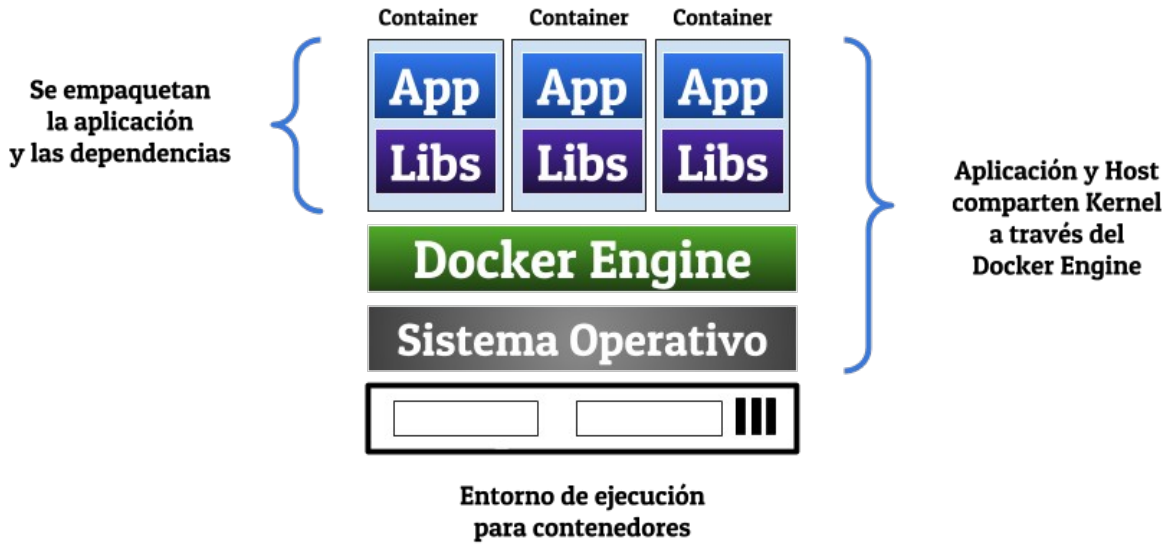
Mini apuntes docker

Introducción

El siguiente paso en la evolución, fue la aparición de los **CONTENEDORES**, eso que anteriormente hemos llamado "**máquinas virtuales ligeras**" . Su arquitectura general se puede ver en la siguiente imagen:



Juan Diego Pérez Jiménez. *Entorno de ejecución basado en contenedores* (Dominio público)

Y sus principales características son las siguientes:

- Los contenedores utilizan el **mismo Kernel Linux** que la máquina física en la que se ejecutan gracias a la estandarización de los Kernel y a características como los Cgroups y los Namespaces. Esto **elimina la sobrecarga** que en las máquinas virtuales suponía la carga total del **sistema operativo invitado.**
- Me permiten **aislar las distintas aplicaciones** que tenga en los distintos contenedores (salvo que yo estime que deban comunicarse).
- **Facilitan la distribución de las aplicaciones** ya que éstas se empaquetan junto con sus dependencias y pueden ser ejecutadas posteriormente en cualquier sistema en el que se pueda lanzar el contenedor en cuestión.
- Se puede pensar que se añade una capa adicional el **Docker Engine**, pero esta capa **apenas añade sobrecarga** debido a que se hace uso del mismo Kernel.

Instalación

Consultar la documentación oficial
https://docs.docker.com/engine/install/

Ejecutar Docker como un usuario que no sea root, es decir, ejecutarlo sin sudo. Esto es fundamental porque si no trabajar con Docker llega a ser un engorro.
Habilitar o deshabilitar el servicio Docker al inicio, según nos interese. Por defecto el servicio se habilita al inicio y la sobrecarga sobre el sistema es mínima.

```
# Crear el grupo docker si no se ha creado durante la instalación
> sudo groupadd docker
# Añadir nuestro usuario al grupo creado en el apartado anterior
> sudo usermod -aG docker $USER
# Salir de sesión o reiniciar (en algunas máquinas virtuales). Puedo también activar los cambios a los grupos con la siguiente orden
> newgrp docker
```

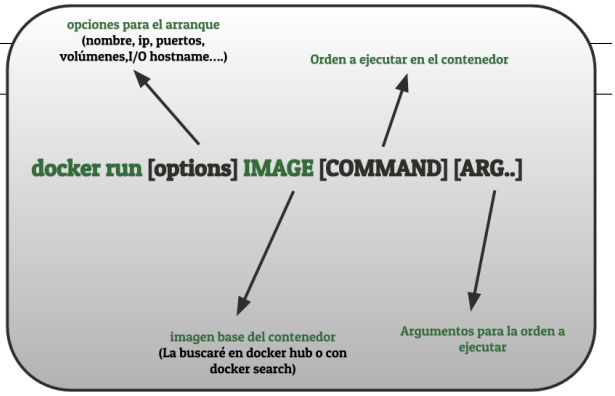
```
# Si quiero habilitar el servicio docker al iniciar el sistema. (recomendado para desarrollo)
> sudo systemctl enable docker
# Si quiero que el servicio docker no esté habilitado.
> sudo systemctl disable docker
```

