Docker: the nice and the gory

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 enpaquetar una aplicación junto con su *runtime* dentro de una **imagen**, que luego se ejecutará dentro de un **contenedor**.
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
```

The nice about Docker

Usando la CLI

Comandos útiles para manejar Docker

- `docker run`: crear un container. Optiones útiles:
  - `--rm`: borrar el container cuando se detenga.
  - `-i`: *interactive*, habilita la salida estándar del container.
  - `-t`: habilita la entrada de teclado al container, generalmente usado junto con `-i`.
  - `-p puertoexterno:puertointerno`: *publish*, publica (mapea) un puerto externo en el host a un puerto dentro del container, por ejemplo `-p 8080:80` mapea las conexiones entrantes en el puerto 8080 del host al 80 del container.
  - `--name=lalala` *self-explainatory*.
  - `-v /path/en/el/host:/path/dentro/del/container` *volume*, mapea un volúmen de docker o path en el host a un path dentro del container. Sirve también para archivos y se le puede agregar `:ro` para indicar *read-only*.

Usando la CLI

- `docker ps`: Listar containers corriendo (con `-a` lista también containers detenidos)
- `docker start/stop`: Arrancar/detener containers.
- `docker images`: Listar imágenes disponibles en el sistema.
- `docker rmi`: Borrar imágenes.
- `docker network`: Listar/borrar/crear una network administrada por Docker.
- `docker exec -it container comando`: Ejecutar `comando` dentro de `container`. Sirve para "meterse" dentro y hacer cambios.

Docker Compose

Hermoso.
Es un archivo YAML que describe containers a crear.
Permite comunicación entre containers con nombre de DNS y sin linkear.
Sintaxis:
https://docs.docker.com/compose/compose-file/

```
version: "2"
services:
  nginx:
    image: nginx:latest
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    networks:
      - web
    volumes:
      - /data/ssl:/ssl:ro
      - /data/nginx.conf:/data/nginx.conf:ro
  nextcloud:
    image: nextcloud:apache
    restart: unless-stopped
    volumes:
      - /data/nextcloud:/var/www/html/data
    expose:
      - "80"
    networks:
      - web
    depends_on:
      - nginx
      - nextcloud-db
  nextcloud-db:
    image: postgres:latest
    restart: unless-stopped
    environment:
      - "POSTGRES_PASSWORD=******"
      - "POSTGRES_USER=nextcloud"
      - "POSTGRES_DB=nextcloud"
    volumes:
      - /data/nextcloud-db:/var/lib/postgresql/data
    expose:
      - "5432"
    networks:
      - web
```

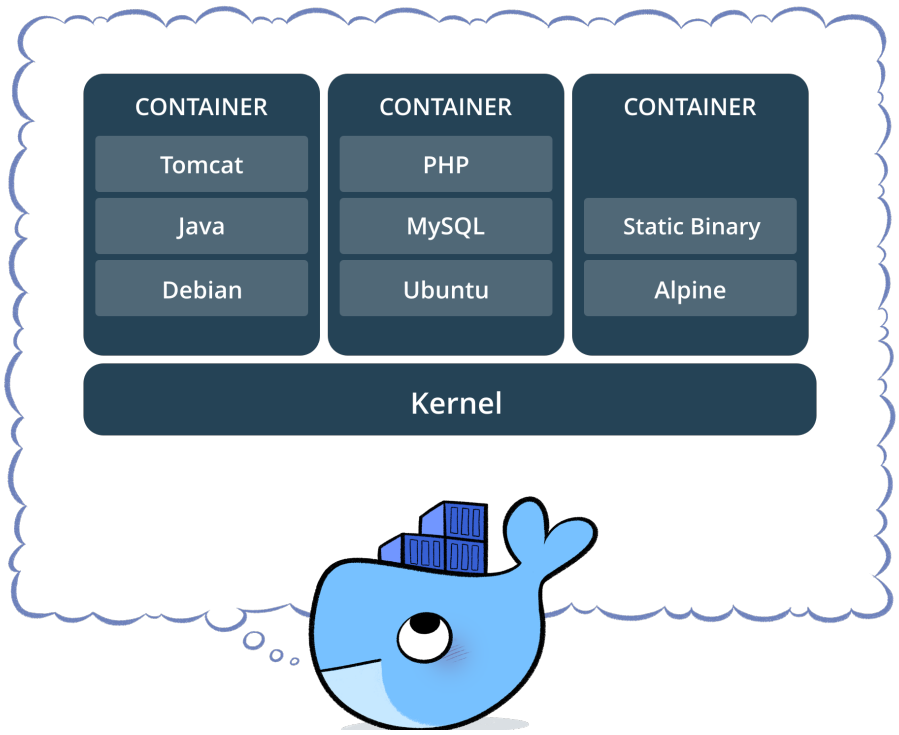
The **gory** about Docker

¿Qué es un container?

