Docker

Juanda

Mayo 2015

Contents

1 Taller sobre Docker **7**

1.1 Estructura Taller 7

2 Introducción **9**

2.1 ¿Qué es docker? 9

2.2 Definiciones de docker 9

2.3 Docker es un entorno chroot 10

2.4 Docker es un contrato entre el sysadmin y el desarrollador 10

2.5 ¿Cómo es un administrador de sistemas? 10

2.6 ¿Cómo es un desarrollador? 11

2.7 Instalación de paquetes para un sysadmin 11

2.8 Gestión de paquetes para un desarrollador 12

2.9 ¿Qué piensa el sysadmin del desarrollador? 12

2.10 ¿Qué piensa el desarrollador del administrador? 12

2.11 Docker es un contrato entre el sysadmin y el desarrollador 13

2.12 Docker es un empaquetador de aplicaciones 13

2.13 Docker es un sistema de virtualización 14

2.14 Virtualización tradicional 15

2.15 Virtualización en Docker 16

2.16 Comparativa características 16

2.17 Tamaño imágenes de Docker 16

2.18 Tiempo de arranque 17

2.19 Integración 17

2.20 Otras ventajas 17
2.21 Historia y futuro 18
2.22 Google Trends 18
2.23 Quien usa docker 19

3 Instalación de Docker **21**

3.1 Requerimientos 21
3.2 Instalación en Ubuntu 14.04 22
3.3 Actualización 22

4 Docker Hub **23**

4.1 ¿Qué es el docker hub? 23
4.2 Registro/login en Docker Hub 23
4.3 Búsqueda de imágenes 23
4.4 ¿Y si utilizamos nuestro propio registro? 24
4.5 Prueba de acceso a nuestro registro 24
4.6 Configuración de docker para acceder al registro 25
4.7 Imágenes en nuestro repositorio 25

5 Trabajar con contenedores **27**

5.1 Ejecución de un contenedor 27
5.2 Contenedores en ejecución 27
5.3 Listado de contenedores 28
5.4 Dar nombre a los contenedores 28
5.5 Comunicación con un contenedor 29
5.6 Parar un contenedor 29
5.7 Ejecutar un contendor en background 29
5.8 Borrado de un contenedor 29

6 Trabajar con imagenes de Docker **31**

6.1 Imágenes en local 31

6.2 Los containers no guardan datos 31

6.3 run vs start 32

6.4 Crear mi imagen: conceptos básicos 32

6.5 Crear una imagen de forma manual 33

6.6 Commit de la imagen 33

6.7 Comandos para un dockerfile 33

6.8 Crear una imagen mediante script 34

6.9 Borrado de imagenes 35

7 Ejemplo para Wordpress: **37**

7.1 Elección de imágenes 37

7.2 Creación de containers 37

7.3 Prueba de funcionamiento 38

8 Bibliografía **39**

8.1 Bibliografía 39

Chapter 1

Taller sobre Docker

1.1 Estructura Taller

Explicación general sobre Docker

Instalación

Configuración Registro de imágenes docker del CPIFP Los Enlaces y descarga de imágenes

Práctica con Docker

Hay poco tiempo, pero Docker es muy eficiente :-)

Chapter 2

Introducción

2.1 ¿Qué es docker?

Vamos a intentar aclararlo antes de empezar el taller

2.2 Definiciones de docker

Un entorno chroot

Un contrato entre desarrolladores y administradores de sistemas

Un empaquetador de aplicaciones

Un sistema de virtualización

2.3 Docker es un entorno chroot

chroot se utiliza normalmente para conexiones ftp o ssh

El usuario no ve cierta parte de la máquina, por ejemplo a otros usuarios.

En docker se **enjaula por las dependencias entre paquetes**, no para aislar usuarios

2.4 Docker es un contrato entre el sysadmin y el desarrollador

Una aplicación se hace por piezas, como las televisiones y los ordenadores

Pero el ensamblaje se hace más de una vez:

- En fase de desarrollo
- En fase de testing
- En fase de producción

Y **no todos las ensamblan de la misma manera** :-(

2.5 ¿Cómo es un administrador de sistemas?

