DOCKER

Instituto de Electrónica Aplicada
Carrera de Ingeniería Electrónica
Facultad de Ingeniería
Universidad Mayor de San Andrés.

CONTENIDO

1. Que es Docker. 1
2. Instalación de Docker 2
3. Contenedores vs. Imágenes 3
   3.1. Imagen 3
   3.2. Contenedores 3
   3.3. Diferencia entre contenedor e imagen 3
4. Contenedores residentes 4
5. Imágenes interactivas 4
6. Etiquetas a las imágenes 5
7. Algunos comandos importantes 5
8. Archivo Dockerfile 6
9. Red básica con contenedores 8
10. Borrar contenedores e imágenes 9
11. Servidores con Docker 9
   11.1. Apache y PHP 9
   11.2. MySql y PhpMyAdmin 10

1. Que es Docker.

Generalmente, para acceder o ejecutar correctamente una aplicación, se requiere tener instaladas una serie de otras aplicaciones. Docker, es un programa de código abierto que permite que una aplicación Linux y sus dependencias, se empaqueten formando lo que se denomina un contenedor y de esta manera, ejecutar la aplicación de forma correcta, sin la necesidad de preocuparse por configuraciones o la instalación de otros programas. Es decir, Docker crea contenedores ligeros y portátiles para que las aplicaciones y/o servicios puedan ser ejecutados en cualquier máquina que tenga instalado Docker, independientemente del sistema operativo con el que se cuente.

La diferencia fundamental entre Docker y la virtualización de máquinas, radica en que el contenedor generado por Docker, no es una máquina virtual. Un contenedor demanda menos

espacio de almacenamiento, mientras que, a una máquina virtual requiere instalar un sistema operativo para su funcionamiento, por su parte, un contenedor de Docker, recurre al sistema operativo que tiene la máquina en la que se ejecuta el contenedor, lo que permite que el contenedor de Docker tome los recursos más básicos del sistema operativo de la máquina en la que se ejecuta y aquellos aspectos específicos sean agregados en el interior del contenedor.

En el presente artículo, se desarrollarán los procedimientos básicos para el trabajo con Docker. Los ejemplos serán aplicados en un sistema operativo Linux, en su distribución Ubuntu versión 16.04.

2. Instalación de Docker

Dependiendo del tipo de instalación que se realizó, puede que sea importante agregar algunas propiedades al sistema, lo que permitirá la instalación de aplicaciones mediante repositorios.

```
sudo system.properties.common
```

Para la instalación de Docker en Ubuntu, en primer lugar, se debe actualizar la base de datos de paquetes.

```
sudo apt-get update
```

Es necesario agregar la clave GPG de Docker, en el repositorio oficial del sistema.

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

Además de incluir el repositorio Docker a las fuentes APT.

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

Como también, actualizar la base de datos de paquetes con los de Docker, desde el repositorio recién agregado:

```
sudo apt-get update
```

Es recomendable realizar la instalación desde el repositorio de Docker, en vez del repositorio predeterminado por Ubuntu 16.04:

```
apt-cache policy docker-engine
```

A continuación, se procederá con la instalación de Docker:

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

Docker estará instalado, el *daemon* iniciado y el proceso habilitado desde el arranque. Una prueba de que se está ejecutando Docker como servicio es:

```
sudo systemctl status docker
```

De forma predeterminada, ejecutar el comando *docker* requiere privilegios de *root*, es decir, se tiene que prefijar el comando con *sudo*. También puede ser ejecutado por un usuario que pertenezca al grupo *docker*, el cual se crea automáticamente durante la instalación de Docker.

Para dejar de escribir *sudo* cada vez que ejecute el comando *docker*, es necesario añadir el nombre del usuario al grupo *docker*:

```
sudo usermod -aG docker $(whoami)
```

Para agregar un usuario al grupo *docker*, en el que no se ha iniciado sesión, se debe declarar explícitamente el nombre de usuario a ser utilizado:

```
sudo usermod -aG docker username
```

El uso de Docker, consiste en enviar una cadena de opciones y comandos seguidos de argumentos. La sintaxis toma la forma de:

```
docker [option] [command] [arguments]
```

Si se desea ver todos los subcomandos disponibles:

```
docker
```

Para identificar la versión instalada, se puede consultar mediante:

```
docker --version
```

3. Contenedores vs. Imágenes

3.1.Imagen

Una imagen podría considerarse como plantilla o una captura del estado de un contenedor. Por ejemplo, una imagen podría contener el sistema operativo Ubuntu, con un servidor Apache y una aplicación web instalada. Las imágenes se utilizan para crear contenedores, y nunca cambian.

