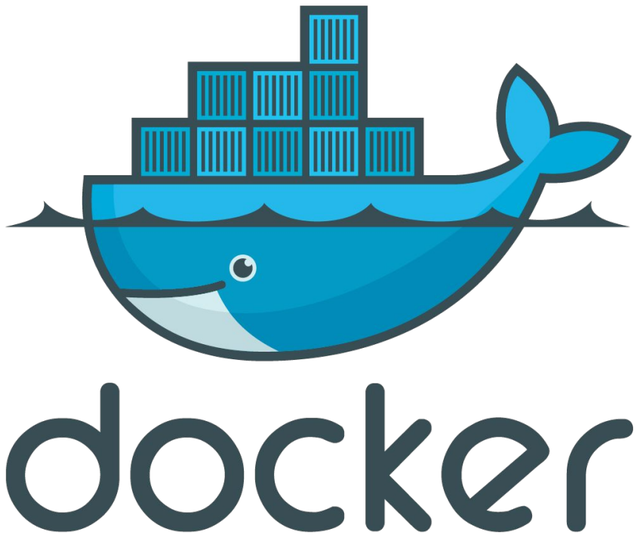


¿Qué es Docker?

¿Qué es Docker?

- Es una plataforma para desarrollar, distribuir y poner en marcha aplicaciones usando la una tecnología de virtualización: contenedores.

- Sólo se puede utilizar en sistemas Linux, aunque podemos utilizarlo en Mac OS X y Windows utilizando máquinas virtuales.

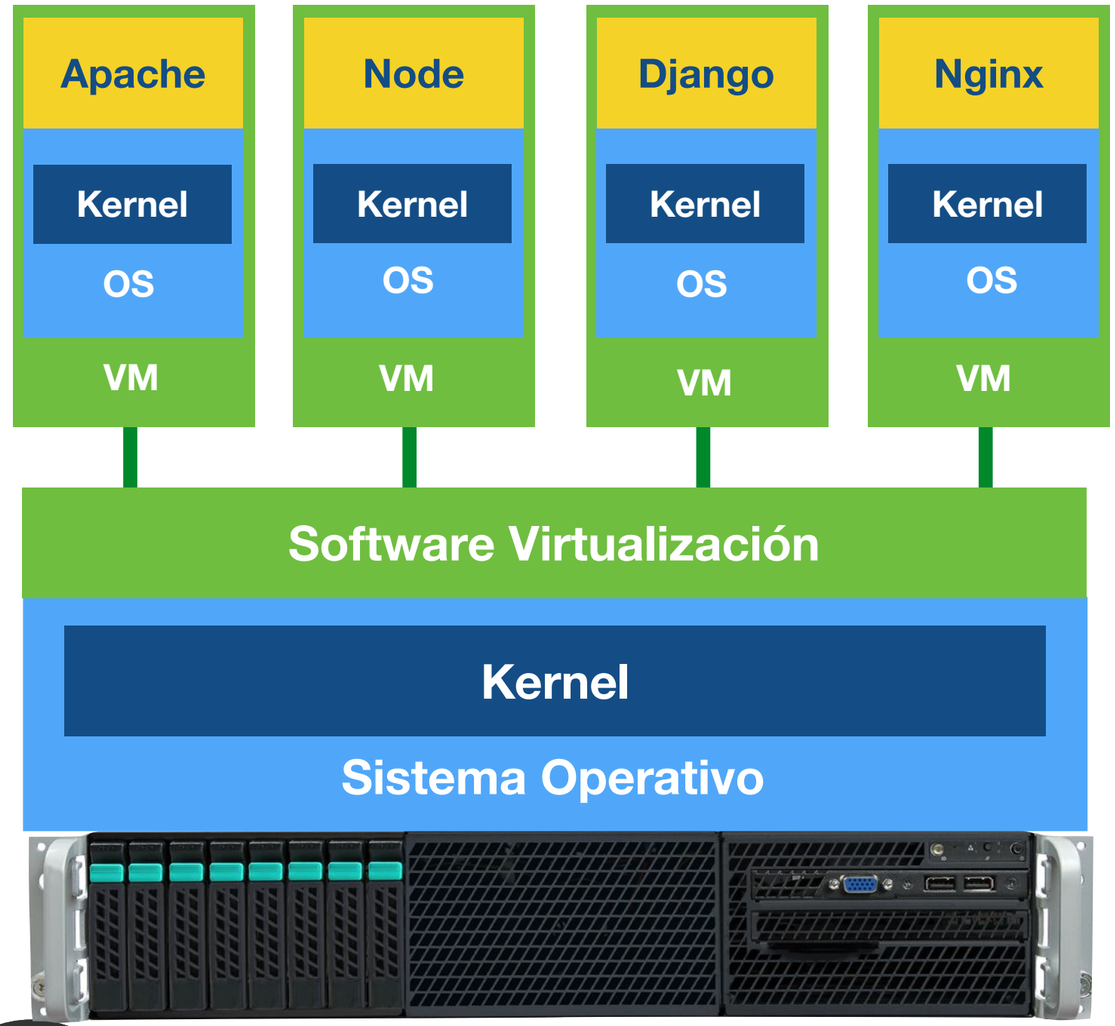
- Para producción: tendremos que utilizar Linux.

