
docker

Docker = estibador

Persona que tiene por oficio cargar y descargar las mercancías de las embarcaciones y se ocupa de la adecuada distribución de los pesos.

DOCKER ENGINE es el **software** que se encargará de la **GESTIÓN DE CONTENEDORES** (cargar y descargar contenedores).

CONTAINER = contenedor

Recipiente de carga apilable para el transporte de mercancías.

En nuestro caso contendrán un sistema operativo, una aplicación, librerías, carpetas, etc.

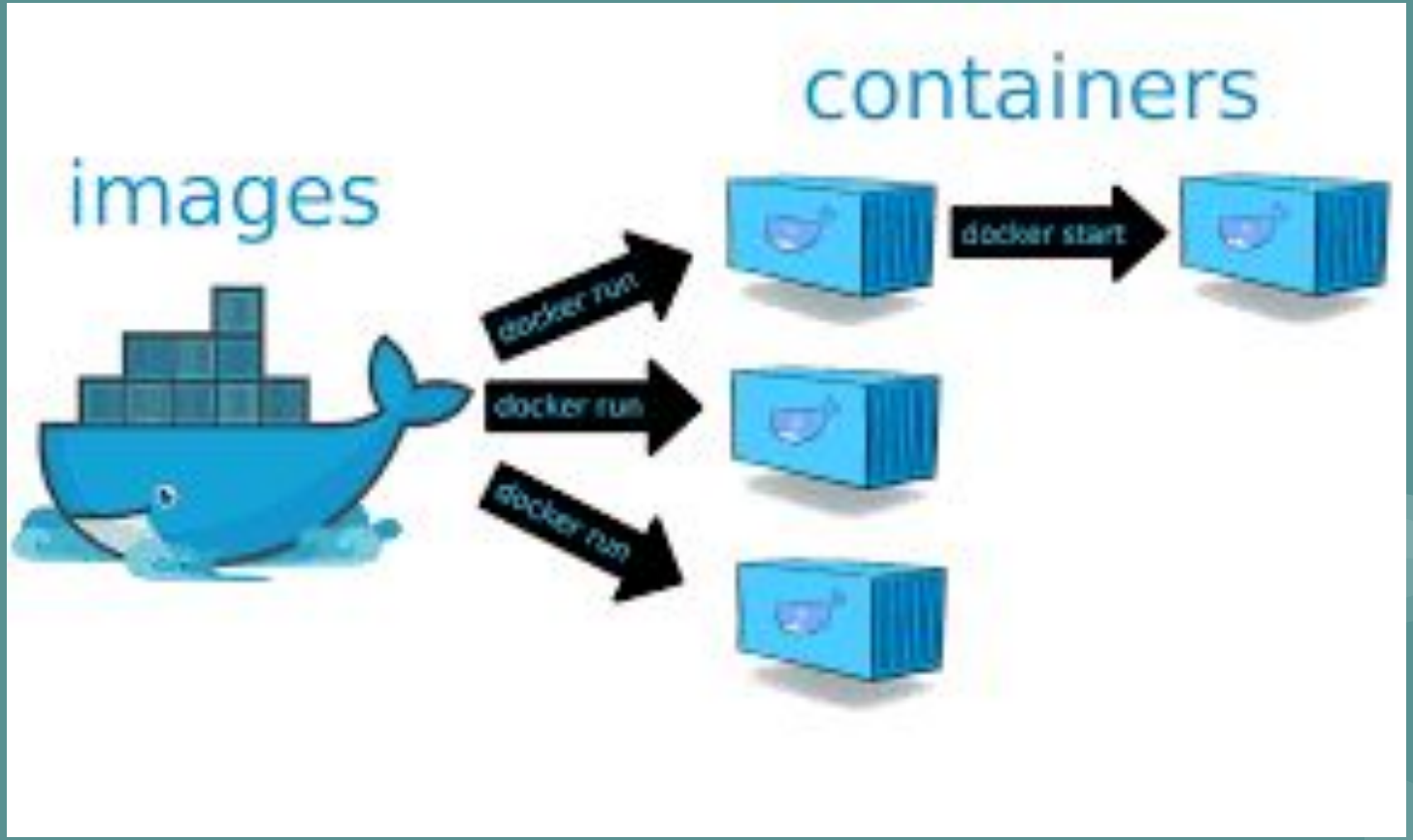
Los **CONTENEDORES** se inicializan cuando corremos **RUN** sobre una **IMAGEN**.

