Recetas Docker Documentation

Versión 0.1.0

Rafael Rodriguez Gayoso

09 de noviembre de 2017

Índice general

1. Capítulo 1. Primeros pasos con Docker **1**
- 1.1. 1.0 Introducción . . . 1
- 1.2. 1.1 Instalación de Docker en Ubuntu 14.04 . . . 1
- 1.3. 1.2 Instalación de Docker en CentOS7 . . . 2
- 1.4. 1.3 Docker a través de Vagrant . . . 2
- 1.5. 1.4 Probar nuevas funcionalidades de Docker . . . 3
- 1.6. 1.5 'Hola Mundo' en Docker . . . 3
- 1.7. 1.6 Ejecutar contenedores en segundo plano . . . 3
- 1.8. 1.7 Crear, iniciar, detener y eliminar contenedores . . . 3
- 1.9. 1.8 Crear una imagen de Docker con un Dockerfile . . . 4
- 1.10. 1.9 Compartir datos con los contenedores . . . 4
- 1.11. 1.10 Compartir datos entre contenedores . . . 5
- 1.12. 1.11 Copiar datos desde y hacia contenedores . . . 5

2. Capitulo 2. Crear y compartir imágenes **7**
- 2.1. 2.0 Introducción . . . 7
- 2.2. 2.1 Guardar los cambios de un contenedor en una nueva imagen . . . 7
- 2.3. 2.2 Guardar contenedores e imágenes en ficheros Tar . . . 8
- 2.4. 2.3 Nuestro primer Dockerfile . . . 8
- 2.5. 2.4 Migración desde Vagrant a Docker . . . 9
- 2.6. 2.5 Uso de Packer para crear una imagen Docker . . . 10
- 2.7. 2.6 Publicar imágenes en Docker Hub . . . 10

3. Licencia **13**
- 3.1. Licencia . . . 13

CAPÍTULO 1

Capítulo 1. Primeros pasos con Docker

1.1 1.0 Introducción

El núcleo de Docker es su motor (Docker engine), que no es más que un 'demonio' que nos permite crear y gestionar contenedores. Antes de comenzar a trabajar con el, debemos instalarlo en nuestro sistema.

Las primeras recetas de este capítulo nos guiarán en los pasos para instalarlo y poder comenzar a usarlo en nuestro equipo. La documentación oficial de Docker abarca la instalación en multitud de sistemas operativos. En esta guía veremos los casos de Ubuntu 14.04 y CentOS7. Si queremos probar Docker a través de Vagrant, iremos a la receta *1.3 Docker a través de Vagrant*.

Una vez instalado Docker, practicaremos con los comandos básicos para poder crear y mantener contenedores. A continuación, se introduce el concepto de fichero manifiesto; **Dockerfile**. Y por último, veremos como se gestionan los datos a través de uno o varios contenedores.

1.2 1.1 Instalación de Docker en Ubuntu 14.04

Para instalar Docker en Ubuntu 14.04 ejecutaremos los siguientes comandos:

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

Para comprobar que la instalación se ha realizado correctamente, ejecutamos el comando:

```
$ sudo docker --version
Docker version 17.07.0-ce
[...]
```

Se puede parar, iniciar y reiniciar el servicio. Por ejemplo, para reiniciarlo:

```
$ sudo service docker restart
```

Para que un usuario sin privilegios pueda hacer uso de Docker, ejecutamos uno de los siguientes comandos:

```
$ sudo usermod -aG docker %USER
--
$ sudo gpasswd -a $USER docker
```

1.3 1.2 Instalación de Docker en CentOS7

Instalamos Docker a través del gestor de paquetes `yum`. CentOS usa `systemd` para gestionar los servicios, por lo que haremos uso del comando `systemctl` para iniciarlo:

```
$ sudo yum update -y
$ sudo yum -y install docker
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
-logout-
$ sudo systemctl start docker
$ docker run hello-world
```

De forma alternativa, podemos hacer uso del script de instalación:

```
$ sudo yum update -y
$ curl -sSL https://get.docker.com | sudo sh
$ sudo usermod -aG docker $USER
$ sudo systemctl start docker
```

1.4 1.3 Docker a través de Vagrant

En el caso de querer instalar Docker en un sistema operativo diferente al instalado de forma nativa en nuestro equipo, podemos hacer uso de Vagrant, para por ejemplo, en nuestro sistema OS X instalar Docker sobre Ubuntu.