Existen muchas imágenes públicas con elementos básicos como Java, Ubuntu, Apache y otros, que se pueden descargar y utilizar.

El repositorio oficial de imágenes Docker es: *http://hub.docker.com*

3.2.Contenedores

El concepto de contenedor es como si se restaurara una máquina virtual a partir de una captura. Es decir, son instancias en ejecución de una imagen. Los contenedores son los que ejecutan instrucciones o aplicaciones. A partir de una única imagen, es posible ejecutar varios contenedores.

3.3.Diferencia entre contenedor e imagen

Como las imágenes no cambian, se crea un contenedor a partir de una imagen y mientras se esté ejecutando el contenedor y se cambia o instala alguna herramienta, al detener el contenedor y al volver a ejecutar la misma imagen, los cambios no se verán reflejados.

Si se desea obtener una lista de las imágenes en ejecución, se recurre al comando:

```
docker ps
```

Para tener una lista de todas las imágenes:

```
docker ps -a
```

O simplemente los identificadores de las imágenes:

```
docker ps -a -q
```

Cada imagen y contenedor genera un identificador, el cual consiste en un número tomado al azár por el sistema, que estará vigente mientras la imagen o el contenedor permanezcan en el equipo.

4. Contenedores residentes

Para iniciar un contenedor basta con ejecutar el comando *run*, como ejemplo se iniciará un contenedor que simplemente presentará la hora en la pantalla.

```
docker run jpetazzo/clock
```

Sin embargo, al momento de paralizar o cerrar el contenedor, éste se detiene y se restaura al punto inicial del arranque. Para que el contenedor se encuentre residente en memoria, se debe recurrir a la opción *-d*.

```
docker run -d jpetazzo/clock
```

Como se mencionó, todos los contenedores en ejecución son nombrados, de forma inicial, con un número de identificación (ID), el cual es generado de forma aleatoria. Para ver los registros históricos del contenedor se recurre a la opción *logs*.

```
docker logs id
```

Se indicará, de forma general, como *id* al identificador del contenedor, ya que este número variará entre las diferentes computadoras.

Para ver los últimos 5 o *n* registros del contenedor se procede con:

```
docker logs –tail 5 id
```

Si se desea conocer al último contenedor generado:

```
docker ps -l
```

o sólo el *id* de éste:

```
docker ps -l -q
```

Al igual que en Unix, es posible tener en pantalla los registros a medida que se vayan generado:

```
docker logs –tail 1 -f id
```

Y para detener un proceso:

```
docker kill id
```

5. Imágenes interactivas

Docker, ofrece una gran variedad de imágenes en su sitio oficial *hub.docker.com*, donde después del registro es posible descargar estas imágenes de acuerdo con los requerimientos del proyecto. Sin embargo, esto también puede realizarse a través de la línea de comando.

Por ejemplo, se consultará por la imagen *zookeeper*.

```
docker search zookeeper
```

O por la versión *jessie* de Debian.

```
docker pull debian:jessie
```

Para descargar una imagen, se recurre a la opción *pull*, que requiere del nombre de la imagen seguido por el identificador (tag).

A continuación, a modo de ejemplo, se instalará la imagen de una versión del sistema operativo Ubuntu y a ésta, se agregarán los comandos *wget* y *curl*.

```
docker images
docker run -it Ubuntu bash
```

Este último comando también puede ser ejecutado de la siguiente manera:

```
docker run --rm -it ubuntu bash
```

La opción *-it* solicita a Docker, una terminal interactiva y la opción *--rm* borra otros contenedores residentes que previamente hayan sido generados con la misma imagen.

