Desarrollo avanzando de código

Docker y DockerHub

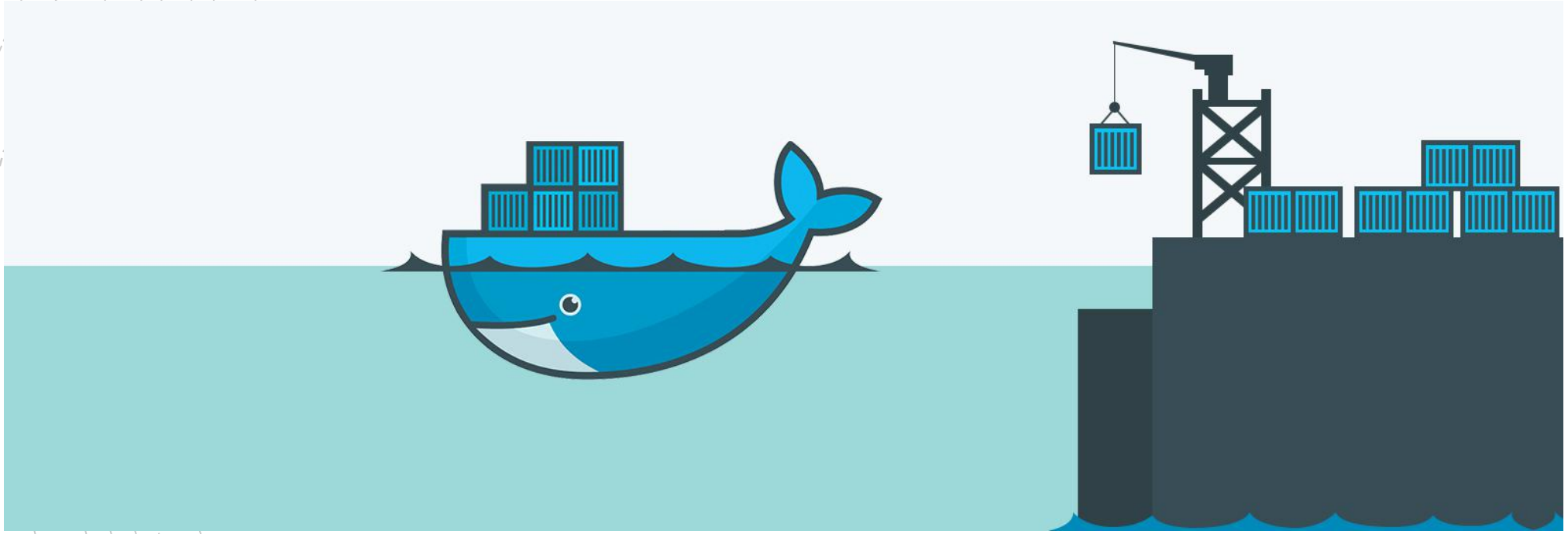
Manejo de contenedores

José Luis González Sánchez

https://github.com/joseluisgs

¿Qué vamos a aprender?

- Aprenderemos a manejar imagenes y contenedores y cómo aplicarlos para mejorar en el desarrollo de software



¿Qué es Docker?

- Docker es una plataforma para desarrollar, lanzar y ejecutar aplicaciones. Permite separar las aplicaciones desarrolladas de la infraestructura donde se desarrollan y trabajar así más comoda y rapidamente.
- Docker permite empaquetar y lanzar una aplicación en un entorno totalmente aislado llamado **contenedor**. Estos contenedores se ejecutan directamente sobre el kernel de la máquina por lo que son mucho más ligeros que las máquinas virtuales.
- Con Docker podemos ejecutar mucho más contenedores para el mismo equipo que si éstos fueran maquina virtuales. De esta manera, podemos probar rapidamente uestra aplicación web, por ejemplo, en múltiples entornos distintos al de donde nos encontramos desarrollando. Realmente es mucho más rápido que hacerlo en una máquina virtual, puesto que reduce el tiempo de carga y el espacio requerido por cada uno de estos contenedores o máquinas.

Contenedores

- Los contenedores se distinguen de las máquinas virtuales en que las máquinas virtuales emulan un ordenador físico en el que se instala un sistema operativo completo, mientras que los contenedores usan el kernel del sistema operativo anfitrión pero contienen las capas superiores (sistema de ficheros, utilidades, aplicaciones).
- Al ahorrarse la emulación del ordenador y el sistema operativo de la máquina virtual, los contenedores son más pequeños y rápidos que las máquinas virtuales. Pero al incluir el resto de capas de software, se consigue el aislamiento e independencia entre contenedores que se busca con las máquinas virtuales.

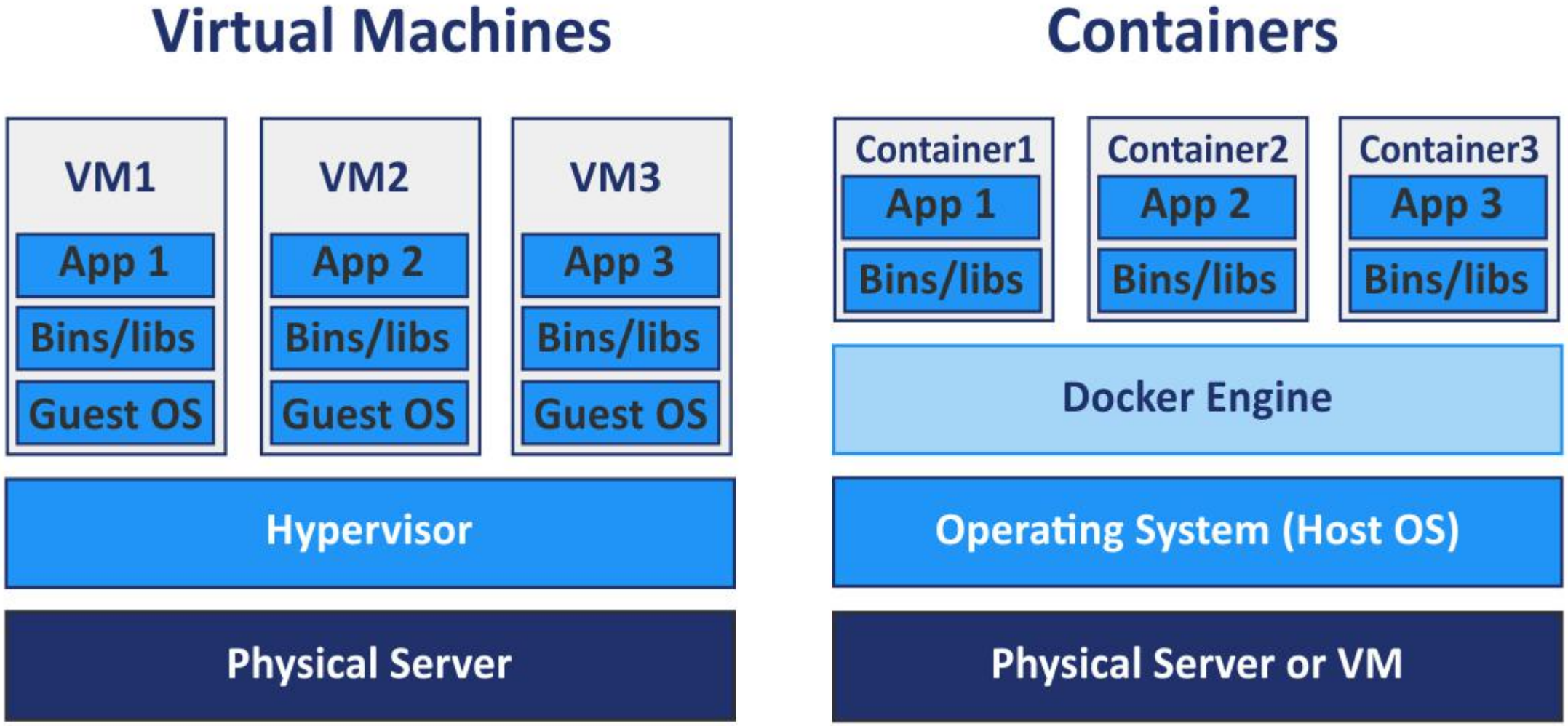
Contenedores

- Los contenedores se distinguen de las máquinas virtuales en que las máquinas virtuales emulan un ordenador físico en el que se instala un sistema operativo completo, mientras que los contenedores usan el kernel del sistema operativo anfitrión pero contienen las capas superiores (sistema de ficheros, utilidades, aplicaciones).
- Al ahorrarse la emulación del ordenador y el sistema operativo de la máquina virtual, los contenedores son más pequeños y rápidos que las máquinas virtuales. Pero al incluir el resto de capas de software, se consigue el aislamiento e independencia entre contenedores que se busca con las máquinas virtuales.