Las imágenes se descargan desde un REGISTRO de imágenes que es un "almacén en la nube" donde los usuarios pueden, entre otras cosas crear, probar, almacenar y distribuir imágenes. Por defecto cuando instalamos docker el registro que vamos a usar es DockerHub que además de todo lo anterior tiene muchas más funcionalidades.

```
#Para descargar imágenes doker pull nombreimagen:versión si no ponemos versión escoge latest
# Descargar una imagen de manera previa
> docker pull ubuntu:18.04
```

Gestión y Ejecución de contenedores

La orden fundamental para ejecutar contenedores que es docker run cuya función principal es poner en ejecución contenedores en base a una imagen de referencia que le indicaremos.Una CUESTIÓN IMPORTANTE que debemos de tener en cuenta al usar docker run es que si vamos a ejecutar un contenedor que usa como base una imagen que no tenemos ésta se descargará de manera automática. Para buscar las imágenes que queremos la opción que os recomiendo es usar el buscador de Docker Hub.




```
Las opciones que tiene docker run son alrededor de 100, las que yo considero
que son más importantes:
  -d o --detach para ejecutar un contenedor (normalmente porque tenga un servicio) en background.
  -e o --env para establecer variables de entorno en la ejecución del contenedor.
  -h o --hostname para establecer el nombre de red parar el contenedor.
  --help para obtener ayuda de las opciones de docker.
  --interactive o -i para mantener la STDIN abierta en el contenedor.
  --ip si quiero darle una ip concreta al contenedor.
  --name para darle nombre al contenedor.
  --net o --network para conectar el contenedor a una red determinada.
  -p o --publish para conectar puertos del contenedor con los de nuestro host.
  --restart que permite reiniciar un contenedor si este se "cae" por cualquier motivo.
  --rm que destruye el contenedor al pararlo.
  --tty o -t para que el contenedor que vamos a ejecutar nos permita un acceso a un terminal para poder ejecutar órdenes en él.
  --user o -u para establecer el usuario con el que vamos a ejecutar el contenedor.
  --volume o -v para montar un bind mount o un volumen en nuestro contenedor.
  --workdir o -w para establecer el directorio de trabajo en un contenedor.
```

```
# Descargar una imagen de manera previa
> docker pull ubuntu:18.04

# Crear un contenedor de ubuntu:18.04 y tener acceso a un shell en él. Si no hemos descargado la imagen de manera previa se descargará.
> docker run -it ubuntu:18.04 /bin/bash
root@ef2bea1d6cb1:/#

# Crear un contenedor de centOs:18.04 y listar el contendido de la carpeta /
> docker run ubuntu:18.04 ls /
bin etc lib lost+found mnt proc run srv tmp var
dev home lib64 media opt root sbin sys usr

# Crear un contenedor de httpd (Servidor Apache)
> docker run httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.6. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.6. Set the 'ServerName' directive globally to suppress this message
[Mon Dec 07 10:01:52.670809 2020] [mpm_event:notice] [pid 1:tid 140412541457536] AH00489: Apache/2.4.46 (Unix) configured -- resuming normal operations
[Mon Dec 07 10:01:52.670973 2020] [core:notice] [pid 1:tid 140412541457536] AH00094: Command line: 'httpd -D FOREGROUND'

# Crear un contenedor de debian 9 y mostrar el contenido de una carpeta establecida con el parámetro -w
```

```
> docker run -it -w /etc debian:9 ls

# Mostrar los contenedores en ejecución (Estado Up)
> docker ps

# Mostrar todos los contenedores creados ya estén en ejecución (Estado Up) o parados (Estado Exited)
> docker ps -a
```

La ejecución de un docker es efímera. Siempre usar el flag -it al ejecutar una orden docker run si es un contenedor que no tiene servicios. Este -it es la unión del flag -i (--interactive) y el flag -t (--tty) que lo que hacen es abrir la entrada estándar del contenedor que estamos ejecutando y permitir la posibilidad de abrir un terminal en el contenedor. Es decir, nos va a permitir interactuar con él.
No debemos añadir al final otra orden que no sea /bin/bash (u otro shell que puedan contener los contenedores) ya que eso sobreescribe la orden de arranque de algunos contenedores (shell o inicio de servicio dependiendo del tipo de contenedor) y será imposible volver a arrancarlos una vez parados. Hay que tener en cuenta que una vez arrancados, si lo hemos hecho bien, siempre podremos ejecutar órdenes en el contenedor, tal y como veremos posteriormente.
Tras salir la primera vez de interactuar con ellos la ejecución acaba, para iniciarlos de nuevo, tenemos la orden docker start cuyo funcionamiento es muy simple.

```
# Arrancar el contenedor con nombre servidorWeb (debe estar parado)
> docker start servidorWeb

# Arrancar el contenedor con ID 511eeed5992d
> docker start 511eeed5992d
```

