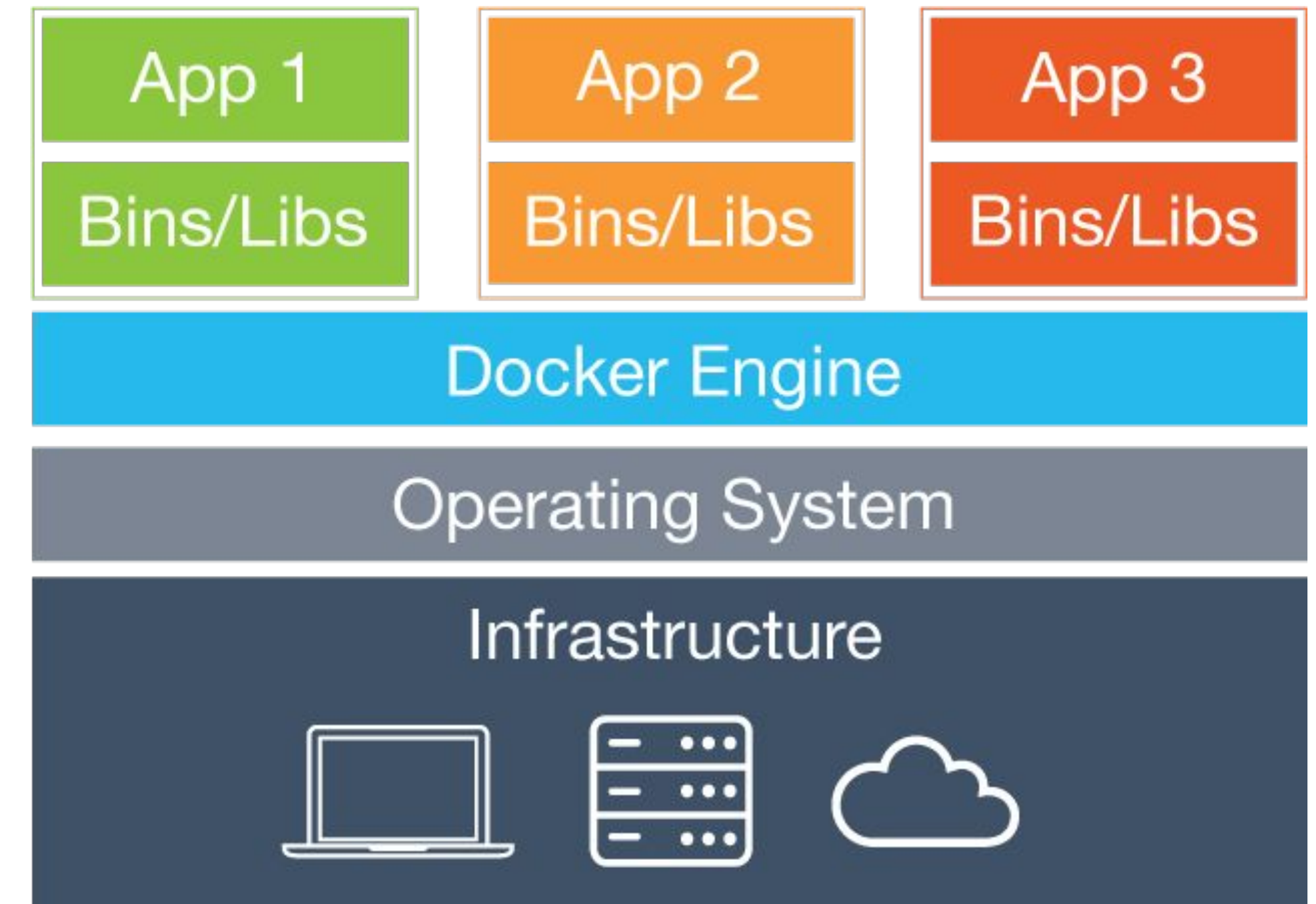
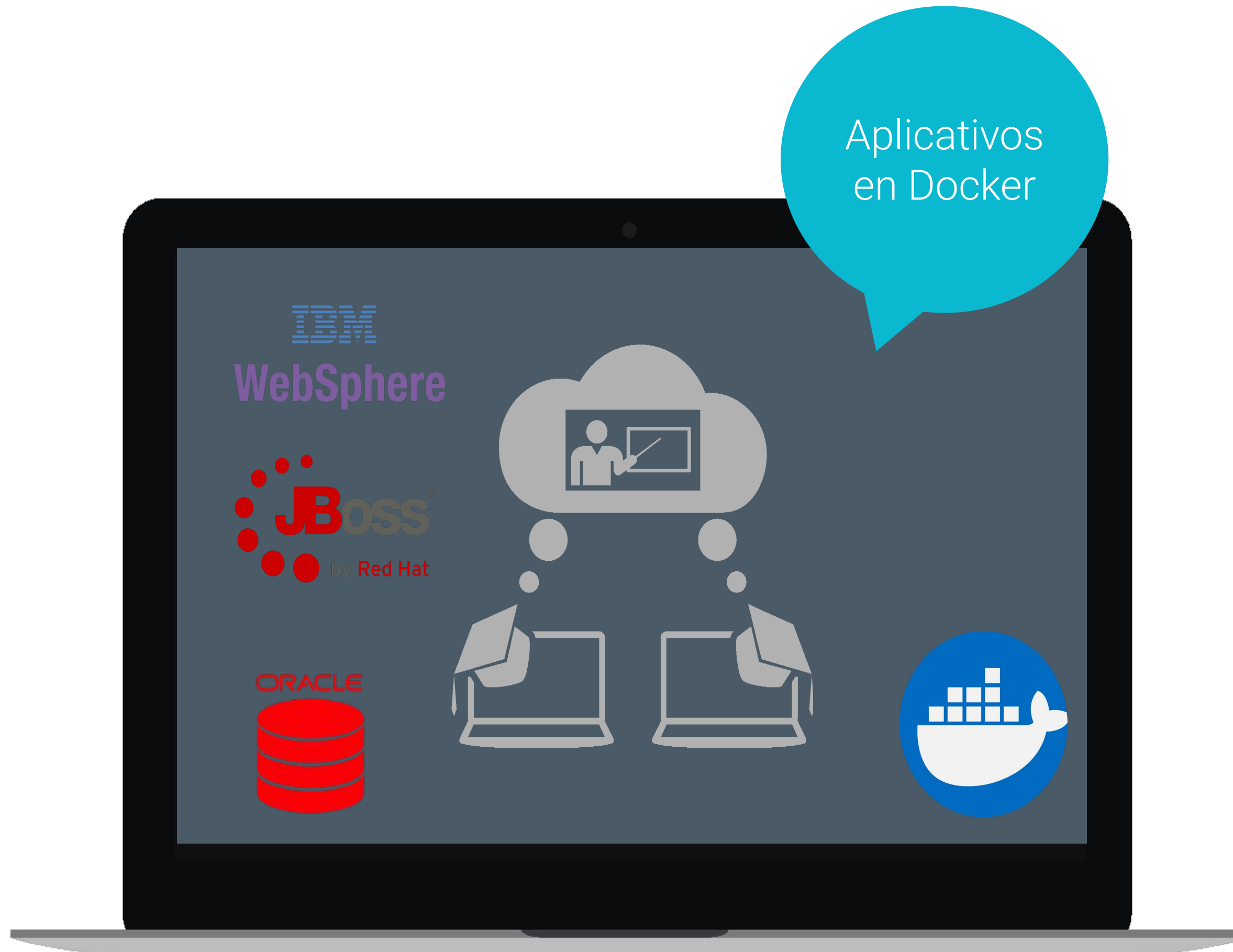


DOCKER



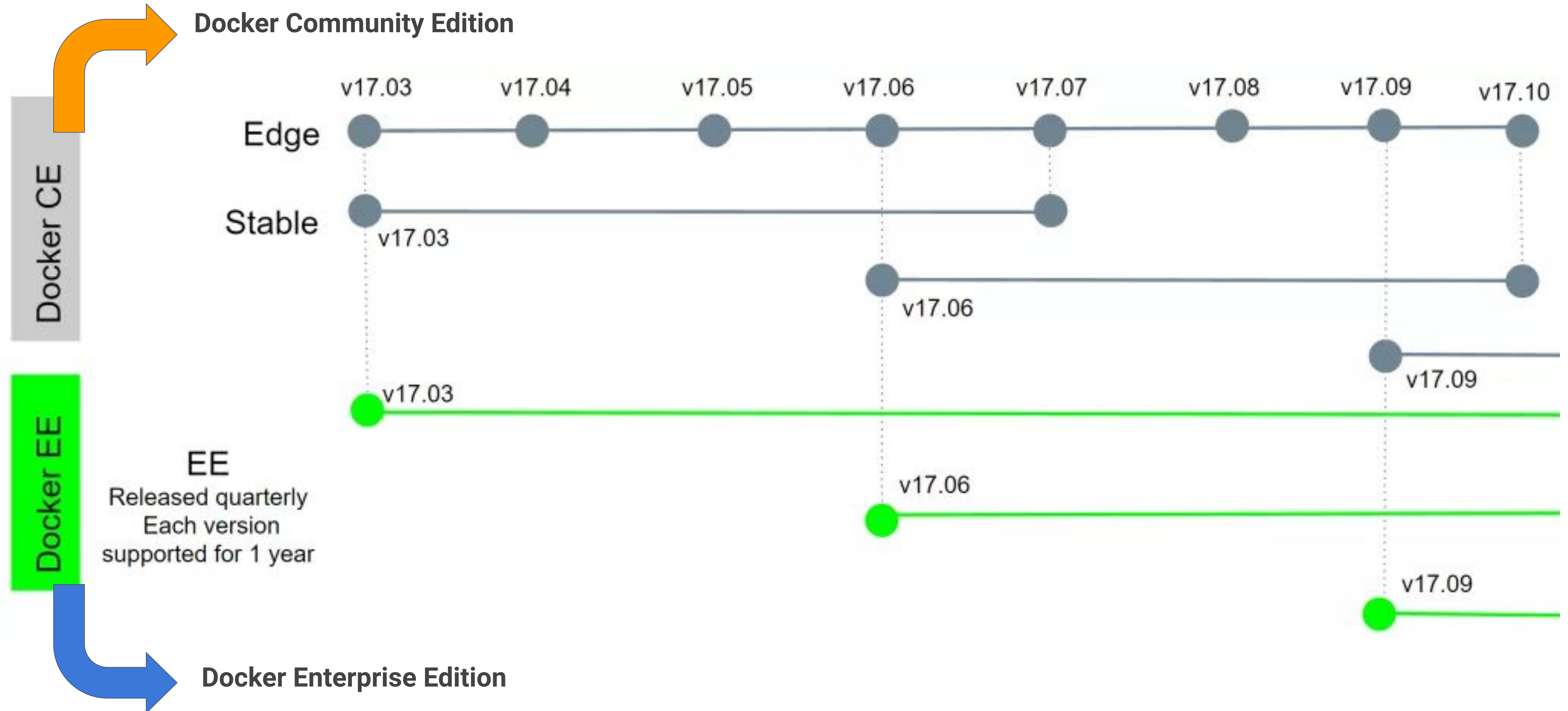
¿Qué es Docker?

Aplicativos
en Docker



Docker es una herramienta que empaqueta o “embala” aplicaciones y sus dependencias en imágenes. Éstas son inmutables, ligeras y portables. Y permite ejecutar “N” veces estas imágenes en contenedores.

¿Qué versiones de Docker tenemos?

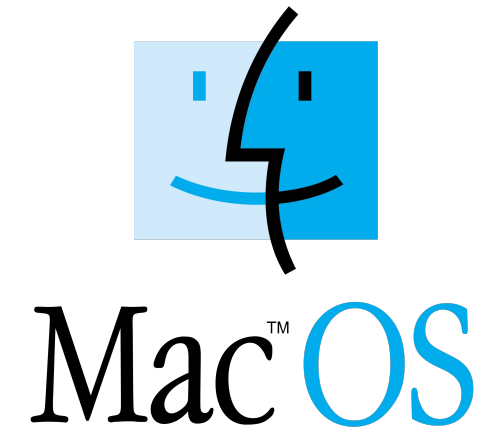


INSTALACIÓN DE DOCKER



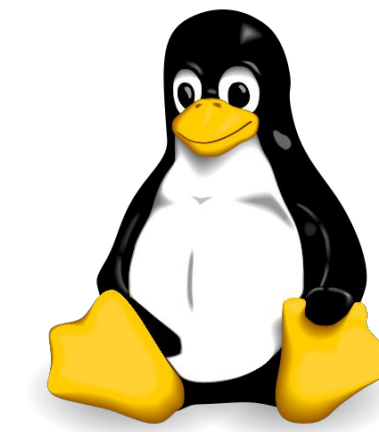
Windows (Pro o Enterprise)

[Docker for windows](#)
[Docker Toolbox](#)
[\(opcional\)](#)
[gitbash\(opcional\)](#)



Mac

[Mac OS >= Yosemite :](#)
[Docker for mac](#)
[Docker toolbox](#)
[\(opcional\)](#)



Linux

`docker: curl -sSL`
`https://get.docker.com/`
`| sh`
[Docker compose](#)
[Docker machine](#)



Verificando instalación de docker

1

\$ docker version

Se debe mostrar información de 2 puntos importantes:

- a.- El demonio docker
- b.- El cliente docker

2

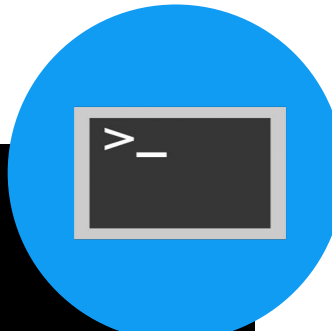
\$ docker info

Se muestra mucha información del uso de docker, como por ejemplo: memoria, red, etc

3

\$ docker run hello-world

Se crea un contenedor y este es el famoso “hola mundo”.



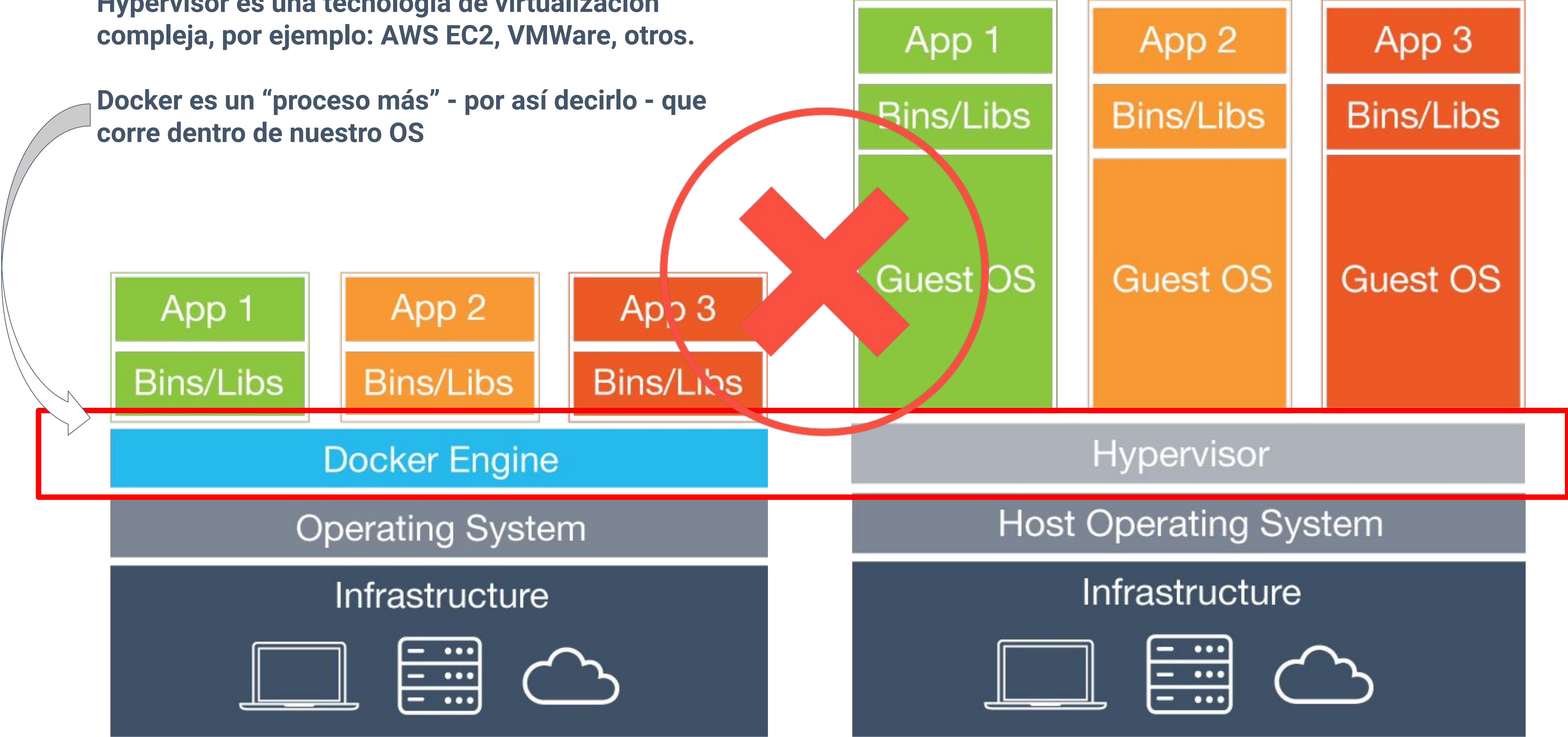
```
$ docker version
Client:
 Version:           18.03.0-ce
 API version:       1.37
 Go version:        go1.9.4
 Git commit:        0520e24302
 Built: Fri Mar 23 08:31:36 2018
 OS/Arch:           windows/amd64
 Experimental:      false
 Orchestrator:      swarm

Server:
 Engine:
  Version:          18.05.0-ce
  API version:      1.37 (minimum version 1.12)
  Go version:       go1.10.1
  Git commit:       f150324
  Built:            Wed May  9 22:20:42 2018
  OS/Arch:          linux/amd64
  Experimental:     false
```


Docker != Virtual Machine

Hypervisor es una tecnología de virtualización compleja, por ejemplo: AWS EC2, VMWare, otros.

Docker es un “proceso más” - por así decirlo - que corre dentro de nuestro OS





Comandos Básicos en Docker

¿Cómo funcionan los comandos?

2 El cliente envía la solicitud al server

```
$ docker version
Client:
 Version:      18.03.0-ce
 API version:  1.37
 Go version:   go1.9.4
 Git commit:   0520e24302
 Built: Fri Mar 23 08:31:36 2018
 OS/Arch:     windows/amd64
 Experimental: false
 Orchestrator: swarm
```

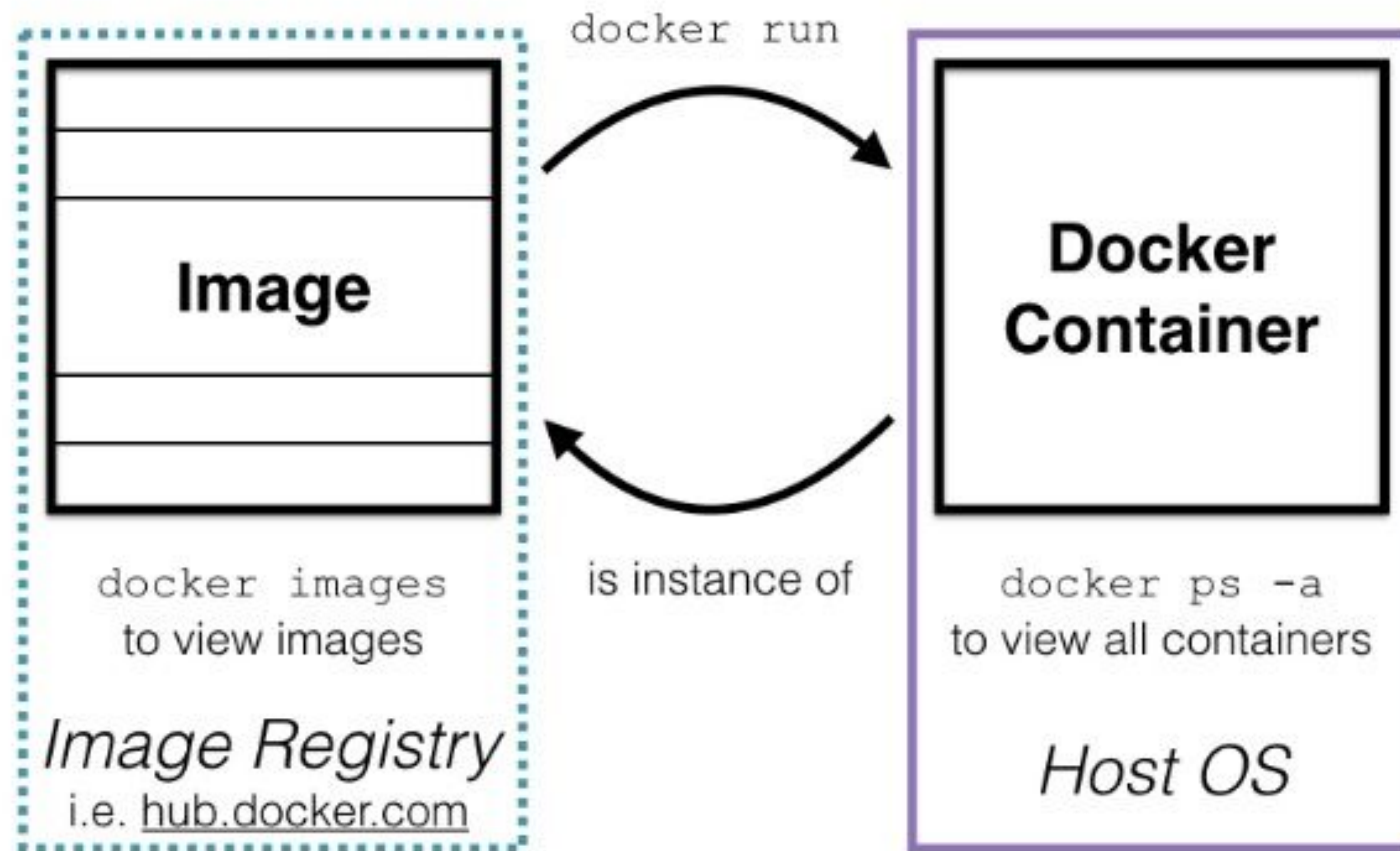
```
Server:
 Engine:
  Version:      18.05.0-ce
  API version:  1.37 (minimum version 1.12)
  Go version:   go1.10.1
  Git commit:   f150324
  Built:        Wed May 9 22:20:42 2018
  OS/Arch:     linux/amd64
  Experimental: false
```

docker run ...

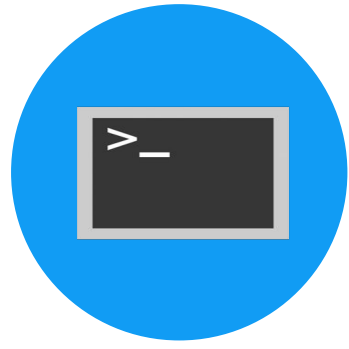
Operating System

3 El servidor interactúa con el OS para ejecutar la orden

Imagen y Contenedor



Buscando imágenes en docker

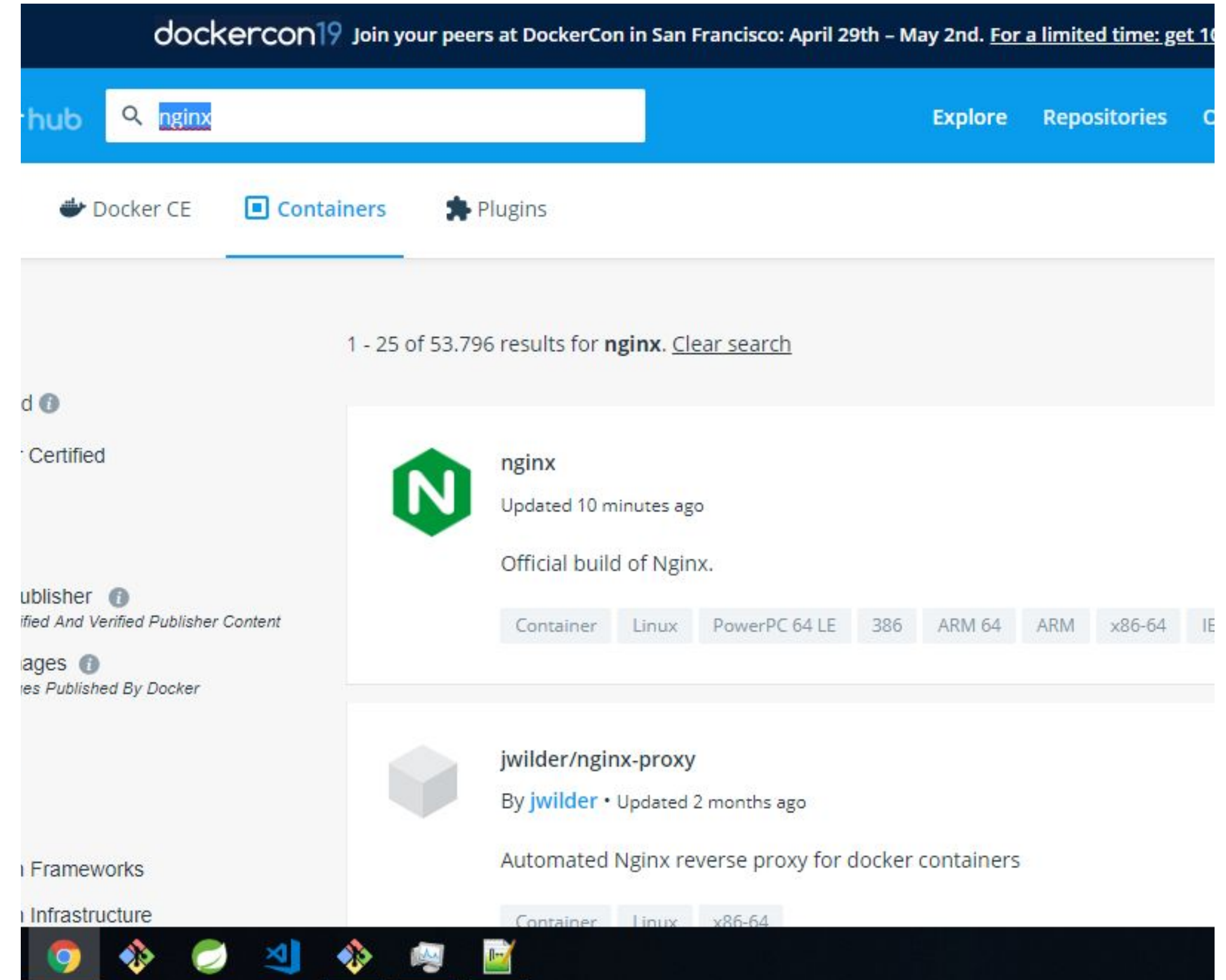


Terminal

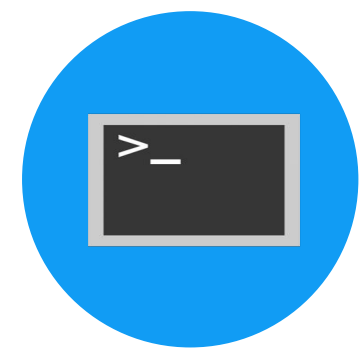
```
$ docker search nginx
```

NAME	DESCRIPTION	STARS	OFFICIAL AUTOMATED
nginx	Official build of Nginx.	11152	[OK]
jwilder/nginx-proxy	Automated Nginx reverse proxy for docker con...	1574	[OK]
richarvey/nginx-php-fpm	Container running Nginx + PHP-FPM capable of...	697	[OK]
jrcs/letsencrypt-nginx-proxy-companion	LetsEncrypt container to use with nginx as p...	94	[OK]
webdevops/php-nginx	Nginx with PHP-FPM	123	[OK]

Browser <https://hub.docker.com/>



Creando nuestro 1er contenedor



Terminal

```
$ docker run -d -p 80:80 --name servidor_nginx nginx
```

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
27833a3ba0a5: Pulling fs layer
e83729dd399a: Pulling fs layer
ebc6a67df66d: Pulling fs layer
ebc6a67df66d: Verifying Checksum
ebc6a67df66d: Download complete
e83729dd399a: Verifying Checksum
e83729dd399a: Download complete
27833a3ba0a5: Verifying Checksum
27833a3ba0a5: Download complete
27833a3ba0a5: Pull complete
e83729dd399a: Pull complete
ebc6a67df66d: Pull complete
Digest: sha256:c8a861b8a1eeef6d48955a6c6d5dff8e2580f13ff4d0f549e082e7c82a8617a2
Status: Downloaded newer image for nginx:latest
dacd4807bece8339fed41bdf9dc0779a1ef01301890b09e2dd6d5d4516ae4731
```

Se crea un contenedor en segundo plano (-d) y se expone en el puerto 80 del host. El contenedor tiene por nombre `servidor_nginx` y usa la imagen nginx

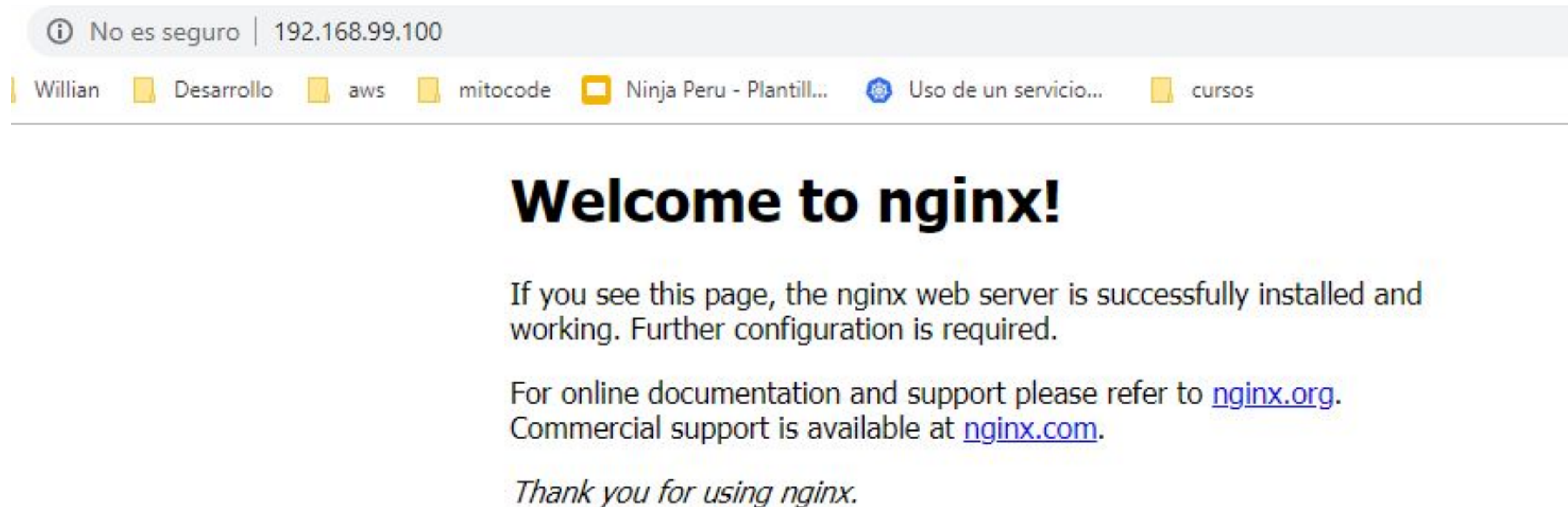
Si la imagen no está descargada en el host, docker busca en el registry público por defecto (docker hub) o en un registry privado

El contenedor creado tiene un ID

Resultado del 1er contenedor



Ingresar en el browser la siguiente URL: <http://localhost>



```
$ docker run -d -p 80:80 --name servidor_nginx nginx
```

Sin -p (--publish), no podemos ingresar a la web de nginx, --publish publica un puerto interno de un contenedor hacia un puerto externo del host

Comandos básicos...

Estructura de los comandos docker:

- Antiguo: `docker <command> (options)`
- Nuevo: `docker <command> <subcommand> (options)`

<code>\$ docker images</code>	(sirve para visualizar las imágenes docker)
<code>\$ docker ps -a</code>	(sirve para mostrar los contenedores incluidos los detenidos)
<code>\$ docker run -it --rm -p 8888:8080 tomcat:8.0</code>	(sirve para crear un contenedor en base a una imagen)
<code>\$ docker rm -f tomcat_server</code>	(sirve para eliminar contenedores a la fuerza -f)
<code>\$ docker rm -f \$(docker ps -aq)</code>	(sirve para eliminar todos los contenedores, incluido los detenidos)
<code>\$ docker rmi -f mysql</code>	(sirve para eliminar una imagen a la fuerza -f)
<code>\$ docker logs -f jboss_server</code>	(sirve para visualizar logs de un contenedor)
<code>\$ docker exec -it jboss_server bash</code>	(sirve para ingresar al terminal de un contenedor)



Gestión de múltiples contenedores...

```
$ docker run -d -p 8000:8080 --name tomcat_server tomcat:8.0-alpine
```

```
$ docker run -d -p 9000:8080 -p 9990:9990 --name jboss_server andreptb/wildfly:8.2.0.Final.jdk8-alpine  
/opt/jboss/wildfly/bin/domain.sh -b 0.0.0.0 -bmanagement 0.0.0.0
```

```
$ docker exec -it jboss_server /opt/jboss/wildfly/bin/add-user.sh admin Admin#70365 --silent
```

```
$ docker run -d -p 9043:9043 -p 9443:9443 --name was_server ibmcom/websphere-traditional:latest
```

```
$ docker exec -it was_server cat /tmp/PASSWORD
```

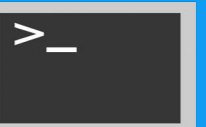
```
$ docker logs -f jboss_server
```

```
$ docker exec -it jboss_server bash
```

```
$ docker rm -f tomcat_server
```

```
$ docker rm -f $(docker ps -aq)
```

```
$ docker rmi -f mysql
```



Gestión de múltiples contenedores...

```
$ docker pull mysql:latest
```

```
$ docker run -d -p 3333:3306 -e MYSQL_ROOT_PASSWORD=12345678 --name mysql_server mysql:5.7.25  
--default-authentication-plugin=mysql_native_password
```

```
$ docker cp ~/Desktop/fuentes_sesion_05/apps/mysql_server/scripts/DDL.sql mysql_server:/tmp
```

```
$ docker exec -it mysql_server bash
```

```
$root@a123e:/# mysql -u root -p
```

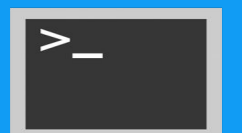
```
$mysql> source /tmp/DDL.sql
```

```
$mysql> exit
```

```
$root@a123e:/# exit
```

```
$ docker stop mysql_server
```

```
$ docker rm mysql_server
```



Monitoreo de Contenedores Docker

Ver Contenedores

con -a se visualiza todos los
containers (detenidos y en
ejecución)

docker ps -a

docker top [container]

Ver PID - UID

Ver el identificador del
proceso

Ver Características

Ver información como IP,
volumen, driver logs, etc

docker inspect [container]

docker stats [container]

Ver Consumo

Cuánto consumo de CPU,
memoria, límites, disco IO

Monitoreo de Contenedores Docker

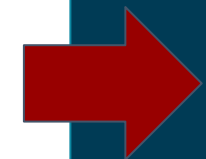
```
$ docker container run -d nginx
```

```
f050ac5534f7ad9f19d9db7de41fc5d13aab39964bd83704c9d35b9bed079dfc
```

```
$ docker container run -d -e MYSQL_RANDOM_ROOT_PASSWORD=true mysql
```

```
94896b5c9d6f2a1e49c1e7c8a873fdb6e8ed8aa9c1a87fce0c4aa258b580eddc
```

1



```
$ docker container ps -a
```

2



```
$ docker container inspect 948
```

3



```
$ docker container stats 948
```

Explorando la Shell en un Container

```
$ docker run -it --name proxy nginx bash
```

```
root@e635ccd51489:/# ls -al
```

```
total 72
```

```
drwxr-xr-x 33 root root 4096 Apr  1 20:42 .
```

```
drwxr-xr-x 33 root root 4096 Apr  1 20:42 ..
```

```
-rwxr-xr-x  1 root root   0 Apr  1 20:42 .dockerenv
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 bin
```

```
drwxr-xr-x  2 root root 4096 Feb  3 13:01 boot
```

```
drwxr-xr-x  5 root root 360 Apr  1 20:42 dev
```

```
drwxr-xr-x 43 root root 4096 Apr  1 20:42 etc
```

```
drwxr-xr-x  2 root root 4096 Feb  3 13:01 home
```

```
drwxr-xr-x 10 root root 4096 Mar 26 12:00 lib
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 lib64
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 media
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 mnt
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 opt
```

```
dr-xr-xr-x 342 root root   0 Apr  1 20:42 proc
```

```
drwx----- 2 root root 4096 Mar 26 12:00 root
```

```
drwxr-xr-x  3 root root 4096 Mar 26 12:00 run
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 sbin
```

```
drwxr-xr-x  2 root root 4096 Mar 26 12:00 srv
```

```
dr-xr-xr-x 13 root root   0 Apr  1 20:42 sys
```

```
drwxrwxrwt  2 root root 4096 Mar 26 23:13 tmp
```

```
drwxr-xr-x 15 root root 4096 Mar 26 12:00 usr
```

```
drwxr-xr-x 15 root root 4096 Mar 26 12:00 var
```

```
root@e635ccd51489:/# exit
```

A blue circular icon containing a white terminal window symbol, which consists of a black rectangle with a white prompt character (>) and a white cursor line (_).

Shell en un Container y publicando puertos

-it hace que sea interactivo y emula una shell interna del contenedor

```
$ docker run -p 80:80 -it --name proxy nginx bash
root@e635ccd51489:/# exit
```

\$ docker container ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9b3ae83870dd	nginx	"bash"	9 minutes ago	Exited (0) 4 seconds ago		proxy

```
$ docker run -d -p 80:80 --name proxy2 nginx
cab07715cc8604460e00fa3147ca8f1d4cf22eaeb3740e38b77d5cec9059a4d8
```

cambiar el comando de inicio hace que no se lance el script de inicio por defecto, tener cuidado!!

\$ docker stop proxy2

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
cab07715cc86	nginx	"nginx -g 'daemon of...'"	4 minutes ago	Exited (0) 4 seconds ago		proxy2
9b3ae83870dd	nginx	"bash"	17 minutes ago	Exited (0) 7 minutes ago		proxy

Attach a shell de un container detenido

```
$ docker run -it --name ubuntu-os ubuntu bash
```

```
root@0dc6456e1ef4:/# cat /etc/os-release
```

```
NAME="Ubuntu"
```

```
VERSION="18.04.2 LTS (Bionic Beaver)"
```

```
root@0dc6456e1ef4:/# exit
```

```
$ docker container ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0dc6456e1ef4	ubuntu	"bash"	2 minutes ago	Exited (0) 2 seconds ago		ubuntu-os

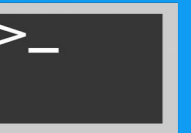
```
$ docker start --help
```

Options:

- a, --attach Attach STDOUT/STDERR and forward signals
- detach-keys string Override the key sequence for detaching a container
- i, --interactive Attach container's STDIN

```
$ docker start -ai ubuntu-os
```

```
root@0dc6456e1ef4:/#
```



¿Puedo descargar todo un container?

```
$ docker run -d --name proxy nginx
```

```
ebc7d57ccc54c560121ff112e590d71cf508a0fd084730c5d904325b260a9483
```

```
$ docker container ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ebc7d57ccc54	nginx	"nginx -g 'daemon of..."	9 seconds ago	Up 8 seconds	80/tcp	proxy

```
$ mkdir ~/Escritorio/temporal && cd ~/Escritorio/temporal
```

```
$ docker export proxy > container_proxy.zip
```

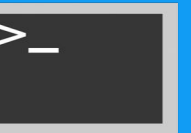
```
$ ls -la
```

```
total 108732
```

```
drwxr-xr-x  2 william william   4096 abr  1 19:03 .
```

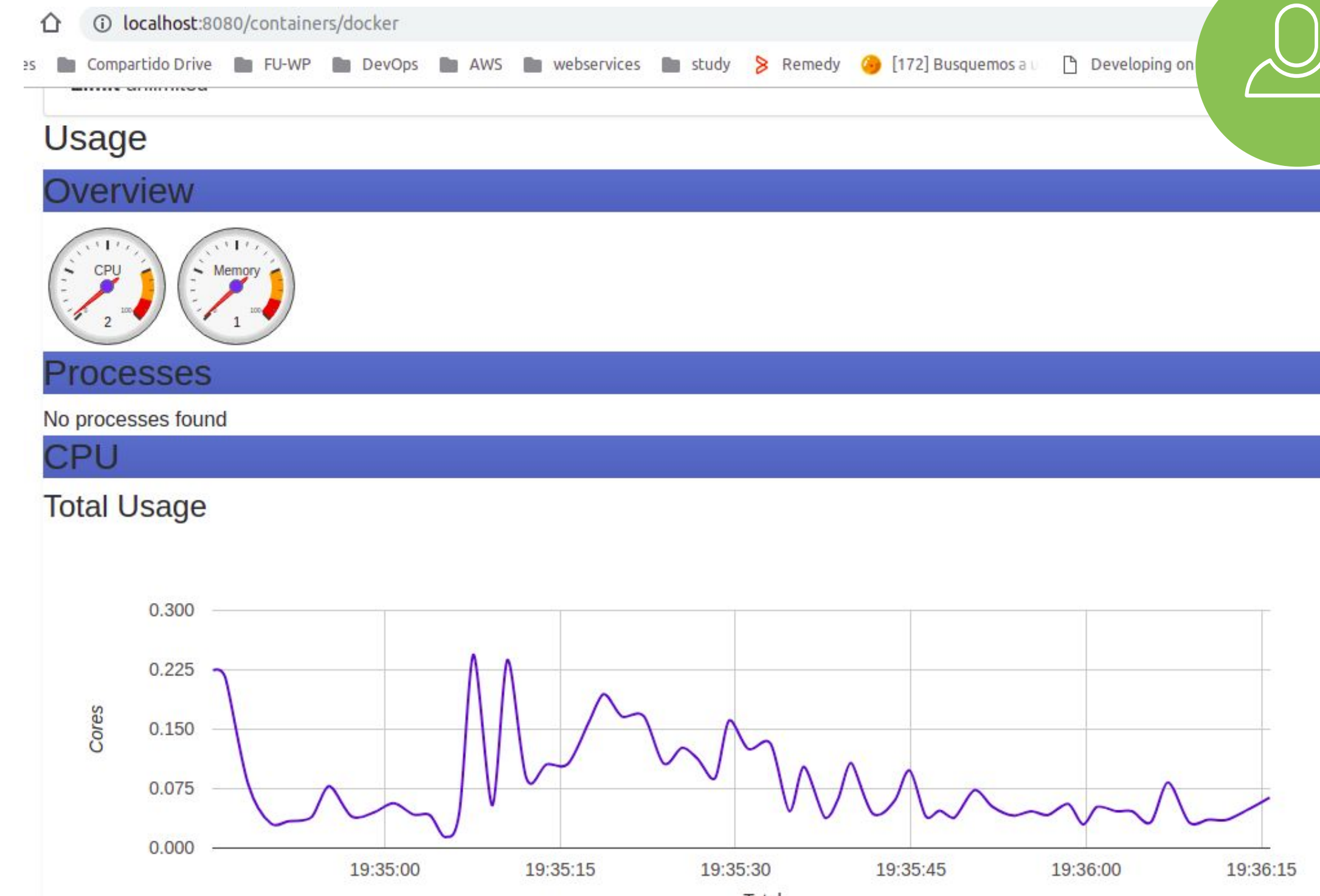
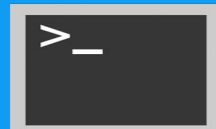
```
drwxr-xr-x 10 william william   4096 abr  1 18:59 ..
```

```
-rw-r--r--  1 william william 111332352 abr  1 19:03 container_proxy.zip
```



Monitoreo con CAdvisor

```
$ docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:latest
```





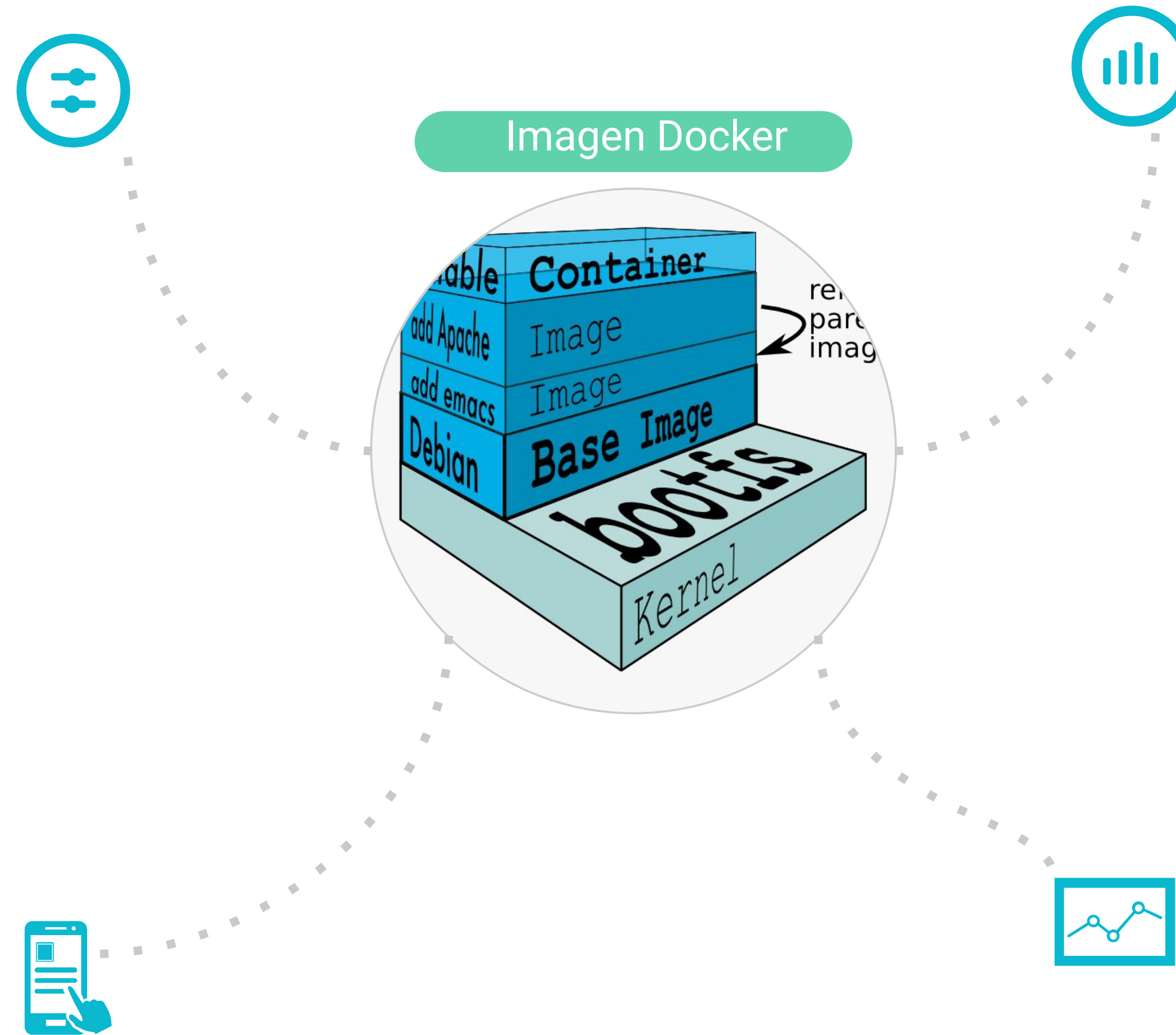
Imágenes Docker

Concepto de imagen docker

Compuesto por N capas, la última capa es de lectura/escritura disponible para crear el contenedor.

El resto son capas de lectura montadas una a una

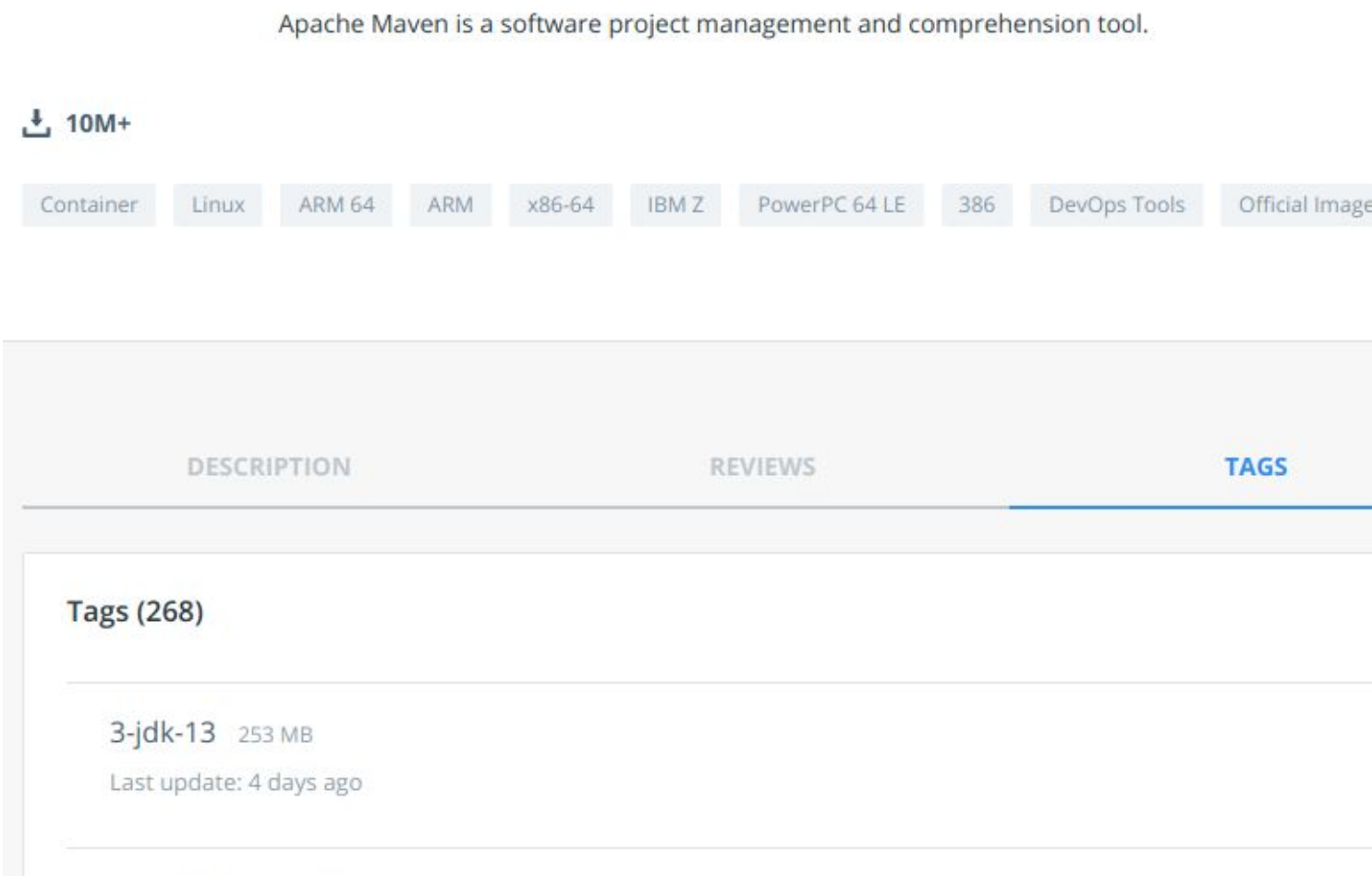
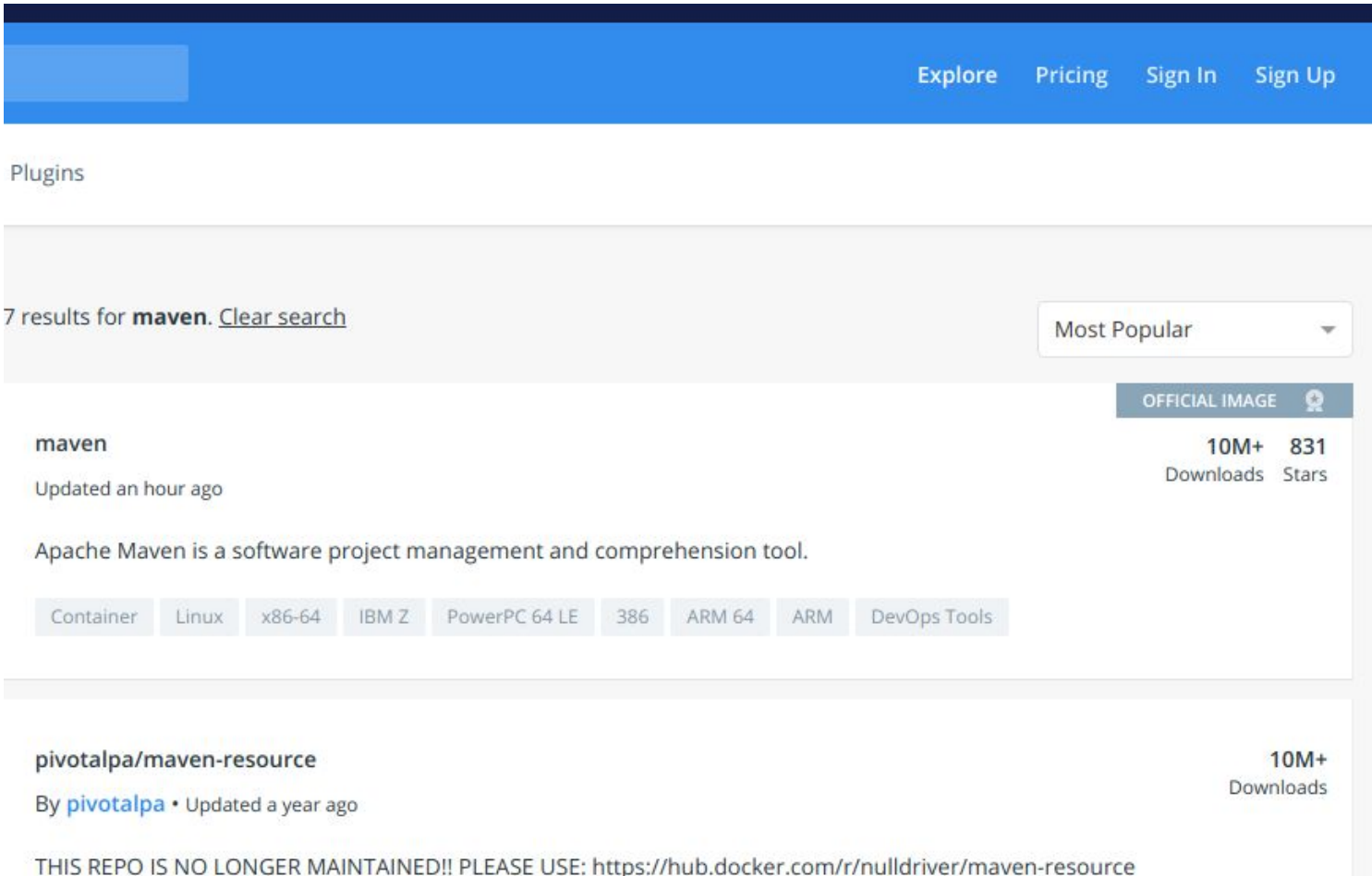
En resumen, es un template - o snapshot por así decirlo - que encapsula una aplicación y sus dependencias



Toda imagen tiene una "Imagen base". Una imagen docker no es un SO. No tiene kernel ni módulos de éste

Las imágenes se versionan mediante tags, podemos decir que se asemeja a los tag de GIT

Documentación de las imágenes



Maven needs the user home to download artifacts to, and if the user does not exist in the image an ext Java property needs to be set.

For example, to run as user `1000` mounting the host' Maven repo

```
$ docker run -v ~/.m2:/var/maven/.m2 -ti --rm -u 1000 -e MAVEN_CONFIG=/var/maven/.m2 maven mvn -Duser.home=/va
```

Image Variants

The `maven` images come in many flavors, each designed for a specific use case.

maven:<version>

This is the defacto image. If you are unsure about what your needs are, you probably want to use this o designed to be used both as a throw away container (mount your source code and start the container t

Imagen Oficial y No Oficial

La web docker hub nos muestra imágenes oficiales y no oficiales (creados por usuarios de la comunidad)

Tags de la imagen

Permiten versionar las imágenes, así podemos “apuntar” o “usar” una imagen que tenga una características en particular, por ejemplo maven con java 8

Cómo usarlo y variantes

La documentación de imágenes oficiales y no oficiales nos dicen cómo usar las mismas y las consideraciones a tener, En las variantes tenemos: normal, slim y **alpine (importante)**



También visita github, ejemplo [aquí](#)

Docker history & inspect de una imagen

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
4456f3d84674	3 days ago	/bin/sh -c #(nop) CMD ["mongod"]	0B	
<missing>	3 days ago	/bin/sh -c #(nop) EXPOSE 27017	0B	
<missing>	3 days ago	/bin/sh -c #(nop) ENTRYPOINT ["docker-entry...	0B	
<missing>	3 days ago	/bin/sh -c #(nop) COPY file:945726c0bedd1f0e...	11kB	
<missing>	3 days ago	/bin/sh -c #(nop) VOLUME [/data/db /data/co...	0B	
<missing>	3 days ago	/bin/sh -c mkdir -p /data/db /data/configdb ...	0B	
<missing>	3 days ago	/bin/sh -c set -x && apt-get update && apt...	282MB	
<missing>	3 days ago	/bin/sh -c echo "deb http://\$MONGO_REPO/apt/...	73B	
<missing>	3 weeks ago	/bin/sh -c #(nop) ARG MONGO_PACKAGE=mongodb...	0B	
<missing>	3 weeks ago	/bin/sh -c set -ex; export GNUPGHOME="\$(mkt...	1.16kB	
<missing>	3 weeks ago	/bin/sh -c #(nop) ENV GPG_KEYS=9DA31620334B...	0B	
<missing>	3 weeks ago	/bin/sh -c mkdir /docker-entrypoint-initdb.d	0B	
<missing>	3 weeks ago	/bin/sh -c set -ex; apt-get update; apt-g...	2.28MB	
<missing>	3 weeks ago	/bin/sh -c #(nop) ENV JSYAML_VERSION=3.10.0	0B	
<missing>	3 weeks ago	/bin/sh -c #(nop) ENV GOSU_VERSION=1.10	0B	
<missing>	3 weeks ago	/bin/sh -c set -eux; apt-get update; apt-g...	7.02MB	
<missing>	3 weeks ago	/bin/sh -c groupadd -r mongodb && useradd -r...	330kB	
<missing>	3 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0B	
<missing>	3 weeks ago	/bin/sh -c mkdir -p /run/systemd && echo 'do...	7B	
<missing>	3 weeks ago	/bin/sh -c rm -rf /var/lib/apt/lists/*	0B	
<missing>	3 weeks ago	/bin/sh -c set -xe && echo '#!/bin/sh' > /...	745B	
<missing>	3 weeks ago	/bin/sh -c #(nop) ADD file:c02de920036d851cc...	118MB	

docker history mongo

docker image insepct mongo

```
[
  {
    "Id": "sha256:4456f3d84674248f83245dc11b2f0394b0a175f63726647f1396c229a6feac79",
    "RepoTags": [
      "mongo:latest"
    ],
    "RepoDigests": [
      "mongo@sha256:efc13c32635f19e1e86850895a5605d8d295e7e4be32ac832da5d63f4c609cb0"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2019-03-29T21:33:05.119204044Z",
    "Container": "bfc1ba359a16d6f7ed049a5bcbe09610e8e2d9daedf08d62b80cb66ab5d55817",
    "ContainerConfig": {
      "Hostname": "bfc1ba359a16",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {
        "27017/tcp": {}
      }
    }
  }
]
```

¿Cómo se crea una imagen?

Dockerfile



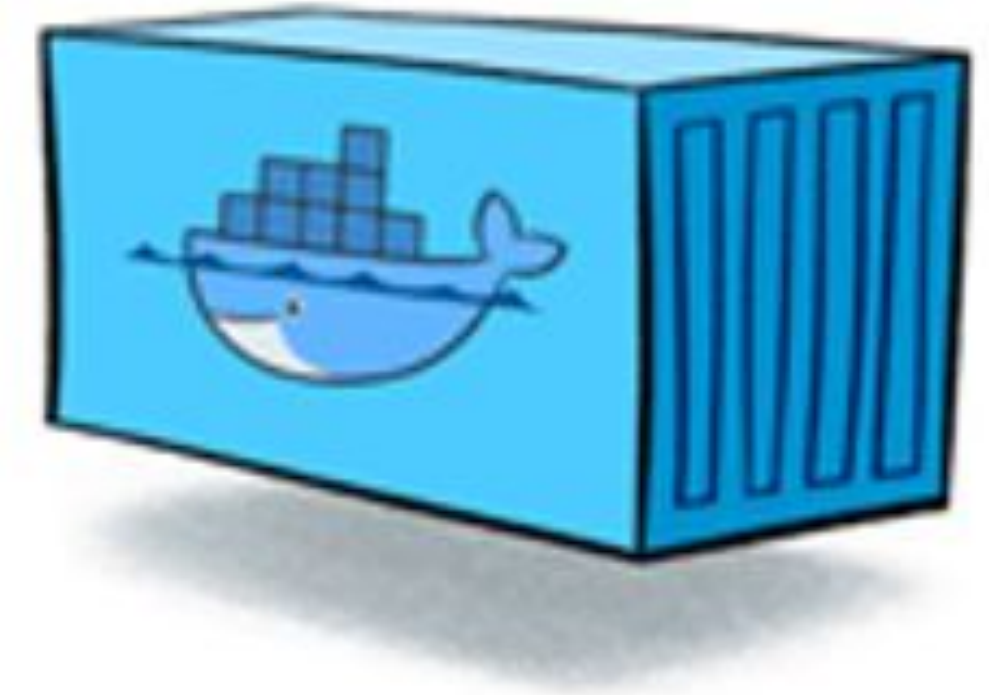
→ Build →

Image



→ Run →

Container



Visual Studio Code

[link](#)

Code editing. Redefined.

Free. Open source. Runs everywhere.

↓ .deb

Debian, Ubuntu...

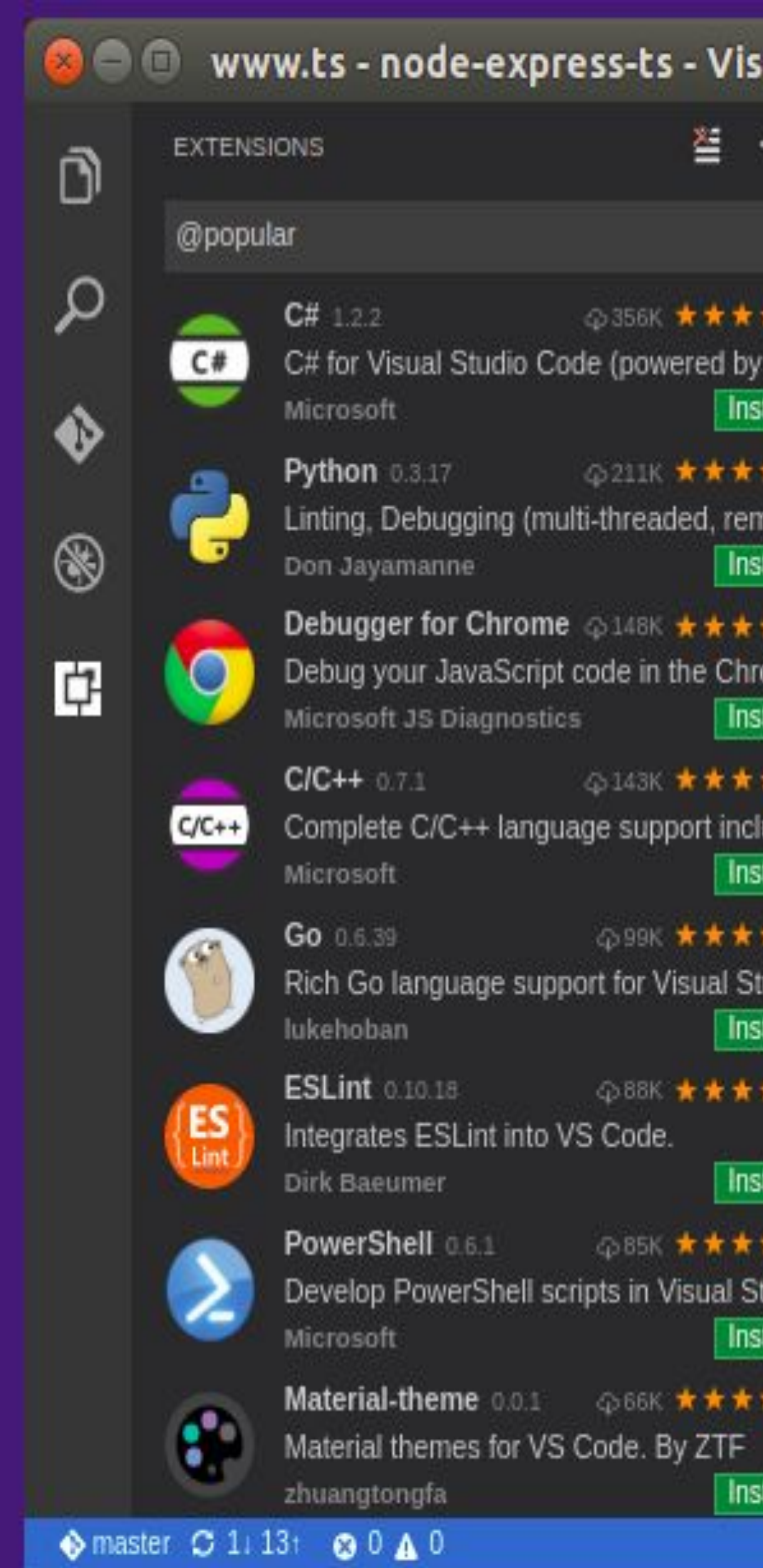
↓ .rpm

Red Hat, Fedora...



Other platforms and Insiders Edition

By using VS Code, you agree to its
license and privacy statement.



Entendiendo nuestro Dockerfile

Dockerfile x

```
1 FROM mysql:5.7.25
2 LABEL maintainer="w.marchanaranda@gmail.com"
3
4 ENV MYSQL_ROOT_PASSWORD=toor
5 ENV MYSQL_DATABASE=demodb
6 ENV MYSQL_ROOT_HOST=%
7
8 COPY ./scripts/ /docker-entrypoint-initdb.d/
9 CMD ["--default-authentication-plugin=mysql_native_password"]
```



Construyendo nuestra imagen | TAG's

Básicamente se trata de extender imágenes para agregarles las funcionalidades que necesitamos, por ejemplo queremos usar un api restful spring boot dentro de un contenedor docker base que tenga java 8. La sintaxis de construcción de una imagen es:

DOCKER BUILD --TAG [IMAGEN_NAME] [CONTEXTO]

ejemplo: *docker build -t wjma90/apirestdemo .*

```
$ cd PATH-DOCKERFILE
```

```
$ docker build -t wjma90/demodb:1.0.0 .
```

```
$ docker run -d -p 3333:3306 --name mysql_server wjma90/demodb:1.0.0
```

```
$ mvn spring-boot:run
```

```
-Dspring-boot.run.arguments=--host=localhost,--port=3333,--database=demodb,--username=root,--password=toor
```

```
$ curl -X POST -H "Content-Type: application/json" -d '{"nombre":"wjma90", "edad":29, "sexo":"M"}'
```

```
http://localhost:8080/api/personas/registrar
```

```
$ docker stop mysql_server
```

```
$ docker start mysql_server (los datos aún están persistentes porque no se ha dañado / eliminado el container)
```



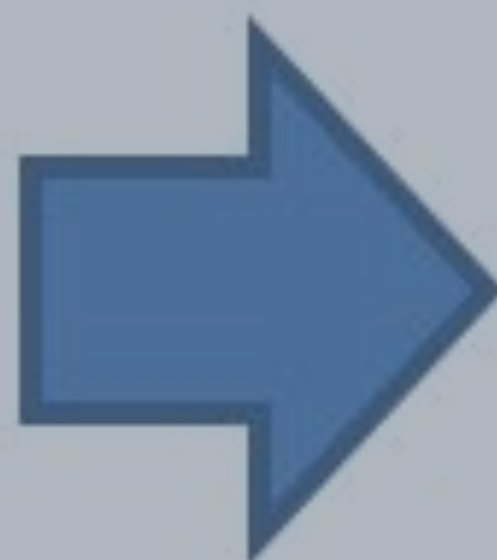
Dockerfile más completo

Dockerfile x

```
1 FROM ibmcom/websphere-traditional:latest
2
3 USER root
4
5 RUN mkdir -p /pr/properties/ && \
6     mkdir -p /pr/logs/ && \
7     chown was:was /pr/webapp/ && \
8     chmod -R 777 /pr/webapp/ && \
9     apt-get update && \
10    apt-get upgrade -y && \
11    apt-get install curl -y
12
13 USER was
14
15 ENV oracle_dns="oracle"
16 ENV oracle_password="password"
17 ENV oracle_sid="XE"
18
19 COPY --chown=was:was ../FUENTES/target/webapp.ear /work/config/webapp.ear
20 COPY --chown=was:was ../tools/ojdbc7.jar /opt/
21 COPY --chown=was:was ../tools/dockerize /usr/local/bin/
22 COPY --chown=was:was ../tools/was-config.props /work/config/was-config.props
23 COPY --chown=was:was ../tools/1-config_datasource.py /work/config/
24 COPY --chown=was:was ../tools/2-install_app.py /work/config/
25
26 RUN /work/configure.sh
```




Dockerfile



Layer by

**Creando
Dockerfiles**

Buenas prácticas en Dockerfiles

1 ORDEN DE COMANDOS

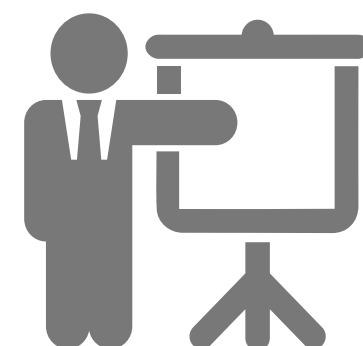
2 USAR COPY vs ADD

3 USAR IMAGENES OFICIALES

4 CONSTRUIR COMPILADOS
EN UN ENTORNO
INMUTABLE

5 CACHE A LAS
DEPENDENCIAS

6 MULTISTAGE PARA
DIFERENTES OBJETIVOS O
ENVIRONMENTS



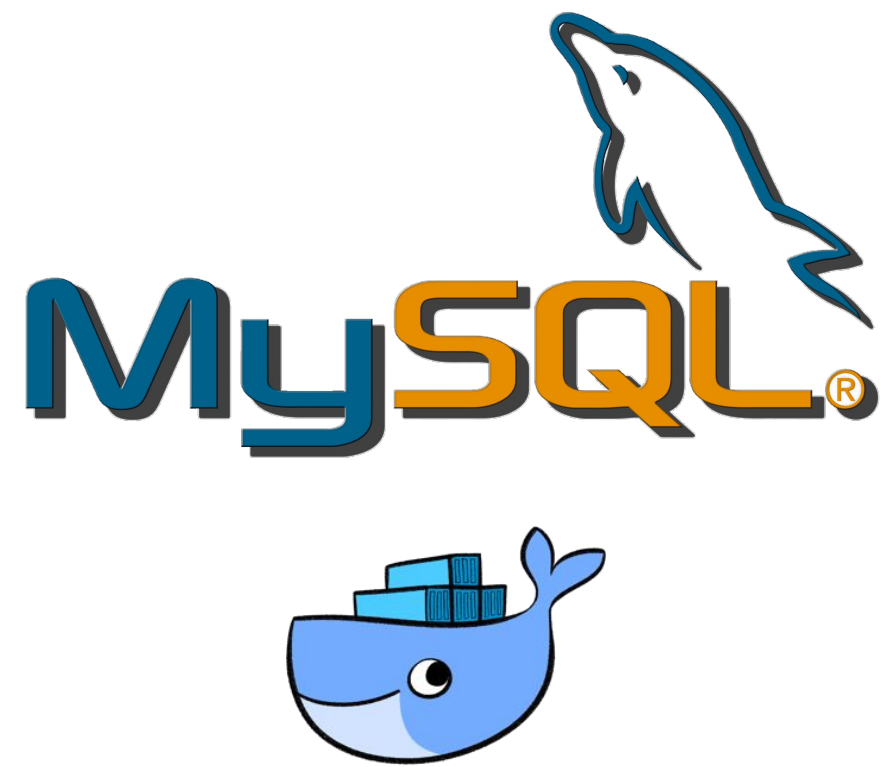
*Hay más recomendaciones que se pueden seguir como por ejemplo: **healthchecks**, **memory_limits**, **dockerignore**, etc, en resumen, siempre es mejor seguir la documentación oficial.*

[aquí](#)



**Link entre
Containers**

Enlazando Containers



Enlace entre Containers

```
$ docker run -d -p 3333:3306 --name mysql_server wjma90/demobd:1.0.0
```

```
$ cd PATHDOCKERFILE/api-persona
```

```
$ mvn clean package -Dmaven.test.skip=true
```

```
$ docker build -t wjma90/apipersonademo:1.0.0 . (verificar que las variables de entorno apunta correctamente a mysql)
```

```
$ docker run -d -p 8080:8080 --link mysql_server --name apipersona wjma90/apipersonademo:1.0.0
```

```
$ docker logs apipersona
```

```
$ curl -X GET http://localhost:8080/api/personas/find/1 && echo " .... terminado "
```

```
....
```

```
....
```

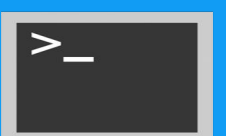
```
....
```

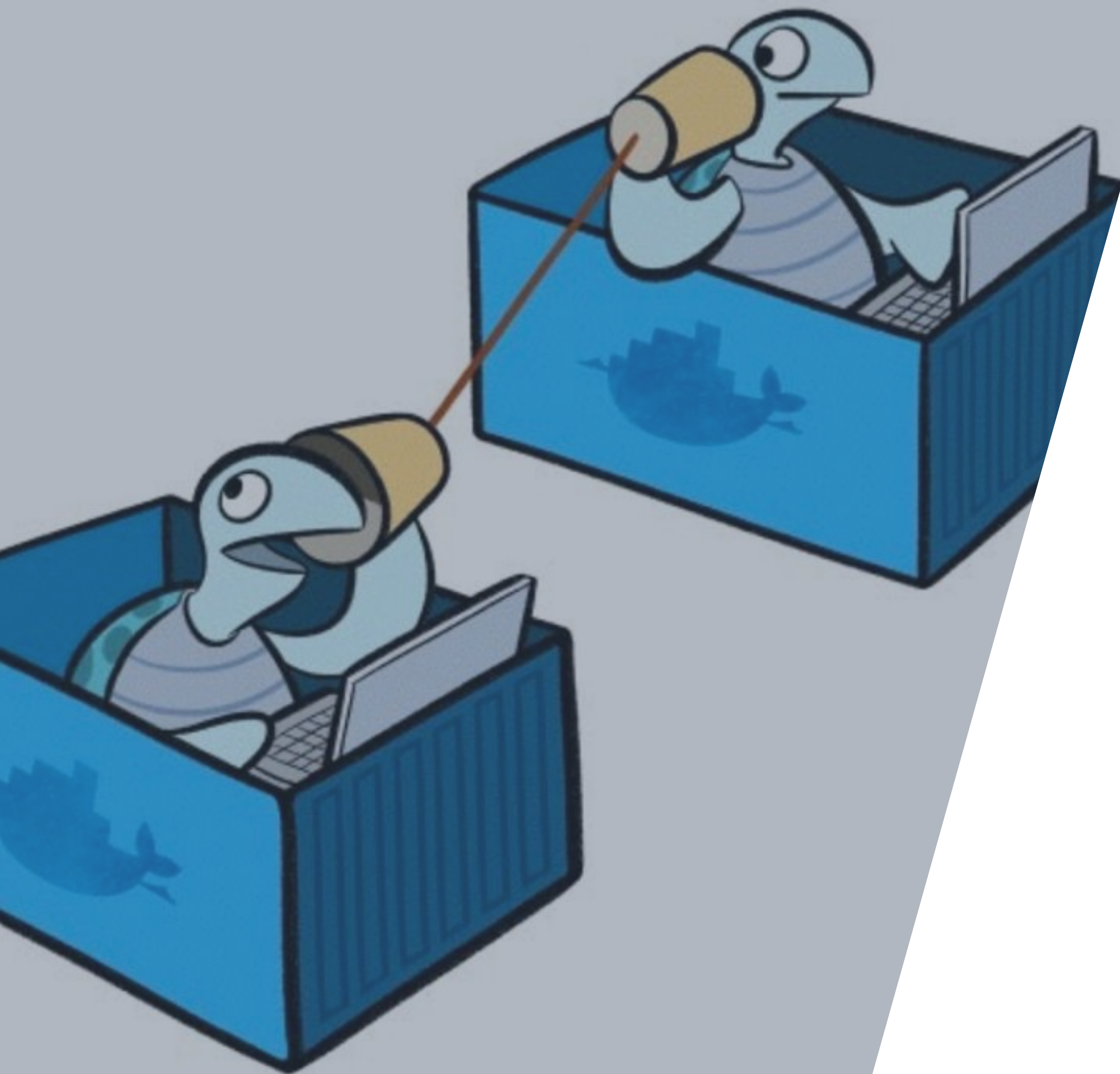
```
....
```

```
....
```

```
....
```

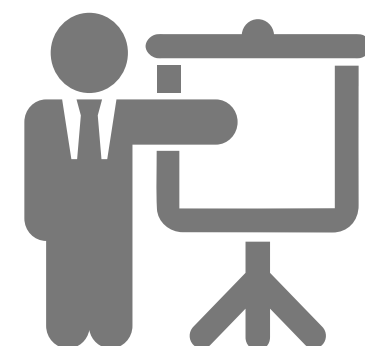
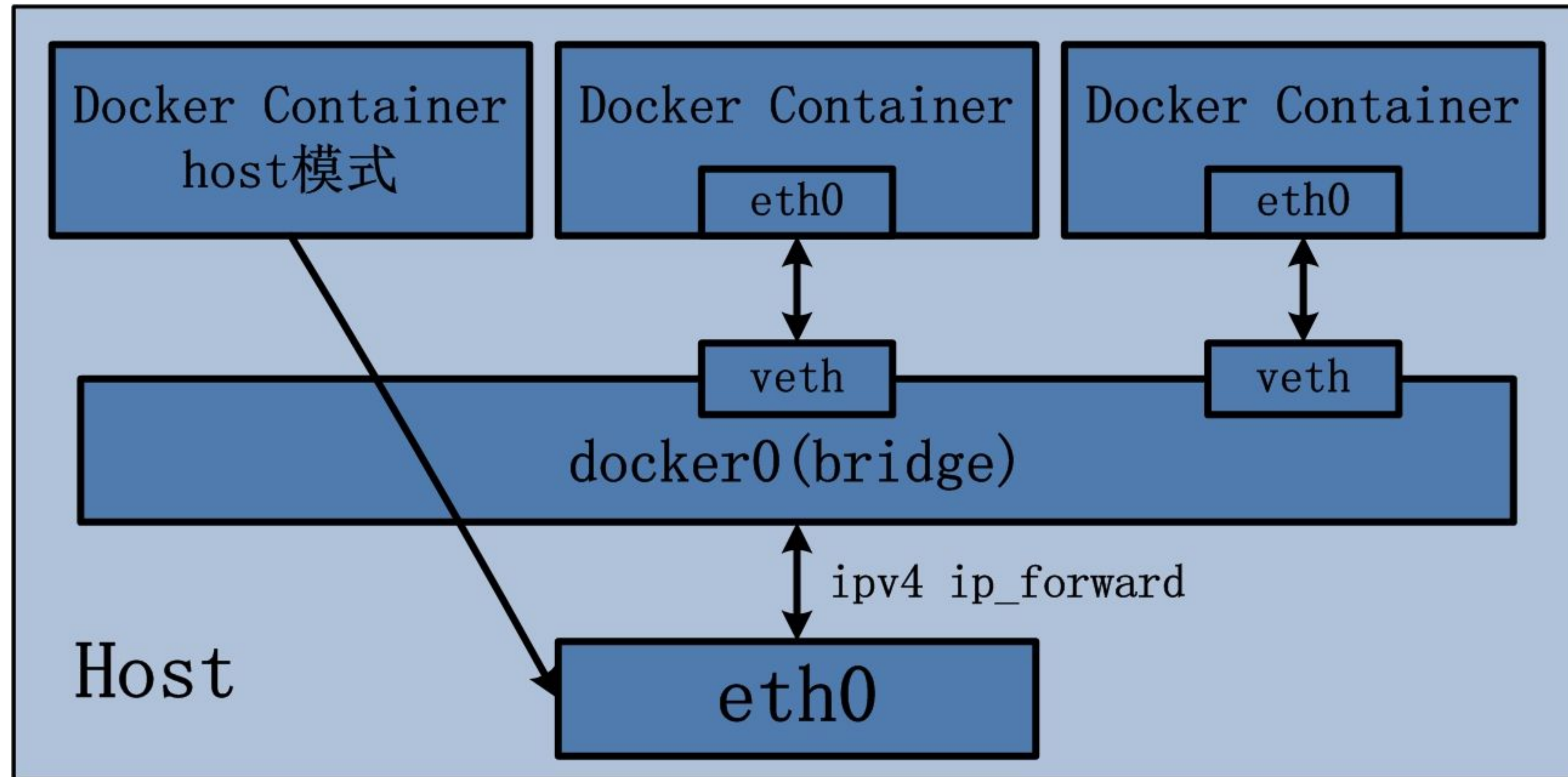
```
....
```





Docker Network

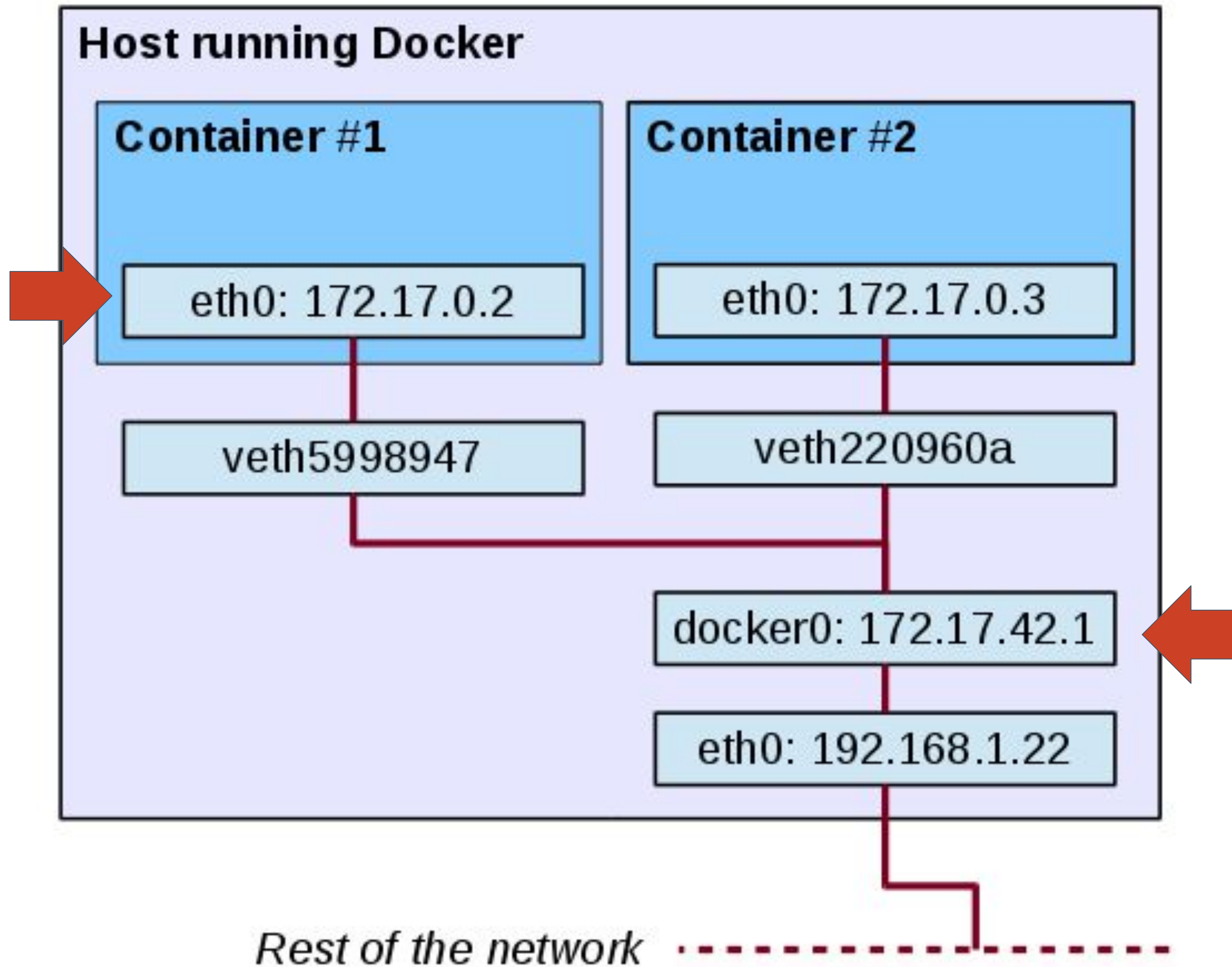
Driver Bridge, Host y None



Docker None hace que el contenedor no tenga salida hacia el host ni hacia otros containers

Driver Bridge

Por defecto, Docker instala una interface en nuestro host. El driver bridge tiene rango de red: 172.17.0.0/16



Explorando el enlace entre containers

```
$ docker ps -a $(docker ps -aq) (si tienen contenedores, eliminarlos previamente)
$ docker run -d -p 3333:3306 --name mysql_server wjma90/demobd:1.0.0
$ docker run -d -p 8080:8080 --link mysql_server --name apipersona wjma90/apipersonademo:1.0.0
```

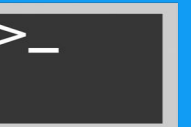
```
$ docker network ls
```

```
$ docker exec -it apipersona sh
```

```
root@7d57ccc5:# cat /etc/hosts ¿Qué info muestra?
```

```
$ docker network inspect bridge
```

```
[{
  "Name": "bridge",
  "Id": "ffe1a880c250b9d85c3772f9e4171dbe112bfbed3c16bdb7dc9477a21e6570ec",
  "Created": "2019-04-01T09:55:45.782870497-05:00",
  "Scope": "local",
  "Driver": "bridge",
  "EnableIPv6": false,
  "IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
      {
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1"
      }
    ]
  }
}]
```



Implementando la topología de red

```
$ docker network create --driver bridge test
```

```
$ docker run -d --network=test --name mysql_server wjma90/demobd:1.0.0
```

```
$ docker run -d -p 8080:8080 --network=test -e host=mysql_server --name apipersona wjma90/apipersonademo:1.0.0
```

¿Logra conectarse?

```
$ docker exec -it apipersona sh
```

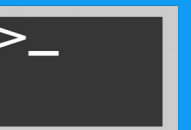
```
root@7d57ccc5:# apk add curl
```

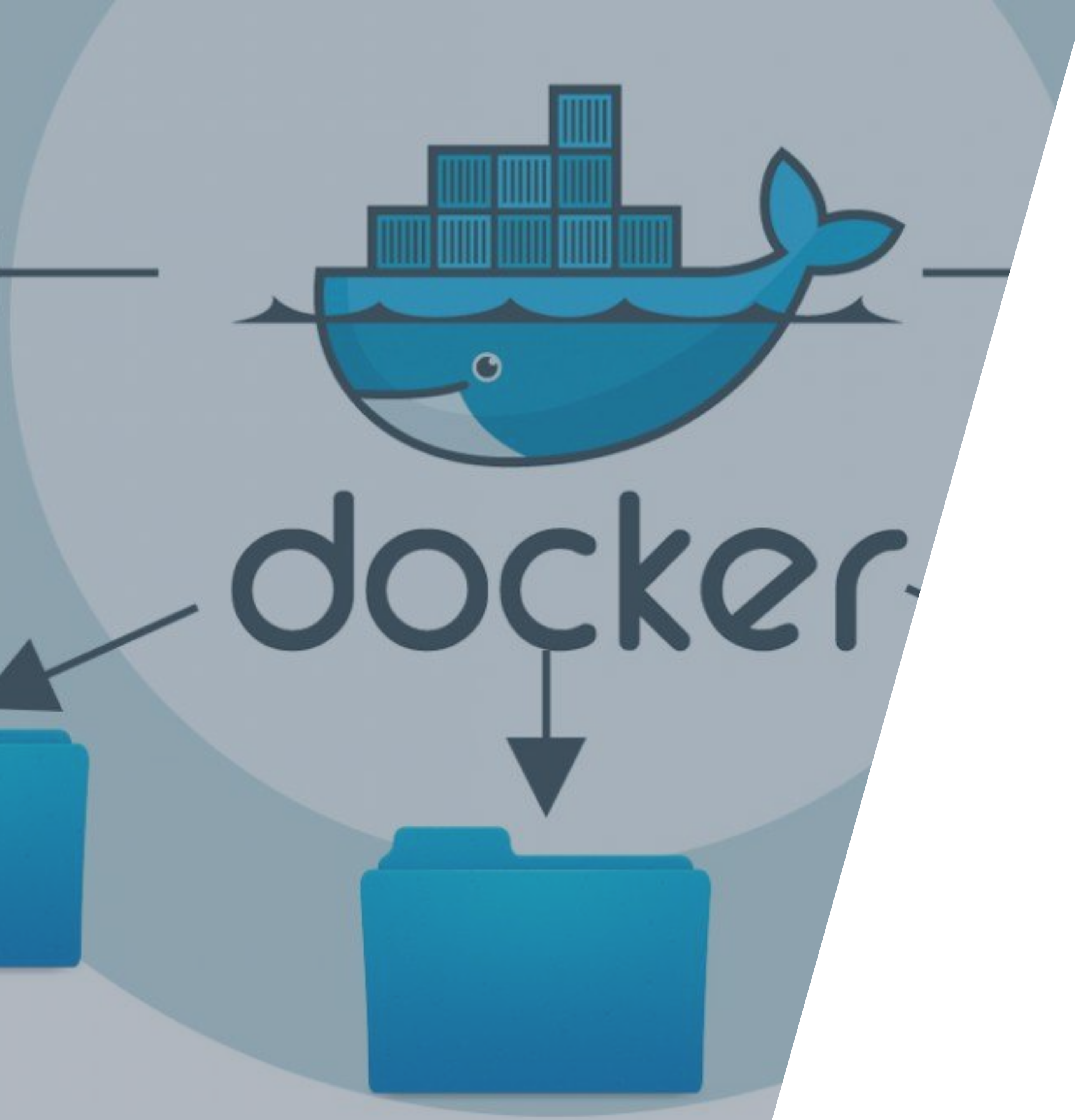
```
root@7d57ccc5:# curl -X GET http://localhost:8080/api/personas/find/1
```

```
$ docker network connect bd_demo apipersona
```

```
$ docker exec -it apipersona sh
```

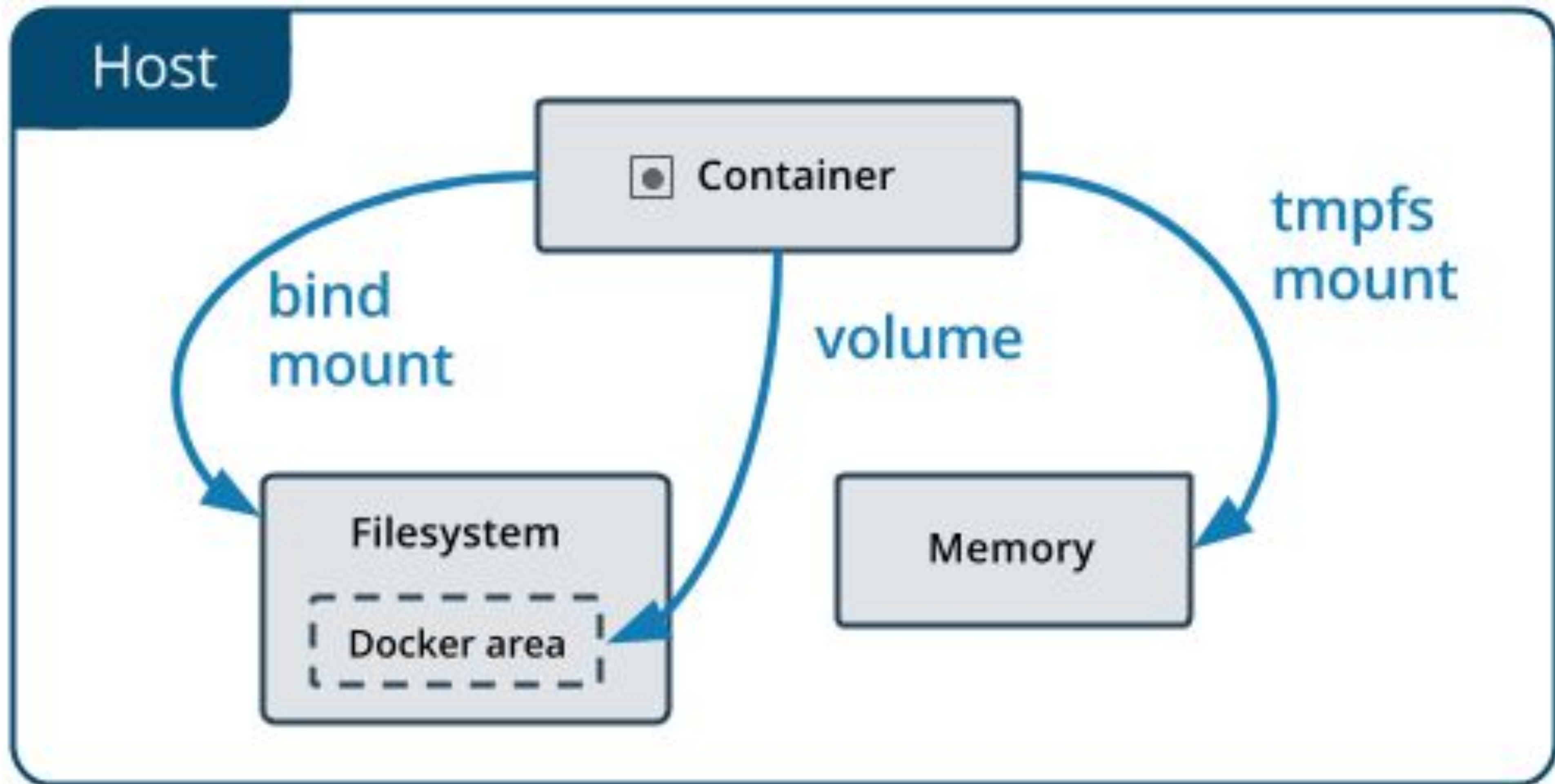
```
root@7d57ccc5:# curl -X GET http://localhost:8080/api/personas/find/1
```





Volumen en Docker

Tipos de Volumen



Practicando volúmenes

```
$ docker volume create myvolume
```

```
$ docker volume inspect myvolume
```

```
$ docker run -d -p 3310:3306 -v /home/william/Escritorio/volume-demo:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=toor  
--name mysql_server mysql:5.7.25
```

```
docker run -d -p 3310:3306 -v myvolume:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=toor --name mysql_server mysql:5.7.25
```

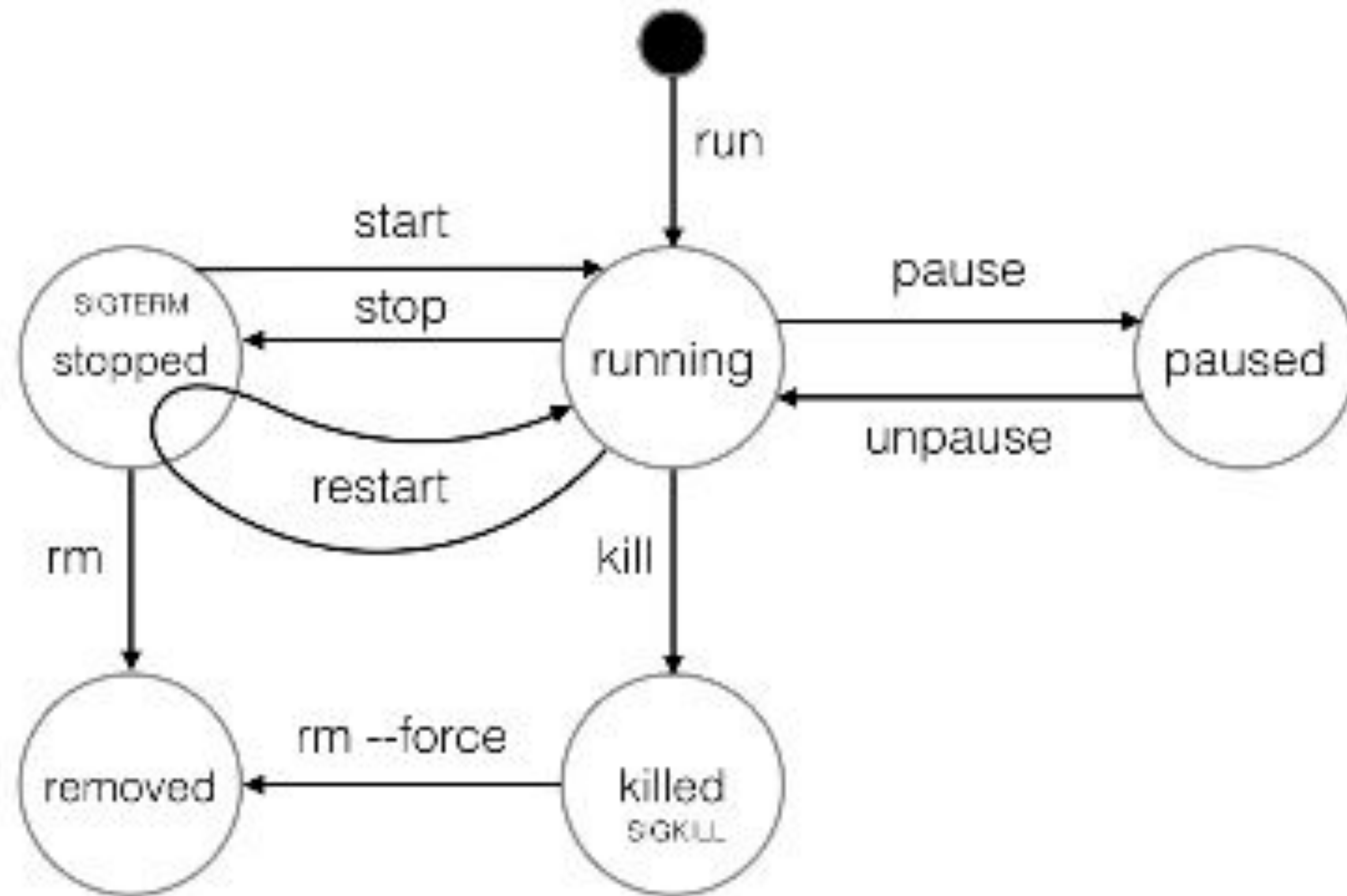
```
docker run -d -p 3310:3306 --mount type=bind,source=/home/william/Escritorio/bindmount,target=/var/lib/mysql \  
--name mysql_server -e MYSQL_ROOT_PASSWORD=toor mysql:5.7.25
```

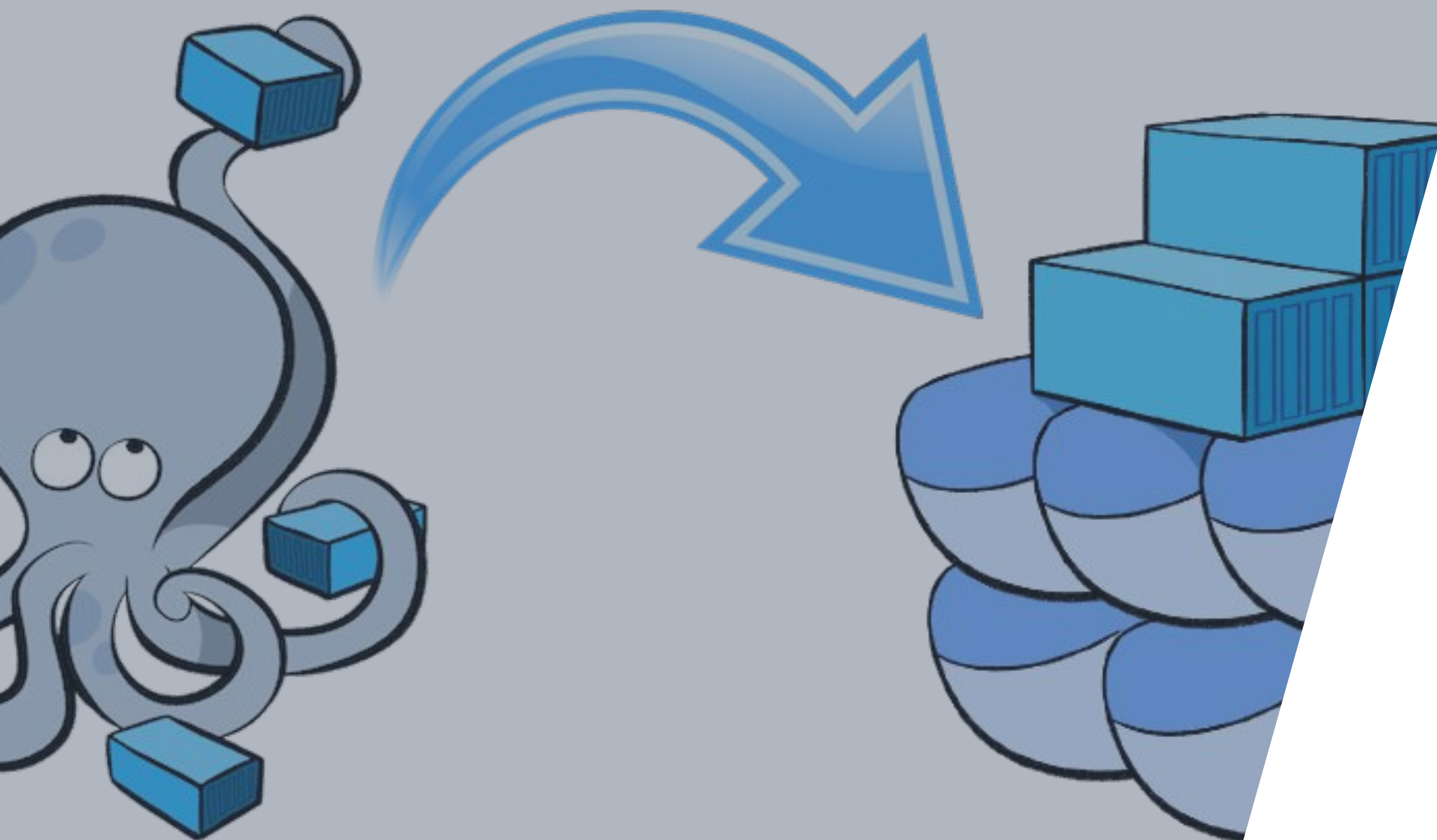
****En type=bind (con --mount), si no existe la ruta, docker dará error.**

```
....  
....  
....  
....  
....  
....  
....  
....
```



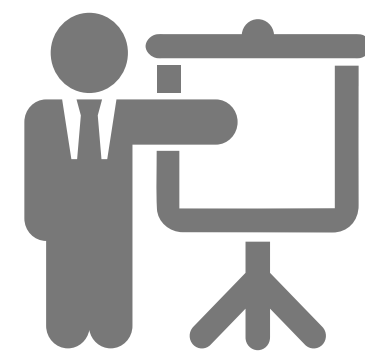
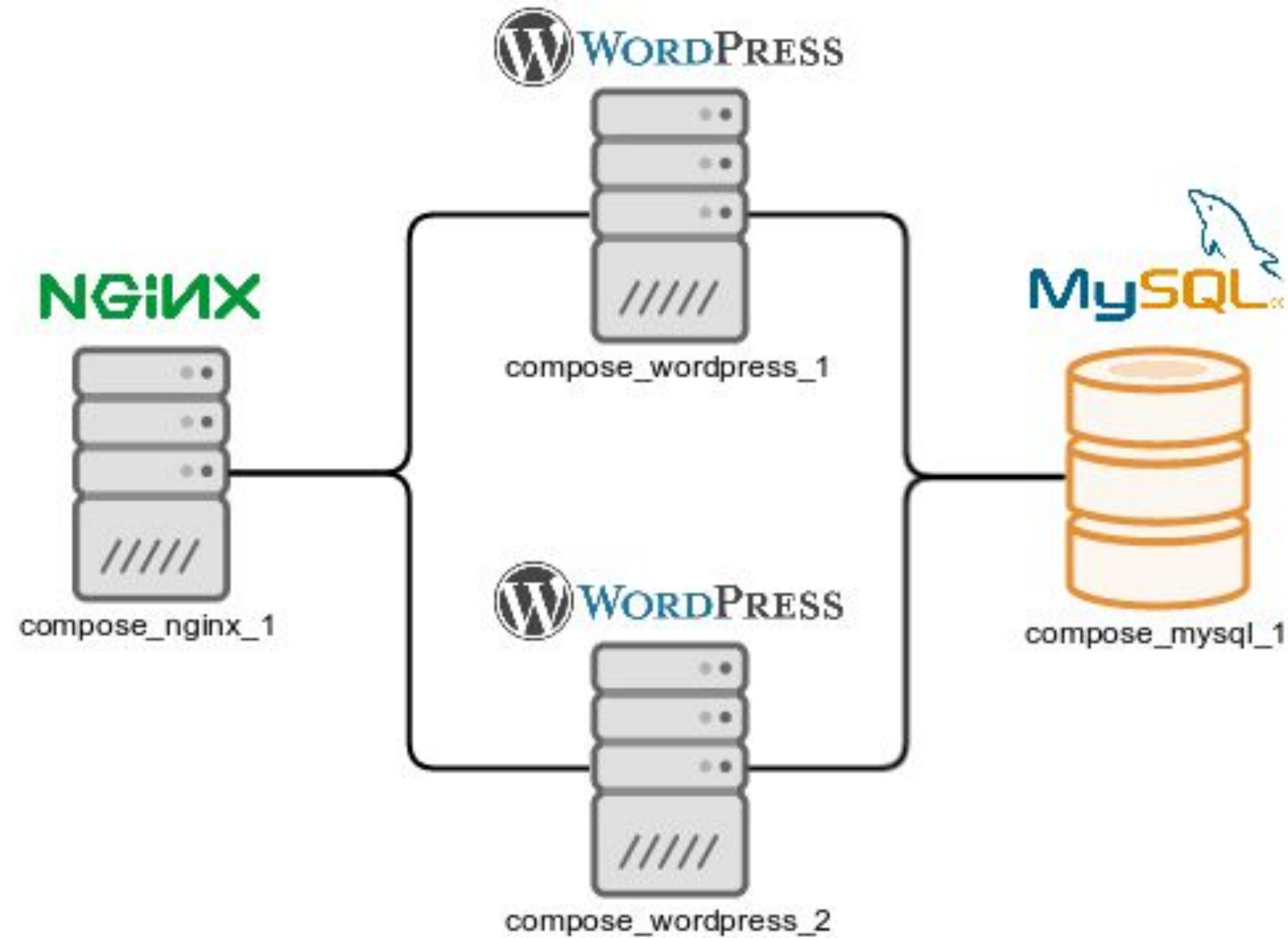
Ciclo de vida de un Container





Docker
Compose

¿Qué es Docker Compose?

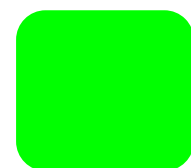


Es una herramienta que nos permite definir y “correr” múltiples contenedores. Simplificando la utilización de comandos docker

Estructura básica de un archivo docker compose



Versión docker compose



Definición de ejecución de un container

```
version: '2'
```

```
services:
```

```
  db:
```

```
    container_name: 'mysql-blog'
```

```
    image: 'mysql'
```

```
    ports:
```

```
      - '3306:3306'
```

```
    environment:
```

```
      - MYSQL_ROOT_PASSWORD= '12345'
```

```
      - MYSQL_DATABASE= 'blog'
```

```
      - MYSQL_USER= 'root'
```

```
    volumes:
```

```
      - ./mysql:/var/lib/mysql
```

```
  web:
```

```
    container_name: wordpress-blog
```

```
    image: wordpress:4.7.5-apache
```

```
    volumes:
```

```
      - ./code:/var/www/html
```

```
    ports:
```

```
      - '8080:80'
```

```
    links:
```

```
      - 'db:mysql'
```

```
    depends_on:
```

```
      - db
```

Estructura básica de un archivo docker compose

1

Version: Indica la versión del archivo docker compose, dependiendo de ésta, usaremos algunas propiedades permitidas

2

Services: Aquí indicaremos las propiedades de los contenedores que podemos etiquetarlos como db , wp. Imagine que son los docker run

3

Image: Indicamos la imagen docker a utilizar, también podemos utilizar un dockerfile mediante un propiedad más llamada build

4

Ports: Es el equivalente a -p en docker run. Nos permite usar un puerto interno del contenedor en un puerto de nuestro host

5

Environments: Equivalente a -e en docker run. Permite inyectar variables de entorno al contenedor. Puede sobre-escribir lo indicado en el dockerfile

6

Volumes: Equivalente a -v en docker run. Nos permite relacionar una carpeta de nuestra máquina con una interna del contenedor.

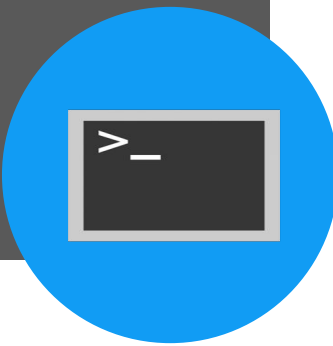
1er Docker Compose

```
version: '3.1'

services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    depends_on:
      - db

  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
```

A blue circular icon containing a white terminal window with a prompt character and an underscore.

Comandos Básicos en Docker Compose

```
$ docker compose build
```

```
$ docker compose up -d
```

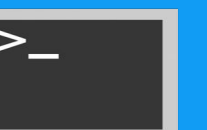
```
$ docker compose ps
```

```
$ docker compose down -v
```

```
$ docker compose up -d wordpress --no-build --no-deps
```

```
$ docker compose stop wordpress
```

```
$ docker compose scale appweb=2, appbackend=3
```



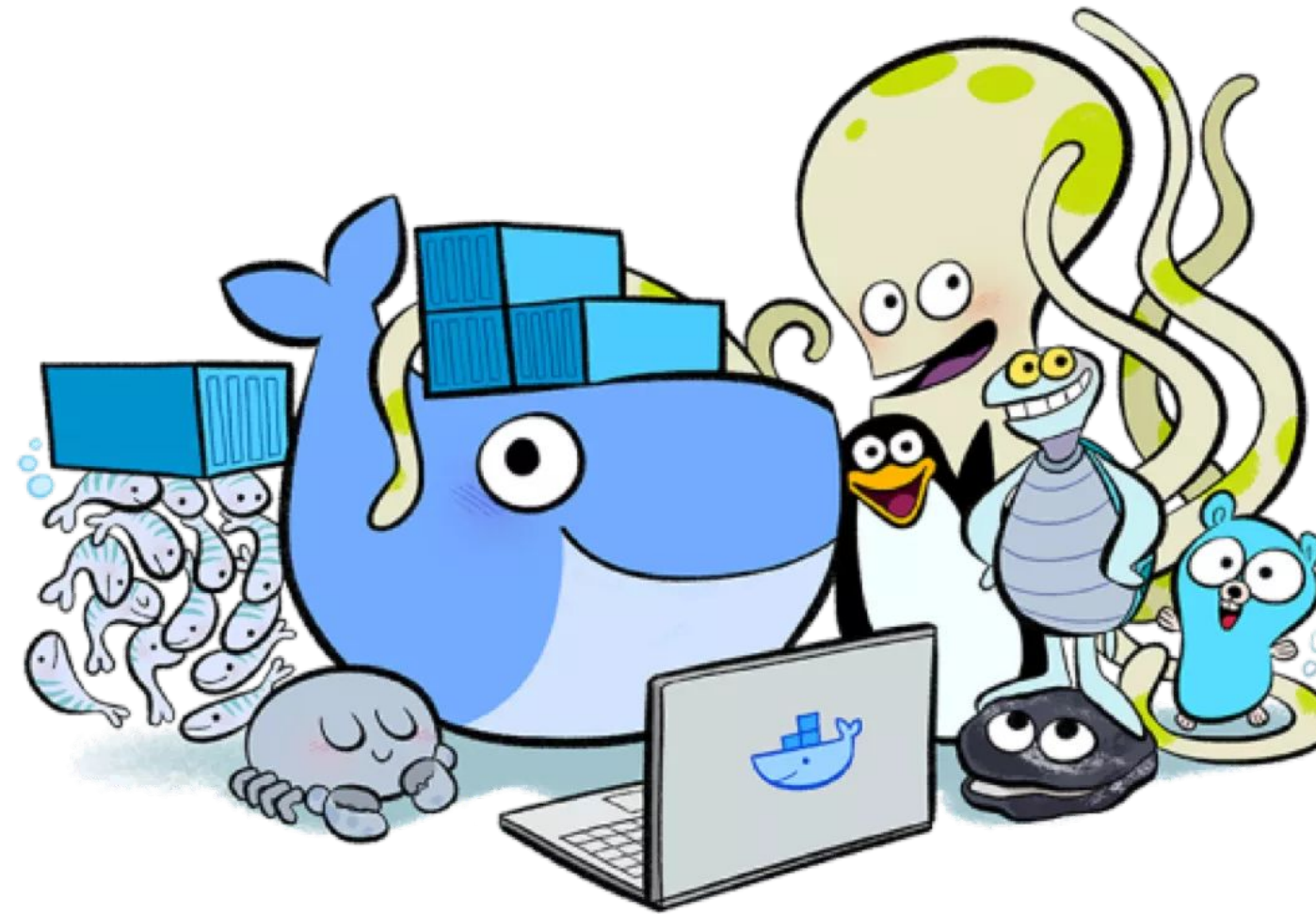
Diferencia entre versión 2 y 3 de Compose

A. Fines de desarrollo / Testing

B. Propiedades soportadas son diferentes a la versión 3. Ejemplo --memory-limit

C. No es antiguo o desfasado como se menciona o se pueda entender, la versión 2.3 salió a la par con la version 3.4 de docker compose

D. No orientado a SWARM



A. Fines de desarrollo / Testing & Cluster Swarm

B. La limitación de recursos es para fines swarm. No se soporta limites de recursos en modo "sin enjambre"

C. Lo recomienda el mismo Docker ya que a futuro se integrará limpiamente a Kubernetes

D. Orientado a SWARM



¿Dudas?

w.marchanaranda@gmail.com

