1 Docker

Docker es una plataforma de código abierto diseñada para facilitar la creación, implementación y ejecución de aplicaciones en contenedores.

Un contenedor se puede definir como un entorno ligero, aislado y portable, que contiene todo lo necesario (código fuente, dependencias, etc.) para ejecutar una aplicación

Un contenedor suele tener un único procesos en ejecución, aunque es posible tener varios.

Una de las ventajas que aporta el uso de contenedores es que garantiza que una aplicación se ejecute de la misma manera en cualquier entorno.

Referencias:

- ¿Qué es Docker?.

1.1 STACK de Contenerización

La siguiente tabla muestra qué lugar ocupa Docker en el stack de contenerización.

	Ejemplos
Plataforma	OpenShift, Docker Enterprise Edition, Rancher, DC/OS
Orquestador	Kubernetes, Docker Swarm, Mesos
Motor de contenerización	Docker, rkt, LXD, cri-o
Sistema Operativo	Windows, Linux, macOS

1.2 Tecnologías de contenerización

Docker no es la única tecnología de contenerización que existe. A continuación se enumeran algunas de las más conocidas.

- Docker
- Podman
- rkt

- LXD
- cri-o

Referencias:

- ¿Qué es un contenedor de Linux?.

1.3 Orquestador

Entre los orquestadores más conocidos se encuentran:

- Kubernetes
- Docker Swarm
- Mesos

Referencias:

- ¿Qué es Kubernetes?.

1.4 Plataforma

También existen plataformas de contenedores que integran un orquestador y un motor de contenerización. Estas herramientas ofrecen un conjunto de herramientas y servicios para facilitar el despliegue de aplicaciones en contenedores.

- OpenShift
- Docker Enterprise Edition
- Rancher
- DC/OS

1.5 Conceptos básicos

1.5.1 Imagen

Podemos decir que las imágenes de Docker son **una instantánea de un contenedor** y que los contenedores se crean a partir de una imagen.

1.5.2 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`.

1.5.3 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.

Para realizar la búsqueda de imágenes podemos hacerlo **desde la web oficial**:

https://hub.docker.com

También podemos hacerlo **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 que han sido creadas por miembros de la comunidad Docker para todo tipo de propósitos.

1.5.4 Dockerfile

Es un **archivo de configuración** para crear imágenes.

Ejemplo:

```
#
# Nginx Dockerfile
#
# https://github.com/dockerfile/nginx
#

# Pull base image.
FROM dockerfile/ubuntu

# Install Nginx.
RUN \
  add-apt-repository -y ppa:nginx/stable && \
  apt-get update && \
  apt-get install -y nginx && \
  rm -rf /var/lib/apt/lists/* && \
  echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
  chown -R www-data:www-data /var/lib/nginx

# Define mountable directories.
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/
    var/log/nginx", "/var/www/html"]

# Define working directory.
WORKDIR /etc/nginx

# Define default command.
CMD ["nginx"]

# Expose ports.
EXPOSE 80
EXPOSE 443
```

1.5.5 Volúmenes

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

Referencia.

1.6 Instalación de Docker

Para realizar la instalación de Docker se recomienda seguir la documentación oficial.

Si has instalado Docker sobre Linux, tendrás que realizar alguna configuración adicional. Se recomienda seguir la documentación oficial sobre los pasos que hay seguir tras una instalación de Docker en Linux.

1.7 Imágenes en Docker

En esta sección vamos a ver los comandos básicos para trabajar con imágenes Docker.

1.7.1 `docker search`

Este comando nos permite buscar imágenes en Docker Hub.

Ejemplo:

Por ejemplo, para buscar todas las imágenes que contengan la palabra *ubuntu* usamos el comando:

```
docker search ubuntu
```

```
NAME                                                    DESCRIPTION
                                        STARS               OFFICIAL
    AUTOMATED
ubuntu                                                  Ubuntu is a Debian-based
     Linux operating …sys   8763                [OK]
dorowu/ubuntu-desktop-lxde-vnc                          Ubuntu with openssh-
    server and NoVNC            242                                 [OK]
...
```

Ejemplo:

Para buscar todas las imágenes que contengan la palabra *wordpress* ejecutaríamos el siguiente comando.

```
docker search wordpress
```

```
NAME                DESCRIPTION                             STARS   OFFICIAL
    AUTOMATED
wordpress           The WordPress rich content mana...  1983    [OK]
bitnami/wordpress   Bitnami Docker Image for WordPress  51                  [OK
    ]
...
```

1.7.2 docker pull

Este comando nos permite descargar una imagen de Docker Hub.

Ejemplo:

Por ejemplo, para descargarnos la imagen `ubntu` ejecutaríamos lo siguiente.

```
1 docker pull ubuntu
```

Ejemplo:

Para descargarnos la imagen `wordpress` haríamos lo siguiente.

```
1 docker pull wordpress
```

1.7.3 docker images

Muestra un listado con todas las imágenes locales disponibles.