Le preocupa la estabilidad de SU máquina

- Quiere usar lo de siempre
- *Si algo funciona, para que cambiarlo*

Tiene un tick y ejecuta a menudo comandos del tipo:

- **top**
- **who**

- **tail -f /var/log/syslog**
- **uptime**
- ...

2.6 ¿Cómo es un desarrollador?

Le preocupa la funcionalidad de sus aplicaciones

Quiere usar lo último:

- **Node.js**
- **Rust**
- **Go**
- **Microservices**
- **Cassandra**
- **Hadoop**

2.7 Instalación de paquetes para un sysadmin

Instala paquetes a nivel de S.O.

- Se instalan de manera global, para todo el sistema

Debe ser rígido y cuidadoso con las versiones:

```
# apt-get install xxxx
...
You might want to run 'apt-get -f install' to correct these:
The following packages have unmet dependencies:
xxxxxxx xxxxx
```

2.8 Gestión de paquetes para un desarrollador

Trabaja en varios proyectos

Utilizan sus propios gestores de paquetes

Instalación de paquetes

- Instalación global, para todo el sistema. Raro, raro, raro.
- Instalación local, específica para cada proyecto
  - En Python podría ser mediante pip dentro de un entorno virtualenv
  - En Ruby mediante RubyGems
  - nodejs con su npm
  - bower para desarrollo en frontend
  - ....

2.9 ¿Qué piensa el sysadmin del desarrollador?

El desarrollador quiere entrar en SU máquina

Y además le quiere tocar SUS paquetes

2.10 ¿Qué piensa el desarrollador del administrador?

Un rarito: no quiere compartir su máquina

El último obstáculo para llegar a la meta

2.11 Docker es un contrato entre el sysadmin y el desarrollador

El administrador solo se debe encargar del despligue de los containers

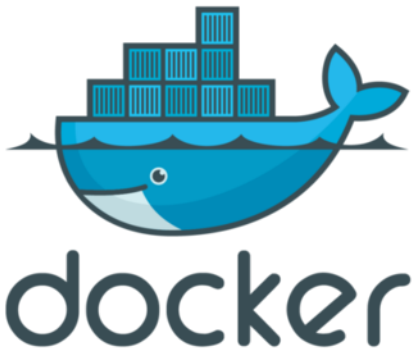
El desarrollador puede hacer lo que le venga en gana, siempre y cuando lo meta en un container

2.12 Docker es un empaquetador de aplicaciones

Crea un container para la app de modo que se ejecuten igual en distintas máquinas

Build, Ship and Run Any App, Anywhere

Podríamos pensar en un container como una *máquina virtual sin SO propio*



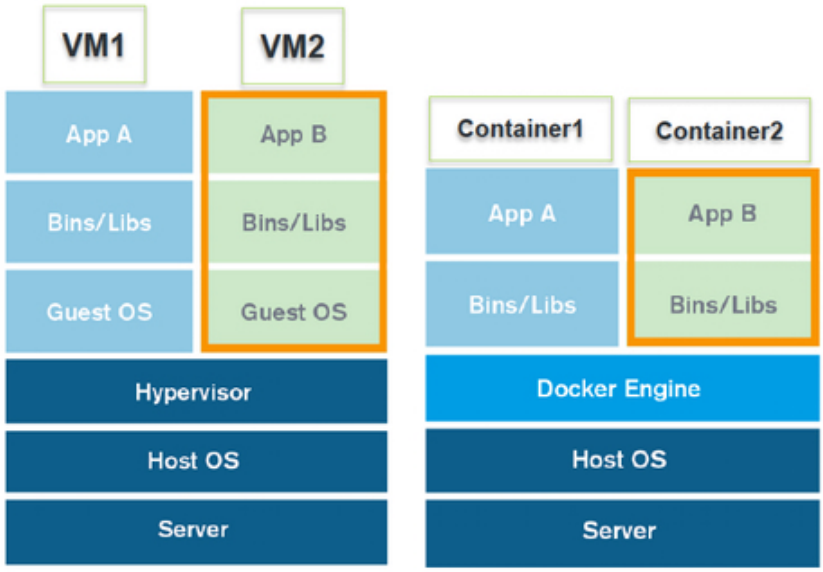
2.13 Docker es un sistema de virtualización