A través de un script de aprovisionamiento podemos configurar automáticamente una nueva máquina virtual. A continuación podemos observar el contenido del fichero *Vagrantfile*:

```
VAGRANTFILE_API_VERSION = "2"

$bootstrap=<<SCRIPT
apt-get update
apt-get -y install wget
wget -qO- https://get.docker.com/ | sh
gpasswd -a vagrant docker
service docker restart
SCRIPT

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "ubuntu/trusty64"

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--memory", "1024"]
  end
```

```
  config.vm.provision :shell, inline: $bootstrap

end
```

Descarga: `Vagrantfile`

1.5 1.4 Probar nuevas funcionalidades de Docker

Si deseamos probar funcionalidades recién publicadas, y que todavía no están en la versión estable, debemos hacer uso de la versión de desarrollo (experimental)[1].

En una distribución Linux ejecutamos los siguientes comandos para obtener las 'nightly builds':

```
$ wget -qO- https://experimental.docker.com/ | sh
$ docker version | grep Version
[...]
```

1.6 1.5 'Hola Mundo' en Docker

Una vez instalado Docker, el siguiente paso es la ejecución de nuestro primer contenedor, y dentro del mismo ejecutar el comando `echo Hola mundo`:

```
$ docker run busybox echo Hola mundo
[...]
```

Si queremos ir un poco más allá, y tener acceso a una sesión con terminal, debemos ejecutar `/bin/bash` dentro del contenedor, y sin olvidarnos de las opciones `-t` y `-i`, para poder conectar con dicha terminal de forma interactiva:

```
$ docker run -ti ubuntu:14.04 /bin/bash
```

1.7 1.6 Ejecutar contenedores en segundo plano

Ahora veremos como ejecutar un contenedor en segundo plano a través de la opción `-d`:

```
$ docker run -d -p 1234:1234 python:2.7 python -m SimpleHTTPServer 1234
$ docker ps
```

Si abrimos un navegador para acceder al puerto 1234 de *localhost*, veremos el contenido del directorio raíz dentro del contenedor.

1.8 1.7 Crear, iniciar, detener y eliminar contenedores

Para ejecutar las diferentes acciones tendremos que utilizar las opciones `create`, `start`, `stop`, `kill` y `rm`. Para consultar sus diferentes opciones, haremos uso de la opción `-h`.

[1] https://blog.docker.com/2015/06/experimental-binary/

Hasta ahora hemos usando el comando `docker run` para crear contenedores. Otra opción es usar el comando `docker create`. Usando el mismo ejemplo del servidor HTTP en python, ahora no indicamos la opción `-d`. Una vez creado el contenedor, será necesario ejecutar la opción `start` para iniciarlo:

```
$ docker create -P --expose=1234 python:2.7 python -m SimpleHTTPServer 1234
$ docker ps -a
$ docker start [ID]
$ docker ps
```

Para detener un contenedor, podemos escoger entre `docker kill` (que envía una señal `SIGKILL` al contenedor) o `ddocker stop` (que envía una señal `SIGTERM`, y después de un periodo de gracia, envía un `SIGKILL`). El resultado es que el contenedor será parado y no aparecerá en la lista de contenedores en ejecución. Como no ha desparecido, podemos volver a iniciarlo con `docker restart`, o eliminarlo definitivamente con `docker rm`.

Si tenemos un número elevado de contenedores que queremos eliminar, podemos ejecutar el siguiente comando:

```
$ docker rm $(docker ps -aq)
```

1.9 1.8 Crear una imagen de Docker con un Dockerfile

Sabemos como crear contenedores a partir de imágenes descargadas. Ahora veremos como modificarlas a nuestro gusto.

Un *Dockerfile* es un fichero de texto que especifica los pasos necesarios para crear la imagen (instalación de paquetes, nuevos directorios, etc.).

Haremos uso de *busybox* en la que declararemos una variable de entorno. El contenido del *Dockerfile* es el siguiente:

```
FROM busybox
ENV foo=bar
```

Y a continuación creamos la nueva imagen *busybox2* a través del comando `docker build`:

```
$ docker build -t busybox2 .
[...]
```

Una vez finalizado, podemos comprobar que aparece en el listado de `docker images`, y si creamos un nuevo contenedor basado en esta imagen, tendremos acceso a la variable de entorno *foo*.

1.10 1.9 Compartir datos con los contenedores

Veremos como acceder a los datos de nuestro equipo anfitrión desde dentro de un contenedor. Con la opción `-v` en el comando `docker run` se monta un volumen del 'host' dentro del contenedor.