Una vez obtenido el *prompt* del sistema operativo correspondiente al contenedor, se pueden ejecutar diversos comandos.

```
# wget
```

En la imagen oficial de Ubuntu, el comando *wget* no está disponible, por lo tanto, tampoco en el contenedor, sin embargo, es posible instalarlo, siempre y cuando el equipo tenga una conexión activa a Internet u otro repositorio local. Para ello, se procede con las mismas instrucciones que emplea el sistema operativo Ubuntu, ya que en realidad se está inmerso en un contenedor que representa a un equipo donde se ejecuta Ubuntu.

```
# apt-get update
# apt-get install -y wget
# wget
```

Ahora el programa *wget* sí estará disponible. Se puede repetir este ejercicio con el comando *curl*.

```
# apt install -y curl
# curl
# exit
```

Este nuevo contenedor difiere de la imagen original, puesto que, se instalaron dos aplicaciones que originalmente no estaban disponibles y para conocer estas diferencias, se puede ejecutar el comando:

```
docker diff id
```

En caso de que se desee transformar este contenedor en una imagen, es posible mediante:

```
docker commit id
```

De esta manera, se generará un nuevo identificador, tal que es posible recuperar la nueva imagen con los cambios realizados.

```
docker run -it nuevo_id bash
# wget
```

Ahora, el comando *wget* estará disponible para su utilización.

6. Etiquetas a las imágenes

Las nuevas imágenes, de forma inicial, se generan con un número de identificación aleatorio, pero es posible renombrarlas a través:

```
docker images
docker tar id nombre_de_la_imagen
```

7. Algunos comandos importantes

Reiniciar el servicio de Docker

```
sudo service docker restart
```

Listar a los contenedores activos
```
docker ps
```

Lista con todos los contenedores -*a* (all)
```
docker ps -a
```

Lista con los identificadores de todos los contenedores
```
docker ps -aq
```

Lista de los identificadores de aquellos contenedores activos
```
docker ps -q
```

Último contenedor ejecutado
```
docker ps -l
```

Remueve un contenedor por medio del *id*.
```
docker rm id
```

Lista de las imágenes disponibles en la máquina local
```
docker images
```

Lista con los identificadores de las imágenes disponibles en la máquina local
```
docker images -q
```

Remueve de forma obligatoria una imagen a través del *id*
```
docker rmi -f id
```

Presenta información acerca de la opción *rmi*
```
docker rmi --help
```

8. Archivo Dockerfile

Una opción muy útil de Doker, es la generación de una imagen “personalizada”, una que contenga todas las aplicaciones, opciones y configuraciones que se requieren para realizar un trabajo en particular. Esta nueva imagen, puede ser generada a partir de una ya existen, por medio del archivo *Dockerfile*. A continuación, se desarrollará un ejemplo donde se presentan los comandos más importantes para este fin.

Es recomendable ubicar una carpeta donde se generará la imagen, para este caso, se crea una carpeta con el nombre *mi_imagen*.
```
mkdir mi_imagen
cd mi_imagen
```

Dentro de la carpeta destino, se debe crear un archivo con el nombre *Dockerfile*. Este nombre debe ser respetado.
```
touch Dockerfile
```

Puede editarse el archivo *Dockerfile* con cualquier programa editor de texto.
```
nano Dockerfile
```

Una vista de *Dockerfile* es:

```
1    FROM ubuntu
2    RUN apt-get update
3    RUN apt-get install -y wget
```

En la línea 1, se evocará a la imagen del sistema operativo Ubuntu, si esta no existe en la máquina local, realizará la descarga de la misma.
En la línea 2, con la imagen ya descargada y en ejecución se procederá con la actualización.
En la línea 3, se instalará de forma desatendida el programa *wget*.
En cada uno de estos pasos se crea y destruye un contenedor antes de obtener la imágen final.

Para que el *batch* sea ejecutado se procede mediante:
```
docker build -t mi_ubuntu .
```

Revisando el listado de imágenes se puede evidenciar la presencia de una nueva, denominada *mi_ubuntu.*
```
docker images
```

Se puede ejecutar la nueva imagen a través de:
```
docker run --it mi_ubuntu bash
```

Para ver la bitácora de ejecución de la nueva imagen:
```
docker history mi_ubuntu
```

Y para eliminar la imagen de forma obligatoria:
```
docker rmi -f mi_ubuntu
```

Docker, permite la ejecución de comandos al momento de generar la imagen. Continuando con el ejemplo anterior, se procederá con la identificación del número IP a través de la página web *http://ifconfig.co/ip* (Se recomienda revisar el sitio web para conocer más acerca de las opciones que ofrece).

Editando el archivo *Dockerfile*, agregar la siguiente línea:
```
4    CMD wget -O- -q http://ifconfig.co/ip
```

Será necesario reconstruir la imagen o generar una nueva, este caso se generará una nueva bajo el nombre *mi_direccion_ip*.
```
docker build -t mi_direccion_ip
```

Se puede apreciar la nueva imagen
```
docker images
```

Y al momento de ejecutarla
```
docker run mi_direccion_ip
```

ésta responderá con una dirección IP, que dependerá de la conexión a Internet que se posea.