El Host debe ser Linux

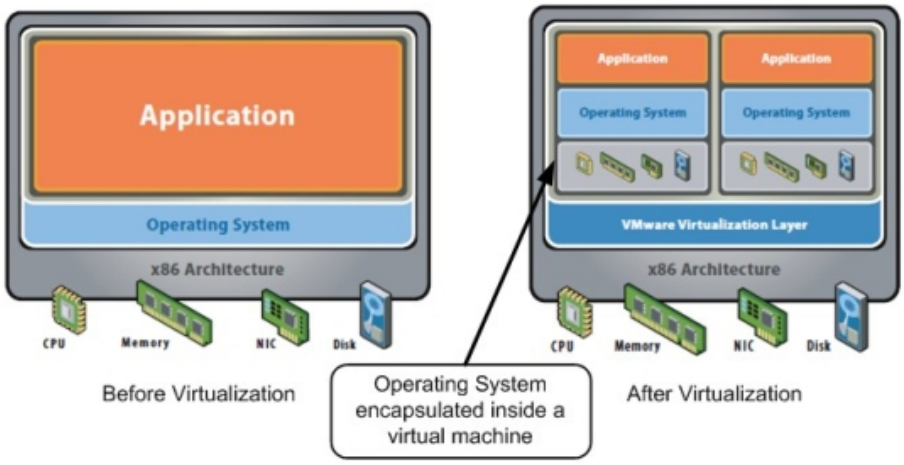
Podríamos hablar de una máquina virtual ligera

Los sistemas de virtualización tradicionales intentan optimizar/adelgazar la capa común (Host OS y Hypervisor).

- Host OS e Hypervisor pueden ser una única capa



2.14 Virtualización tradicional



Un sistema completamente virtualizado obtiene su propio conjunto de recursos (cpu, ram, disco o red)

2.15 Virtualización en Docker

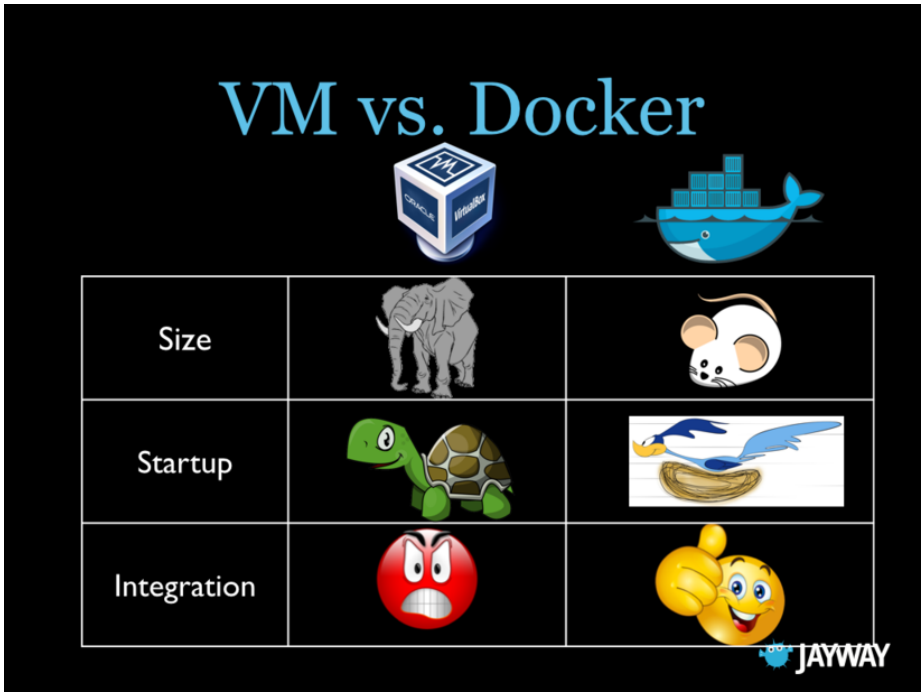
El nivel de aislamiento es menor:

- Los procesos que se ejecutan en cada container se pueden ver desde el host:

```
pstree Docker
```

Consume menos recursos y es más ligero (¡un único sistema operativo!)