Por ejemplo, para compartir el directorio de trabajo del host con el directorio */shared* del contenedor:

```
$ ls
data.txt
$ docker run -ti -v "$PWD":/shared ubuntu:14.04 /bin/bash
```

Por defecto, el montaje se realiza en modo lectura-escritura. En caso de querer realizar un montaje de sólo-lectura, debemos indicarlo al final de la ruta `"$PWD":/shared:ro`. Podemos comprobarlo con la opción `inspect`:

```
$ docker inspect -f {{.Mounts}} [ID]
```

1.11 1.10 Compartir datos entre contenedores

Tal y como vimos en la receta anterior, si omitimos la ruta de nuestro equipo anfitrión, se crea un *contenedor de datos*. Ese volumen se crea dentro del contenedor como un sistema de ficheros de lectura-escritura. A pesar de que la gestión del volumen es responsabilidad de Docker, es posible el acceso desde el equipo anfitrión:

```
$ docker run -ti -v /shared ubuntu:14.04 /bin/bash
root@id:/# touch /shared/foobar
root@id:/# ls /shared
foobar
root@id:/# exit
$ docker inspect -f {{.Mounts}} [ID]
[{.... /var/lib/docker/volumes/id /_data /shared local true}]
$ sudo ls /var/lib/docker/volumes/.....
foobar
```

Desde el equipo anfitrión es posible acceder al volumen a través de la ruta `/var/lib/docker/volumes/`. Los cambios pueden ser vistos desde el contenedor:

```
$ sudo touch /var/lib/docker/volumes/$id/foobar2
$ docker start $id
$ docker exec -ti $id /bin/bash
root@id:/# ls /shared
foobar foobar2
```

Para poder compartir este volumen con otros contenedores, haremos uso de la opción `--volumes-from`:

```
$ docker run -v /data --name data ubuntu:14.04
$ docker ps

$ docker inspect -f {{.Mounts}} data
```

Aunque el contenedor no se está ejecutando, se puede montar el volumen en otro contenedor:

```
$ docker run -ti --volumes-from data ubuntu:14.04 /bin/bash
root@id:/# touch /data/foobar
root@id:/# exit
$ sudo ls /var/lib/docker/volumes/$id...
foobar
```

1.12 1.11 Copiar datos desde y hacia contenedores

Si queremos copiar datos a un contenedor que carece de volumen montando, tendremos que utilizar la opción `docker cp`. Para ver su funcionamiento, ejecutamos los siguientes comandos:

```
$ docker run -d --name testcopy ubuntu:14.04 sleep 360
$ docker exec -ti testcopy /bin/bash
root@id:/# cd /root
root@id:/# echo 'Estoy en el contenedor' > file.txt
root@id:/# exit
```

Ahora, para poder acceder al fichero desde el equipo anfitrión:

```
$ docker cp testcopy:/root/file.txt .
$ cat file.txt
Estoy en el contenedor
```

Y para copiar un fichero hacia el contenedor:

```
$ echo 'Estoy en el host' > host.txt
$ docker cp host.txt testcopy:/root/host.txt
```

CAPÍTULO 2

Capitulo 2. Crear y compartir imágenes

2.1 2.0 Introducción

Después de trabajar con las operaciones básicas en Docker, en este capítulo veremos como crear y compartir nuestras propias imágenes. Nos puede interesar el empaquetado de una aplicación existente, o bien construír una nueva desde cero.

Cuando creamos un contenedor a partir de una imagen, y realizamos cambios en el mismo, a continuación, podemos realizar un 'commit' para crear una nueva imagen personalizada con los cambios realizados. Aunque este proceso no es facilmente reproducible, por lo que se recomienda el uso de un fichero *Dockerfile*.

Para poder compartir nuestras imágenes con otros usuarios, debemos hacer uso de un repositorio compartido, por ejemplo, Docker Hub. Veremos como hacer uso de este servicio, así como automatizar su despliegue desde Github o Bitbucket.

2.2 2.1 Guardar los cambios de un contenedor en una nueva imagen

Creamos un contenedor interactivo y actualizamos la base de datos de paquetes:

```
$ docker run -ti ubuntu:14.04 /bin/bash
root@id:/# apt-get update
```

Cuando salimos del contenedor, se detiene, pero aún está disponible para poder realizar un *commit* con los cambios realizados y crear una nueva imagen *ubuntu:update*:

```
$ docker commit $id ubuntu:update
[...]
$ docker images
```

En este momento, podemos parar y eliminar el contenedor, ya que podremos crear uno nuevo a partir de la imagen recién creada.