Ejemplo:

Para ver el listado de de las imágenes que tenemos descargadas en nuestro equipo, ejecutaríamos el siguiente comando.

```
1 docker images
```

```
1 REPOSITORY          TAG                 IMAGE ID            CREATED
                SIZE
2 wordpress           latest              fcf3e41b8864        2 weeks ago
             408MB
3 ubuntu              latest              2d696327ab2e        2 months ago
            122MB
```

Ejemplo:

El modificador `-q` nos permite mostrar solamente el identificador de la imagen en el listado de salida. Esta opción nos será de utilidad cuando quiera eliminar todas las máquinas de forma masiva.

```
1 docker images -q
```

```
1 fcf3e41b8864
2 2d696327ab2e
```

1.7.4 docker rmi

Este comando nos permite eliminar una o varias imágenes.

Por ejemplo, para eliminar la imagen *wordpress* usamos:

```
1 docker rmi wordpress
```

1.7.5 docker rmi $(docker images -q)

Este comando nos permite eliminar todas las imágenes que tenemos en local.

```
1 docker rmi $(docker images -q)
```

1.8 Creación de contenedores en Docker

Para crear contenedores en Docker se utiliza el comando `docker run`.

Existen dos formas de crear un contenedor en Docker:

- `docker run -it`: Crea contenedores en **modo interactivo** que se ejecutan en primer planno y que nos permiten interactuar con ellos a través de la entrada estándar STDIN.
- `docker run -d`: Crea contenedores en **modo detached** con que se ejecutan en segundo plano.

1.9 Creación de contenedores en modo interactivo

En esta sección vamos a ver algunos ejemplos de cómo crear contenedores en modo interactivo.

1.9.1 Creación de un contenedor con Alpine Linux

Alpine Linux es una distribución Linux muy ligera. La imagen de Alpine Linux para Docker ocupa menos de 5 MB.

```
1 REPOSITORY          TAG                 IMAGE ID            CREATED
                SIZE
2 alpine              latest              196d12cf6ab1        2 months ago
          4.41MB
```

El gestor de paquetes de Alpine Linux es `apk`. En la documentación oficial podemos encontrar más detalles sobre cómo usarlo.

Ejemplo:

```
1 docker run -it --name alpine-container --rm alpine
```

- `docker run` es el comando que nos permite crear un contenedor a partir de una imagen Docker.
- El parámetro `-i` nos permite mantener interaccionar con el contenedor a través de la entrada estándar STDIN.
- El parámetro `-t` nos asigna un terminal dentro del contenedor.
- Los dos parámetros `-it` nos permiten usar un contenedor como si fuese una máquina virtual tradicional.

- El parámetro `--name` nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre Docker nos asignará un nombre automáticamente.
- El parámetro `--rm` hace que cuando salgamos del contenedor, éste se elimine y no ocupe espacio en nuestro disco.
- `alpine` es el nombre de la imagen. Si no se indica lo contrario buscará las imágenes en el repositorio oficial Docker Hub.

Una vez ejecutado el comando anterior nos aparecerá un terminal del contenedor que acabamos de crear.

```
/ #
```

Si quisiéramos instalar `nano` en el contenedor tendríamos que hacer lo siguiente.

1) Actualizar el índice de paquetes disponibles

```
apk update
```

2) Añadir el nuevo paquete al sistema.

```
apk add nano
```

Para salir del contenedor escribimos el comando `exit`.

```
exit
```

Como hemos iniciado el contenedor con el parámetro `--rm`, al salir del contenedor, éste se elimina y no ocupa espacio en nuestro disco. Podemos comprobarlo con siguiente comando.

```
docker ps -a
```

1.9.2 Creación de un contenedor con Ubuntu

```
docker run -it --name ubuntu --rm ubuntu
```

1.9.3 Creación de un contenedor con Nginx

```
docker run -it --name webserver --rm -p 80:80 nginx
```

1.10 Creación de contenedores en modo *detached*

En esta sección vamos a ver algunos ejemplos de cómo crear contenedores en modo *detached*.

1.10.1 Creación de un contenedor con Nginx en modo *detached*

```
docker run -d --name webserver --rm -p 80:80 nginx
```

Con el parámetro `-d` indicamos que queremos ejecutar el contenedor en background.

Con el parámetro `-v` podemos crear un volumen para mapear un directorio de nuestro equipo con el directorio que utiliza Nginx para servir las páginas webs.

También podemos hacer uso de `$(pwd)` para indicar que queremos crear un volumen en nuestro directorio actual.

```
docker run -d --name webserver --rm -p 80:80 -v $(pwd):/usr/share/nginx/html
    nginx
```

1.10.2 Creación de un contenedor con MySQL en modo *detached*

```
docker run -d --name mysql --rm -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v /
    home/josejuan/data:/var/lib/mysql mysql:5.7.22
```

Podemos hacer uso de `$(pwd)` para indicar que queremos crear el volumen en nuestro directorio actual.

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

1.10.3 Creación de un contenedor con Adminer en modo *detached*

En primer lugar debe existir un contenedor con MySQL Server.

