DOCKER

¿QUÉ ES DOCKER?

Docker es una herramienta que permite **empaquetar y distribuir aplicaciones** con todas sus dependencias en contenedores, que pueden ser desplegados con gran facilidad en todo tipo de sistemas operativos, servidores, nubes públicas o privadas etc.

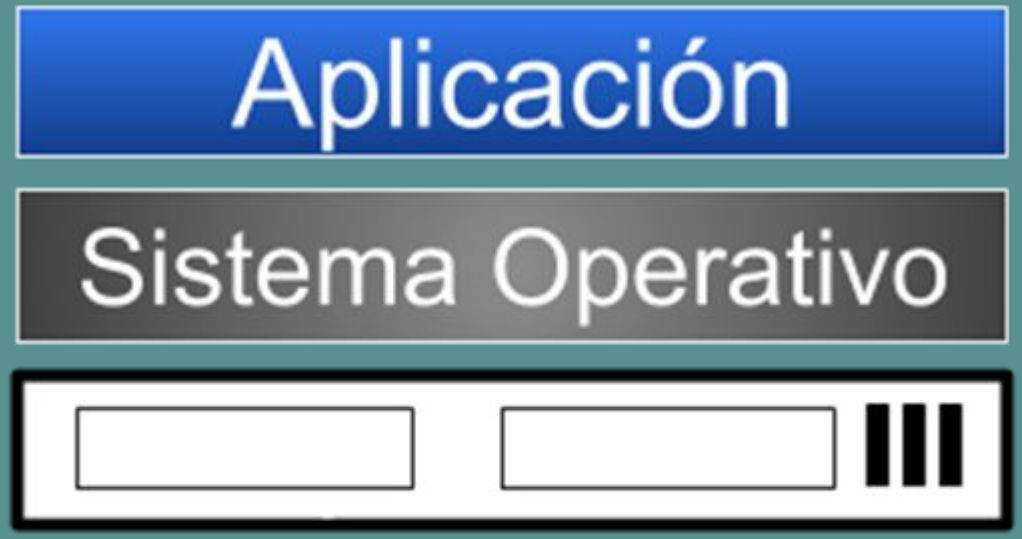
¿QUÉ ES DOCKER (DE MANERA FORMAL)?

Docker es una **tecnología de virtualización "ligera"** cuyo elemento básico es la utilización de contenedores en vez de máquinas virtuales y cuyo objetivo principal es el despliegue de aplicaciones encapsuladas en dichos contenedores.

EVOLUCIÓN EN EL DESPLIEGUE DE APLICACIONES

- Arquitectura de un único servidor
- Virtualización
- Contenedores

ARQUITECTURA DE UN ÚNICO SERVIDOR. ARQUITECTURA



Servidor Físico

ARQUITECTURA DE UN ÚNICO SERVIDOR. LIMITACIONES

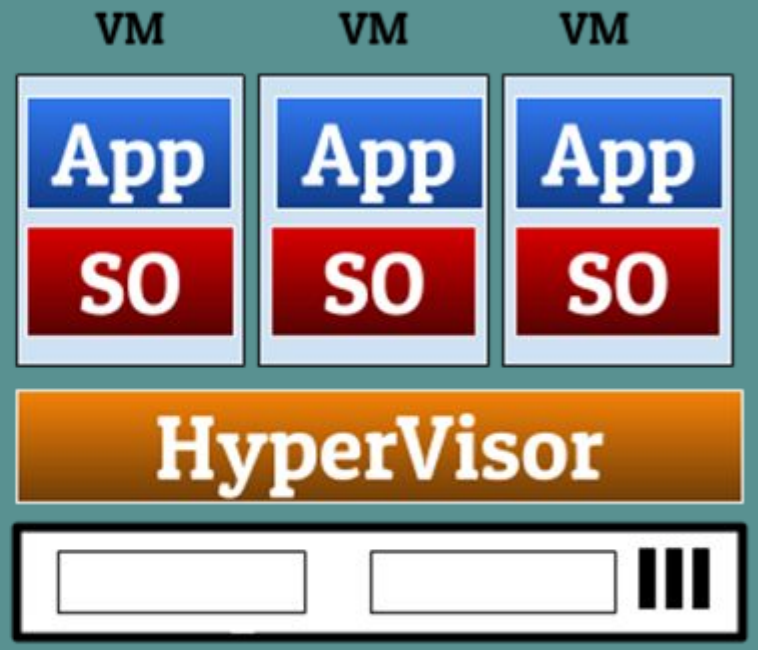
- Era un enfoque de costes elevados porque era necesaria una máquina de precio elevado.
- El despliegue de aplicaciones era un proceso lento que podía suponer en algunos casos una parada del servicio.
- El escalado de las aplicaciones era costoso y complicado. Si nuestra máquina llegaba un momento que se quedaba corta no había más remedio que sustituirla por otra más potente.

ARQUITECTURA DE UN ÚNICO SERVIDOR. LIMITACIONES

- La migración a otro sistema era un proceso complicado. La configuración del nuevo servidor, de su sistema operativo y de todas las dependencias tenía que ser compatible. Esto era algo complicado de gestionar y en algunos casos difícil de conseguir.
- En muchos momentos, aquellos en los que el servidor no se utilizaba aprovechando su potencia, se estaban desperdiciando recursos.
- Había mucha dependencia del fabricante del servidor

VIRTUALIZACIÓN. ARQUITECTURA

Servidor Físico
VIRTUALIZACIÓN HOSTED



Servidor Físico
VIRTUALIZACIÓN NATIVA

VIRTUALIZACIÓN. BENEFICIOS

- Hay un mejor aprovechamiento de los recursos. Un servidor grande y potente se puede compartir entre distintas aplicaciones.
- Los procesos de migración y escalado no son tan dolorosos, simplemente le doy más recursos a la máquina virtual dentro de mi servidor o bien muevo la máquina virtual a un nuevo servidor, propio o en la nube, más potente y que también tenga características de virtualización.
- Esto además hizo que aparecieran nuevos modelos de negocio en la nube que nos permiten en cada momento tener y dimensionar las máquinas virtuales según nuestras necesidades y pagar únicamente por esas necesidades.

VIRTUALIZACIÓN. INCONVENIENTES

- Todas las máquinas virtuales siguen teniendo su propia memoria RAM, su almacenamiento y su CPU que será aprovechada al máximo...o no.
- Para arrancar las máquinas virtuales tenemos que arrancar su sistema operativo al completo.
- La portabilidad no está garantizada al 100%.

CONTENEDORES. ARQUITECTURA

Se empaquetan
la aplicación
y las dependencias

Aplicación y Host
comparten Kernel
a través del
Docker Engine

Entorno de ejecución
para contenedores

CONTENEDORES. CARACTERÍSTICAS

- 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).

CONTENEDORES. CARACTERÍSTICAS

- 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.

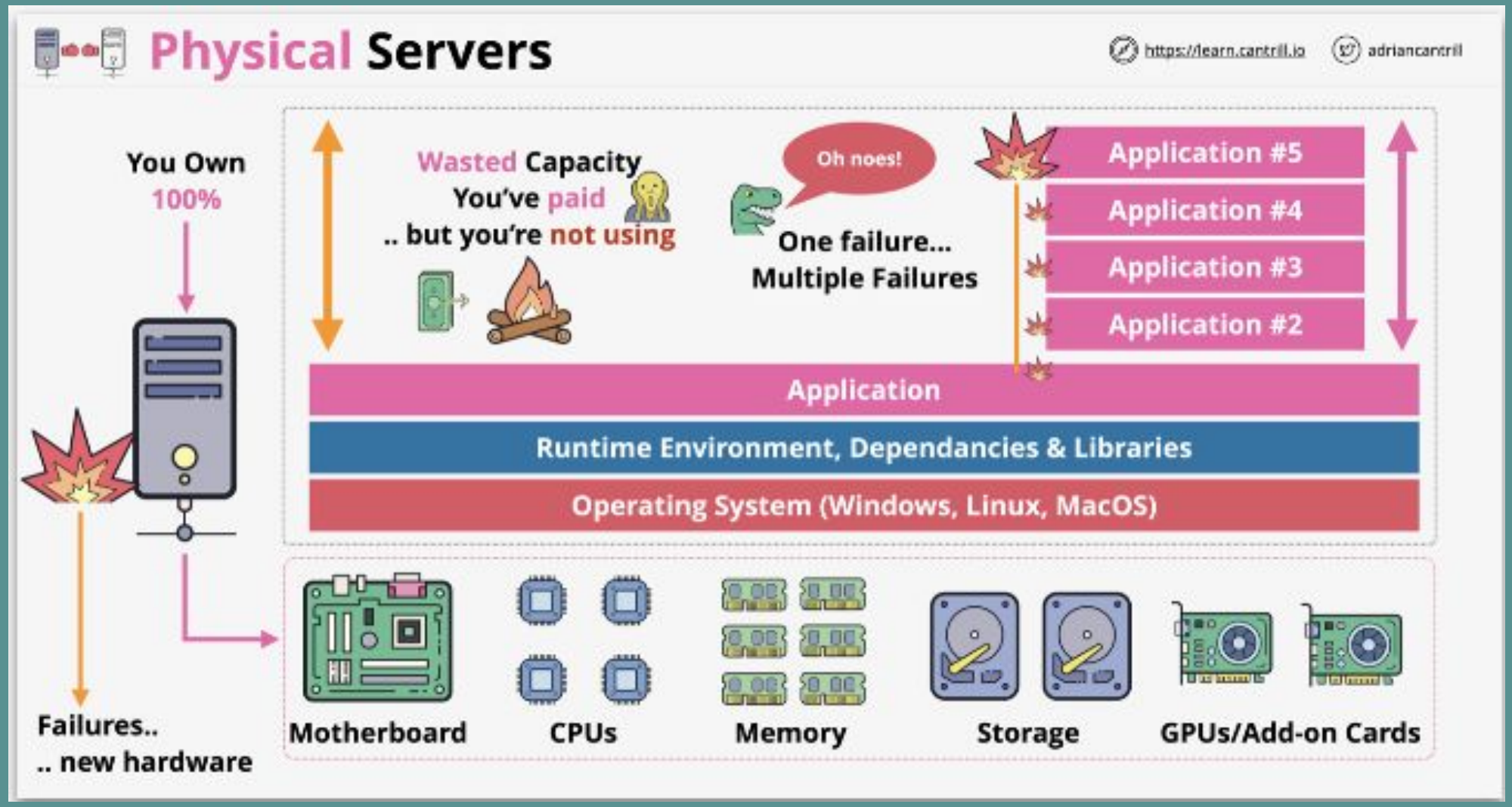
CONTENEDORES. BENEFICIOS

- Una mayor velocidad de arranque, ya que prescindimos de la carga de un sistema operativo invitado. Estamos hablando de apenas segundos para arrancar un contenedor (a veces menos).
- Una gran portabilidad, ya que los contenedores empaquetan tanto las aplicaciones como sus dependencias de tal manera que pueden moverse a cualquier sistema en el que tengamos instalados el Docker Engine, y este se puede ser instalado en casi todos, por no decir todos.
- Una mayor eficiencia ya que hay un mejor aprovechamiento de los recursos. Ya no tenemos que reservar recursos, como hacemos con las máquinas virtuales, sin saber si serán aprovechados al máximo o no.

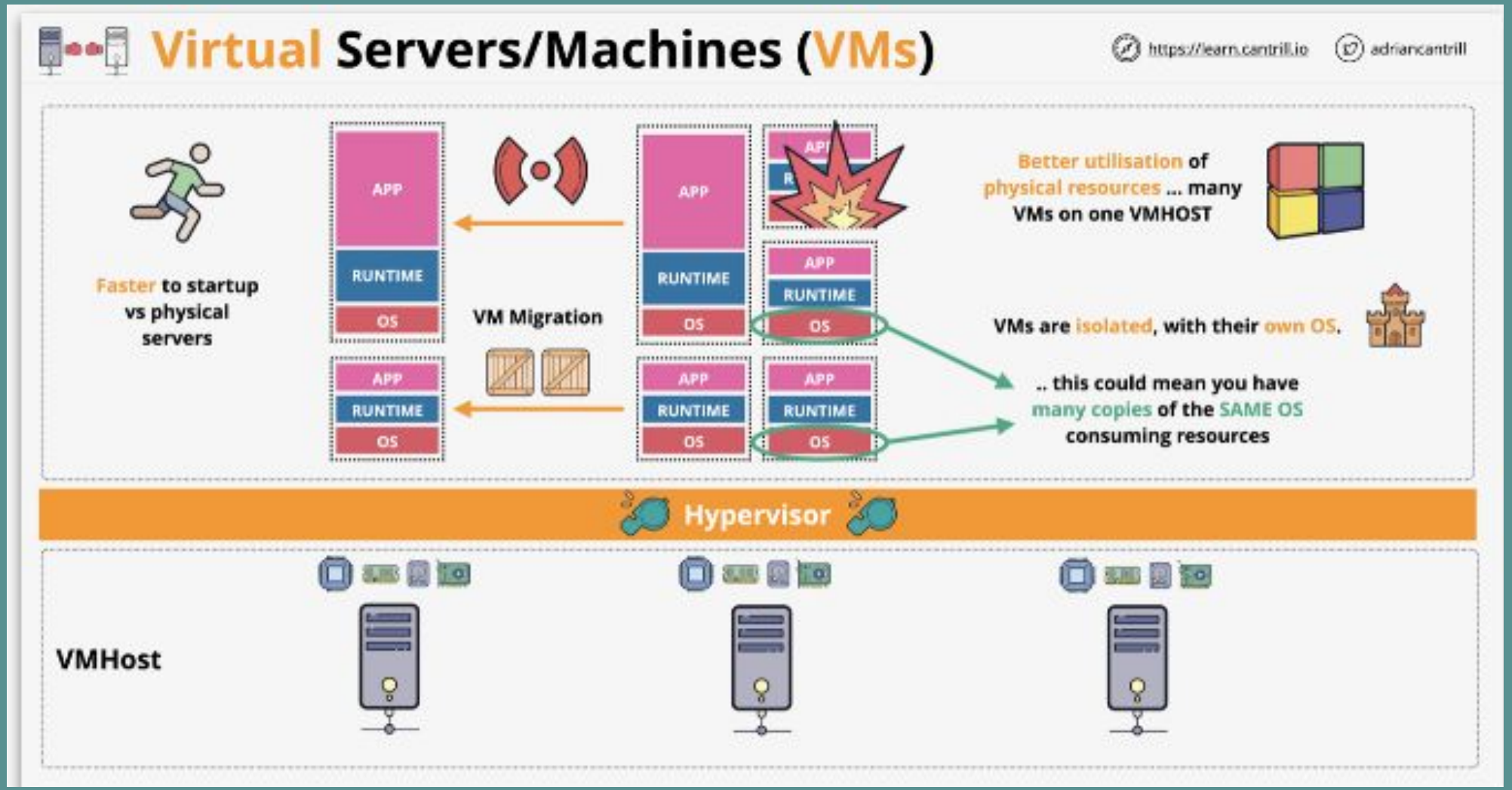
CONTENEDORES. INCONVENIENTE

- Los contenedores son más frágiles que las máquinas virtuales y en ocasiones se quedan en un estado desde el que no podemos recuperarlos. No es algo frecuente, pero ocurre y para eso hay soluciones como la orquestación de contenedores.