2.16 Comparativa características



2.17 Tamaño imágenes de Docker

Las máquinas virtuales ocupan bastante (varios GBytes)

- No son lo más práctico para almacenaje
- Menos todavía para transferencia

Las imágenes de docker ocupan menos de 1GByte

```
$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             VIRTU
mongo                           latest              5242d1368ac4        9 days ago          255.9
node                            latest              f709efdf393f        9 days ago          710.9
mysql                           5                   56f320bd6adc        2 weeks ago         282.9
mysql                           latest              56f320bd6adc        2 weeks ago         282.9
wordpress                       4.1.2-fpm           e83a380ff338        2 weeks ago         432.9
wordpress                       apache              4cde06295aba        2 weeks ago         459.8
ubuntu                          14.04.2             b7cf8f0d9e82        2 weeks ago         188.3
ubuntu                          latest              b7cf8f0d9e82        2 weeks ago         188.3
ubuntu                          trusty              b7cf8f0d9e82        2 weeks ago         188.3
itzg/minecraft-server           latest              94759ec42cdd        5 weeks ago         397.7
```

2.18 Tiempo de arranque

El tiempo de arranque de una máquina virtual se mide en minutos.

El tiempo de arranque de un container a menudo es inferior a 1 segundo

- Cuesta lo mismo crear un container nuevo que reiniciar la aplicación del container que se ha quedado colgada
- Se introduce un concepto nuevo, containers de un solo uso

2.19 Integración

Para integrar máquinas virtuales en un host, debemos establecer la red.

Mediante Docker la integración de containers es directa.

2.20 Otras ventajas

Como consumen tan pocos recursos, podemos generar **un container por cada servicio**:

- bbdd
- servidor web
- servicio de caché

- servicio de backup

Al ser menos específicos, se vuelven más **reusables**

Al ser reusables, **se comparten mediante repositorios**

Las arquitecturas de las aplicaciones se vuelven más dinámicas y testeables

- Podemos cambiar nuestro servidor web de Apache a Nginx en cuestión de segundos.

2.21 Historia y futuro

Antes de las VM:

- Por cada aplicación se utilizaba un servidor
  - Se cumplian las dependencias de las aplicaciones
  - Un fallo en una aplicación no afectaba a otra
  - El 95% del tiempo el servidor estaba ocioso.

Con las VM:

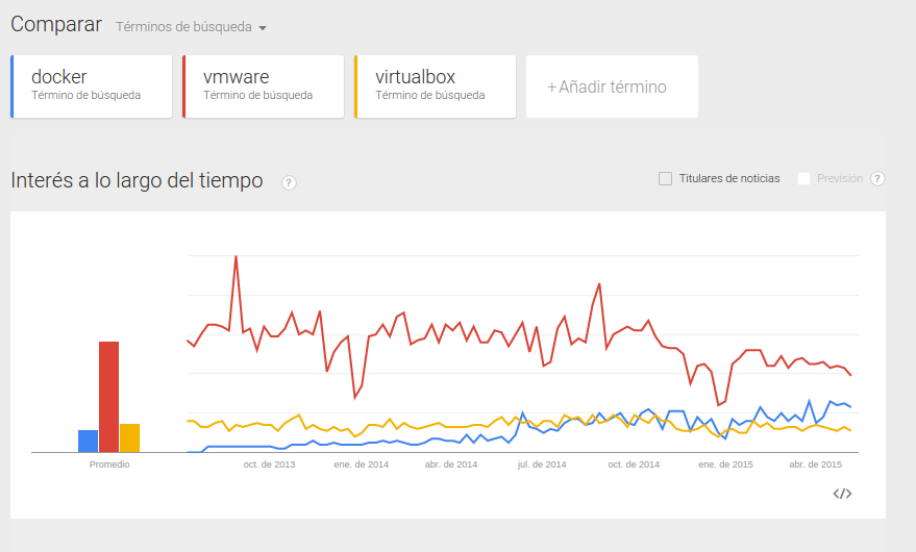
- Se usa un servidor en el que se instala una máquina virtual por servicio:
  - Se optimiza el uso del servidor
  - Aún así necesitamos una máquina virtual por aplicación para dar estabilidad y cumplir dependencias.

