30/10/2025 08:10 1/7 docker (first contact)

docker

un poco de historia

- docker monta un sistema de containerización que permite lanzar nuevas instancias de S.O. aprovechando los recursos (hard) de la máquina Y el kernel del SO ya corriendo, lo que hace más óptimo el aprovechamiendo de recursos que las máquinas virtuales (que recrean un ordenador al completo, con sus propios kernels y sistemas de ficheros)
- Se basa en la tecnología LXC (LinuX Containers) presente en el kernel desde la versión 3.8
- · Grandes beneficios:
 - portabilidad de aplicaciones
 - aislamiento de procesos
 - prevenir la fragilidad del exterior
 - manejo de recursos
- Partes de funcionamiento de Docker:
 - docker daemon
 - o docker CLI
 - docker image index
- Elementos de Docker:
 - o contenedores
 - o imágenes
 - dockerfiles

enlaces de interés

- docs docker:
 - https://docs.docker.com/engine/tutorials/dockerizing/
 - 2. https://docs.docker.com/engine/tutorials/usingdocker/
 - 3. https://docs.docker.com/engine/tutorials/dockerimages/
 - 4. https://docs.docker.com/engine/tutorials/networkingcontainers/
 - 5. https://docs.docker.com/engine/tutorials/dockervolumes/
- otros enlaces de la red (básicos)
 - http://www.muylinux.com/2016/04/19/tutorial-docker
 - https://www.adictosaltrabajo.com/tutoriales/docker-for-dummies/
 - https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-getting-started
 - $\circ\ https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04$
- más enlaces (más avanzados)
 - https://www.digitalocean.com/community/tutorials/docker-explicado-como-crear-contenedores-dedocker-corriendo-en-memcached-es

instalación

- 1. Asegurarse que disponemos de https y certificados en APT:
 - \$ sudo apt-get install apt-transport-https ca-certificates
- 2. Añadir clave GPG de los repositorios DOCKER:
 - \$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-

keys 58118E89F3A912897C070ADBF76221572C52609D

3. Añadir un nuevo repositorio en /etc/apt/sources.list.d/docker.list:

```
deb https://apt.dockerproject.org/repo ubuntu-xenial main
```

4. Actualizar información respositorios, instalar:

```
$ sudo apt-get update ; sudo apt-get install docker.io
```

5. Añadir nuestro usuario al grupo de DOCKER para poder trabajar sin SUDO:

```
$ sudo usermod -a -G docker $USER
```

- 6. Refrescamos el grupo para poder trabajar sin cerrar sesión (este comando hace de docker nuestro grupo principal durante la sesión):
 - \$ newgrp docker

imágenes

Las imágenes son las plantillas a partir de las cuales creamos los contenedores de trabajo (como instanciar el objeto de una clase en OOP)

existe en https://www.docker.com/ un repositorio de imágenes para descargar, tanto oficiales de distros como de usuarios.

• para buscar imágenes disponibles en contenedores:

```
$ docker search <cadena>
```

• para descargar una imagen:

```
$ docker pull <nombre imagen>
```

• para ver las imágenes descargadas:

```
$ docker images
```

• para borrar una imagen:

```
$ docker rmi <nombre_imagen>
```

• añadir tag a imagen (crea copia de la imagen):

```
$ docker tag <nombre imagen> <usuario>/<nombre_imagen>:<tag>
```

Pendiente:

• \$ docker images --digest | head

30/10/2025 08:10 3/7 docker (first contact)

contenedores

El contenedor (en el principio de su existencia) coge la referencia de una imagen y se crea.

Se pueden crear tantos contenedores de la misma imagen como se precisen

mantenimiento (de contenedores)

• listar todos los contenedores existentes:

```
$ docker ps -a
```

• listar todos los contenedores activos:

```
$ docker ps
```

• listar los últimos contenedores creados:

```
$ docker ps -l
```

• parar un contenedor:

```
$ docker stop <nombre contenedor>
```

• borrar un contenedor (no ha de estar en funcionamiento):

```
$ docker rm <nombre_contenedor>
```

• convertir un contenedor en imagen:

```
$ docker commit -m "<comentario>" -a "<autor>" <id_contenedor>
<repositorio>:<nombre_imagen>
```

el nombre del contenedor puede ser su CONTAINER_ID o si NAME (según aparece en el listado generado con el comando correspondiente)

ejecución

• crear (si no existe) y ejecutar en modo interactivo (i) con pseudoterminal (t):

```
$ docker run -it <nombre contenedor>
```

• ejecutar en modo interactivo, borrar al salir:

```
$ docker run -it --rm <nombre_contenedor>
```

• poner en marcha un contenedor:

```
$ docker start <nombre_contenedor>
```

• poner en marcha contenedor como demonio (-d), mapeo de puerto y ejecución de comando:

```
$ docker run -d <nombre_imagen> -p <puerto_local>:<puerto_docker> <comando>
```

• poner en marcha contenedor con nombre propio:

- \$ docker run -d <nombre imagen> --name <nombre propio contenedor>
- poner en marcha contenedor mapeando los puertos aleatoriamente (se consulta en \$ docker ps:
 - \$ docker run -d -P <nombre imagen>
- acceder a un contenedor en marcha:
 - \$ docker attach <nombre contenedor>
- salir de un contenedor (detach), haciendo que continue funcionando: CONTROL+P + CONTROL+Q
- abrir un terminal en un contenedor:
 - \$ docker exec -it <nombre contenedor> bash

funcionamiento

- mostrar la salida (STDOUT) de un contenedor, con watch (estilo tail -f:
 - \$ docker -f logs <nombre_contenedor>
- mostrar procesos de un contenedor:
 - \$ docker top <nombre_contenedor>
- información del contenedor en formato JSON:
 - \$ docker inspect <nombre_contenedor>
- recoger una clave del JSON:
 - \$ docker inspect -f '<clave JSON>' <nombre_contenedor>

networking

Por defecto, el motor de docker crea una serie de redes para trabajar con los contenedores:

- host
- bridge

Aunque permite crear propias y añadir contenedores a ellas para que se hablen de manera aislada

- mostrar redes de docker:
 - \$ docker network ls
- inspeccionar red:
 - \$ docker network inspect < red>
- desconectar contenedor de una red:
 - \$ docker network disconnect < red> < nombre_contenedor>
- mapear puerto del contenedor, devuelve puerto de máquina local:

30/10/2025 08:10 5/7 docker (first contact)

```
$ docker port <nombre contenedor> <puerto>
```

• crear red propia (tipo bridge, flag -d):

```
docker network create -d bridge <nombre_red>
```

ejecutar contenedor con una red propia:

```
$ docker run -d --network=<nombre_red> <nombre_imagen>
```

• añadir contenedor en marcha a una red propia:

```
$ docker network connect <nombre_red> <nombre_contenedor>
```

• averiguar IP de contenedor:

```
docker inspect --format='{{range
   .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <nombre_contenedor>
```

Data Volumes

es un directorio especificado trabajando en UFS¹⁾ que:

- los Volumenes se inicializan al crear el contenedor. Si existe información en el punto de montaje, esta se copiará al Volumen
- los Volumenes se puede compartir y reusar entre contenedores
- los cambios se aplican directamente en los Volumenes
- los cambios en los Volumenes no se aplican cuando se actualiza una imagen
- los Volumenes no son destruidos al destruir el contenedor que los usa
- añadir un Volumen a un contenedor al vuelo:

```
$ docker run -d -P --name web -v /webapp training/webapp
```

• crear un Volumen para su posterior uso:

```
$ docker volume create -d <plugin> -o size=20GB <nombre_volumen>
```

- «flocker» es un plugin de Volumen que provee de multi-hosts
- o lista de plugins para la creación de Volumenes
- añadir Volumen existente a contenedor:

```
$ docker run -d -P -v <nombre_volumen>:/<path_contenedor> ....
```

• añadir un directorio local en un contenedor, con opción de solo-lectura (ro):

```
$ docker run -d -P --name web -v <path_local>:<path_contenedor>[:ro]
training/webapp python app.py
```

• añadir fichero local en contenedor:

```
$ docker run --rm -it -v ~/.bash_history:/root/.bash_history <nombre_imagen>
<comando>
```

 crear Volumen usando contenedor y compartir entre varios contenedores. El directorio será /dbdata. Se pueden añadir varios –volumes. Permite contatenar entre los diferentes contenedores (db3):

```
$ docker create -v /dbdata --name dbstore training/postgres /bin/true
$ docker run -d --volumes-from dbstore --name db1 training/postgress
$ docker run -d --volumes-from dbstore --name db2 training/postgress
$ docker run -d --volumens-from db1 --name db3 training/postgress
```

- mostrar Volumenes:
 - \$ docker volume ls
- mostrar Volumenes clgados (huérfanos):

```
$ docker volume ls -f dangling=true
```

borrar volúmenes:

```
$ docker volume rm <nombre_volumen>
  * borrar Volumenes con el borrado del contenedor:<code bash>$ docker rm -v
<nombre_contenedor>
```

Pendiente:

Volume labels

otros comandos

```
$ docker version
```

\$ docker info

dockerfiles

los «dockerfiles» son ficheros de procesamiento que le dicen a docker como construir una nueva imagen de manera «desatendida»

formato del dockerfile:

```
FROM ubuntu
MAINTAINER <autor>
ENV http_proxy http://user:pass@proxy/
ENV https_proxy http://user:pass@proxy/
RUN apt-get update
RUN apt-get install apache2 -y
RUN echo "<h1>Apache with Docker</h1>" > /var/www/html/index.html
EXPOSE 80
ENTRYPOINT apache2ctl -D FOREGROUND
```

- 1. FROM: imagen base
- 2. MAINTAINER: autor de la imagen
- 3. ENV: variables de entorno en la imagen base
- 4. RUN: ejecuta una sentencia en la imagen base
- 5. EXPOSE: abrimos el puerto especificado en el contenedor para que se pueda mapear desde el anfitrion
- 6. ENTRYPOINT: que se debe ejecutar cada vez que se ejecute el contenedor

\$ docker build -t <autor>/<imagen> <path_fichero_docker>

repositorio de imágenes

para subir una imagen a los repositorios públicos (Docker Hub)

• autentificación con docker.com:

```
$ docker login -u <usuario>
```

• subir imagen a repositorio:

```
$ docker push <usuario>/<nombre_imagen>
```

1

Union File System - https://docs.docker.com/engine/reference/glossary/#union-file-system

From:

https://miguelangel.torresegea.es/wiki/ - miguel angel torres egea

Permanent link:

https://miguelangel.torresegea.es/wiki/linux:docker:start?rev=1476920680

Last update: 19/10/2016 16:44

