux muy ligera. Esta imagen ocupa menos de 6 MB.

```
$ docker search alpine
```

```
NAME                      DESCRIPTION                              STARS     OFFICIAL    AUTOMATED
alpine                    A minimal Docker image based on ...      5797      [OK]
mhart/alpine-node         Minimal Node.js built on...              444
anapsix/alpine-java       Oracle Java 8 (and 7) with GLIBC ...     428                   [OK]
frolvlad/alpine-glibc     Alpine Docker image with gl              218                   [OK]
gliderlabs/alpine         Image based on Alpine Linux will ...     180
...
```

Docker Hub nos informa de cuales son las imágenes oficiales. Por seguridad, se recomienda hacer uso exclusivamente de las imágenes oficiales.

Puede encontrar más información sobre las imágenes oficiales de Docker en la documentación de la web oficial.

Las imágenes que aparecen marcadas como AUTOMATED son imágenes que Docker Hub ha generado automáticamente a partir del código fuente de un repositorio externo y se han enviado a los repositorios de Docker.

Puede encontrar más información sobre los *builds* automáticos en la documentación oficial de Docker.

Ejemplo de búsqueda de Imágenes Docker con filtros.

Buscar las imágenes de Alpine Linux que tengan al menos 10 estrellas.

```
$ docker search --filter stars=10 alpine
```

Buscar las imágenes de Alpine Linux que sean oficiales.

```
$ docker search --filter is-official=true alpine
```

Descargar de imágenes desde un *registry* diferente aDocker Hub

```
$ docker pull --help

Usage:  docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull an image or a repository from a registry

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
  -q, --quiet                   Suppress verbose output
```

```
$ docker pull alpine
```

```
$ docker pull alpine:latest
```

Equivalentes

```
$ docker pull alpine:3.7
```

Para una versión en especifico

Descargar de imágenes desde Docker Hub

El siguiente comando descargará la imagen curso-docker/test-image de un registry local que estará escuchando en el puerto 5000 (myregistry.local:5000):

```
$ docker pull myregistry.local:5000/curso-docker/test-image
```

Mostrar las imágenes que tenemos descargadas

```
$ docker images --help
```

```
Usage:  docker images [OPTIONS] [REPOSITORY[:TAG]]

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	775349758637	11 days ago	64.2MB
httpd	latest	d3017f59d5e2	12 days ago	165MB
alpine	latest	965ea09ff2eb	3 weeks ago	5.55MB
mariadb	latest	a9e108e8ee8a	3 weeks ago	356MB
mediawiki	latest	1d774b717f24	3 weeks ago	733MB
joomla	latest	37b651a98b60	3 weeks ago	457MB
mysql	latest	c8ee894bd2bd	3 weeks ago	456MB
hello-world	latest	fce289e99eb9	10 months ago	1.84kB

Eliminar imágenes

```
$ docker rmi alpine
```

Por su nombre

```
$ docker rmi 6d1ef012b567
```

Por su ID

```
$ docker rmi $(docker images -aq)
```

Eliminar todas las imágenes

Creación y ejecución de contenedores

Docker

Hasta aquí sólo hemos manipulado las imágenes contenidas en el Docker hub.

Para la creación y ejecución de un contenedor a partir de una imagen usamos la siguiente instrucción.

```
docker run image
```

```
docker run --help
```

Es recomendable

Hello World!

En Docker Hub existe una imagen oficial que contiene un ejemplo de "Hello World!". Este contenedor lo único que hace es mostrar un mensaje de bienvenida.

```
$ docker run hello-world
```

Veamos con detalle qué es lo que ha ocurrido.

1. En primer lugar busca si la imagen hello-world existe en el repositorio local de imágenes de nuestro equipo. Como no la ha encontrado, la ha descargado automáticamente de Docker Hub. Por lo tanto, hemos visto que no es necesario descargar la imagen previamente con docker pull.
2. En segundo lugar se crea un contenedor a partir de la imagen hello-world y se inicia.
3. Se despliega un mensaje de bienvenida y cuando finaliza la ejecución el contenedor se detiene.

Listar los contenedores que están en ejecución

```
$ docker ps
```

```
$ docker ps -a
```

Lista los contenedores (en ejecución y detenidos)