Con containers

- Evitamos tener que usar una máquina virtual (SO) por servicio

2.22 Google Trends

Tendencias de búsqueda en Sillicon Valley desde Julio 2013 hasta ahora



2.23 Quien usa docker

Todavía pocos

Empresas pioneras en Internet como:


Bai du 百度

Chapter 3

Instalación de Docker

3.1 Requerimientos

Windows o Mac

- Hace falta virtualizar :-(
- Pero es rápido :-)
  - Se usa boot2docker
  - Es una distribución de Linux ligera: 24MB RAM con arranque en ~5s

Linux (Ubuntu):

- 64 bits
- Kernel 3.10 o superior
- Perfecto en 14.04 (Trusty)

¿Cuál es la versión de mi kernel?

```
$ uname -r
3.13.0-51-generic
```

3.2 Instalación en Ubuntu 14.04

Instalamos los paquetes necesarios:

```
$ sudo apt-get update
$ wget -qO- https://get.docker.com/ | sh
```

Comprobamos que se esté ejecutando:

```
$ sudo service docker status
```

Configuramos docker para poderlo usar sin usuario root:

```
$ sudo usermod -aG docker administrador
```

3.3 Actualización

Nos interesa la versión 1.6 que tiene cambios significativos (Abril 2015)

Comprobamos que la versión que tenemos ahora sea actual:

```
$ docker version
Client version: 1.6.0
Client API version: 1.18
Go version (client): go1.4.2
Git commit (client): 4749651
OS/Arch (client): linux/amd64
Server version: 1.6.0
Server API version: 1.18
Go version (server): go1.4.2
Git commit (server): 4749651
OS/Arch (server): linux/amd64
```

¿Cómo actualizar?

```
$ wget -N https://get.docker.com/ | sh
```

Otra opción:

```
$ sudo su
# service docker stop
# curl -sSL https://test.docker.com/ubuntu | sh
```

Chapter 4

Docker Hub

4.1 ¿Qué es el docker hub?

Es un repositorio para descargar imágenes (nuestras o de otros)

Es un reporitorio donde subir nuestras imágenes (públicas o privadas)

Tiene servicios automatizados (webhooks)

Se integra con GitHub y BitBucket

4.2 Registro/login en Docker Hub

El registro no es necesario para descargarse imágenes (**push**)

El registro es necesario para subir imágenes (**pull**)

Nos podemos registrar vía consola, más rápido que por web:

```
$ docker login
```

4.3 Búsqueda de imágenes

Mediante línea de comandos, por ejemplo **docker search ubuntu**

Vía web:

- https://registry.hub.docker.com/>

Normalmente utilizaremos repositorios oficiales (más garantías)

4.4 ¿Y si utilizamos nuestro propio registro?

Lo necesitamos para hacer pruebas desde aula ya que el ancho de banda no es suficiente.

¿Cómo configurar un registro privado?

Nuestro registro está configurado en la máquina **dockerreg.inf.enlaces**

- Comprobamos que nuestra máquina resuelve la Ip del registro: **dockerreg.inf.enlaces**
  - Para poder trabajar con nuestro registro, necesitamos como DNS la IP 172.30.160.254
  - Otra opción es añadir la entrada en el fichero */etc/hosts*

4.5 Prueba de acceso a nuestro registro

Intentamos descargar una imágen de nuestro registro:

```
$ docker pull dockerreg.inf.enlaces:5000/ubuntu
```

La cosa no va bien:

```
FATA[0000] Error response from daemon: v1 ping attempt failed with error:
Get https://dockerreg.inf.enlaces:5000/v1/_ping: tls: oversized record
received with length 20527. If this private registry supports only HTTP
 or HTTPS with an unknown CA certificate, please add
‘--insecure-registry dockerreg.inf.enlace:5000‘ to the daemon’s arguments.
In the case of HTTPS, if you have access to the registry’s CA certificate,
no need for the flag; simply place the CA certificate at
/etc/docker/certs.d/dockerreg.inf.enlaces:5000/ca.crt
```

Nuestro registro privado de momento es muy básico:

- No se puede consultar (*docker search*)
- No tiene certificados (acceso sin https)