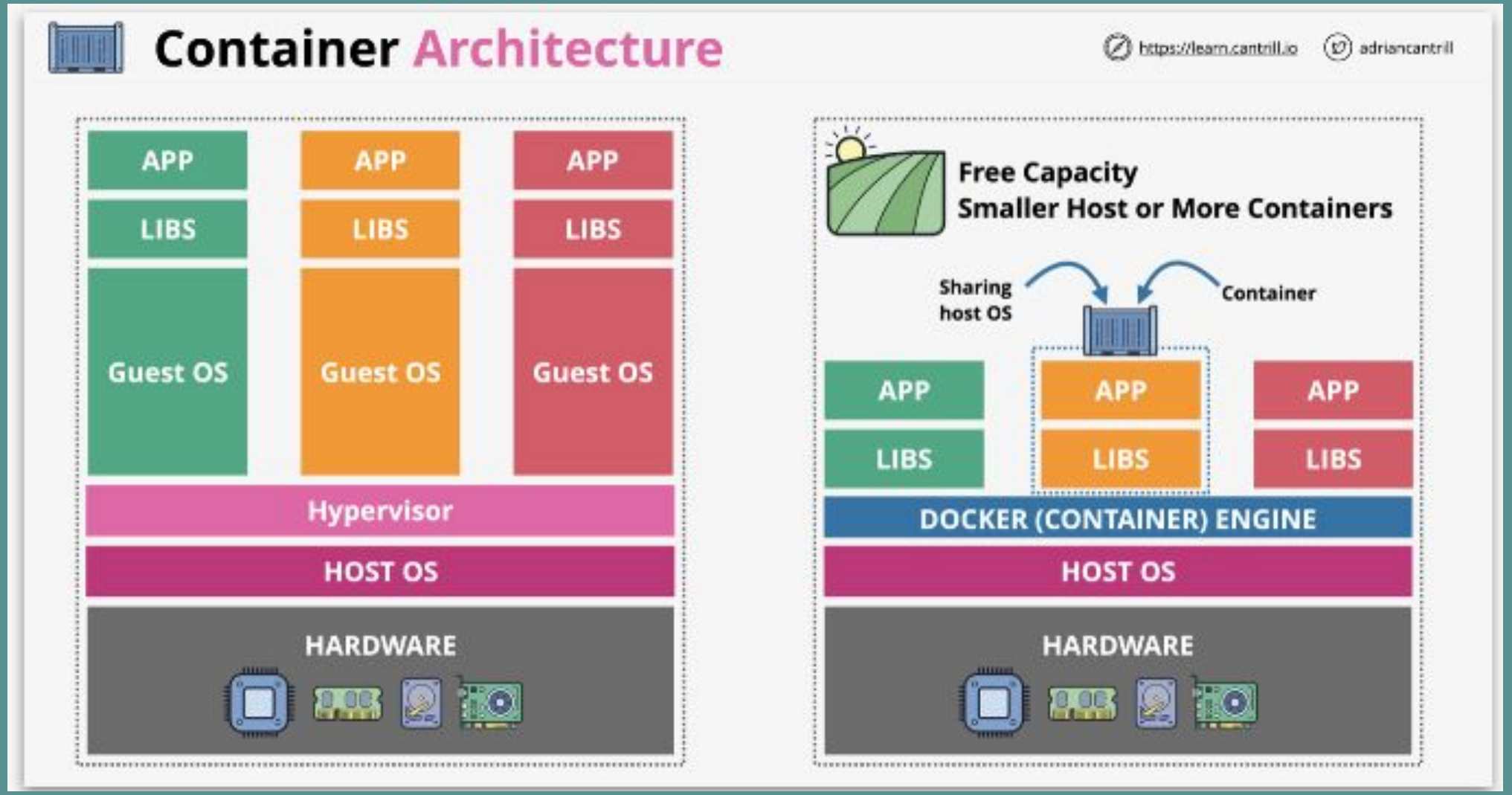
RESUMEN DE ARQUITECTURAS



RESUMEN DE ARQUITECTURAS



RESUMEN DE ARQUITECTURAS



IMAGEN

- Plantilla (ya sea de una aplicación de un sistema) que podremos utilizar como base para la ejecución posterior de nuestras aplicaciones (contenedores).
- Archivo 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

- 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 cual 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
- 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/

NOTA: 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.

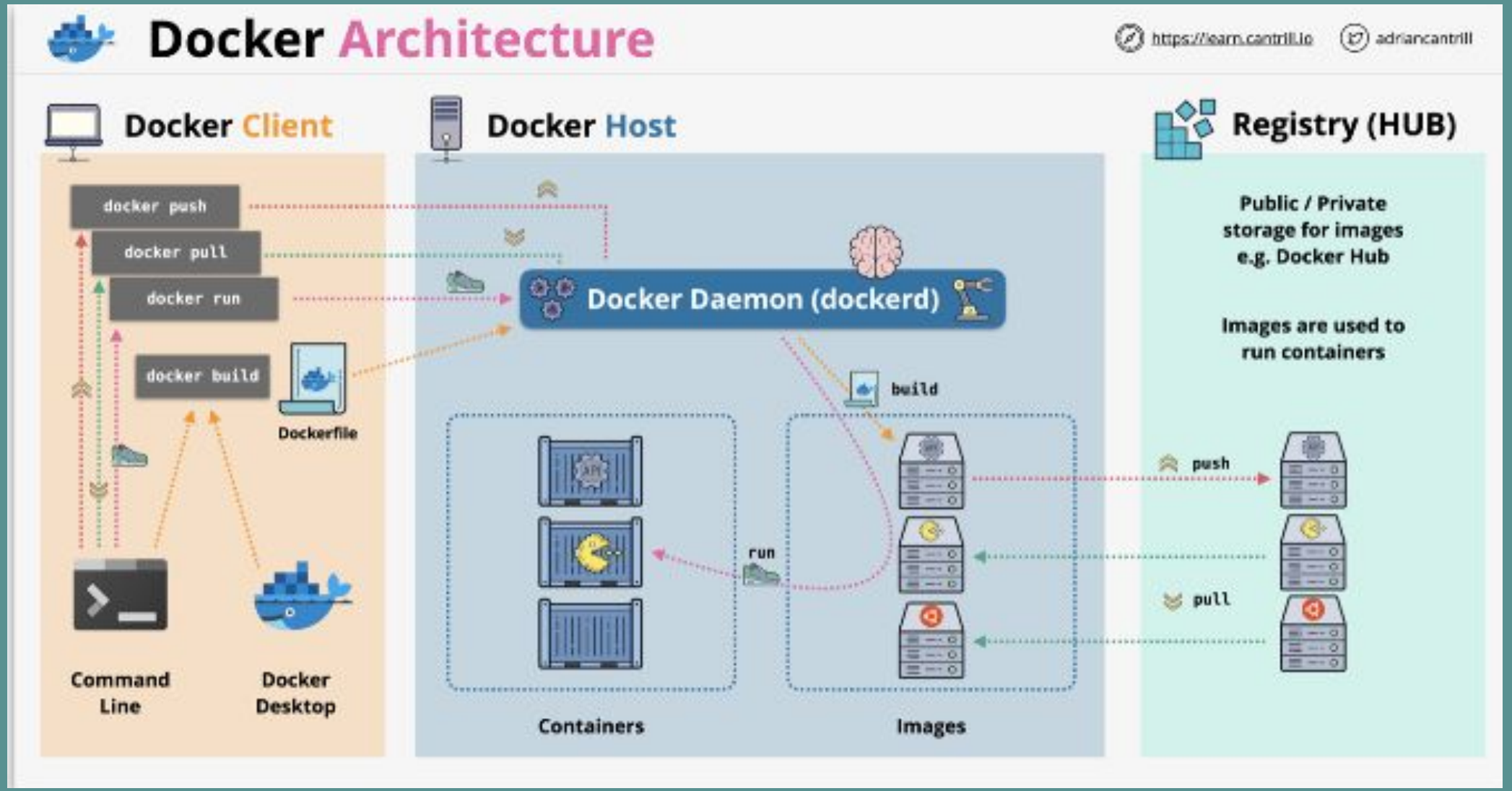
DOCKER HUB. VENTAJAS

- Tiene una gran variedad de imágenes disponibles para que usemos. La gran mayoría son públicas y gratuitas.
- Me permite crear y distribuir imágenes de manera muy sencilla. No olvidemos que es el repositorio por defecto para toda instalación de Docker.
- Me permite crear organizaciones para poder crear equipos y añadir posteriormente miembros, con sus respectivos permisos. ESTO YA NO ES GRATUITO
- Dispone de un interfaz web de fácil utilización.

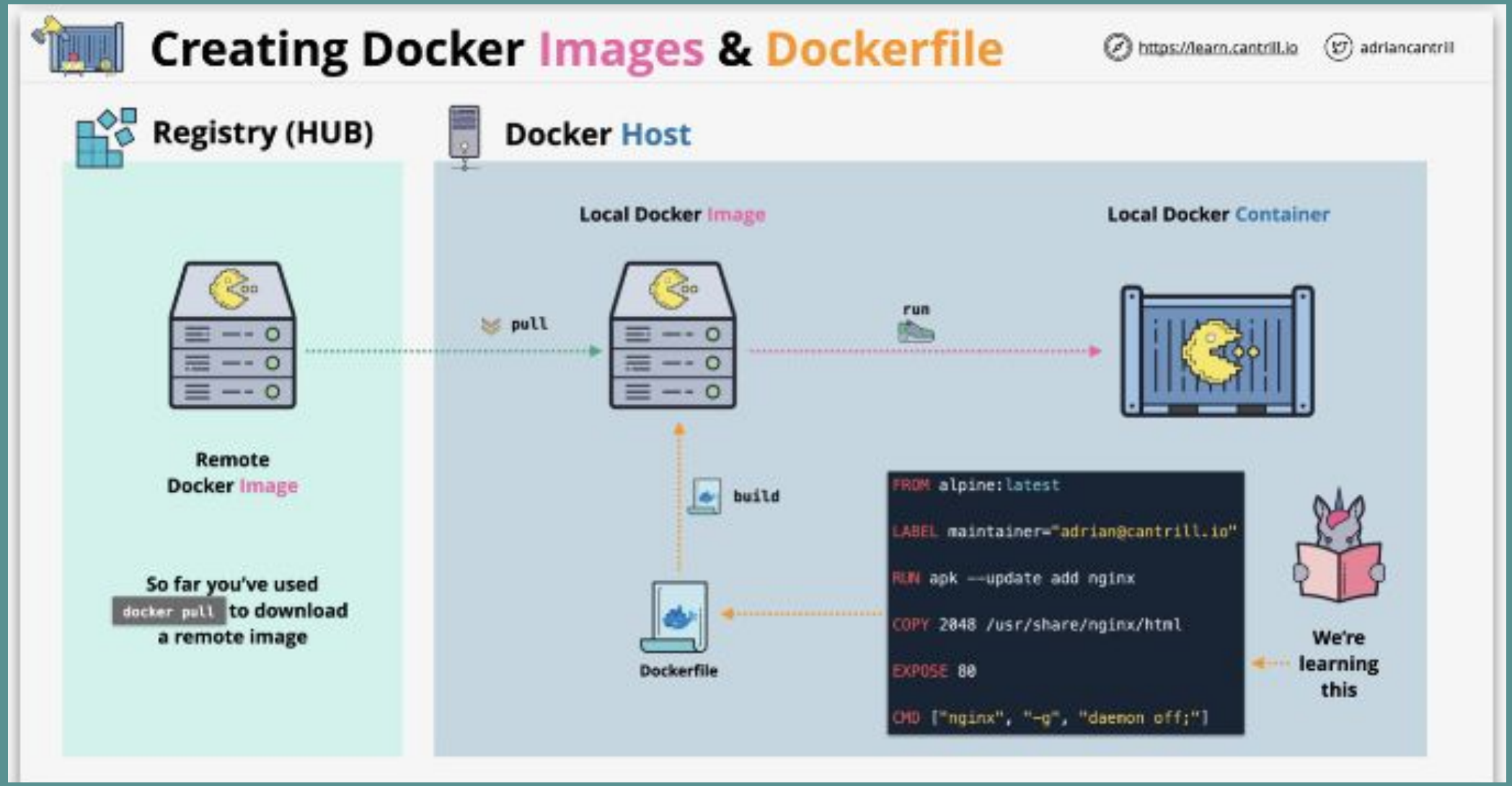
DOCKER HUB. COMUNIDAD

- Una de las grandes ventajas de la tecnología de contenedores que nos ofrece Docker es el ecosistema alrededor.
- A través de Docker Hub (https://hub.docker.com/) podemos acceder a gran cantidad de contenedores gratuitos y libres que nos permiten todo tipo de tecnologías de una manera muy rápida y sin prácticamente interferir en el rendimiento de nuestro sistema operativo.

