Docker

Permite resolver problemas de construir, distribuir y ejecutar software en diferentes plataformas.

Arquitectura de Docker

Docker tiene una arquitectura tipica cliente-servidor en donde 'el servidor" no es un servidor como los servidores web sino que es un **servicio** del sistema operativo o "Daemon"

Este **servicio** esta corriendo en la **maquina local** y el cliente, que le habla, es otro servicio que también corre en la maquina local que seria el **CLI**

El "servidor" es quien crea los contenedores y los administra, no el "cliente"

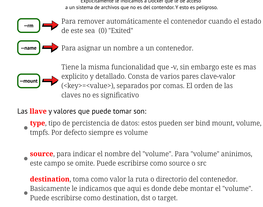
Deamon

Un Daemon (llamado así en sistemas UNIX), servicio o programa residente es un tipo especial de proceso informático no interactivo, es decir, que se ejecuta en **segundo plano** en vez de ser controlado directamente por el usuario.

Los servicios o Daemon son la columna vertebral del Sistema Operativo.

Contenedores
(la pieza fundamental de Docker)

Es una agrupación de varios procesos. También puede ser un solo proceso.



Ejecutar comandos en contenedores o iniciarlos

$ docker run <opciones> <contenedor> <comando>

Ejecuta un comando en un contenedor nuevo. Lo podemos usar para correr o iniciar un nuevo contenedor.

-t Para asignar una pseudo-tty

-i Para mantener el STDIN (0, entrada de teclado) abierto incluso si no esta conectado a una Psudoterminal o terminal

-d Ejecuta el contenedor en segundo plano e imprime el ID del contenedor.

-p Le decimos que puerto de nuestra maquina (**host**) estará atado (binded) al puerto **X** del contenedor. Básicamente publicar los puertos de un contenedor en el host.

```
Docker -p <puerto-maquina>:<puerto-contenedor>

PORTS
0.0.0.0:8080->80/tcp
```

-v Indica que un directorio o ruta (filesystem) de nuestra maquina (**host**) estará atado (binded) a un directorio o ruta del contenedor (filesystem). Básicamente se almacenando los datos en nuestra maquina por lo tanto, si se elimina el contenedor no se pierden los datos.

```
Docker --mount <path-maquina>:<path-contenedor>
Explicitamente le indicamos a Docker que le de acceso
a un sistema de archivos que no es del contenedor Y esto es peligroso.
```

--rm Para remover automáticamente el contenedor cuando el estado de este sea (0) "Exited"

--name Para asignar un nombre a un contenedor.

--mount Tiene la misma funcionalidad que -v, sin embargo este es mas explicito y detallado. Consta de varios pares clave-valor (<key>=<value>), separados por comas. El orden de las claves no es significativo

Las **llave** y valores que puede tomar son:

- **type**, tipo de persistencia de datos: estos pueden ser bind mount, volume, tmpfs. Por defecto siempre es volume
- **source**, para indicar el nombre del "volume". Para "volume" anónimos, este campo se omite. Puede escribirse como source o src
- **destination**, toma como valor la ruta o directorio del contenedor. Basicamente le indicamos que aqui es donde debe montar el "volume". Puede escribirse como destination, dst o target.
- **readonly**, si esta opción esta activa entonces el "volumen" solo será de lectura. Por defecto es de lectura y escritura.
- **volume-driver**, para especificar el nombre del "driver" a utilizar. Por defecto es local
- **volume-opt**, toma un par clave-valor, con el cual podemos asignar la configuración especifica del "driver"

```
Docker --mount type=volume, src=<VOLUME-NAME>, dst=<CONTAINER-PATH> \
       /redis/data, volume-driver=local, volume-opt=type=nfs
```

Todas las opciones de volúmenes están disponibles para --mount y -v, sin embargo, cuando se utilizan volúmenes con servicio, solo --mount esta permitido.

--tmpfs Para montar dentro del contenedor una ruta o directorio temporal, es decir se guarda en memoria los datos y cuando se elimine el contenedor se elimina toda la información registrada.

-e Para colocar variables de entorno.

