# DOCKER

Docker funciona con una tecnología que se llama “contenedores de Linux”.

## Contenedores vs Máquinas Virtuales

Un contenedor se compone por:

* Namespaces - Permite a la aplicación que se ejecuta en un contenedor tener una vista de los recursos del OS.
* Cgroups - Limitan y miden los recursos del OS.
* Chroot - Permite al contenedor tener una vista del file system para él mismo.

Un contenedor trabaja directo sobre el kernel del OS, lo que lo hace más liviano que una máquina virtual porque no se tendrá que instalar un OS por contenedor y el tiempo de arranque es inmediato a diferencia de las VMs que necesitan de un OS instalado en cada una y tardan algunos minutos en iniciar un kernel y cargar los módulos, etc. Un contenedor entonces es un proceso aislado con recursos definidos e inician instantáneamente.

Un contenedor no sustituye a las VMs, sino que nos permite empaquetar nuestra aplicación y ejecutarla en un entorno aislado de manera portable por lo que se pueden tener varios contenedores sobre una VM.

## Instalación

### Windows o Mac

1. Se descarga la última versión de DockerToolbox para Windows o Mac.
2. Se corre el instalador.
3. Ejecutar Docker Quickstart Terminal para que nos cree una VM de Linux donde correrá el demonio de Docker.

NOTA: Si se tienen problemas para que el Quickstart cree la VM se debe de indicar al BIOS que permita la virtualización, por lo que necesitaremos reiniciar la máquina y modificar la configuración de virtualización y después ejecutar de nuevo Docker Quickstart Terminal.

1. Correr comando “docker-machine ls” para obtener la IP de la VM y guardarla para futuros usos.

### Linux

1. Correr el script que se encuentra en <https://get.docker.com/> (el mismo script contiene la línea que debemos ejecutar para instalar docker “curl -fsSL get.docker.com -o get-docker.sh”).
2. Correr el comando que nos comenta al final del instalador para que nuestro usuario tenga permisos para ejecutar el demonio.

## Ver las opciones posibles de algún comando de docker

docker [comando] --help

Ejemplo:

**docker ps --help** - Muestra todas las posibles opciones que podemos utilizar con el comando ps como -a, --filter, etc.

## Contenedores

### Comandos

#### Crear y correr contenedores nuevos a partir de imagen

docker run [opciones] [imagen] [comando] [args]

*NOTA: Si la imagen no la tenemos descargada, la instalará con la versión latest.*

##### Opciones

###### -it

Crea contenedor y permite interactuar con él.

Ejemplo:

**docker run -it ubuntu bash** - Se meterá al contenedor Ubuntu permitiendo interactuar con él como si estuviéramos en una VM pero en realidad estamos en el contenedor de Ubuntu o una emulación de un servidor Linux.

###### -d

Corre el conteiner en background (como demonio)

Ejemplo:

**docker run -d ubuntu ping -c 10 www.google.com** - Deja corriendo el comando ping en el contenedor y podemos seguir trabajando en otra cosa.

###### -P

Crea contenedores como demonio, pero accesibles desde la IP de VM, útil para correr servidores como Tomcat sin necesidad de instalarlo en localhost.

Ejemplo:

**docker run -P -d tomcat** - Corre una instancia de Tomcat.

###### --name [nombre de contenedor]

Asigna un nombre a un contenedor.

Ejemplo:

**docker run -it --name micontenedor ubuntu bash** - Crea contenedor con nombre micontenedor para que sea más fácil volver a él sin necesidad de buscar su PID.

#### Ver los logs de algún contenedor que está corriendo como demonio

docker logs [opciones] [id/nombre del contenedor]

*NOTA: Sin opciones solo muestra lo que tiene el log al momento en que se llama.*

##### Opciones

###### -f

Ver cómo cambia el log en tiempo real.

Ejemplo:

**docker logs -f micontenedor**

#### Mostrar id y status de contenedores

docker ps [opciones]

*NOTA: Sin opciones muestra solo los contenedores activos.*

##### Opciones

###### -a

Muestra todos los contenedores que se tienen.