Contenedores

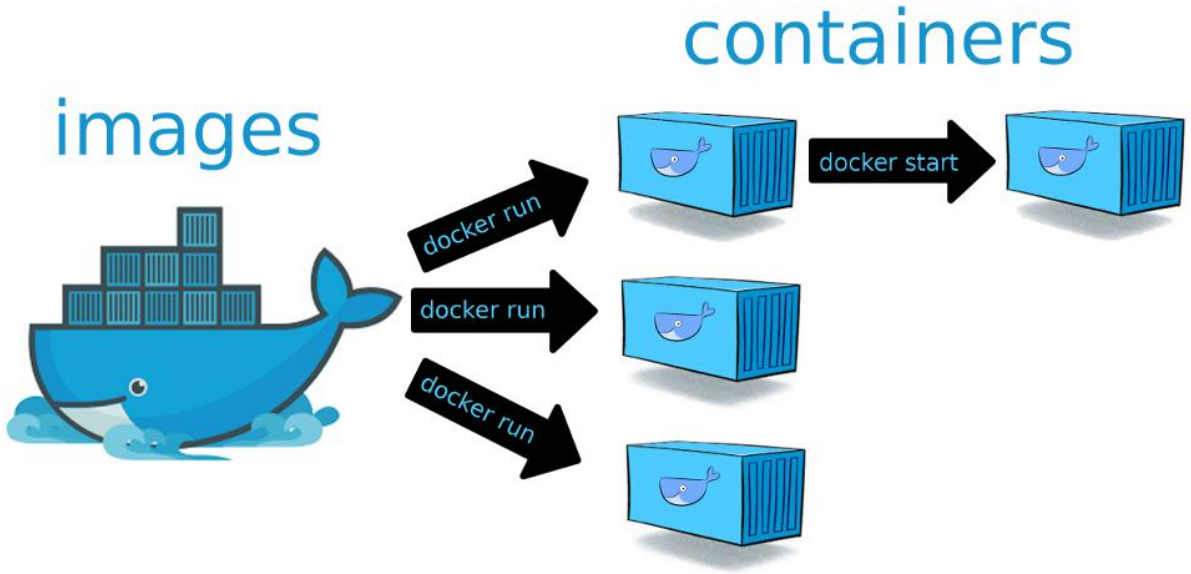
- Docker no es virtualizado, no hay un hipervisor. Los procesos que corren dentro de un contenedor de docker se ejecutan con el mismo kernel que la máquina anfitrión. Linux lo que hace es aislar esos procesos del resto de procesos del sistema, ya sean los propios de la máquina anfitrión o procesos de otros contenedores.
- Además, es capaz de controlar los recursos que se le asignan a esos contenedores (cpu, memoria, etc).
- Internamente, el contenedor no sabe que lo es y a todos los efectos es una distribución GNU/Linux independiente, pero sin la penalización de rendimiento que tienen los sistemas virtualizados.
- Así que, cuando ejecutamos un contenedor, estamos ejecutando un servicio dentro de una distribución construida a partir de una "receta". Esa receta permite que el sistema que se ejecuta sea siempre el mismo, independientemente de si estamos usando Docker en Ubuntu, Fedora o, incluso, sistemas privativos compatibles con Docker. De esa manera podemos garantizar que estamos desarrollando o desplegando nuestra aplicación, siempre con la misma versión de todas las dependencias.

Contenedores



Imagenes

- Una Imagen es una plantilla de solo lectura que contiene las instrucciones para crear un contenedor Docker. Pueden estar basadas en otras imagenes, lo cual es habitual. Para ello usaremos distintos ficheros como el dockfile.
- Por ejemplo una imagen podría contener un sistema operativo Ubuntu con un servidor Apache y tu aplicación web instalada.