Virtualización de contenedores

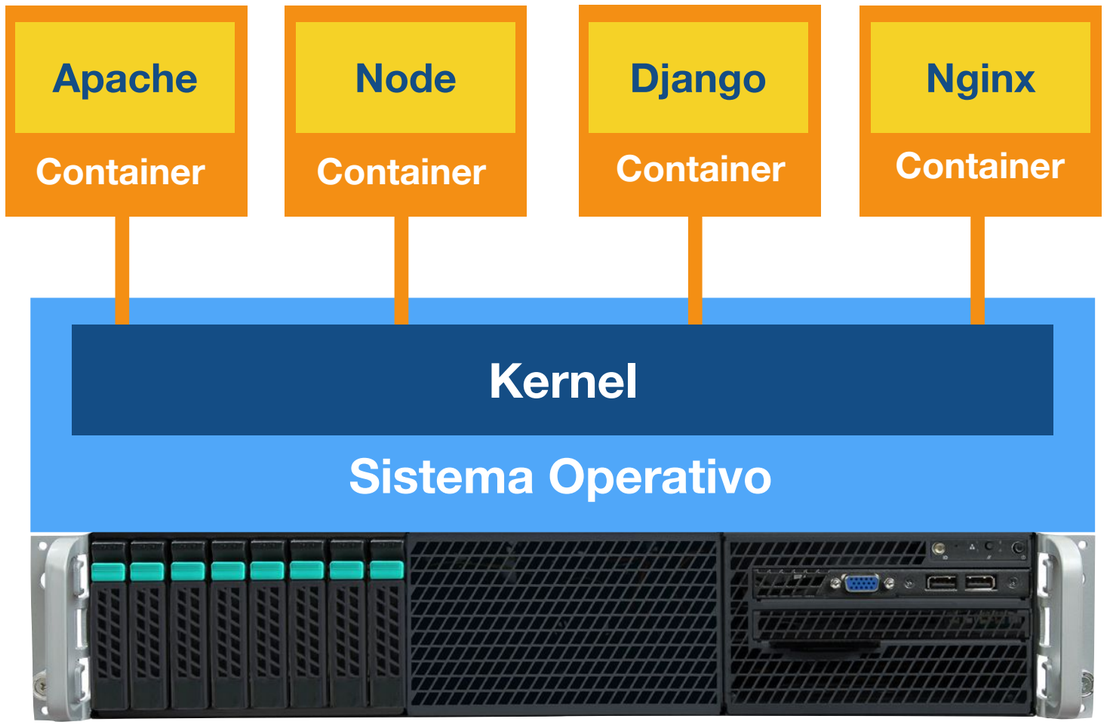
Virtualización de contenedores

- Es una tecnología propia de Linux (no es de Docker)
- Vistos desde fuera, los contenedores son como máquinas virtuales
- Cada contenedor tiene su propio sistema de ficheros, procesos, memoria y puertos de red (como una máquina virtual)
- La diferencia es que utilizan el kernel del sistema operativo sobre el que se crean (todos los Linux usan el mismo kernel)
- Por eso son mucho más ligeros que una máquina virtual