Ejecutamos servicios con puertos y variables de entorno y nombre al contenedor

```
# Ejecuto un servidor Apache sin el flag -d ni redirección de puertos. Se bloquea el terminal mostrando los logs y tendré que salir con Ctrl+C
> docker run httpd

AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.22. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.22. Set the 'ServerName' directive globally to suppress this message
[Mon Dec 07 18:27:28.561909 2020] [mpm_event:notice] [pid 1:tid 140253864719488] AH00489: Apache/2.4.46 (Unix) configured -- resuming normal operations
[Mon Dec 07 18:27:28.562072 2020] [core:notice] [pid 1:tid 140253864719488] AH00094: Command line: 'httpd -D FOREGROUND'

# Ejecuto un servidor Apache en background y accediendo desde el exterior a través del puerto 8888 de mi máquina.
> docker run -d -p 8888:80 httpd

# Creación de un servidor de base de datos mariadb accediendo desde el exterior a través del puerto 3306 y estableciendo una contraseña de root mediante una variable de
# entorno
> docker run -it -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root mariadb

# Damos el nombre de servidorBD a un contenedor de la imagen mysql:8.0.22
> docker run -d --name servidorBD -p 3306:3306 mysql:8.0.22

# Damos el nombre de servidorWeb a un contenedor de la imagen httpd:latest (Apache)
> docker run -d --name servidorWeb -p 80:80 httpd
```

docker exec [opciones] nombre_contenedor orden [argumentos]

```
Algunas de las opciones más importantes son:
   -it (-i y -t juntos) si vamos a querer tener interactividad con el contenedor ejecutando un shell (/bin/bash normamente). Una vez tenemos el terminal ya podremos trabajar desde dentro del propio sistema.
   -u o --user si quiero ejecutar la orden como si fuera un usuario distinto del de root.
   -w o --workdir si quiero ejecutar la orden desde un directorio concreto.
```

```
# Obtener un terminal en un contenedor que ejecutar un servidor Apache (httpd) y que se llama web
> docker exec -it web /bin/bash

root@5d96ce1f7374:/usr/local/apache2#

# Mostrar el contenido de la carpeta /usr/local/apache2/htdocs del contenedor web. Como no hace falta interactividad no es necesario -it
> docker exec web ls /usr/local/apache2/htdocs
```

```
# Crear directamente un fichero "HOLA MUNDO" en el directorio raíz del servidor apache. Utilizo sh -c para ordenes compuestas o complejas
> docker exec -it web sh -c "echo 'HOLA MUNDO' > /usr/local/apache2/htdocs/index.html"
```

---

la orden **docker cp** que me permite mover ficheros desde mi sistema al contenedor y desde el contenedor a mi sistema. Su sintaxis es muy sencilla y la vamos a ilustrar con dos ejemplos, uno en cada sentido:

```
# Copiar mi fichero prueba.html al fichero /usr/local/apache2/htdocs/index.html de mi contenedor llamado web que es un servidor Apache (httpd)
> docker cp prueba.html web:/usr/local/apache2/htdocs/index.html

#Copiar el fichero index.html que se encuenta en /usr/local/apache2/htdocs/index.html de mi contenedor llamado web un fichero llamado test.html en mi directorio HOME
> docker cp web:/usr/local/apache2/htdocs/index.html $HOME/test.html
```

Obtención de información de los contenedores

varios comandos docker que me van a ayudar a obtener información de un contenedor.

La orden docker ps.
La orden docker inspect.
La orden docker logs.

```
 # Mostrar los contenedores que están en ejecución
> docker ps

# Mostrar todos los contenedores, estén parados o en ejecución (-a o --all)
> docker ps -a

# Añadir la información del tamaño del contenedor a la información por defecto (-s o --size)
> docker ps -a -s

# Mostrar información del último contenedor que se ha creado (-l o --latest). Da igual el estado
> docker ps -l

# Filtar los contenedores de acuerdo a algún criterio usando la opción (-f o --filter)
# Filtrado por nombre
> docker ps --filter name=servidor_web

# Filtrado por puerto. Contenedores que hacen público el puerto 8080
> docker ps --filter publish=8080

# Información detallada del contenedor
# Por nombre. Por ejemplo: Mostrar información detallada del contenedor cuyo nombre es jenkins
> docker inspect jenkins

# Por id. Por ejemplo: Mostrar información detallada del contenedor cuyo id es 5e5adf6815bc
> docker inspect 5e5adf6815bc

# Mostrar la ip del contenedor
>  docker inspect --format 'La ip es {{.NetworkSettings.Networks.bridge.IPAddress}}' jenkins
La ip es 172.17.0.2

# Mostrar las redirecciones de puertos del contenedor
> docker inspect --format 'Las redirecciones de puertos son {{.NetworkSettings.Ports}}' jenkins

Las redirecciones de puertos son map[50000/tcp:[{0.0.0.0 50000}] 8080/tcp:[{0.0.0.0 9090}]]

# Por nombre. Por ejemplo: Mostrar los logs del contenedor cuyo nombre es jenkins
> docker logs jenkins

# Por id. Por ejemplo: Mostrar los logs cuyo id es 5e5adf6815bc
> docker logs 5e5adf6815bc

# Opción -f o --follow . Sigue escuchando la salida que pueden dar los logs del contenedor
> docker logs -f jenkins

# Opción  --tail 5. Muestra las 5 últimas líneas de los logs del contenedor en cuestión
> docker logs --tail 5 jenkins
```

