Introducción a la virtualización basada en contenedores

Prof. Antonio J. Pérez Millán

CFGS Admón. Sistemas Informáticos y Redes

Introducción a la virtualización con docker

Introducción

Docker es una herramienta diseñada para beneficiar tanto a desarrolladores, testers, como administradores de sistemas, en relación a las máquinas, a los entornos en sí donde se ejecutan las aplicaciones software, los procesos de despliegue, etc.

Realmente el concepto es algo similar al de una máquina virtual, pero un contenedor no es lo mismo que una máquina virtual. Un contenedor es más ligero, ya que mientras que a una máquina virtual necesitas instalarle un sistema operativo para funcionar, un contenedor de Docker funciona utilizando el sistema operativo que tiene la máquina en la que se ejecuta el contenedor.

Intalación de docker en sistemas linux

```
#! /bin/bash

sudo apt-get remove docker docker-engine docker.io containerd runc

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'

sudo apt-get update

sudo apt-get install -y docker-engine
```

Conceptos Básicos para poder empezar

Las imágenes

Una imagen es una plantilla a partir de la cual poder crear instancias a las cuales denominamos contenedores.

```
# el comando para ver las imágenes que tenemos descargadas en nuestro ordenador

docker images
```

```
oot@usuario:/home/usuario# docker images
EPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
boraas/apache-php            latest              bcc4753fec41        27 hours ago        254MB
urnkeylinux/fileserver       latest              d63360e8bb30        12 months ago       804MB
urnkeylinux/fileserver-14.1  latest              a6f3f4bbca1c        3 years ago         478MB
oot@usuario:/home/usuario#
```

Cada imagen dispone de un identificador image id que sirve para ser identificada por el motor de docker.

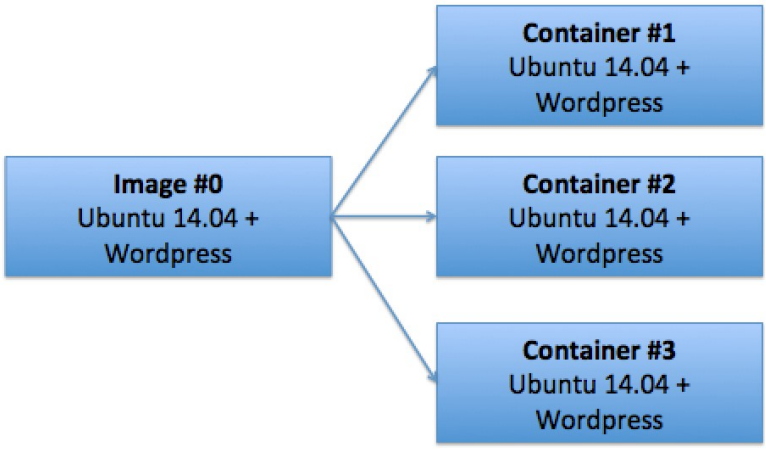
```
#El comando para borrar una imagen

docker rm id-imagen
```

```
root@usuario:/home/usuario# docker rmi d63360e8bb30
Untagged: turnkeylinux/fileserver:latest
Untagged: turnkeylinux/fileserver@sha256:c2f37ec2ea1f3eae04893bb21d81647e3cb65e7e2fe0bf163707efae5b03a1c4
Deleted: sha256:d63360e8bb30fda1479ce0b53417f58de298b90d59f847672b7457fb16ef63b0
Deleted: sha256:248cd7ac00ae5b4928ee39e1e2b76913fd1354d9eddec2ae070b71bd3a40948d
root@usuario:/home/usuario#
```

Los contenedores

Los contenedores son máquinas virtuales creadas a través de plantillas que son las imágenes vistas anteriormente. Dichas máquinas virtuales admiten modificaciones sobre las plantillas en las cuales están basadas.