Máquinas Virtuales vs Contenedores

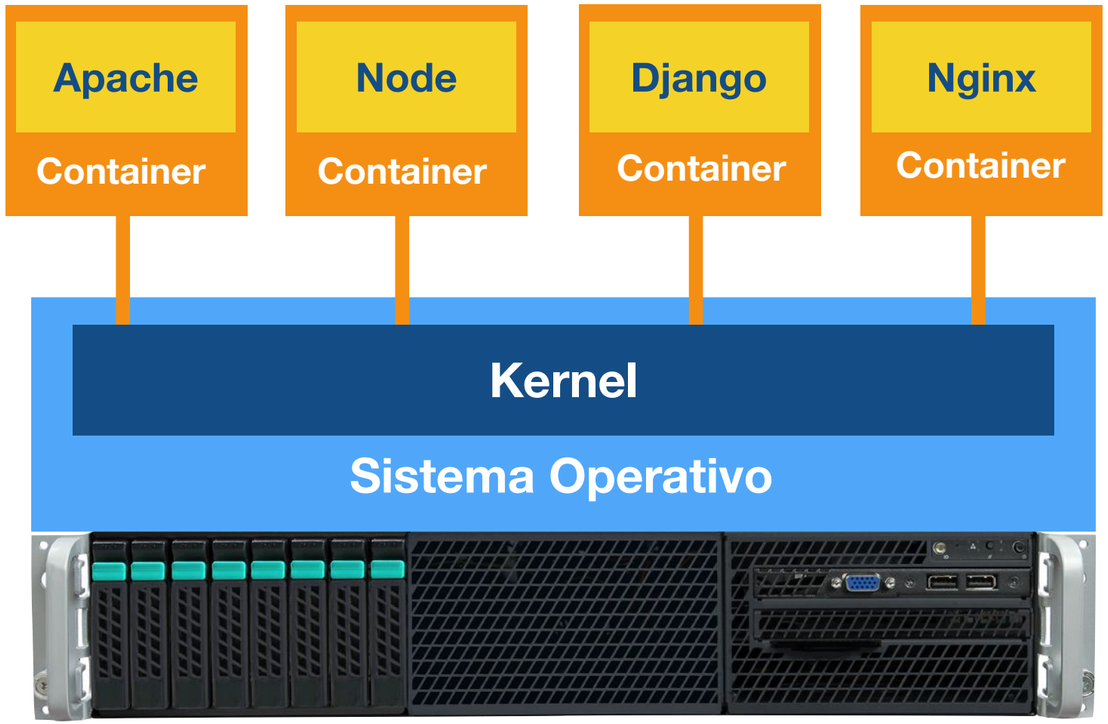


Máquinas Virtuales

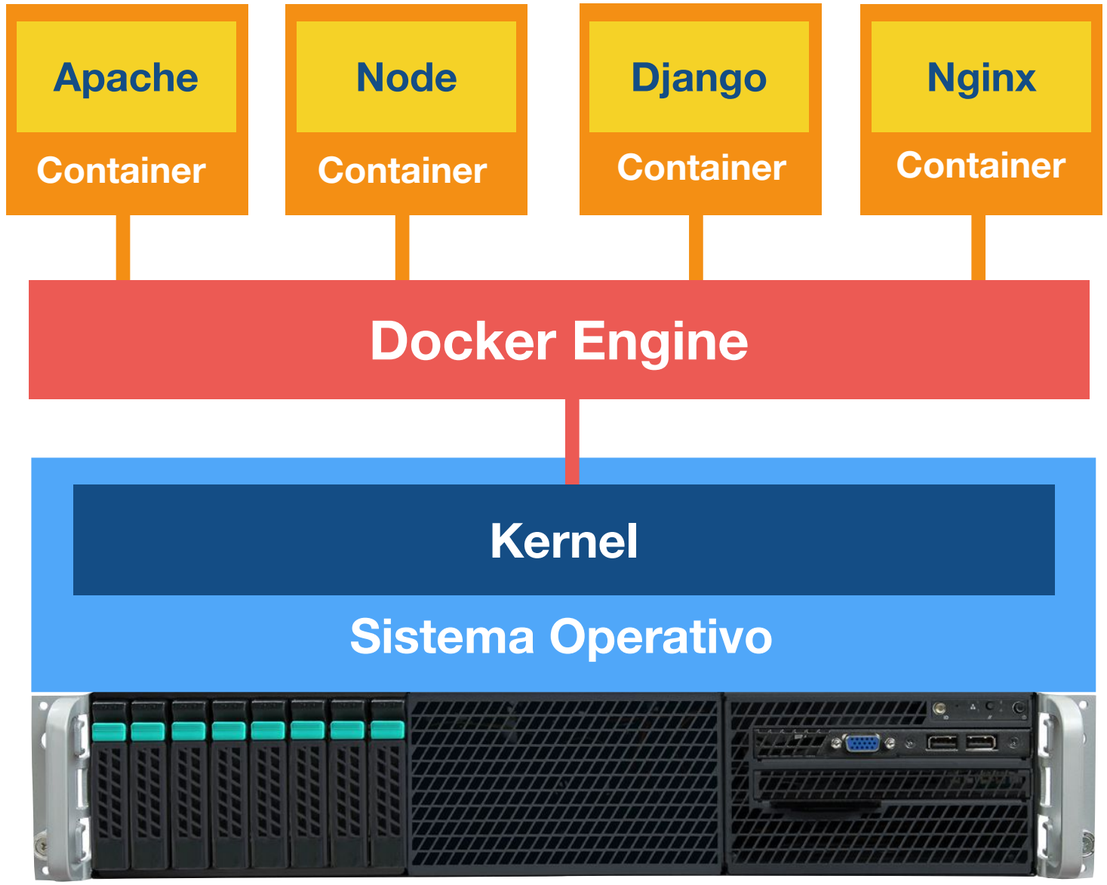


Contenedores

Contenedores vs. Contenedores con Docker



Contenedores



Contenedores

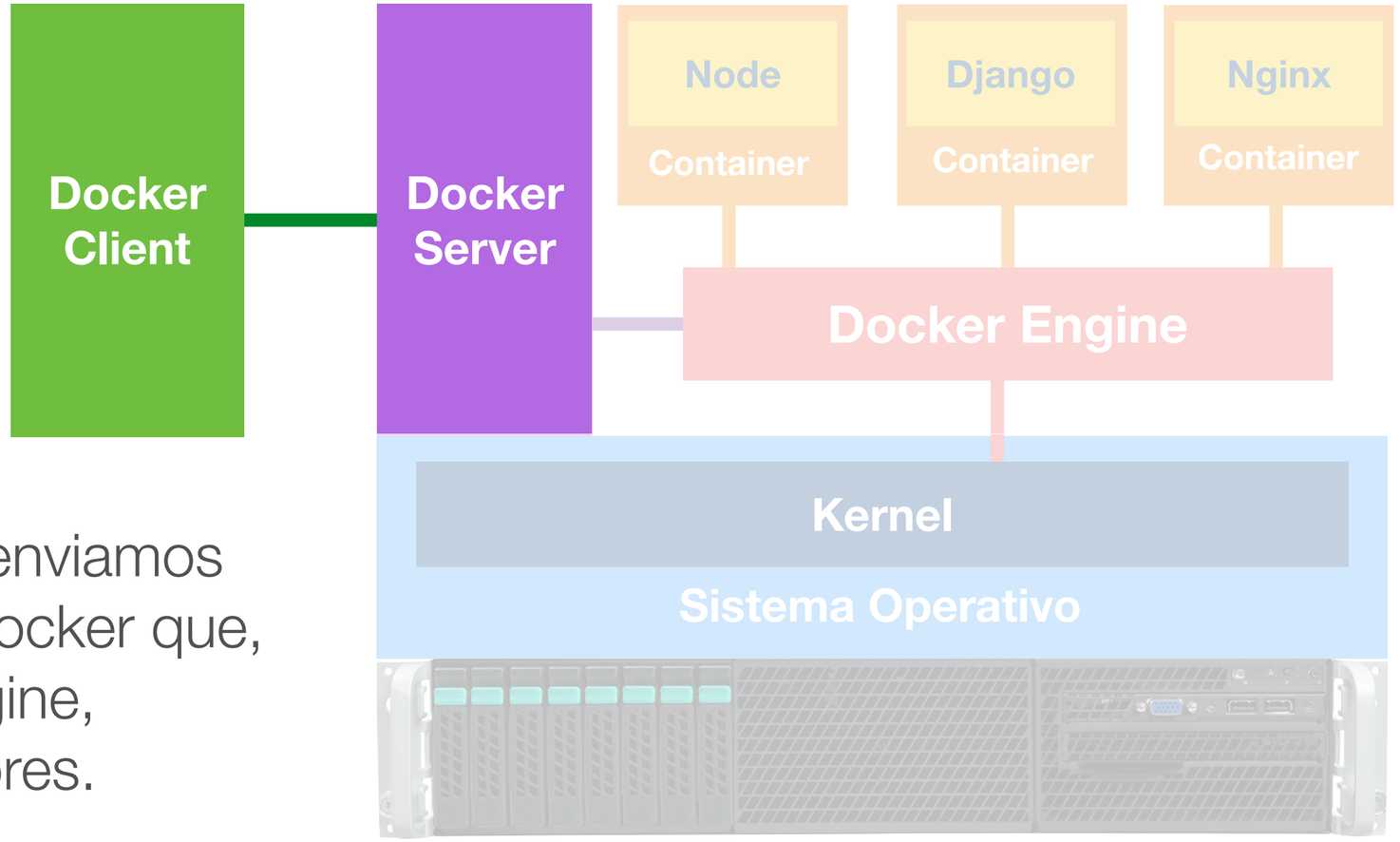
Beneficios de Docker

- Los desarrolladores pueden montar sus aplicaciones en contenedores.
- Los DevOps sólo deben encargarse de desplegar contenedores (no tienen que saber cómo funciona una aplicación dentro de un contenedor para que funcione).
- Mayor portabilidad de una aplicación y mayor escalabilidad
- Más aplicaciones en un host que con máquinas virtuales

¿Cómo funciona Docker?

¿Cómo funciona Docker?

Arquitectura Cliente-Servidor



Desde el cliente de Docker enviamos instrucciones al servidor de Docker que, a través de Docker Engine, gestiona los contenedores.

¿Cómo funciona Docker?

Desde el **cliente** de Docker podemos descargar **imágenes** de un **repositorio** de un **registro** para arrancar **contenedores** que se ejecuten en el **host** controlado por el **servidor** de Docker.

El cliente de Docker

El cliente de Docker

Hay dos tipos de cliente:

- Docker CLI: Command Line Interface / Consola de comandos.
- Kitematic: Interfaz de Usuario. Actualmente en beta.