Gestión de imágenes

#***docker pull*** indicando el nombre de la imagen y la versión de la misma (**TAG**). Si no indicamos nada se descarga la última versión (latest). Si en la primera ejecución docker run no la teníamos descargada, la descarga.

```
# Listas imágenes descargas
> docker images

# Borrado de imagenes, no puedes hacerlo si tienes un contenedor usandola.
# Borrado de la imagen mysql:8.0.22
> docker rmi mysql:8.0.22
# Borrado de una imagen usando su IMAGE ID
> docker rmi dd7265748b5d
# Borrado de dos imágenes (o varias)  a la vez. Puedes usar nombre e IMAGE ID
> docker rmi mysql:8.0.22 mysql:5.7
# Borrar todas las imágenes sin usar
> docker image prune -a
# Borrado de la imágenes creadas hace más de una semana  10 días
> docker image prune --filter until="240h"

# Dos formas de obtener información de la imagen mysql:8.0.22
> docker image inspect mysql:8.0.22
> docker inspect mysql:8.0.22

#docker image build para construir una imagen desde un fichero Dockerfile
#docker image history para que se nos muestre por pantalla la evolución de esa imagen.
#docker image save / docker image load (o docker save / docker load) para guardar imágenes en fichero y cargarlas desde fichero.
#docker image tag ( docker tag) para añadir TAGs (versiones) a las distintas imágenes.
```

Los datos en los contenedores

Los archivos, datos y configuraciones que creemos en los contenedores sobreviven a las paradas de los mismos pero, sin embargo, **son destruidos si el contenedor es destruido**. Y esto, como todos entendemos es una situación no deseable ya que puede echar por tierra nuestro trabajo.
Por lo tanto tenemos que tener muy presente varios aspectos a la hora de afrontar esta situación y la gestión del almacenamiento de los contenedores:

- Los datos de un contenedor mueren con él.
- Los datos de los contenedores no se mueven fácilmente ya que están fuertemente acoplados con el host en el que el contenedor está ejecutándose.
- Escribir en los contenedores es más lento que escribir en el host ya que tenemos una capa adicional.

Ante la situación anteriormente descrita Docker nos proporciona VARIAS SOLUCIONES PARA PERSISTIR DATOS de contenedores.

Los VOLÚMENES docker.
Los BIND MOUNT.

Existen otras formas como los tmpfs mounts para Linux y los named pipes para Windows pero menos comunes.

Si elegimos conseguir la persistencia **usando volúmenes** estamos haciendo que los datos de los contenedores que nosotros decidamos se almacenen en una parte del sistema de ficheros que es gestionada por docker y a la que, debido a sus permisos, sólo docker tendrá acceso.

Esa "ZONA RESERVADA" de docker cambia de un sistema operativo a otro y también puede cambiar dependiendo de la forma de instalación.

Este tipo de volúmenes se suele usar en los siguiente casos:

- Para compartir datos entre contenedores. Simplemente tendrán que usar el mismo volumen.
- Para copias de seguridad ya sea para que sean usadas posteriormente por otros contenedores o para mover esos volúmenes a otros hosts.
- Cuando quiero almacenar los datos de mi contenedor no localmente si no en un proveedor cloud.
- En algunas situaciones donde usando Docker Desktop quiero más rendimiento.

Si elegimos conseguir la persistencia de los datos de los contenedores usando **bind mount** lo que estamos haciendo es ***"mapear"*** una parte de mi sistema de ficheros, de la que yo normalmente tengo el control, con una parte del sistema de ficheros del contenedor.

Este mapeado de partes de mi sistema de ficheros con el sistema de ficheros del contenedor me va a permitir:

- **Compartir ficheros** entre el host y los containers.
- Que otras aplicaciones que no sean docker tengan acceso a esos ficheros, ya sean código, ficheros etc...

```
# Creación de un volumen llamado datos (driver local sin opciones)
> docker volume create data

# Creación de un volumen data especificando el driver local
> docker volume create -d local data

# Creación de un volumen llamando web añadiendo varios metadatos
> docker volume create --label servicio=http --label server=apache Web

# Listar los volúmenes creados en el sistema
> docker volume ls

# Borrar un volumen por nombre
> docker volume rm nombre_volumen

# Borrar un volumen por ID
> docker volume rm a5175dc955cfcf7f118f72dd37291592a69915f82a49f62f83666ddc81f67441

# Borrar dos volúmenes de una sola vez
> docker volume rm nombre_volumen1 nombre_volumen2

# Forzar el borrado de un volumen -f o --force
> docker volume rm -f nombre_volumen

# Borrar todos los volúmenes que no tengan contenedores  asociados
> docker volume prune

# Borrar todos los volúmenes que no tengan contenedores  asociados sin pedir confirmación (-f o --force)
> docker volume prune -f

# Borrar todos los volúmenes sin usar que contengan cierto valor de etiqueta (--filter)
> docker volume prune --filter label=valor

# Información detallada de un volumen por nombre
> docker volume inspect nombre_volumen
```

Como puedo usar los volúmenes y los bind mounts en los contenedores. Para cualquiera de los dos casos lo haremos mediante el uso de dos flags de la orden docker run:

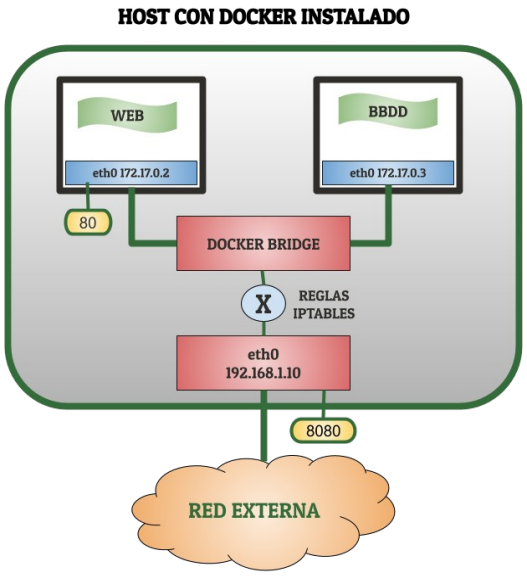
- El flag --volume o -v. Este flag lo utilizaremos para establecer bind mounts
- El flag --mount. Este flag nos servirá para establecer bind mounts y para usar volúmenes previamente definidos (entre otras cosas).

```
# BIND MOUNT (flag -v): La carpeta web del usuario será el directorio raíz del servidor apache. Se crea si no existe
> docker run --name apache -v /home/usuario/web:/usr/local/apache2/htdocs -p 80:80 httpd

# BIND MOUNT (flag --mount): La carpeta web del usuario será el directorio raíz del servidor apache. Se crea si no existe
> docker run --name apache -p 80:80 --mount type=bind,src=/home/usuario/web,dst=/usr/local/apache2/htdocs httpd

# VOLUME (flag --mount). Mapear el volumen previamente creado y que se llama Data en la carpeta raíz del servidor apache
> docker run --name apache -p 80:80 --mount type=volume,src=Data,dst=/usr/local/apache2/htdocs httpd

# VOLUME (flag --mount). Igual que el anterior pero al no poner nombre de volumen se crea uno automáticamente (con un ID como nombre)
> docker run --name apache -p 80:80 --mount type=volume,dst=/usr/local/apache2/htdocs httpd
```

Redes en Docker

El contenedor ni sabe ni es consciente del funcionamiento de la red ni de la plataforma sobre la que funciona. El contenedor es **red y sistema agnóstico**.

Esto se consigue con un sistema de red en el que nosotros podemos "enchufar" distintos dispositivos de red a cada contenedor usando distintos drivers que pueden ser de los siguientes tipos:



- **Bridge:** Es el driver por defecto. Mi equipo actúa como puente del contenedor con el exterior y como medio de comunicación entre los distintos contenedores que tengo en ejecución dentro de una misma red docker.
- **Host:** El contenedor usa directamente la red de mi máquina (el host).
- **Overlay:** Un sistema que conecta distintos servicios docker de máquinas diferentes. Se utiliza para docker Swarm, que es la tecnología de docker para la orquestación de contenedores.
- **MacVlan:** Que nos permite asignar una MAC a nuestro contenedor que parecerá que es un dispositivo físico en nuestra red.
- **None:** Si queremos que el contenedor no tenga conectividad alguna.

Aunque todos estos tipos de drivers tienen su utilizad nos centraremos en el tipo **BRIDGE** que me van a permitir, siempre dentro nuestra máquina local:

- **Aislar los distintos contenedores** que tengo en distintas subredes docker, de tal manera que desde cada una de las subredes solo podremos acceder a los equipos de esa misma subred.
- **Aislar los contenedores del acceso exterior.**
- **Publicar servicios** que tengamos en los contenedores **mediante redirecciones** que docker implementará con las pertinentes reglas de ip tables.

```
# Mostrar todas las redes docker creadas
> docker network ls

# Crear una red. Al no poner nada más coge las opciones por defecto, red bridge local y el mismo docker elige la dirección de red y la máscara
> docker network create red1

# Crear una red (la red2) dándole explícitamente el driver bridge (-d) , una dirección y una máscara de red (--subnet) y una gateway (--gateway)
> docker network create -d bridge --subnet 172.24.0.0./16 --gateway 172.24.0.1 red2

# Eliminar la red red1
> docker network rm red1

# Eliminar una red con un determinado ID
> docker network rm 3cb4100fe2dc
# Eliminar todas la redes que no tengan contenedores asociados
> docker network prune

# Eliminar todas las redes que no tengas contenedores asociados sin preguntar confirmación (-f o --force)
> docker netwokr prune -f

# Eliminar todas las redes que no tengan contenedores asociados y que fueron creadas hace más de 1 hora (--filter)
> docker network prune  --filter until=60m

#NOTA: NO PUEDO BORRAR UNA RED QUE TENGA CONTENEDORES QUE LA ESTÉN USANDO. DEBERÉ PRIMERO BORRAR LOS CONTENEDORES O DESCONECTAR LA RED.
```

```
# Infomración con docker network inspect y docker network ls

# Mostrar el tipo de driver de una red docker (podríamos usar también el ID de la red)
> docker network inspect mi_red  --format 'El driver de {{.Name}} es {{.Driver}}'

# Mostrar solo el ID de las redes (-q o --quiet)
> docker network ls -q

# Mostrar las redes de driver=bridge y nombre=brigde( la red por defecto) (-f o --filter)
> docker network ls -f driver=bridge -f name="bridge"

#CONECTAR DOCKER A REDES
# Arrancar un contenedor de Apache sin especificar red y habilitando la conexión desde el exterior a través del puerto 80. Se conectará por defecto a la red bridge.
> docker run -d --name web -p 80:80 httpd

# Arranchar un contenedor de Apache conectándose a la red red1 que es una red bridge definida por el usuario y habilitando la conexión desde el exterior a través del puerto 8080
> docker run -d --name web2 --network red1 -p 8080:80 httpd

# Arranchar un contenedor de Apache conectándose a la red red1 dándole una ip (que debe pertenecer a esa red)
> docker run -d --name web2 --network red1 --ip 172.18.0.5 -p 8181:80 httpd

# Conectar una nueva red, la red2  al contenedor web2.
> docker network connect red2 web2

# Conectar una red, la red2 al contenedor web y darle una ip (que debe pertenecer a esa red)
> docker network connect --ip 172.28.0.3 red2 web

# Desconectar la red1 del contenedor web2. Debe estar funcionando para poder desconectarse
> docker network disconnect red1 web2
```