###### -q

Muestra solo los ID de los contenedores.

###### --filter=[filtro]

Filtra los contenedores que se mostrarán.

Ejemplo:

**docker ps -a --filter=”exited=1”** - Mostrará solo los contenedores que terminaron con error.

#### Salir del contenedor terminando proceso

exit

#### Detener un contenedor

docker stop [id/nombre del contenedor]

#### Forzar la detención de un contenedor

docker kill [id/nombre del contenedor]

#### Volver a correr un contenedor en estado stopped o killed

docker start [opciones] [id/nombre del contenedor]

*NOTA: Sin opciones lo corre como demonio.*

##### Opciones

###### -a

Corre de nuevo el contenedor, pero no como demonio.

#### Pausar contenedor

docker pause [id/nombre del contenedor]

#### Continuar un contenedor pausado

docker unpause [id/nombre del contenedor]

#### Ver la información a bajo nivel de un contenedor

docker inspect [id/nombre del contenedor]

#### Salir del contenedor sin terminar el proceso

Ctrl+P+Q

#### Regresar a un contenedor corriendo

docker attach [id/nombre del contenedor]

#### Entrar a un contenedor por otra puerta

docker exec -it [id/nombre del contenedor] bash

*NOTA: Si entramos con attach desde otra terminal, todas las instancias harán las mismas cosas; con exec las otras instancias no se mueven por lo que podríamos ver logs o realizar alguna otra tarea sin detener las otras.*

#### Borrar un contenedor

docker rm [opciones] [id/nombre del contenedor]

*NOTA: Para borrar todos los contenedores, le podemos pasar con variable los nombres de todos los contenedores docker rm $(docker ps -a -q).*

##### Opciones

###### -f

Borra un contenedor, aunque está corriendo.

#### Mostrar las diferencias que tiene el contenedor sobre la imagen base

docker diff [id/nombre del contenedor]

## Imágenes

Es una plantilla solo lectura para crear nuestros contenedores y contiene todo lo necesario para que el contenedor realice su acción como puede ser un código compilado, código fuente, librerías, archivos de configuración, variables de entorno, etc.

La imagen no se puede modificar, solo se puede agregar capas de imágenes.

### Comandos

#### Descargar imágenes de Docker Hub

docker pull imagen:tag

*NOTA: Si omitimos el tag, bajará por defecto la versión latest.*

#### Mostrar imágenes existentes

docker images

#### Crear imágenes a partir de un contenedor

docker commit [id/nombre del contenedor] [nombre de la imagen]

#### Ver el historial de alguna imagen para saber cuántas capas tiene

docker history [nombre de la imagen]

#### Asignar un tag a una imagen

docker tag [nombre de la imagen] [nombre del tag]

#### Borrar imágenes

docker rmi [nombre de la imagen]

### Dockerfiles

Se pueden utilizar archivos para crear imágenes instalando lo necesario para nuestro proyecto y que no tenga nuestro contenedor.

#### Generar imágenes a partir de un dockerfile.

1. Se crea archivo dockerfile.

En el archivo se pueden utilizar las siguientes palabras:

* 1. **FROM** para indicar la imagen donde instalaremos cosas.
  2. **RUN** para correr los comandos con los que instalaremos utilidades en cada capa.

Ejemplos:

**FROM ubuntu**

**RUN apt-get install -y vim**

**RUN apt-get install -y curl**

*NOTA: En este ejemplo se agregarán 2 capas, una para vim y otra para curl. No es buena práctica hacerlo de esta manera por cuestiones de performance. Es mejor hacerlo en una sola capa como el ejemplo siguiente:*

**FROM ubuntu**

**RUN apt-get install -y vim && \**

**apt-get install -y curl**

* 1. **CMD** si queremos definir por defecto un comando a ejecutar cuando se crea el contenedor.

Ejemplo 1:

**FROM ubuntu**

**CMD ping www.google.com -c 2**

Ejemplo 2:

**FROM ubuntu**

**CMD [“ping”, “www.google.com”, “-c”, “2”]**

* 1. **ENTRYPOINT** si queremos que nuestro comando interactue con el usuario.

