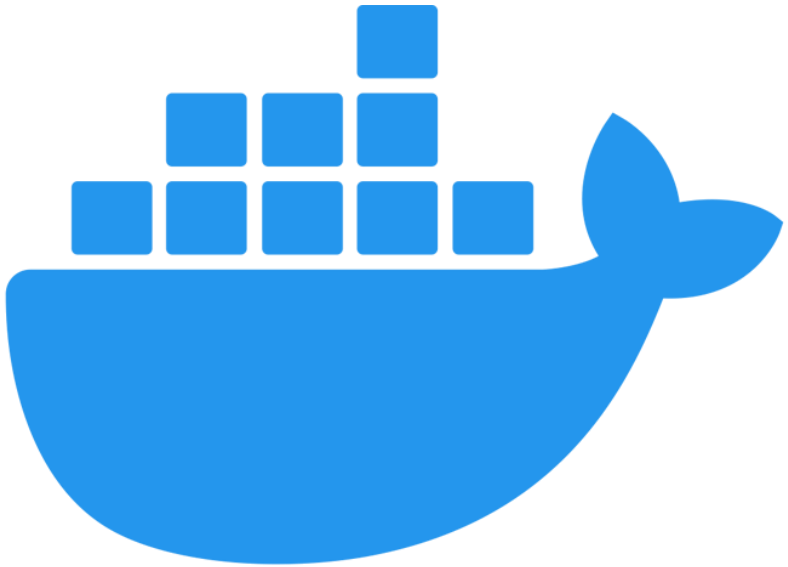
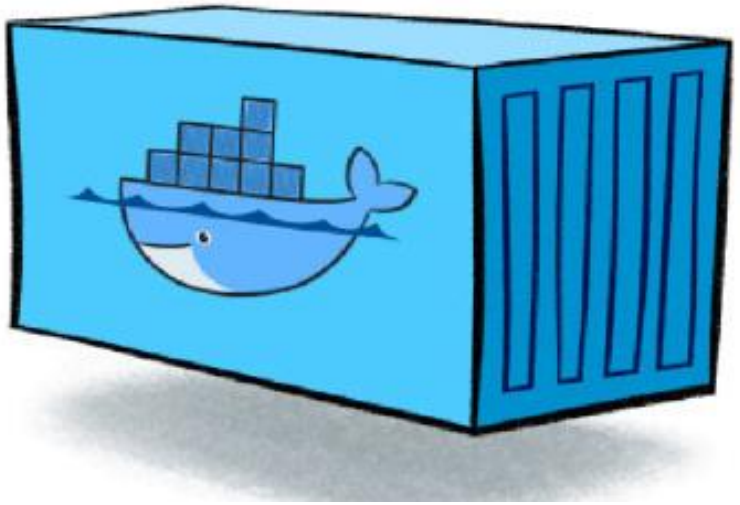
Docker



Que es Docker?

Es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux.

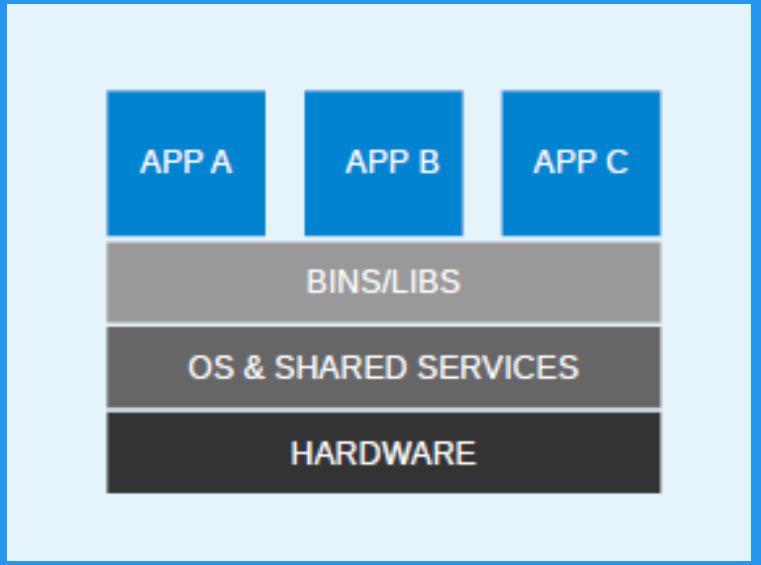
Contenedores?



Los contenedores Docker

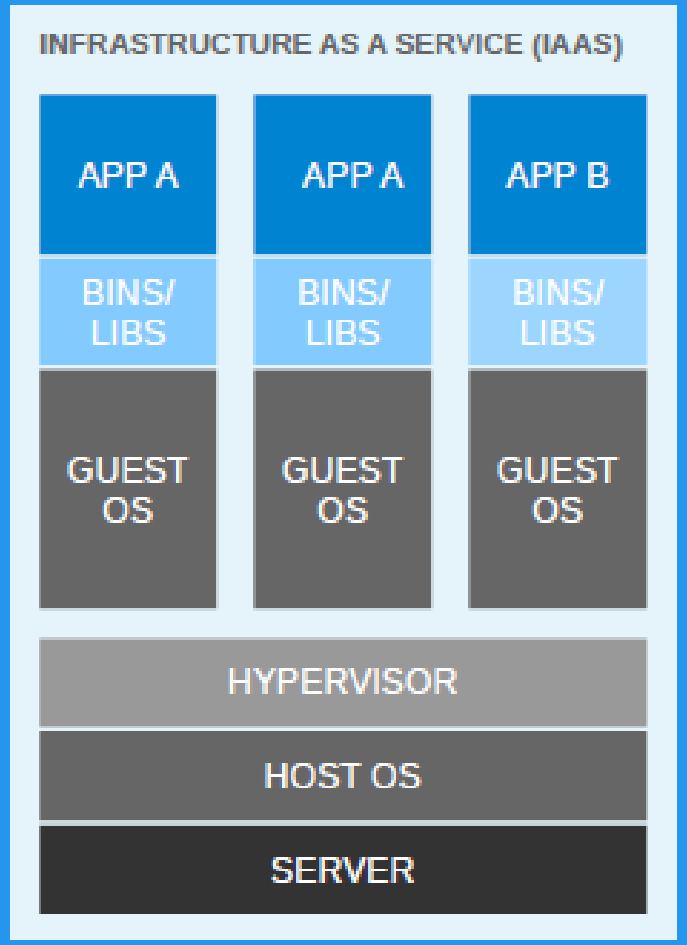
Envuelven una aplicación y sus dependencias en una unidad estandarizada que incluye todo lo que necesita para ejecutarse: código, tiempo de ejecución, herramientas del sistema y bibliotecas. Esto garantiza que la aplicación siempre se ejecutará igual en cualquier lugar que tenga Docker instalado.

Algo de historia

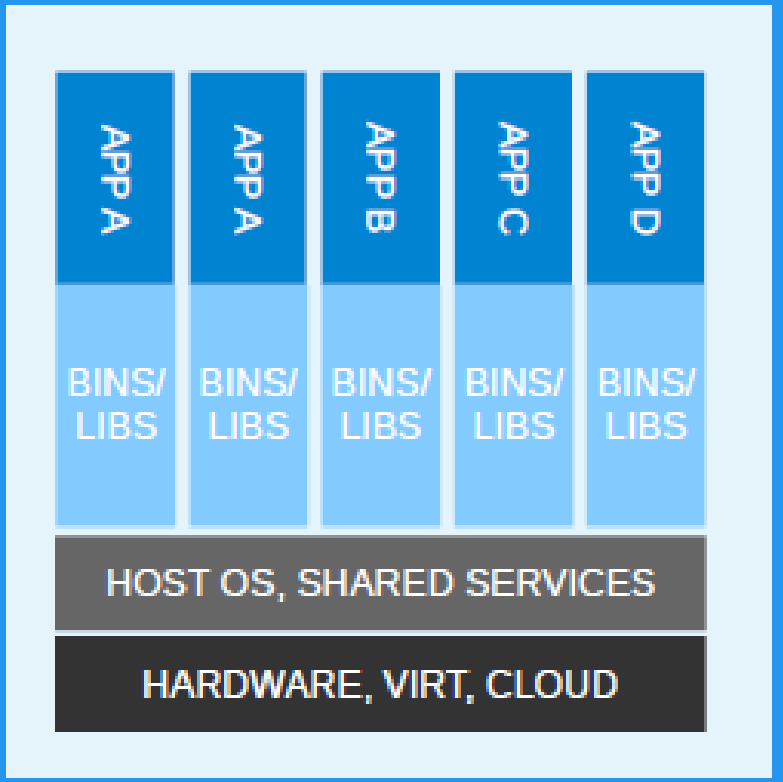


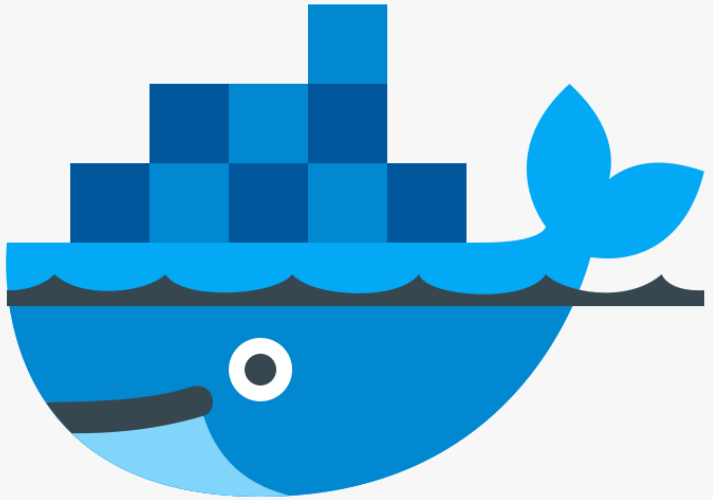
En un principio las aplicaciones se implementaban directamente en la pc del usuario y sobre un determinado SO, compartiendo tiempo de ejecución con otras aplicaciones lo que infrautilizaba recursos del sistema y limitaba a los desarrolladores.

Para implementaciones flexibles, y con el fin de utilizar mejor los recursos del sistema host, se inventó la virtualización. Con hipervisores, como KVM, XEN, ESX, Hyper-V, etc., se emulaban el hardware para máquinas virtuales e implementaba un sistema operativo invitado en cada máquina virtual. Las máquinas virtuales pueden tener un sistema operativo diferente al de su host; esto significaba administrar las revisiones, la seguridad y el rendimiento de cada máquina virtual.
Con la virtualización, las aplicaciones se aíslan a nivel de máquina virtual y se definen por el ciclo de vida de las máquinas virtuales. Esto nos da una mayor flexibilidad a costa de una mayor complejidad y redundancia.



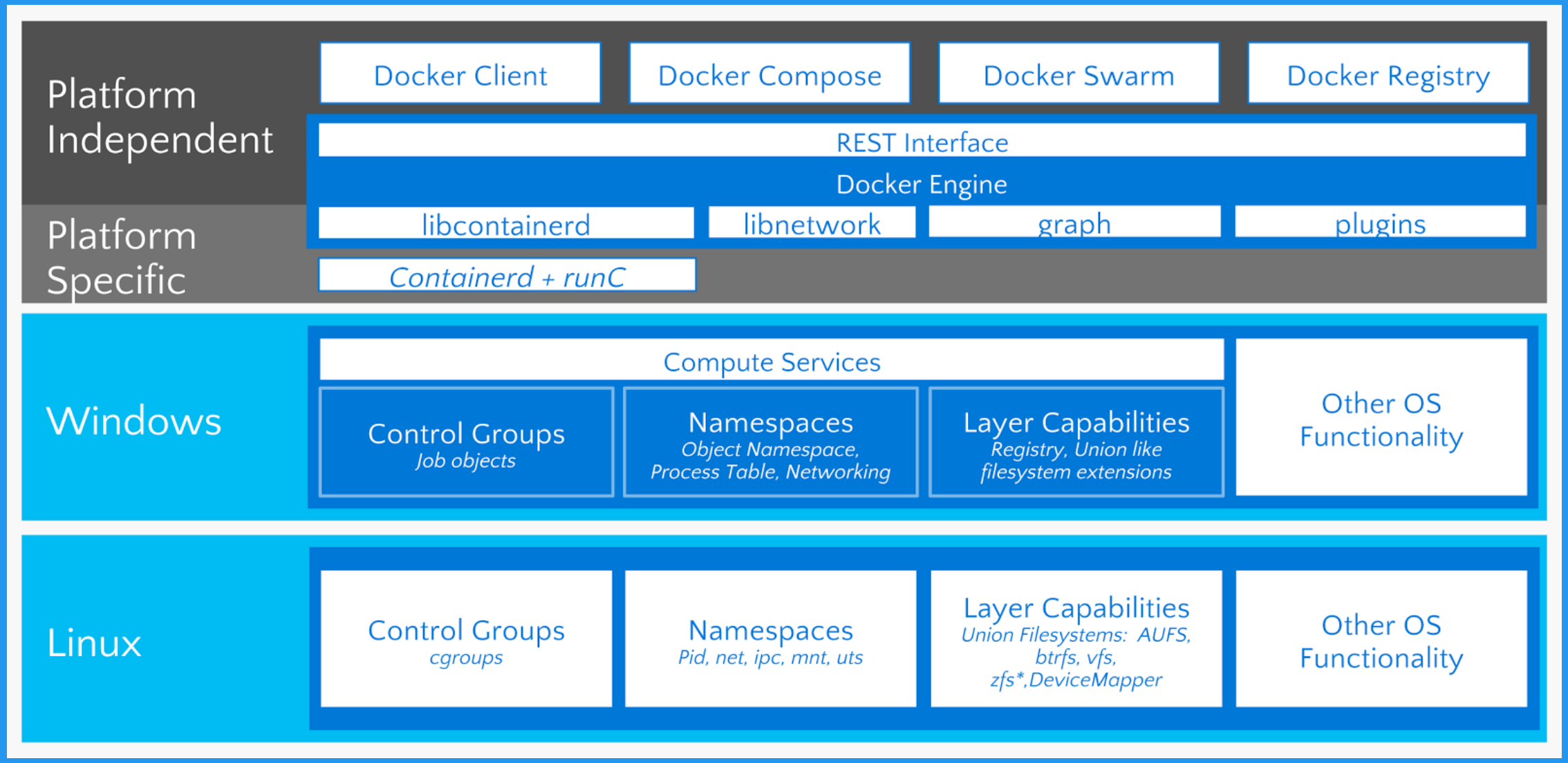
Al pasar el tiempo se elimino la capa de hipervisor para reducir la emulación y complejidad del hardware. Las aplicaciones se empaquetan con su entorno de tiempo de ejecución y se implementan mediante contenedores como OpenVZ, Solaris Zones y LXC. Estos eran menos flexibles en comparación con las máquinas virtuales; por ejemplo, no podemos ejecutar Windows en el sistema operativo Linux. Eran menos seguros si un contenedor se veía comprometido, era posible obtener acceso completo al sistema operativo host. Puede ser complejo para configurar, administrar y automatizar.



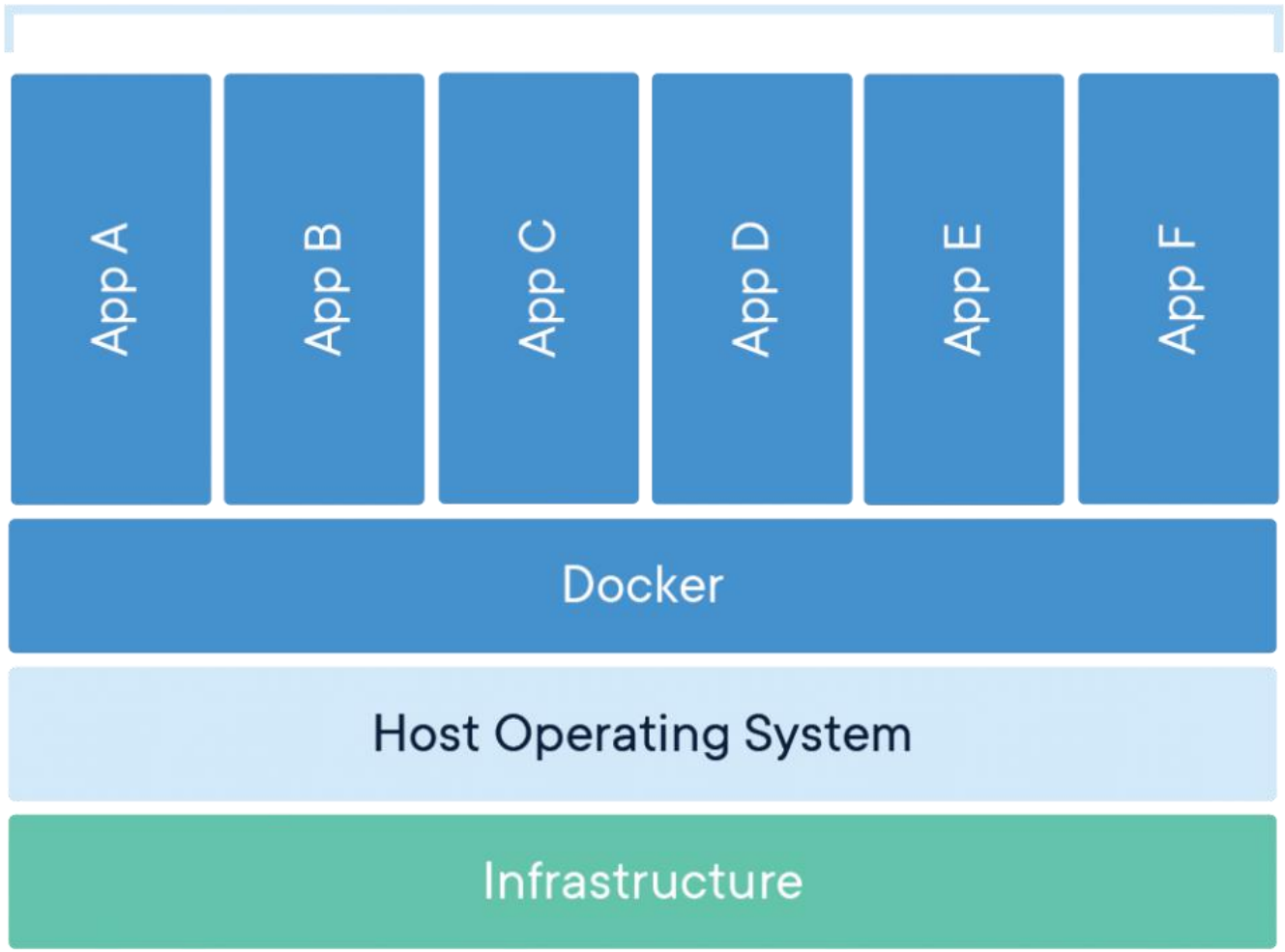


Y entonces llego Docker ...

Docker fue iniciado como un proyecto interno por el fundador de dotCloud Solomon Hykes. Fue lanzado como código abierto en marzo de 2013 bajo la licencia Apache 2.0. Docker utiliza las características de kernel subyacentes del sistema operativo, que permiten la contenedorización.

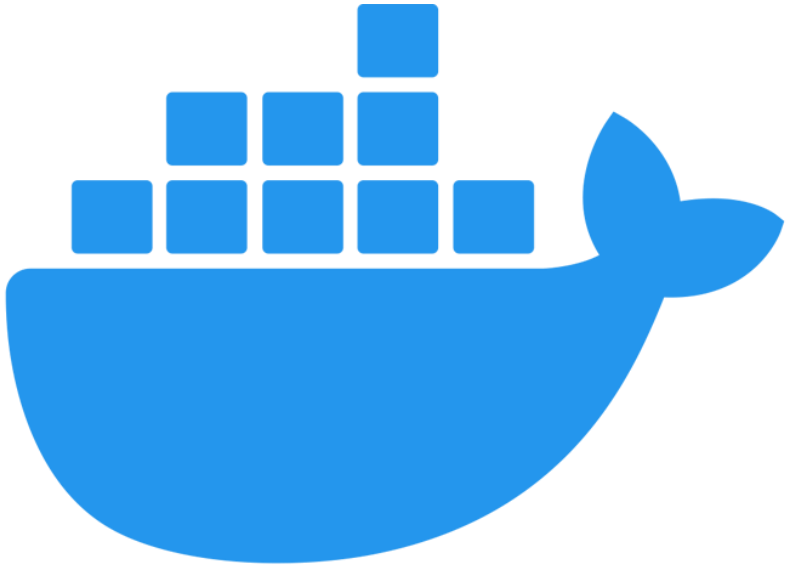

Platform Independent
Docker Client
Docker Compose
Docker Swarm
Docker Registry
REST Interface
Docker Engine
Platform Specific
libcontainerd
libnetwork
graph
plugins
Containerd + runC
Windows
Compute Services
Control Groups
Job objects
Namespaces
Object Namespace, Process Table, Networking
Layer Capabilities
Registry, Union like filesystem extensions
Other OS Functionality
Linux
Control Groups
cgroups
Namespaces
Pid, net, ipc, mnt, uts
Layer Capabilities
Union Filesystems: AUFS, btrfs, vfs, zfs*,DeviceMapper
Other OS Functionality

Containerized Applications



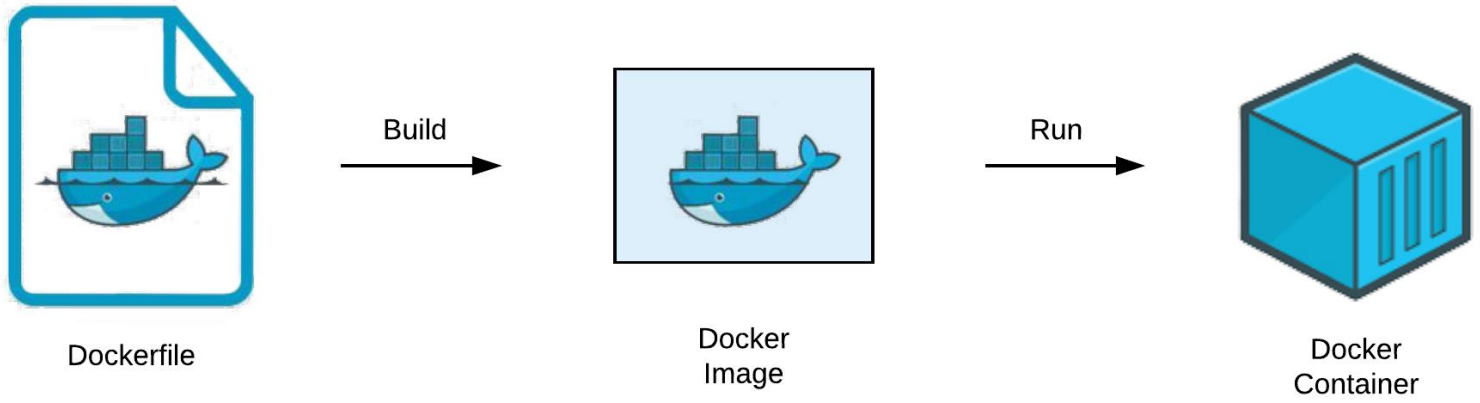
Los contenedores son una abstracción en la capa de la aplicación que agrupa el código y las dependencias juntas. Se pueden ejecutar varios contenedores en la misma máquina y compartir el núcleo del sistema operativo con otros contenedores, cada uno de los cuales se ejecuta como procesos aislados en el espacio del usuario. Los contenedores ocupan menos espacio que las máquinas virtuales (las imágenes de los contenedores suelen tener un tamaño de decenas de MB), pueden manejar más aplicaciones y requieren menos máquinas virtuales y sistemas operativos.

Los contenedores aíslan el software de su entorno y aseguran que funcione de manera uniforme a pesar de las diferencias, por ejemplo, entre desarrollo y puesta en escena.



Imágenes Docker

Una imagen es una “plantilla”, que captura del estado de un contenedor.
Por ejemplo una imagen podría contener un sistema operativo Ubuntu con un servidor Apache y tu aplicación web instalada.
Hay muchas imágenes públicas con elementos básicos como Java, Ubuntu, Apache…etc, que se pueden descargar y utilizar.
Normalmente cuando creas imágenes, partimos de una imagen padre a la que le vamos añadiendo cosas (p.e: una imagen padre con Ubuntu y Apache, que hemos modificado para instalar nuestra aplicación).



Dockerfile

Un Dockerfile es una archivo mediante el cual puedes construir una Imagen Docker segun tus especificaciones a travez de comandos que se ejecutaran de manera automatica.

Documentacion:

https://docs.docker.com/engine/reference/builder/

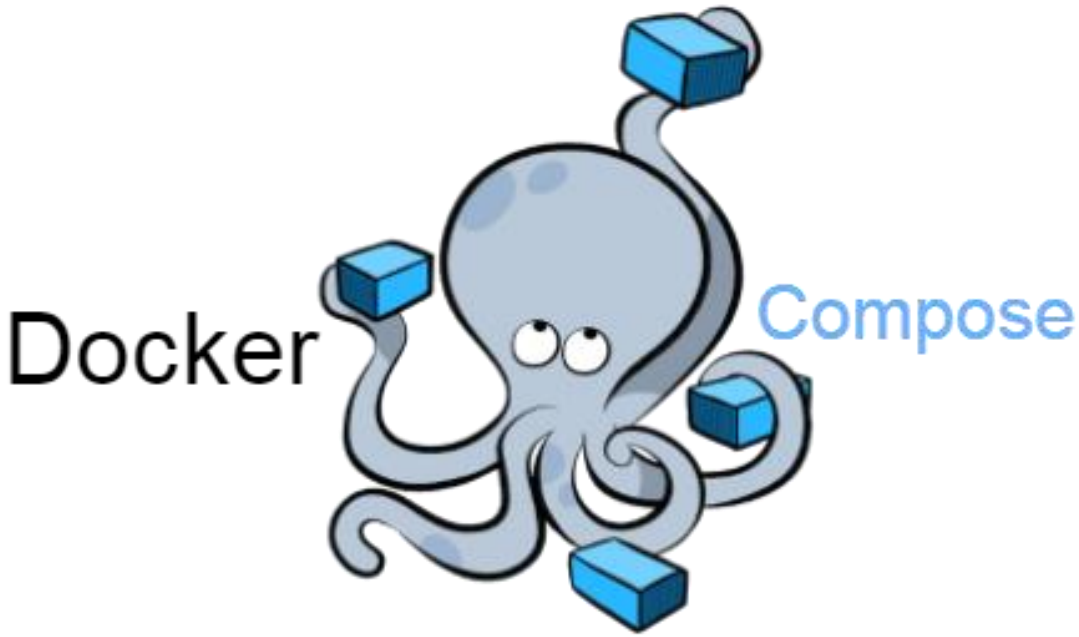
Comandos básicos

- **ADD** - permite copiar archivos desde el SO host al contenedor.
- **CMD** - permite ejecutar comandos desde el contenedor.
- **ENTRYPOINT** - especifica una aplicacion que se inicie cada vez que se cree un nuevo contenedor.
- **ENV** – especifica variables de entorno
- **EXPOSE** - especifica los puertos de conexion mediantes lo cuales podremos acceder al contenedor desde fuera del mismo.
- **FROM** - especifica la imagen base bajo la cual se construira el contenedor.
- **MAINTAINER** - define el nombre y correo del creador del contenedor
- **RUN** - ejecuta directivas del comando run en la construccion del contenedor.
- **USER** - especifica los IDs de los usuarios que podran acceder al contenedor.
- **VOLUME** - habilita el acceso desde el contenedor a un directorio del Sistema Host
- **WORKDIR** - especifica el path o direccion desde donde se ejecutaran los commandos de la shell
- **LABEL -** define etiquetas de la imagen.

E jemplo de un Dockerfile

Instalacion de MongoDB sobre una imagen de Ubuntu

```
# Descargamos la imagen de Ubuntu y le asignamos un titulo.
FROM dockerfile/Ubuntu AS mongo-server

# Comando a ejecutar: actualizamos el sistema, instalamos mongoDB y eliminamos los
archivos de dependencias para evitar actualizaciones automaticas
RUN apt-get update && apt-get install -y mongodb-org && rm -rf /var/lib/apt/lists/*

# Definimos el directorio que competiremos con el SO Host.
VOLUME ["/data/db"]

# Definimos nuestro directorio de trabajo.
WORKDIR /data

# Definimos el comando mongod para ejecutar el servidor de mongo en cuanto se inicie el
contenedor.
CMD ["mongod"]

# Definimos los puertos de conexion.
#   - 27017: process
#   - 28017: http
EXPOSE 27017
EXPOSE 28017

# Definimos datos del creador
MAINTAINER Oliver oliverzulett@gmail.com
```



Docker-Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones que requieran de varios contenedores, Docker compose trabaja a base de un archivo .yml en el que se especifica un Stack de aplicaciones en contenedores que en conjunto crean un servicio.

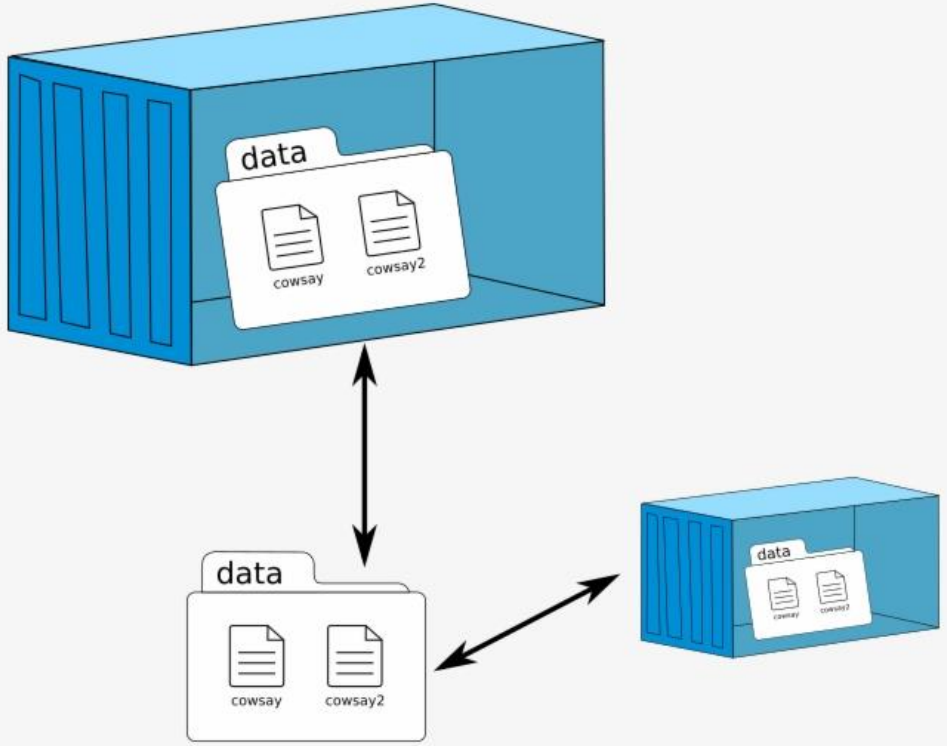
Documentación:
https://docs.docker.com/compose/compose-file/

E jemplo de un archivo .yml

Creación de un servicio REST con mongoDB y express

```
#Version de Docker-Compose a utilizar
version: '3.1'

# servicios que se definiran
services:
  # definición del primer servicio: Mongo
  mongo:
    # Docker compose nos permite utilizar dockerfiles para crear contenedores asi que utilizamos el que creamos en el ejemplo
    # anterior.
    context: .   # dirección del dockerfile
    dockerfile: Dockerfile

  # Definición del segundo servicio: Express
  mongo-express:
    # imagen que se utilizara
    image: mongo-express
    # especifica la política de reinicio
    restart: always
    # puertos de conexion
    ports:
      - 8081:8081
    # variables de entorno
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: root
```

Volúmenes



Los volúmenes son unidades de almacenamientos asignadas fuera del sistema en las que podremos almacenar los datos generados en un contenedor a su vez podremos modificar y agregar datos desde fuera del contenedor, esto nos garantiza una persistencia de datos pese a que un contenedor ya no este disponible. Cabe señalar que varios contenedores pueden acceder al mismo volumen a la vez.

Comandos básicos

Operaciones básicas:

```
docker pull <image-name>:<versión>      # descarga una imagen desde Docker Hub
docker image ls          # lista las imágenes descargadas
docker image <name o id>         # elimina una imagen

# crear y ejecutar un contenedor en segundo plano
docker –-name <container-name> run –d <image-name>
docker ps       # muestra los contenedores que se están ejecutando
docker start <cont-name o id>   # inicia un contenedor detenido
docker stop <sont-name o id>    # detiene un contenedor
docker rm <cont-name o id>      # elimina un contenedor

# crear una imagen Docker apartir de un dockerfile
docker build –t <path>/<repo-name>:<tag> .

# crea los contenedores definidos en el .yml con Docker compose
docker-compose –f <path>/<compose-file>.yml

# crea un contenedor que utilizara un volumen
docker --name <c-n> ~v <path del Host>:<path del contenedor> -d <image-name>

# comandos completos:
https://docs.docker.com/engine/reference/commandline/docker/
```