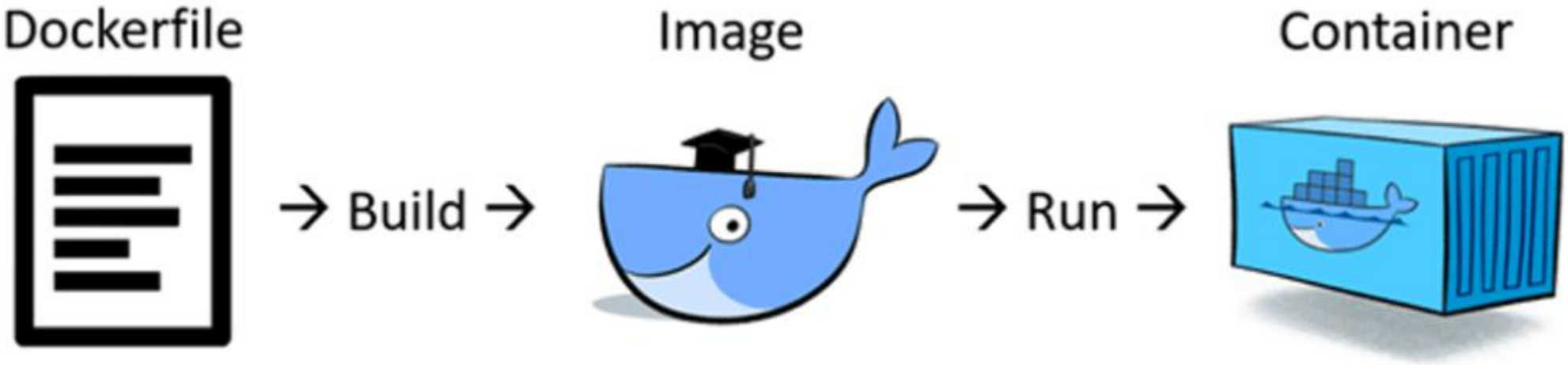


Imágenes y contenedores

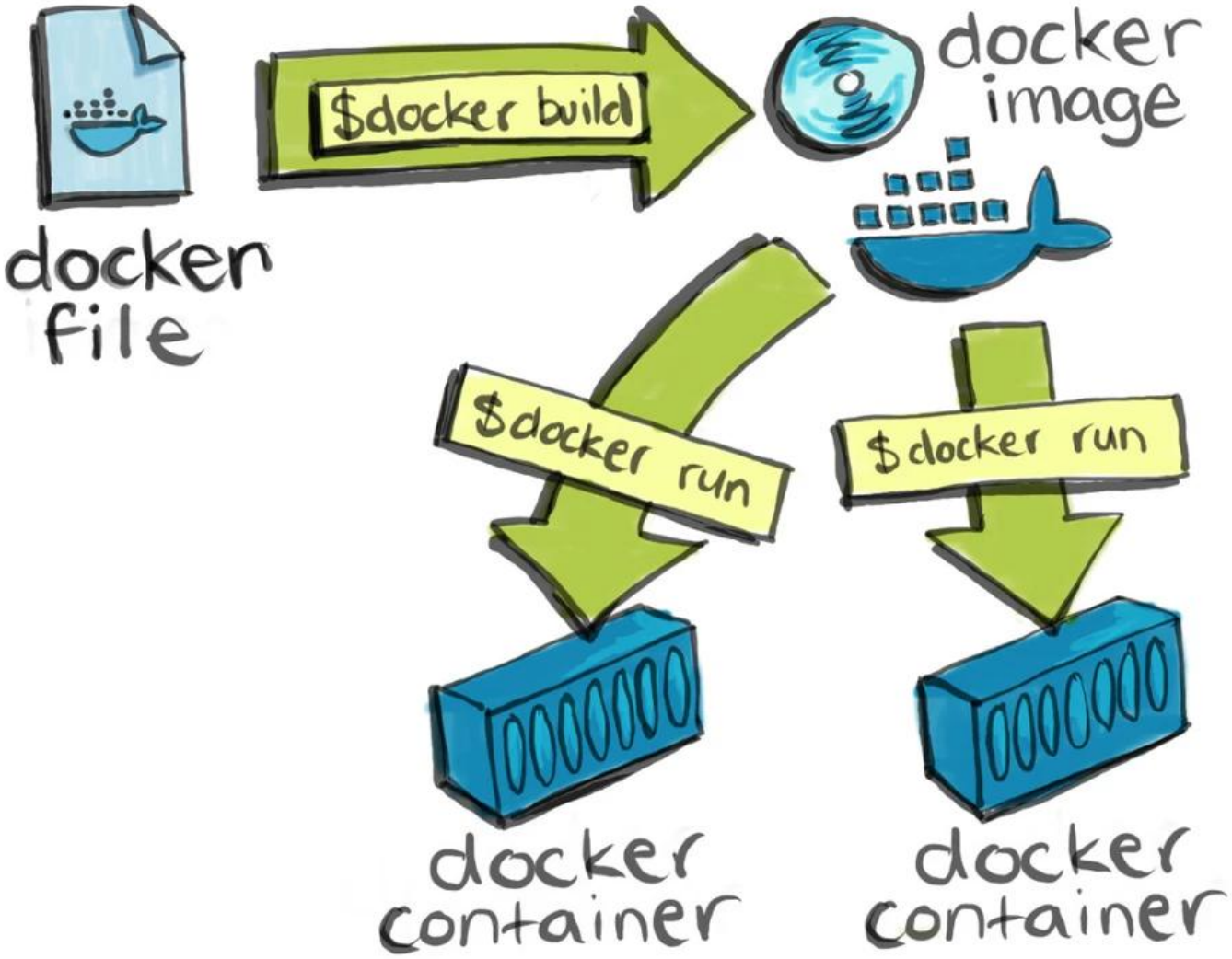
- Por lo tanto, un contenedor es s una instancia ejecutable de una imagen.
- Esta instancia puede ser creada, iniciada, detenida, movida o eliminada a través del cliente de Docker o de la API.
- Las instancias se pueden conectar a una o más redes, sistemas de almacenamiento, o incluso se puede crear una imagen a partir del estado de un contenedor.
- Se puede controlar cómo de aislado está el contenedor del sistema anfitrión y del resto de contenedores.
- El contenedor está definido tanto por la imagen de la que procede como de las opciones de configuración que permita.
- Por ejemplo, la imagen oficial de MariaDb permite configurar a través de opciones la contraseña del administrador, de la primera base de datos que se cree, del usuario que la maneja, etc.

Dockerfile

- Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen que, posteriormente, se convertirá en los contenedores que ejecutamos en el sistema
- Es como la receta para definir la imagen, que posteriormente lanzaremos como contenedor

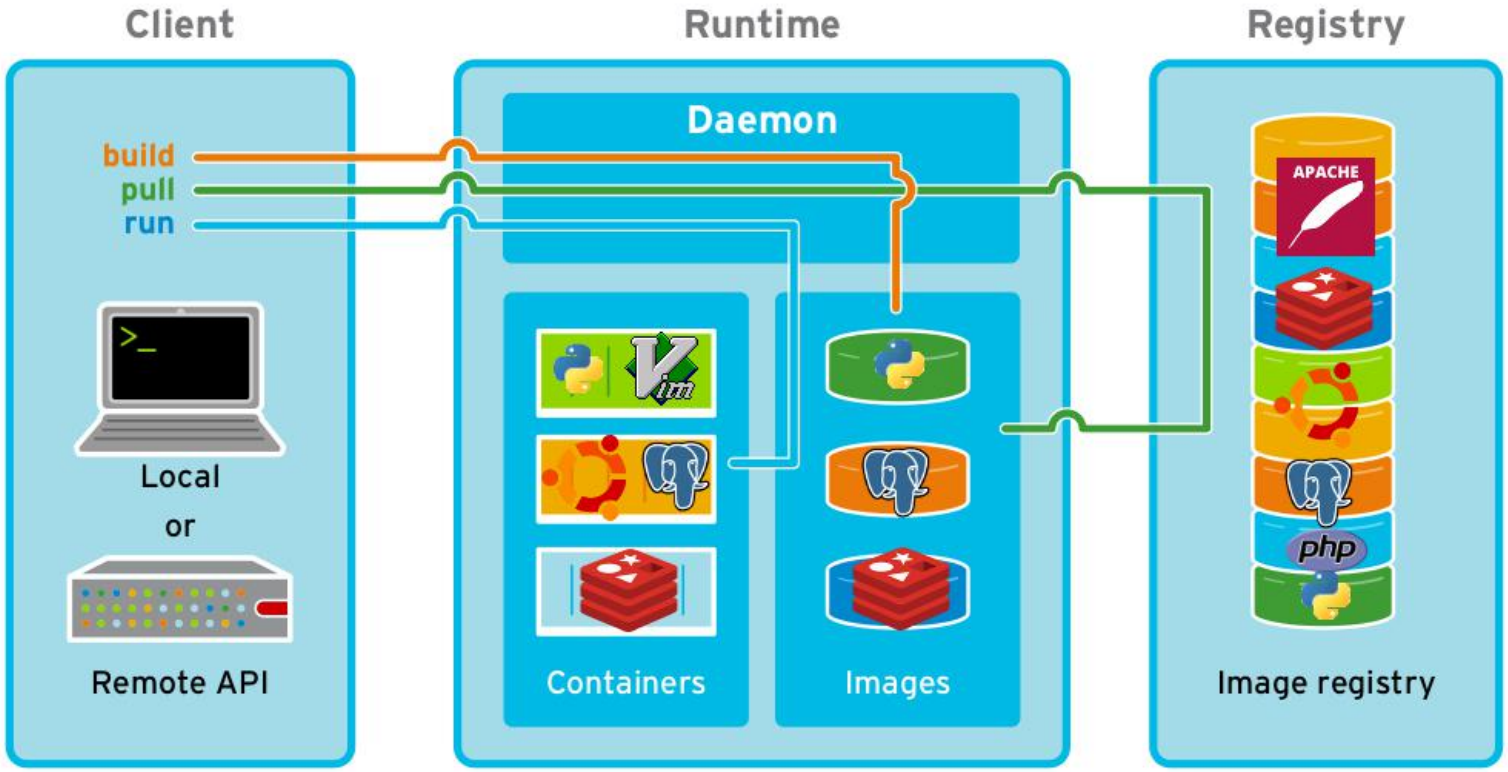


Resumiendo


$docker build
docker image
docker file
$docker run
$docker run
docker container
docker container

DockerHub

- DockerHub es el registro de imagenes donde podemos subir nuestras imágenes o encontrar imagene sya hechas con la que trabajar sobre ellas. Podemos usarlo de manera muy similar a la que usamos GitHub y nos podemos dar de alta en dicho servicio para almacenar nuestras imágenes



