Docker

Sumario

1 Introducción

Docker es una evolución del concepto de máquina virtual. Básicamente consiste en una máquina virtual Linux que al ejecutarse sobre un sistema Linux usa las propiedades del kernel de la máquina real sobre la que se esté ejecutando. Por lo tanto, un contenedor de docker no incluye sistema operativo (usa el de la máquina Linux sobre el que se está ejecutando). También permite ejecutarse sobre otros sistemas operativos usando virtualización.

Hay dos conceptos importantes a tener en cuenta:

- **Imágen**: Una imagen sería el equivalente a un sistema de ficheros de una máquina virtual. En este sistema de ficheros se pueden copiar aplicaciones y configuraciones. Las imágenes se basan en capas de forma que sobre una imagen se puede añadir otra que modifique/amplíe los archivos, ejecutables,… que allí se almacenen.
- **Contenedor**: Es una instancia de docker que se ejecuta sobre una imagen. Sería el equivalente a una máquina virtual en ejecución.

2 Primeros pasos en Docker

En “hub.docker.com” se pueden encontrar imágenes ya creadas de docker. Para descargar una imagen se teclea en un terminal:

```
docker pull nombre-imagen
```

Por ejemplo:

```
$ docker pull busybox
```

Para arrancar un contenedor de docker se escribe:

```
docker run nombre-imagen
```

Si se necesita que la sesión sea interactiva, abrir un shell:

```
docker run -it nombre-imagen
```

En el caso de que la imagen no exista, se descargará de Docker Hub de forma automática.

Para ejecutar un contenedor en segundo plano y que no se cierre al cerrar el terminal, se usará la opción -d (--detach):

```
docker run -d nombre-imagen
```

Para obtener la lista de imágenes en nuestro equipo:

```
docker images
```

Para borrar una imagen se hace con:

```
docker rmi nombre-imagen
```

Se puede especificar una versión a ejecutar o descargar desde Docker Hub usando un **tag**. Si no se especifica la versión, se bajará siempre la última disponible “latest”, pero se puede especificar la versión escribiendo la versión a continuación del nombre de la imagen separada por dos puntos. Por ejemplo:

```
$ docker pull busybox:uclibc
uclibc: Pulling from library/busybox
Digest: sha256:702b62…….b5a1130dff8b8484352
Status: Downloaded newer image for busybox:uclibc
docker.io/library/busybox:uclibc

$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
busybox      latest    62aedd01bd85   12 days ago   1.24MB
busybox      uclibc    62aedd01bd85   12 days ago 1.24MB
```

Las imágenes están formadas por capas. Por ejemplo, se puede tener una imagen de Ubuntu y sobre ella añadir otra capa que puede ser un servidor de mysql. La imagen resultante tendrá dos capas Ubuntu + mysql.
Las capas comunes se comparten entre imágenes, lo cual ahorra espacio de almacenamiento en disco.

Al ejecutar una imagen se genera un contenedor, que es un proceso que ejecuta esa imagen. En un equivalente a un sistema de virtualización, como puede ser VirtualBox, la imagen es el sistema de ficheros y el contenedor en la máquina que se está ejecutando sobre dicho sistema de ficheros.

Se pueden tener dos contenedores funcionando los cuales serán independientes el uno del otro. Por ejemplo:

en un terminal se ejecuta:

```
$ docker run -it busybox
```

y en otro:

```
$ docker run -it busybox:uclibc
```

Se tendrán en ejecución dos terminales de dos contenedores diferentes de dos imágenes diferentes.

Con el comando:

```
docker ps
```

se pueden ver los contenedores en ejecución. Por ejemplo:

```
$ docker ps
CONTAINER ID   IMAGE            COMMAND   CREATED          STATUS          PORTS
NAMES
e7700a0dad74   busybox          "sh"      11 seconds ago   Up 11 seconds
great_lewin
9b5ac682bde5   busybox:uclibc   "sh"      46 seconds ago Up 45 seconds
eager_hopper
```

Como se puede ver en la columna final, docker ha asignado un identificador y un nombre a cada contenedor. En este caso los nombres son “great_lewin” o “eager_hopper” y los identificadores “e7700a0dad74” y “9b5ac682bde5”.