ARQUITECTURA DOCKER



ARQUITECTURA DOCKER



ARQUITECTURA DOCKER

Ventajas de Docker:

- Puedo probar todas versiones de los distintos sistemas que van apareciendo.
- PARA PROBAR Y USAR CUALQUIER SERVICIO Y CUALQUIER APLICACIÓN NO TENGO QUE INSTALAR NADA EN MI SISTEMA, sea cual sea el servicio o la aplicación que se me ocurra, siempre la tengo en Docker Hub (la busco, averiguo cuál es la versión que quiero y lanzo el contenedor necesario).
- En cualquier equipo donde se haya instalado docker de manera previa, el contenedor que quiero que los demás usen va a funcionar con total seguridad.

INSTALACIÓN DE DOCKER EN WINDOWS

https://youtu.be/ozp84CCh0Uc?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgI

INSTALACIÓN DE DOCKER EN LINUX

- UBUNTU 20.04: https://youtu.be/PoRA7dAhhHA?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgI

- CENTOS8:

  https://youtu.be/NQfnWLOlonY?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgI

INSTALACIÓN DE DOCKER. RECOMENDACIONES

- Ejecutar Docker como un usuario que no sea root, es decir, ejecutarlo sin sudo.

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

INSTALACIÓN DE DOCKER. RECOMENDACIONES

- 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, así que recomiendo dejarlo así si lo vamos a usar habitualmente.

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.

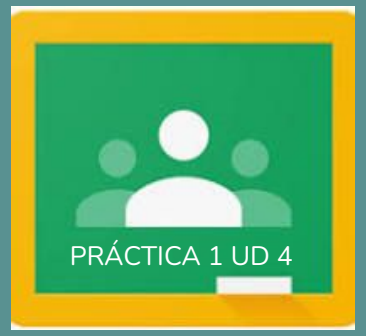
> sudo systemctl disable docker

INSTALACIÓN DE DOCKER. RECOMENDACIONES

- Comprobar que Docker se ha instalado correctamente. Para ello ejecutamos:

```
> docker run hello-world
```

NOTA: al hacer esto, la imagen hello-word se descarga desde el repositorio que se encuentra en el registro que vayamos a utilizar, en nuestro caso DockerHub. Muestra el mensaje de bienvenida, que es la consecuencia de crear y arrancar un contenedor basado en esa imagen.



TAREA 1. INSTALACIÓN DE DOCKER.

- Instalar en vuestro equipo la versión adecuada de docker.
- Configurar vuestro sistema para poder ejecutar docker sin permisos de administrador (en caso de ser necesario).
- Ejecutar el comando docker que me permite obtener la versión instalada y el comando docker para obtener el "HOLA MUNDO" de esta tecnología.
- Crear una cuenta personal de DockerHub.

https://docs.docker.com/get-docker/

https://hub.docker.com/

DOCKER

EJECUCIÓN DE CONTENEDORES.

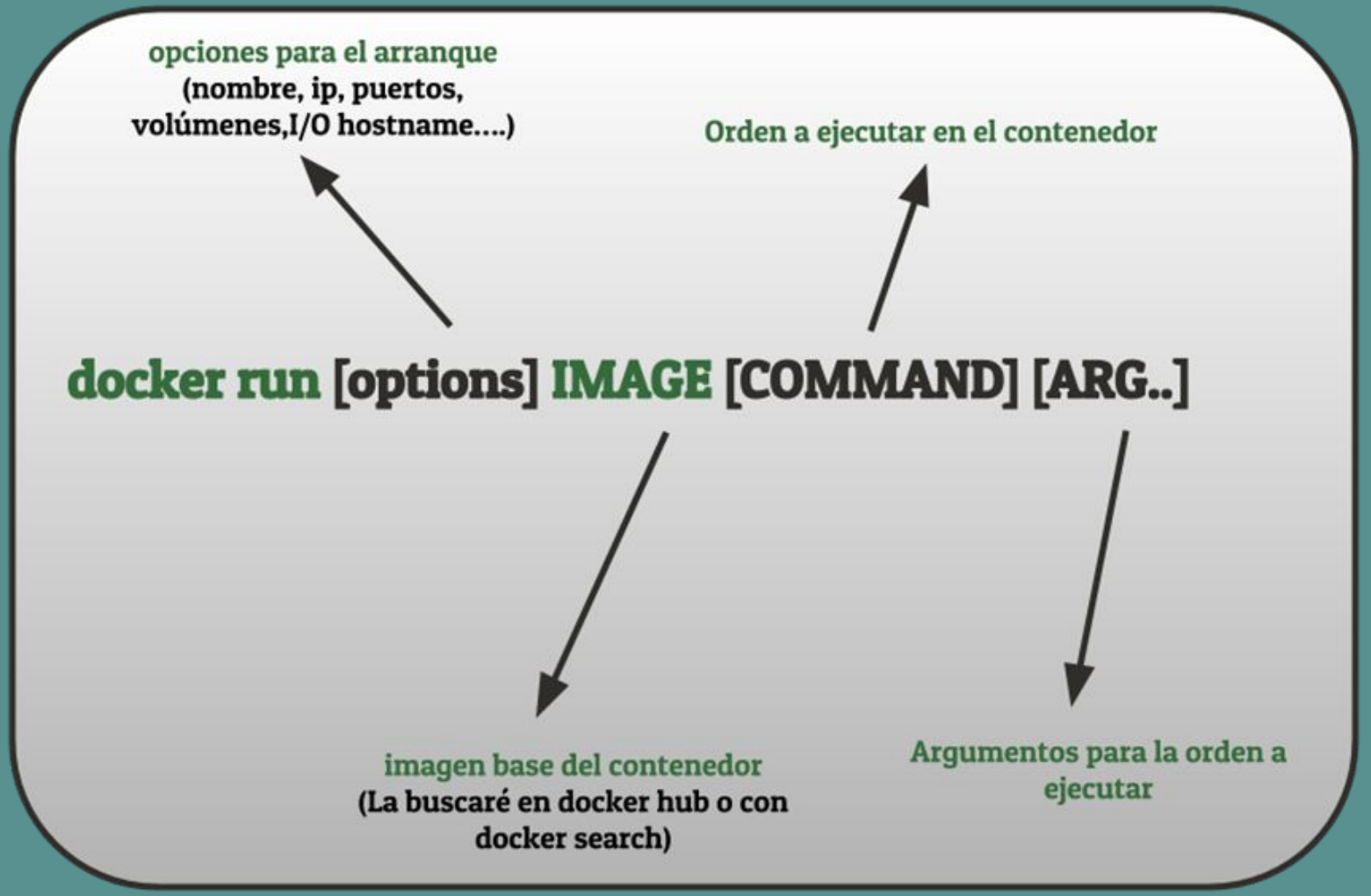
- Para descargar desde el repositorio una imagen con la versión indicada: *docker pull nombre_imagen:version*

  NOTA: si no indicamos versión, descarga la última versión.

- Para poner en ejecución contenedores en base a una imagen de referencia que le indiquemos: *docker run*

  NOTA: Una CUESTIÓN IMPORTANTE que debemos tener en cuenta al usar docker run es que si ejecutamos 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.

EJECUCIÓN DE CONTENEDORES.



EJECUCIÓN DE CONTENEDORES.OPCIONES

- *-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 para el contenedor.
- *--help* para obtener ayuda de las opciones de docker.
- *--interactive o -i* para mantener la STDIN* abierta en el contenedor.

*Entrada estándar (teclado)

EJECUCIÓN DE CONTENEDORES.OPCIONES

- *--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.

EJECUCIÓN DE CONTENEDORES.OPCIONES

- *--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.
- *--wordirk o -w* para establecer el directorio de trabajo en un contenedor.

EJECUCIÓN DE CONTENEDORES. EJERCICIOS

Descargar una imagen Ubuntu 18:04 de manera previa

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.

Crear un contenedor de centOs:18.04 y listar el contenido de la carpeta /

EJECUCIÓN DE CONTENEDORES.SOLUCIÓN

Descargar una imagen Ubuntu 18:04 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 contenido de la carpeta /

> docker run centOs:18.04 ls /

bin etc lib lost+found mnt proc run srv tmp var

dev home lib64 media opt root sbin sys usr

EJECUCIÓN DE CONTENEDORES

¿Cómo podemos comprobar qué contenedores tenemos? Con *docker ps*

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

EJECUCIÓN DE CONTENEDORES

IMPORTANTE!

Si haces *docker ps -a*, verás que los contenedores que acabamos de crear están parados y son, por tanto, inaccesibles. Para solucionar esto, al hacer docker run:

- **Siempre** usar el flag **-it** al ejecutar una orden docker run si es un contenedor que no tiene servicios, para abrir la entrada estándar del contenedor que estamos ejecutando y permitir la posibilidad de abrir un terminal en el contenedor.
- **Al final** siempre debe ir una **orden que abra un shell (/bin/bash)**. Si no lo hacemos, se sobreescribirá la orden de arranque de algunos contenedores y será imposible volver a arrancarlos.

ARRANQUE DE CONTENEDORES

Si los contenedores no son contenedores con servicios* se paran tras salir la primera vez de interactuar con ellos. Para iniciarlos de nuevo: *docker start*

Ejemplo:

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

> docker start servidorWeb
```

* Contenedores que alojan servicios, como por ejemplo, servidores web, de bases de datos, etc.

EJECUCIÓN DE SERVICIOS

Para la ejecución de contenedores vamos a tener que en cuenta varias cosas:

- Usar el flag -d para que el servicio se ejecute en modo background o dettach. Si no lo hacemos se bloqueará el terminal mostrando el log del servicio (en ciertas ocasiones puede interesarnos) y tendremos que salir del mismo con Ctrl+C. Esto para el contenedor, aunque podremos arrancarlo posteriormente.

EJECUCIÓN DE SERVICIOS

- Si queremos que el servicio que vamos a lanzar sea accesible desde el exterior tendremos que añadir el flag -p de la siguiente manera *-p PUERTO_EN_HOST:PUERTO_EN_CONTENEDOR* que normalmente sería el puerto por defecto del servicio. Esto es una REDIRECCIÓN DE PUERTOS.
- Podemos tener varias reglas -p al arrancar (dependiendo del servicio será necesario) y es muy importante recordar que no podemos tener dos servicios escuchando en el mismo puerto. Si lo intentamos se nos mostrará un mensaje de error.

EJECUCIÓN DE SERVICIOS

- También debemos comprobar y definir, si es necesario, las variables de entorno que puede tener el contenedor. Se describen en la página de las imágenes en DockerHub y para usarlas tenemos que usar el flag *-e NOMBRE_VARIABLE=VALOR*
  Ejemplo:

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
```

ASIGNANDO NOMBRES A LOS CONTENEDORES

Usaremos el flag *--name* con *docker run*, para poder elegir nombres que tenga relación con la función que va a desempeñar dicho contenedor. Ejemplo:

```
# 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
```

EJECUTANDO ÓRDENES EN LOS CONTENEDORES

Para realizar operaciones como:

- Instalar paquetes.
- Modificar o ver el contenido de ciertos ficheros.
- Habilitar ciertos módulos de servicios
- etc...

Lo haremos con *docker exec* (el contenedor debe estar en ejecución)

EJECUTANDO ÓRDENES EN LOS CONTENEDORES

docker exec [opciones] nombre_contenedor orden [argumentos]

Algunas de las opciones más importantes son:

- *-it (-i y -t juntos)* si queremos interactividad con el contenedor ejecutando un shell (/bin/bash normamente).
- *-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.

EJECUTANDO ÓRDENES EN LOS CONTENEDORES

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.

> 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"

EJECUTANDO ÓRDENES EN LOS CONTENEDORES

NOTA: LOS CONTENEDORES VIENEN CON SOLO LO IMPRESCINDIBLE INSTALADO. SI QUIERO INSTALAR ALGO DEBO NORMALMENTE HACER ANTES UN *APT UPDATE* (ya que la mayoría son basados en Debian).

Con la orden *docker cp* puedo mover ficheros desde mi sistema al contenedor y desde el contenedor a mi sistema

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER PS

La orden *docker ps* nos va a servir para obtener información de los contenedores ya arrancados:

- El estado del contenedor (Parado EXITED o Funcionado UP).
- La imagen de la que deriva el contenedor.
- El tamaño actual del contenedor.
- La orden que ejecuta el contenedor al arrancar(ENTRYPOINT).
- El nombre del contenedor.
- Cuándo fue creado el contenedor.
- Las redirecciones de puertos, en caso de haberlas.

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER PS. EJEMPLOS

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

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER PS. EJEMPLOS

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

> docker ps -l

Filtrar 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

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER INSPECT

La orden *docker inspect* nos muestra información en formato JSON y nos da datos sobre aspectos como:

- El id del contenedor.
- Los puertos abiertos y sus redirecciones
- Los bind mounts y volúmenes usados.
- El tamaño del contenedor.
- La configuración de red del contenedor.
- El ENTRYPOINT (que es lo que se ejecuta al hacer docker run).
- El valor de las variables de entorno.
- Y muchas más cosas....