Instalar Docker

- Debido a que, dependiendo de la distribución, la forma de instalarlo difiere, es mejor consultar la documentación oficial para saber como instalar Docker en tu máquina.
- https://docs.docker.com/get-docker/
- Desistalar versiones anteriores

  sudo apt-get remove docker docker-engine docker.io containerd runc
- Configurando el repositorio

  sudo apt-get update

  sudo apt-get install \

  apt-transport-https \

  ca-certificates \

  curl \

  gnupg-agent \

  software-properties-common

Instalar Docker

- Añadimos la clave GPG

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

- Añadimos el repositorio

  sudo add-apt-repository \

  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \

  $(lsb_release -cs) \

  stable"

- Instalamos

  sudo apt-get update

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

Instalar Docker

- Añadimos el usuario, de esta manera nos ahorramos hacer sudo siempre

  sudo usermod -aG docker $USER

- Instalamos Docker Compose

  sudo apt install docker-compose

- Probamos el primer contenedor

  sudo docker run hello-world

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Comandos Docker

- Para ver los comandos disponibles (en general y en particular):

  sudo docker

  sudo docker image

  sudo docker network

- Para ver las opciones de cada comando:

  sudo docker cp --help

- Los comandos tienen a su vez opciones. Los nombres de las opciones van precedidas de los caracteres -- y entre la opción y su valor se puede escribir un espacio o el carácter =.

  sudo docker COMANDO --OPCIÓN=VALOR

  sudo docker COMANDO --OPCIÓN VALOR

- Si el valor de una opción contiene espacios, escriba el valor entre comillas

  sudo docker COMANDO --OPCIÓN="VALOR CON ESPACIOS"

  sudo docker COMANDO --OPCIÓN "VALOR CON ESPACIOS"

Comandos con imágenes

- Para gestionar las imágenes, se utiliza el comando:

  sudo docker image OPCIONES

- Para descargar una imagen:

  sudo docker image pull REPOSITORIO

  sudo docker pull REPOSITORIO

- Para ver las imágenes ya descargadas:

  sudo docker image ls

- Para borrar una imagen (se deben borrar previamente los contenedores basados en esa imagen):

  sudo docker image rm IMAGEN

Comandos con contenedores

- Para crear un contenedor (y ponerlo en marcha):

  sudo docker run --name=CONTENEDOR REPOSITORIO

  El problema de este comando es que dejamos de tener acceso a la shell y sólo se puede parar el proceso desde otro terminal.

  Lo habitual es poner en marcha el contenedor en modo separado (detached), es decir, en segundo plano, y así podemos seguir utilizando la shell:

  sudo docker run -d --name=CONTENEDOR REPOSITORIO

- Si queremos ver la secuencia de arranque del contenedor, podemos poner en marcha el contenedor en modo pseudo-tty, que trabaja en primer plano, pero del que podemos salir con Ctrl+C.

  sudo docker run -t --name=CONTENEDOR REPOSITORIO

Comandos con contenedores

- Al crear el contenedor se pueden añadir diversas opciones:

  Para incluir el contenedor en una red privada virtual (y que se pueda comunicar con el resto de contenedores incluidos en esa red):

  sudo docker run --name=CONTENEDOR --net=RED REPOSITORIO

  Para que el contenedor atienda a un puerto determinado, aunque internamente atienda un puerto distinto:

  sudo docker run --name=CONTENEDOR -p PUERTO_EXTERNO:PUERTO_INTERNO REPOSITORIO

  Para establecer variables de configuración del contenedor:

  sudo docker run --name=CONTENEDOR -e VARIABLE=VALOR REPOSITORIO

  Las variables de configuración se pueden consultar en el repositorio del que obtenemos la imagen.

Comandos con contenedores

- Para ver los contenedores en funcionamiento:

  sudo docker ps

- Para ver los contenedores en funcionamiento o detenidos:

  sudo docker ps -a

- Para detener un contenedor:

  sudo docker stop CONTENEDOR

- Para detener todos los contenedores:

  sudo docker stop $sudo docker ps -aq)

- Para borrar un contenedor:

  sudo docker rm CONTENEDOR

- Para poner en marcha un contenedor detenido:

  sudo docker start CONTENEDOR

Comandos con contenedores

- Para entrar en la shell de un contenedor:

  sudo docker exec -it CONTENEDOR /bin/bash

- Para entrar en la shell del contenedor como root:

  sudo docker exec -u 0 -it CONTENEDOR /bin/bash

- Para salir de la shell del contenedor:

  exit

- Para copiar (o mover) archivos entre el contenedor y el sistema anfitrión o viceversa:

  sudo docker cp CONTENEDOR:ORIGEN DESTINO

  sudo docker cp ORIGEN CONTENEDOR:DESTINO

Comandos de red

- Para gestionar las redes, se utiliza el comando:

  sudo docker network OPCIONES

- Para crear una red:

  sudo docker network create RED

- Para ver las redes existentes:

  sudo docker network ls

- Para ver información detallada de una red (entre ella, los contenedores incluidos y sus IP privadas):

  sudo docker network inspect RED

- Para borrar una red:

  sudo docker network rm RED

Comandos de sistema

- Para ver el espacio ocupado por las imágenes, los contenedores y los volúmenes:

  sudo docker system df

- Para eliminar los elementos que no están en marcha:

  Contenedores:

  sudo docker container prune

  Imágenes:

  sudo docker image prune

  Volúmenes:

  sudo docker volume prune

  Redes:

  sudo docker network prune

  todo:

  sudo docker system prune

Repaso de comandos con Hello-Run

- Comprueba que inicialmente no hay ningún contenedor creado (la opción -a hace que se muestren también los contenedores detenidos, sin ella se muestran sólo los contenedor que estén en marcha):

  sudo docker ps -a

  o también

  sudo docker container ls -a

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
```

- Comprueba que inicialmente tampoco disponemos de ninguna imagen:

  sudo docker image ls

```
docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

Repaso de comandos con Hello-Run

- Docker crea los contenedores a partir de imágenes locales (ya descargadas), pero si al crear el contenedor no se dispone de la imagen local, Docker descarga la imagen de su repositorio.
- La orden más simple para crear un contenedor es:

  sudo docker run IMAGEN
- Crea un contenedor con la aplicación de ejemplo hello-world. La imagen de este contenedor se llama hello-world:

  sudo docker run hello-world
- Comprueba que inicialmente tampoco disponemos de ninguna imagen:

  sudo docker image ls

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              bf756fb1ae65        9 months ago        13.3kB
```

Repaso de comandos con Hello-Run

- Si listamos ahora los contenedores existentes ...

  sudo docker ps -a

- ... se mostrará información del contenedor creado:

  sudo docker run hello-world

```
$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                          PORTS               NAMES
10a829dbd778        hello-world         "/hello"            About a minute ago   Exited (0) About a minute ago                       strange_cerf
```

- Cada contenedor tiene un identificador (ID) y un nombre distinto. Docker "bautiza" los contenedores con un nombre peculiar, compuesto de un adjetivo y un apellido.

Repaso de comandos con Hello-Run

- Podemos crear tantos contenedores como queramos a partir de una imagen. Una vez la imagen está disponible localmente, Docker no necesita descargarla y el proceso de creación del contenedor es inmediato (aunque en el caso de hello-world la descarga es rápida, con imágenes más grandes la descarga inicial puede tardar un rato)
- Normalmente se aconseja usar siempre la opción -d, que arranca el contenedor en segundo plano (detached) y permite seguir teniendo acceso a la shell (aunque con hello-world no es estrictamente necesario porque el contenedor hello-world se detiene automáticamente tras mostrar el mensaje).
- Al crear el contenedor hello-world con la opción -d no se muestra el mensaje, simplemente muestra el identificador completo del contenedor.