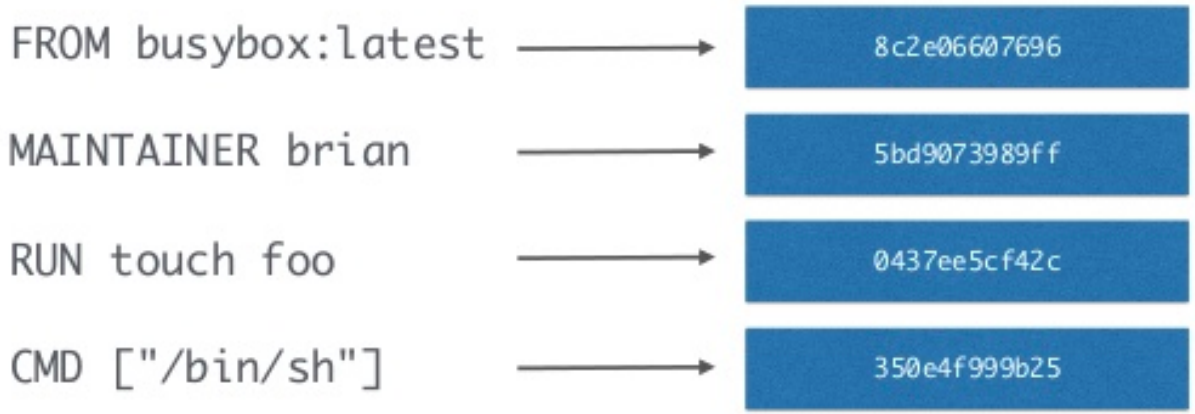
Un **container** es una funcionalidad del sistema operativo que nos permite ejecutar ciertas aplicaciones dentro de un **namespace** aparte.

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.



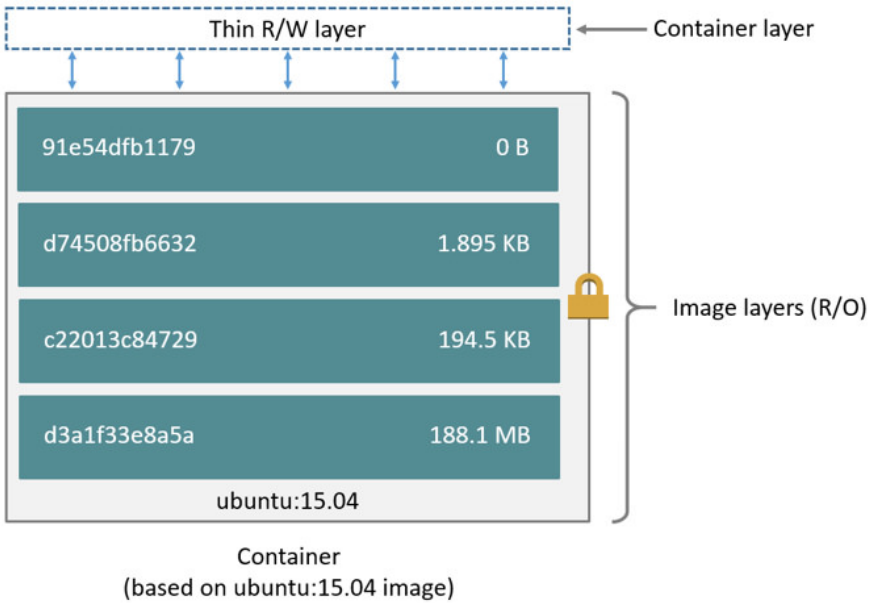
El concepto de "layer"



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"

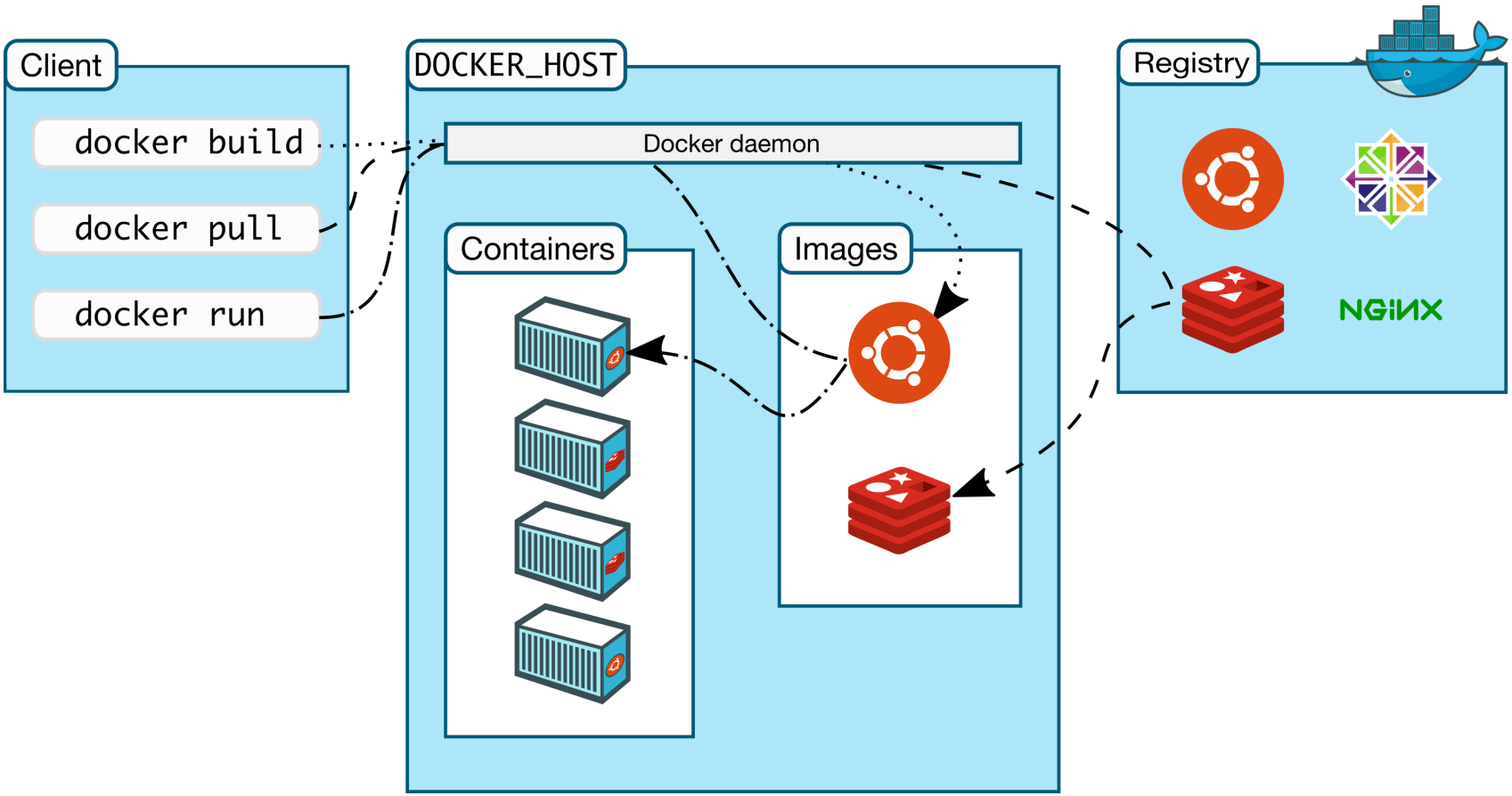


Cada layer contiene únicamente los cambios (ya sean archivos, comandos o metadatos) con respecto a la layer anterior, se identifican con un hash y corresponden cada una a un comando del Dockerfile. Todas las layers que componen una imagen son **read only**.

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 backgroud 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 sobreescribibles. 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`.

Namespaces

El *kernel* Linux nos permite aislar los siguientes subsistemas en namespaces:

- **mnt**, Mount: puntos de montaje y sistema de archivos.
- **pid**, Process ID: árbol de procesos.
- **net**, Network: placas de red, tablas de routeo, firewall...
- **ipc**, Inter-process Communication: colas, pilas, sockets...
- **uts**: hostname, domain name.
- **user**: IDs de usuario y grupo.

Networking en containers

Las aplicaciones en general publican servicios, cómo accedemos si estamos dentro de un container?