$ docker exec <opciones> <contenedor> <comando>

Ejecuta un comando en un contenedor que se encuentra corriendo o esta activo

-t Para asignar una pseudo-tty

-i Para mantener el STDIN (0, entrada de teclado) abierto incluso si no esta conectado a una Psudoterminal o terminal

-d Detached mode: Ejecuta el comando en segundo plano.

Estado de Docker

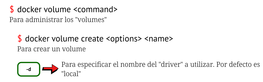
$ docker ps

Para listar los contenedores. Por defecto solo muestra los que se están ejecutando o están activos.

-a Para listar **todos** los contenedores, incluyendo los contenedores que finalizaron o están desactivados.

-q Para mostrar únicamente las ID's de los contenedores

Cuando termina de ejecutar los procesos o las instrucciones de ese contenedor, se sale dejando el estado "exited":



$ docker inspect <id-container | name-container>

Muestra todos los detalles internos en JSON del contenedor. Esta información nos ayuda para saber el estado del contenedor.

Comandos básicos

$ docker rename <name-container> <new-name>

Para cambiar el nombre de un contenedor.

$ docker rm <id-container | name-container>

Para eliminar uno o mas contenedores

-f Para forzar la eliminación de un contenedor en ejecución (es como utilizar SIGKILL)

$ docker logs <id-container | name-container>

Coge los registros de un contenedor y los imprime.

$ Docker kill <opciones> <id-container | name-container>

Para matar uno o mas contenedores en ejecución

-s Para enviar al contenedor el tipo de señal. Por defecto se envía "KILL"

$ Docker pull <nombre-image>:<versión>

Para descargar una imagen desde el Docker hub a nuestra maquina. En caso de no colocar la versión, Docker tomara por defecto la ultima versión de la imagen (latest)

$ Docker rmi <options> <image-name>

Para remover 1 o mas imágenes.

$ Docker tag <image-fuente:tag> <user/repo:tag>

Crea una etiqueta que hace referencia o apunta a la "image fuente".

Volúmenes

Los volúmenes se almacenan en una parte del sistema de archivos de mi maquina que **es administrado por Docker** (/var/lib/docker/volume/ en Linux). Los procesos que no son de Docker **no deberian** modificar esta parte del sistema de archivos. Los volúmenes son la mejor forma de conservar datos en Docker. También se puede almacenar volúmenes de forma remota, conectándonos por SSH.

$ docker volume <command>

Para administrar los "volumes"

$ docker volume create <options> <name>

Para crear un volume

-d Para especificar el nombre del "driver" a utilizar. Por defecto es "local"

-o Mapa key=value, con el cual podemos asignar la configuración especifica del "driver". Por defecto esta vacio (map[])

$ docker volume ls <options>

Para listar todos los "volume" creador implicitamente(anónimos, Docker le asigna un nombre aleatorio) o explicitamente(con nombre) en Docker.

-f Para filtrar

-q Para mostrar únicamente los nombres de los "volume"

$ docker volume prune <options>

Para remover todos los "local volume" que no se están usando.

$ docker volume rm <options>

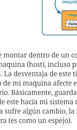
Para remover 1 o mas "volume". No se puede remover un "volume" que este siendo usado por un contenedor.

recuperado de: https://docs.docker.com/storage/volumes/

NOTAS

Luis Andrés Villegas Sanchez

Bind mounts



Permite montar dentro de un contenedor, un directorio del sistema de archivos de mi maquina (host), incluso pueden ser archivos o directorios importantes del sistema. La desventaja de este tipo de persistencia es que, es posible que algún proceso de mi maquina afecte este directorio o archivo dentro del directorio. Básicamente, guardar la información del contenedor generando una salida de este hacia mi sistema de archivos, pero si la carpeta donde se genera la salida sufre algún cambio, la información que percibe el contenedor también cambiara (es como un espejo).

Nota: Si el directorio del proyecto ya poseen contenido a dentro (carpetas o archivos), sobreescribiran su contenido en el contenedor, en otras palabras lo opacaran por completo.

Tmpfs

A diferencia de los volúmenes y los Bind mount (montajes de enlace), un montaje tmpfs es temporal y solo persiste en la memoria de mi maquina. Cuando el contenedor se detiene, el montaje tmpfs se elimina y los archivos escritos allí no se conservarán.