```
$ sudo docker run -d hello-world
f3059d7adaf9b2bb28b749bba44785fa331f5334b6cf87c5c9b7c7ccc13e8d29

$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                          PORTS               NAMES
f3059d7adaf9        hello-world         "/hello"            13 seconds ago      Exited (0) 12 seconds ago                           affectionate_varahamihira
10a829dbd778        hello-world         "/hello"            4 minutes ago       Exited (0) 4 minutes ago                            strange_cerf
```

Repaso de comandos con Hello-Run

- Los contenedores se pueden destruir mediante el comando rm, haciendo referencia a ellos mediante su nombre o su id. No es necesario indicar el id completo, basta con escribir los primeros caracteres (de manera que no haya ambigüedades).
- Borra los dos contenedores existentes:

  sudo docker rm 1ae

  sudo docker rm affectionate_varahamihira

- Comprueba que ya no quedan contenedores

Repaso de comandos con Hello-Run

- Podemos dar nombre a los contenedores al crearlos:

  sudo docker run -d --name=hola-1 hello-world

- Al haber utilizado la opción -d únicamente se mostrará el ID completo del contenedor
- Si listamos los contenedores existentes ...

  sudo docker ps -a

- ... se mostrará el contenedor con el nombre que hemos indicado y podremos trabajar con él
- Si intentamos crear un segundo contenedor con un nombre ya utilizado ...

  sudo docker run -d --name=hola-1 hello-world

- Docker nos avisará de que no es posible
- Podemos ver las imagenes con:

  sudo docker images

  Y borrarlas con:

  sudo images rm nombre_imagen

- Prueba a borrar la imagen hello_world y compruébalo

Jugando con contenedores

- Crea un contenedor que contenga un servidor Apache a partir de la imagen bitnami/apache
- La opción -P hace que Docker asigne de forma aleatoria un puerto de la máquina virtual al puerto asignado a Apache en el contenedor. La imagen bitnami/apache asigna a Apache el puerto 8080 del contenedor para conexiones http y el puerto 8443 para conexiones https.

  ```
  sudo docker run -d -P --name=apache-1 bitnami/apache
  ```

- Consulta el puerto del host utilizado por el contenedor ...

  ```
  sudo docker ps -a
  ```

- ... se mostrará el contenedor con el nombre que hemos indicado:

- Abre en el navegador la página inicial del contenedor y compruebe que se muestra una página que dice "It works!".

Jugando con contenedores

- En este apartado vamos a modificar la página web inicial de Apache del contenedor Docker.
- Debemos tener en cuenta que modificar el contenido de un contenedor tal y como vamos a hacer en este apartado sólo es aconsejable en un entorno de desarrollo, pero no es aconsejable en un entorno de producción porque va en contra de la "filosofía" de Docker.
- Los contenedores de Docker están pensados como objetos de "usar y tirar", es decir, para ser creados, destruidos y creados de nuevo tantas veces como sea necesario y en la cantidad que sea necesaria.
- En el apartado siguiente realizaremos la misma tarea de una forma más conveniente, modificando no el contenedor sino la imagen a partir de la cual se crean los contenedores.

Jugando con contenedores

- Cree un segundo contenedor que contenga un servidor Apache a partir de la imagenhttpd
- sudo docker run -d -P --name=apache-daw httpd
- Consulte el puerto del host utilizado por el contenedor ...

  sudo docker ps -a

  ... se mostrarán los dos contenedores creados
- Crea la nueva página index.html ...

  code index.html
- Entre en la shell del contenedor para averiguar la ubicación de la página inicial:

  sudo docker exec -it apache-daw /bin/bash

  es: /usr/local/apache2/htdocs
- Salimos de la imagen

  exit
- Copiamos nuestra página web en esa ruta

  docker cp index.html apache-daw:/usr/local/apache2/htdocs
- Recargamos el ordenador... Ya tienes tu paǵina ejecutada con Docker :)

Jugando con contenedores

- Vamos a intentar ahorrarnos el copiar los ficheros siempre. Probamos lo siguiente desde donde tenemos la página web

  docker run -dit --name miapache -p 5555:80 -v "$PWD":/usr/local/apache2/htdocs httpd

  Una vez lanzado y puesto en marcha podemos ir a la dirección http://localhost:5555 con nuestro navegador y visitar nuestro sitio web funcionando desde el contenedor.

- ¿Qué ha pasado?

  El contenedor tendrá asociado el nombre miapache que podrá ser utilizando a partir de su creación para realizar cualquier operación sobre él

  Se ha mapeado el puerto 80 del contenedor sobre el puerto 5555 de la máquina real. De esa manera, conectándonos al puerto 5555 de nuestro equipo podremos ver que ocurre en el 80 del contenedor, justamente donde estará escuchando el Apache del mismo

  Hemos mapeado la ruta actual de nuestro equipo con la ruta /usr/local/apache2/htdocs del contenedor, que es donde Apache va a ir a buscar la web que aloje. Suponemos que el directorio actual de mi máquina tengo el sitio web que quiero testear

  Indicamos que queremos crear y lanzar un contenedor utilizando la imagen del Hub de Docker, concretamente con la versión latest

  También hemos indicado con la opción -d que queremos que la máquina se lance en segundo plano

  con it dejamos que tengamos acceso a la consola por defecto.

  Puedes entrar a la maquina con docker exec -it miapache /bin/bash y ver en la carpeta htdocks que esta alli todo.

  Prueba a crear una página nueva a ver que pasa

Dockerfile

- Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen que, posteriormente, se convertirá en el patrón de nuestros contenedores. Para ello le añadimos:
- FROM: indica la imagen base sobre la que se construirá la aplicación dentro del contenedor.

  FROM <imagen>

  FROM <imagen>:<tag>

  Por ejemplo la imagen puede ser un sistema operativo como Ubuntu, Centos, etc. O una imagen ya existente en la cual con base a esta queramos construir nuestra propia imagen.

- RUN: nos permite ejecutar comandos en el contenedor, por ejemplo, instalar paquetes o librerías (apt-get, yum install, etc.). Además, tenemos dos formas de colocarlo:

  Opción 1 -> RUN <comando>

  Esta instrucción ejecuta comandos Shell en el contenedor.

  Opción 2 -> [“ejecutable”,” parametro1”,” parametro2”]

  Esta otra instrucción bastante útil, que permite ejecutar comandos en imágenes que no tengan /bin/sh.

- ENV -> establece variables de entorno para nuestro contenedor, en este caso la variable de entorno es DEBIAN_FRONTEND noninteractive, el cual nos permite instalar un montón de archivos .deb sin tener que interactuar con ellos.

  ENV <key><valor>

Dockerfile

- Ahora vamos a revisar otro dockerfile, en el cual crearemos una imagen con base Ubuntu 16:04 y haremos las siguientes sub-tareas:
  - Actualizar sus paquetes
  - Instalar el paquete Nginx
  - Añadir un archivo en una ruta especifica dentro del contenedor
  - Exponer un puerto

```
# Descarga la imagen de Ubuntu 16.04
FROM ubuntu:16.04

# Actualiza la imagen base de Ubuntu 16.04
RUN apt-get update

# Ejecuta el commando apt-get install y elimina determinados archivos y temporales
RUN apt-get install -y nginx \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Indica los puertos TCP/IP los cuales se pueden accede a los servicios del contenedor
EXPOSE 80