Este identificador o nombre se puede usar para parar la ejecución de un contenedor. Para ello se escribirá:

```
docker stop identificador
```

Por ejemplo:

```
$ docker stop  eager_hopper
```

Importante: Cada vez que se ejecuta una imagen con “docker run” se genera un nuevo contenedor, por lo que los datos se guardarán en dicho contenedor. Por línea de comandos se puede especificar que los datos se guarden en otro lugar.

Si se ejecuta:

```
docker ps -a
```

Se mostrará un historial de los contenedores ejecutados y los que se están ejecutando actualmente. Cada contenedor se puede volver a usar especificando su identificador y se podrán recuperar los datos almacenados en dicho contenedor. Hay un recolector de basura que los eliminará si no se usan durante cierto tiempo.

Por ejemplo:

```
$ docker start e7700a0dad74
```

iniciará el contenedor con identificador “e7700a0dad74” en el caso de que dicho contenedor esté parado.

3 Logs

Se pueden ver los logs de un contenedor usando:

```
docker logs identificador
```

Con la opción -f:

```
docker logs -f identificador
```

muestra todos los logs generados y espera a que se generen nuevos logs para que se muestran también. Ésto es útil para estudiar el comportamiento de un servidor en tiempo real.

IMPORTANTE: Se guardarán en el log todos los mensajes que se escriban en la salida estándar. Habrá que tenerlo en cuenta en el caso de que se esté escribiendo una aplicación que se quiera ejecutar en un contenedor.

4 Ejecución de comandos en un contenedor ya iniciado

Se pueden ejecutar comandos en un contenedor que ya se esté ejecutando:

```
docker exec identificador comando
```

Por ejemplo:

```
$ docker exec e7700a0dad74 ls
```

ejecutará el comando “ls” y mostrará la salida.

Se puede usar el parámetro -it para que muestre un terminal en que se pueden ejecutar comandos. Por ejemplo:

```
$ docker exec -it e7700a0dad74 sh
```

Se puede borrar un contenedor con:

```
docker rm identificador
```

Algunas imágenes al ejecutarlas necesitan parámetros adicionales. Generalmente lo indicará al ejecutar la imagen.

5 Dockerfiles

Son archivos que sirven para crear imágenes de docker. El proceso que se sigue es el que se indica en el siguiente esquema:

Dockerfile ---build---> **Imagen** --- run ---> **Contenedor**

Necesariamente se deben llamar “Dockerfile”.

Archivo Dockerfile:

```
# Dockerfile

FROM ubuntu  ← Imagen en la que se basará la imagen que se va a construir

RUN apt-get install apache2

COPY  code/ /var/www ← Copia el contenido del directorio code/ en /var/www de la
imagen

CMD [ “apache2” ]
```

Otro ejemplo para construir una imagen de nodejs:

```
FROM node         ← Imagen en la que se basará la imagen que se va a construir

WORKDIR /app      ← Directorio que se creará en la imagen y se toma como pwd
COPY . .          ← Copia el contenido del directorio actual en /app
RUN yarn install --production  ← Ejecuta un comando en /app

CMD [“node”, “/app/src/index.js”] ← Comando que se ejecutará al arrancar el
contenedor
```

Una vez escrito el archivo Dockerfile, para construir la imagen se ejecutará:

```
docker build -t nombre-imagen .
```

Se le pueden añadir tags al nombre de la imagen para distinguir diferentes versiones de la misma imagen:

```
docker build -t nombre-imagen:tag .
```

6 Mapeando puertos del contenedor a la máquina real

Con la opción -p se pueden mapear puertos del contenedor a la máquina real. Por ejemplo:

```
docker run -p 8080:8000 servidor-python
```

Hace que a través del puerto 8080 de la máquina real se pueda acceder al puerto 8000 del contenedor.