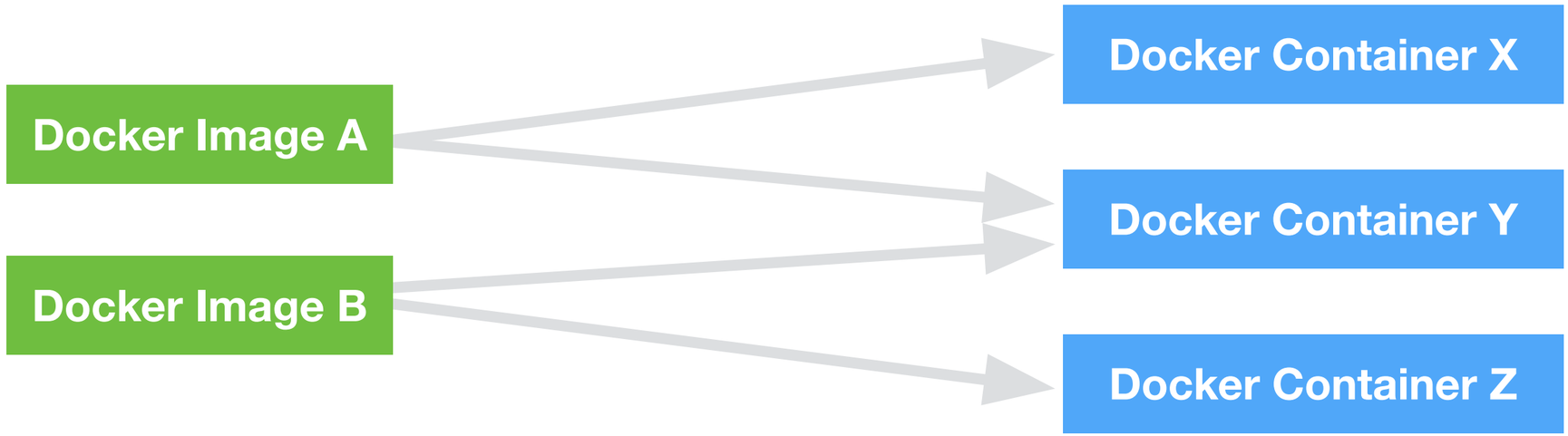
Imágenes y contenedores

¿Qué son las imágenes?

- Son plantillas para crear contenedores
- Son de sólo lectura (no podemos modificarlas)
- Las creamos nosotros u otros usuarios de Docker
- Se almacenan en repositorios (y éstos en un registro)
- Docker Hub es un registro público de Docker

¿Qué son los contenedores?

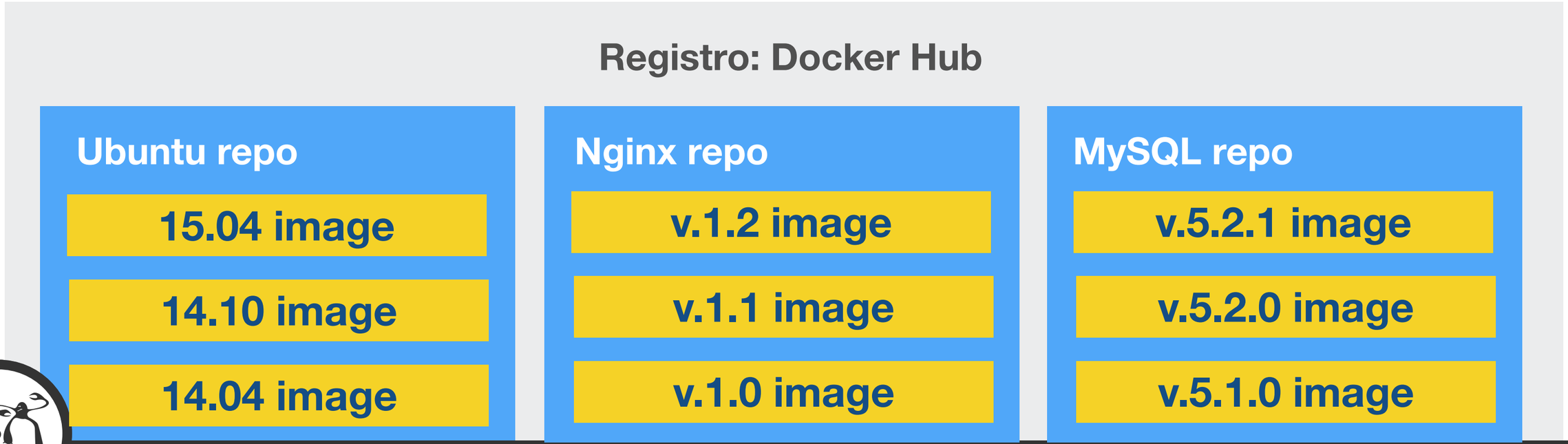
- Son una o más imágenes siendo ejecutadas
- Unidades de ejecución aisladas con su propio espacio de memoria, procesos, sistema de red y sistema de archivos