# Establece el commando del proceso de inicio del contenedor

CMD [“nginx”]
```

Dockerfile

- ADD -> esta instrucción copia archivos a un destino especifico dentro del contenedor, normalmente nos sirve para dejar ubicados ciertos archivos que queremos mover entre directorios, podemos usar como origen una URL o un .tar y descomprimirlo

  ADD <fuente> <destino>

  ADD ./script.sh /var/tmp/script.sh

- COPY -> esta instrucción copia archivos o directorios a un destino especifico dentro del contenedor.

  COPY <fuente> <destino>

  COPY ./script.sh /var/tmp/script.sh

- MAINTAINER: Este nos permite indicar el nombre del autor del dockerfile.

  MAINTANER <nombre> <" correo">

  MAINTAINER JL Gonzalez "jlgs@cifpvirgendegracia.com"

- CMD -> esta instrucción nos provee valores por defecto a nuestro contenedor, es decir, mediante esta podemos definir una serie de comandos que solo se ejecutaran una vez que el contenedor se ha inicializado, pueden ser comandos Shell con parámetros establecidos.

  CMD ["ejecutable", "parámetro1", "parámetro2"], este es el formato de ejecución.

  CMD ["parámetro1", "parámetro2"], parámetro por defecto para punto de entrada.

  CMD comando parámetro1 parámetro2, modo shell

Dockerfile

- ENTRYPOINT -> la instrucción entrypoint define el comando y los parámetros que se ejecutan primero cuando se ejecuta el contenedor. En simples palabras, todos los comandos pasados en la instrucción docker run <image> serán agregados al comando entrypoint

  Opción 1 -> la forma exec es donde especificamos comandos y argumentos, como la sintaxis de los formatos JSON.

  ENTRYPOINT ["executable", "param1", "param2"]

  Opción 2 -> la otra forma es ejecutar un script para ejecutar los comandos que queremos como entrada en el contenedor.

  COPY ./script-entrypoint.sh /

  ENTRYPOINT ["/script-entrypoint.sh"]

  CMD ["postgres"]

- VOLUME -> esta instrucción crea un volumen como punto de montaje dentro del contenedor y es visible desde el host anfitrión marcado con otro nombre.

  VOLUME /var/tmp

- USER -> determina el nombre de usuario a utilizar cuando se ejecuta un contenedor, y adicionalmente cuando se ejecutan comandos como RUN, CMD, ENTRYPOINT o WORKDIR.

  WORKDIR ruta/de/Proyecto

Personalizando imágenes

- Vamos a crear nuestra primera imagen de apache con php, para ello usaremos el fichero Dockerfile. Lo creamos teniendo en cuenta: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- El fichero Dockerfile define una una plantilla que le dice como serán nuestros contenedores

```
# Imagen a usar
FROM php:7.0-apache
# copiamos todos los ficheros en el directorio en cuestion
COPY src/ /var/www/html
# Exponemos el puerto 80
EXPOSE 80
```

- Creamos la imagen con

  docker build -t miapache-php . (el punto es importante le dice donde esta el fichero Dockerfile, qque es en este directorio)

- Comprobamos que existe

  docker images

- Lanzamos la imagen

  docker run -dit --name miapache-php -p 5555:80 miapache-php

Personalizando imagenes

- A partir de aqui podremos parar y arrancar ese contenedor (o borrarlo).

  docker stop miapache-php

  docker start miapache-php

- En este modelo de trabajo, cada ves que cambiemos el contenido de nuestra web deberemos regenerar la imágen local, podemos (o no) eliminar los contenedores viejos y volver a crear/correr el contenedor.

- Agregar contenido a la carpeta de trabajo y rearmar todo.

  docker rmi -f miapache-php (borramos la imagen aunque esré siendo usada)

  docker rm   -f miapache-php (borramos el contenedor aunque se esté ejecurando)

  docker build -t miapache-php . (construimos la imagen)

  docker run -dit --name miapache-php -p 5555:80 miapache-php (la ejecutamos)

- Podemos crearnos un script .sh para hacerlo todo, como run.sh

```
#!/bin/bash
docker rmi  -f miapache-php
docker rm   -f miapache-php
docker build -t  miapache-php .
docker run -dit --name miapache-php -p 5555:80 miapache-php
```

Persistencia de datos

- Por defecto ya hemos indicado que un contenedor está aislado de todo. Hemos visto como podemos conectar el contenedor a un puerto de red para poder acceder a él. Eso incluye al sistema de archivos que contiene. De tal manera que si se elimina el contenedor, se eliminan también sus archivos.
- Si queremos almacenar datos (una web, una base de datos, etc.) dentro de un contenedor necesitamos una manera de almacenarlos sin perderlos.
- Docker ofrece tres maneras:

  A través de volúmenes, que son objetos de Docker como las imágenes y los contenedores.

  Montando un directorio de la máquina anfitrión dentro del contenedor.

  Almacenándolo en la memoria del sistema (aunque también se perderían al reiniciar el servidor).
- Lo normal es usar volúmenes, pero habrá ocasiones en que es preferible montar directamente un directorio de nuestro espacio de trabajo. Por ejemplo, para guardar los datos de una base de datos usaremos volúmenes, pero para guardar el código de una aplicación o de una página web montaremos el directorio.
- La razón para esto último es que tanto nuestro entorno de desarrollo como el contenedor tengan acceso a los archivos del código fuente. Los volúmenes, al contrario que los directorios montados, no deben accederse desde la máquina anfitrión.

Persistencia de datos

- La opción --mount permite crear el enlace entre el directorio de la máquina virtual y el contenedor. La opción tiene tres argumentos separados por comas pero sin espacios: type=bind,source=ORIGEN-EN-MÁQUINA-VIRTUAL,target=DESTINO-EN-CONTENEDOR. Ambos directorios deben existir previamente.

  ```
  docker run -dit --name miapache-php -p 5555:80 --mount type=bind,source="$PWD/src",target=/var/www/html miapache-php
  ```

Persistencia de datos

- En vez de guardar los datos persistentes en la máquina host, Docker dispone de unos elementos llamados volúmenes que podemos asociar también a directorios del contenedor, de manera que cuando el contenedor lea o escriba en su directorio, donde leerá o escribirá será en el volumen.
- Los volúmenes son independientes de los contenedores, por lo que también podemos conservar los datos aunque se destruya el contenedor, reutilizarlos con otro contenedor, etc. La ventaja frente a los directorios enlazados es que pueden ser gestionados por Docker. Otro detalle importante es que el acceso al contenido de los volúmenes sólo se puede hacer a través de algún contenedor que utilice el volumen.
- Vamos a repetir un ejemplo similar al ejemplo anterior, pero utilizando un volumen en vez de un directorio enlazado. En este caso, enlazaremos el directorio /html con un volumen de Docker. directorio de la máquina virtual, el contenedor servirá las páginas contenidas en el directorio de la máquina virtual.
- Podemos crera un volumen de la siguiente manera

```
docker volume create mi-volumen
```

Persistencia de datos

- Crea un contenedor Apache.

  La opción --mount permite crear el volumen o usar uno creado . La opción tiene tres argumentos separados por comas pero sin espacios: type=volume,source=NOMBRE-DEL-VOLUMEN,target=DESTINO-EN-CONTENEDOR. El directorio de destino debe existir previamente.

  sudo docker run -dit -p 5555:80 --name miapache-php --mount type=volume,source=vol-miapache-php,target=/var/www/html miapache-php

- Compruebe que se ha creado un volumen con el nombre asignado al crear el contenedor