Docker hub

Es un repositorio oficial en el que los desarrolladores de imágenes pueden registrar y subir sus plantilla para ser descargadas por el resto de usuarios.

```
# El comando para hacer busquedas de imágenes en docker hub

docker search [patron de busqueda]

docker search wordpress
```

```
# El comando para poder hacer descargas  de imágenes almacenadsas en el docker hub es:

docker pull nombre-imagen

docker pull wordpress
```

La dirección del docker hub :
https://hub.docker.com/

Algunas imágenes interesantes dentro del docker hub:

https://hub.docker.com/search?q=bitnami&type=image

https://hub.docker.com/search?q=turnkeylinux&type=image

DockerFile

Es un archivo de configuración que se utiliza para crear imágenes. En dicho archivo indicamos qué es lo que queremos que tenga la imagen, y los distintos comandos para instalar las herramientas. Esto sería un ejemplo de DockerFile para tener una imagen de Ubuntu con Git instalado:

```
FROM ubuntu:14.04
RUN apt-get update
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get -qqy install git
```

Para compilar ficheros con directivas de docker se utiliza el comando docker build

```
docker build -t “etiqueta” .
```

Algunas de las directivas que pueden ir dentro de un Dockerfile son:

MAINTAINER: Nos permite configurar datos del autor, principalmente su nombre y su dirección de correo electrónico.

ENV: Configura las variables de entorno.

ADD: Esta instrucción se encarga de copiar los ficheros y directorios desde una ubicación especificada y los agrega al sistema de ficheros del contenedor. Si se trata de añadir un fichero comprimido, al ejecutarse el guión lo descomprimirá de manera automática.

COPY: Es la expresión recomendada para copiar ficheros, similar a ADD.

EXPOSE: Indica los puertos en los que va a escuchar el contenedor. Hay que tener en cuenta que esta opción no consigue que los puertos sean accesibles desde el host; para esto debemos utilizar la exposición de puertos mediante la opción-p

VOLUME: Esta es una opción que muchos usuarios de la Web estaban esperando como agua de mayo. Nos permite utilizar en el contenedor una ubicación de nuestro equipo anfitrión,y así, poder almacenar datos de manera permanente. Los volúmenes de los contenedores siempre son accesibles en el host anfitrión, en la ubicación:`/var/lib/docker/volumes/`

Volúmenes

No es una buena práctica guardar los datos persistentes dentro de un contenedor de Docker. Para eso están los volúmenes, fuera de los contenedores. Así podremos crear y borrar contenedores sin preocuparnos por que se borren los datos. Además los volúmenes se utilizan para compartir datos entre contenedores. Para crear volúmenes al mismo tiempo que se arranca el contenedor podemos indicar una carpeta local a la que enlazaremos una carpeta existente dentro del contenedor.

```
#asociar una carpeta de la máquina local a una existente en un contenedor
docker run -p 80:80 -p 443:443 -v /carpeta-local/:/var/www/html/ -d eboraas/apache-php
```

Otros comandos para gestionar volúmenes son:

```
docker volume create
docker volume ls
docker volume rm
docker volume inspect
```

Ejecución de contenedores

```
docker run -v <unidad_host>:<unidad_docker> -it --name
<nombre_contenedor> -d <nombre_imagen>
```

donde:

- -v nos permite asociar una carpeta (volumen) de la máquina anfitriona a una carpeta del contenedor
- -p nos permite redirigir puertos de la máquina anfitriona a los puertos del contenedor
- -d nos permite ejecutar la máquina en segundo plano.
- -it nos permite ejecutar la máquina de forma interactiva con una terminal.
- --name nos permite darle un nombre al contenedor, en caso de no asignarle un nombre docker nos creará un nombre aleatorio.

Mas posibilidades del comando docker run

docker run –help

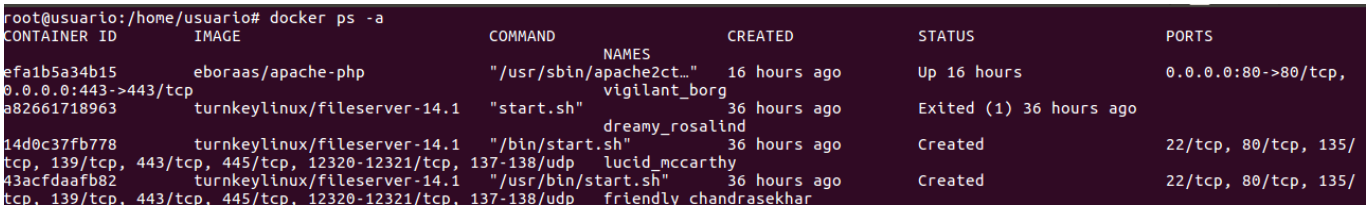
De igual manera que arrancamos contenedores podemos también pararlos, pausarlos y volverlos a arrancar con el los modificadores stop, pause y start sobre el comando docker.

```
docker stop id_contenedor
docker start id_contenedor
docker pause id_contenedor
docker restart id_contenedor
```