Ejemplo 1:

**FROM ubuntu**

**ENTRYPOINT ping**

Ejemplo 2:

**FROM ubuntu**

**ENTRYPOINT [“ping”]**

*NOTA: En estos ejemplos al correr el docker run de nuestro contenedor entonces podremos mandarle las opciones que hacen falta para completar el comando.*

**docker run ubuntu\_ping www.google.com -c 2**

NOTA: Se puede mezclar CMD y ENTRYPOINT para que en CRM se tengan las opciones por defecto, pero para poderlo utilizar así se tiene que tener en formato de arreglo.

**FROM ubuntu**

**ENTRYPOINT [“ping”]**

**CMD [“-c”, “2”, “www.google.com”]**

* 1. **COPY** o **ADD** si queremos copiar algún archivo a otro lado. Se utiliza ADD más cuando se necesite descomprimir lo que copiaremos u obtener archivos de una URL.

Ejemplo1:

**FROM ubuntu**

**COPY archivo.txt /tmp/archivo.txt**

**CMD cat /tmp/archivo.txt**

Ejemplo 2:

**FROM ubuntu**

**ADD http://cdn.meme.am/instances/66627195.jpg /tmp/img.txt**

* 1. **EXPOSE** si nuestro contenedor será algo que estará escuchando sobre un puerto.

Ejemplo:

**EXPOSE 5000**

* 1. **WORKDIR** si queremos especificar una ruta donde encontrará los recursos que indicamos en el dockfile.

Ejemplo:

**FROM ubuntu**

**WORKDIR /tmp**

**COPY archivo.txt ./**

**CMD cat archivo.txt**

* 1. **MAINTAINER** para asignarle un autor a la imagen. Si lo utilizamos tiene que ser lo primero que esté después del FROM.

Ejemplo:

**FROM ubuntu**

**MANTAINER jazzo01@hotmail.com**

* 1. **ENV** para asignar variables de entorno a la imagen.

Ejemplo:

**FROM ubuntu**

**ENV key value**

1. Corremos docker build [opciones] [ruta donde esta nuestro dockerfile].

*NOTA: Docker utiliza un cache para saber si esa misma imagen ya fue creada para no realizar todo el proceso cada vez que ejecutamos build, esta cache se basa en exacto string y orden de las instrucciones, por lo que, si movemos de lugar alguna instrucción, docker lo tomara como una imagen nueva.*

Opciones:

**-t**

Para especificarle un nombre a la nueva imagen.

Ejemplo:

**docker build .**

**--no-cache**

Para que docker no utilice el cache y siempre instale todo lo que dice nuestro dockerfile.

## DockerHub

Podemos exportar o importar imágenes hechas por uno mismo, para ello se necesita estar registrado en <https://hub.docker.com>

### Exportar una imagen a DockerHub

1. Asignar un tag adecuado para compartirlo.

**docker tag [nombre de la imagen] [nombre del tag con formato nombre\_usuario\_dockerhub/nombre\_imagen]**

1. Enviar tag a DockerHub

**docker push [nombre del tag con formato nombre\_usuario\_dockerhub/nombre\_imagen]**

## Configuración avanzada de Docker para producción (en servidor Linux)

### PENDIENTE PORQUE NO ES NECESARIO PARA LOS FINES INMEDIATOS

Lo primero que tenemos que saber es si el demonio se encuentra corriendo como servicio (si instalamos el demonio con los scripts o la manera recomendada por Docker); si es así entonces podemos configurarlo con los comandos service o systemctl.

Utilizaremos el comando service cuando nuestra versión de Linux utiliza sistema upstart (versiones anteriores).

Utilizaremos el comando systemctl cuando nuestra versión de Linux utiliza systemd (nuevas versiones).

### Comandos

#### Detener el demonio de docker manualmente

sudo service docker stop

#### Iniciar el demonio de docker manualmente

sudo service docker start

#### Reiniciar el demonio de docker

sudo service docker restart