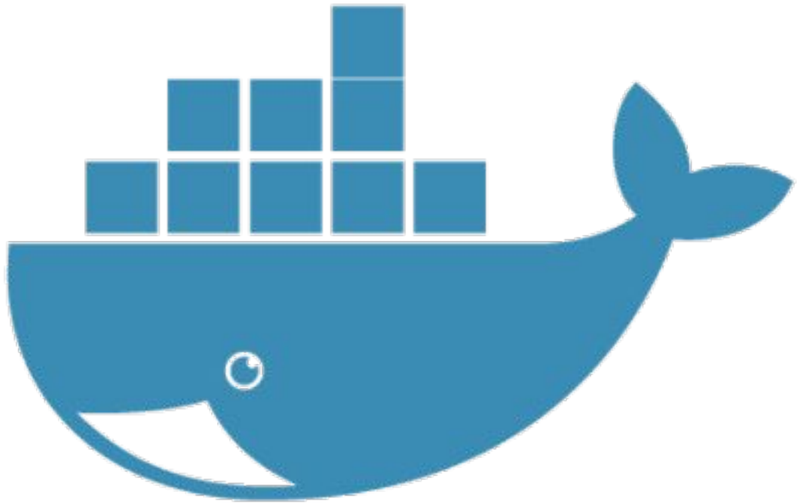


Registros y repositorios

Registros y repositorios

- Un registro es un lugar donde se almacenan repositorios
- En los repositorios podemos encontrar diferentes versiones de una imagen (como si un repo de Github se tratara). Se identifican por tags.