La **IMAGEN** es la base para crear un contenedor.

Son esencialmente una instantánea de un contenedor (como se ve el contenedor en un momento específico). Las imágenes se crean con el comando **BUILD**,

Una vez creada una imagen, se pueden almacenar en el **Docker Hub.**

Hay muchas imagenes publicas diponibles y podemos crear nuestras propias imagenes privadas.

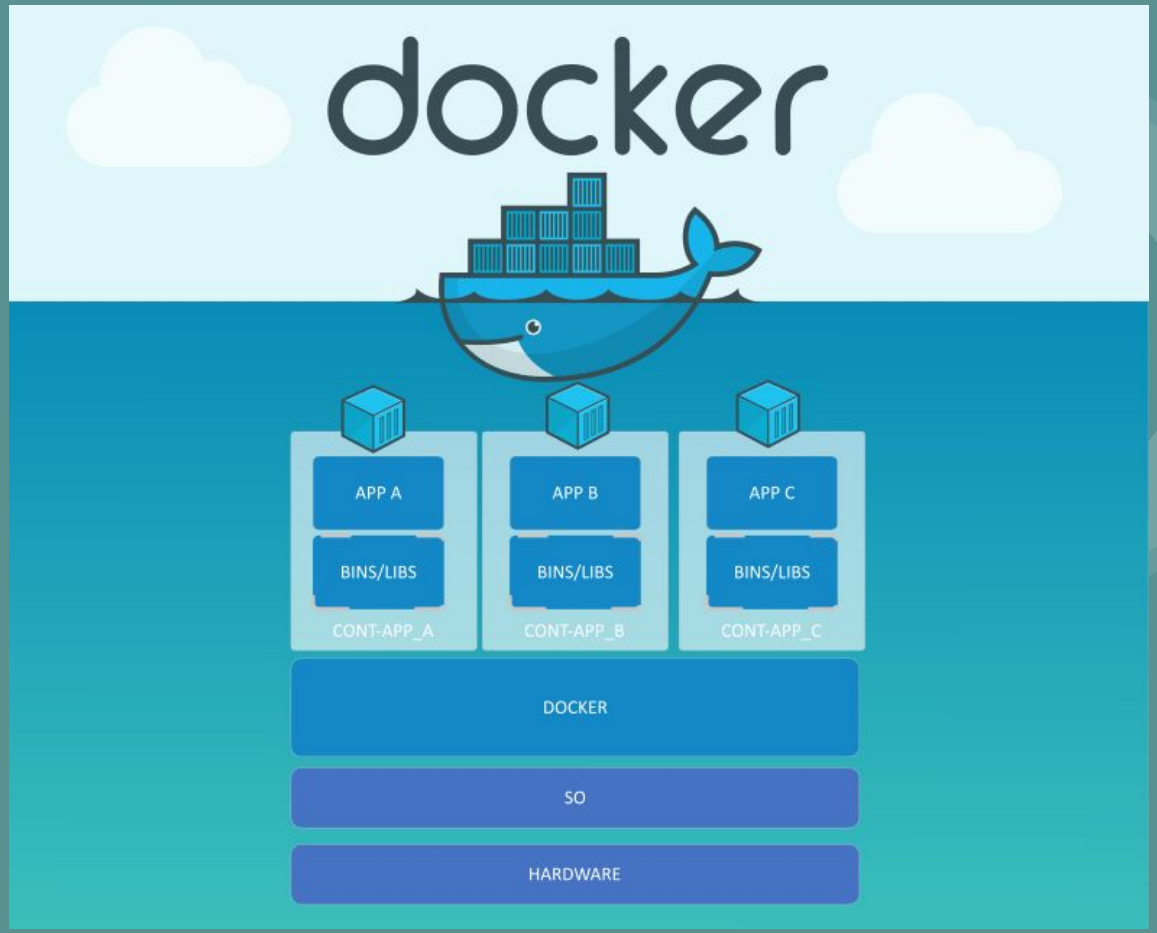

containers
images
docker run