```
CONTAINER ID  IMAGE         COMMAND   CREATED         STATUS                     PORTS  NAMES
c48138039ade hello-world    "/hello"  12 seconds ago  Exited (0) 12 seconds ago         docha_can
```

Creación de un contenedor para ejecutar un comando

```
$ docker pull alpine
$ docker images
```

Descargamos y mostramos las imágenes en nuestro repositorio

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              965ea09ff2eb        2 weeks ago         5.55MB
```

```
FROM scratch ①
ADD alpine-minirootfs-3.10.3-x86_64.tar.gz / ②
CMD ["/bin/sh"] ③
```

archivo Dockerfile de la imagen alpine (veremos más adelante los Dockerfile)

① Esta instrucción indica que está utilizando la imagen *scratch* como imagen base. Esta imagen es una imagen especial que se corresponde con una **imagen vacía**.

② La instrucción ADD copia el archivo *alpine-minirootfs-3.10.3-x86_64.tar.gz* al directorio raíz del sistema de archivos de la imagen y lo descomprime. El archivo alpine-minirootfs-3.10.3-x86_64.tar.gz es un archivo que podemos ver en el mismo repositorio de GitHub donde está alojado el Dockerfile.

③ Indica que el contenedor ejecutará esta instrucción cuando se inicie.

Creación de un contenedor para ejecutar un comando

Es posible ejecutar comandos dentro del contenedor indicando el comando después del nombre de la imagen. Veamos la sintaxis de docker run.

```
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

```
$ docker run alpine cat /etc/os-release

NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.10.3
PRETTY_NAME="Alpine Linux v3.10"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://bugs.alpinelinux.org/"
```

Cuando indicamos un comando a la hora de ejecutar un comando con docker run, estamos reemplazando el comando que aparece definido en la instrucción CMD del Dockerfile, por el comando que le estamos indicando.

Creación de un contenedor en modo interactivo

Para que un contenedor no se detenga al ejecutarse debemos indicarle que queremos iniciarlo en modo interactivo.

```
$ docker run -it --name alpinec alpine
/#
```

prompt para poder interaccionar con el contenedor

- `docker run` es el comando que nos permite crear un contenedor a partir de una imagen Docker.
- El parámetro `-i` nos permite interaccionar con el contenedor a través de la entrada estándar STDIN.
- El parámetro `-t` nos asigna un terminal dentro del contenedor.
- Los dos parámetros `-it` nos permiten usar un contenedor como si fuese una máquina virtual tradicional.
- El parámetro `--name` nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre Docker nos asignará un nombre automáticamente.
- `alpine` es el nombre de la imagen. En primer lugar buscará la imagen en local y si no está disponible la buscará en el repositorio oficial Docker Hub.

Eliminar un contenedor

Para eliminar el contenedor que está detenido y está ocupando espacio en nuestro disco ejecutaremos el comando docker rm.

```
$ doker rm -f alpinec
```

o

```
$ doker rm -f e9af6c8dbffb
```

A partir de este momento **siempre** que vayamos a crear un nuevo contenedor **añadiremos el parámetro** `--rm` para que cuando se detenga se elimine automáticamente.

Para **salir del contenedor y detenerlo**:

- Escribir `exit`
- Pulsar `CTRL` + `D`

Para **salir del contenedor SIN detenerlo**:

- Pulsar `CTRL` + `P` + `Q`

Acceder al terminal de un contenedor que está en ejecución

```
$ docker attach --help

Usage:  docker attach [OPTIONS] CONTAINER

Attach local standard input, output, and error streams to a running container

Options:
      --detach-keys string   Override the key sequence for detaching a container
      --no-stdin             Do not attach STDIN
      --sig-proxy            Proxy all received signals to the process (default true)
```

Nos permite acceder al terminal de un **contenedor que está en ejecución** indicando su nombre o su ID. Tenga en cuenta que **no crea un nuevo terminal (tty)**, sino que usa el terminal original que está en ejecución de modo que **si salimos del terminal con `exit` el contenedor se detendrá**.

Ejemplo de attach

Ejecutamos un contenedor en modo *detached* (-d) y le añadimos la opción (-it) para poder interaccionar con él a través de un terminal.

Vamos a ejecutar en el contenedor el comando `/usr/bin/top` en modo *batch* (-b) para que siga ejecutándose en segundo plano. Si no utilizásemos el modo batch el contenedor se detendría una vez que finaliza la ejecución del comando.