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

Una vez que la instancia de MySQL está en ejecución podemos crear el contenedor con Adminer.

```
docker run --link mysql:db -p 8080:8080 adminer
```

1.10.4 Creación de un contenedor con PostgreSQL en modo *detached*

```
docker run -d --name postgresql --rm -p 5432:5432 -e POSTGRES_PASSWORD=
    mysecretpassword postgres
```

Nota: El nombre de usuario para conectar con el servidor PostgreSQL es `postgres`.

1.10.5 Creación de un contenedor con pgadmin4 en modo *detached*

Este contenedor lanza pgAdmin 4, una aplicación web que nos permite administrar una base de datos PostgreSQL.

```
docker run -p 80:80 \
-e "PGADMIN_DEFAULT_EMAIL=user@domain.com" \
-e "PGADMIN_DEFAULT_PASSWORD=SuperSecret" \
-d dpage/pgadmin4
```

1.11 Ejecución de comandos en un nuevo contenedor

1.11.1 `docker run`

El comando `docker run` nos permite ejecutar un comando en un contenedor.

Por ejemplo, para ejecutar el comando `cat /etc/os-release` en el contenedor *ubuntu* haríamos lo siguiente.

```
docker run ubuntu cat /etc/os-release
```

Y como salida tendríamos el siguiente resultado.

```
NAME="Ubuntu"
VERSION="18.04.1 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.1 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-
    policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

El contenedor finaliza su ejecución una vez que ha finalizado la ejecución del comando.

1.12 Ejecución de comandos en un contenedor que está en ejecución

1.12.1 `docker exec`

Nos permite ejecutar comandos concretos en un contenedor o abrir un terminal como si fuera una máquina virtual.

Ejemplo:

Permite ejecutar un comando en un contenedor que se está ejecutando.

```
docker exec -it webserver ls -la
```

Ejemplo:

Podemos lanzar como comando `/bin/sh` para abrir una consola e interactuar con el contenedor como si fuera una «máquina virtual».

```
docker exec -it webserver /bin/sh
```

1.13 Administración de contenedores

1.13.1 docker ps

Este comando muestra todos los contenedores **que hay en ejecución**.

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED
                STATUS              PORTS               NAMES
```

1.13.2 docker ps -a

Muestra todos los contenedores, los que están ejecución y los que están detenidos.

```
docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
                STATUS                      PORTS               NAMES
cfc8008e704b        ubuntu              "/bin/echo 'Hello ..."   7 seconds ago
          Exited (0) 5 seconds ago                              boring_almeida
```

1.13.3 docker stop

Permite detener un contenedor que está en ejecución.

En este ejemplo estaría deteniendo un contenedor con el id `abc1102e802c`.

```
docker stop abc1102e802c
```

También puedo detener todos los contenedores que hay en ejecución con el siguiente comando.

```
docker stop $(docker ps -a -q)
```

1.13.4 docker start

Permite iniciar un contenedor que está detenido.

1.13.5 docker rm

Para eliminar un contenedor que no está en ejecución referenciado por el nombre `wordpress` usamos:

```
1 docker rm wordpress
```

También podemos eliminarlo indicando su id. Por ejemplo:

```
1 docker rm 99ed74b743ec
```

Para eliminar todos los contenedores que no están ejecución.

```
1 docker rm $(docker ps -a -q)
```

1.13.6 docker logs

Muestra información de log de un contenedor.

1.13.7 docker inspect

Muestra información de bajo nivel de una imagen o un contenedor.

2 Dockerfile

Es un **archivo de configuración** para crear imágenes.

Ejemplo:

```
#
# Nginx Dockerfile
#
# https://github.com/dockerfile/nginx
#

# Pull base image.
FROM dockerfile/ubuntu

# Install Nginx.
RUN \
  add-apt-repository -y ppa:nginx/stable && \
  apt-get update && \
  apt-get install -y nginx && \
  rm -rf /var/lib/apt/lists/* && \
  echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
  chown -R www-data:www-data /var/lib/nginx

# Define mountable directories.
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/
    var/log/nginx", "/var/www/html"]

# Define working directory.
WORKDIR /etc/nginx

# Define default command.
CMD ["nginx"]

# Expose ports.
EXPOSE 80
EXPOSE 443
```

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.

3 Docker Compose

3.1 Comandos básicos

- `docker compose up`. Crea e inicia los contenedores.
- `docker compose up -d`. Crea e inicia los contenedores en modo *detach*.
- `docker compose down`. Detiene los contenedores que están en ejecución.
- `docker compose down -v`. Detiene los contenedores que están en ejecución y elimina los volúmenes.
- `docker compose ps`. Muestra los contenedores que están ejecución.
- `docker compose ps -a`. Muestra todos los contenedores incluyendo los que están detenidos.
- `docker compose logs`. Muestra las últimas líneas de los archivos de *logs* de los contenedores.
- `docker compose logs -f`. Muestra los *logs* de los contenedores en tiempo real.
- `docker compose exec`. Permite ejecutar un comando dentro de un contenedor.
- `docker compose start`. Inicia los contenedores que están detenidos.
- `docker compose stop`. Detiene los contenedores que están en ejecución.
- `docker compose build`. Reconstruye los contenedores.