Por supuesto la orden ***docker connect*** tienen más opciones que podemos consultar en la referencia y , adicionalmente, hay varios flags de la orden docker run que están relacionados con redes y que pueden resultar de interés:

--dns para establecer unos servidores DNS predeterminados.
--ip6 para establecer la dirección de red ipv6
--hostname o -h para establecer el nombre de host del contenedor. Si no lo establezco será el ID del mismo.

Construyendo nuestras imágenes

La personalización para conseguir nuestras propias imágenes la vamos a conseguir de dos maneras:

Partiendo de un contenedor que tenemos en ejecución y sobre el que hemos realizado modificaciones.

De manera declarativa a través del fichero **Dockerfile** y un proceso de construcción que veremos que puede ser manual o automático.

```
# Creación de una nueva imagen a partir del contenedor con nombre ejemplo (tag=latest)
> docker commit ejemplo usuarioDockerHub/ubuntu20netutils

# Igual que la anterior pero añadiendo versión (tag)
> docker commit ejemplo usuarioDockerHub/ubuntu20netutils:1.0

# Igual que la anterior pero pausando el contenedor durante el commit (--pause/-p) y añadiendo un mensaje describiendo el commit (--message/-m)
> docker commit -m "Versión con Nmap" -p ejemplo usuarioDockerHub/ubuntu20netutils:1.1

# Igual que la anterior pero añadiendo la información del autor (--author/-a)
> docker commit -a "Juan Diego Pérez" -m "Versión con Nmap" -p ejemplo usuarioDockerHub/ubuntu20netutils:1.1

# Guardar la imagen ubuntu20netutils:1.1 al ficher u20v1.1.tar
> docker save usuarioDockerHub/ubuntu20netutils:1.1 > u20v1.1.tar

# Lo mismo que en el apartado anterior sin la redirección y especificando el fichero (--output / -o)
> docker save --output u20v1.1.tar usuarioDockerHub/ubuntu20netutils:1.1

# Carga la imagen con nombre imagen.tar (--input / -i)
> docker load --input imagen.tar
```

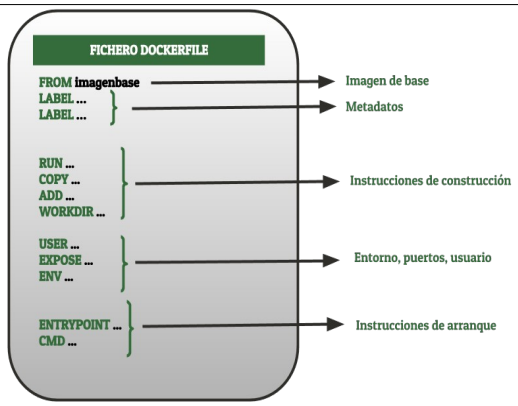
```
# Autentificación en DockerHub
> docker login

# Subir una imagen ubuntu20netutils:1.1 a DockerHub
> docker push usuarioDockerHub/ubuntu20netutils:1.1

# Subir una imagen ubuntu20netutils:1.1 a DockerHub suprimiendo la salida que se muestra sobre la información del proceso de subida (--quiet / -q)
> docker push -q usuarioDockerHub/ubuntu20netutils:1.1

# Subir a DockerHub todas las versiones (tags) de la imagen ubuntu20netutils (--all-tags / -a)
> docker push -a usuarioDockerHub/ubuntu20netutils
```

Dockerfile y Opciones

"Un Archivo Dockerfile es un conjunto de instrucciones que serán ejecutadas de forma secuencial para construir una nueva imagen docker".
Si queremos conectarla y subirla a dockerhub tendremos que tener cuenta y logearnos en ella meciante los comandos

docker login # Nos logearemos contra nuestra cuenta de dockerhub
docker push usuario/imagen:tag # Subiremos la imagen con tag a la cuenta del usuario, si estamos logeados

Las órdenes más comunes son:



FROM: Sirve para especificar la imagen sobre la que voy a construir la mía. Ejemplo: FROM php:7.4-apache

LABEL: Sirve para añadir metadatos a la imagen mediante clave=valor. Ejemplo: LABEL company=iesalixar

COPY: Para copiar ficheros desde mi equipo a la imagen. Esos ficheros deben estar en el mismo contexto (carpeta o repositorio). Su sintaxis es COPY [--chown=<usuario>:<grupo>] src dest. Por ejemplo: COPY --chown=www-data:www-data myapp /var/www/html

ADD: Es similar a COPY pero tiene funcionalidades adicionales como especificar URLs y tratar archivos comprimidos.

RUN: Ejecuta una orden creando una nueva capa. Su sintaxis es RUN orden / RUN ["orden","param1","param2"]. Ejemplo: RUN apt update && apt install -y git. En este caso es muy importante que pongamos la opción -y porque en el proceso de construcción no puede haber interacción con el usuario.