Se puede mezclar con la opción -d para ejecutarlo en segundo plano:

```
docker run -dp 8080:8000 servidor-python
```

7 Compartiendo directorios o archivos entre un contenedor y la máquina real (volúmenes)

Se puede compartir un directorio o archivo con entre un contenedor y la máquina real, para ello se usará la opción “-v” en el comando “docker run”:

```
docker run -v ruta-absoluta-máquina-real:ruta-absoluta-contenedor nombre-imagen
```

Por ejemplo:

```
docker run -v /app:/datos/ej -p 8000:8000 servidor-python
```

Compartirá la carpeta “/app” de la máquina real y aparecerá como “/datos/ej” en el contenedor.

De forma idéntica se pueden compartir archivos:

```
docker run -v /app/otro.html:/datos/ej.html -p 8000:8000 servidor-python
```

El archivo “/app/otro.html” de la máquina real se compartirá con el contenedor con el nombre “/datos/ej.html”.

Esto se puede hacer para conservar bases de datos entre diferentes instancias del contenedor.

También se pueden compartir varios archivos, simplemente añadiendo más opciones “-v” a la línea de comandos:

```
docker run -v /app/otro.html:/datos/ej.html -v /app/index.html:/datos/index.html  -p 8000:8000 servidor-python
```

También se puede usar para “sobrescribir” un archivo que ya existiera en el contenedor. Por ejemplo, si el archivo “index.html” ya existe en el contenedor y se quiere sobrescribir con otro:

```
docker run -v /app/otro.html:/datos/index.html -p 8000:8000 servidor-python
```

Se puede usar para cambiar configuraciones o código de una imagen ya existente sin necesidad de modificarla.

Si en lugar de usar rutas absolutas en la máquina real, se desean usar rutas relativas, la ruta se deberá comenzar con “./”:

```
docker run -v ./otro.html:/datos/index.html -p 8000:8000 servidor-python
```

8 Subiendo imágenes a Docker hub y cambiando el nombre a las imágenes

Se pueden subir imágenes a Docker Hub si se dispone de cuenta en dicho servicio.

Con:

```
docker login
```

nos conectamos al servicio.
Es importante poner correctamente el nombre y el tag antes de subir la imagen. El nombre de la imagen será:
"nuestro-nombre-de-usuario-en-docker-hub/nombre-imagen:tag"

Con el comando:
```
docker tag nuevo-nombre-image identificador-de-la-imagen
```

Se puede cambiar el nombre a la imagen. Realmente mantiene el nombre de la imagen y se puede ver que hay dos imágenes con el mismo identificador.

Para subir una imagen a Docker Hub se usará:
```
docker push nombre-imagen
```
En el proceso se puede ver que lo que se sube realmente son las capas que no se tenían.

9 Usando varios contenedores

Se pueden ejecutar dos contenedores y comunicarlos a través de una red. Para crear una red se usará el comando:
```
docker network create nombre-red
```

Para ejecutar un contenedor dentro de la red se usará:
```
docker run --network nombre-red --network-alias nombre-contenedor nombre-imagen
```
donde el parámetro --network sirve para indicar el nombre de la red en la que se va a ejecutar el contenedor y el parámetro --network-alias es un alias que se le va a dar al contenedor dentro de la red para poder encontrarlo más fácilmente.