Sin embargo, la página web *http://ifconfig.co* tiene diversas opciones, por ejemplo: presenta la ciudad desde donde se realiza la consulta.
```
docker run mi_direccion_ip http://ifconfig.co/city
```

Pero la imagen creada presenta un error, debido a que el comando no puede ser ejecutado por el contenedor, para solucionar este inconveniente, se deben declarar variables de ingreso, lo que es posible también realizar a través del archivo *Dockerfile*.

Editando el archivo:
```
4    ENTRYPOINT [ “wget”, “-O-”, “-q”]
```

Como en el paso anterior, se regenera la imagen *mi_direccion_ip*
```
docker build -t mi_direccion_ip
```

Al momento verificar el envío de argumentos al contenedor, se tiene:
```
docker run mi_direccion_ip http://ifconfig.co/ip
```

Dando como resultado la dirección IP:
Dirección IP

```
docker run mi_direccion_ip http://ifconfig.co/city
```

Presentando la ciudad desde donde se realiza la consulta:
Ciudad

```
docker run mi_direccion_ip http://ifconfig.co/country
```

O el país:
País

Finalmente, si se desea acceder al *prompt* del contenedor, se debe ejecutar:
```
docker run -it --entrypoint bash mi_direccion_ip
```

9. Red básica con contenedores

Para demostrar el acceso a los contenedores por medio de los puertos de red, se empleará un servidor web básico como es del de *jpetazzo/web*
```
docker run -d –publish-all jptazzo/web
```

Si no existe la imagen en el equipo, ésta será descargada.

Con la opción *publish-all*, expone el puerto del contenedor a través de uno generado de forma aleatoria. Para identificar este puerto se ejecuta:
```
docker ps
```

En el ejemplo, se redireccionó el puerto 32768 al 8000. Por lo tanto, para acceder al servidor web se debe apuntar al URL y el puerto identificado:
http://*ip*:32768

Donde el *ip* corresponde a la dirección IP del equipo anfitrión.

También es posible asignar un número de puerto específico y no uno aleatorio. En este caso será el puerto 88
```
docker kill id
doker run -d -p 88:8000 jpetazzo/web
```

Para visualizar el contenido del servidor web:

```
curl localhost:88
```

Si no se conoce la dirección IP se puede ejecutar el comando:

```
docker inspect –format ‘{{.NetworkSetting.IPAddress}}’
```

Se tendrá como respuesta, por ejemplo:
172.17.0.2

10.Borrar contenedores e imágenes

Conociendo las diferencias entre un contenedor y una imagen, se puede borrar del repositorio, sea una imagen o un contenedor.

```
docker rm $(docker ps –all --filter=”status=exited” --quiet)
```

Con el comando anterior, se borrarán todos aquellos contenedores que tengan como estado *exited*.

Para borrar todas aquellas imágenes que no se utilizan, se puede ejecutar:

```
docker rmi $(docker images --filter=“dangling=true” --quiet)
```

En algunos casos, se debe indicar que el borrado sea de forma obligatoria o forzada.

```
docker rmi -f $(docker images)
```

11.Servidores con Docker

Para concluir, se procederá con la realización de dos ejemplos que demuestran la interconexión de contenedores y su interacción con carpetas locales.

11.1. Apache y PHP

En este primer ejercicio, se desea contar con un servidor web que soporte la programación PHP, donde las carpetas de almacenamiento y archivos de configuración estarán fuera del contenedor y radicarán en la máquina local.

La estructura del proyecto tendrá la siguiente distribución de archivos:

```
./Dockerfile
./apache_config.conf
./www/index.php
```

Editar el archivo Dockerfile

```
1    FROM ubuntu:latest
2    MAINTAINER Nombre del responsable
3    # Instalar apache, PHP y programas suplementarios: openssh-server, curl y lynx-cur empleados para depurar
     el contenedor
4    RUN apt-get update &&
            apt-get -y upgrade &&
            DEBIAN_FRONTEND=noninteractive apt-get -y install \
            apache2 php7.0 php7.0-mysql libapache2-mod-php7.0 \
            curl lynx-cur
5    # Habilita el apache mods
```

```
RUN a2enmod php7.0
RUN a2enmod rewrite
# Actualiza el archivo PHP.ini, habilita los tags de programación y un ingreso silencioso.
RUN sed -i "s/short_open_tag = Off/short_open_tag = On/"
/etc/php/7.0/apache2/php.ini
RUN sed -i "s/error_reporting = .*$/error_reporting = E_ERROR | E_WARNING |
E_PARSE/" /etc/php/7.0/apache2/php.ini
# Definición de las variables de entorno de Apache
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid
# Define el puerto de publicación
EXPOSE 80
# La carpeta www será enlazada a la carpeta /var/www/site del contenedor
ADD www /var/www/site
# Actualiza el sitio inicial con el archivo de configuración creado en el equipo local
ADD apache-config.conf /etc/apache2/sites-enabled/000-default.conf
# Se iniciará el servidor en modo residente
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

El archivo apache-config.conf