```
$ docker run -dit \
--rm \
--name topdemo \
alpine /usr/bin/top -b
```

```
$ docker attach topdemo
```

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

Acceder al terminal de un contenedor que está en ejecución

```
docker exec --help

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
```

Nos permite **ejecutar un comando en un contenedor que está en ejecución** indicando su nombre o su ID. `exec` **ejecuta el comando en un proceso nuevo, asignándonos un nuevo terminal.**

Esto significa que **si salimos del contenedor con** `exit`**, el contenedor no detendrá su ejecución.**

Ejemplo de exec

```
$ docker run -dit \
--rm \
--name topdemo \
alpine /usr/bin/top -b
```

```
$ docker  exec -it topdemo /bin/sh
```

```
/#
```

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

Si ejecutamos `ps aux` para ver los procesos que están en ejecución dentro del contenedor veremos lo siguiente.

```
# ps aux

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  36480  3032 pts/0    Ss+  10:25   0:00 /usr/bin/top -b
root        16  0.6  0.0   4624   816 pts/1    Ss   10:28   0:00 /bin/sh
root        23  0.0  0.1  34396  2816 pts/1    R+   10:28   0:00 ps aux
```

Vemos como el proceso con **PID 1** es el que está ejecutando el comando `/usr/bin/top -b` y el proceso con **PID 16** es el del comando `/bin/sh` que es el que hemos ejecutado don `docker exec`.

stop y start contenedores

```
$ docker start container_name
```

```
$ docker stop container_name
```

Creación de un contenedor en segundo plano

Hasta este momento hemos visto dos formas usar los contenedores:

1. Creamos un contenedor para ejecutar un comando dentro de él, esperamos a que finalice el comando y cuando el comando finaliza el contenedor se detiene.
2. Creamos un contenedor en modo interactivo, donde podemos acceder a un terminal y ejecutar comandos en él.

Existe otra posibilidad, que además es la más utilizada, consiste en la ejecución de un contenedor en segundo plano mientras que éste ejecuta una aplicación en primer plano.

Para ejecutar un contenedor en segundo plano se utiliza la opción -d (detach).

Ejemplo contenedor en segundo plano

Utilizaremos la imagen oficial httpd, que es de Apache HTTP Server.

```
$ docker run -d \
--rm \
--name httpdc \
httpd
```

```
$ docker ps
```

Comprobamos que el contenedor está en ejecución:

```
$ docker logs -f httpdc
```

Para ver los registros de salida (STDOUT)

```
$ docker inspect httpdc
```

Con el comando docker inspect podemos obtener información sobre el contenedor

Ejemplo contenedor en segundo plano

Si buscamos la dirección IP del contenedor vemos que es una dirección privada del tipo 172.17.0.x

```
$ docker inspect httpdc| grep IPAddress
"IPAddress": "172.17.0.2
```

El contenedor está en la red bridge y por lo tanto sólo es accesible desde el servidor que está ejecutando el servicio de Docker o desde otro contenedor de la misma red (En Linux, en macOS tiene otro comportamiento).

```
# curl http://172.17.0.2
<html><body><h1>It works!</h1></body></html>
```

Por lo tanto, para acceder al servidor web desde nuestra red y no sólo desde el servidor que está ejecutando el servicio de Docker tendremos que exponer los puertos.

```
$ docker rm -f httpdc
```

Eliminamos el contenedor

Exponer puertos

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

Existen dos opciones:

1. -p
2. -P

```
puerto_local:puerto_contenedor
```

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

redireccionar el puerto 81 de nuestra máquina con el puerto 80 del contenedor.

```
$ curl http://localhost:81
```

Exponer puertos

se seleccionará un puerto aleatorio de nuestra máquina y se redirigirá al puerto 80 del contenedor.