Por ejemplo, para crear un contenedor con la base de datos MariaDB y poder administrarla con un segundo contenedor que cuente con PhpMyAdmin:

```
# Se arranca el contenedor con MariaDB
docker run --detach --network red-contenedores --network-alias host-mariadb --name
contenedor-mariadb --env MARIADB_USER=admin --env MARIADB_PASSWORD=admin-password
--env MARIADB_ROOT_PASSWORD=root-password mariadb:latest
# Se arranca el contenedo con PhpMyAdmin
docker run --detach --network red-contenedores --name contenedor-phpmyadmin -d -e
PMA_HOST=host-mariadb -p 8080:80 phpmyadmin
```

Como se puede ver los dos contenedores están en la misma red y se ha puesto un alias al contenedor con MariaDB ("host-mariadb"). Dicho alias es como si se hubiese introducido en el servidor de DNS y todos los miembros de la red pueden consultar dicho DNS.

10 Variables de entorno y contenedores

En el ejemplo anterior se puede ver que se le pasan variables de entorno al contenedor usando el parámetro "--env". Por ejemplo:

```
docker run --detach --network red-contenedores --network-alias host-mariadb --name
contenedor-mariadb --env MARIADB_USER=admin --env MARIADB_PASSWORD=admin-password
--env MARIADB_ROOT_PASSWORD=root-password mariadb:latest
```

Dichas variables de entorno dependen de la imagen y se pueden consultar en su página de “Docker hub”.

11 docker-compose.yaml

Es un archivo que permite lanzar varios contenedores con una configuración determinada. No hace falta declarar la red, pues docker-compose creará una red para los contenedores que se han definido y usará como alias de red el nombre de cada contenedor.

El archivo “docker-composer.yaml” se ejecutará con el comando:

```
docker-compose up -d
```

La opción “-d” es para la ejecución en segundo plano.

Para parar los contenedores se usará:

```
docker-compose down
```

Un ejemplo de fichero “docker-compose.yaml” sería:

```
version: "3.7"

services:
      contenedor_mariadb:
            image: mariadb:latest
            volumes:
                  - ./datos:/var/lib/mysql
            environment:
                  MARIADB_USER: admin
                  MARIADB_PASSWORD: admin-password
                  MARIADB_ROOT_PASSWORD: root-password

      contenedor_phpmyadmin:
            image: phpmyadmin
            ports:
                  - 8080:80
            environment:
                  PMA_HOST: contenedor_mariadb
```

Como se puede ver hay diferentes tags:

- services: Indica los contenedores que se van a crear. Dentro de este tag se indican los contenedores poniendo directamente un tag con el nombre del contenedor.
- Nombres de los contenedores “contenedor_phpmyadmin”, “contenedor_mariadb”: Se ponen como tags y se pueden elegir los que se deseen. Dentro de estos tags se pondrán las características del contenedor.
- image: Nombre de la imagen en la que se va a basar el contenedor.
- environment: Variables de entorno que se le van a pasar al contenedor. Las variables se indican con tags.
- ports: sirve para “mapear” puertos entre el contenedor y la máquina real.
- volumes: sirve para compartir carpetas o archivos entre el contenedor y la máquina real.

Nota: Si se ejecuta varias veces el comando “docker-compose up”, intenta ejecutar los contenedores que se encuentren parados. A veces, un servicio tarda más que otro y puede hacer que un contenedor falle al no poder conectar con el servicio correspondiente.

Importante: En el ejemplo para la base de datos se comparten las carpetas “./datos:/var/lib/mysql” . Las bases de datos MySQL y MariaDB guardan sus datos en la carpeta “/var/lib/mysql”. Si se quiere compartir dicha información entre contenedores, se debe compartir dicha carpeta “/var/lib/mysql”.

12 Guardando y cargando imágenes en/desde archivos

Se ha visto que se pueden subir o descargar imágenes a docker hub, pero también es posible guardar dichas imágenes en un archivo (con la extensión “.tar”) y cargalos en nuestro sistema.

Para guardar una imagen se usará el comando:

```
docker save -o “nombre-archivo.tar” imagen
```

Por ejemplo:

```
docker save -o ejemplo.tar busybox:uclibc
```

Para cargar una imagen desde un archivo se usará:

```
docker load -i “nombre archivo.tar”
```

Por ejemplo:

```
docker load -i ejemplo.tar
```