WORKDIR: Establece el directorio de trabajo dentro de la imagen que estoy creando para posteriormente usar las órdenes RUN,COPY,ADD,CMD o ENTRYPOINT. Ejemplo: WORKDIR /usr/local/apache/htdocs

EXPOSE: Nos da información acerca de qué puertos tendrá abiertos el contenedor cuando se cree uno en base a la imagen que estamos creando. Es meramente informativo. Ejemplo: EXPOSE 80

USER: Para especificar (por nombre o UID/GID) el usuario de trabajo para todas las órdenes RUN,CMD Y ENTRYPOINT posteriores. Ejemplos: USER jenkins / USER 1001:10001

ARG: Para definir variables para las cuales los usuarios pueden especificar valores a la hora de hacer el proceso de build mediante el flag --build-arg. Su sintaxis es ARG nombre_variable o ARG nombre_variable=valor_por_defecto. Posteriormente esa variable se puede usar en el resto de la órdenes de la siguiente manera $nombre_variable. Ejemplo: ARG usuario=www-data. NO SE PUEDE USAR EN ENTRYPOINT Y CMD

ENV: Para establecer variables de entorno dentro del contenedor. Puede ser usado posteriormente en las órdenes RUN añadiendo $ delante de el nombre de la variable de entorno. Ejemplo: ENV WEB_DOCUMENT_ROOT=/var/www/html NO SE PUEDE USAR EN ENTRYPOINT Y CMD

ENTRYPOINT: Para establecer el ejecutable que se lanza siempre cuando se crea el contenedor con docker run, salvo que se especifique expresamente algo diferente con el flag --entrypoint. Su síntaxis es la siguiente: ENTRYPOINT <command> / ENTRYPOINT ["executable","param1","param2"]. Ejemplo: ENTRYPOINT ["service","apache2","start"]

CMD: Para establecer el ejecutable por defecto (salvo que se sobreescriba desde la order docker run) o para especificar parámetros para un ENTRYPOINT. Si tengo varios sólo se ejecuta el último. Su sintaxis es CMD param1 param2 / CMD ["param1","param2"] / CMD["command","param1"]. Ejemplo: CMD ["-c" "/etc/nginx.conf"] / ENTRYPOINT ["nginx"].

Comando docker build [opciones] path|url|-

```
# Construcción de una imagen sin nombre ni versión estando el Dockerfile en el mismo directorio donde se ejecuta docker build
> docker build .
```

```
# Construcción de una imagen especificando nombre y versión estando el Dockerfile en el mismo directorio donde se ejecuta docker build (--tag/-t)
> docker build -t  usuario/nombre_imagen:1.0 .

# Construcción de una imagen especificando un repositorio en GitHub donde se encuentra el Dockerfile. Ese repositorio es el contexto de construcción
> docker build -t usuarioDockerHub/nombre_imagen:1.1 https://github.com/...../nombre_repo.git#nombre_rama_git

# Construcción de una imagen usando una variable de entorno estando el Dockerfile en el mismo directorio donde se ejecuta docker build (--build-arg)
> docker build --build-arg user=usuario -t  usuarioDockerHub/nombre_imagen:1.0 .

# Construcción de una imagen sin usar las capas cacheadas por haber realizado anteriormente imágenes con capas similares y estando el Dockerfile en el mismo directorio donde se ejecuta docker build (--no-cache)
> docker build --no-cache -t  usuarioDockerHub/nombre_imagen:1.0 .

# Construcción de una imagen especificando nombre,versión y especificando la ruta al fichero Dockerfile mediante el flag --file/-f
> docker build -t  usuario/nombre_imagen:1.0 -f /home/usuario/DockerProject/Dockerfile
```

Orquestadores de contenerdores

Hasta ahora hemos estado hablando de contenedores en solitario pero la realidad es que las aplicaciones actuales están formadas de varias aplicaciones o servicios.
Los orquestadores de contenedores son herramientas para desplegar grupos de contenedores que forman parte de una misma aplicación, entorno, o que necesitan un orden en su ejecución.
Dentro de la propia Docker disponemos de **Docker-Compose**, pero si quisiéramos poder trabajar con docker y otras tecnologías de contenedores podríamos usar **Kubernetes (K8s)**.

La forma de trabajo suele ser la siguiente:
1.- Describir de manera declarativa todo los contenedores que conforman mi aplicación en un archivo usualmente con formato YAML.
2.- Lanzar la puesta en marcha de todos los recursos declarados en el archivo anterior

Puede ver dos ejemplos de uso de estas tecnología en las siguientes columnas.

