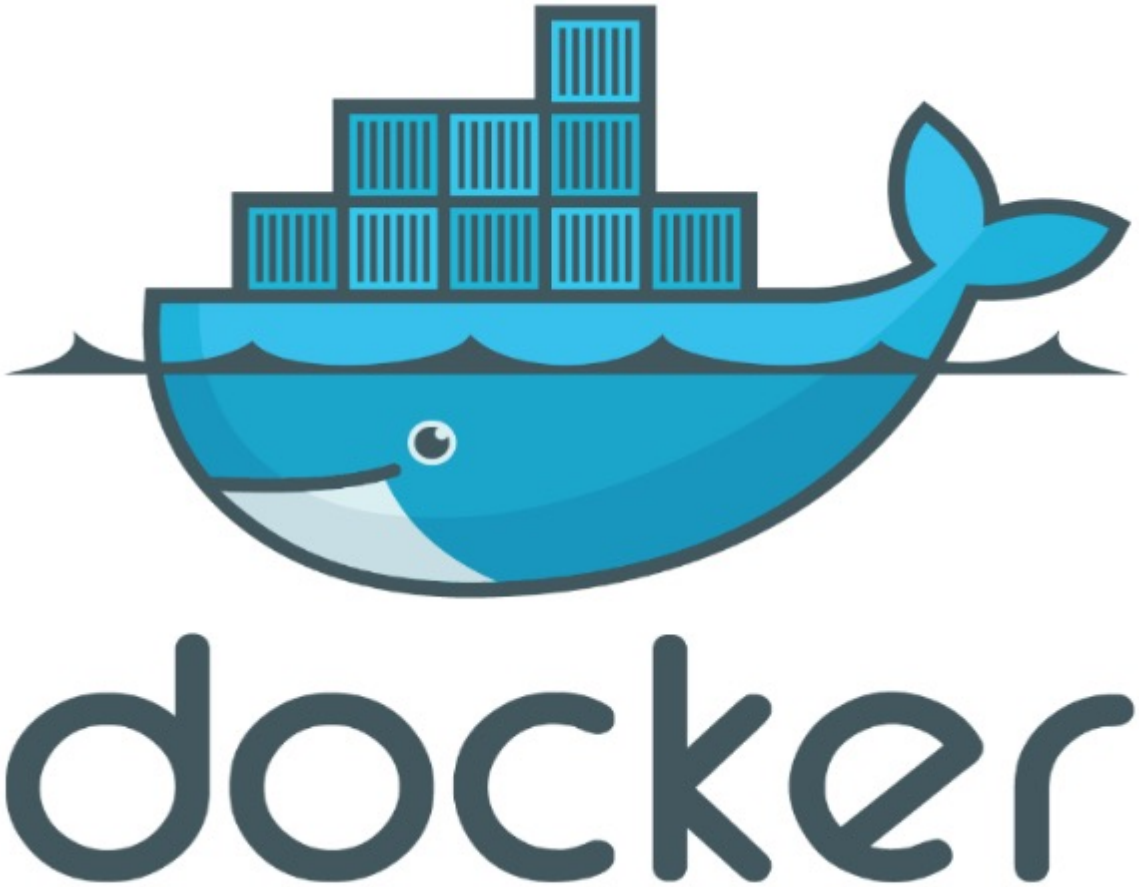
docker

DOCKER

1. Introducción a Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Explicado más claramente, cualquier usuario que quiera acceder a una aplicación software, necesita que se ejecute en un ordenador. Además, dependiendo de la aplicación, el ordenador también necesita ciertos requisitos software para que la aplicación se ejecute correctamente.

Docker permite meter en un contenedor (una caja cerrada) todo aquello que mi aplicación necesita para ser ejecutada junto a la propia aplicación. De esta forma puedo "transportar" este contenedor a cualquier máquina que tenga instalado Docker y ejecutar la aplicación sin tener que preocuparme de qué versiones de software tiene instalada esa máquina, o si tiene los elementos necesarios para que funcione mi aplicación. Ejecutando la aplicación software desde el contenedor de Docker, esta encontrará dentro de él todas las librerías y todo lo que necesita para funcionar correctamente.

1.1 Docker vs Máquinas virtuales

¿Por qué usar Docker en lugar de Maquinas Virtuales?

- Una máquina virtual proporciona un entorno con más recursos de los que necesitan la mayoría de las aplicaciones.
- Los contenedores se “arrancan” en fracciones de segundo, en lugar de minutos (como las VM).
- Los contenedores ocupan MB en lugar de GB en disco.
- Los contenedores permiten el despliegue de actualizaciones en caliente.
- Se pueden ejecutar de 10 a 100 veces más contenedores que máquinas virtuales en el mismo equipo.