Imágenes

Las imágenes en docker son principalmente **la base** o las **plantillas** utilizadas para la **creación** de contenedores. Además también tienen la gran ventaja de ser livianas y fácilmente adquiribles. Las imágenes docker por defecto se descargan desde https://hub.docker.com/.

Algo que debemos tener en cuenta es que las imágenes **no van a cambiar**, es decir, una vez este realizada no la podremos cambiar.

```
$ docker image <command>
```
Para administrar las imagen

```
$ docker image ls <options> <repository>:tag
```
Para listar las imágenes.

```
$ docker image prune <options>
```
Para remover todas las imágenes que no se utilizan.

```
$ docker image rm <options> <image-name>
```
Para remover una o mas imágenes.

```
$ docker push <options>  <name>:<tag>
```
Para publicar una imagen.

```
$ docker pull <options>  <name>:<tag>
```
Para traerme o descargar una imagen

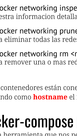
Construir nuestras imágenes

Para construir nuestras propias imágenes, necesitamos un archivo llamado **DockerFile.** Este archivo es como la receta para crear imágenes.

/proyecto/DockerFile

```
Siempre se debe empezar con el FROM, es nuestra
imagen base sino empezamos con esto entonces
no va a funcionar.
FROM <imagen base>

Copia los archivos o directorios en la ruta <src> (es
decir la ruta del proyecto o contexto de build)
hacia el sistema de archivos del contenedor en la
ruta <dest>                                  Podemos escribirlo de
COPY  <src> <dest>                           2 formas
COPY  ["<src>", "<dest>"]

Es parecido al comando $ cd, Indica el directorio base
de la aplicación y es en este directorio donde se
ejecutaran las instrucciones o layers que le siguen en
el DockerFile, esta pueden ser RUN, CMD,
ENTRYPOINT, COPY Y ADD
WORKDIR  <path>

Sirve para ejecutar comandos únicamente cuando
construimos (build) la imagen. Es decir solo se ejecutan
los comando cuando hacemos un $ docker build
RUN  <comando>
RUN  ["executable", "param1", "param2"]

Le dice a Docker que el contenedor escucha en los
puertos red especificados cuando corremos el
contenedor. Es decir cuando ejecutamos el comando
$ docker run. Por defecto el puerto escucha en TCP.
EXPOSE  <port> | <port/protocol>

Es el comando que se ejecutara por defecto cuando
corremos el contenedor. Es decir solo se ejecuta cuando
hacemos un $ docker run. Solo puede haber una
instrucción CMD en el DockerFile
CMD  <comando> <param1> <param2>
CMD  ["executable", "param1", "param2"]
CMD  ["param1", "param2"]

Funciona como la instruccion CMD, pero configura al
contenedor como si fuera un ejecutable
ENTRYPOINT  <comando> <param1> <param2>
ENTRYPOINT  ["executable", "param1", "param2"]

Funciona como la instruccion COPY, pero esta permite
traer o descargar archivos de una url y copiarlos en el
contenedor
ADD  <src> <dest>
ADD  ["<src>", "<dest>"]
```

```
$ docker build <options>  [ <path | ruta del proyecto | contexto build> | <url> ]
```
Para crear una imagen a partir de un DockerFile



Docker se trae todo lo de la imagen base y se la añade a la nueva imagen que se va a crear.

docker build -t ubuntu-platzi .


```
-t
```
Para asignar nombre y opcionalmente una etiqueta en el formato 'nombre:etiquete'

```
-f
```
Para especificar el nombre del DockerFile. Por defecto es 'PATH/DockerFile'

```
-f
```
Para especificar el nombre del DockerFile. Por defecto es 'PATH/DockerFile'

Networking

Los tipos de controladores o "drivers" de red que tenemos disponibles son: bridge, host (mi maquina), null

Bridge

Permite que los contenedores conectados a la misma red se comuniquen, al tiempo que proporciona aislamiento de los contenedores que no están conectados a esa red. Este controlador de Docker instala automáticamente reglas en la máquina host para que los contenedores en diferentes redes no puedan comunicarse directamente entre sí. Esta es la red por defecto.