Para la monitorización de los contenedores que están en ejecución y los recursos que consumen podemos utilizar el comando

```
docker stats
```

Ver contenedores en ejecución



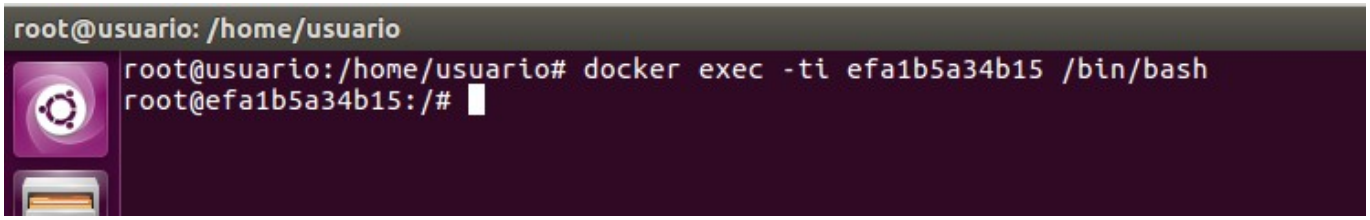

```
docker ps -a
```

Para ver solamente los identificadores de los contenedores usamos el parámetro -q

Ejecución de comandos en contenedores que están ejecutandose

Para ejecutar algún comando dentro de un contenedor que esté en ejecución podemos usar el comando docker exec. Como siempre si queremos ver todas las opciones de dicho comando podemos teclear docker exec --help

```
docker exec -ti identificador-contenedor /bin/bash
```



Inspección de contenedores

Para ver las características que tiene un contenedor que hemos creado podemos utilizar el comando docker inspect. De esta inspección podemos extraer información interesante para nosotros como la dirección ip asignada al contenedor, los volúmenes que tiene, el comando de inicio, los puertos expuestos...etc.

```
docker inspect id-contenedor
```

Vínculo entre contenedores

En ocasiones podemos querer que dos contenedores tengan vinculación directa de manera que uno de ellos sea capaz de resolver (DNS) directamente el nombre del otro. Para ello podemos utilizar el parametro --link en el comando run .

Para ver este concepto imaginemos que tenemos dos contenedores. El primero de ellos presta servicio de base de datos mediante mariadb. El segundo es un servicio web que tiene un portal desde el que administrar el primer servidor. Para que este último contenedor funcione tiene que tener un vínculo con el primer contenedor.

```
docker run -dti --name bd -v $PWD/datos:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=wordpress -e MYSQL_DATABASE=wordpress mariadb:latest

docker run -dti -p 8080:8080 --link bd:basedatos --rm adminer
```

Creación de imágenes a partir de contenedores

En ocasiones nos puede interesar guardar los contenedores con todas las modificaciones que le hayamos hecho para a partir de ellas generar nuestra propia imagen.

```
docker commit id-contenedor -a "autor" -m mensaje identificador_imagen/version
```

```
root@usuario:/home/usuario# docker commit  -a "aperez@ceslopedevega.com" -m "contenedor phpapache" efa1b5a34b15 aperez/apachephp:v1
sha256:c1baeef3a5ce915c88a9bf5ab2d4f4a57e1645cb1d6d4083134f42c5a3810ebd
root@usuario:/home/usuario# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
aperez/apachephp              v1                  c1baeef3a5ce        5 seconds ago       254MB
eboraas/apache-php            latest              bcc4753fec41        2 days ago          254MB
turnkeylinux/fileserver-14.1  latest              a6f3f4bbca1c        3 years ago         478MB
root@usuario:/home/usuario#
```

Una posibilidad adicional es la de poder guardar las imágenes creadas en un fichero con extensión .tar para poder migrarlas a otro sistema. Para ello docker dispone del parámetro save.

```
docker save aperez/apachephp > copia.tar
```

```
Loaded image: aperez/apachephp:v1
root@usuario:/home/usuario# docker save aperez/apachephp > copia.tar
root@usuario:/home/usuario#
```