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER INSPECT

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

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER LOGS

Con *docker logs* podemos obtener información sobre lo que está pasando en el contenedor.

Me va a servir tanto para contenedores que estén parados como para contenedores en ejecución. Ejemplos:

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

OBTENIENDO INFORMACIÓN DE LOS CONTENEDORES. DOCKER LOGS

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 CONTENEDORES.

Necesitaremos realizar operaciones como las siguientes:

- Parar un contenedor que no estamos necesitando o que, puede ser, esté ejecutando un servicio que ocupe un puerto que queremos ocupar con otro servicio o contenedor.
- Eliminar un contenedor que instalamos y que ya no necesitamos. Puede ser que ya ni nos acordemos del motivo por el cual teníamos "eso" en nuestro sistema.
- Queremos iniciar un contenedor que estaba parado pero que vamos a volver a necesitar.
- Queremos reiniciar un contenedor para que nuevas opciones de configuración sean aplicadas.

GESTIÓN DE CONTENEDORES.

Para operaciones de ese tipo tenemos las siguientes órdenes docker:

- *docker stop* para detener el contenedor, ya sea por nombre o por ID.
- *docker rm* para borrar el contenedor, ya sea por nombre o por ID.
- *docker start* para iniciar un contenedor que estaba parado previamente, ya sea por nombre o por ID.
- *docker restart* para reiniciar un contenedor que previamente ya estaba en ejecución.

GESTIÓN DE CONTENEDORES.

Parar un contenedor en ejecución que se llame servidorWeb

> docker stop servidorWeb

Parar un contenedor en ejecución cuyo ID es ea9b922190d8 pero esperando 10 segundos (-t o --time)

> docker stop -t 10 ea9b922190d8

Borrar un contenedor que se llama servidorBD

> docker rm servidorBD

Borrado un contenedor que se llame jenkins aunque esté en ejecución (--force o -f)

> docker rm -f Jenkins

GESTIÓN DE CONTENEDORES.

Inicio de un contenedor con nombre jenkins

> docker start jenkins

Inicio de un contenedor con nombre jenkins pero haciendo el attach de la entrada estándar para poder interactuar con él (-i o --interactive)

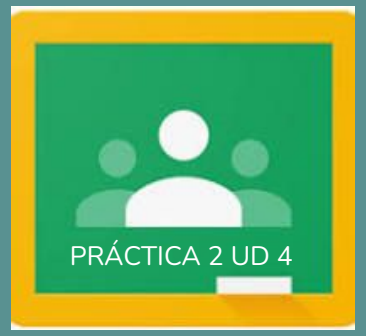
> docker start -i jenkins

Reinicio de un contenedor con ID ea9b922190d8

> docker restart ea9b922190d8

Reinicio de un contenedor con ID ea9b922190d8 pero esperando 10 segundos (-t o --time)

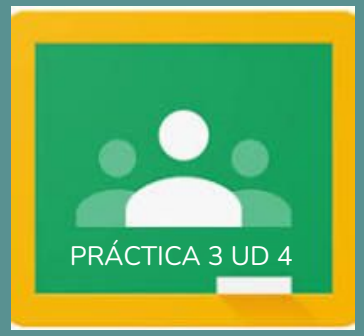
> docker restart -t 10 ea9b922190d8



TAREA 2. DESCARGA DE IMÁGENES

Descargar las siguientes imágenes y, una vez descargadas, mostrarlas con la orden de Docker cli correspondiente:

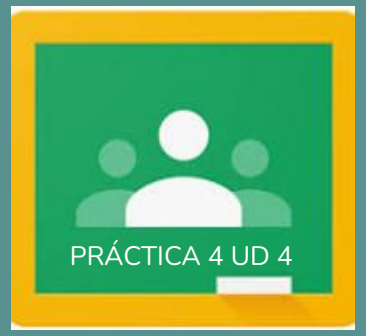
- ubuntu:18.04
- centos:8
- debian:9
- mariadb:latest
- mysql:5.7
- httpd
- tomcat:9.0.39-jdk11
- jenkins/jenkins:lts
- php:7.3-apache



TAREA 3. ARRANCAR UN CONTENEDOR

En esta tarea vas a arrancar un contenedor al que llamaremos ubuntu, de la imagen ubuntu:18.04. Una vez arrancado realizar las siguientes operaciones:

- Salir del contenedor y comprobar que ese contenedor se ha parado.
- Rearrancar el contenedor desde tu equipo y comprobar que está funcionando.
- Sin entrar en el contenedor, mostrar por pantalla el fichero /etc/os-release.



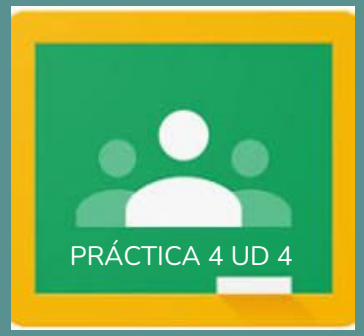
TAREA 4. EJECUCIÓN DE SERVICIOS DESDE CONTENEDORES

En esta tarea vas a ejecutar varios servicios web y de bases de datos sobre contenedores.

- Arranca un contenedor que ejecute una instancia de la imagen php:7.3-apache, que se llame web y que sea accesible desde tu equipo en el puerto 8181.
- Colocar en el directorio raíz del servicio web de dicho contenedor un fichero llamado index.html con el siguiente contenido:

  <h1>HOLA SOY XXXXXXXXXXXXXXX</h1>

  NOTA: Deberás sustituir XXXXXXXXXX por tu nombre y tus apellidos.

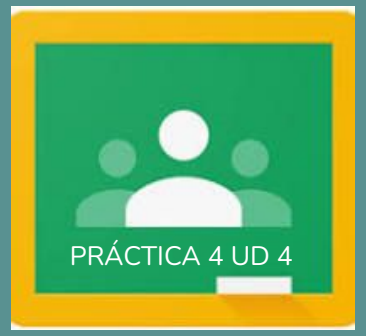


TAREA 4. EJECUCIÓN DE SERVICIOS DESDE CONTENEDORES

- Colocar en ese mismo directorio raíz un archivo llamado index.php con el siguiente contenido:

  <?php phpinfo(); ?>

- Arrancar un contenedor que se llame bbdd y que ejecute una instancia de la imagen mariadb para que sea accesible desde el puerto 3336.



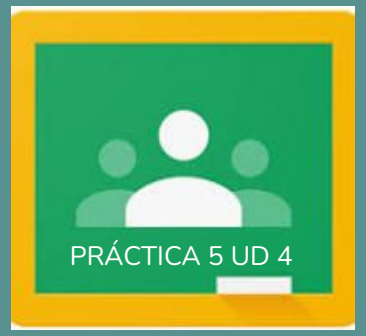
TAREA 4. EJECUCIÓN DE SERVICIOS DESDE CONTENEDORES

- Antes de arrancarlo visitar la página del contenedor en Docker Hub (https://hub.docker.com/_/mariadb) y establecer las variables de entorno necesarias para que:

  - La contraseña de root sea root.

  - Crear una base de datos automáticamente al arrancar que se llame prueba.

  - Crear el usuario invitado con la contraseña invitado.



TAREA 5. OBTENIENDO INFORMACIÓN DE CONTENEDORES

Partiendo de los contenedores en ejecución de la tarea anterior y ejecutando la orden de docker cli adecuada obtener la siguiente información:

- Dirección IP del contenedor web.
- Redirección de puertos del contenedor web.
- Dirección IP del contenedor bbdd.
- Redirección de puertos del contenedor bbdd.

DOCKER

DESCARGA DE IMÁGENES

Opción 1 :al hacer *docker run* indicando, para la ejecución del contenedor, una imagen base que no hayamos descargado previamente. En ese caso, se descargará la imagen y posteriormente empezará a ejecutarse el contenedor si todos los parámetros están bien. Ejemplo:

Supondremos que es la PRIMERA VEZ que vamos a usar esa imagen y no la hemos descargado

> docker run -it -d --name mysql8 -p 3306:3306 mysql:8.0.22

DESCARGA DE IMÁGENES

Opción 2: usando el comando *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). Esta es recomendable:

- Porque me permite actualizar una determina pareja imagen:versión a su última actualización. Sólo tendré que hacer docker pull con la misma imagen:versión

- Porque tiene otras opciones que son útiles a nivel de usuario, como la de no mostrar toda la información de las capas (opción "-q" o "--quiet").

MOSTRAR IMÁGENES DESCARGADAS

Para ello usaremos la orden *docker images*. La información que se nos muestra es:

- REPOSITORY: Nombre de la imagen en el repositorio. Por ejemplo: mysql.
- TAG: Versión de la imagen que hemos descargado.
- IMAGE ID: Un identificador que es único para cada imagen. Siempre podemos usar este ID en vez del nombre.
- CREATED: Hace cuánto se creó la imagen.
- SIZE: Tamaño de la imagen.

BORRADO DE IMÁGENES DESCARGADAS

Podemos hacerlo con:

- *Docker rmi*→ docker rmi mysql:8.0.22
- *Docker image rm* → docker image rm mysq:8.0.22

No podemos borrar una imagen que use un contenedor. Para forzar ese borrado: *docker rmi -f <nombre o id>*

Para borrar todas las imágenes que no están siendo usadas por contenedores: *docker image prune -a*

OBTENIENDO INFORMACIÓN DE IMÁGENES

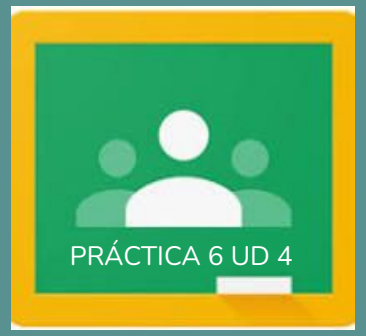
A través de las órdenes *docker image inspect / docker inspect.* Obtendremos:

- El id y el checksum de la imagen.
- Los puertos abiertos.
- La arquitectura y el sistema operativo de la imagen.
- El tamaño de la imagen.
- Los volúmenes.
- El ENTRYPOINT que es lo que se ejecuta al hacer docker run.
- Las capas.

NOTA: También podemos obtener información en la página de la imagen en DockerHub.

MÁS COMANDOS RELACIONADOS CON IMÁGENES

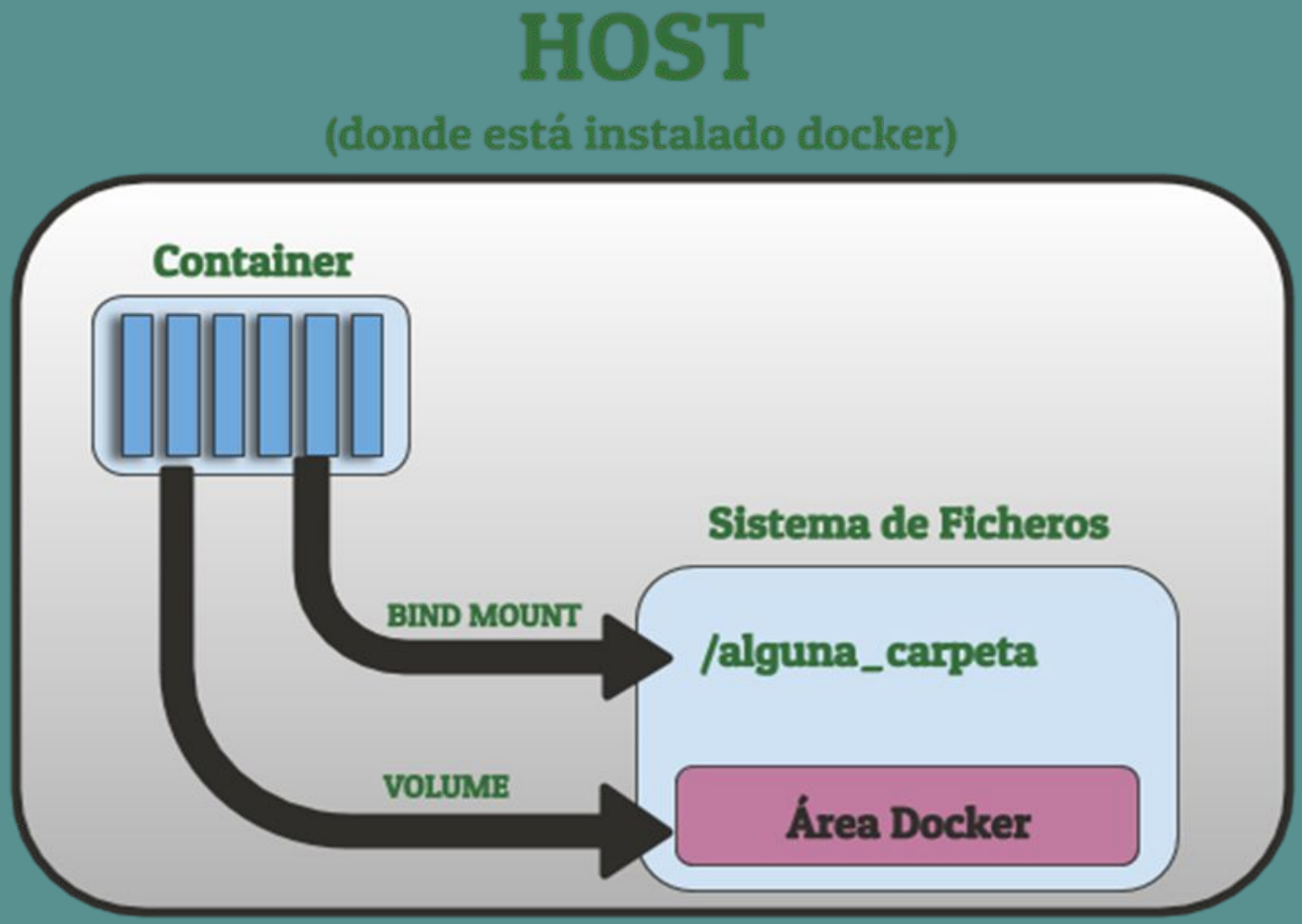
- *docker image history* para que se nos muestre por pantalla la evolución de esa imagen.
- *docker image save / docker image load* (o docker save / docker load) para guardar imágenes en fichero y cargarlas desde fichero (con docker load -i nombre-del-archivo.tar).
- *docker image tag (docker tag)* para añadir TAGs (versiones) a las distintas imágenes.