docker run
docker run
docker start

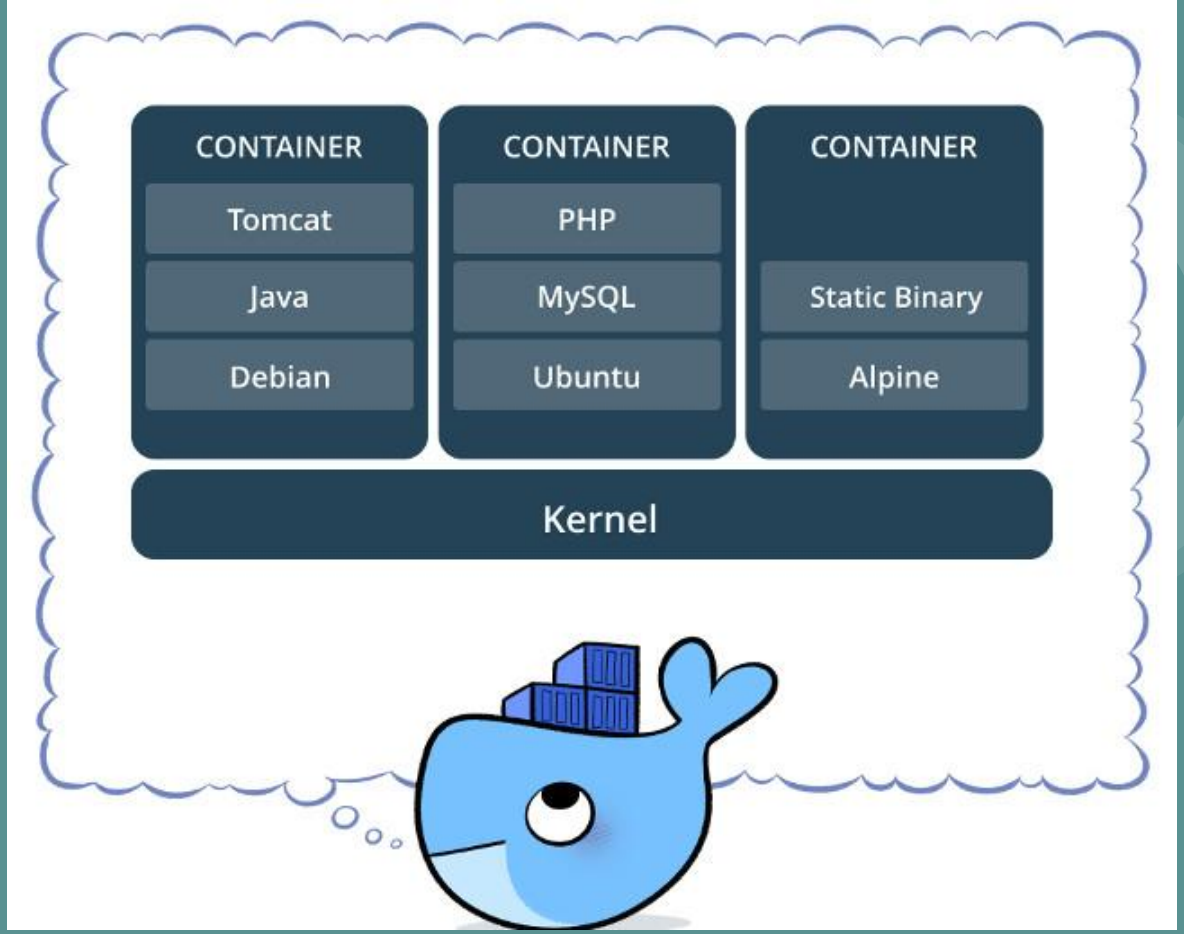
HOST = ANFITRION

Es la MAQUINA FISICA o MAQUINA VIRTUAL que corre en segundo plano el servicio DOCKER DEAMON.

El HOST puede ser tu computadora, un servidor en la nube, una máquina virtual en un hosting, etc.
Sobre ese HOST se corren los CONTENEDORES.
Un CONTENEDOR puede tener un sistema operativo (SO) distinto al HOST, aplicaciones, librerías,etc.