Ejemplo de docker-composer.yml de bitnami/prestashop

```
version: '2'
services:
  mariadb:
    image: docker.io/bitnami/mariadb:10.4
    environment:
            # ALLOW_EMPTY_PASSWORD is recommended only for development.
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=bn_prestashop
      - MARIADB_DATABASE=bitnami_prestashop
    volumes:
      - 'mariadb_data:/bitnami/mariadb'
  prestashop:
    image: docker.io/bitnami/prestashop:1.7
    ports:
      - '80:8080'
      - '443:8443'
    environment:
      - PRESTASHOP_HOST=localhost
      - PRESTASHOP_DATABASE_HOST=mariadb
      - PRESTASHOP_DATABASE_PORT_NUMBER=3306
      - PRESTASHOP_DATABASE_USER=bn_prestashop
      - PRESTASHOP_DATABASE_NAME=bitnami_prestashop
            # ALLOW_EMPTY_PASSWORD is recommended only for development.
      - ALLOW_EMPTY_PASSWORD=yes
    volumes:
      - 'prestashop_data:/bitnami/prestashop'
    depends_on:
      - mariadb
volumes:
  mariadb_data:
    driver: local
  prestashop_data:
    driver: local
```

Al ejecutar ***docker-compose up*** se levanta toda la aplicación, es decir, todos los contenedores que la conforman.

Ejemplo de configuración de archivo deployment de owncloud para kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nextcloud-deployment
  labels:
    app: nextcloud
    type: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nextcloud
      type: frontend
  template:
```

```
metadata:
  labels:
    app: nextcloud
    type: frontend
spec:
  containers:
    - name: contenedor-nextcloud
      image: nextcloud
      ports:
        - containerPort: 80
          name: http-port
        - containerPort: 443
          name: https-port
      env:
        - name: MYSQL_HOST
          value: mariadb-service
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: bd-datos
              key: bd_user
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: bd-passwords
              key: bd_password
        - name: MYSQL_DATABASE
          valueFrom:
            configMapKeyRef:
              name: bd-datos
              key: bd_dbname
```

Con el comando de gestión del cluster para kubernetes ***kubectl apply archivo.yml*** gestionamos su puesta en marcha.

Glosario de términos

» Imagen

De una manera simple y muy poco técnica una imagen es una plantilla (ya sea de una aplicación de un sistema) que podremos utilizar como base para la ejecución posterior de nuestras aplicaciones (contenedores). Si queremos una descripción más técnica y detallada diremos que es un archivos comprimido en el que, partiendo de un sistema base, se han ido añadiendo capas cada uno de las cuáles contiene elementos necesarios para poder ejecutar una aplicación o sistema. No tiene estado y no cambia salvo que generemos una nueva versión o una imagen derivada de la misma.

» Contenedor

Es una imagen que junto a unas instrucciones y variables de entorno determinadas se ejecuta. Tiene estado y podemos modificarlo. Estos cambios no afectan a la imagen o "plantilla" que ha servido de base.

» Repositorio

Almacén, normalmente en la nube desde el cuál podemos descargar distintas versiones de una misma imagen para poder empezar a construir nuestras aplicaciones basadas en contenedores.

» Docker

Plataforma, mayormente opensource, para el desarrollo, empaquetado y distribución de aplicaciones de la empresa Docker Inc (anteriomente Dot Cloud Inc). Es un término que se suele utilizar indistintamente al del Docker Engine.

» Docker Engine

Aplicación cliente-servidor que consta de tres componentes: un servicio dockerd para la ejecución de los contenedores, un API para que otras aplicaciones puedan comunicarse con ese servicio y una aplicación de línea de comandos docker cli que sirve para gestionar los distintos elementos (contenedores, imágenes, redes, volúmenes etc..)

» Docker Hub

Registro de repositorios de imágenes de la empresa Docker Inc. Accesible a través de la URL https://hub.docker.com/

Guía de comandos básicos

Administrar imágenes (docker image)

Descargar imagen
docker image pull [nombre_imagen:version]

Listar imágenes
docker image ls

Ver capas de la imagen
docker image history [nombre_imagen|id_imagen]

Eliminar imagen
docker image rm [nombre_imagen|id_imagen]

Limpiar imágenes que no tienen contenedores asociados
docker image prune

Administrar contenedores (docker container) comando por defecto , es lo mismo que poner solo la orden docker

Lista todos los contenedores iniciados y apagados
docker container ls -a

Ejecuta un contenedor en background e imprime el id.
docker container run -d [nombre_imagen|id_imagen]

Ejecuta un contenedor y accedemos a la terminal del contenedor.
docker container run -it [nombre_imagen|id_imagen] bash

Muestra el log del contenedor.
docker container logs [nombre_contenedor|id_contenedor]

Detiene un contenedor en ejecución
docker container stop [nombre_contenedor|id_contenedor]

Inicia un contenedor detenido
docker container start [nombre_contenedor|id_contenedor]

Elimina un contenedor detenido (Use -f para eliminar un contenedor en ejecución)
docker container rm [nombre_contenedor|id_contenedor]

Muestra el detalle completo de un contenedor
docker container inspect [nombre_contenedor|id_contenedor]

Elimina todos los contenedores detenidos (Use -f para también eliminar los contenedores en ejecución)
docker container rm $(docker container ls -qa)

Ingresar a la terminal de un contenedor en ejecución
docker container exec -it [nombre_contenedor|id_contenedor] bash

Administrar redes (docker network)

Listar redes
docker network ls

Revisar el detalle de una red
docker network inspect [nombre_red|id_red]

Administrar volumenes datos (docker volume)

Listar volumen
docker volume ls

Revisar el detalle de un volumen
docker volume inspect [nombre_volumen|id_volumen]

Docker compose (orquestador configurando docker-compose.yml)

Ejecutar servicios en background
docker-compose up -d

Compila imagenes con los cambios en el docker-compose.yaml
docker-compose build

Detiene los servicios
docker-compose stop