4.6 Configuración de docker para acceder al registro

Habilitamos las conexiones inseguras en /etc/defaul/docker mediante esta línea:

```
DOCKER_OPTS="--insecure-registry dockerreg.inf.enlaces:5000"
```

Reiniciamos docker:

```
$ sudo service docker restart
```

Probamos su funcionamiento

```
$ docker pull dockerreg.inf.enlaces:5000/ubuntu
```

Podemos comprobar que ya tenemos esa imagen almacenada en local

```
$ docker images
```

4.7 Imágenes en nuestro repositorio

Las justas para hacer la demo:

```
administrador@dockerreg:~$ docker images
REPOSITORY                              TAG                 IMAGE ID            CREATED
wordpress                               latest              5ff368875b77        2 days ago
dockerreg.inf.enlaces:5000/wordpress    latest              5ff368875b77        2 days ago
registry                                2.0                 2971b6ce766c        9 days ago
dockerreg.inf.enlaces:5000/ubuntu       latest              07f8e8c5e660        10 days ago
ubuntu                                  latest              07f8e8c5e660        10 days ago
dockerreg.inf.enlaces:5000/mysql        latest              56f320bd6adc        2 weeks ago
mysql                                   latest              56f320bd6adc        2 weeks ago
```

Observa que están duplicadas

Cambiamos su repositorio para que estén accesibles de forma privada.

Chapter 5

Trabajar con contenedores

5.1 Ejecución de un contenedor

```
$ docker run dockerreg.inf.enlaces:5000/ubuntu /bin/echo 'Hello world'
```

Si la imagen (ubuntu:14.04) no existe, se descargará de forma automática

Una vez descargada, se introduce en un contenedor y se ejecuta el comando echo.

Utilizamos nuestro registro mediante **dockerreg.inf.enlaces:5000/ubuntu**
pero podríamos usar el de docker y escribir simplemente:

```
$ docker run ubuntu /bin/echo 'Hello world'
```

5.2 Contenedores en ejecución

Mediante el comando:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

¿Nuestro contenedor no aparece?

- Solo aparecen los contenedores en ejecución
- El contenedor ha hecho el comando echo y se ha parado.

5.3 Listado de contenedores

Si queremos ver todos los contenedores:

```
$ docker ps -a
CONTAINER ID        IMAGE                         COMMAND               CREATED
0e487f71a145        ubuntu:14.04                  "/bin/bash"           2 days ago
```

Suele ser útil en ocasiones ver el último contenedor creado, mediante el comando **docker ps -l** (last)

5.4 Dar nombre a los contenedores

Docker da un nombre a cada contenedor que arranca:

```
$ docker ps -a
CONTAINER ID        IMAGE                         COMMAND                CREATED
41fce3d709d1        mongo:latest                  "/entrypoint.sh /bin   2 days ago
a51e9bf35ebe        mongo:latest                  "/entrypoint.sh mong   2 days ago
0e487f71a145        ubuntu:14.04                  "/bin/bash"            2 days ago
97579d465b6e        itzg/minecraft-server:latest  "/start"               2 days ago
76fb784e7ce2        itzg/minecraft-server:latest  "/start"               2 days ago
d8a4d884ca3d        itzg/minecraft-server:latest  "/start"               2 days ago
3c950e258972        itzg/minecraft-server:latest  "/start"               2 days ago
35c33d510980        secure_registry:latest        "registry cmd/regist   3 days ago
aa87743bd795        secure_registry:latest        "registry cmd/regist   3 days ago
```

Para referirnos a los contenedores podemos utilizar:

- Su id (en realidad es más largo de lo que aquí aparece)
- Parte de su id
- Su nombre

Podemos personalizar el nombre de los contenedores mediante el **parámetro –name** cuando lo arranquemos:

```
$ docker run --name miubuntu ubuntu:14.04 /bin/echo 'Hello world'
```

5.5 Comunicación con un contenedor

Podemos arrancar el contenedor y acceder vía terminal:

```
$ docker run -t -i --name micontenedor ubuntu:14.04 /bin/bash
```

La salida estandar del contenedor se manda a consola:

```
$ docker run --name holaMundo ubuntu:14.04 /bin/bash -c "while true; do echo Hola Mundo; sleep 1;
```