```
$ docker volume ls
DRIVER              VOLUME NAME
local               vol-miapache-php
```

- Los volúmenes son entidades independientes de los contenedores, pero para acceder al contenido del volumen hay que hacerlo a través contenedor, más exactamente a través del directorio indicado al crear el contenedor.

Persistencia de datos

- Creeamos ahora un nuevo contenedor que use el mismo volumen

  sudo docker run -dit -p 6666:80 --name miapache-php-2 --mount type=volume,source=vol-miapache-php,target=/var/www/html miapache-php

- Vamos a modificar el html y meterlo en el contenedor primero

  docker cp index.html miapache-php:/var/www/html

- Podemos ver que se ha modificado en los dos sitios (contenedores)

- También podemos crear volumens automáticos (no elegimos su nombre y se crean sobre la marcha) usando la opción -v y enlazarlos a un directorio nuestro, como ya hemos hecho antes.

  docker run -dit --name miapache-php -p 5555:80 -v "$PWD/src":/var/www/html miapache-php

Persistencia de datos

- Los volúmenes son independientes de los contenedores, pero Docker tiene en cuenta qué volúmenes están siendo utilizados por un contenedor.
- Si intenta borrar el volumen del ejemplo anterior mientras los contenedores están en marcha, Docker muestra un mensaje de error que indica los contenedores afectados:

  sudo docker volume rm vol-apache

  Error response from daemon: remove vol-apache: volume is in use - [a6c8a30f7b1dc7a4ef165046daff226ee

  1d6a69573269ca24d57b5b4b6802881, 3b1bcc5a67f38853810972b1da8a67148fad78c6cd6f22b2c823d141be59c81c]

  Detenga los contenedores:

  sudo docker stop apache-volume-1

  sudo docker stop apache-volume.2

- Si intenta de nuevo borrar el volumen del ejemplo anterior ahora que los contenedores están detenidos, Docker sigue mostrando el mensaje de error que indica los contenedores afectados:

  sudo docker volume rm vol-apache

  Error response from daemon: remove vol-apache: volume is in use - [a6c8a30f7b1dc7a4ef165046daff226ee

  1d6a69573269ca24d57b5b4b6802881, 3b1bcc5a67f38853810972b1da8a67148fad78c6cd6f22b2c823d141be59c81c]

  Borre los contenedores:

  sudo docker rm apache-volume-1

  sudo docker rm apache-volume.2

Persistencia de datos

- Si intenta de nuevo borrar el volumen del ejemplo anterior ahora que no hay contenedores que utilicen el volumen, Docker ahora sí que borrará el volumen:

  sudo docker volume rm vol-apache

  vol-apache

  Compruebe que el volumen ya no existe:

  docker volume ls

  DRIVER VOLUME NAME

- Tenga en cuenta que al borrar un volumen, los datos que contenía el volumen se pierden para siempre, salvo que hubiera realizado una copia de seguridad.

Levantando un WordPress

Para crear un blog con WordPress necesitamos tener una base de datos dónde almacenar las entradas. Así que empezaremos creándola y después crearemos el contenedor de nuestro blog.

```
docker run -d --name wordpress-db \
    --mount source=wordpress-db,target=/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=secret \
    -e MYSQL_DATABASE=wordpress \
    -e MYSQL_USER=manager \
    -e MYSQL_PASSWORD=secret mariadb:10.3.9
```

- El principal cambio en docker run con respecto a la última vez es que no hemos usado -p (el parámetro para publicar puertos) y hemos añadido el parámetro -d.
- Lo primero que habremos notado es que el contenedor ya no se queda en primer plano. El parámetro -d indica que debe ejecutarse como un proceso en segundo plano. Así no podremos pararlo por accidente con Control+C.

Levantando un Wordpress

- Lo segundo es que vemos que el contenedor usa un puerto, el 3306/tcp, pero no está linkado a la máquina anfitrión. No tenemos forma de acceder a la base de datos directamente. Nuestra intención es que solo el contenedor de WordPress pueda acceder.
- Luego una serie de parámetros -e que nos permite configurar nuestra base de datos.
- Por último, el parámetro --mount nos permite enlazar el volumen que creamos en el paso anterior con el directorio /var/lib/mysql del contenedor. Ese directorio es donde se guardan los datos de MariaDB. Eso significa que si borramos el contenedor, o actualizamos el contenedor a una nueva versión, no perderemos los datos porque ya no se encuentran en él, si no en el volumen. Solo lo perderíamos si borramos explícitamente el volumen.

Levantando un Wordpress

- Creamos el directorio donde queremos almacenar nuestr WordPress

  mkdir -p ~/Sites/wordpress && cd ~/Sites/wordpress

- Y dentro de este directorio arrancamos el contenedor:

```
docker run -d --name wordpress \
    --link wordpress-db:mysql \
    --mount type=bind,source="$(pwd)"/wordpress,target=/var/www/html \
    -e WORDPRESS_DB_USER=manager \
    -e WORDPRESS_DB_PASSWORD=secret \
    -p 8080:80 \
    wordpress:4.9.8
```

- Cuando termine la ejecución, si accedemos a la dirección http://localhost:8080/, ahora sí podremos acabar el proceso de instalación de nuestro WordPress. Si listamos el directorio wordpress comprobaremos que tenemos todos los archivos de instalación accesibles desde el directorio anfitrión.

Levantando un WordPress

- Ejercicios:

  Para los contenedores, tanto el de WordPress como el MariaDB.

  Borra ambos.

  Vuelve a crearlos y mira como ya no es necesario volver a instalar WordPress.

  Vuelve a borrarlos y borra también el volumen.

  Vuelve a crear el volumen y los contenedores y comprueba que ahora sí hay que volver a instalar WordPress.

  Explica por qué

Usando red propia

- También podemos crear una red propia y asignarla a nuestro contenedores con una ip específica, de esta manera estos contenedores solo se inteconectan entre ellos y no con la red por defecto. Los aislamos

  sudo docker network create mi-network

- Creamos los contenedores asignado la nueva red

```
docker run -d --name wordpress-db \
    --mount source=wordpress-db,target=/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=secret \
    -e MYSQL_DATABASE=wordpress \
    -e MYSQL_USER=manager \
    -e MYSQL_PASSWORD=secret mariadb:10.3.9

docker run -d --name wordpress \
    --link wordpress-db:mysql \
    --mount type=bind,source="$(pwd)"/wordpress,target=/var/www/html \
    -e WORDPRESS_DB_USER=manager \
    -e WORDPRESS_DB_PASSWORD=secret \
    -p 8080:80 \
    wordpress:4.9.8
```

Docker Compose

- El cliente de Docker es engorroso para crear contenedores, así como para crear el resto de objetos y vincularlos entre sí.
- Para automatizar la creación, inicio y parada de un contenedor o un conjunto de ellos, Docker proporciona una herramiento llamada Docker Compose.
- Compose es una herramienta para definir y ejecutar aplicaciones multi-contenedor. Con un solo comando podremos crear e iniciar todos los servicios que necesitamos para nuestra aplicación.
- Los casos de uso más habituales para docker-compose son:
  - Entornos de desarrollo
  - Entornos de testeo automáticos (integración contínua)
  - Despliegue en host individuales (no clusters)
- Compose tiene comandos para manejar todo el ciclo de vida de nuestra aplicación:
  - Iniciar, detener y rehacer servicios.
  - Ver el estado de los servicios.
  - Visualizar los logs.
  - Ejecutar un comando en un servicio.

Docker Compose

- Vamos a crear el fichero docker-compose.yaml