2.3 2.2 Guardar contenedores e imágenes en ficheros Tar

Usaremos los comandos `save` y `load` para crear un *tarball* de una imagen, o bien los comandos `import` y `export` para contenedores.

Usamos un contenedor detenido, y lo exportamos a un fichero *tar*:

```
$ docker ps -a
[...]
$ docker export $id > update.tar
```

Ahora, en lugar de utilizar el comando `commit`, usamos el comando `import`:

```
$ docker import - update < update.tar
[...]
$ docker images
```

Si queremos trabajar con imágenes que ya han sido creadas con el comando `commit`:

```
$ docker save -o update1.tar update
$ docker rmi update
$ docker load < update1.tar
$ docker images
```

2.4 2.3 Nuestro primer Dockerfile

Para poder automatizar la construcción de una imagen de Docker, tendremos que detallar los pasos en un fichero tipo 'manifiesto' llamado *Dockerfile*. Este fichero de texto usa un conjunto de instrucciones para definir la imagen base para el nuevo contenedor, que aplicaciones deben instalarse y sus dependencias, los ficheros presentes en la imagen, los puertos accesibles desde el exterior, y el comando a ejecutar en el inicio del contenedor, así como otras muchas configuraciones.

Veremos como crear nuestro primer *Dockerfile*:

```
FROM ubuntu:14.04

ENTRYPOINT ["/bin/echo"]
```

La directiva `FROM` indica la imagen base de la cual partiremos. La directiva `ENTRYPOINT` indica el comando que se ejecutará cuando se inicie un contenedor basado en esta nueva imagen. Para poder crear esta imagen, ejecutamos el siguiente comando:

```
$ docker build .
[...]
$ docker images
```

En este momento, disponemos de una nueva imagen para poder generar nuevos contenedores:

```
$ docker run [ID] Hola Docker !!
Hola Docker !!
```

También podemos utilizar la directiva `CMD` en un *Dockerfile*. Y presenta la ventaja de que se puede sobreescribir su contenido cuando se crea un contenedor pasando un nuevo `CMD` como argumento a `docker run`. Vamos a crear una nueva imagen con las siguientes instrucciones:

```
FROM ubuntu:14.04

CMD ["/bin/echo"] , "Hola Docker !"]

$ docker build .
[...]
$ docker run [ID]
Hola Docker !
```

Parece que es lo mismo que en el caso anterior, pero si le pasamos un nuevo argumento al comando `docker run`, se ejecutará en lugar de `/bin/echo`:

```
$ docker run [ID] /bin/date
```

Hasta ahora hemos hecho referencia a nuestras nuevas imágenes a través de su ID generado. Podemos crear una nueva imagen con el nombre `curso` y la etiqueta `hello`, a través de la opción `-t`.

```
$ docker build -t curso:hello .
$ docker images
[...]
```

Más información:

- Buenas prácticas para diseñar un Dockerfile.
- Dockerfiles para CentOS.

2.5 2.4 Migración desde Vagrant a Docker

Si hemos estado trabajando con Vagrant en nuestro entorno de desarrollo, podremos reutilizar los ficheros *Vagrantfile* para usarlos en Docker.

Un ejemplo de Vagrantfile que usa Docker como proveedor:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

   config.vm.provider "docker" do |d|
      d.build_dir = "."
   end

    config.vm.network "forwarded_port", guest: 5000, host: 5000
end
```

La opción `build_dir` busca un fichero *Dockerfile* en el mismo directorio que el Vagrantfile. Vagrant ejecuta el coamndo `docker build`, e inicia el contenedor:

```
$ vagrant up --provide=docker
[...]
```

Una vez que `vagrant up` finaliza, el contenedor se estará ejecutando y tendremos una nueva imagen. En este momento, podemos hacer uso de los comandos `vagrant docker-logs` y `vagrant docker-run`. Los comandos estándar de Vagrant, como `vagrant status` y `vagrant destroy` funcionarán con el contenedor.

Ver ejercicio en Github con ficheros de ejemplo para consultar.

2.6 2.5 Uso de Packer para crear una imagen Docker

Packer es una herramienta de HashiCorp que es capaz de crear imágenes idénticas para diferente plataformas a partir de una plantilla. Por ejemplo, desde una plantilla se pueden crear imágenes para Amazon EC2, VMWare, VirtualBox y DigitalOcean. Una de esas plataformas es Docker.

La siguiente plantilla muestra tres etapas. En la primera se indica el *builder*; que en este caso concreto es Docker, así como la imagen a utilizar *ubuntu:14.04*. La segunda define la etapa de aprovisionamiento, un shell sencillo. Y finalmente, los pasos de post-procesado.