5.6 Parar un contenedor

Lo pararemos mediante *docker stop* :

```
$ docker stop holaMundo
```

5.7 Ejecutar un contendor en background

El caso anterior hubiera sido más útil ejecutarlo en modo background

```
$ docker run -d --name holaMundo ubuntu:14.04 /bin/bash -c "while true; do echo Hola Mundo; sleep
```

Nos da un error ya que ya tenemos un contenedor (aunque parado) con ese nombre:

```
FATA[0000] Error response from daemon: Conflict. The name "holaMundo" is already in use by
container 514041140ff2. You have to delete (or rename) that container to be able to reuse that na
```

Podremos ver su salida estándar mediante **docker logs container-id**

5.8 Borrado de un contenedor

Para borrar un contenedor es necesario que este parado.

Recordemos comandos:

- Ver contenedores en ejecución: **docker ps**
- Ver contenedores parados o en ejecución: **docker ps -a**
- Parar un contenedor **docker stop container-id**
- Borrar un contenedor **docker rm container-id**

Chapter 6

Trabajar con imagenes de Docker

6.1 Imágenes en local

Las imágenes se pueden bajar del Hub de Docker (**docker search**) o de nuestro registro y nuestro equipo las guarda en local.

```
$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             VIRTU
secure_registry                 latest              87f01c32b094        3 days ago          548.5
juanda/ubuntu                   latest              f83eee440560        6 days ago          202.7
registry                        2.0                 2971b6ce766c        8 days ago          548.1
golang                          1.4                 ca0f230b927e        9 days ago          517.2
mongo                           latest              5242d1368ac4        10 days ago         255.9
node                            latest              f709efdf393f        10 days ago         710.9
mysql                           5.6.24              56f320bd6adc        2 weeks ago         282.9
mysql                           latest              56f320bd6adc        2 weeks ago         282.9
wordpress                       4.1.2-fpm           e83a380ff338        2 weeks ago         432.9
```

6.2 Los containers no guardan datos

Creamos un container nuevo e instalamos un servidor web Apache:

```
$ docker run --name ubuntu -t -i ubuntu:latest  /bin/bash
root@5b973e0439da:/# apt-get install -y apache2
root@5b973e0439da:/# exit
```

Una vez que salimos del container, el proceso deja de correr y el container se para.

```
$ docker ps
$ docker ps -a
```

Si lo volvemos a ejecutar, la instalación de Apache ha desaparecido!!!!

```
$ docker run --name ubuntu -t -i ubuntu:latest  /bin/bash
```

6.3 run vs start

El comando **docker run** crea un container nuevo a partir de la imagen anterior por eso Apache2 desaparece

El comando **docker start** arranca el container que está parado, en su estado actual.

- Podemos conectarnos posteriormente y comprobar como Apache permanece instalado mediante el comando **docker exec**

```
$ docker start ubuntu /bin/bash
$ docker exec -t -i <contenedor> /bin/bash
```

6.4 Crear mi imagen: conceptos básicos

Normalmente no guardaremos datos de nuestras aplicaciones en las imagenes:

- Queremos que las **imagenes** sean **reusables**
- Queremos que sean **ligeras**

Para **guardar datos**, utilizaremos **volúmenes**

Para crear una imagen hay dos métodos:

- De **forma manual**
- Mediante un **script** que automatice el proceso: **Dockerfile**
  - Algo parecido al uso de Vagrant para Vmware y VirtualBox

6.5 Crear una imagen de forma manual

Debemos partir de una **imagen base**, en mi caso **ubuntu:latest**

Arrancamos la imagen entrando en consola e instalamos los paquetes que queramos para la imagen

```
$ docker run -t -i dockerreg.inf.enlaces:5000/ubuntu
root@40457ff8d020:/# apt-get install -y apache2
....
root@40457ff8d020:/# exit
```

6.6 Commit de la imagen

Una vez que tenemos la imagen base con las modificaciones realizadas, haremos el commit

- Tendremos una nueva imagen en local

```
$ docker commit -m "Instalado Apache2" -a "juanda" 40457ff8d020 juanda/ubuntu-apache
13f1ed84dbcc46ea554e92a307041233252e436311fa88bd5e8c63027c770e1a
```

40457ff8d020 es el id del container

juanda/ubuntu-apache es el repositorio para la imagen

- Ojo, si queremos subirla al Hub de Docker, debe coincidir con nuestro usuario en Docker.