TAREA 6. GESTIÓN DE IMÁGENES

- Descargar la imagen Ubuntu:20.04 desde Docker Hub.
- Obtener toda la información de esa imagen y volcarla el fichero info.txt.
- Instanciar esa imagen creando un contenedor llamado modulo3. Usar el comando docker adecuado para comprobar que efectivamente ese contenedor se ha creado (en ejecución o no).
- Intentar borrar la imagen Ubuntu:20.04. ¿Has podido borrar la imagen? Responde razonadamente.
- Realizar las operaciones necesarias para poder borrar la imagen.

PERSISTENCIA DE DATOS EN DOCKER



OPCIÓN 1 - VOLÚMENES DOCKER

Los datos de los contenedores que nosotros decidamos se almacenan en una parte del sistema de ficheros que es gestionada por docker y a la que, debido a sus permisos, sólo docker tendrá acceso. 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 sino en un proveedor cloud.

OPCIÓN 1 - VOLÚMENES DOCKER

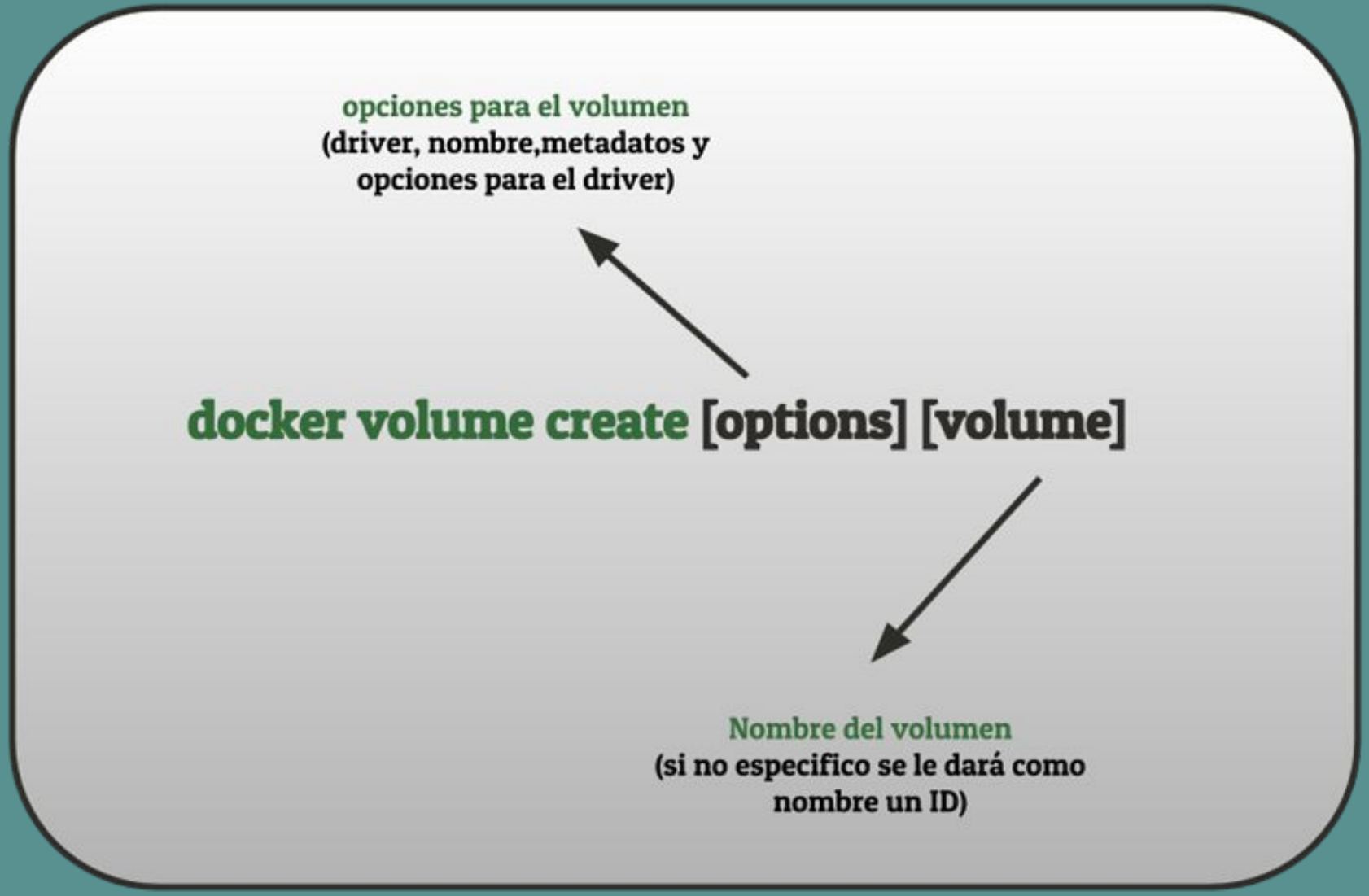
De manera general podemos decir que esa zona reservada es:

- /var/lib/docker/volumes en las distribuciones de Linux si lo hemos instalado desde paquetes estándar.
- /var/snap/docker/common/var-lib-docker/volumes en Linux si hemos instalado docker mediante snap.
- C:\ProgramData\docker\volumes en las instalaciones de Windows.
- /var/lib/docker/volumes también en Mac aunque se requiere que haya una conexión previa a la máquina virtual que se crea.

OPCIÓN 2 - BIND MOUNTS

- Sirve para "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.
- Me va a permitir:
  - Compartir ficheros entre el host y los containers.
  - Que otras aplicaciones que no sean docker tengan acceso a esos ficheros.
- Es la opción aconsejable en fase de DESARROLLO.

CREACIÓN DE VOLÚMENES



CREACIÓN DE VOLÚMENES. OPCIONES

- --driver o -d para especificar el driver elegido para el volumen. Si no especificamos nada el driver utilizado es el local que es el que nos interesa desde el punto de vista de desarrollo porque desarrollamos en nuestra máquina. En el caso de Linux el driver local es overlay2, pero existen otras posibilidades como aufs, btrfs, zfs, devicemapper o vfs.
- --label para especificar los metadatos del volumen mediante parejas clave-valor.

CREACIÓN DE VOLÚMENES. OPCIONES

- --opt o -o para especificar opciones relativas al driver elegido. Si son opciones relativas al sistema de ficheros puedo usar una sintaxis similar a las opciones de la orden mount.
- --name para especificar un nombre para el volumen. Es una alternativa a especificarlo al final que es la forma que está descrita en la imagen superior.

CREACIÓN DE VOLÚMENES. EJEMPLO

Creación de un volumen llamado data

> docker volume create data

Creación de un volumen llamando web añadiendo varios metadatos

> docker volume create --label servicio=http --label server=apache Web

ELIMINACIÓN DE VOLÚMENES

Para la eliminación de los volúmenes creados tenemos dos opciones:

- docker volume rm: para eliminar un volumen en concreto (por nombre o por id).
- docker volume prune: para eliminar los volúmenes que no están siendo usados por ningún contenedor

ELIMINACIÓN DE VOLÚMENES. EJEMPLOS

Borrar un volumen por nombre

> docker volume rm nombre_volumen

Borrar dos volúmenes de una sola vez

> docker volume rm nombre_volumen1 nombre_volumen2

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

INFORMACIÓN DE VOLÚMENES.

Podemos hacerlo de dos formas:

- Usando *docker volume ls,* que nos proporciona una lista de los volúmenes creados y algo de información adicional.
- Usando d*ocker volume inspect*, que nos dará una información mucho más detallada del volumen que hayamos elegido.

ASOCIANDO ALMACENAMIENTO A CONTENEDORES

Tener en cuenta:

- Se sobreescribirá la carpeta destino en el sistema de ficheros del contenedor, en caso de que exista.
- Si nuestra carpeta origen no existe y hacemos un bind mount, tendremos en el contenedor una carpeta vacía.
- Si usamos imágenes de DockerHub: leer la información de cada imagen en su página para ver cómo persistir los datos de esa imagen y cuáles son las carpetas importantes.

ALMACENAMIENTO Y CONTENEDORES. EJEMPLOS

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

ALMACENAMIENTO Y CONTENEDORES. EJEMPLOS

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

ALMACENAMIENTO Y CONTENEDORES. EJEMPLOS

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

ALMACENAMIENTO Y CONTENEDORES. EJEMPLOS

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

ALMACENAMIENTO Y CONTENEDORES. DOCKER INSPECT

En la salida de la orden docker inspect sobre dicho contenedor:

VOLUMEN EN USO

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "Web",
        "Source": "/var/lib/docker/volumes/Web/_data",
        "Destination": "/usr/local/apache2/htdocs",
        "Driver": "local",
        "Mode": "z",
        "RW": true,
        "Propagation": ""
    }
],
```

BIND MOUNT EN USO

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/pekechis/web",
        "Destination": "/usr/local/apache2/htdocs",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

USO DE LOS VOLÚMENES Y BIND MOUNTS

- Hacer copias de seguridad de contenidos:

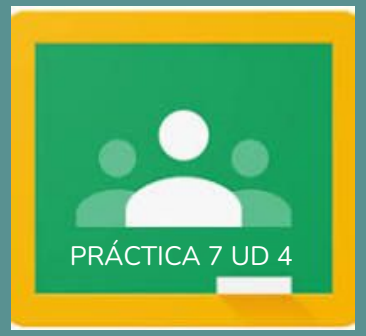
https://youtu.be/rV9mEsPQJW0?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgl

- Compartir contenidos entre contenedores:

https://youtu.be/jlYQZIbSeng?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgl

- Probar una nueva versión de contenedor para comprobar si la actualización de nuestro sistema puede dar problemas:

https://youtu.be/qdURCnir3dY?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgl



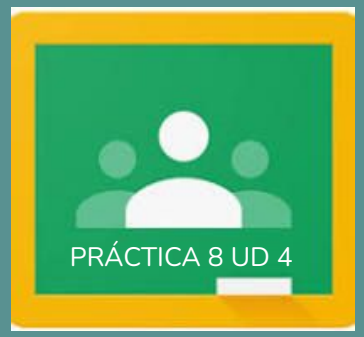
TAREA 7. VOLÚMENES

1.- Crear los siguientes volúmenes con la orden docker volume: volumen_datos y volumen_web

2.- Una vez creados estos contenedores:

- Arrancar un contenedor llamado c1 sobre la imagen php:7.4-apache que monte el volumen_web en la ruta /var/www/html
- Arrancar un contenedor llamado c2 sobre la imagen mariadb que monte el volumen_datos en la ruta /var/lib/mysql y cuya contraseña de root sea admin.

3.- Parar y borrar el contenedor c2 y tras ello borrar el volumen volumen_datos.



TAREA 8. BIND MOUNTS

1.- Crea una carpeta llamada saludo y dentro de ella crea un fichero llamado index.html con el siguiente contenido:

<h1>HOLA SOY XXXXXX</h1>

Deberás sustituir ese XXXXX por tu nombre.

2.- Una vez hecho esto arrancar dos contenedores basados en la imagen php:7.4-apache que hagan un bind mount de la carpeta saludo en la carpeta /var/www/html del contenedor.

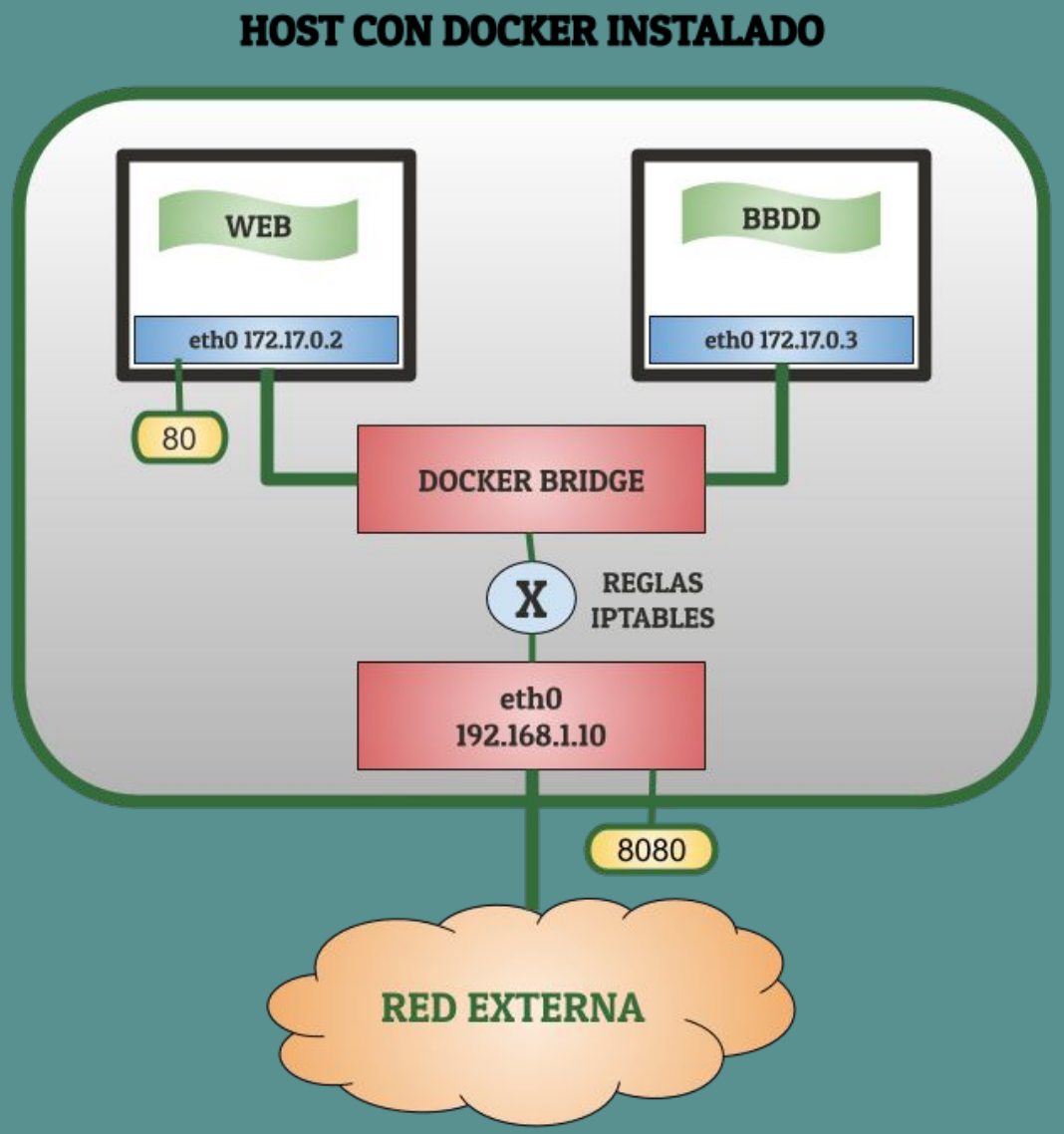
3.- Uno de ellos deberá redireccionar su puerto 80 al 8181 y el otro al 8282. Y su nombres serán c1 y c2.

REDES. DRIVERS DE TIPO BRIDGE

Los drivers de tipo BRIDGE me van a permitir, siempre dentro de 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.

REDES. DRIVERS DE TIPO BRIDGE



Aplicación típica compuesta por dos servicios, un servidor web y un servidor de base de datos, cada uno de ellos en un contenedor diferente.
Mi servidor web es accesible al exterior en el puerto 8080.

GESTIONANDO REDES

- Para mostrar las redes docker creadas:
  docker network ls

- Para crear redes:

  docker network create

- Para borrar redes:

  docker network rm / docker network prune

CREANDO REDES

Ejemplos de creación de redes:

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

ELIMINANDO REDES

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 tengan contenedores asociados sin preguntar confirmación (-f o --force)

> docker netowkr 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

INFORMACIÓN SOBRE REDES

- Mediante la orden docker network ls, que presentamos en el apartado anterior y que además tiene diversas opciones algunas de las cuales veremos posteriormente.

- Mediante la orden docker network inspect, que nos mostrará una información mucho más detallada con todas las características de la red

ASOCIANDO REDES A CONTENEDORES

Para realizar esta "conexión" debemos de tener en cuenta los siguientes aspectos:

- Al arrancar un contenedor podemos especificar a qué red está conectado inicialmente usando el flag --network seguido del nombre de la red a la que queremos conectarlo.

- Si al arrancar un contenedor no especificamos una red, el contenedor se conectará a la red por defecto, la red "bridge" que usa el driver "bridge".

ASOCIANDO REDES A CONTENEDORES

- Al arrancar un contenedor no puedo "conectarlo" inicialmente a más de una red.

- Tras crear el contenedor puedo conectarlo a más redes o desconectarlo de alguna red. Dependiendo de si he elegido la red por defecto o no, podré o no podré hacer esa conexión o desconexión en caliente (con el contenedor funcionando).

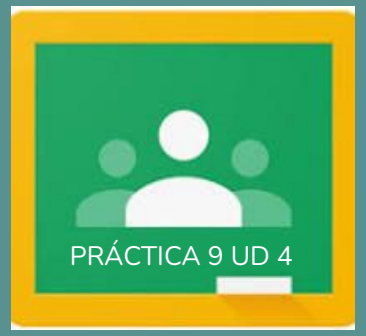
ASOCIANDO REDES A CONTENEDORES. EJEMPLOS

Arrancar 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

Arrancar 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



TAREA 9. REDES

En esta actividad vamos a crear dos redes de ese tipo (BRIDGE) con los siguientes datos:

Red1
Nombre: red1
Dirección de red: 172.28.0.0