```
$ docker run -d \
--rm \
--name httpdc \
-P \
httpd
```

```
$ docker ps
```

```
CONTAINER ID  IMAGE  COMMAND              CREATED        STATUS        PORTS                   NAMES
96922862a483  httpd  "httpd-foreground"   3 seconds ago  Up 1 second   0.0.0.0:32770->80/tcp   httpdc
```

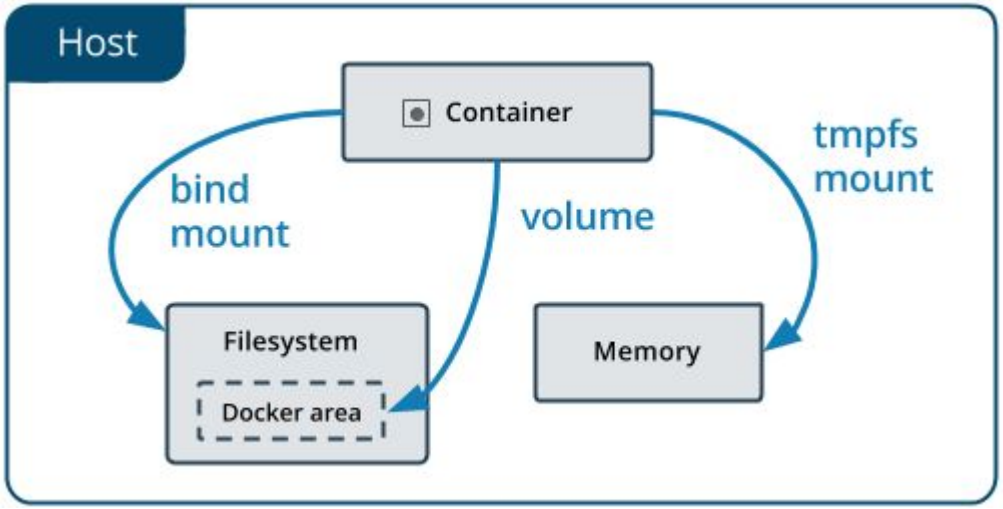
```
$ curl http://localhost:32770
```

Almacenamiento en Docker

Por defecto, todos los archivos que se crean dentro de un contenedor se almacenan en la última capa del sistema de archivos (la capa de lectura/escritura), esto quiere decir que los datos que tenemos en esta capa se perderán cuando el contenedor se elimine y no podremos compartirlos con otros contenedores.

Docker nos ofrece dos posibilidades para implementar persistencia de datos en los contenedores:

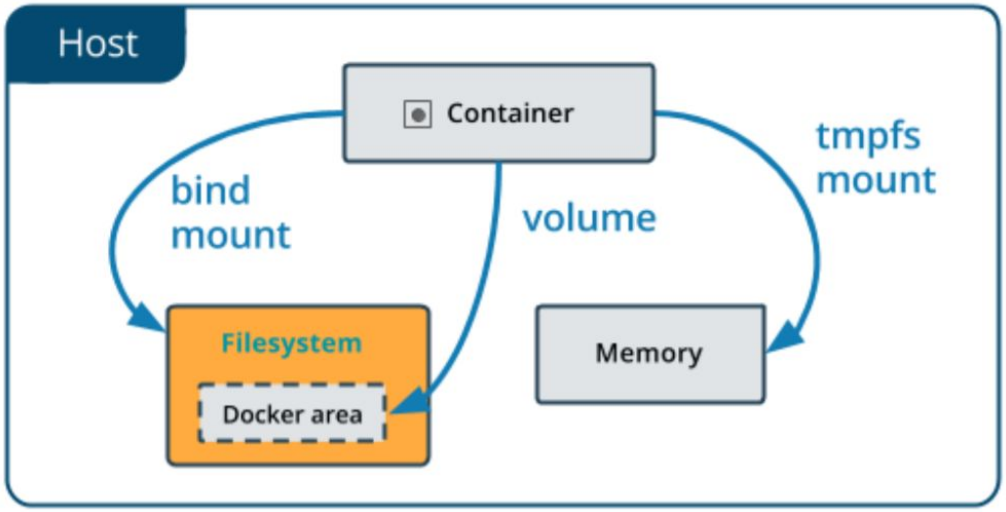
1. *Bind mouts*
2. *Volumes*



Manage data in Docker. Imagen de Docker.com

Bind mounts en Docker

Los *bind mounts* pueden estar almacenados en cualquier directorio del sistema de archivos de la máquina host. Estos archivos pueden ser consultados o modificados por otros procesos de la máquina host o incluso por otros contenedores Docker.



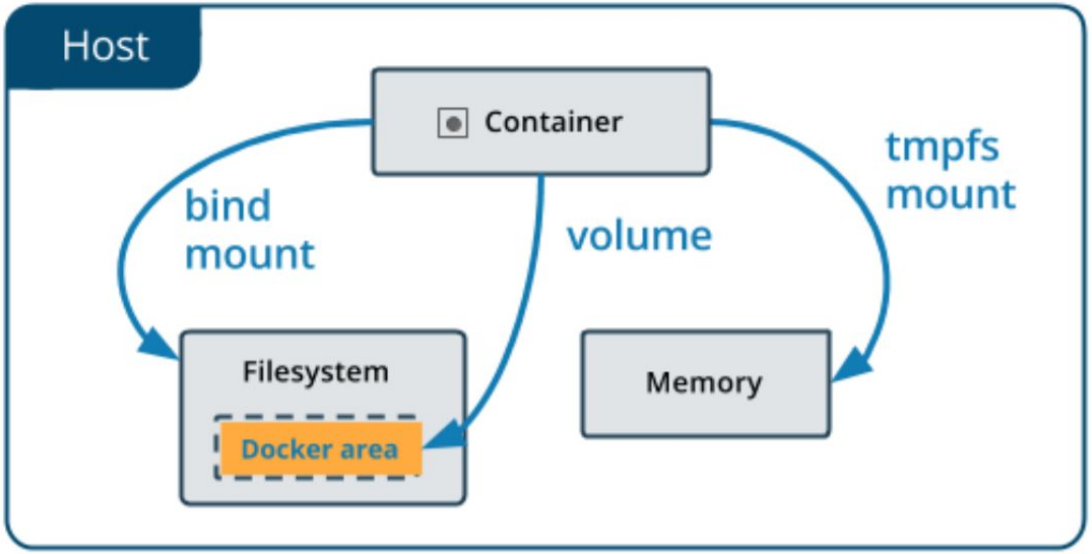
Manage data in Docker. Imagen de Docker.com

Volumes en Docker

Los *volumes* se almacenan en la máquina host dentro del área del sistema de archivos que gestiona Docker. Por ejemplo, en Linux será el directorio `/var/lib/docker/volumes`.

Otros procesos de la máquina host no deberían modificar estos archivos, sólo deberían ser modificados por contenedores Docker.

Desde la documentación oficial de Docker nos aseguran que esta es la mejor forma de implementar persistencia de datos en los contenedores Docker.




