



Docker: the gory details

Leandro Kollenberger

redbee studios

30 de Marzo de 2018

Por qué presenciar esta charla

- Sos desarrollador, usás Docker hace tiempo en tu día a día pero te interesa saber algo más de detalle.
- Sos de ops, pero nunca tocaste Docker a fondo.
- Te interesó la charla.
- Hay comida.

¿Qué es Docker?

Docker es una herramienta que nos facilita empaquetar una aplicación junto con su *runtime* dentro de una **imagen**, que luego se ejecutará dentro de un **container**.

Gracias a la gran cantidad de **imágenes base** que hay disponibles en el **Docker Hub** y la simple sintaxis del **Dockerfile**, crear imágenes propias con nuestra aplicación es muy simple.

Dockerfile

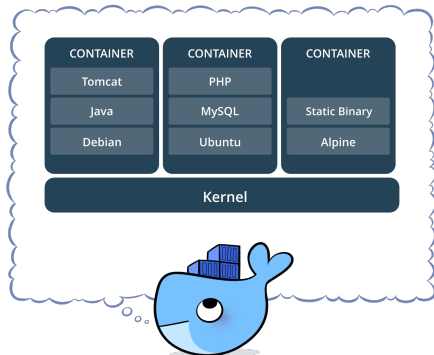
```
FROM debian:9
RUN echo "echo 'Hola mundo!'" \
    > /hi.sh && \
    chmod +x /hi.sh
ENTRYPOINT hi.sh
```

¿Qué es un container?

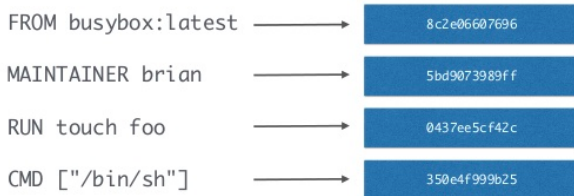
Un **container** es una funcionalidad del sistema operativo que nos permite ejecutar ciertas aplicaciones dentro de un **namespace** aparte.

Un **namespace** es una separación lógica de los recursos del sistema, como por ejemplo CPU, memoria RAM, puntos de montaje (disco), árbol de procesos, network, etc.

De esta forma un container nos provee seguridad y aislamiento de las aplicaciones que corren en él, similar a lo que nos permite una VM, sin el overhead de simular hardware y correr un sistema operativo entero, ya que todos los containers corriendo en un mismo host comparten su kernel.



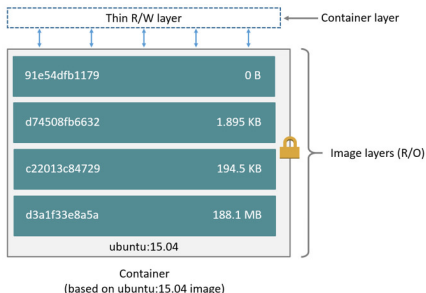
El concepto de "layer"



Las imágenes de Docker que podemos usar a diario están compuestas por una o más **layers**. Éstas imágenes contienen un sistema de archivos correspondiente al sistema operativo o la aplicación que queremos ejecutar. Esta imagen está identificada por un hash y opcionalmente uno o más **tags**, como por ejemplo *redis:4.0.6* o *debian:latest*.

Cada layer contiene únicamente los cambios (ya sean archivos, comandos o metadatos) con respecto a la layer anterior, se identifican con un hash y corresponden cada una a un comando del Dockerfile. Todas las layers que componen una imagen son **read only**.

El concepto de "layer"

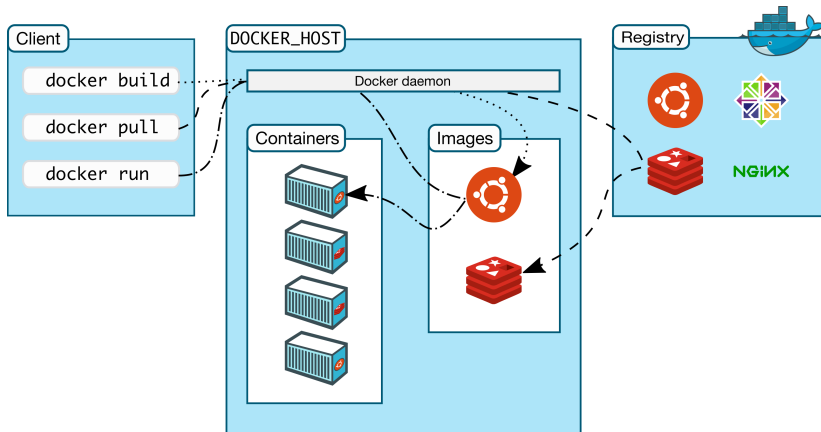


Cada layer contiene únicamente los cambios (ya sean archivos, comandos o metadatos) con respecto a la layer anterior, se identifican con un hash y corresponden cada una a un comando del Dockerfile. Todas las layers que componen una imagen son **read only**.

Al momento de crear un nuevo container a base de una imagen, se crea una pequeña layer **read-write**, que contiene los cambios creados por el container en ejecución (por ejemplo, por la aplicación).

Esta última layer se **pierde al eliminar el container**.

Arquitectura de Docker



Arquitectura de Docker

- **Cliente:** Aplicación de consola que se comunica con el daemon. Puede comunicarse tanto con el daemon de la máquina local (por socket de Unix) como con una remota (por TCP). Ejemplo: `docker run`, `docker image ls`.
- **Daemon:** Servicio de Docker corriendo en background que gestiona imágenes y containers.
- **Registry:** Repositorio remoto de imágenes. En detalle, almacena tres tipos de datos:
 - **Layers** (cada una con su hash y el comando de Dockerfile que la generó).
 - **Imágenes**, básicamente una lista de hashes de cada una de las layers que la componen, más metadatos y un hash propio de la imagen.
 - **Tags** (human readable), que apuntan a hashes de imágenes. Son sobrescribibles. Ejemplos: `redis:4.0.6`, `jess/routersploit` o `docker.dev.redbee.io/ms-lalala-rest:release-v1.2.1`. Pueden referirse a la library (imágenes oficiales, primer caso), imágenes dentro del Docker Hub pero de terceros (segundo caso) u otras registries privadas (tercer caso). Si no se indica una versión por defecto es `latest`.



Gracias!
Preguntas?