Son como una MAQUINA INDEPEDIENTE del HOST pero que corren sobre el.
A diferencia de las maquinas virtuales, NO recrean un sistema operativo completo y NO virtualiza hardware.
En Docker lo que se hace es usar las funcionalidades del Kernel para encapsular un sistema, de esta forma el proyecto que corre dentro de él, no tendrá conocimiento de que está en un contenedor.


docker
APP A
BINS/LIBS
CONT-APP_A
APP B
BINS/LIBS
CONT-APP_B
APP C
BINS/LIBS
CONT-APP_C
DOCKER
SO
HARDWARE


CONTAINER
Tomcat
Java
Debian
CONTAINER
PHP
MySQL
Ubuntu
CONTAINER
Static Binary
Alpine
Kernel

Ejemplo

En desarrollo, supongamos que le doy mantenimiento a dos sistemas, uno con PHP 5 y uno con PHP 7.
Necesitaríamos el servidor web **APACHE con dos versiones de PHP corriendo en simultáneo**.

Si creo dos contenedores, con apache instalado y la versión de PHP que necesito para mis sistemas en cada uno, **solo necesito:**

- **tener docker engine corriendo** en mi sistema
- y **subir los dos contenedores** configurados para ser accesibles

No se generan conflictos de versiones, puede accederlos simultáneamente, no tuve que hacer una instalación compleja de software y solo con una DOCKERFILE puedo compartir el mismo **STACK** ya configurado con mis compañeros de trabajo, sin importar bajo qué sistema operativo lo corran mientras tengan Docker instalado.

Porque es útil?

La curva de aprendizaje es dura al principio, y luego se ven los beneficios:

- acorta los tiempos de SETUP del ambiente en desarrollo,

- nos permite compartir nuestro código rápidamente y sin incompatibilidades entre sistemas,

- acerca a los desarrolladores a devops con menor dificultad.

- nos permite hacer nuestra aplicación portable y ahorra tiempo en el deploy en servidores,

- nos brinda una capa extra de seguridad,

Verificar la instalación

systemctl status docker
o
docker version

sudo systemctl enable docker - para que corra automáticamente cuando se inicia el sistema

COMMANDOS

docker version - muestra la versión instalada de docker

docker --help - muestra la ayuda de docker

docker ps - lista los contenedores corriendo (ps = process)

docker container ls - lista los contenedores corriendo (ls = list)

docker container ls -a -lista los contenedores en cualquier estado (stoped, exited, paused, etc)

Estados de los contenedores:
https://roytuts.com/what-are-the-possible-states-of-docker-container/

DOCKER HUB

Es el repositorio del cual Docker toma las imágenes mediante las cual provisiona nuestros contenedores en donde nuestra aplicación reside. Toma en cuenta que para poder explorar el DockerHub es necesario crear una cuenta dentro del sitio.

https://hub.docker.com/

Official Image son imágenes mantenidas por la organización que desarrolla ese software.

docker search <texto_a_buscar> - buscar imagenes desde la linea de commandos

docker search --help - mas opciones de búsqueda

COMANDOS DE CONTENEDOR

docker container run <nombre_imagen>- levanta un contenedor

docker container logs <nombre del contenedor> - muestra que esta sucediendo dentro del contenedor