Este tipo de controlares se suelen utilizar cuando sus aplicaciones se ejecutan en **contenedores independientes que necesitan comunicarse**.

host

Para utilizar la red de mi computadora. Básicamente se elimina el aislamiento de red entre el contenedor y la maquina host, es decir se usa la red del host directamente. Esto no se suele usar.

none

Para deshabilitar la red de un contenedor.

```
$ docker networking <command>
```
Para administrar los redes

```
$ docker networking create <options> <red-name>
```
Para crear una red

```
--attachable
```
Habilitar la conexion **manual** del contenedor. Basicamente permitimos que otros conectores se puedan unir a esta red.

```
-d
```
Para especificar el "driver" o controlador que administra la red. Por defecto es "bridge"

```
-o
```
Para colocar las configuraciones especificas del "driver" o controlador.

```
$ docker networking connect <options> <network-name> <container>
```
Para conectar un contenedor a una red.

```
$ docker networking disconnect <options> <network-name> <container>
```
Para desconectar un contenedor de una red.

```
$ docker networking inspect <options> <network-name>
```
Muestra información detallada de una o mas redes.

```
$ docker networking prune <options>
```
Para eliminar todas las redes que no se estan utilizando.

```
$ docker networking rm <network-name>
```
Para remover una o mas redes.

nota

Si dos contenedores están conectados en la misma red, pueden verse entre si utilizando como **hostname** el nombre del contenedor.

Docker-compose (todo en uno)

Es una herramienta que nos permite describir de forma **declarativa** la arquitectura de nuestra aplicación, utiliza composefile (docker-compose.yml).

```
$ docker-compose <options> <command>
```
```
$ docker-compose up <options> <command>
```
Construye, (re)crea, inicia y conecta contenedores a un servicio.

```
-d
```
Detached mode: Para ejecutar el contenedor en segundo plano.

```
--scale SERVICE=NUM
```
Para escalar un servicio especificando el numero de contenedores.

```
$ docker-compose down <options> <services>
```
Detiene los contenedores y elimina los contenedores, redes, volúmenes y imágenes.

```
-v
```
Para eliminar los volúmenes con nombre en la sección 'volumes' en el archivo de docker compose y también los volúmenes anónimos atados (bind) al contenedor.

```
--rmi type
```
Para remover imágenes. Se debe definir el tipo:
'all': Para remover todas las imágenes usadas por cualquier servicio.
'local': Para remover solo las imagen que no tiene un custom tag

```
$ docker-compose ps <options> <services>
```
Lista los contenedores.

```
-q
```
Para mostrar únicamente las IDs

```
-a
```
Para mostrar todos los contenedores detenidos. Incluidos los contenedores que se crearon por el comando **run**

```
$ docker-compose build <options> <services>
```
para construir o reconstruir servicios. Los servicios que se crean por primera vez, se crean y luego se etiquetan como 'project_service'. Si cambia el 'Dockerfile' de un servicio o el contenido de su directorio de compilación, puede ejecutar 'docker-compose build' para reconstruirlo.

NOTAS

Luis Andrés Villegas Sanchez

$ docker-compose logs <options> <service>
Para ver la salida (output) de los contenedores.

-f → Para darle un seguimiento continuo a la salida (output) del contenedor.

$ docker-compose run <options> <service> <command>
Para ejecutar un comando en un servicio nuevo.

$ docker-compose exec <options> <service> <command>
Ejecuta un comando en un contenedor que se encuentra corriendo o esta activo (running).

$ docker-compose scale <options> <service>=<num>
Para establecer el número de contenedores que se ejecutarán para un servicio. Esto ya no se usa.

nota
De forma predeterminada, Compose configura una única red para su aplicación. Cada contenedor de un servicio se une a la red predeterminada y es accesible para otros contenedores en esa red y detectable por ellos con un nombre de host idéntico al nombre del contenedor, al nombre del servicio o alias.

docker-compose.yml

El archivo Compose es un archivo YAML que define servicios , redes y volúmenes . La ruta predeterminada para un archivo de redacción es ./docker-compose.yml.