Podemos usar lo mejor de ambos, ejecutando contenedores en un hosts que sea una máquina virtual.

1.2 Conceptos básicos

- Imagen. Una imagen es un «paquete ligero», capaz de ejecutarse sin depender de ningún otro software y que incluye todo lo necesario para ejecutar una aplicación. Las imágenes de Docker son una instantánea de un contenedor y los contenedores se crean a partir de una imagen.
- Contenedor. Un contenedor es una instancia en ejecución de una imagen que puede contener uno o más procesos ejecutándose. Para crear un contenedor solo hay que iniciar una imagen con el comando docker run.
- Docker Hub. Docker Hub es un repositorio donde están alojadas las imágenes base que podemos utilizar en nuestros contenedores. En Docker Hub pueden existir imágenes públicas y privadas.

  Podemos realizar búsqueda de imágenes desde la web oficial: https://hub.docker.com

  También desde consola con el comando docker search. Por ejemplo, para buscar todas las imágenes que contengan la palabra ubuntu usamos el comando: $ docker search ubuntu

  En Docker Hub podemos encontrar imágenes oficiales y otras creadas por miembros de la comunidad Docker.
- Dockerfile. Es un archivo de configuración para crear imágenes. Los comandos más habituales en un Dockerfile son:

FROM: Indica la imagen que vamos a utilizar. Primero buscará la imagen en local y si no la encuentra la descargará de Internet.

MAINTAINER: Datos de la persona que mantiene el contenedor.

RUN: Ejecuta una instrucción en el contenedor y hace un commit de los resultados.

ADD: Añade un archivo o un directorio al contenedor.

ENV: Nos permite configurar variables de entorno en el contenedor. Pueden ser sustituidas pasando la opción -env al usar el comando docker run. Ejemplo: docker run -env <key>=<valor>.

EXPOSE: Indica que el contenedor escucha en los puertos especificados durante su ejecución.

CMD: Solo puede existir una instrucción CMD en un Dockerfile, si colocamos más de uno, solo el último tendrá efecto. Esta instrucción nos permite indicar que se ejecuten instrucciones por defecto al iniciar un contenedor.

ENTRYPOINT: Nos permite indicar el comando que queremos que se ejecute de forma indefinida en nuestro contenedor. Si al iniciar un contenedor con docker run hacemos uso del parámetro -entrypoint podemos omitir los comandos especificados en esta instrucción.

- Volúmenes. Los volúmenes son el mecanismo que utiliza Docker para hacer persistentes los datos en un contenedor Docker.

1.3 Instalación de Docker

Para realizar la instalación de Docker es recomendable seguir la documentación oficial, donde podemos encontrar los pasos necesarios para su instalación en nacOS, Windows 10 y algunas distribuciones de servidores Linux. Para instalar docker tomando como SO base Debian 9, hay que seguir los siguientes pasos:

$ sudo apt-get update

- Instalar los paquetes que permitan a apt usar los repositorios sobre HTTPS:

$ sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common

- Añadir la GPG key oficial de Docker:

$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -

- Verificar que ahora se tiene la clave con la siguiente huella dactilar 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, buscándola mediante los 8 últimos caracteres.

$ sudo apt-key fingerprint 0EBFCD88
pub 4096R/0EBFCD88 2017-02-22
Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid Docker Release (CE deb) <docker@docker.com>
sub 4096R/F273FCD8 2017-02-22

- Añadir los repositorios de Docker para Debian:

$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"

- Instalar Docker:

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

Podemos comprobar la versión instalada de Docker con la siguiente opción:

$ docker --version
Docker version 18.03.1-ce, build 9ee9f40

Si ejecutamos hello-world, al no existir la imagen, la descargará y ejecutará:

$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

1.4 Un ejemplo sencillo. Crear un contenedor Apache

$ docker run -d -p 82:80 httpd

Descarga la imagen httpd, si no existe localmente y lanza el contenedor.

-d lanza el contenedor en modo detached, de no hacerlo así, el comando no devolverá el control.

82:80 asocia el puerto 82 al puerto 80 (el primer puerto que aparece es el del host y el segundo el del contenedor).