docker container stop <nombre_del_contenedor> - detiene un contenedor

docker container rm -f <nombre_contenedor> - elimina un contenedor

docker container inspect <id> - container details

docker exec -it <nombre_contenedor> bash - connect to the running container

EJEMPLO

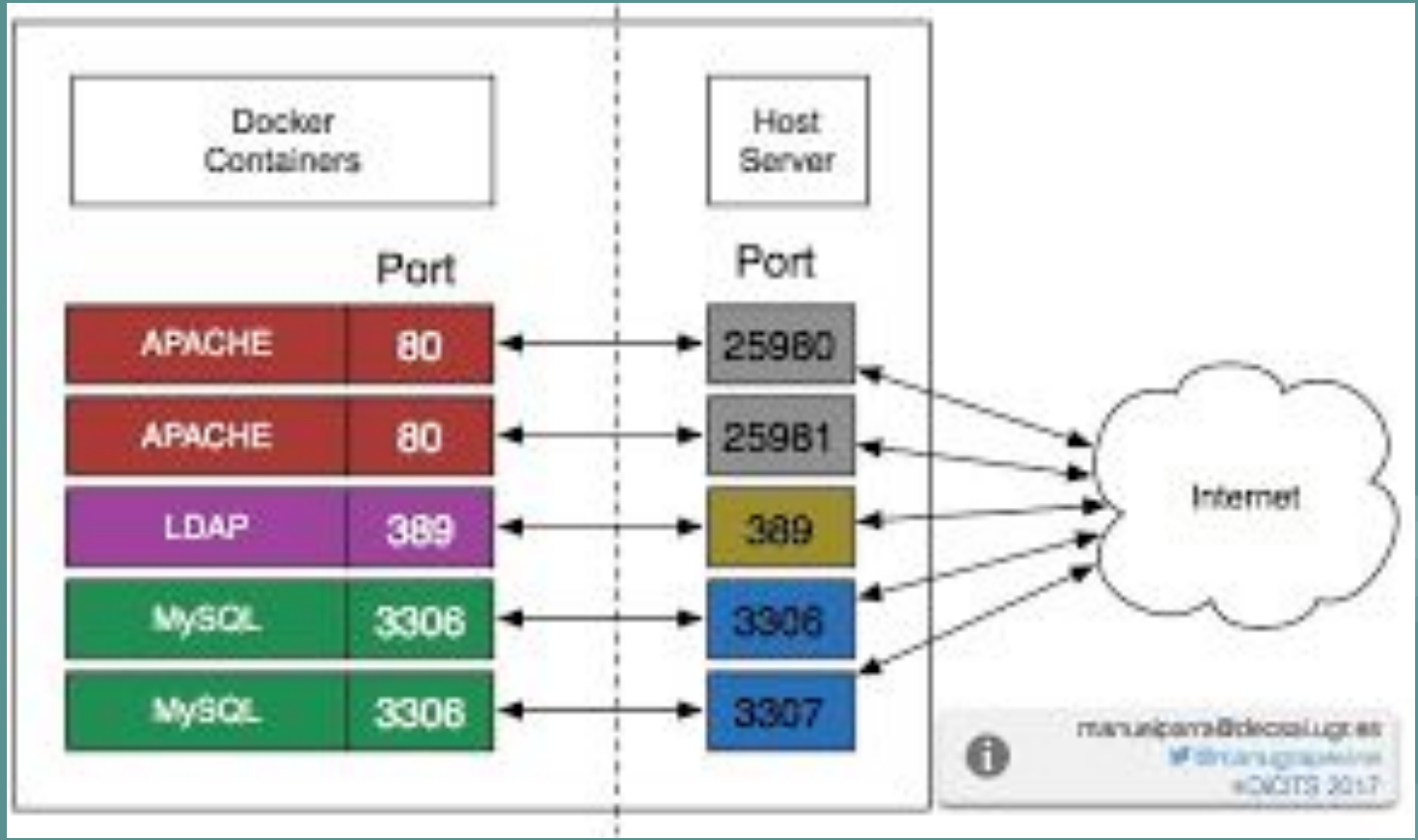
docker container run --name=mi_contenedor --publish 8080:80 --detach nginx

--name= <nombre> - Asigna nombre por defecto si no se asigna explicitamente

--publish <puerto_de_acceso> : <puerto_al_que_se_apunta_dentro_del_contenedor> - publica un puerto para que sea accedido desde fuera del contenedor
Los servidores web tienen asignado por defecto el puerto 80, por eso en nuestro caso linkeamos a ese puerto.
Depende el servicio que se vaya a correr el puerto al que apuntaremos: Mysql 3306, Postgresql 5432, etc

--detach - que corra en segundo plano, esto nos devuelve el control de la terminal una vez que el contenedor cominza a correr, si no es especificado nos mostraria los logs y no podriamos ingresar mas commandos sin parar el contenedor.

nginx - es la tag de la imagen que nos permite bajar la ultima version de la imagen, pero se pueden usar otras versiones, para ver las tags pueden ir a docker hub, y acceder a la imagen.