Máscara de red: 255.255.0.0
Gateway: 172.28.0.1

Red2
Nombre: red2
Es resto de los datos será proporcionados automáticamente por Docker.

TAREA 10. REDES

Deberemos realizar los siguiente pasos:

- Poner en ejecución un contenedor de la imagen ubuntu:20.04 que tenga como hostname host1, como IP 172.28.0.10 y que esté conectado a la red1. Lo llamaremos u1.
- Entrar en ese contenedor e instalar la aplicación ping (apt update && apt install inetutils-ping).
- Poner en ejecución un contenedor de la imagen ubuntu:20.04 que tenga como hostname host2 y que esté conectado a la red2. En este caso será docker el que le de una IP correspondiente a esa red. Lo llamaremos u2.
- Entrar en ese contenedor e instalar la aplicación ping (apt update && apt install inetutils-ping).

DOCKER

PERSONALIZAR IMÁGENES

Vamos a realizar la personalización de las imágenes para que se adecúen a nuestras necesidades. Lo que buscamos es distribuir nuestras imágenes para que puedan ser usadas sin problemas en cualquier sistema en el que docker esté instalado.

Lo haremos:

- 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.

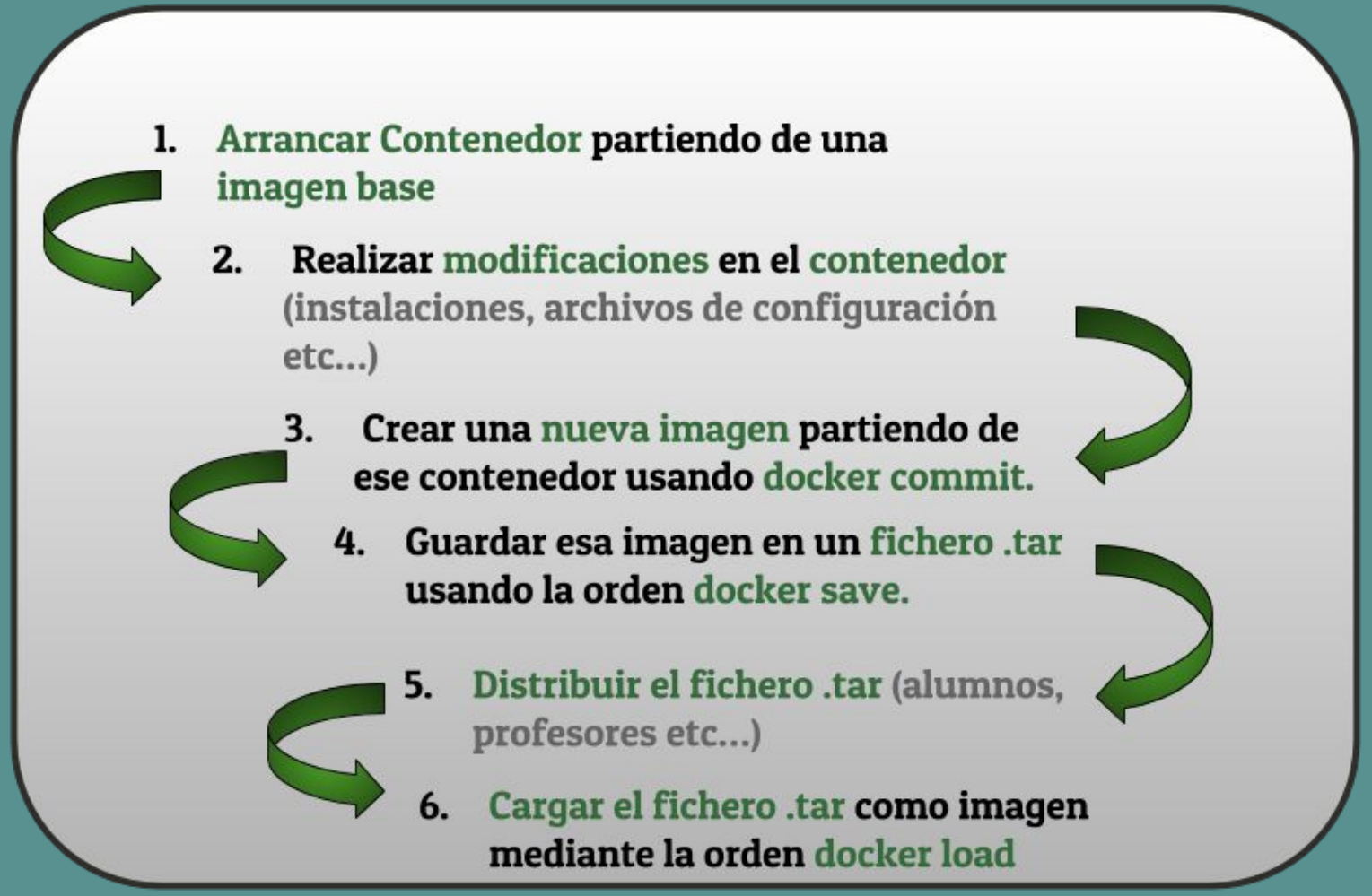
IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN

Dos opciones:

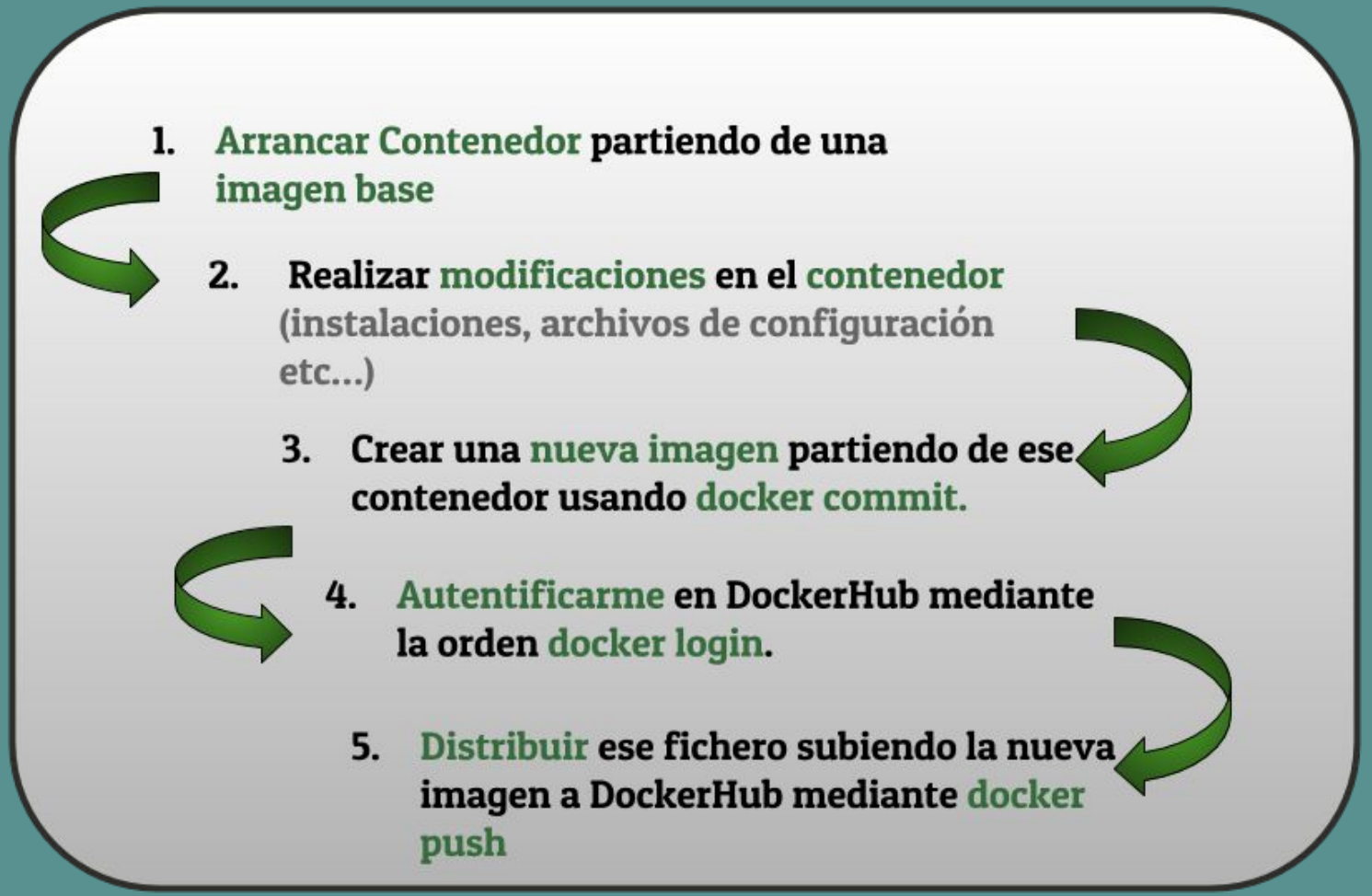
- Utilizar la secuencia de órdenes docker commit / docker save / docker load. En este caso la distribución se producirá a partir de un fichero.
- Utilizar la pareja de órdenes docker commit / docker push. En este caso la distribución se producirá a través de DockerHub.

IMPORTANTE: AL HACER COMMIT DEBEMOS AÑADIR EL NOMBRE DE NUESTRO USUARIO DE DOCKERHUB SI QUEREMOS SUBIRLO.

IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN (A PARTIR DE UN FICHERO)



IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN (A TRAVÉS DE DOCKERHUB)



IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN. EJEMPLOS

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

IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN. EJEMPLOS

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

IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN. EJEMPLOS

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

IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN. EJEMPLOS

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

IMÁGENES DESDE UN CONTENEDOR EN EJECUCIÓN. EJEMPLOS

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

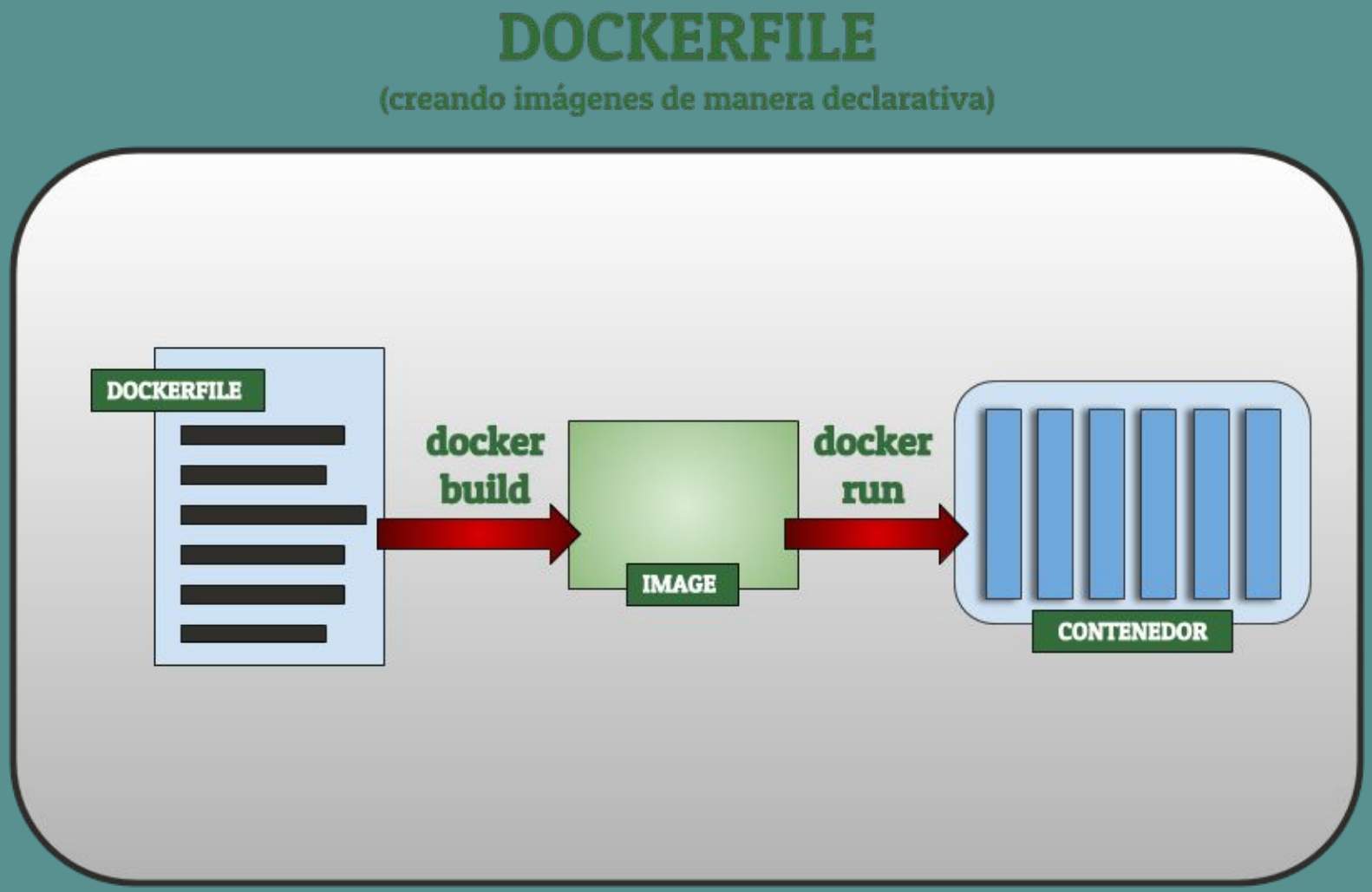
Subir a DockerHub todas las versiones (tags) de la imagen ubuntu20netutils (--all-tags / -a)

> docker push -a usuarioDockerHub/ubuntu20netutils

Vídeo resumen:

https://youtu.be/eWkqN9U5yJU?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgI

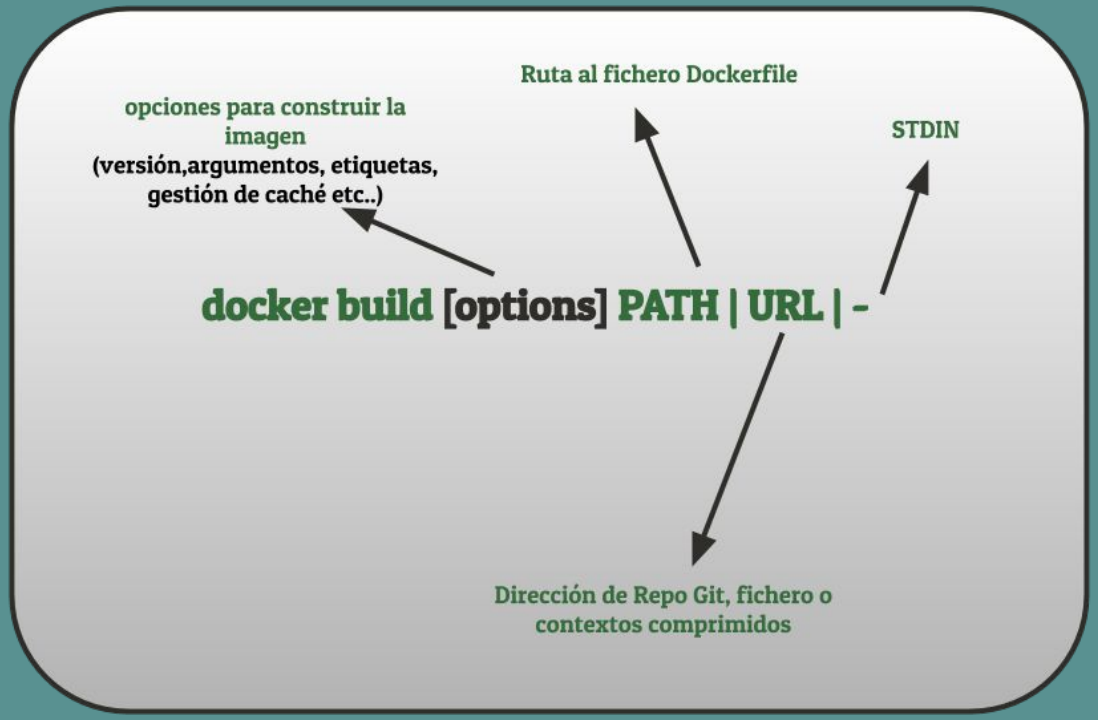
UTILIZANDO FICHEROS DOCKERFILE



UTILIZANDO FICHEROS DOCKERFILE

UTILIZANDO FICHEROS DOCKERFILE

Para construir las imágenes necesitamos un fichero Dockerfile dentro de un contexto, ya sea en mi equipo o un repositorio exterior, y la orden docker build cuya estructura general es la siguiente:



UTILIZANDO FICHEROS DOCKERFILE. EJEMPLOS

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

UTILIZANDO FICHEROS DOCKERFILE. EJEMPLOS

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)

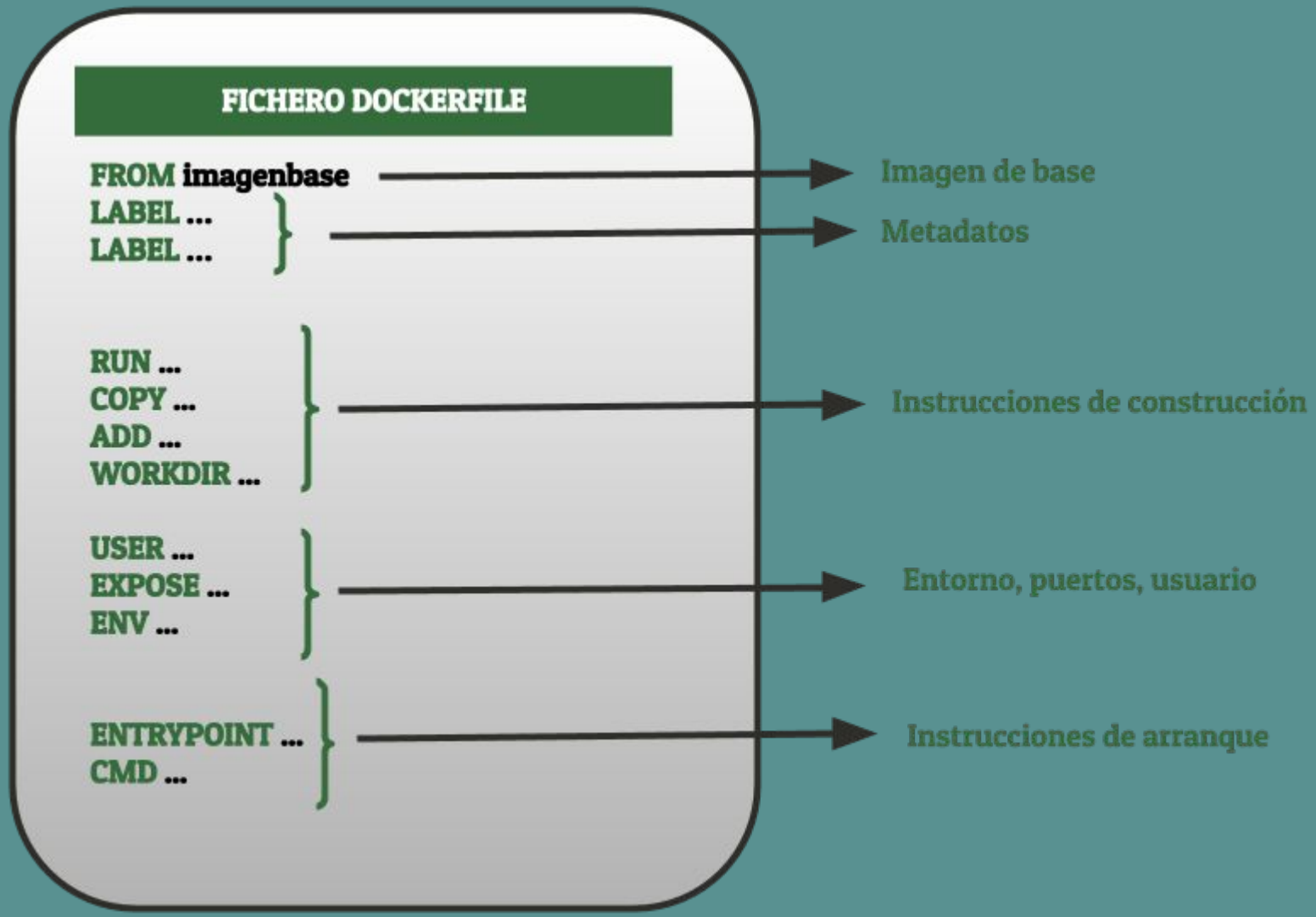
> docker build --no-cache -t usuarioDockerHub/nombre_imagen:1.0 .

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

NOTA: Si quiero que la imagen construida sea distribuida mediante DockerHub debo poner como prefijo de la imagen mi nombre de usuario de DockerHub.

UTILIZANDO FICHEROS DOCKERFILE. EJEMPLOS



ÓRDENES FICHEROS DOCKERFILE.

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

ÓRDENES FICHEROS DOCKERFILE.

- 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

ÓRDENES FICHEROS DOCKERFILE.

- 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

ÓRDENES FICHEROS DOCKERFILE.

- 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"]

ÓRDENES FICHEROS DOCKERFILE.

- 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”].

FICHERO .DOCKERIGNORE

- El funcionamiento de este tipo de ficheros es análogo al funcionamiento de los ficheros .gitignore que excluyen una serie de ficheros del control de versiones.

- LOS ARCHIVOS QUE SE RECOJAN EN EL FICHERO .DOCKERIGNORE NO PASARÁN A LA IMAGEN EN EL PROCESO DE CONSTRUCCIÓN DE LA IMAGEN.

REFERENCIA COMPLETA FICHEROS DOCKERFILE:
https://docs.docker.com/engine/reference/builder/

FICHERO .DOCKERIGNORE GENÉRICO