```
{
"builders": [
  {
    "type": "docker",
    "image": "ubuntu:14.04",
    "commit": "true"  }],
"provisioners": [
  {
    "type": "shell",
    "script": "bootstrap.sh"
  }
],
"post-processors": [
   {
     "type": "docker-tag",
     "repository": "how2dock/packer",
     "tag": "latest"
   }
]
}
```

Podemos comprobar esta plantilla y ejecutarla para construír la imagen con dos comandos:

```
$ packer validate template.json
$ packer build template.json
```

Para poder probar Packer, en el siguiente archivo hay un Vagrantfile que instala Docker dentro de la máquina y descarga Packer.

Descarga: `Packer_test`

TODO: Ejercicio con Ansible.

2.7 2.6 Publicar imágenes en Docker Hub

Una vez que hemos generado una imagen de cierta utilidad para el resto de usuarios de Docker, tenemos a nuestra disposición Docker Hub, a donde la podemos subir y hacerla accesible, o no, para el resto de comunidad Docker.

Para poder hacer uso del servicio de Docker Hub, debemos completar las siguientes acciones:

- Crear una cuenta en Docker Hub
- Iniciar sesión desde nuestro host con Docker

- Subir nuestra imagen (push)

Una vez creada nuestra cuenta en Docker Hub, accedemos a nuestro sistema con Docker, seleccionamos una imagen y la publicamos en nuestro repositorio público:

1. Iniciamos sesión con `docker login`. Tendremos que introducir las credenciales.
2. Etiquetamos una de nuestras imágenes con el nombre de usuario de Docker Hub.
3. Hacemos *push* hacia el repositorio.

El primer paso (login) guarda las credenciales en `~/.docker/config.json`. Una vez consultadas las imágenes disponibles, hacemos uso de una de ellas, por ejemplo:

```
$ docker tar packer [usuario]/packer
$ docker images
```

Una vez modificada la etiqueta con el formato requerido por Docker Hub, estamos preparados para realizar la subida:

```
$ docker push [usuario]/packer
```

Cuando finalice la subida, cualquier usuario puede realizar una descarga de dicha imagen con el comando `docker pull [usuario]/packer`.

En caso de no conocer el nombre exacto de una imagen, podemos realizar una búsqueda con el comando `docker search`:

```
$ docker search postgres
```

Entre los resultados podemos observar la imagen oficial (primera), y el resto serán las creadas por usuarios de Docker Hub.

CAPÍTULO 3

Licencia

3.1 Licencia

3.1.1 Reconocimiento-CompartirIgual 3.0 España (CC BY-SA 3.0 ES)

Esto es un resumen inteligible para humanos (y no un sustituto) de la licencia, disponible en los idiomas siguientes: Aranés Asturiano Castellano Euskera Galego

Usted es libre de:

Compartir - copiar y redistribuir el material en cualquier medio o formato

Adaptar - remezclar, transformar y crear a partir del material para cualquier finalidad, incluso comercial

El licenciador no puede revocar estar libertades mientras cumpla los términos de la licencia.

Bajo las condiciones siguientes:

Reconocimiento - Debe reconocer adecuadamente la autoría[1], proporcionar un enlace a la licencia e indicar si se han realizado cambios[2]. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador, o lo recibe por el uso que hace.

[1] Si se facilitan, debe proporcionar el nombre del creador y las partes a reconocer, un aviso de derechos de explotación, un aviso de licencia, una nota de descargo de responsabilidad y un enlace al material. Las licencias CC anteriores a la versión 4.0 también requieren que facilite el título del material, si le ha sido facilitado, y pueden tener algunas otras pequeñas diferencias.

[2] En la versión 3.0 y anteriores, la indicación de los cambios sólo se requiere si se crea una obra derivada.

Compartir Igual - Si remezcla, transforma o crea a partir del material, deberá difundir sus contribuciones bajo la misma licencia que el original[3].

No hay restricciones adicionales - No puede aplicar términos legales o medidas tecnológicas que legalmente restrinjan realizar aquello que la licencia permite.

[3] También puede utilizar una licencia compatible de la lista de https://creativecommons.org/compatiblelicenses.