Docker Containers
Host Server
Port
Port
APACHE
80
APACHE
80
LDAP
389
MySQL
3306
MySQL
3306
25980
25981
389
3306
3307
Internet

EJEMPLO

docker container run --detach -e MYSQL_PASSWORD=True mysql --restart=always

-e <NOMBRE_VARIABLE_DE_AMBIENTE>=<valor_variable_de_ambiente> - permite asignar variables de ambiente

--restart=always - permite reiniciar el contenedor en caso que la máquina host sea reiniciada, se reinicie el docker deamon u ocurra un error.

EJERCICIO

1- Crear dos contenedores nginx que corran en puertos diferentes.

2- Acceder desde el buscador a localhost:puerto_publicado

3- Desde el bash del contenedor modificar el archivo /usr/share/nginx/html/index.html de forma que sea visible que pagina de inicio pertenece a qué contenedor.

4- Acceder a los index de cada contenedor en dos buscadores en simultáneo.

5- Detener los contenedores

6- Borrar los contenedores

7- Limpiar las imagenes

ALMACENAMIENTO PERSISTENTE

BIND MOUNT

-utiliza una d**irectorio que nosotros designamos** para almacenar datos y el contenedor lee directo ese desde ese directorio.

VOLUME

-el **directorio que asigna docker** donde se guardan los archivos y dentro de un area maejada por Docker.

COMANDOS DE ALMACENAMIENTO PERSISTENTE

docker container run --mount source=<nombe_del_volumen>,destination=<ruta_de_archivos_dentro_del_contenedor> nginx- crea un volumen con nombre, dentro del contenedor en la ruta especificada.

Todo lo que guardemos en esa ruta se vera reflejado en el area de docker del host, por lo que no se perdera al borrar el contenedor.

docker container run -v <ruta_de_archivos_en_HOST>:<ruta_de_archivos_dentro_del_CONTENEDOR> nginx- crea un bind mount que linkea la ruta dentro del contenedor con la del host.

COMANDOS DE ALMACENAMIENTO PERSISTENTE

docker container <nombre_contenedor> inspect -nos muestra entre otras información del contenedor los bind mounts y/o volumenes que el contenedor posee. (MOUNTS)

sudo docker volume ls - nos lista los volumenes disponibles

docker volume rm <id_del_volumen> - borra el volumen especificado

EJERCICIO

1- Crear un contenedor nginx con un volumen.

2- Desde el bash del contenedor modificar el archivo /usr/share/nginx/html/index.html de forma que sea visible que pagina de inicio pertenece a qué contenedor.

3- Acceder al index del contenedor con el buscador.

4- Detener el contenedor

5- Borrar el contenedor

6- Repetir los pasos 1 (volumen que apunta a la misma ruta) y 2

7- Acceder al index del contenedor con el buscador.

El volumen permitio que la informacion persistiera entre contenedores.

IMAGENES o BUILDS

Dockerfile es el archivo que nos permite crear nuestras propias imágenes empaquetando nuestra aplicación.

Las imágenes están formadas por otras imágenes.
Ej. de Dockerfile:

```
FROM nginx:alpine
WORKDIR /usr/share/nginx/html
COPY favicon.ico *.html *.js *.css *.svg ./
```

FROM - en qué imagen nos basamos
WORKFILE - establece el directorio del trabajo dentro del contenedor (como hacer cd en una carpeta específica)
COPY - copia la lista de archivos especificados (en este caso favicon.ico y todos los que tengan extensión html,js,css,svg) al directorio destino especificado por el ultimo parametro (./)

los * son comodines (cualquier nombre de archivo que termine con .htm por ejemplo)