- `net=host`: desactivamos el namespace de networking, comparte puertos y servicios con las aplicaciones comúnes del host.
- `net=bridge`: ponemos un switch virtual en el host, placas de red dentro del container, las conectamos ahí y abrimos puertos en el host. Funciona parecido a un router hogareño: comparte una IP (la del host) y forwardea puertos a donde corresponda. Ejemplo: `docker run -d -p 1234:80 nginx` levanta un container de nginx y forwardea conexiones entrantes al host en puerto 1234 al puerto 80 del container.
- `net=none`: desactiva la red completamente, *duh.*

Storage drivers (*a.k.a. La magia detrás de las layers*)

Los storage drivers (también conocidos como *graph drivers*) son drivers del kernel Linux que permiten montar las layers una encima de otra, agregando cada una sus respectivos cambios. Opciones:

- `overlay2`: Recomendada, es la más moderna.
- `overlay`: Recomendada (si el kernel no soporta `overlay2`).
- `aufs`: Vieja, se usaba antes de que existiera `overlay`). No recomendada.
- `btrfs`
- `devicemapper`:
  - `direct-lvm`: Recomendada en caso de tener LVM con thinpool.
  - `direct-loop`: No recomendada, lentísima.

Más allá de Docker

Los containers son divertidos, pero complejos de administrar!

- Kubernetes.
- Apache Mesos.
- Hashicorp Nomad.
- Docker Swarm.

Gracias!
Preguntas?

redbee®