Posteriormente podemos ejecutar un contenedor con la nueva imagen

```
$ docker run -t -i -rm juanda/ubuntu-apache
```

El parámetro rm lo ponemos para que borre el contenedor después de usarlo (container de un solo uso)

6.7 Comandos para un dockerfile

FROM: Para definir la imagen base

MAINTAINER: Nombre e email del mantenedor de la imagen

COPY: Copiar un fichero o directorio a la imagen

ADD: Para copiar ficheros desde urls. También tars, que descomprimie.

RUN: Para ejecutar un comando dentro del container.

CMD: Comando por defecto cuando ejecutamos un container. Se puede sobreescribir desde la CLI.

ENV: Variables de entorno

EXPOSE: Para definir los puertos del contenedor. Se deberán añadir de forma explícita en la llamada desde la CLI.

VOLUME: Para definir directorios de datos que quedan fuera de la imagen.

ENTRYPOINT: Comando a ejecutar de forma obligatoria al correr una imagen.

USER: Usuario para RUN, CMD y ENTRYPOINT.

WORKDIR: Directorio para ejecutar los comandos RUN, CMD, ENTRYPOINT, ADD y COPY

6.8 Crear una imagen mediante script

Dockerizamos por ejemplo un servicio apt-cacher-ng

Creamos el fichero dockerfile

```
FROM        ubuntu
MAINTAINER  SvenDowideit@docker.com

VOLUME      ["/var/cache/apt-cacher-ng"]
RUN     apt-get update && apt-get install -y apt-cacher-ng

EXPOSE      3142
CMD     chmod 777 /var/cache/apt-cacher-ng && /etc/init.d/apt-cacher-ng start && tail
```

Creo la imagen, que llamaré juanda/apache-php:

```
$ docker build -t juanda/apt-cacher-ng .
```

Lo ejecutamos:

```
$ docker run -d -p 9999:3142 juanda/apa-cacher-ng
```

Probamos el acceso desde el navegador

```
http://localhost:9999
```

6.9 Borrado de imagenes

Para borrar una imagen es necesario que no la use ningún contenedor

El comando de borrado es **docker rmi**

Para ver el listado de imágenes que tenemos utilizaremos **docker images**

Chapter 7

Ejemplo para Wordpress:

7.1 Elección de imágenes

Wordpress

- Tutorial para usar la imagen
- Dockerfile y más en GitHub

Mysql

- Tutorial para usar la imagen
- Dockerfile y más en GitHub

7.2 Creación de containers

Creo dos contenedores para volumenes de datos:

```
$ docker create -v /home/juanda/project/bbdd:/var/lib/mysql --name bbdd dockerreg.inf.enlaces:500
$ docker create -v /home/juanda/project/web:/var/www/html --name web dockerreg.inf.enlaces:5000/w
```

Uno para mysql:

```
$ docker run --volumes-from bbdd --name mysql -e MYSQL_ROOT_PASSWORD="xxxx" -d dockerreg.inf.enla
```

Otro para Apache y php:

```
$ docker run --volumes-from web --name apache --link mysql:mysql -d -p 8080:80 dockerreg.inf.enla
```

7.3 Prueba de funcionamiento

Ejecutamos localhost:8080 y vemos que accedemos a Wordpress

Otra opción hubiera sido, sin utilizar containers para datos:

- Mapeamos los datos del wordpress al container del wordpress
- Mapeamos los datos de la bbdd al container de mysql

```
docker run -v /home/juanda/project/mysql:/var/lib/mysql --name mysql -e MYSQL_ROOT_PASS
docker run -v /home/juanda/project/wordpress:/var/www/html --name apache --link mysql:m
```

Chapter 8

Bibliografía

8.1 Bibliografía

https://docs.docker.com/

https://www.digitalocean.com/community/tags/docker?type=tutorials

http://www.jayway.com/2015/03/21/a-not-very-short-introduction-to-docker/

http://stackoverflow.com/