```
$ docker volume --help
```

Manage data in Docker. Imagen de Docker.com

Ejemplo de bind mounts en Docker

¿Cuándo sería apropiado utilizar un volumen de tipo *bind mount*?

- Para compartir archivos de configuración entre la máquina host y el contenedor.
- Para compartir el código de las aplicaciones entre la máquina host y el contenedor en un entorno de desarrollo.

Para montar un directorio podemos utilizar los flags *-v* o *--mount*. En nuestros ejemplos utilizaremos -v.

En el caso de los *bind mounts* tendremos tres campos separados por dos puntos (:) y tendrán el siguiente orden:

1. En primer lugar se indica el archivo o directorio de la máquina host.
2. En segundo lugar se indica en qué archivo o directorio lo vamos a montar dentro del contenedor.
3. El tercer parámetro es opcional, y puede ser una lista separada por comas con las siguientes opciones: *ro*, *consistent*, *delegated*, *cached*, *z* y *Z*.

Ejemplo de bind mounts en Docker

```
$ docker run -dit \
--rm \
--name alpinec \
-v /home/jesus/target:/app \
alpine
```

Indicando el *path* completo

```
$ docker run -d \
--rm \
--name alpinec \
-v  "$(pwd)"/target:/app \
alpine
```

Utilizar la salida del comando *pwd* para construir el *path*

```
$ docker run -d \
--rm \
--name alpinec \
-v  "$PWD"/target:/app \
alpine
```

Utilizar la variable *$PWD* para construir el *path*

```
$ docker inspect alpinec
```

Podemos inspeccionar el contenido del contenedor para verificar que el directorio se ha creado correctamente.