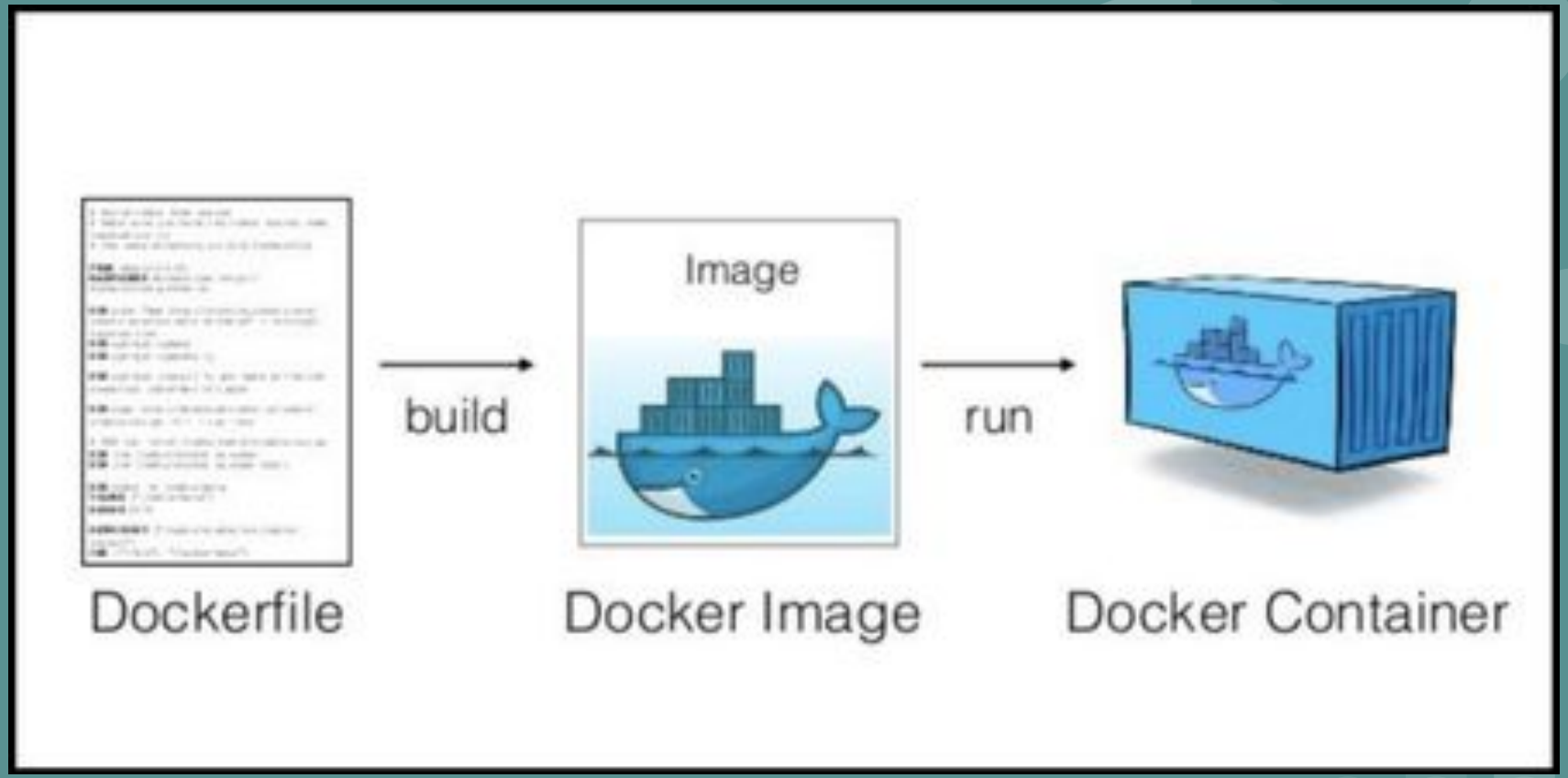


Image
build
run
Dockerfile
Docker Image
Docker Container

IMAGEN

El archivo **.dockerignore** NO permite que los archivos que se especifican dentro del él sean agregados a nuestra imagen.

Por ejemplo si tuviéramos un archivo con variables de ambiente que pudieran tener información delicada como credenciales, deberíamos agregarlo al .dockerignore

IMAGEN

Creando una imagen:

docker image build --tag <nombre_imagen>:<tag> . - crea una imagen con el dockerfile del directorio actual

docker image build --tag <nombre_imagen>:<tag> --file - permite buscar el dockerfile con otro nombre u en otra ubicación

sudo docker image ls - listados de imágenes disponibles

sudo docker image push <nombre_imagen>:<tag> sube la imagen a docker hub

DOCKERFILE

Para más referencias de como armar el archivo dockerfile visite:
https://docs.docker.com/engine/reference/builder/