Para probarlo, usar un navegador con la dirección: http://ip_de_servidor:82

1.5 Imágenes interesantes de Docker.

En https://hub.docker.com/explore/ se encuentran las imágenes ordenadas por popularidad. Algunas destacadas son:

- alpine: Linux reducido
- nginx: Servidor web Nginx
- httpd: Servidor web Apache
- ubuntu: Ubuntu
- redis: Base de datos Redis (clave-valor)
- mongo: Base de datos MongoDB (documentos)
- mysql: Base de datos MySQL (relacional)
- postgres: Base de datos PostgreSQL (relaional)
- node: Node.js
- registry: Registro de imágenes on-premise
- php, elasticsearch, haproxy, wordpress, rabbitmq, python, openjdk, tomcat, jenkins, redmine, flink, spark, …

2. Operaciones sobre contenedores.

2.1 Mostrar Contenedores

$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d2f73e6acd51	httpd	"httpd-foreground"	11 minutes ago	Up 11 minutes	0.0.0.0:82->80/tcp	upbeat_stonebraker

Los nombres generados para los contenedores son aleatorios si no se usa el parámetro -name al crearlos.

- Mostrar todos los contenedores, también los detenidos

$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d2f73e6acd51	httpd	"httpd-foreground"	20 minutes ago	Exited (0) 2 minutes ago		upbeat_stonebraker

2.2 Detener y reanudar contenedores

- Primero, obtener con docker ps el CONTAINER ID del contenedor que queremos detener.

$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d2f73e6acd51	httpd	"httpd-foreground"	11 minutes ago	Up 11 minutes	0.0.0.0:82->80/tcp	upbeat_stonebraker

- Detener el contenedor

$ docker stop d2f73e6acd51

- Reanudar un contenedor

$ docker start d2f73e6acd51

- Tras reanudar el contenedor, vuelve a aparecer cuando hacemos docker ps

$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d2f73e6acd51	httpd	"httpd-foreground"	9 hours ago	Up 10 seconds	0.0.0.0:82->80/tcp	upbeat_stonebraker

- Detener todos los contenedores en ejecución

Primero obtenemos los identificadores de los contenedores en ejecución con docker ps -q. Ese comando lo podemos encerrar entre apóstrofes y pasar su resultado a otro comando en la misma línea.

$ docker stop `docker ps -q`

- Iniciar una lista de contenedores

$ docker start d2f73e6acd51 9811efbf6e45 178c2d03f2e7

2.3 Abrir un terminal en un contenedor

$ docker exec -it d2f73e6acd51 bash

root@d2f73e6acd51:/usr/local/apache2#

Se inicia una sesión como root en el contenedor. En la terminal del contenedor podemos ejecutar comandos del sistema operativo (ls, df -h, cat /proc/cpuinfo, …). La cantidad y el tipo de comandos dependerá de la imagen usada para crear el contenedor. Para abandonar la sesión bastará con escribir exit.

2.4 Copia de datos

El almacenamiento en un contenedor no es persistente. Se eliminan los datos escritos en él tras su eliminación.

docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-

docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH

Como ejemplo vamos a crear en nuestro host un archivo index.html y lo copiaremos en el contenedor para sustituir la página de inicio del servidor Apache.

```
<!-- Ejemplo de archivo index.html -->
<html>
  <body>
    <h1>Docker es una maravilla</h1>
  </body>
</html>
```

Ahora copiamos el archivo index.html al contenedor con docker cp

$ docker cp index.html d2f73e6acd51:/usr/local/apache2/htdocs/

2.5 Eliminación de un contenedor

- Primero paramos el contenedor con docker stop y luego lo eliminamos con docker rm

$ docker stop d2f73e6acd51

$ docker rm d2f73e6acd51

- También se puede eliminar directamente un contenedor en ejecución forzando su eliminación

$ docker rm -f <id>

Al crear un nuevo contenedor a partir de la imagen httpd comprobamos que la página de inicio modificada anteriormente se eliminó junto al contenedor eliminado.

$ docker run -d -p 82:80 httpd

2.6 Resumen de comandos básicos para contenedores

$ docker info

$ docker version

$ docker run <image> // Crea un contenedor a partir de una imagen. Si no tenemos la imagen en local, la descarga

$ docker run -d -p 82:80 nginx: Crea un contenedor en modo deattached accesible desde el puerto 82

$ docker stop|start <id>: Detiene|Continúa un contenedor

$ docker ps -a: Listado de contenedores (-a muestra también los parados)

$ docker ps -q: Listado de los ids de los contenedores

$ docker stop `docker ps -q`: Para todos los contenedores que devuelve el subcomando `docker ps -q`

$ docker rm <id>: Borra un contenedor si está parado

$ docker rm -f <id>: Fuerza el borrado de un contenedor aunque esté parado

$ docker exec -it <id> sh: Abre una terminal en el contenedor

$ docker exec <id> ls: Ejecuta el comando ls en el contenedor para mostrar sus archivos