```
$ docker exec -it alpinec /bin/bash
```

Ejemplo de bind mounts en Docker solo lectura

```
$ docker run -dit \
--rm \
--name alpinec \
-v /home/jesus/target:/app:ro \
alpine
```

Indicando el *path* completo

```
$ docker inspect alpinec
```

Podemos inspeccionar el contenido del contenedor para verificar que el directorio se ha creado correctamente.

```
$ docker exec -it alpinec /bin/bash
```

Ejemplo de *volume* en Docker

Vamos a utilizar la imagen oficial mysql.

Descarguen la imagen de Docker hub

```
$ docker volume create mysql_data
```

Creamos un volumen interno gestionado por Docker

```
$ docker run -d \
--rm \
--name mysqlc \
-e MYSQL_ROOT_PASSWORD=root \
-p 3308:3306 \
-v mysql_data:/var/lib/mysql \
mysql
```

Recuerden al detener *mysql* el contenedor se eliminara (opcional)

```
$ docker exec -it mysqlc /bin/bash
```

Abrimos un terminal en el contenedor para interaccionar con él.

Ejemplo de *volume* en Docker

Vamos a utilizar la imagen oficial mysql.

Descarguen la imagen de Docker hub

```
# mysql -u root -p
```

Una vez que estamos dentro del contenedor nos conectamos desde la consola de MySQL

```
DROP DATABASE IF EXISTS tienda;
CREATE DATABASE tienda CHARSET utf8mb4;
USE tienda;
CREATE TABLE fabricante (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL
);
INSERT INTO fabricante VALUES(1, 'Asus');
INSERT INTO fabricante VALUES(2, 'Lenovo');
INSERT INTO fabricante VALUES(3, 'Hewlett-Packard');
SHOW TABLES;
SELECT * FROM fabricante;
```

Creamos una nueva base de datos con los siguientes datos

```
$ docker stop mysqlc
```

Detenemos el contenedor. Como hemos iniciado el contenedor con la opción --rm al detenerlo se eliminará automáticamente.

Construir imágenes con Dockerfile

Docker puede construir imágenes automáticamente leyendo las instrucciones de un archivo Dockerfile, que es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comandos para ensamblar una imagen.

```
FROM ubuntu:18.04

RUN apt-get update && apt-get install -y \
    python3

COPY . /opt/app01

CMD ["python3", "/opt/app01/holamundo.py"]
```

Inicia con un SO base

Instalar dependencias

Copiar código fuente

Ejecutar comando

Construir imágenes con Dockerfile

1.- Crear una carpeta llamada ejemplo_dockerfile

2. En el directorio ejemplo_dockerfile , crear un archivo llamado Dockerfile con el siguiente contenido

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    python3
COPY . /opt/app01
CMD ["python3", "/opt/app01/holamundo.py"]
```

3.En el directorio ejemplo_dockerfile , crear un archivo llamado holamundo.py con el siguiente contenido:

```
input("Hola Mundo")
```

4.Construir la imagen con *docker build*, de la siguiente forma donde "- t" nos indica la etiqueta para identificar nuestra imagen que será hola mundo:

```
$ docker build -t holamundo .
```

5.Creamos un contenedor a partir de nuestra imagen recién creada (puedes verificar su creación con docker images):

```
$ docker run -it holamundo
```

NOTA: como es un script interactivo no en segundo plano, se debe utilizar la opción -it.

Subir una imagen a Docker Hub

En primera instancia se debe ingresar al Docker Hub con docker login:

```
usuario@ubuntu01:~/ejemplo_dockerfile $ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you
don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: jesusabundis
Password:                                          #number_letter
WARNING! Your password will be stored unencrypted in
/root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
usuario@ubuntu01:~/ ejemplo_dockerfile$
```

Para Crear y subir nuestra imagen de holamundo2, será de la siguiente forma:

```
usuario@ubuntu01:~/ejemplo_dockerfile $ docker build -t
jesusabundis/holamundo2 .

usuario@ubuntu01:~/ ejemplo_dockerfile $ sudo docker push
jesusabundis/holamundo2
```

En la primera línea se observa la creación de una imagen nueva una pequeña variación, donde **jesusabundis**, es el nombre de usuario en docker hub.