La forma de importar una imagen que previamente hayamos copiado es mediante el modificador load del comando docker.

```
docker load -i fichero.tar
```

```
root@usuario:/home/usuario# docker load -i copia.tar
Loaded image: aperez/apachephp:v1
root@usuario:/home/usuario# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
aperez/apachephp              v1                  c1baeef3a5ce        About an hour ago   254MB
eboraas/apache-php            latest              bcc4753fec41        2 days ago          254MB
turnkeylinux/fileserver-14.1  latest              a6f3f4bbca1c        3 years ago         478MB
root@usuario:/home/usuario#
```

Si lo que queremos es exportar un contenedor completo podemos usar el modificador export en el comando docker.

```
docker export id_contenedor > fichero.tar
```

```
root@usuario:~# docker export efa1b5a34b15 > fichero.tar
root@usuario:~#
```

De igual manera si lo que queremos es importar un contenedor disponemos del comando import correspondiente

```
docker import fichero.tar
```

Redes en docker

Docker crea tres tipos de redes:

- redes bridge: es el modo de direccionamiento por defecto asignado a los contenedores direcciones ips relacionadas con el adaptador de red principal de docker (172.17.0.1), que actuará como servidor de dhcp y como puerta de enlace de todos los contenedores por defecto.
- redes host: en la que el contenedor utilizará la misma ip que la máquina anfitriona.
- none: para tener contenedores aislados sin configuración de red.

Comandos para el uso de redes

```
#conectar/desconectar un contenedor a una red:
docker connect/disconnect nombre-red
```

```
#Crear un nuevo segmento de red

docker network create -d tipo-de-red nombreNuevaRed

#Ver las redes disponibles

docker network ls

#Ver las características de una red

docker network inspect NombreRed

# purgar y eliminar todas las redes que no están siendo usadas por contenedores.

docker network prune

# arrancar un contenedor y que su configuración de red se asocia a un determinada red ya existente.

docker run -d –network redlocal –name nombre

#para ver las características de red de un contenedor

docker container inspect nombre-contenedor

#Para unir un contenedor a una red podemos utilizar el comando:

docker network connect redlocal nombre-contenedor

#Para hacer un vínculo directo a través de una red entre dos contenedores

 docker network connect --link contenedor1:red2 test contenedor2

# docker network connect --link contenedor2:red2 test contenedor1
```

Orquestando varios contenedores

A veces puede ser interesante que cuando tenemos que controlar y arrancar mas de un contenedor utilizar un componente de docker denominado docker-compose . Dicho componente se puede instalar como un complemento adicionarl al servicio de docker y permite crear scripts en formato yaml o yml . Dichos scripts contienen todos los comandos necesarios para orquestar (arrancar) varios contenedores a la vez sin necesidad de tener que interactuar con la consola de comandos.

La referencia completa de comandos para hacer scripts de docker-compose la podemos encontrar en este enlace:

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

```
Version: '3.8' # version del script
services: # declarion de los servicios/contenedores a usar
  db: # declaracion del contendor mysql
    image: mysql:5.7 # imagen a usar
    volumes: # volumens a vincular
      - "./.dbdata/db:/var/lib/mysql"
    restart: always # reiniciar automático y arranque automático al iniciar docker
    environment: # variables de entorno
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}


  wordpress:
    depends_on: # dependencia de este contenedor
      - db # contenedor del que depende
    image: wordpress:latest
    links: # vincular contenedores para que se puedan "ver"
      - db # contendor vinculado
    ports:
      - "80:80" # Puertos del exterior vinculado con el del contenedor
    restart: always
    volumes:
      - ./wordpress:/var/www/html
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: ${MYSQL_USER}
```

```
    WORDPRESS_DB_PASSWORD: ${MYSQL_PASSWORD}


phpmyadmin:
  depends_on:
    - db
  image: phpmyadmin/phpmyadmin
  links:
    - db
  ports:
    - "8080:80"
  restart: always
```

Para compilar el script que hemos generado nos vamos al directorio donde está el fichero .yaml y ejecutamos:

```
docker-compose up -d # Modo demonio en segundo plano
```

Para parar el cluster de contenedores:

```
docker-compose down
```

Para eliminar todos los contenedores que forman parte del cluster:

```
docker-compose rm
```