$ docker cp <id>:./dockerenv .: Copia el fichero dockerenv del contenedor en nuestro sistema de archivos local

$ docker rm -f `docker ps -a | grep "wordpress" | awk '{print $1}'`: Eliminar todos los contenedores creados a partir de una imagen

3. Creación de imágenes propias

3.1 El Dockerfile

Para construir una imagen, se crea un Dockerfile con las instrucciones que especifican lo que va a ir en el entorno, dentro del contenedor (redes, volúmenes, puertos al exterior, archivos que se incluyen). Indicar cómo y con qué construir la imagen y conseguimos que el build de la aplicación definida en el contenedor se comporte de la misma forma en cualquier lugar que se ejecute. Hacemos que sea repetible.

- Fragmento de Dockerfile para construir una imagen con Ubuntu como base y definiendo dónde se montará un volumen externo

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev
WORKDIR /app
ENV DEBUG=True
EXPOSE 80
VOLUME /data
```

(La última línea crea un punto de montaje en el contenedor. A la hora de crearlo le haremos corresponder normalmente un directorio del host)

3.2 Imágenes

Se construyen con docker build a partir de un Dockerfile. Se crean en un contexto (normalmente añadiendo archivos del directorio de trabajo del host a la imagen -p.e. el código fuente de la aplicación). Con FROM (normalmente primera instrucción del Dockerfile) inicializamos el sistema de archivos de la imagen (p.e. si es ubuntu obtenemos el sistema de archivos de Ubuntu). Muchas imágenes disponibles en Docker Hub usan Alpine (una distribución ligera de Linux) en lugar de Ubuntu, Fedora o CentOS, debido a su menor tamaño. Cada instrucción del Dockerfile genera una nueva capa (con la diferencia) en ese sistema de archivos. Al hacer build las capas existentes en el registro local no se vuelven a crear.

(Una imagen comprimida de Alpine está en torno a los 2 MB, mientras que una imagen comprimida de Ubuntu está entre 40 y 80 MB)

3.3 Ejemplo de contenedor para aplicaciones web en PHP.

Un contenedor que incluya de forma estática una aplicación (p.e. la última versión de la aplicación). El proceso a seguir es:

* Creación de la aplicación.
* Creación del Dockerfile para generación de la imagen.
* Generación de la imagen.

- A partir de una carpeta nueva crearemos lo siguiente:

* Archivo Dockerfile
* Carpeta html con los scritps de nuestra aplicación
* Archivo html/index.php con el código de nuestra aplicación

- El Dockerfile

```
FROM tutum/apache-php
ADD html /var/www/html
EXPOSE 80
```

- Archivo html/index.php de ejemplo