```
version: '3'

services:
  db:
    image: mariadb:10.3.9
    container_name: mariadb
    volumes:
      - data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=manager
      - MYSQL_PASSWORD=secret
  web:
    image: wordpress:4.9.8
    container_name: wordpress
    depends_on:
      - db
    volumes:
      - ./wordpress:/var/www/html
    environment:
      - WORDPRESS_DB_USER=manager
      - WORDPRESS_DB_PASSWORD=secret
      - WORDPRESS_DB_HOST=db
    ports:
      - 8080:80

volumes:
  data:
```

Docker Compose

- Iniciar servicios. Vamos a ejecutar esta aplicación y luego procederemos a explicarla:

  docker-compose up -d

- docker-compose ps solo muestra información de los servicios que se define en docker-compose.yaml, mientras que docker muestra todos.
- El nombre de cada contenedor se define con container_name: mariadb, si no se pone será el nombre del directorio y de la imagen
- Detener servicios

  docker-compose stop

- Borrar servicios

  docker-compose down

- Esto borra los contenedores, pero no los volúmenes. Así que si hemos creado bien la aplicación nuestros datos están a salvo.
- Si queremos borrar también los volúmenes:

  docker-compose down -v

Docker Compose

- Estructura de la configuración. Veamos la configuración por partes:

  version: '3'. Compose se actualiza a menudo, con lo que el archivo de configuración va adquiriendo nuevas funcionalidades. La versión '3' (es una cadena, importante poner comillas) es la última y para conocer todas sus características mira la página de referencia de la versión 3 de Compose.

- volumes: data:

  Ya hemos indicado que es importante guardar los datos volátiles de las aplicaciones en volúmenes. En este caso hemos creado un volumen llamado data. Recordemos que Compose siempre añade como prefijo el nombre del directorio, con lo que el nombre real del volumen es wordpress_data. Podemos comprobarlo con el cliente de docker como hicimos en el capítulo de volúmenes:

- Nos saltamos la sección de redes (networks) y vamos a la sección de servicios, que son los contenedores que precisa o componen nuestra aplicación

Docker Compose

- **Primero la base de datos:**

```
services:
    db:
        image: mariadb:10.3.9
        volumes:
            - data:/var/lib/mysql
        environment:
            - MYSQL_ROOT_PASSWORD=secret
            - MYSQL_DATABASE=wordpress
            - MYSQL_USER=manager
            - MYSQL_PASSWORD=secret
```

- Después de abrir la parte de servicios, el primer nivel indica el nombre del servicio db, que genera el contenedor wordpress_db. Lo que vemos a continuación es lo mismo que hicimos en la sección anterior pero de forma parametrizada. Si recordamos, para levantar nuestra base de datos, indicamos la imagen (línea 3), luego montamos los volúmenes (línea 4), y después indicamos las variables de entorno que configuraban el contenedor (línea 6).
- Es decir, lo anterior es equivalente, excepto por el nombre, a:

```
$ docker run -d --name wordpress-db \
        --mount source=wordpress-db,target=/var/lib/mysql \
        -e MYSQL_ROOT_PASSWORD=secret \
        -e MYSQL_DATABASE=wordpress \
        -e MYSQL_USER=manager \
        -e MYSQL_PASSWORD=secret mariadb:10.3.9
```

Docker Compose

- Y después nuestro WordPress:

```
services:
    web:
        image: wordpress:4.9.8
        depends_on:
            - db
        volumes:
            - ./target:/var/www/html
        environment:
            - WORDPRESS_DB_USER=manager
            - WORDPRESS_DB_PASSWORD=secret
            - WORDPRESS_DB_HOST=db
        ports:
            - 8080:80
```

- En este caso la equivalencia es al comando:

```
$ docker run -d --name wordpress \
    --link wordpress-db:mysql \
    --mount type=bind,source="$(pwd)"/wordpress,target=/var/www/html \
    -e WORDPRESS_DB_USER=manager \
    -e WORDPRESS_DB_PASSWORD=secret \
    -p 8080:80 \
    wordpress:4.9.8
```

Docker Compose

- La equivalencia de los parámetros es la siguiente:

parámetro Docker	parámetro Composer
--link	depends_on
--mount	volumes
-e	environment
-p,--publish	ports
image	From

Docker LAMP

- Apache con PHP

```
# Indicamos la versión
version: '3.7'

# Iniciamos los servicios
services:
  # Apache con PHP
  php-httpd:
    image: php:7.3-apache
    container_name: apache-php
    ports:
      - 80:80
    volumes:
      - './src:/var/www/html'
```

Docker LAMP

- MariaDB

```
mariadb:
    image: mariadb:10.5.2
    container_name: mariadb-lamp
    volumes:
      - mariadb-volume:/var/lib/mysql
    environment:
      TZ: 'Europe/Rome'
      MYSQL_ALLOW_EMPTY_PASSWORD: 'no'
      MYSQL_ROOT_PASSWORD: 'rootpwd'
      MYSQL_USER: 'testuser'
      MYSQL_PASSWORD: 'testpassword'
      MYSQL_DATABASE: 'testdb'

volumes:
  mariadb-volume:
```

Docker LAMP

- Variables a tener en cuenta

  TZ: Zona para las fechas

  MYSQL_ALLOW_EMPTY_PASSWORD: Habilita el uso de password en blanco para root

  MYSQL_ROOT_PASSWORD: Campo obligatorio, es pasword de root

  MYSQL_DATABASE: Opcional, es el nombre de la base de datos inicial

  MYSQL_USER: Opcional, es el nombre del usuario que nos crearemos para la base de datos

  MYSQL_PASSWORD: Es el password de MYSQL_USER

Docker LAMP

- PHP MyAdmin

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin
   container_name: phpmyadmin-lamp
  links:
    - 'mariadb:db'
  ports:
    - 8081:80
```

Usamos links para segurarnos que se enlanza en la misma red con MariaDB

Docker Compose y Docker Files

- Podemos personalizar aún mas usando dockerfiles con docker compose, usando build, en vez de image
- Apache

```
# Indicamos la versión
version: '3.7'
# Iniciamos los servicios
services:
  # Apache con PHP
  apache-php:
    build: ./apache-php
    container_name: apache-php-lamp
    ports:
      - 80:80
    volumes:
      - './src:/var/www/html'
    networks:
      - lamp-network
```

Docker Compose y Docker Files

- En el directorio apache-php tenemos nuestro dockfile

```
# Apache + PHP + complementos
FROM php:7.4-apache
# Complementos, es importante que no nos pasemos con el run!! (cuanto más lineas peor, más lento)
RUN docker-php-ext-install pdo pdo_mysql mysqli
```

Docker Compose y Docker Files

- MariaDB

```
# Maria DB
  mariadb:
    build: ./mariadb
    container_name: mariadb-lamp
    volumes:
      #  - ./mariadb_data:/var/lib/mysql podríamos usar un directorio local llamado mariadb_data y no volumen
      - mariadb-volume:/var/lib/mysql
    networks:
      - lamp-network
```

Docker Compose y Docker Files

- En el directorio mariadb tenemos el Dockerfile y un fichero sql que se ejecuta al inicar por primera vez la BD creando las tablas y los datos que queramos que tenga

```
## MariaDB
FROM mariadb:10.5
# Configuramos BBDD
ENV MYSQL_ROOT_PASSWORD 123
ENV MYSQL_USER joseluis
ENV MYSQL_PASSWORD 123
ENV MYSQL_DATABASE testdb
# Copiamos los ficheros sql para que se ejecuten
COPY ./sql /docker-entrypoint-initdb.d/
```

Docker Compose y Docker Files

- En el directorio mariadb/sql, es el fichero para inicializar nuestra BD. Mantén el nombre a init-db-sql

```
USE testdb;

CREATE TABLE test (
        nombre varchar(30),
        email varchar(50)
);

INSERT INTO test (nombre, email)
VALUES
        ('Jose Luis', 'joseluis@docker.com'),
        ('Soraya', 'soraya@docker.com'),
        ('Victor', 'victor@docker.com');
```

Docker Compose y Docker Files

- PHPMyAdmin, Volumen y Redes (para que estén asiladas del resto de contenedores)

```
# PHPMyAdmin
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin-lamp
    links:
      - 'mariadb:db'
    ports:
      - 8081:80
    networks:
      - lamp-network

volumes:
  mariadb-volume:

networks:
  lamp-network:
    driver: bridge
```

Referencias

- Docker: Sitio oficial
- DockerHub: Imágenes
- Docker File: Manejo de Dockerfile
- Docker Compose: Manejo de Compose