```
# Esa carpeta app tiene el contenido de un repositorio

#Excluyo la carpeta .git

app/.git

#Excluyo el fichero .gitignore

app/.gitignore

#Excluyo todos los archivos dentro de la carpeta log pero dejo la carpeta

app/log/*

#Excluyo todos los archivos dentro de la carpeta tmp pero dejo la carpeta.

app/tmp/*

#Excluyo el archivo README.md

app/README.md
```

FICHERO .DOCKERIGNORE PARA UNA APLICACIÓN NODE

```
# Esa carpeta nodeapp tiene el contenido de un repositorio

#Excluyo la carpeta .git

nodeapp/.git

#Excluyo el fichero .gitignore

nodeapp/.gitignore

#Excluyo la carpeta node_modules. Eso me obliga a hacer npm install al arrancar el contenedor

nodeapp/node_modules

#Excluyo el fichero creado por el editor de código

.vscode
```

EJEMPLO DE FICHEROS DOCKER

Dockerfile que hace modificaciones en un servidor de Aplicaciones Tomcat

https://gist.github.com/pekechis/438a7aecfc9ecc67cb8d2bd1988875b4

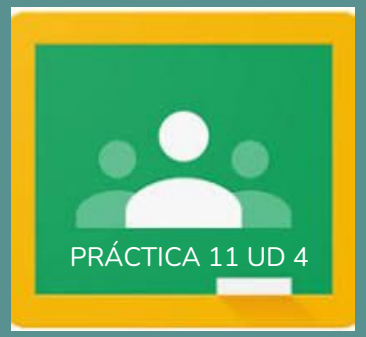
Dockerfile que descarga el código de WP actualizado y lo pone en funcionamiento en un servidor Web Apache con PHP.

https://gist.github.com/pekechis/50089bf90443bac115572a71b8ec42ac

DOCKERFILE para proyecto Django

https://gist.github.com/pekechis/d7237427bbee51a3ad1d0f3865f696fd

Vídeo: https://youtu.be/oiZORiVh3Gs?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgl



TAREA 11. CREACIÓN DE IMÁGENES

1.- Arrancar un contenedor sobre la imagen ubuntu:20.04 y sobre él realizar las siguientes operaciones:

Instalación del editor nano (apt install nano).

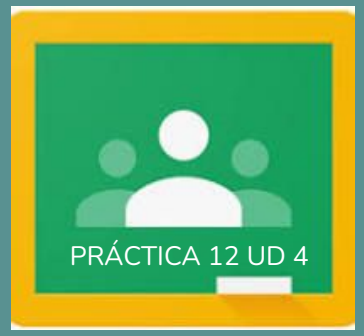
Instalación del editor vim (apt install vim).

Instalación de las herramientas de red (apt install inetutils-tools).

Instalación de las herramientas dns (apt install dnsutils).

Creación del usuario usuario con contraseña usuario (adduser usuario)

2.- Tras realizar dichas instalaciones y utilizando la orden docker commit crear una imagen que se llame de la siguiente manera: TuNombreUsuarioDockerHub/a61 y subirla a DockerHub utilizanzo la orden docker push. Recordad que antes tendréis que hacer docker login.



TAREA 12. CREACIÓN DE IMÁGENES

1.- Partiendo de la imagen php:7.4-apache construir un Dockerfile que realice lo siguiente:

- Instalar nano (apt install -y nano)
- Instalar git (apt install -y git)
- Colocar en el directorio raíz del servidor apache (/var/www/html) dos ficheros:
    - index.html que contenga HOLA SOY XXXXXX sustituyendo XXXXX por tu nombre
    - info.php que contengo el siguiente código <?php phpinfo(); ?>

2.- Una vez creado dicho Dockerfile construir la imagen, que se deberá llamar TuNombreUsarioDockerHub/a62.

ASEGURANDO CONTENEDORES

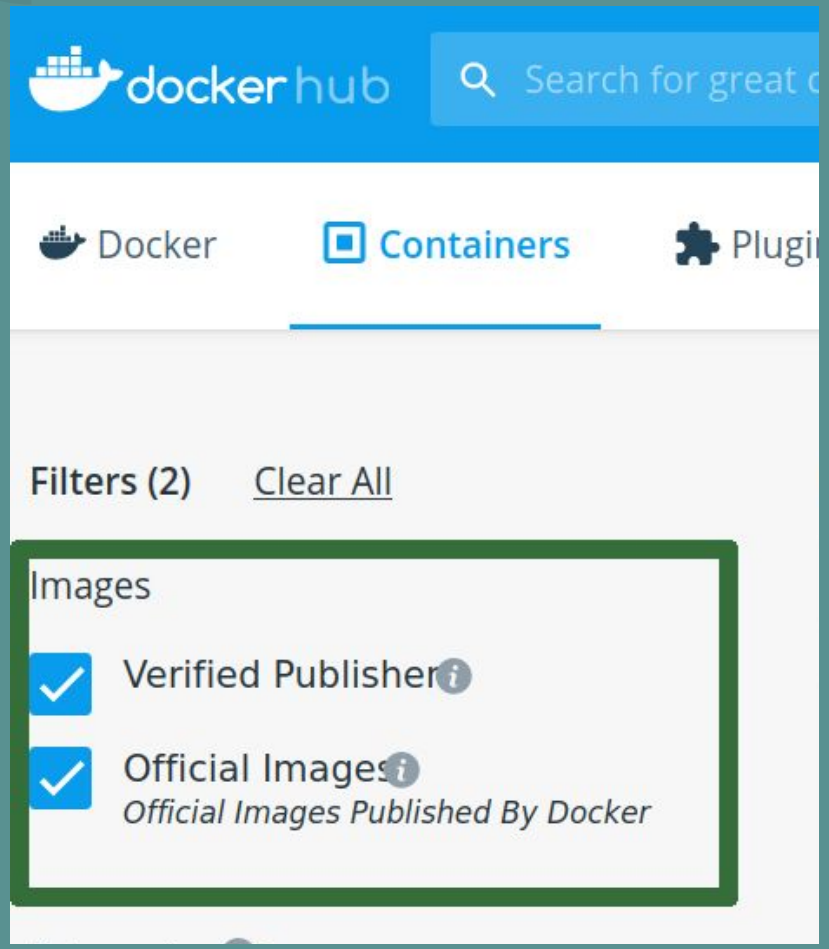
Si en algún momento queremos desplegar nuestro código en forma de contenedor en un orquestador como Kubernetes o Docker Swarm tendremos que tomar ciertas medidas. Hablaremos de la seguridad en los contenedores desde tres perspectivas:

- La seguridad en las imágenes docker.
- La seguridad en el proceso de build.
- La seguridad al arrancar los contenedores.

SEGURIDAD. IMÁGENES

- Intentar siempre usar una imagen de un usuario verificado o una imagen publicada por la propia empresa Docker.
- Verificar la integridad de la imagen que nos descargamos.
- Intentar usar imágenes que tengan lo mínimo.
- Restringir los privilegios de los datos a los que pueden acceder los contenedores.
- En caso de crear nuestras propias imágenes pagar por el servicio de DockerHub "Vulnerability Scanner" o utilizar herramientas que sirve para ese mismo propósito.

SEGURIDAD. IMÁGENES



SEGURIDAD. IMÁGENES.DCT

- Habilitar el mecanismo de firma digital llamado DCT (Docker's Content Trust) para verificar la INTEGRIDAD y la AUTORÍA de las imágenes que nos descargamos desde DockerHub.
- Solo podré hacer PULL / RUN / BUILD con imágenes "Trusted" (ni siquiera con las mías) salvo que explícitamente haga alguna excepción añadiendo el flag ---disable-content-trust a la ejecución de dichas órdenes.

SEGURIDAD. IMÁGENES.DCT

- Si quiero habilitarlo tengo que poner la variable de entorno DOCKER_CONTENT_TRUST=1.
- En linux se hace con:

```
# Añado la siguiente línea al final del fichero .bashrc que sirve para añadir esa variable de entorno

export DOCKER_CONTENT_TRUST=1

# Recargo el .bashrc

> source /home/miusuario/.bashrc
```

- En Windows, añadiendo DOCKER_CONTENT_TRUST=1 delante de la orden docker o creando la variable de entorno.

SEGURIDAD. FIRMA DE IMÁGENES

Si quiero firmar digitalmente mis imágenes tengo que seguir los siguientes pasos:

1. Generar la parejas de claves público/privada. La privada se coloca en ~/.docker/trust/private.
2. Compartir mi clave pública con el servidor Notary asociado a DockerHub. TENGO QUE GENERAR UNA CLAVE PÚBLICA PARA CADA REPOSITORIO (Colección de versiones de la misma imagen).
3. Firmar la imagen. Este proceso firma y a la vez hace un push.

SEGURIDAD. FIRMA DE IMÁGENES

Lo vemos con un ejemplo:

1. Generar la pareja de claves

> docker trust key generate miclave.pub

2. Comparto mi clave pública con el servidor Notary en el repositorio para las imágenes del repositorio usuario/nombreimagen. El nombre del firmante es miclave y miclave.pub es la clave pública generada en el proceso anterior. Recordad que un mismo repositorio puede contener varias versiones (TAGS) de una misma imagen.

> docker trust signer add --key miclave.pub miclave usuario/nombrerepositorio

3. Firmo la imagen

> docker trust sign usuario/nombreimagen[:tag]

SEGURIDAD. IMÁGENES MÍNIMAS

Dos buenas prácticas en este sentido son:

- Si estamos construyendo nuestra propia imagen usaremos una imagen de base que sea de tamaño mínimo. Un ejemplo es bitnami/minideb.

- Usar el fichero .dockerignore en nuestro proceso de build. Así evitamos que nuestra imagen tenga más tamaño del necesario.

SEGURIDAD. RESTRICCIÓN DE PRIVILEGIOS

- Si vamos a usar un volumen o un bind mount en el que no queremos que se pueda escribir desde el contenedor podemos hacerlo añadiendo la opción readonly al flag --mount.
- Pero si por el contrario quiero evitar que se escriba en una carpeta propia del contenedor tengo que usar el flag --read-only de la orden docker run. El contenedor no podrá realizar ningún tipo de escritura en su sistema de ficheros

SEGURIDAD. RESTRICCIÓN DE PRIVILEGIOS

Ejemplos:

Realizo un bind mount evitando que desde el contenedor se pueda escribir en mi carpeta:

```
>  docker run -it --mount type=bind,src=/home/pekechis/pruebaPHP,
dst=/var/www/html,readonly -p 8686:80 php:7.4-apache
```

Arranco un contenedor en modo solo lectura.

```
>  docker run -it --read-only ubuntu:20.04 /bin/bash
```

SEGURIDAD. BUILD. ADD & COPY

Cuando copiamos elementos de forma recursiva incrementamos la posibilidad de que suceda lo siguiente:

- Aumentar el tamaño de la imagen de manera innecesaria.
- Copiar en el contenedor ficheros sensibles que contengan claves, tokens de APis etc...

En ambos casos tenemos que tener un fichero .dockerignore debidamente configurado y que excluya explícitamente del proceso de copia recursiva archivos con claves como *.ENV, *.pem etc..

SEGURIDAD. BUILD. CMD & ENTRYPOINT

Los procesos que se ejecutan en un contenedor (al igual que en otros sistemas) no deben ejecutarse como root, en especial aquellos que son servicios expuestos al exterior. Para ello:

1. Crear un usuario y un grupo para ese usuario (RUN).
2. Ejecutar todas las instrucciones del Dockerfile que tengan que ser realizadas como root (RUN, COPY, ADD,WORKDIR etc....).
3. De manera previa a ejecutar el ENTRYPOINT y/o el CMD cambiar el usuario de ejecución de las órdenes al usuario creado previamente (USER).
4. Definir el ENTRYPOINT y/o CMD y que esa órdenes lance procesos pertenecientes al usuario creado.

SEGURIDAD. BUILD. CMD & ENTRYPOINT

Ejemplo:

```
# Imagen que vamos a usar

FROM XXXXX

# Creación del usuario para arrancar el servicio

RUN addgroup -S usuario && adduser -S usuario -G usuario.

# Lista de órdenes que serán de ROOT

.........

# Establezco el usuario que ejecutará las siguientes órdenes.

USER usuario

# Defino el ENTRYPOINT, se ejecutará como usuario.

ENTRYPOINT .....
```

SEGURIDAD. ARRANQUE DE CONTENEDORES

Para aumentar la seguridad al desplegar los contenedores podemos actuar en dos sentidos:

- Limitar los recursos que puede usar el contenedor. Esto será de especial utilidad para evitar ataques de DoS.
- Deshabilitar "capabilities" del contenedor que voy a arrancar.

SEGURIDAD. ARRANQUE DE CONTENEDORES

Para limitar recursos hay un serie de flags de docker run que podemos usar, entre ellos los más destacados son:

- --memory/-m que establece el límite de memoria que puede llegar a usar un contenedor.
- --memory-reservation que es similar al anterior pero es un límite blando. Si se sobrepasa docker intentará reducir la memoria consumida por el contenedor.
- --cpus que limita el número de cpus del sistema que un contenedor puede utilizar.

SEGURIDAD. ARRANQUE DE CONTENEDORES

Para limitar recursos:

- --memory/-m que establece el límite de memoria que puede llegar a usar un contenedor.
- --memory-reservation que es similar al anterior pero es un límite blando. Si se sobrepasa docker intentará reducir la memoria consumida por el contenedor.
- --cpus que limita el número de cpus del sistema que un contenedor puede utilizar.

Ejemplo:

Arranco un contenedor con un límite de 4GB y 4cpus

> docker runt -it -m 4Gb --cpus=4 httpd .....

SEGURIDAD. ARRANQUE DE CONTENEDORES

Para limitar capacidades: con el flag --cap-drop. Algunas capacidades en Linux son:

- AUDIT_WRITE: Escribe mensajes en los log de auditoría del Kernel.
- CHOWN: Para permitir cambios en el UID y GUID de los ficheros.
- NET_BIND_SERVICE: Permite asociar un socket a un puerto que puede ser accedido desde Internet.
- NET_ADMIN: Configuración de interfaces, masquerading, enrutamiento, iptables etc...
- SETUID: Manipulación del UID de los procesos etc...
- SETGUID: Manipulación del GUID de los procesos etc..
- FOWNER: Para manipulación de todo tipo de permisos y ACLs en ficheros.

SEGURIDAD. VÍDEO RESUMEN

https://youtu.be/F1jbW7ytYBE?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgI