Docker

WORKSHOP

“De Novato al más Allá”

Marcos Pablo Russo

Administrador GNU/Linux, Infraestructura, Auditor, SOC, Autodidacta, ex-profesor de la UTNFra Arquitectura y Sistemas Operativos. Creador de la Distribución de Informática Forense CondorLinux y luego Huemul (Centrux).

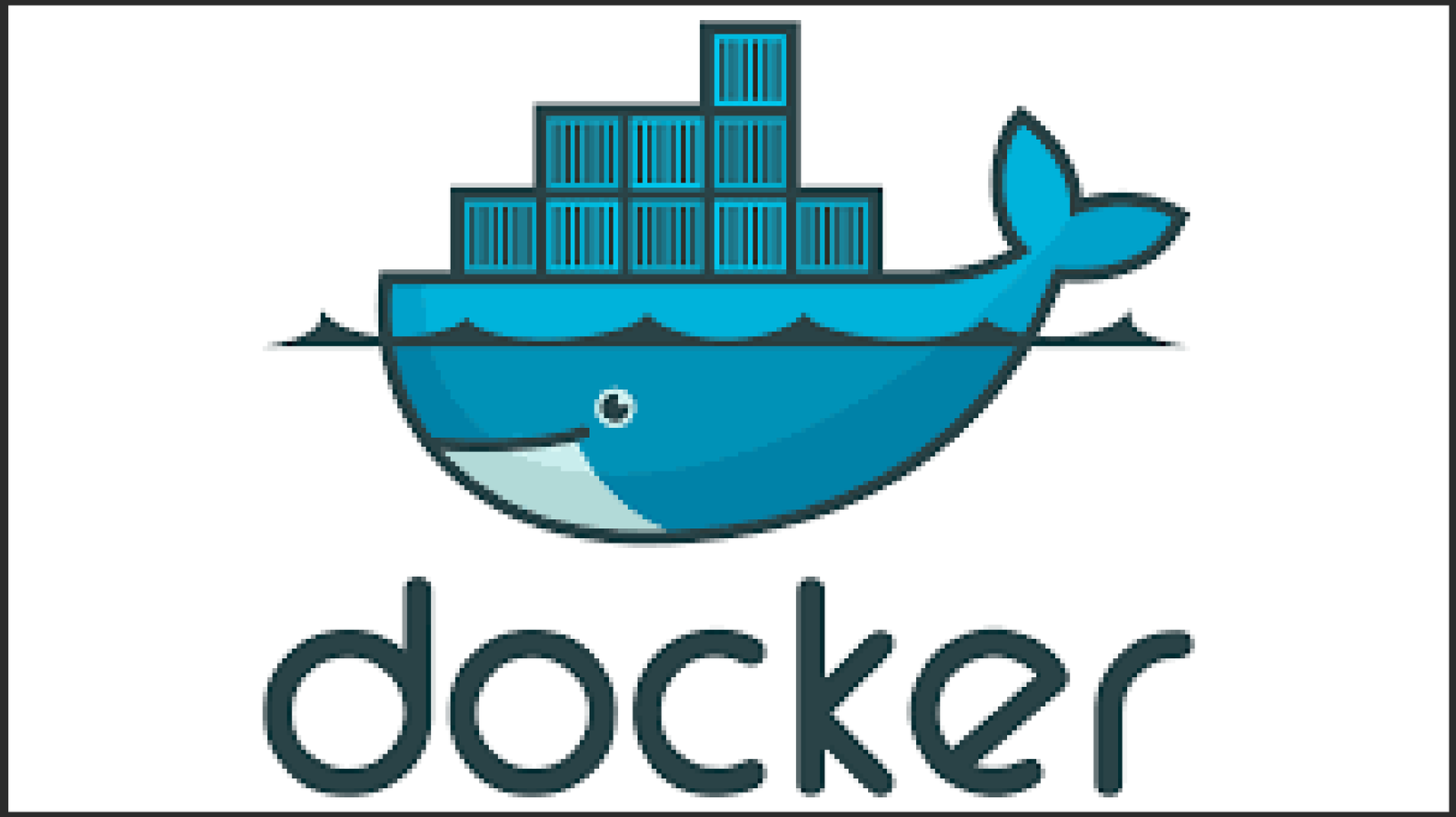
Expositor en varios países, como Guatemala y México.

También dictado de charla en la Ekoparty,

- Ekoparty.
- CEAT Centro de Estudios en Administración Tributaria FCE UBA.
- Ejercito Argentino
- etc.

Capacitacitador en:

- Ciberseguridad en Mundoe.
- Informática Forense en CPCI
- Dictado de clase de OSINT en HackAcademy (Ekoparty)


docker

¿ Qué es Docker ?

- Docker es una herramienta tanto para el sysadmin como developer.
- Ayuda a automatizar el despliegue de aplicaciones dentro de los contenedores de software, proporcinando una capa adicional de abstracción eliminando el overhead que provoca correr un Sistema Operativo virtual para cada una sola aplicación.
- Además de la propia aplicación que gestiona los contenedores (a los que llamamos docker engine o docker), existe un repositorio de imágenes creadas por la comunidad, llamada docker hub.
- Tanto los developers como los sysadmin pueden compartir entornos de trabajo (imágenes).

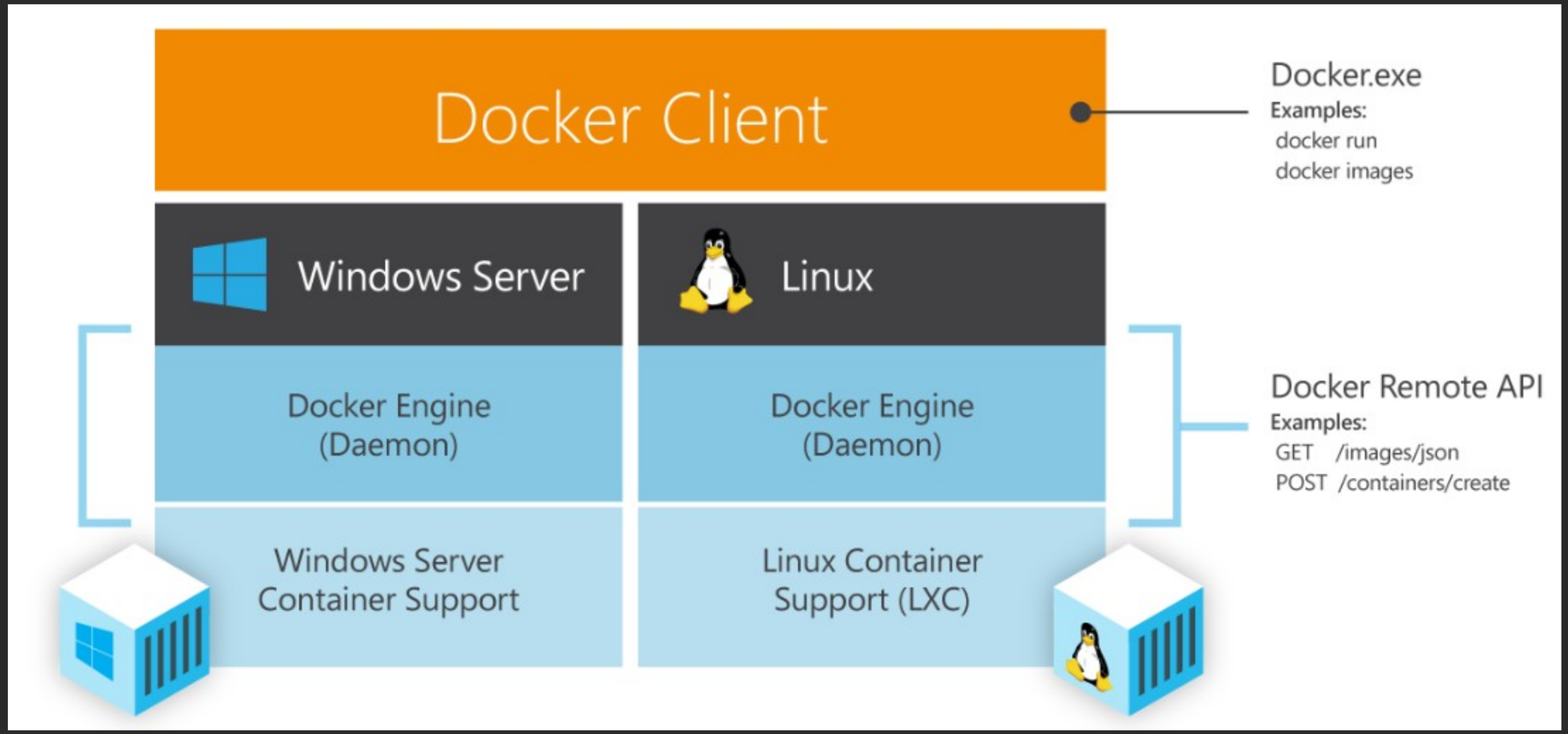
¿ Qué es Docker ?

- Corre en equipos de 64 bits.
- Gran parte de éxito de docker se basa en su fácil portabilidad y su ligereza.
- Soporta los sistemas operativos Windows, Mac y obviamente GNU/Linux.
- Docker es tanto cliente como servidor.

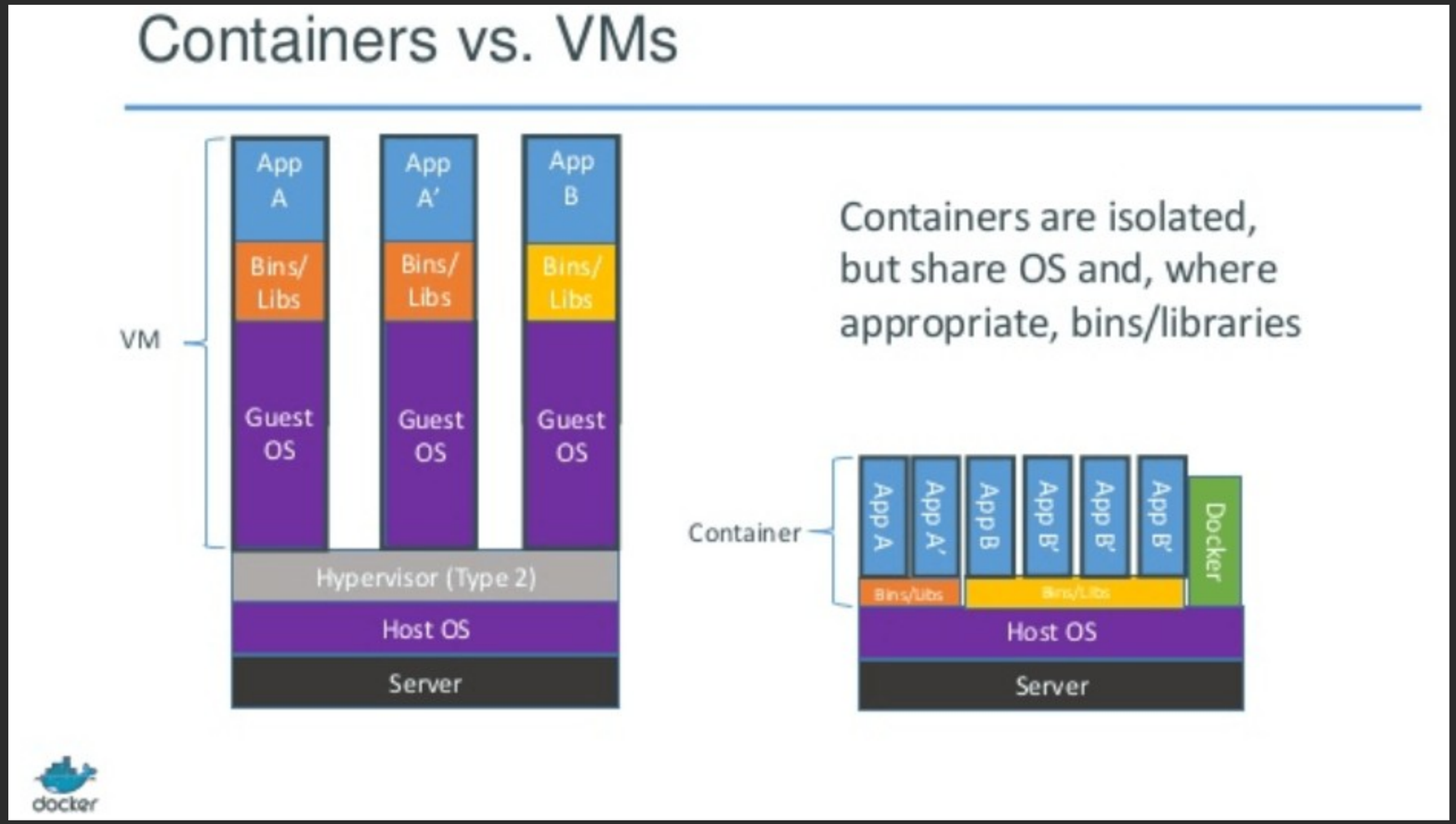
¿ Qué no es Docker ?

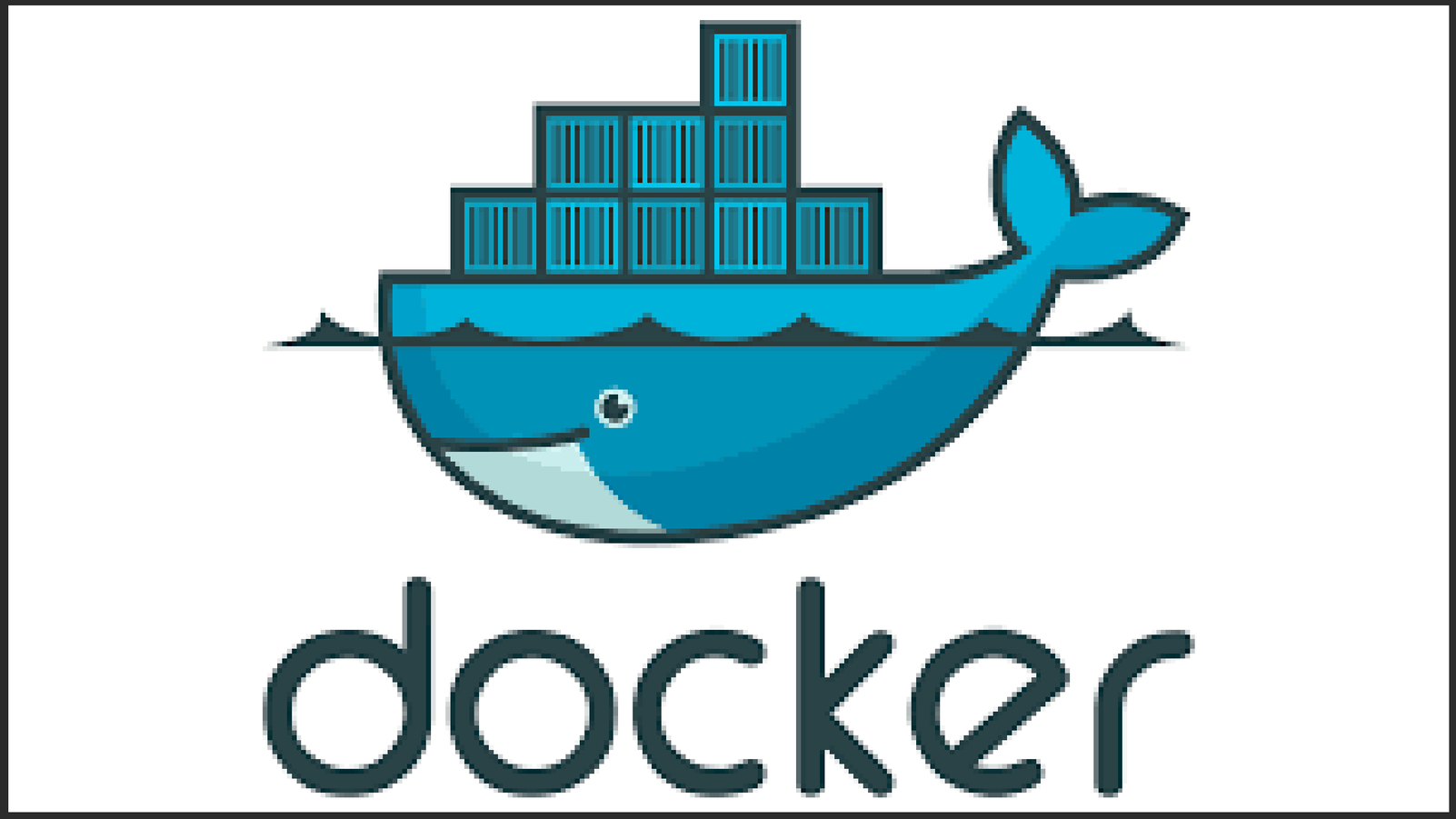
- Un gestor de entornos de desarrollo virtuales (vagrant).
- Un software de virtualización (hypervisor).
- (kvm, xen, vmware, virtualbox, etc).
- Un gestor de configuración (puppet/chef).
- Un simple contenedor de software (lxc).
- Aunque se parece como servidor.

¿ Qué es Docker ?



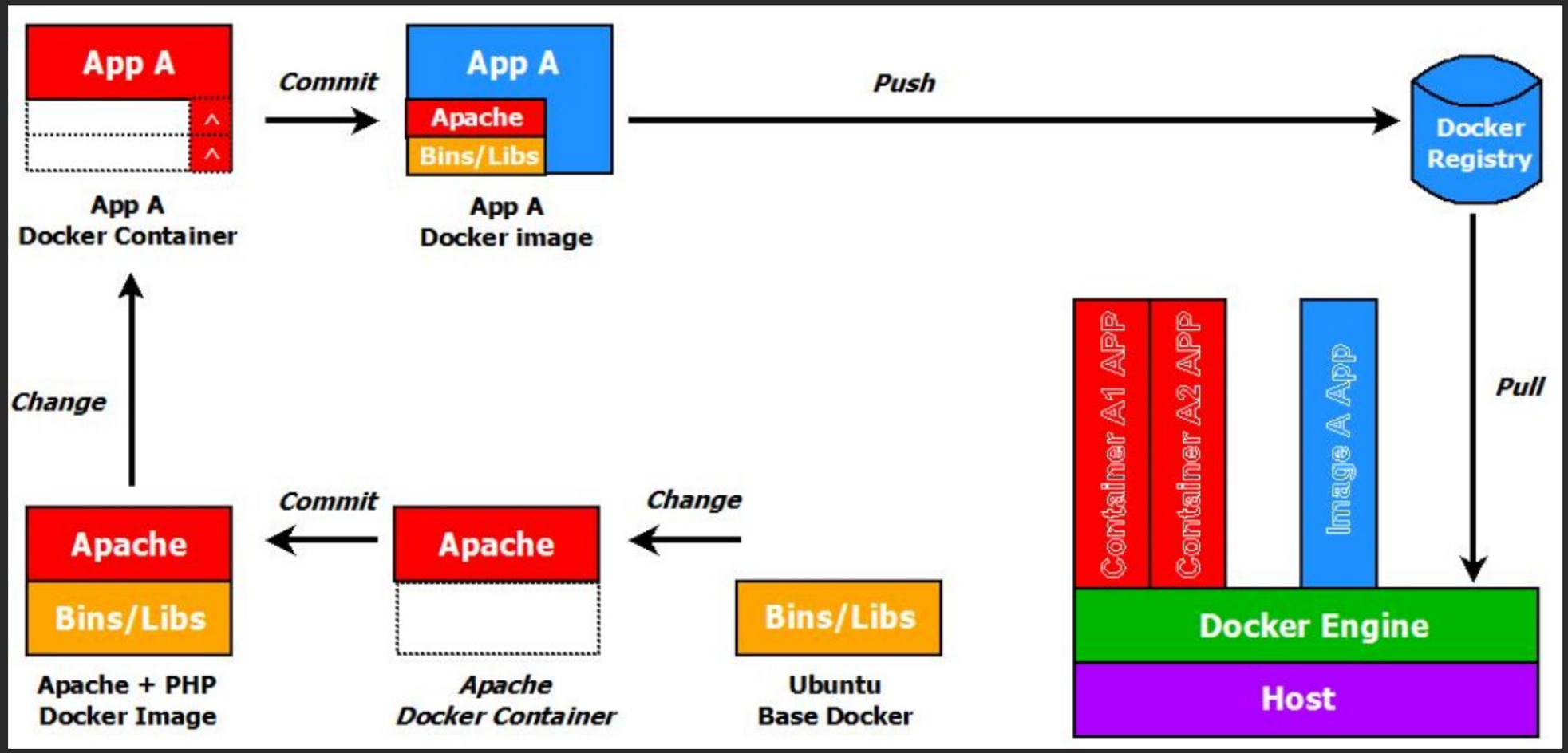
¿ Qué es Docker ?



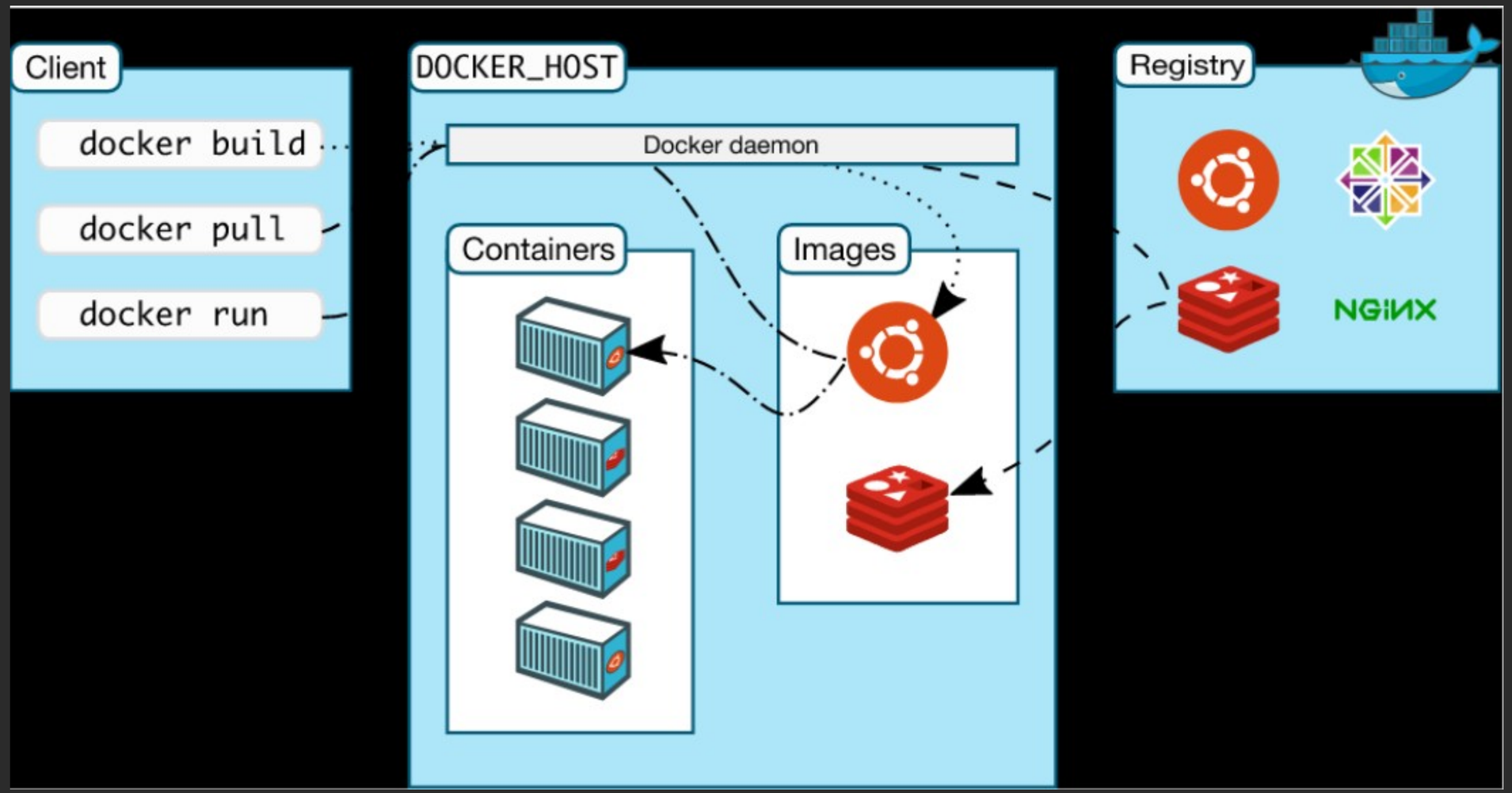


CICLO DE VIDA

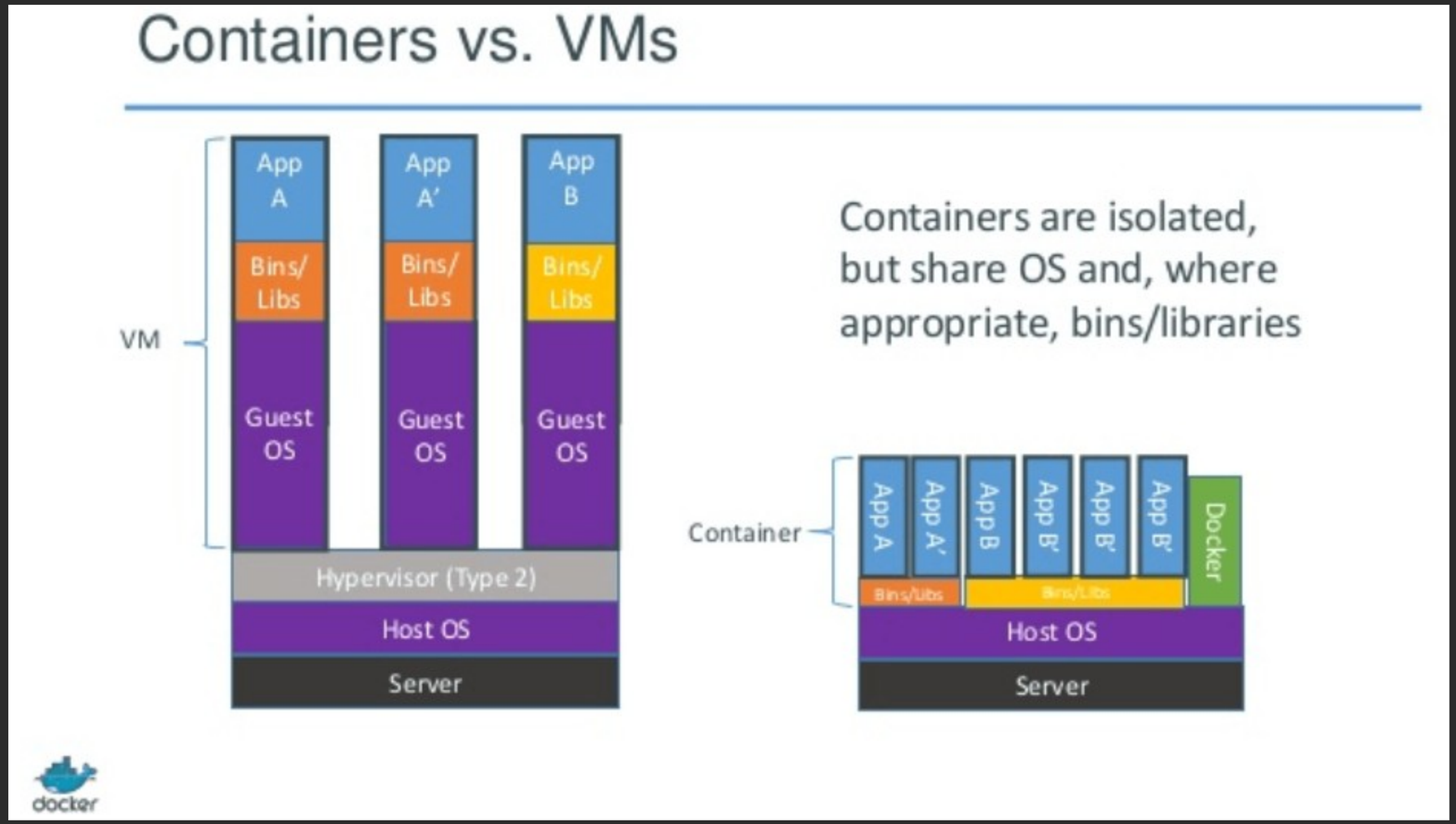
Ciclo de Vida

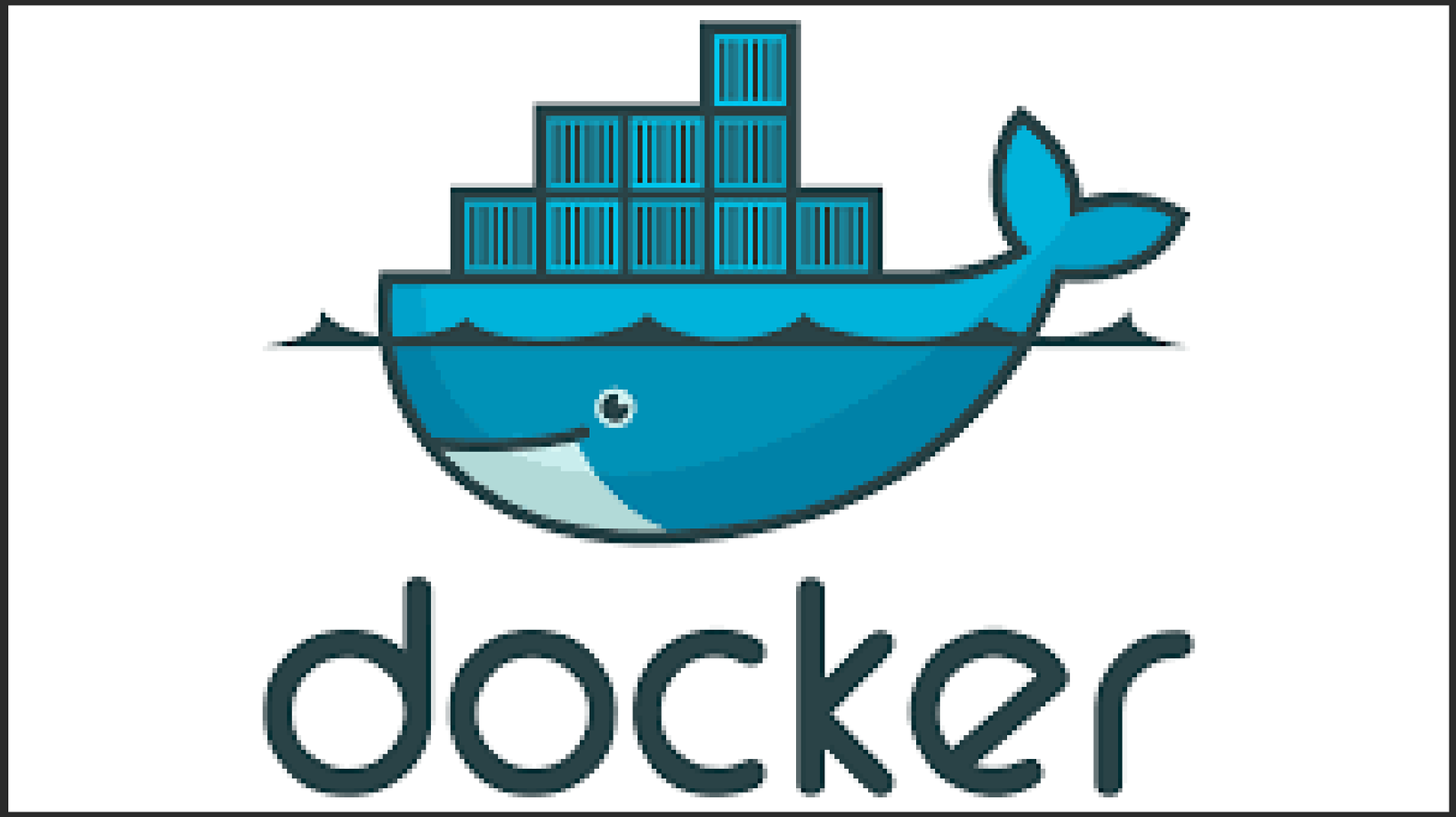


Ciclo de Vida



Arquitectura de Docker

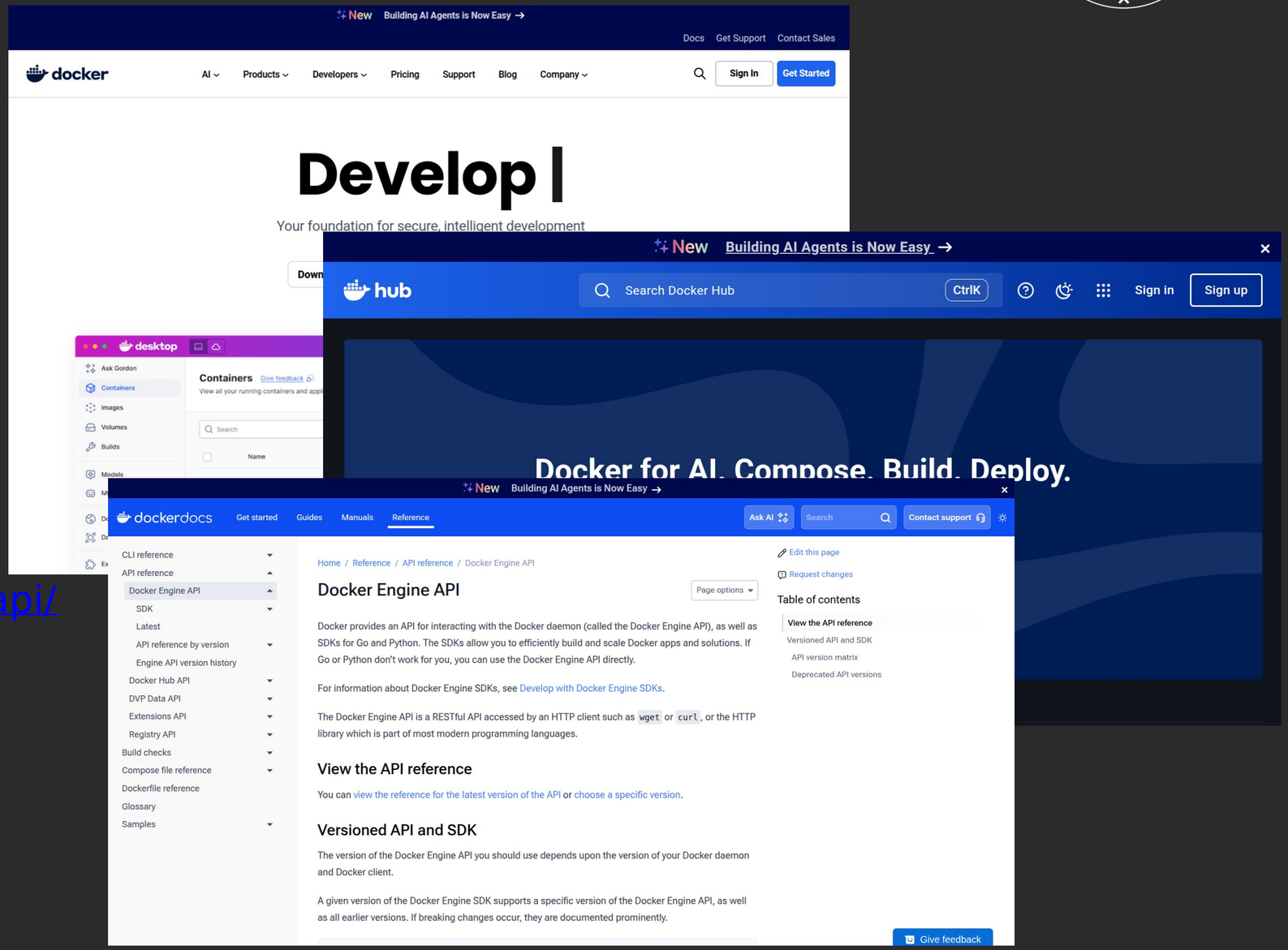


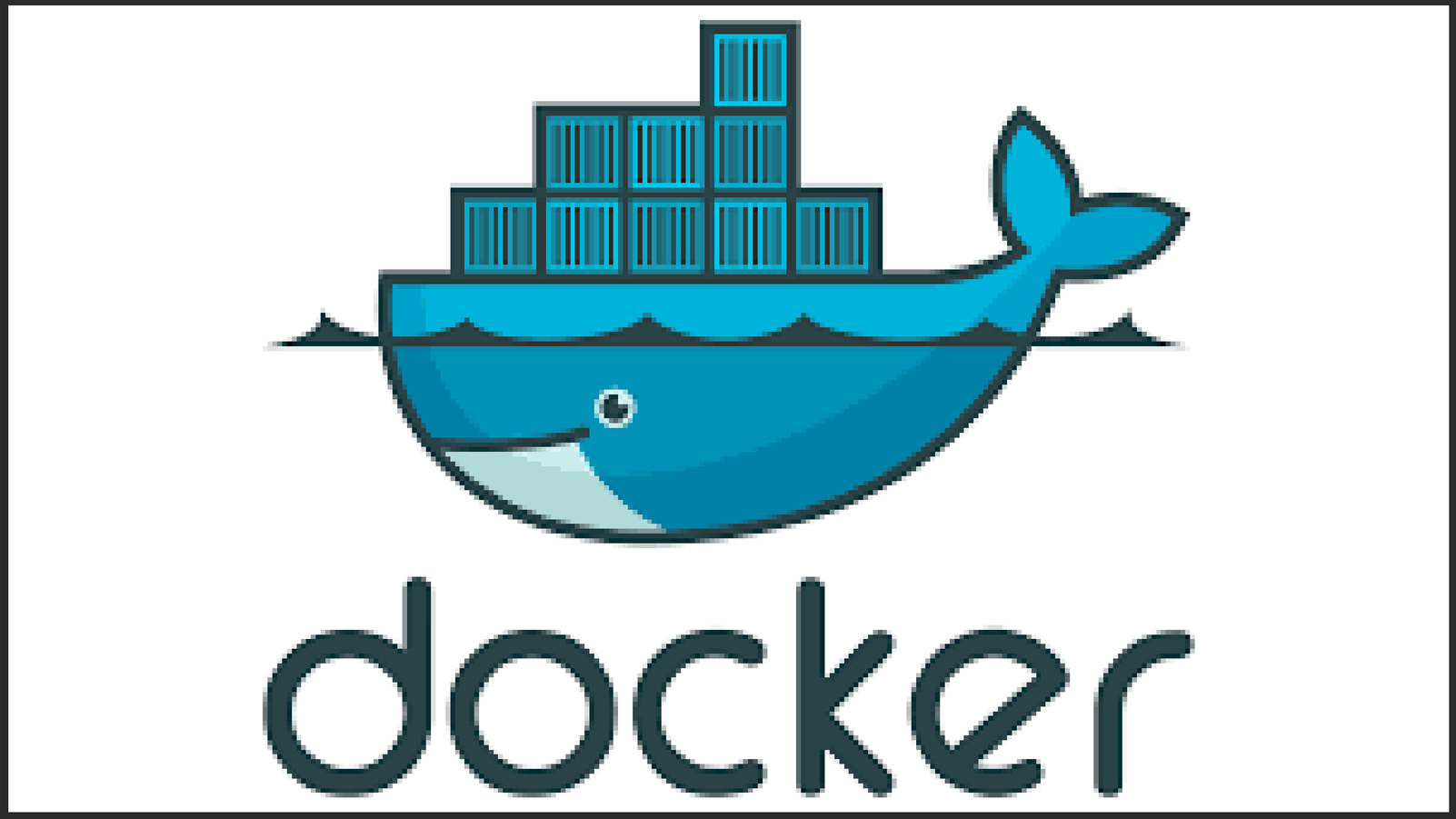


WEB

Web

- WEB
  - https://www.docker.com
- HUB
  - https://hub.docker.com
- DOCUMENTACION
  - https://docs.docker.com
- API
  - https://docs.docker.com/engine/api/





INSTALACIÓN

Instalación

- DEBIAN
  - https://docs.docker.com/engine/install/debian/
- UBUNTU
  - https://docs.docker.com/engine/install/ubuntu/

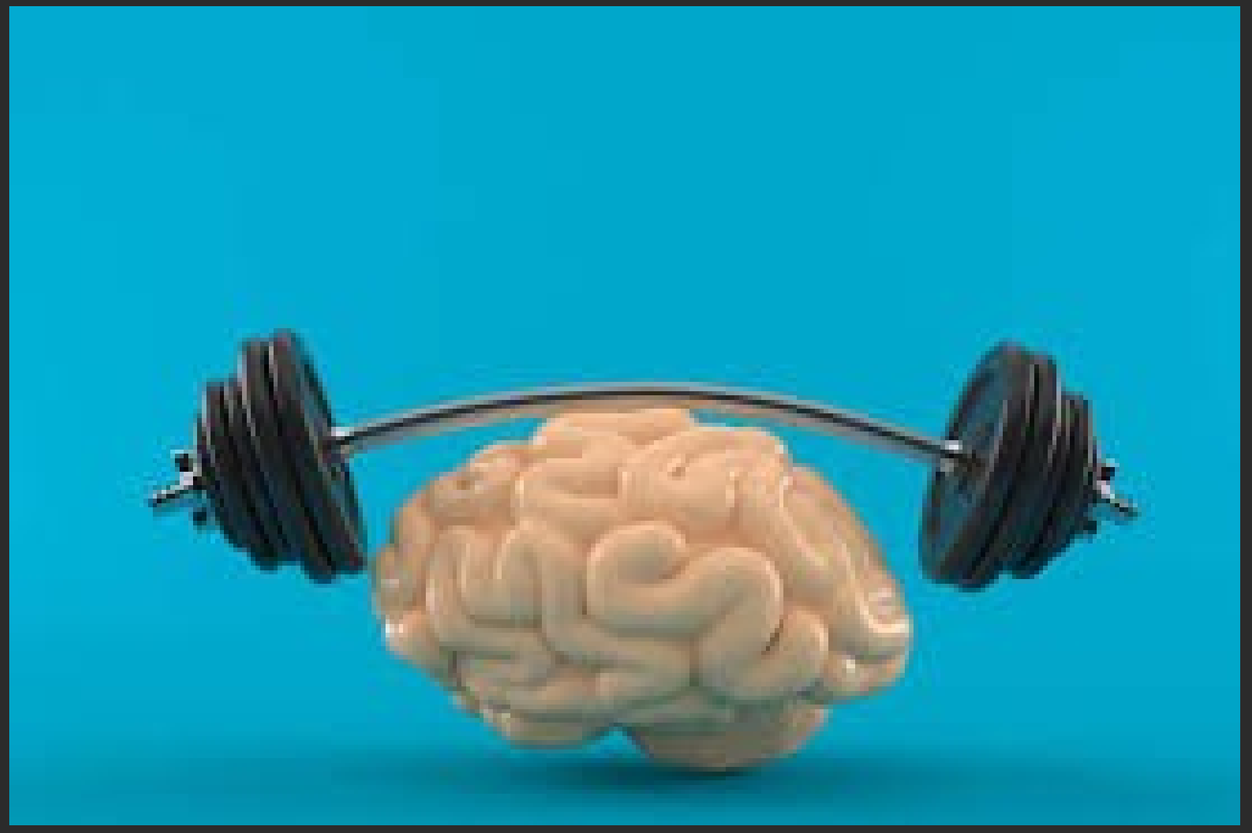
Instalación

```
localhost:~# ps aux | grep docker
 2106 root      0:21 /usr/bin/dockerd -p /run/docker.pid
 2237 root      0:36 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
 2701 root      0:00 grep docker
```

Instalación

- # usermo -G docker -a USUARIO

! Vamos a Practicar !



Comandos

- # ifconfig

```
localhost:~# ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:05:C7:D4:60
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 08:00:27:9D:FD:59
          inet addr:192.168.1.109  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe9d:fd59/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10072 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1517 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3094468 (2.9 MiB)  TX bytes:164694 (160.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

- # route

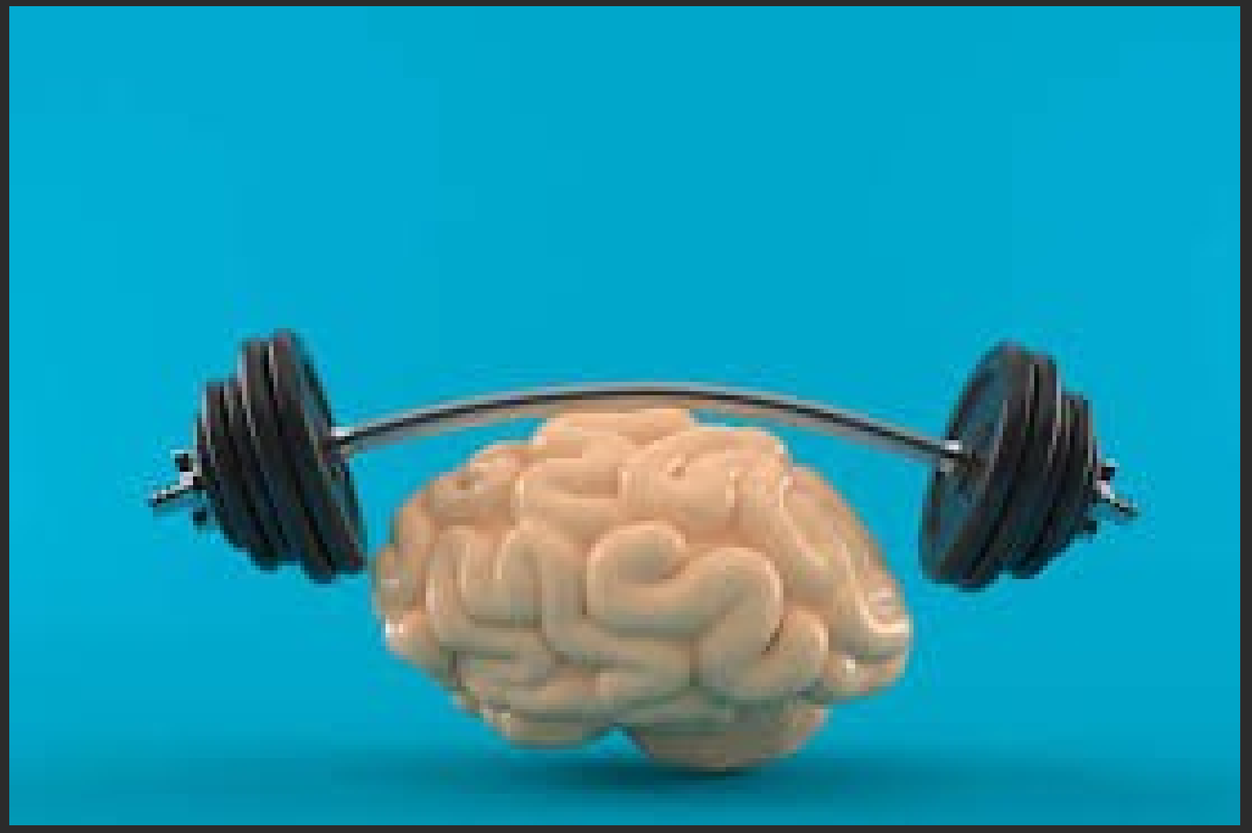
```
localhost:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.1     0.0.0.0         UG    202    0        0 eth0
172.17.0.0      *               255.255.0.0     U     0      0        0 docker0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
```

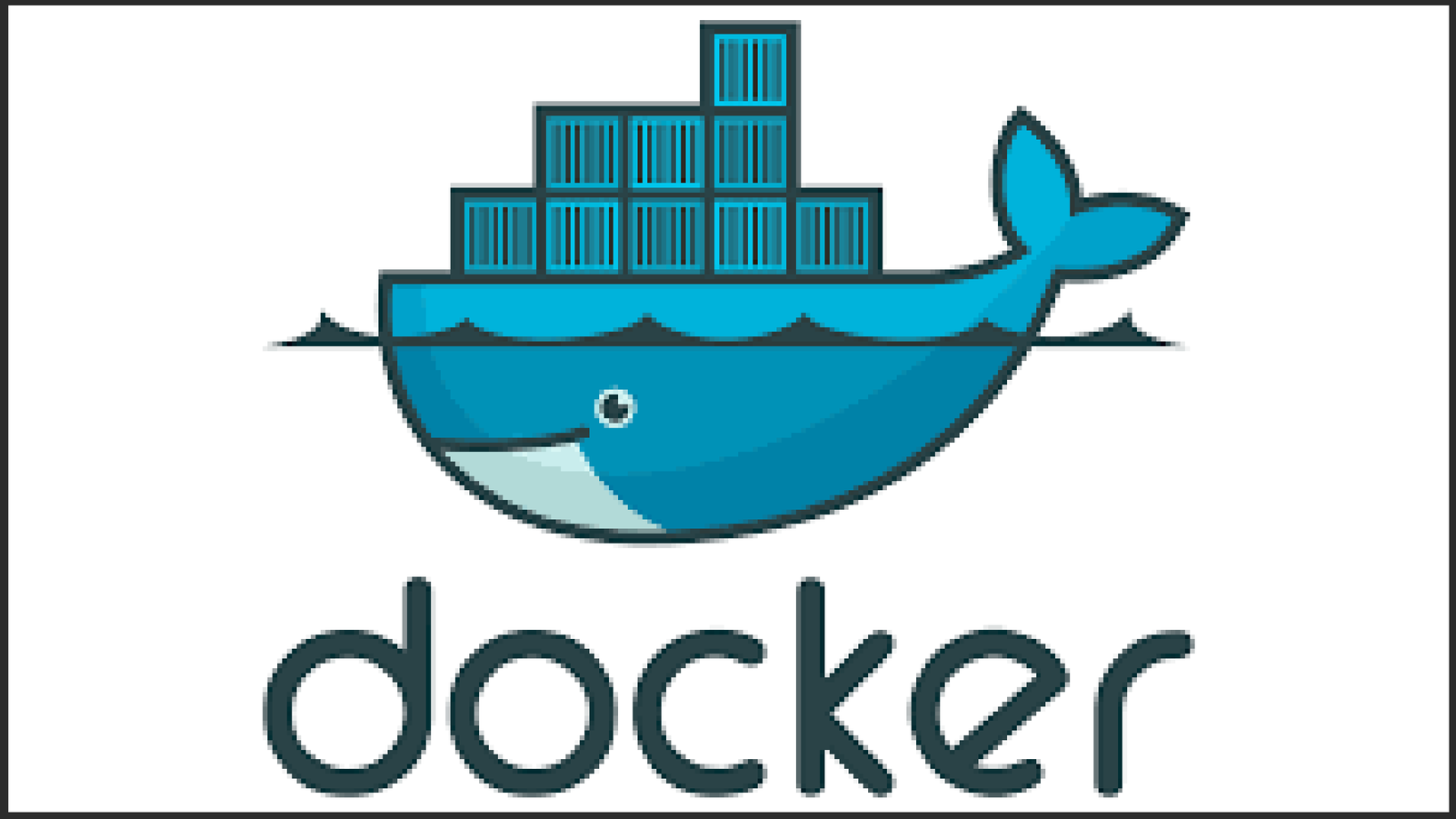
Comandos

- # iptables -L -n

```
localhost:~# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER-USER  all  --  0.0.0.0/0            0.0.0.0/0
DOCKER-ISOLATION-STAGE-1  all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER (1 references)
target     prot opt source               destination

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
target     prot opt source               destination
DOCKER-ISOLATION-STAGE-2  all  --  0.0.0.0/0            0.0.0.0/0
RETURN     all  --  0.0.0.0/0            0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
target     prot opt source               destination
DROP       all  --  0.0.0.0/0            0.0.0.0/0
RETURN     all  --  0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
target     prot opt source               destination
RETURN     all  --  0.0.0.0/0            0.0.0.0/0
```

! Vamos a Practicar !



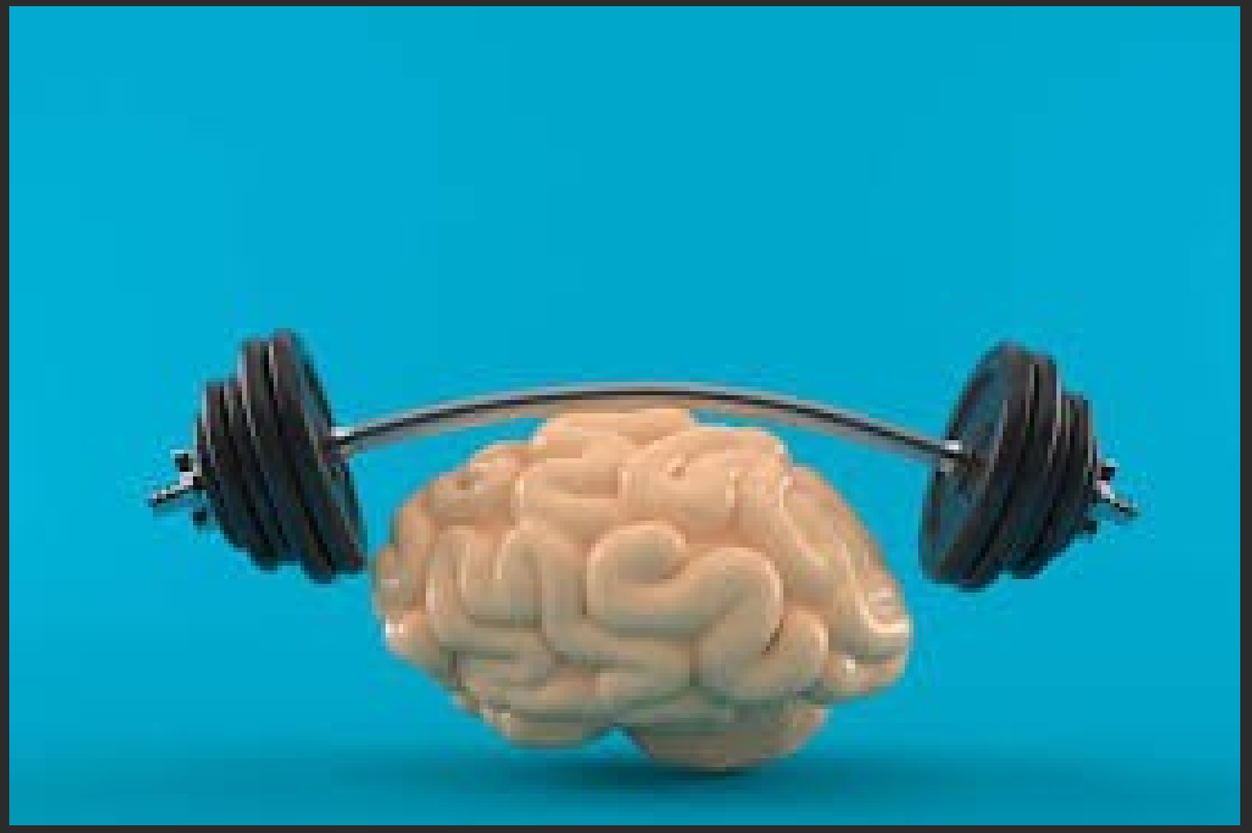


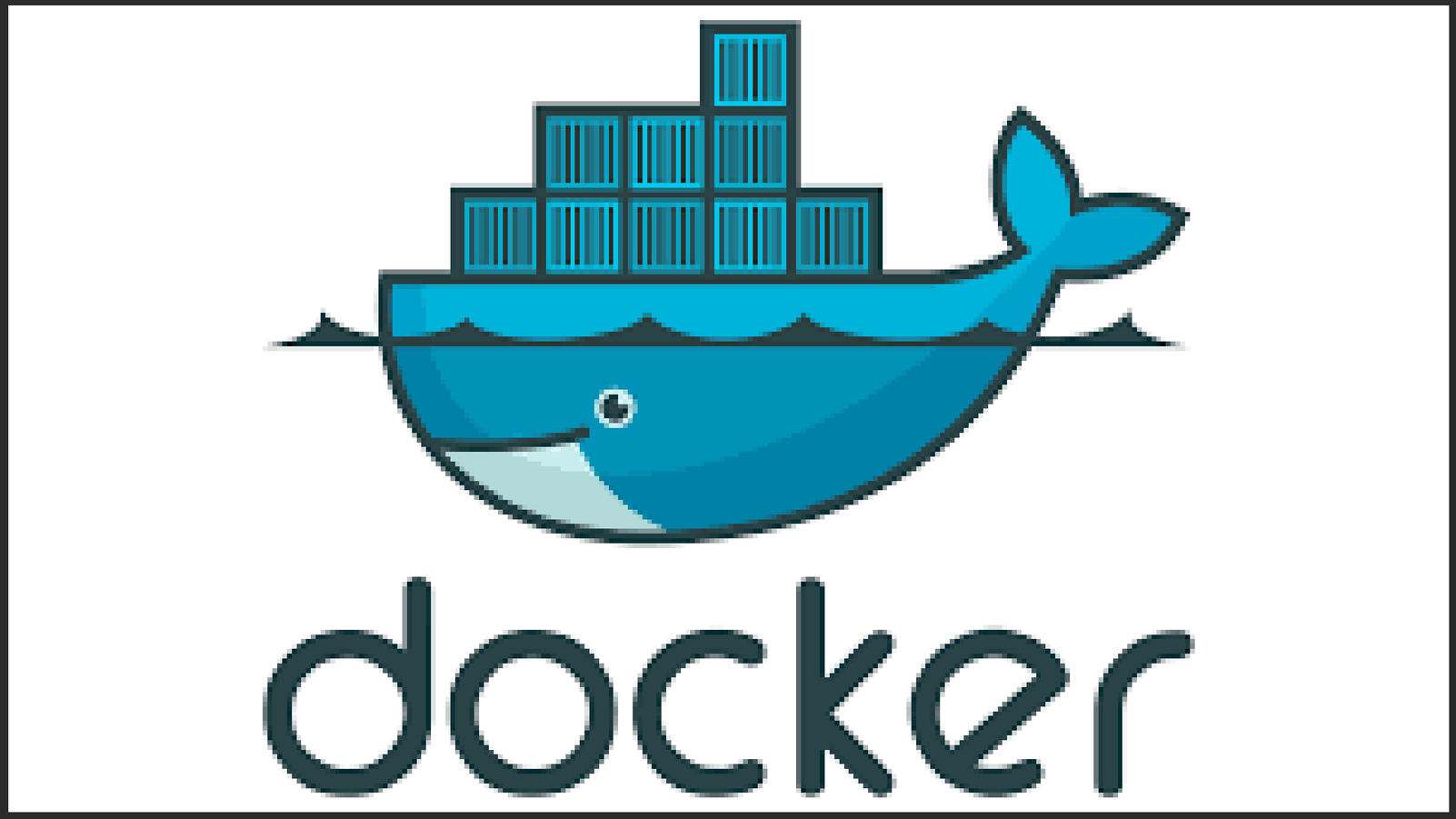
Comandos Básicos

Comandos Básicos

- $ docker
- $ docker version
- $ docker info
- o
- $ docker system info
- $ docker system df
- $ docker system events

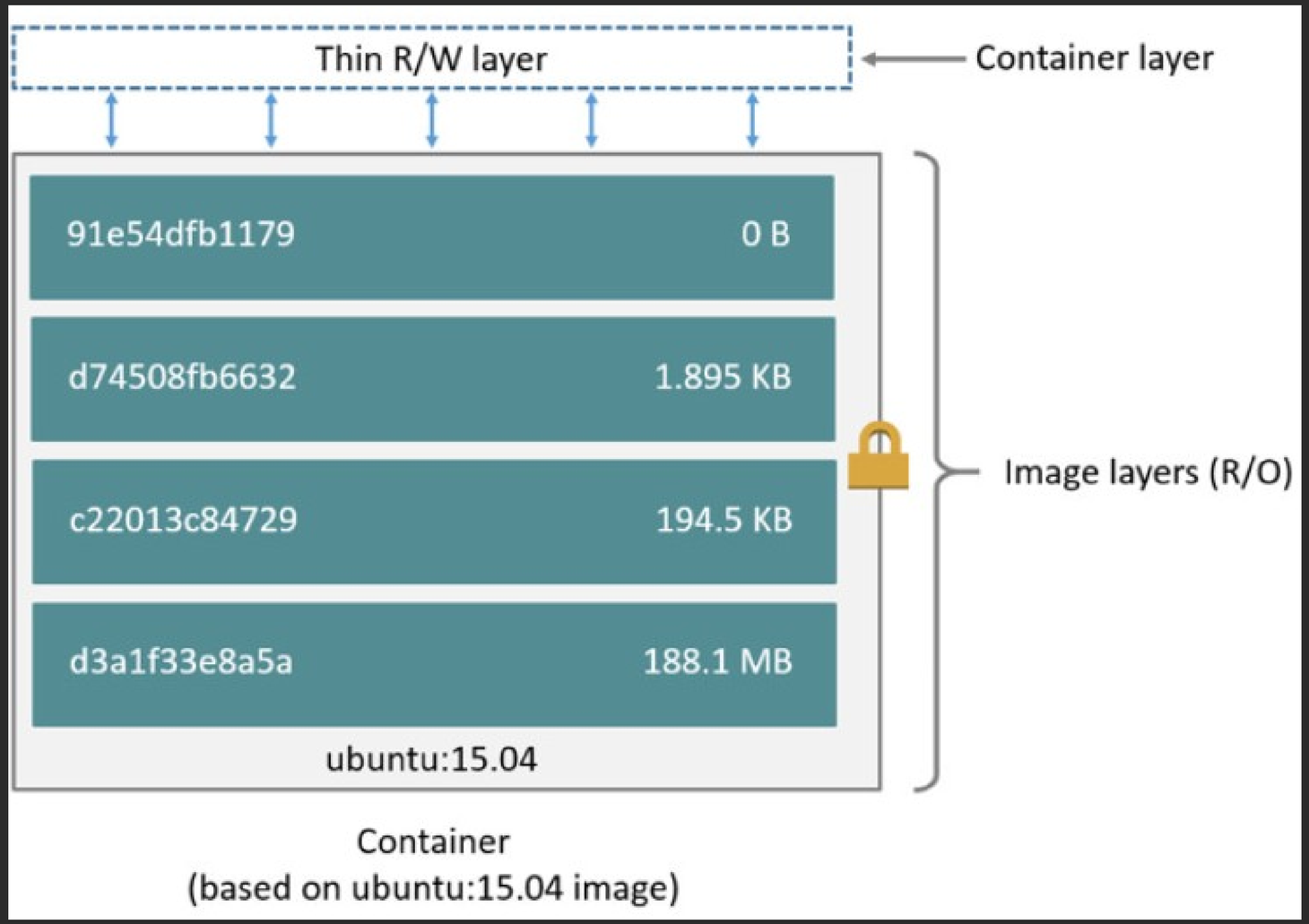
! Vamos a Practicar !



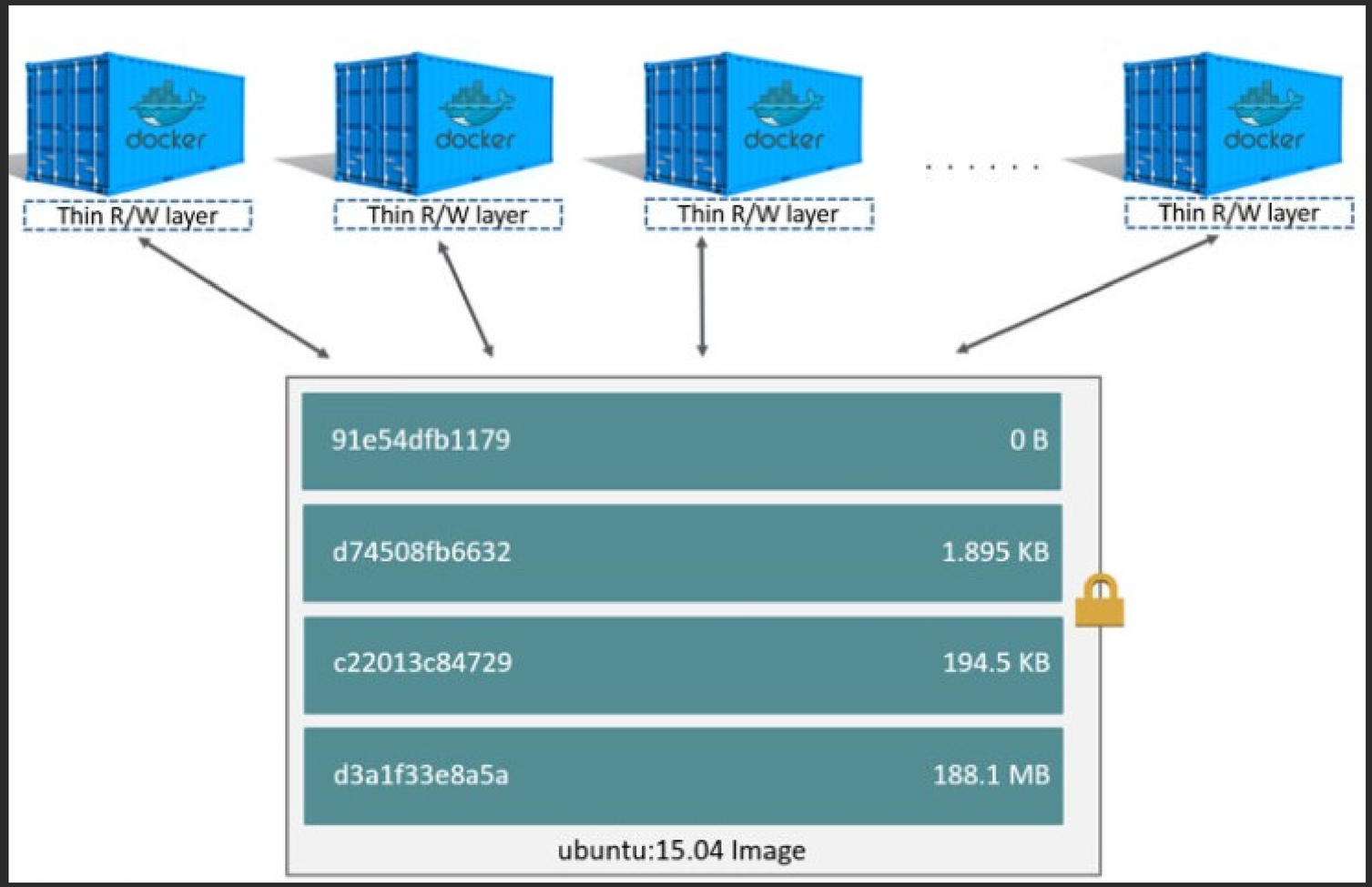


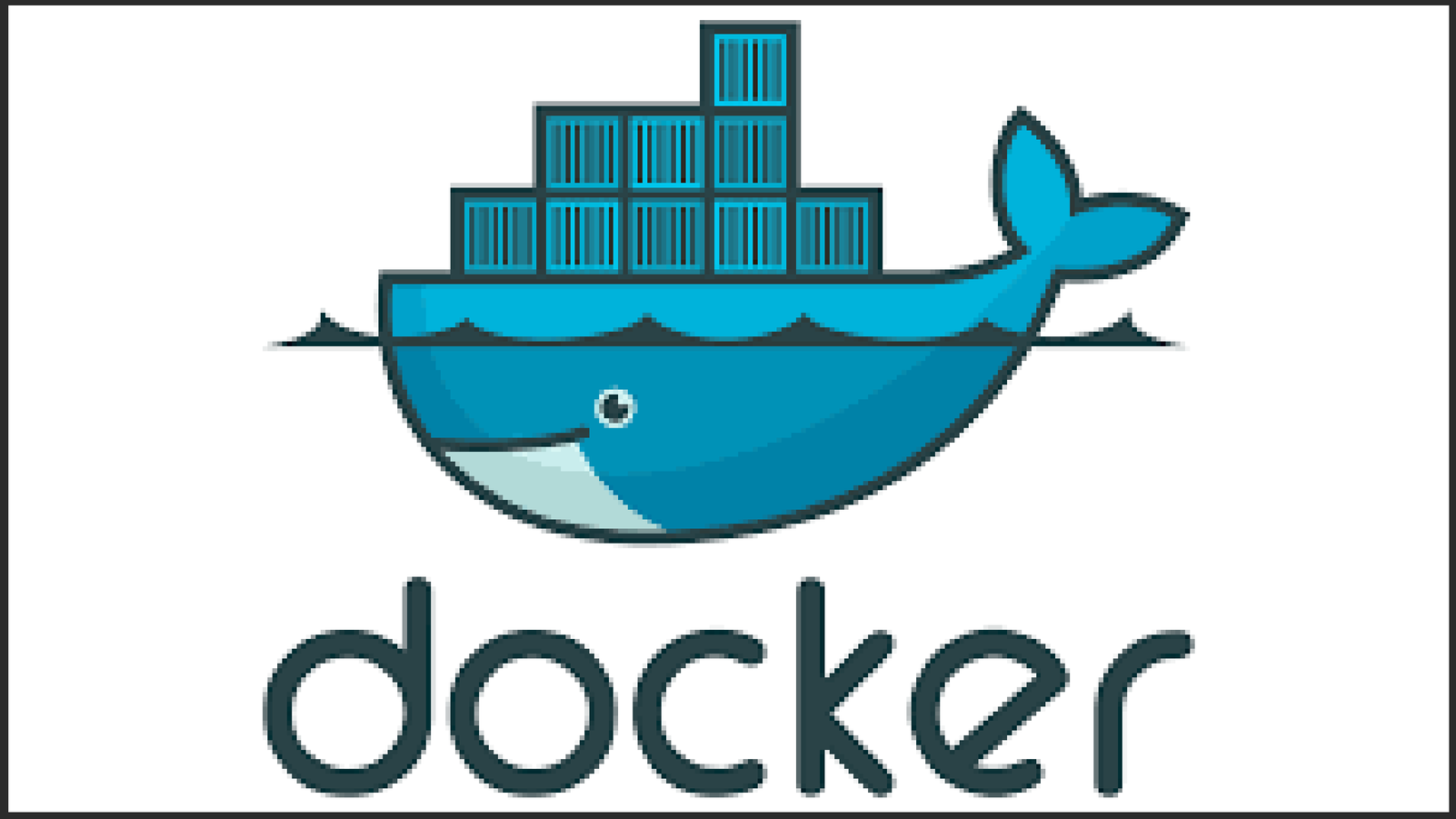
Imágenes y Capas

Imagenes y Capas



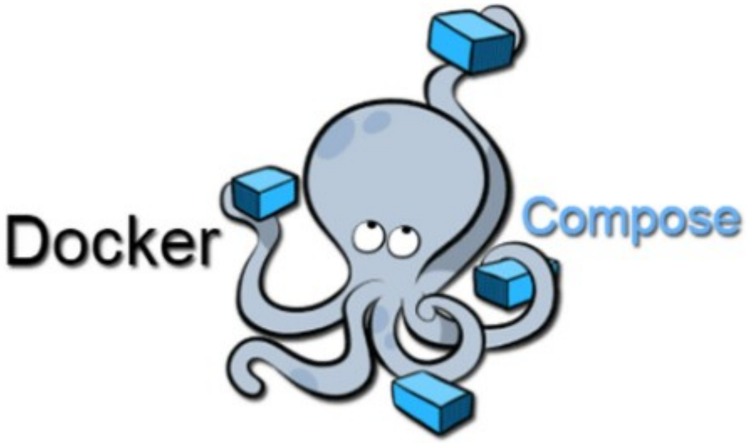
Imagenes y Capas





Herramientas


docker

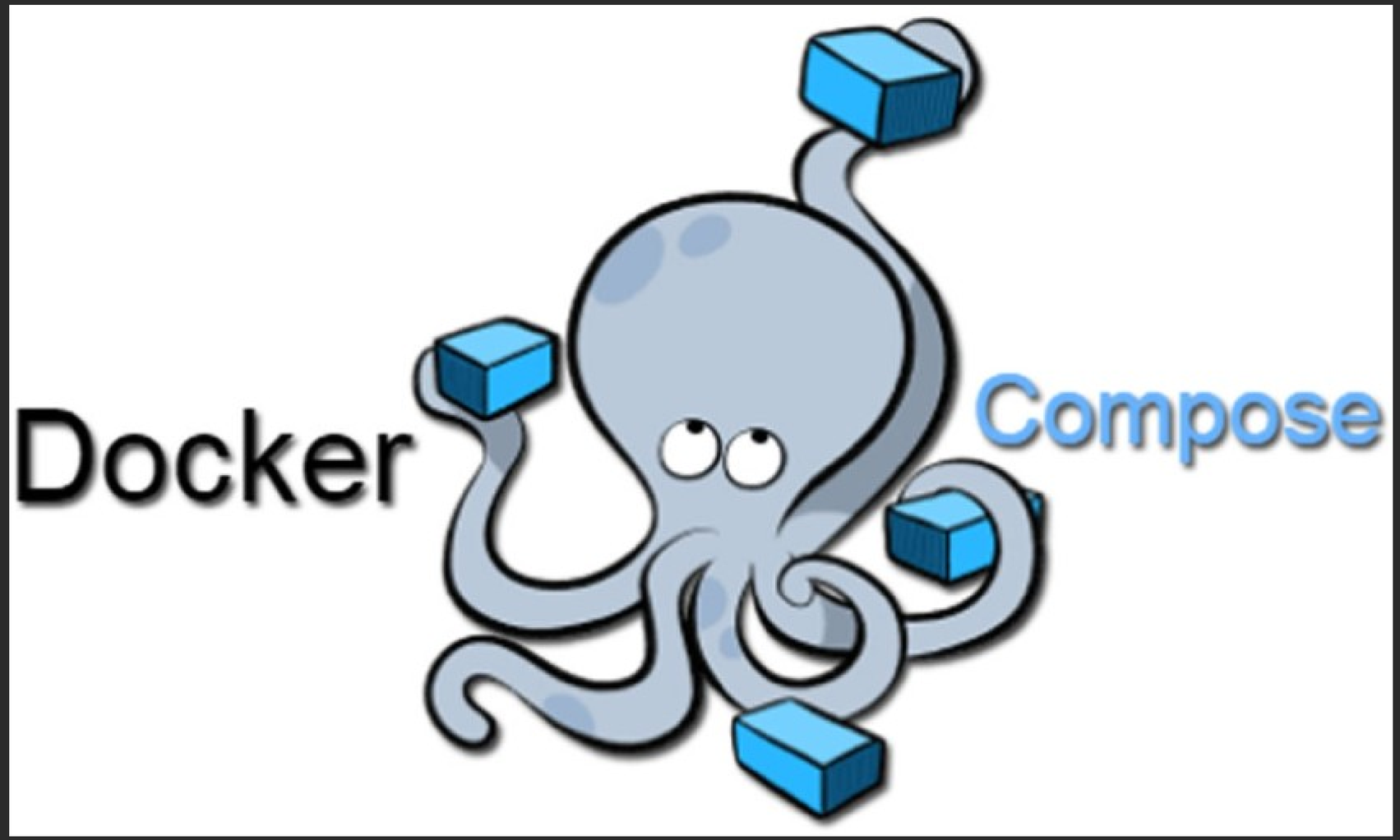

Docker
Compose


docker
REGISTRY


dockerhub


docker
SWARM


docker
MACHINE


Docker
Compose

Docker Compose

- Es una herramienta que nos permite simplificar el uso de Docker, generando scripts que facilitan el diseño y la construcción de servicio.
- De está forma podemos crear distintos contenedores y en cada contenedor crearemos diferentes servicios.


docker
REGISTRY
TM & © 2015 Docker, Inc.

Docker Registry

- Un Registry es un lugar donde almacenar las imágenes de contenedores que luego utilizaremos para crear nuestros contenedores.

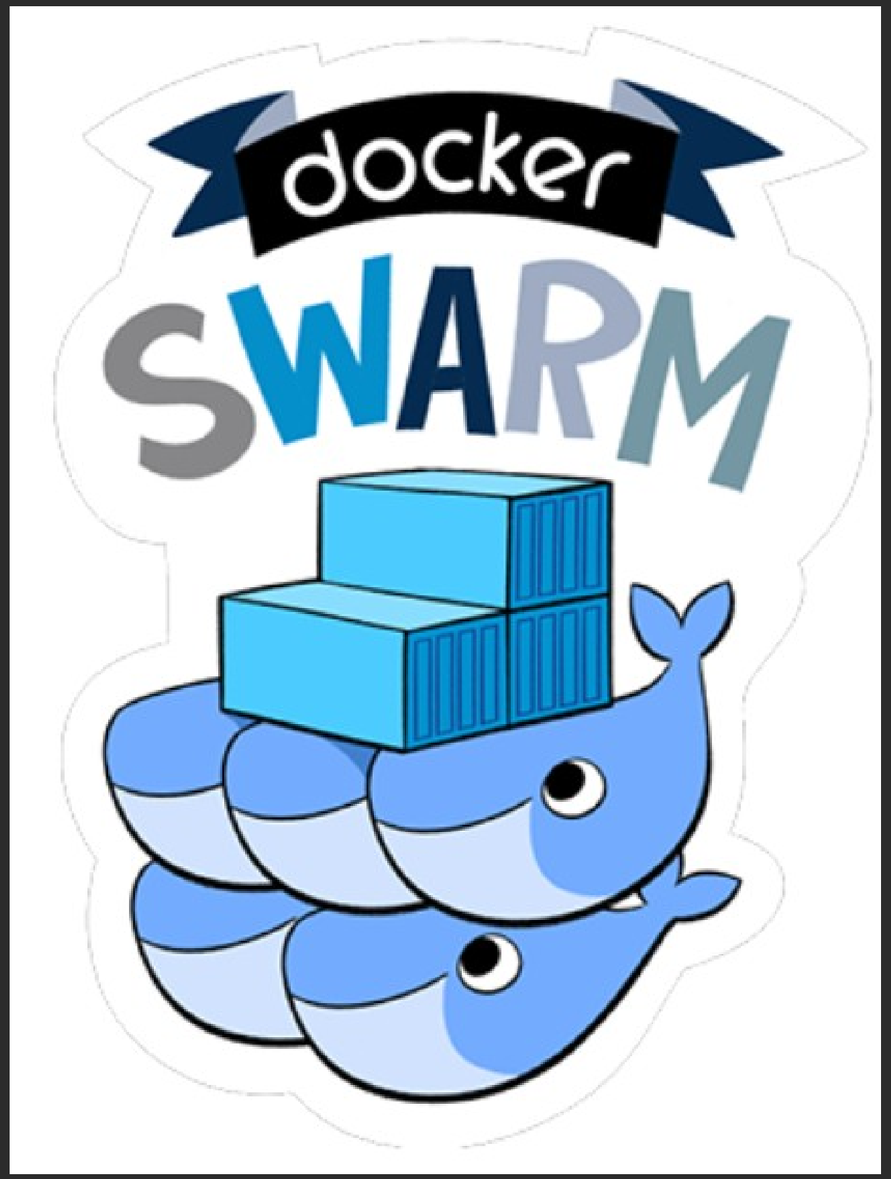

docker hub

Docker Hub

Docker también tiene un registro público gratuito, Docker Hub, que puede alojar sus imágenes de Docker personalizadas, pero existen situaciones en las que no le convendrá que su imagen esté disponible públicamente. Las imágenes normalmente contienen todo el código necesario para ejecutar una aplicación. Por lo tanto, usar un registro privado es preferible cuando se utiliza software propio.

Docker Hub

```
$ docker login
```


docker
SWARM

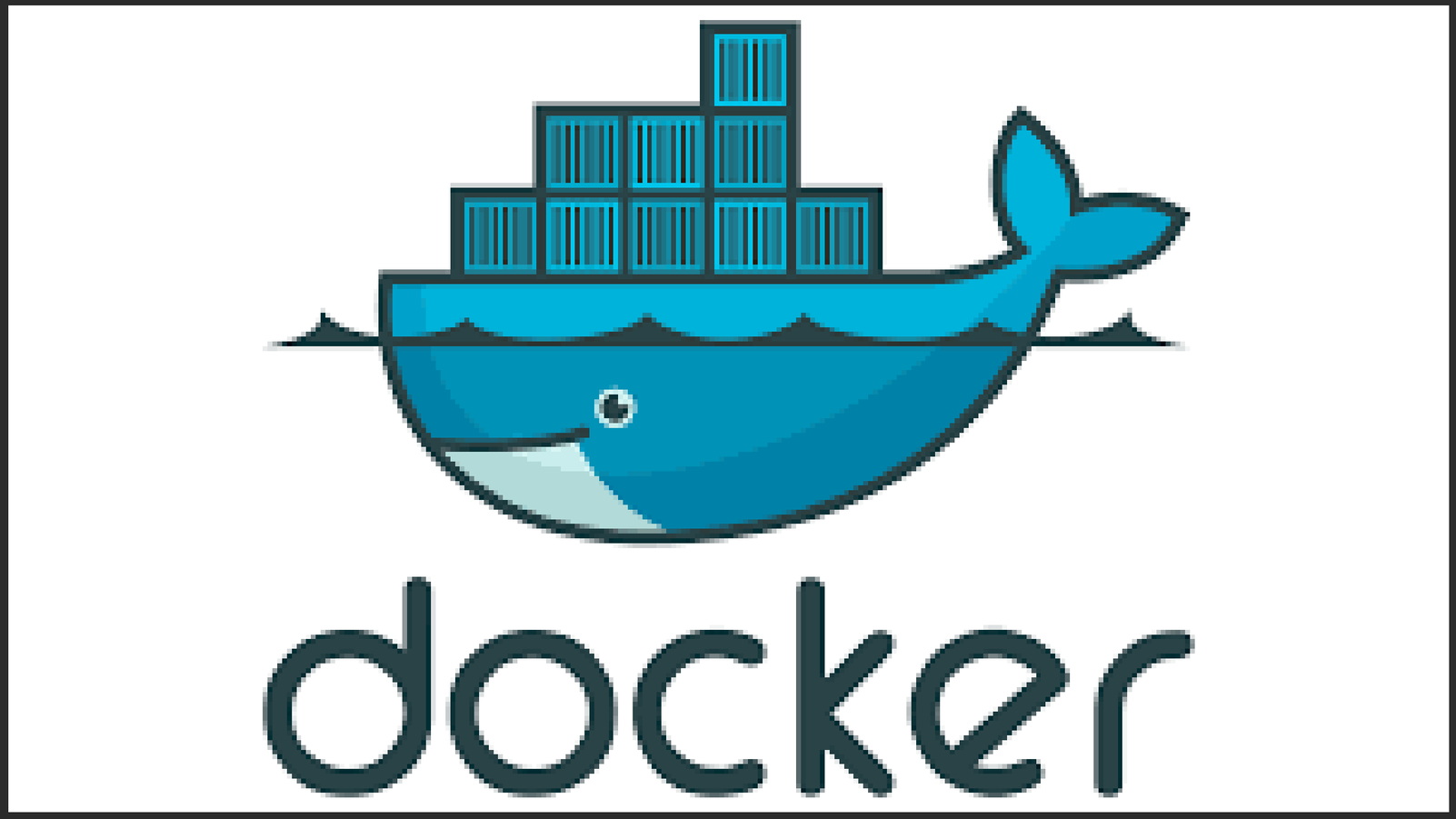
Docker Swarm

Es una herramienta que permite a los desarrolladores implementar contenedores en modo swarm. Un clúster Swarm consiste en múltiples nodos. Los nodos de administración realizan la orquestación y la administración del clúster.


docker
MACHINE

Docker Machine

Es una herramienta que nos ayuda a crear, configurar y manejar máquinas (virtuales o físicas) con Docker. Este utiliza distintos drivers que nos permiten crear y configurar Docker en distintos entornos y proveedores, por ejemplo Virtualbox, AWS, VMWARE, etc.



Imágenes

¿ Qué son las imágenes ?

- Es una especie de plantilla, es una imagen que contiene un Sistema Operativo base GNU/Linux, también puede contener servicios, como por ejemplo Apache, Samba, etc.
- Cada archivo de imagen de Docker se compone de una serie de capas. Estas capas que se combinan en una sola imagen. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un commando, como ejecutar o copiar, se crea una nueva capa.
- Docker utiliza estas capas para construer nuevos contenedores, lo cual hace mucho más ráido el proceso de construcción.
- Los sistemas operativos son la base minima para poder ser utilizadas, para crear los contenedores. Las imagines de solo lectura y estos nunca cambian.
- Hay muchas imágenes públicas con elementos básicos, que podemos tomar de base para agregar las aplicaciones necearias o utilizer imagines ya preparadas como la base de datos mariadb, apache, samba, etc.

¿ Qué son las imágenes ?

- Las imágenes se identifican por un ID, y un par nombre-version, por ejemplo: ubuntu:18.04, ubuntu:latest, ubuntu, etc.
- Las imagines las bajamos desde hub.docker.com, que como comentamos tenemos las imagines publicas y algunas privadas.

Docker Image

Realizamos las búsquedas de las imágenes:

$ docker search ubuntu

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys…	12386	[OK]	
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface …	541		[OK]
websphere-liberty	WebSphere Liberty multi-architecture images …	273	[OK]	
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi…	253		[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session…	241		[OK]
ubuntu-upstart	Upstart is an event-based replacement for th…	110	[OK]	
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	50		[OK]
open-liberty	Open Liberty multi-architecture images based…	46	[OK]	
ubuntu-debootstrap	debootstrap --variant=minbase --components=m…	44	[OK]	

Docker Image

Realizamos la bajada de una imagen:

```
$ docker pull ubuntu
```

O

```
$ docker pull ubuntu:latest
```

O

```
$ docker pull ubuntu:18.04
```

Docker Image

Ver las imágenes que tenemos bajadas.

$ docker image ls

O

$ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sherloq	latest	478b5cd2b823	24 hours ago	5.33GB
metafinder	latest	e0867ba35d07	4 days ago	1.09GB
gobuster	3.1.0	5e9d19a42482	6 days ago	14.2MB
emailfinder	latest	f4e2bf97e9af	7 days ago	1.03GB
phpmyadmin	latest	48880736f710	12 days ago	474MB
golang	latest	ee23292e2826	12 days ago	862MB
ghcr.io/linuxserver/plex	latest	088913d59c31	2 weeks ago	664MB
nginx	1.0	e68b5bd5f9c0	2 weeks ago	161MB
dnsenum	latest	ab6da8a9f522	2 weeks ago	180MB

Docker Image

Ver el historial de una imagen.

$ docker image history ubuntu

```
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
7e0aa2d69a15   7 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      7 weeks ago    /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      7 weeks ago    /bin/sh -c [ -z "$(apt-get indextargets)" ]     0B
<missing>      7 weeks ago    /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      7 weeks ago    /bin/sh -c #(nop) ADD file:5c44a80f547b7d68b…   72.7MB
```

Docker Image

Inspeccionar la imagen.

$ docker image inspect ubuntu


```
[
    {
        "Id": "sha256:7e0aa2d69a153215c790488ed1fcec162015e973e49962d438e18249d16fa9bd",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:adf73ca014822ad8237623d388cedf4d5346aa72c270c5acc01431cc93e18e2d",
            "ubuntu@sha256:cf31af331f38d1d7158470e095b132acd126a7180a54f263d386da88eb681d93"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-04-23T22:21:37.494427352",
        "Container": "8be4461fbc3e2d9db869e63a5e140b10a085b1799c12e8a83b32aba9344eba78",
        "ContainerConfig": {
            "Hostname": "8be4461fbc3e",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
```

Docker Image

Cambiar el nombre de la imagen, versión y subir a docker hub.

$ docker image tag ubuntu marcospr1974/miubuntu:latest

$ docker login -> Tener creado en hub.docker.com un usuario.

$ docker image push marcospr1974/miubuntu:latest

Docker Image

Realizar backups de todas las imágenes con nombre ubuntu sin importar la versión.

$ docker image save ubuntu –o ubuntu_latest.tar

Cargar todas las imagenes que realizamos los backups.

$ docker image load –i ubuntu_latest.tar

Docker Image

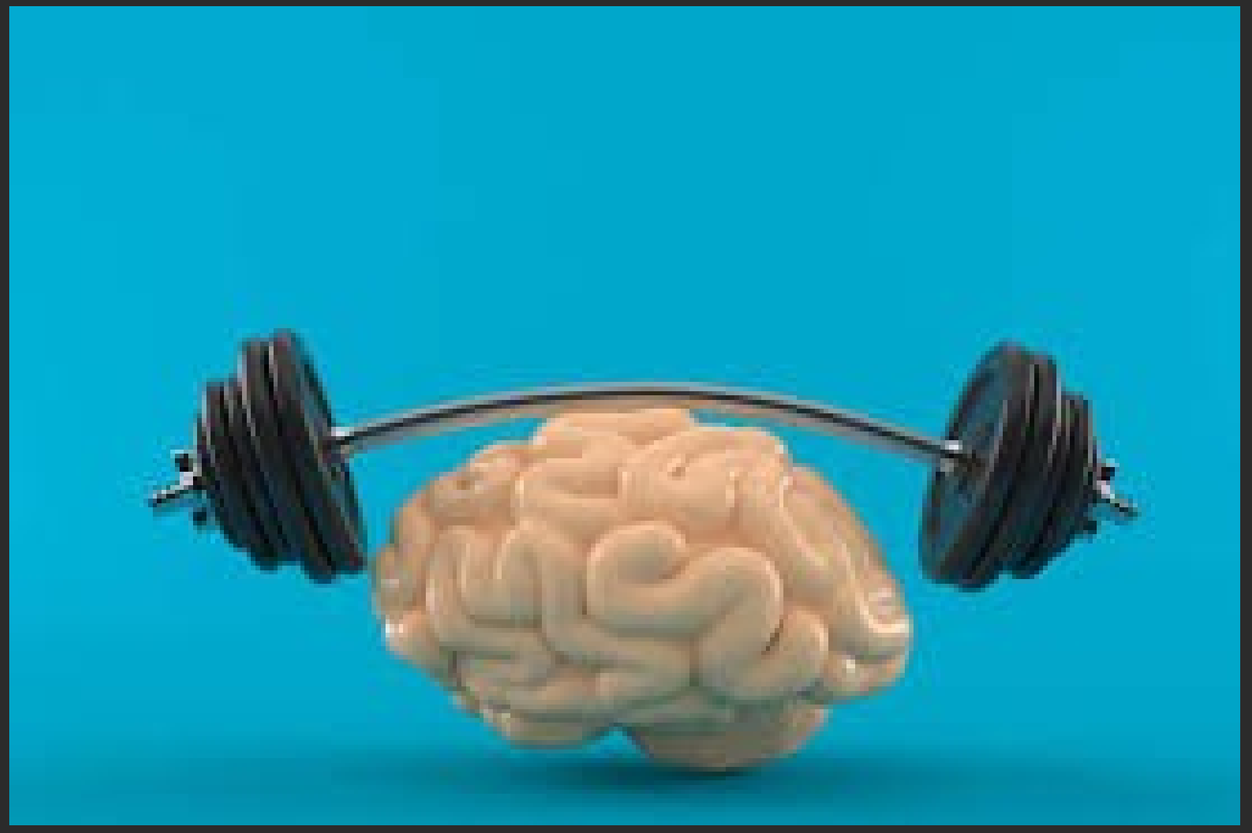
Borrar una image.

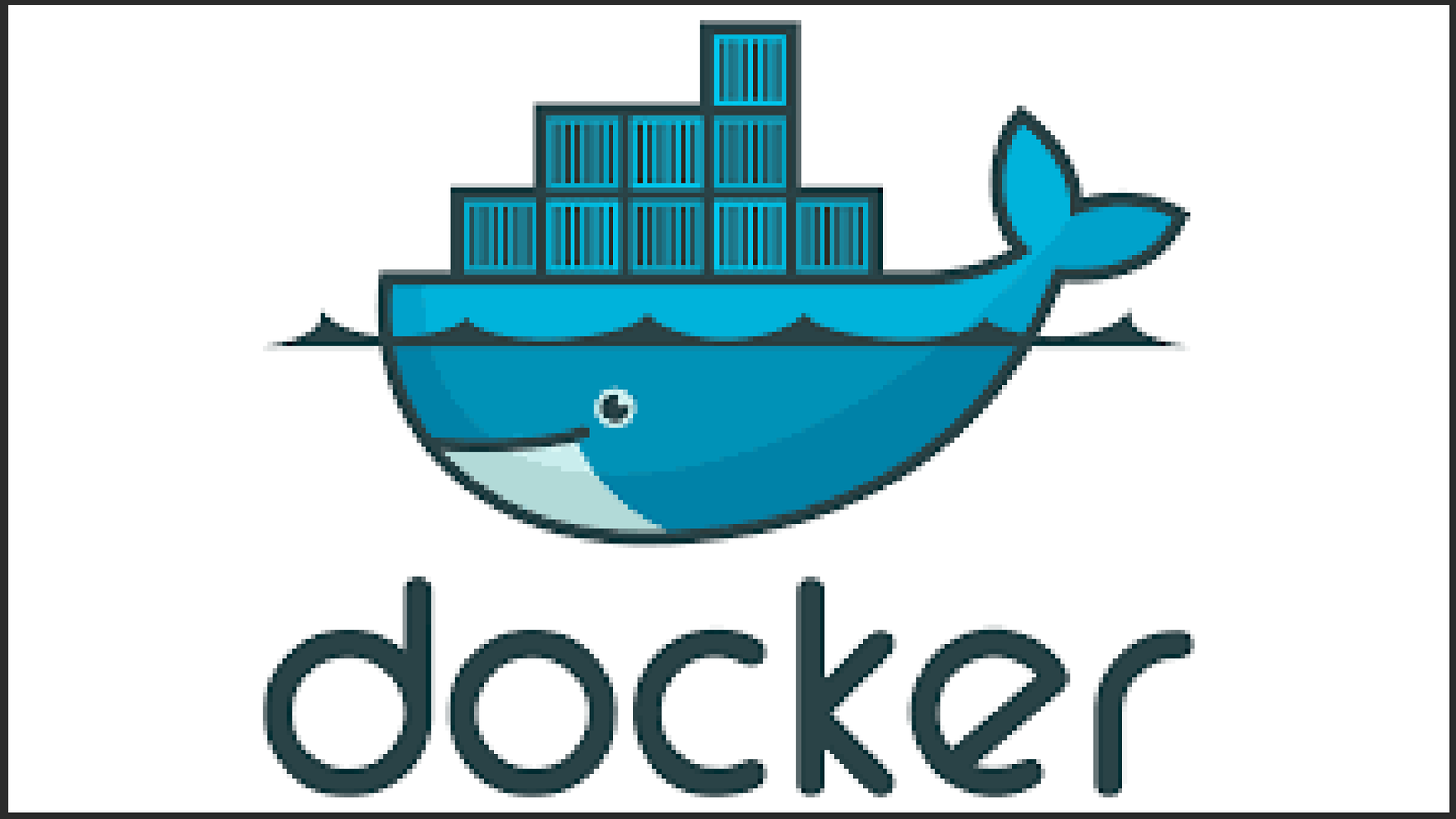
$ docker image rmi ubuntu

Borrar las imágenes que indican como nombre y versión <none>.

$ docker image prune

! Vamos a Practicar !





Containers

Docker Containers

Son un conjunto de tecnologías que juntos forman un contenedor (de Docker), esté conjunto de tecnologías se llaman:

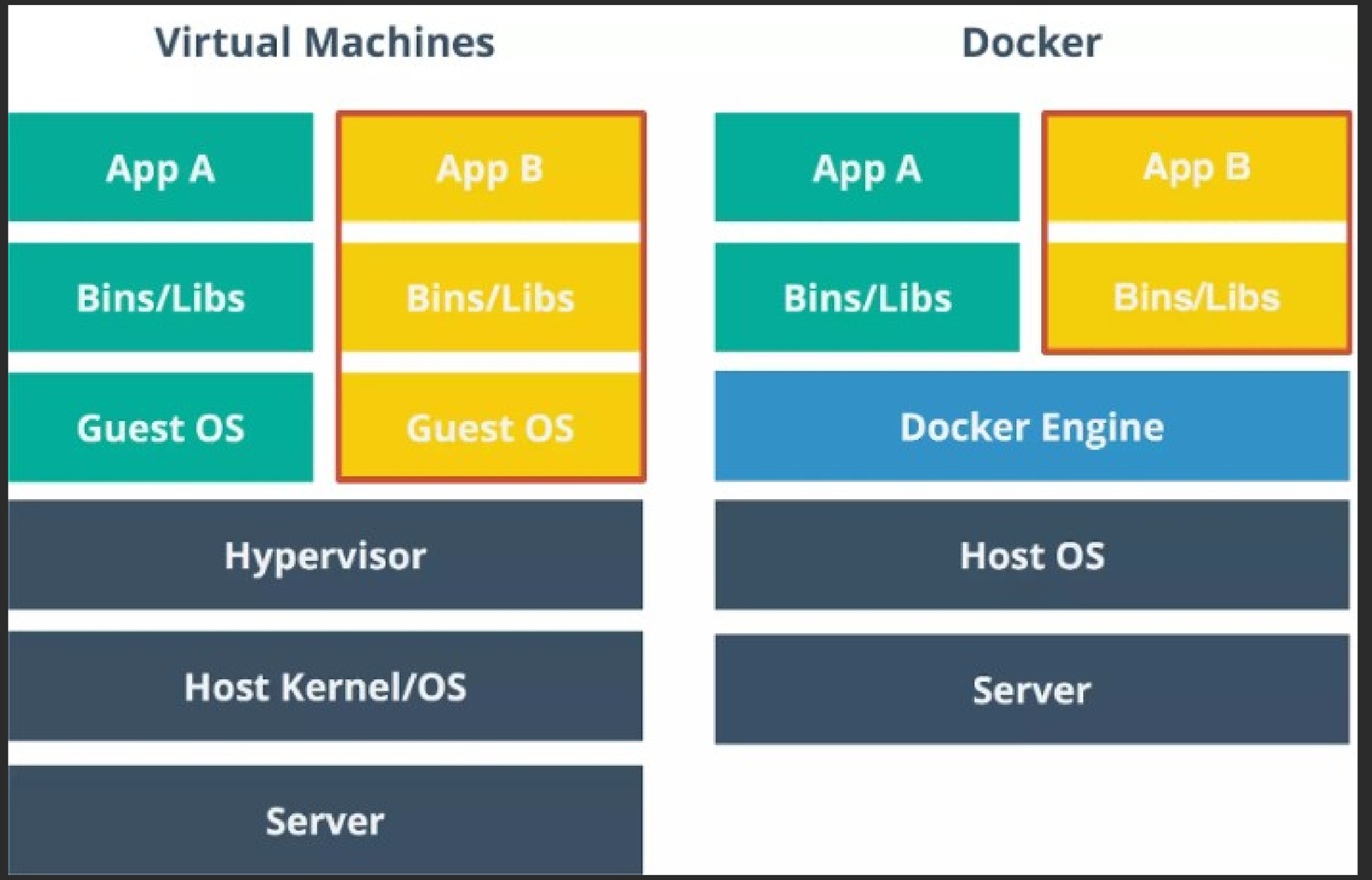
- **Namespaces**: Permite a la aplicación que corre en un contenedor de Docker tener una vista de los recursos del sistema operativo.
- **Cgroups**: Permite limitar y medir los recursos que se encuentran disponibles en el sistema operativo.
- **Chroot**: Permite tener en el contenedor una vista de un sistema “falso” para el mismo, es decir, crea su propio entorno de ejecución con su propio root y home.

Docker Containers

Algunas de las características más notables de un contenedor son:

- Los contenedores son más livianos (ya que trabajan directamente sobre el Kernel) que las maquinas virtuales.
- No es necesario instalar un sistema operativo por contenedor.
- Menor uso de los recursos de la máquina.
- Mayor cantidad de contenedores por equipo físico.
- Mejor portabilidad.

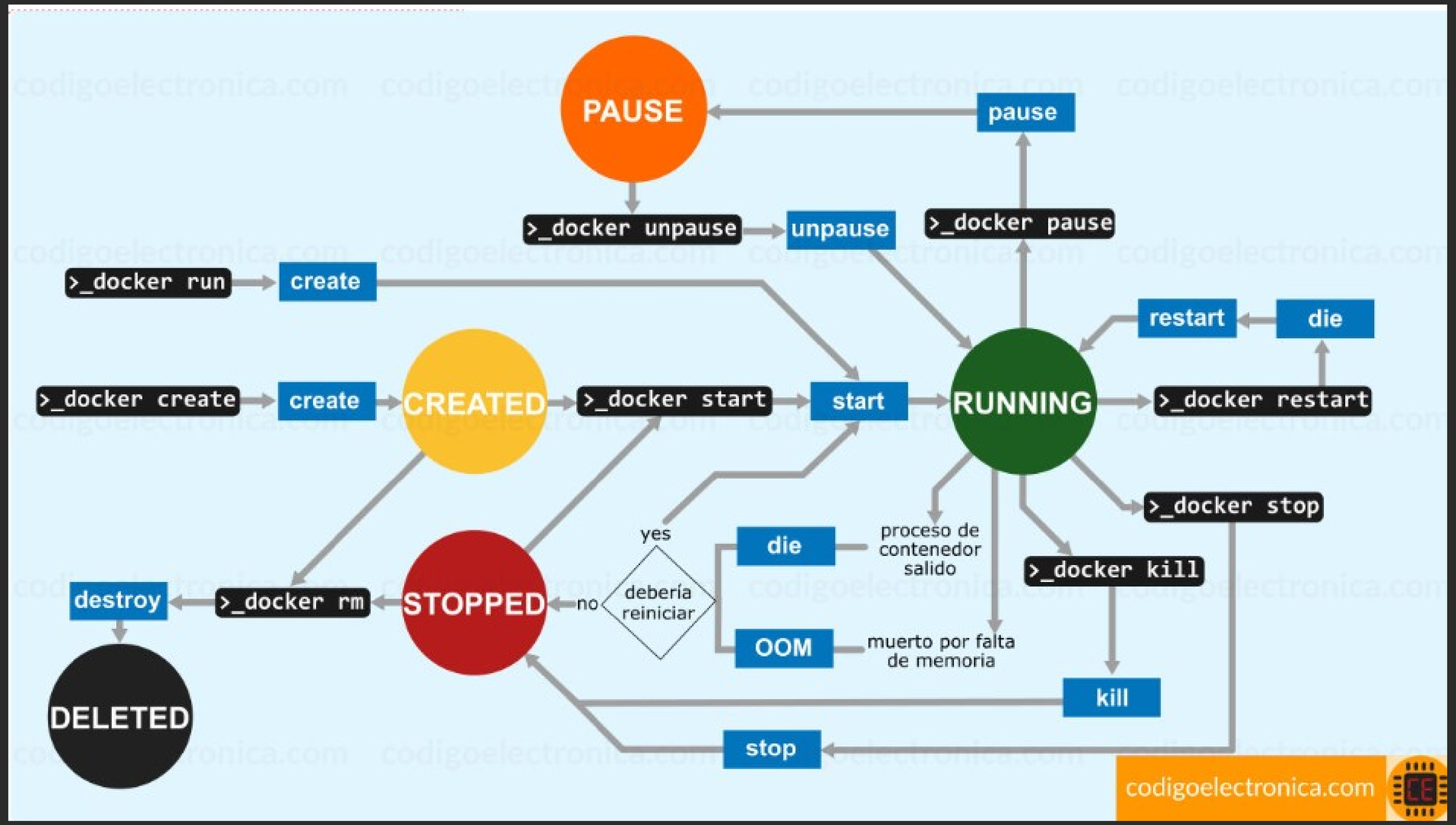
Docker Containers



Docker Containers – Ciclo de Vida

- **created** : el contenedor ha sido creado, pero nunca ha sido ejecutado.
- **restarting** : el contenedor se está reiniciando.
- **running** : el contenedor está en ejecución.
- removing : el contenedor está siendo borrado.
- **paused** : el contenedor está en modo suspendido.
- **exited** : el proceso principal del contenedor ha sido parado.
- **dead** : el contenedor está siendo borrado, pero por algún motivo, recursos ocupados, etc, éste no se ha podido borrar. La mayor diferencia entre exited y dead, es que un contenedor en estado exited se puede volver a arrancar y el dead no.

Docker Containers – Ciclo de Vida



Armado de un Container

$ docker container run –ti ubuntu /bin/bash

El comando docker run primero crea una capa de contenedor grabable sobre la imagen especificada y luego la inicia usando el comando especificado. Es decir, la ejecución de Docker es equivalente a la API /containers/create then /containers/(id)/start. Un contenedor detenido se puede reiniciar con todos sus cambios anteriores intactos usando docker start.

Armado de un Container

$ docker container pause ID_CONTAINER

El comando docker pause suspende todos los procesos en los contenedores especificados. En Linux, esto usa el congelador cgroup. Tradicionalmente, cuando se suspende un proceso, se utiliza la señal SIGSTOP, que se puede observar cuando el proceso se suspende. Con el congelador cgroup, el proceso no es consciente, y no puede capturar, que se está suspendiendo, y posteriormente se reanuda. En Windows, solo se pueden pausar los contenedores de Hyper-V.

Armado de un Container

$ docker container unpause ID_CONTAINER

El comando docker unpause anula la suspensión de todos los procesos en los contenedores especificados. En Linux, hace esto usando el congelador cgroup

Armado de un Container

$ docker container ps

$ docker container ps –a

Lista todos los contenedores.

Armado de un Container

$ docker container stop ID_CONTAINER

Parar uno o mas contenedores.

Armado de un Container

$ docker container kill ID_CONTAINER

El subcomando kill docker mata uno o más contenedores. El proceso principal dentro del contenedor se envía señal SIGKILL (predeterminado), o la señal que se especifica con la opción --signal. Puede eliminar un contenedor utilizando la ID, el prefijo de identificación o el nombre del contenedor.

Armado de un Container

$ docker container attach ID_CONTAINER

Sumarse a un container en ejecución.

Armado de un Container

$ docker container rename ID_CONTAINER NUEVO-NOMBRE

Renombrar un container.

Armado de un Container

$ docker container rm ID_CONTAINER

Retire uno o más contenedores.

Armado de un Container

$ docker container prune

Borrar uno o mas container en estado Exit (stop).

Armado de un Container

$ docker container inspect ID_CONTAINER

Inspeccionar la configuración del container.

Armado de un Container

$ docker container export ID_CONTAINER –o CONTAINER.tar

Exportar la estructura del container, exporta la estructura del Sistema operativo.

Armado de un Container

$ docker container logs ID_CONTAINER

$ docker container logs -f ID_CONTAINER

Ver los logs del container.

Armado de un Container

$ docker container cp ARCHIVO ID_CONTAINER:/Directorio

Copiar un archivo local al container.

Armado de un Container

$ docker container exec ID_CONTAINER COMANDO

Ejecutar comandos dentro del container.

Armado de un Container

$ docker container stats ID_CONTAINER

Estado de la memoria, cpu, etc de un container en ejecución.

```
CONTAINER ID   NAME        CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O     PIDS
f56f998001da   wordpress   0.02%     23.27MiB / 7.656GiB   0.30%     2.81MB / 0B   22.9MB / 0B   6
```

Armado de un Container

$ docker container top ID_CONTAINER

Realiza un top, de los procesos que se están ejecutando en el container.

```
UID                 PID                 PPID                C                   STIME               TTY
                TIME                CMD
root                4637                4054                0                   jul07               ?
                00:00:08            apache2 -DFOREGROUND
www-data            8459                4637                0                   jul07               ?
                00:00:00            apache2 -DFOREGROUND
www-data            8465                4637                0                   jul07               ?
                00:00:00            apache2 -DFOREGROUND
www-data            8466                4637                0                   jul07               ?
                00:00:00            apache2 -DFOREGROUND
www-data            8474                4637                0                   jul07               ?
                00:00:00            apache2 -DFOREGROUND
www-data            8476                4637                0                   jul07               ?
                00:00:00            apache2 -DFOREGROUND
```

Armado de un Container

$ docker container commit ID_CONTAINER REPOSITORY:TAG

Realizar un commit, sumando la imagen + el container, generando una nueva imagen.

Ejecutando una Base de Datos

```
$ mkdir –p Docker/mariadb/db
$ cd Docker/mariadb
$ docker run \
      --name mariadb \
      --hostname mariadb \
      -e MARIADB_ROOT_PASSWORD=secret \
      -e MARIADB_DATABASE=wordpress \
      -e MARIADB_USER=wordpress \
      -e MARIADB_PASSWORD=wpdebian \
      -p 3306:3306/tcp \
      -v $PWD/db:/var/lib/mysql \
      -d mariadb:latest
```

- Password del usuario root
- Nombre de la base de datos
- Usuario de la base de datos
- Password del usuario wordpress
- Puerto Origen:Puerto Destino
- Directorio donde guarda la base de datos
- Repositorio:Version

Ejecutando una Base de Datos

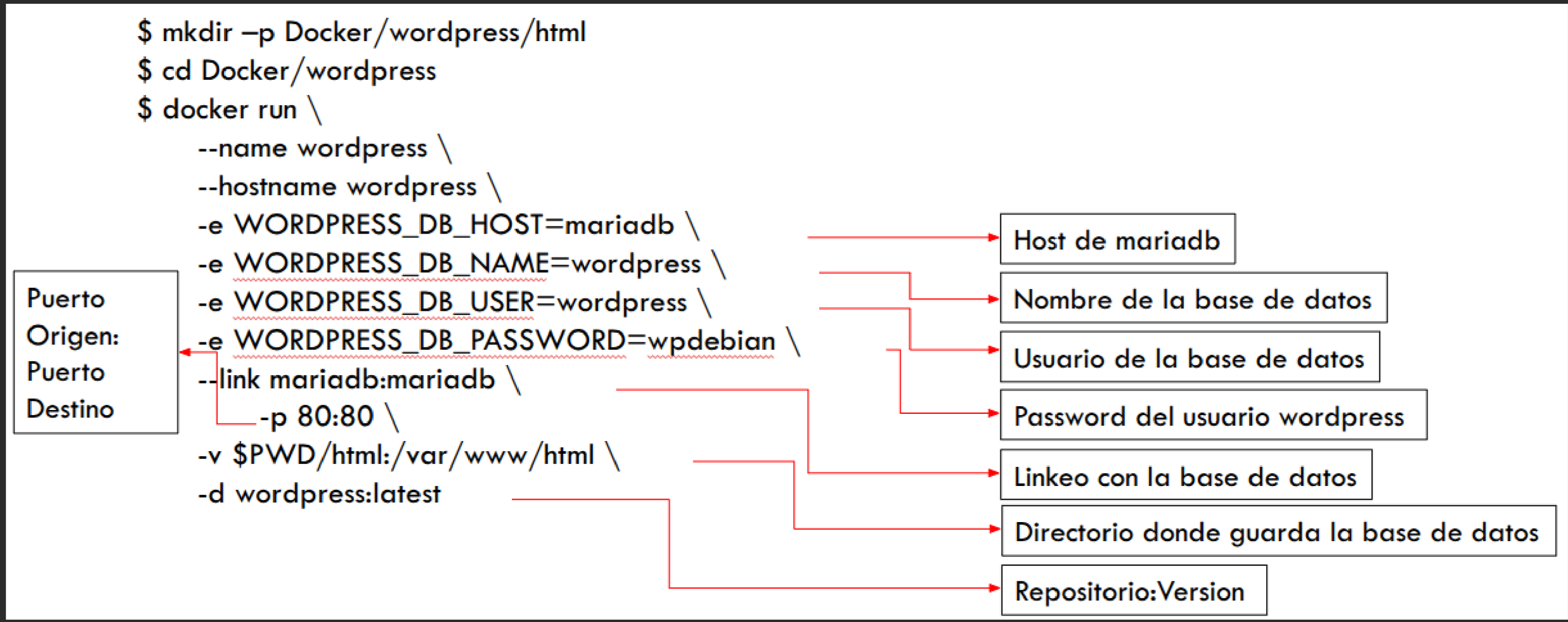
$ docker container ps

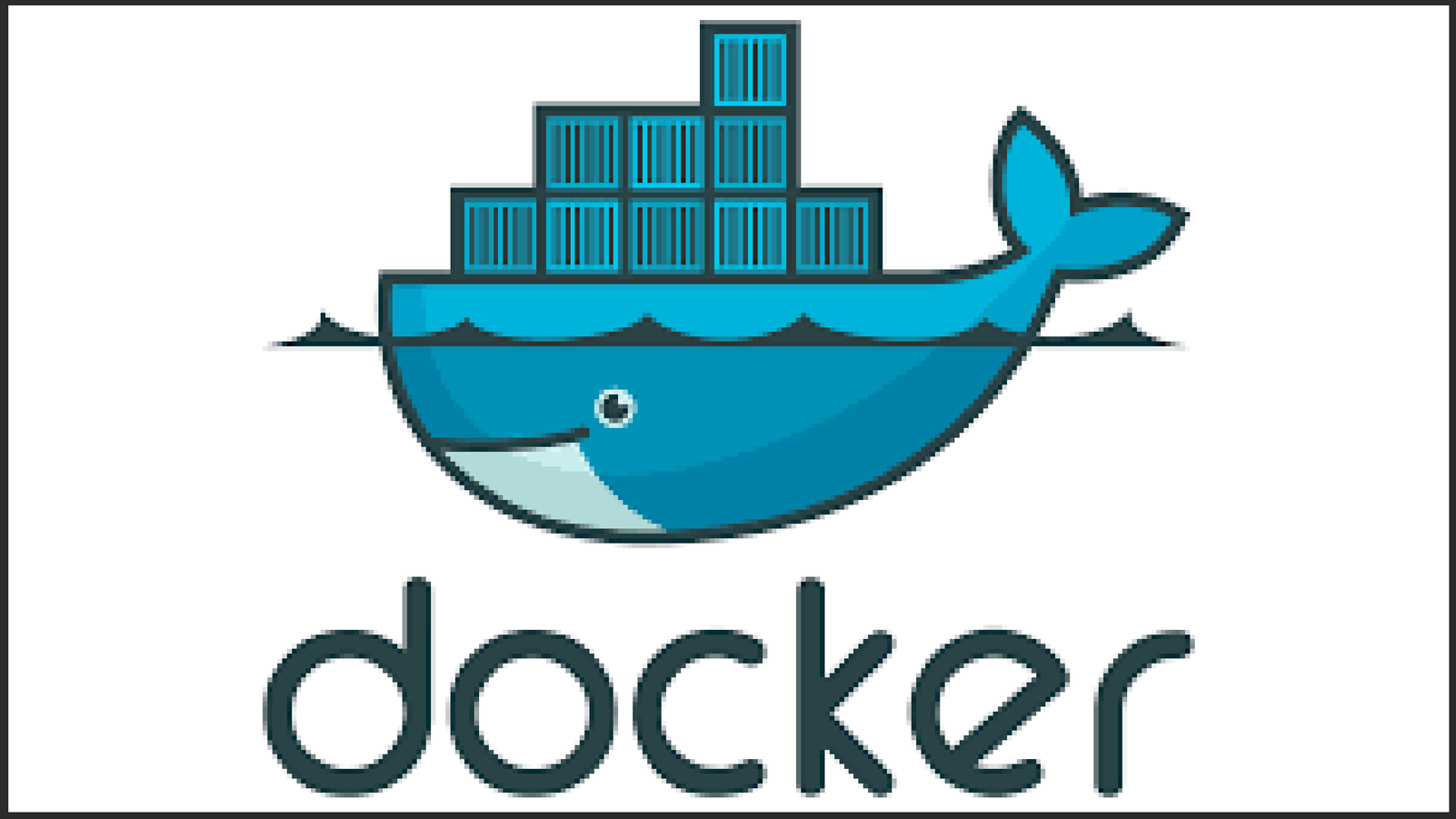
$ docker container log mariadb

$ ls -l db

$ docker container exec –ti mariadb /bin/bash

Ejecutando una Base de Datos





Dockerfile

¿ Qué son los dockerfile ?

Es un archivo de configuración que se utiliza para crear imágenes. En dicho archivo indicamos qué es lo que queremos que tenga la imagen, y los distintos comando para instalar las herramientas. Esto sería un ejemplo de Dockerfile para tener una imagen de Ubuntu con la aplicación de Git instalada.

```
FROM ubuntu:latest
RUN apt-get update \
   && apt-get install git -y
```

¿ Qué son los dockerfile ?

$ cat Dockerfile

```
FROM busybox
ENV foo /bar
WORKDIR ${foo}
ADD . $foo
```

$ cat .dockerignore

```
*.md
!README*.md
README-secret.md
```

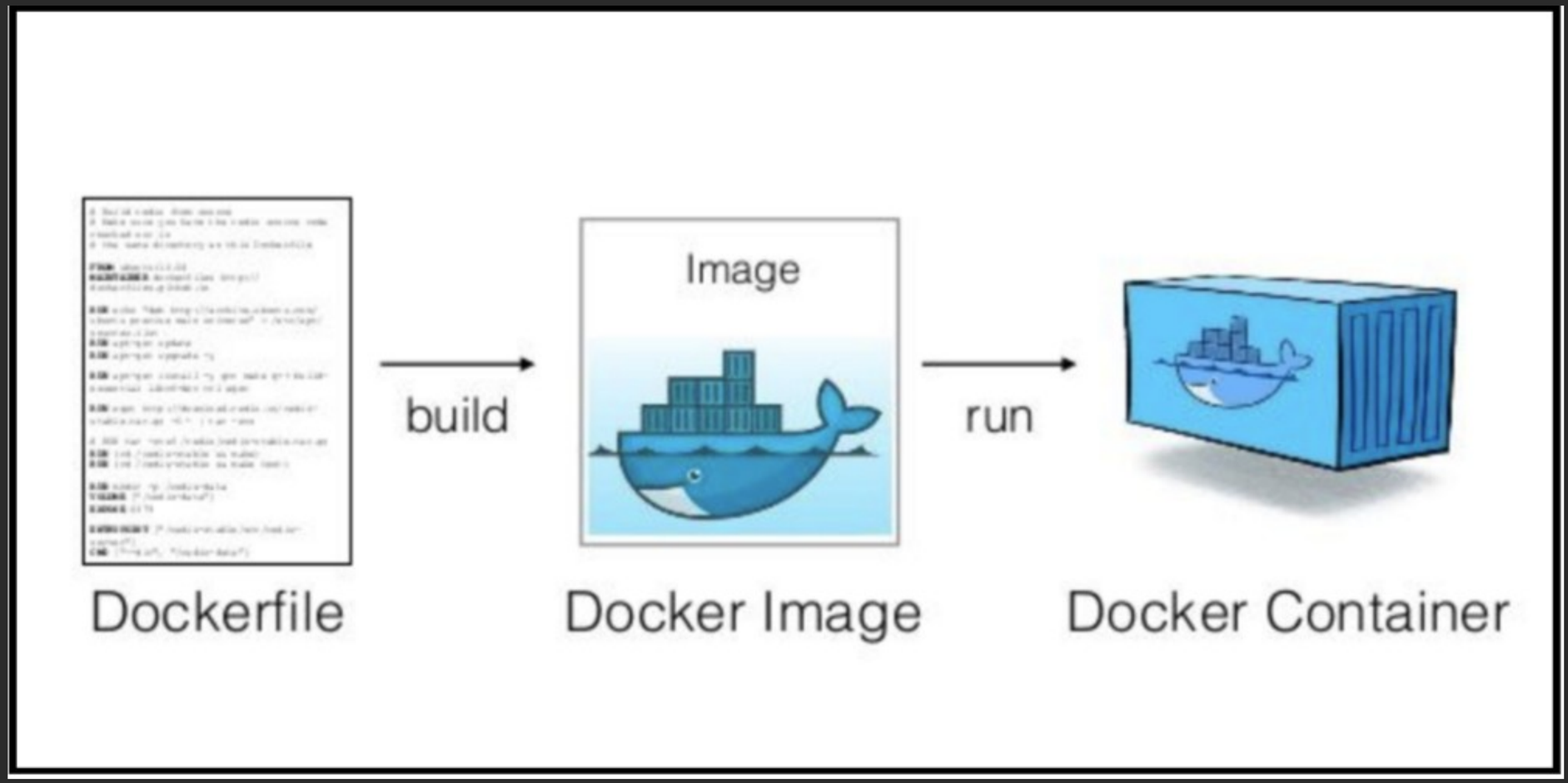
Excluye todos los ficheros con extensión *.md

Este fichero lo permite

$ docker build . -t dockerignore:latest

$ docker container run -ti --rm dockerignore /bin/sh

Dockerfile



Dockerfile

- **LABEL** -> Agrega metadatos en la imágen.
- **FROM** -> Indica que imagen utiliza.
- **EXPOSE** ->Especifica un el puerto de conexión.
- **#** -> Comentario
- **CMD** -> Especifica que commando que se va a corer en la imagen.
- **ENTRYPOINT** -> Programa que se va a ejecutar en la imagen y este no es modificable.
- **COPY** -> Copia archivos a la imagen.
- **ADD** -> Copia y descomprime archivos en la imagen.

Dockerfile

- **RUN** -> Ejecutar commandos para el armado de la imagen.
- **ENV** -> Indica variables de entorno.
- **VOLUME** -> Expose un archivo/directorio para ser utilizado en la ejecición del container.
- **USER** -> Utilizar un usuario determinado.
- **WORKDIR** -> Directorio d trabajo.
- **SHELL** -> Shell por defecto que se va utilizer.

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Dockerfile

$ nano Dockerfile

```
FROM busybox
SHELL ["/bin/sh"]
```

Shell por defecto

$ docker build . –t shell

$ docker run –ti --rm shel

Dockerfile

$ nano Dockerfile

```
FROM ubuntu:latest

LABEL nombre="echo" \
      creador="Marcos Pablo Russo" \
      email="marcospr1974@gmail.com" \
      version="1.0" \
      description="Ejemplo de Dockerfile"

COPY hola.txt /tmp

ENTRYPOINT ["/usr/bin/cat", "/tmp/hola.txt"]
```

Mediante el comando **COPY** permite solo copiar archivos.

$ docker build . –t REPOSITORIO:VERSION

$ docker build . –t echo:1.0

Dockerfile

$ nano Dockerfile

```
FROM ubuntu:latest

LABEL nombre="Figlet" \
      creador="Marcos Pablo Russo" \
      email="marcospr1974@gmail.com" \
      version="1.0" \
      description="Ejemplo de Dockerfile"

RUN apt-get update \
 && apt-get install figlet -y \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

ENTRYPOINT ["/usr/bin/figlet"]
CMD ["-h"]
```

$ docker build . –t figlet:1.0

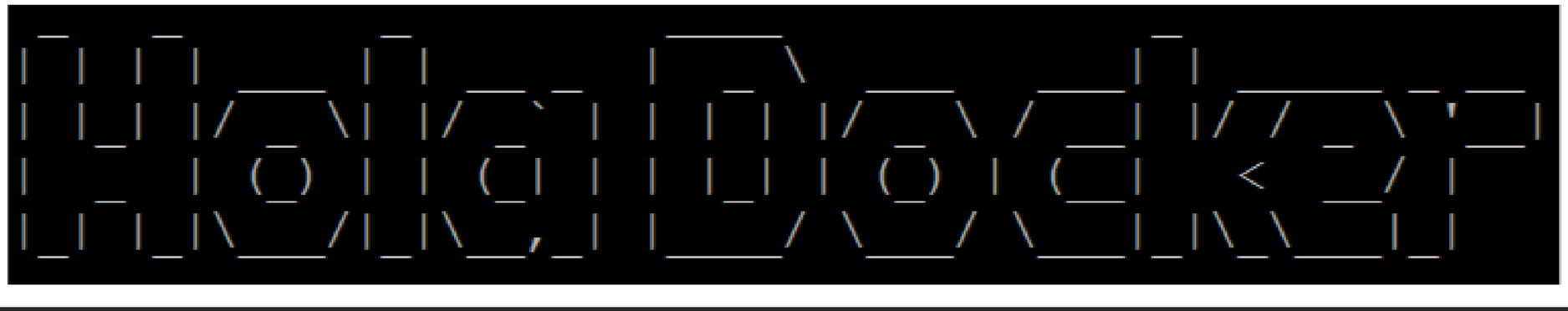
Dockerfile

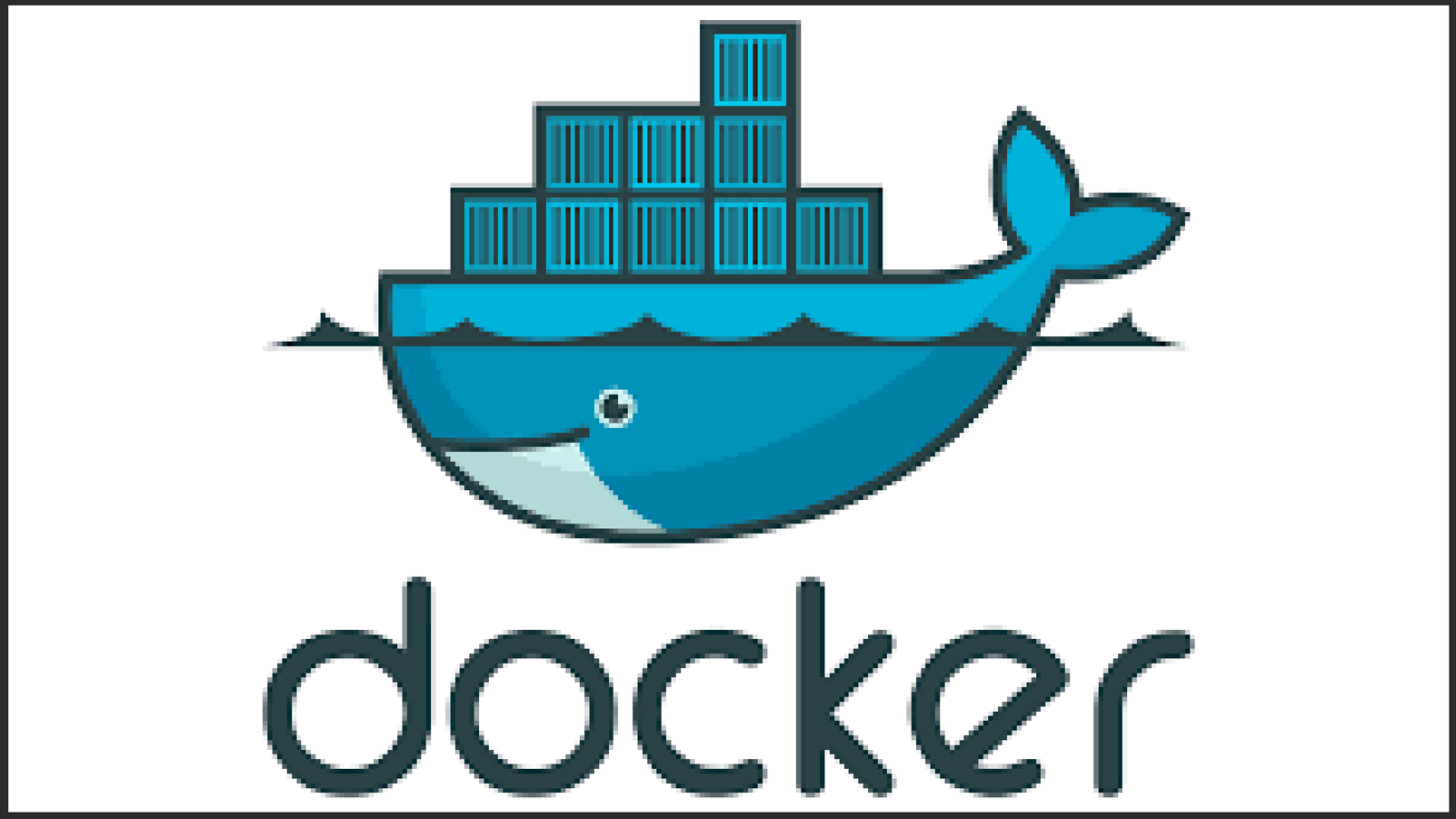
$ docker image ls

$ docker run –ti figlet:1.0

```
/usr/bin/figlet: invalid option -- 'h'
Usage: figlet [ -cklnoprstvxDELNRSWX ] [ -d fontdirectory ]
              [ -f fontfile ] [ -m smushmode ] [ -w outputwidth ]
              [ -C controlfile ] [ -I infocode ] [ message ]
```

Dockerfile





Docker Compose

¿ Qué es Docker Compose ?

Es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de está forma es mas sencillo crear contendores que se relacionen entre sí, conectarlos, habilitar puertos, volumenes, etc.

Nos permite lanzar un solo comando para crear e iniciar todos los servicios desde su configuración(YAML), esto significa que puedes crear diferentes contenedores y al mismo tiempo diferentes servicios en cada contenedor, integrarlos a un volumen común e iniciarlos y/o apagarlos, etc.

Este es un componente fundamental para poder construir aplicaciones y microservicios.

Docker-Compose funciona en todos los entornos: production, staging, development, testing, así como flujos de trabajo basados en Continuous Integration(CI).

Docker Compose

Extension es .yml o .yaml.

$ vi docker-compose.yml

$ docker compose up –d

$ docker volume ls

$ ls –l /var/lib/docker/volumes/

$ docker compose config

```
version: "3.1"
services:
   wordpress:
      image: wordpress
      hostname: wordpress
      container_name: wordpress
      environment:
         WORDPRESS_DB_HOST: mariadb
         WORDPRESS_DB_NAME: wordpress
         WORDPRESS_DB_USER: wp
         WORDPRESS_DB_PASSWORD: wpdebian
      volumes:
         - wordpress:/var/www/html
      ports:
         - 80:80
      restart: always

   mariadb:
      image: mariadb
      hostname: mariadb
      container_name: mariadb
      environment:
         MARIADB_ROOT_PASSWORD: debian
         MARIADB_DATABASE: wordpress
         MARIADB_USER: wp
         MARIADB_PASSWORD: wpdebian
      volumes:
         - mariadb:/var/lib/mysql
      ports:
         - 3306:3306
      restart: always

volumes:
  wordpress:
  mariadb:
```

Docker Compose

$ cat docker-compose.yml

```
version: '3.1'
services:
  web:
    image: nginx:latest
    hostname: "web${TAG}"
    container_name: "web${TAG}"
    ports:
      - 8080:80
    restart: always
```

Docker Compose

Podemos utilizar un archivo llamado .env para sustituir las variables de entorno por medio de este archivo.

$ cat .env

TAG=1

O

$ export TAG=1

Docker Compose

$ cat docker-compose.yml

```
version: '3.1'
services:
  web:
    image: nginx:latest
    hostname: "web${TAG}"
    container_name: "web${TAG}"
    env_file:
      - ./config/env
    ports:
      - 8080:80
    restart: always
```

Lo saca del archivo **.env**

Variables de entorno

$ cat config/env

DEBUG=1

Docker Compose

$ docker compose ps

Name	Command	State	Ports
calibre-web	/bin/bash -c /init/start.sh	Up	0.0.0.0:8085->8083/tcp,:::8085->8083/tcp
docker-registry	/entrypoint.sh /etc/docker ...	Up	0.0.0.0:5000->5000/tcp,:::5000->5000/tcp
mariadb-nx	docker-entrypoint.sh mysqld	Up	0.0.0.0:3306->3306/tcp,:::3306->3306/tcp
mariadb-wp	docker-entrypoint.sh mysqld	Up	0.0.0.0:3307->3306/tcp,:::3307->3306/tcp
nextcloud	/entrypoint.sh apache2-for ...	Up	0.0.0.0:8083->80/tcp,:::8083->80/tcp
plex	/init	Up	
portainer	/portainer	Up	0.0.0.0:9000->9000/tcp,:::9000->9000/tcp
portia	/app/docker/entry	Up	
qbittorrent	/init	Up	0.0.0.0:6881->6881/tcp,:::6881->6881/tcp, 0.0.0.0:6881->6881/udp,:::6881->6881/udp, 8080/tcp, 0.0.0.0:8088->8088/tcp,:::8088->8088/tcp
samba	/sbin/tini -- /usr/bin/sam ...	Up (healthy)	137/udp, 138/udp, 0.0.0.0:139->139/tcp,:::139->139/tcp, 0.0.0.0:445->445/tcp,:::445->445/tcp
server_netdata_1	/usr/sbin/run.sh	Up (healthy)	0.0.0.0:19999->19999/tcp,:::19999->19999/tcp
spiderfoot	/usr/bin/python3 ./sf.py - ...	Up	0.0.0.0:5001->5001/tcp,:::5001->5001/tcp
transmission	/sbin/tini -- /usr/bin/tra ...	Up (healthy)	51413/tcp, 51413/udp, 0.0.0.0:9091->9091/tcp,:::9091->9091/tcp
webmap	bash /startup.sh	Up	0.0.0.0:8002->8000/tcp,:::8002->8000/tcp
wordpress	docker-entrypoint.sh apach ...	Up	0.0.0.0:8200->80/tcp,:::8200->80/tcp

Docker Compose

$ docker compose images

Container	Repository	Tag	Image Id	Size
calibre-web	technosoft2000/calibre-web	latest	3b9d7e296e4f	1.314 GB
docker-registry	registry	2	eefcac9e3856	26.24 MB
mariadb-nx	mariadb	latest	eff629089685	407.6 MB
mariadb-wp	mariadb	latest	eff629089685	407.6 MB
nextcloud	nextcloud	latest	44f9792fd39d	868.5 MB
plex	ghcr.io/linuxserver/plex	latest	0dcda982b2fa	692.4 MB
portainer	portainer/portainer	latest	62771b0b9b09	79.14 MB
portia	scrapinghub/portia	latest	c50c5f4820df	1.383 GB
qbittorrent	ghcr.io/linuxserver/qbittorrent	latest	2d63c963674f	390.2 MB
samba	dperson/samba	latest	aac8a52c5b16	52.07 MB
server_netdata_1	netdata/netdata	latest	46258c2f95d4	329 MB
spiderfoot	spiderfoot	3.3	2dfe41cf12d5	597.3 MB
transmission	dperson/transmission	latest	d97d530ced09	14.71 MB
webmap	marcositu/webmap	latest	c910c818e432	1.797 GB
wordpress	wordpress	latest	c2dd1984ad5b	550.5 MB

Docker Compose

$ docker compose logs wordpress

```
Attaching to wordpress
wordpress           | WordPress not found in /var/www/html - copying now...
wordpress           | Complete! WordPress has been successfully copied to /var/www/html
wordpress           | No 'wp-config.php' found in /var/www/html, but 'WORDPRESS_...' variables supplied; copying 'wp-config
-docker.php' (WORDPRESS_DB_HOST WORDPRESS_DB_NAME WORDPRESS_DB_PASSWORD WORDPRESS_DB_USER)
wordpress           | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.24
.0.14. Set the 'ServerName' directive globally to suppress this message
wordpress           | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.24
.0.14. Set the 'ServerName' directive globally to suppress this message
wordpress           | [Fri Jun 18 23:03:31.111464 2021] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.38 (Debian) PHP/7.
4.20 configured -- resuming normal operations
wordpress           | [Fri Jun 18 23:03:31.112043 2021] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND
'
```

Docker Compose

$ docker compose top wordpress

```
wordpress
  UID        PID       PPID    C    STIME   TTY      TIME                CMD
-------------------------------------------------------------------------------
root       532480    532454   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
www-data   532635    532480   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
www-data   532636    532480   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
www-data   532637    532480   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
www-data   532638    532480   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
www-data   532639    532480   0    20:03   ?      00:00:00   apache2 -DFOREGROUND
```

Docker Compose

$ docker compose events wordpress

```
2021-06-18 20:11:53.622902 container exec_create: /bin/sh -c /usr/sbin/health.sh 76a56294d5a2c24f2bb2a67a5688259141912f9ff
caf985faf52e88d492b647b (execID=1b8f2f8ed76d0b7cc008b286d7ed0e3484faac853c490beaa139f6888ea33f59, image=netdata/netdata:la
test, name=server_netdata_1)
2021-06-18 20:11:53.623292 container exec_start: /bin/sh -c /usr/sbin/health.sh 76a56294d5a2c24f2bb2a67a5688259141912f9ffc
af985faf52e88d492b647b (execID=1b8f2f8ed76d0b7cc008b286d7ed0e3484faac853c490beaa139f6888ea33f59, image=netdata/netdata:lat
est, name=server_netdata_1)
2021-06-18 20:11:53.833998 container exec_die 76a56294d5a2c24f2bb2a67a5688259141912f9ffcaf985faf52e88d492b647b (execID=1b8
f2f8ed76d0b7cc008b286d7ed0e3484faac853c490beaa139f6888ea33f59, exitCode=0, image=netdata/netdata:latest, name=server_netda
ta_1)
```

Docker Compose

$ docker compose exec wordpress ls –l /


Gracias por participar

Marcos Pablo Russo

HTTPS://GITHUB.COM/MARCOSPR1974

MARCOSPR1974@GMAIL.COM