El formato del archivo esta basado en la versión 3.8

Esto se refiere al formato del archivo compose, dependiendo de este valor tendremos disponibles algunas propiedades y otras no.

versión: '<number>'

Aquí definimos todos los servicios que utilizara nuestra aplicación. Cada servicio contiene toda la configuración que será aplicada a cada contenedor.
Es parecido a realizar **$ docker run**

services:

Servicio llamado web
web:

Para asignar un nombre de contenedor personalizado, en lugar de un nombre predeterminado generado.

Debido a que los nombres de contenedores de Docker deben ser únicos, no puedes escalar un servicio más allá de 1 contenedor si ha especificado un nombre personalizado.

container_name: <name>

Especificamos la imagen con el que crearemos el contenedor. Puede ser un repositorio/etiqueta o la ID de la imagen. Si la imagen no existe, se descargara de https://hub.docker.com/, a menos que se haya especificado la propiedad **build** en el servicio, en este caso la compila usando las opciones especificadas en el Dockerfile y le coloca la etiqueta <image-name> de la propiedad **image**

image: <image-name>

Aquí definimos la ruta del Dockerfile o el contexto que se utilizaran al momento de crear el contenedor.

Si especificamos la propiedad image, entonces nombra la imagen construida con el valor de esta propiedad. Opcioanlemte le podemos poner el tag

build: <path-context> Por lo general siempre colocamos "build: ."

El **contexto** es una ruta, directorio, url o repositorio de git que contenga el archivo Dockerfile.

```
build:
  context: /dir
  dockerfile: Dockerfile-alternativo   Por si queremos usar un Dockerfile diferente
  args:
    variable: 1
    test: hello
```

Argumentos de compilación, son variables de entorno accesibles solo durante el proceso de compilación. Esta variables deben estar agregadas en el Dockerfile.

Para agregar las variables de entorno que usara el servicio. Cualquier valor booleano (true, false) debe incluirse entre comillas

```
enviroment:
  SESSION_KEY: a2bc23
  - SESSION_KEY=a2bc23
```

Parara definir las dependencias entre los servicios. Esto permite que los servicios definidos aquí se inicien antes. En este ejemplo, **db** se inicia antes que **web**. Sin embargo depends_on no espera a que db este "listo" antes de comenzar web, solo espera hasta que db este iniciado.

```
depends_on:
  - db
```

Para exponer puertos a la maquina host. La asignación de puertos es incompatible con la propiedad **network_mode:** host

```
ports:
  - "3000:3000"   HOST:CONTAINER
  - "3000-3010:3000"   Para asignar rangos de puertos en el host
  - "3000-3010:8080-9000"
  - "3000:3000/udp"   Para asignar el protocolo. Por defecto es tcp
```

```
ports:
  target: 80   Puerto del contenedor
  published: 8080   Puerto expuesto públicamente o el de la maquina host
  protocol: tcp   El protocolo del puerto (tcp o udp)
  mode: host
```

Exponga los puertos sin publicarlos en la máquina host; solo serán accesibles para los servicios vinculados o linkeados.

```
expose:
  - "3000"
```

Son las redes a las que se unirán los contendores. De forma predeterminada se crear una red para la aplicación y los contenedores se unen a ella. Esta red recibe un nombre basado en el nombre del proyecto.

```
networks:
  - red-name
```

Para montar alguna ruta del host (Bind mounts) o volúmenes. Si desea reutilizar un volumen en varios servicios, se debe definir un volumen en el nodo de nivel superior.

```
volumen:
  - "/data/db:/data/db:ro"   <source>:<target>:<mode>
```

donde **SOURCE** puede ser una ruta de host o un nombre de volumen. **TARGET** es la ruta del contenedor donde se monta el volumen. Los modos estándar son **ro** solo lectura y **rw** lectura-escritura (predeterminado).

```
volumen:
  - type: bind   El tipo de montaje volume, bind, tmpfs o npipe
    source: /data/db   Una ruta en el host para un montaje de enlace o el nombre de un volumen definido en el nodo de nivel superior.
    target: /data/db   La ruta en el contenedor donde se monta el volumen
  - type: volume
    source: my-data
    target: /data
```

Servicio llamado db
db:

```
  image: mongo

networks:
  red-name:

volumes:
  my-data:
```

Referencias

https://docs.docker.com/compose/compose-file/

NOTAS

Luis Andrés Villegas Sanchez