```
<?php
  echo "Hola desde Docker";
?>
```

- Construcción de la imagen.

$ docker build -t pruebaphp .

Con -t definimos una etiqueta o nombre de la imagen. Al construir la imagen pasa a nuestro registro local.

- Listado de imágenes locales

$ docker image ls

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
pruebaphp	latest	152781e32617	14 hours ago	245MB

- Creación de un contenedor a partir de la imagen

$ docker run -d -p 83:80 pruebaphp

Un posible inconveniente que podemos encontrar en este ejemplo es que la aplicación va incluida en la propia imagen, por lo que para actualizar la aplicación deberemos crear una nueva imagen, y después crear un nuevo contenedor a partir de ella desechando el contenedor anterior.

- Ejemplo de contenedor con volumen externo

En este ejemplo la aplicación la tendremos aparte en un volumen externo accesible por el contenedor. De esta forma, si nuestra aplicación está vinculada a un repositorio, la actualización de la aplicación se realiza descargando la última versión del repositorio, manteniendo intacto el contenedor.

La forma de usar volúmenes con Dockerfile consiste en:

Añadir en el Dockerfile la lista de carpetas que se montarán con volúmenes externos

Al crear el contenedor indicar el punto de montaje en el host remoto en forma de ruta absoluta

El Dockerfile

FROM tutum/apache-php

VOLUME /var/www/html

EXPOSE 80

$ docker run -d -p 83:80 -v /home/daw/html:/var/www/html

3.4. Descarga y subida de imágenes a Docker Hub

Etiquetar la imagen antes de subirla a Docker Hub

$ docker tag phpprueba ualmtorres/phpprueba:v0

Subida de la imagen a Docker Hub

$ docker push <usuario>/<image>

Al hacer push las capas que ya estén subidas no se vuelven a subir. En cuanto una instrucción del Dockerfile cambia una capa, invalida al resto y hay que volver a generar las instrucciones de las capas restantes. Por tanto, colocaremos antes en el Dockerfile lo que menos cambie.

Al hacer pull sólo se descargan las capas nuevas.

Si cambiamos en el host archivos de los que se incluyen en la imagen se genera una capa nueva invalidando la caché.

$ docker pull wordpress

$ docker run -d -p 80:80 --name my_wordpress wordpress

3.5. Resumen de comandos básicos para imágenes

$ docker login

$ docker run -d ngninx

$ docker pull <image>

$ docker image ls: Lista imágenes locales

$ docker inspect <image>: Propiedades de una imagen

$ docker image rm <image>: Elimina una imagen local

4. Aplicaciones con varios contenedores

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker con varios contenedores.

Usaremos un archivo docker-compose.yml para configurar los servicios de la aplicación. Los servicios son las partes de la aplicación (p.e. un servicio para el almacenamiento de los datos y otro para el front-end)

En un mismo host podemos tener varios entonos aislados. Compose usa nombres de proyecto para mantener a los entornos aislados. De forma predeterminada se usa el nombre del directorio desde donde se lanza la aplicación.

para obtener la versión y saber si está instalado: $ docker-compose --version

Instalación de docker-compose en Debian 9: $ sudo apt install docker-compose

4.1. Flujo de trabajo básico con Docker Compose

Crear el archivo docker-compose.yml con los servicios de la aplicación (p.e. php y mysql)

Construir y lanzar el entorno en modo dettached con docker-compose up -d

Detener el entorno con docker-compose down

4.2. Comandos básicos para Docker Compose

$ docker-compose up -d Construye y lanza el entorno en modo dettached

$ docker-compose down Detiene el entorno

$ docker-compose pull Descarga las imágenes, pero no inicia los contenedores

$ docker-compose rm [-fs] Borra los contedores parados. Con -fs los detiene y fuerza su borrado

5. Integración de Docker en el módulo "Despliegue de Aplicaciones Web" de 2º DAW

En el tema 1 " Implantación de arquitecturas web", se podría probar el siguiente contenedor:
- Crear un contenedor Bind, añadiendo un volumen local

$ docker run --name bind -d --restart=always --publish 53:53/tcp --publish 53:53/udp --publish 10000:10000/tcp --volume /tmp:/data sameersbn/bind:9.11.3-20190315

(Para configurar el servidor DNS acceder via WEBMIN, puerrto 10000 con el usuario: root y contraseña: password)

En el tema 2 "Configuración y administración de servidores Web", se podrían probar los siguientes contenedores:

- Crear un contenedor Apache, añadiendo un volumen local

$ docker run -d -p 82:80 -v /home/daw/html:/usr/local/apache2/htdocs httpd

-v monta el volumen local /home/daw/html en la ruta /usr/local/apache2/htdocs del contenedor, de manera que podremos probar nuestro sitio web situándolo localmente en /home/daw

(probar en el navegador con la dirección: http://ip_de_servidor:82)

- Crear un contenedor MySQL con persistencia

```
$ docker run -d --rm --name mysql -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v $(pwd)/mysql:/var/lib/mysql mysql:5.7.22
```

```
(probar con $ mysql -u root -p -h 127.0.0.1)
```

- Crear un contenedor phpmyAdmin enlazado a mysql con persistencia

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

```
(probar en el navegador con la dirección: http://ip_de_servidor:8080)
```

En el tema 3 "Configuración y administración de servidores de aplicaciones", se podrían probar el siguiente contenedor:

- Crear un contenedor Tomcat, añadiendo un volumen local

$ docker run -it --rm -p 8888:8080 --volume /tmp/tomcat:/usr/local/tomcat/webapps tomcat:8.0

En el tema 4 "Instalación y administración de servidores de transferencia de archivos", se podrían probar el siguiente contenedor:

- Crear un contenedor Ftp, añadiendo usuarios manualmente

$ docker run -d --name ftpd_server -p 21:21 -p 30000-30009:30000-30009 -e "PUBLICHOST=localhost" -e "ADDED_FLAGS=-d -d" stilliard/pure-ftpd:hardened

$ docker exec -it ftpd_server sh -c "export TERM=xterm && bash"

(entramos en el contenedor docker)

$ pure-pw useradd bob -f /etc/pure-ftpd/passwd/pureftpd.passwd -m -u ftpuser -d /home/ftpusers/bob

(creamos manualmente el usuario bob, a continuación nos solicitará la contraseña 2 veces)

6. Fuentes

https://ualmtorres.github.io/SeminarioDockerPresentacion

https://guiadev.com/docker-vs-maquinas-virtuales-mejor

https://es.wikipedia.org/wiki/Docker_(software)