```
<VirtualHost *:80>
  ServerAdmin me@mydomain.com
  DocumentRoot /var/www/site

  <Directory /var/www/site/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    Order deny,allow
    Allow from all
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

El archivo www/index.php

```
<? echo "<p>Hello?</p>"; ?>
```

Para generar la imagen y ejecutar el contenedor:

```
docker build -t mi_sitio
docker run -p 8080:80 -d mi_sitio
docker run -it -p 8080:80 mi_sitio bash
docker run -it -p 8080:80 -v /folder_www:/var/www/site mi_sitio bash
# apache2ctl start
```

Para salir del contenedor sin detenerlo se debe presionar c*trl p q*

Y para reingresar, conociendo el *id* del contenedor

```
docker attach id
```

11.2. MySQL y PhpMyAdmin

Primero se deben descargar las imágenes del servidor de base de datos MySQL y el cliente PhpMyAdmin.

```
docker pull mysql/mysql
docker pull phpmyadmin/phpmyadmin
```

Una vez concluida la descarga, es bueno verificar la existencia de las imágenes dentro el directorio de Docker.

```
docker images
```

Para ejecutar la base de datos y el cliente se debe tomar en cuenta que, el usuario principal de MySQL requiere de una clave de acceso, esto se puede definir al momento de realizar la llamada al contenedor. Con la opción *-e* se puedes definir las variables de entorno, este caso la clave será 0000. Por otra parte, se está nombrando al contendor como *mysql* el que será empleado para acceder mediante el cliente PhpMyAdmin.

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=0000 -d mysql/mysql-server
```

Una vez que el contenedor *mysql* se encuentra en ejecución, resta acceder desde el cliente PhpMyAdmin, el que se encuentra en otro contendor, para ello, se emplea la opción *link*. Otro aspecto importante es el redireccionamiento de los puertos, en este caso se empleará el puerto 8080 local que apuntará al 80 del contenedor.

```
docker run --name myadmin -d --link mysql:db -p 8080:80 phpmyadmin/phpmyadmin
```

Para realizar una prueba y verificar el funcionamiento del servicio:
http://*dirección_ip*:8080

Como en los casos anteriores, es posible acceder al *bash* mediante

```
docker exec -i -t id bash
```

Un aspecto que debe tomarse en cuenta es el acceso a la base de datos como usuario *root*, generalmente este acceso se encuentra restringido a la dirección local (localhost), para que se pueda trabajar con la base de datos con este usuario se debe modificar la tabla *user*.

```
# mysql -uroot -p
Passwd 0000
Mysql> use mysql;
       Select host, user from user;
       > update user set host='%' where host='localhost'
```

Para terminar, si se desea ejecutar el contenedor con las bases de datos residentes en la máquina local, se puede emplear la opción *-v* y a continuación la dirección de la carpeta local, seguida de la carpeta correspondiente al contenedor.

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=0000 -v
/home/user/datadir/:/var/lib/mysql -d mysql/mysql-server
```

12. CONCLUSIÓN.

Docker, es una herramienta muy útil para el desarrollo de nuevas aplicaciones, ya que estas imágenes pueden ser compartidas entre un grupo de programadores y estos a su vez, pueden continuar con la evolución de un proyecto, realizando un trabajo colaborativo más eficiente.

Docker también es muy bueno para contar con entornos de pruebas. Es muy sencillo crear y borrar un contenedor, además de ser muy ligeros, por lo que es posible ejecutar varios contenedores en una misma máquina. Esto beneficia a la parte de sistemas, ya como los contenedores son más ligeros que las máquinas virtuales, se reduce el número de máquinas necesarias para contar con un entorno de desarrollo.

Ing. Ramiro Vladimir Mora Miranda
Docente Investigador