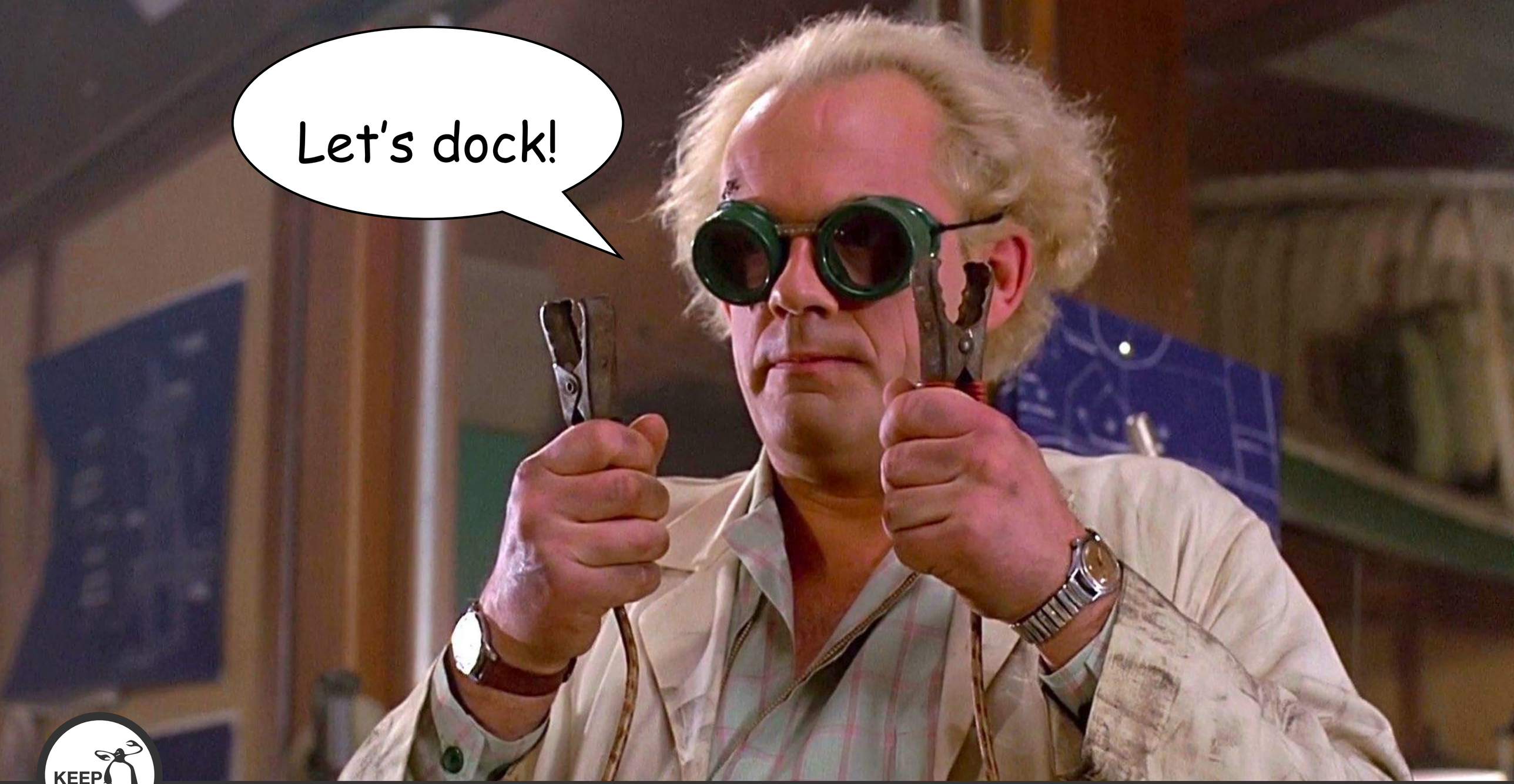


dockerhub

Docker hub

- Registro público oficial de repositorios de Docker
- Podemos encontrar imágenes oficiales de los principales proyectos open source: Ubuntu, Nginx, node, etc.
- Generalmente, basaremos nuestras imágenes en imágenes oficiales de Docker Hub para crear nuestras aplicaciones.
- Es gratis 👋

https://hub.docker.com



En Windows y OS X

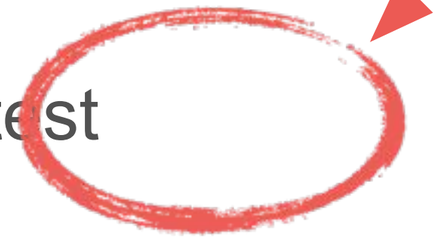
Docker Quickstar Terminal

Hello world

docker run hello-world

Hello world

Cuando no indicamos ningún tag en la imagen, Docker usa por defecto latest



1. Docker descarga la imagen **hello-world:latest**
2. Arranca un contenedor basado en la imagen descargada
3. El contenedor ejecuta el comando **echo “Hello World”**
4. El contenedor termina su ejecución

Mostrar imágenes locales

```
docker images
```

Ejecutar un contenedor a partir de una imagen

docker run <image>:<tag> <command>

Al terminar de ejecutar el comando, el contenedor finaliza (termina)

Ejecutar un contenedor y acceder a su terminal

docker run -t -i run <image>:<tag> /bin/bash

-i conecta a la entrada estándar STDIN del contened
(para poder teclear)
-t solicita coger un pseudo-terminal

Aunque hagamos cambios en un contenedor, cuando finalice su ejecución no se guardarán dichos cambios.

Procesos de un contenedor

Cuando ejecutamos un contenedor, el comando que le pasamos se ejecuta con PID 1 (dentro del contenedor).

Este proceso podemos verlo desde nuestro sistema, aparecerá como proceso hijo del proceso docker.

Lista de contenedores en ejecución

docker ps

Podemos salir de un contenedor
sin cerrar bash con CTRL + P + Q

Ejecutar un contenedor en background

docker run -d <image>:<tag> <command>

-d detached mode

Ver salida de un contenedor en background

docker logs <container-ID>

Parar un contenedor en background

```
docker stop <container-ID>
```

Ver contenedores ejecutados (parados)

```
docker ps -a
```

Volver a arrancar un contenedor

```
docker start <container-ID>
```

Mapear puertos de un contenedor

docker run <source>:<target> <image>

Guardar cambios en un contenedor: nueva imagen

docker commit <container-id> <repo>:<tag>

Crea una nueva imagen llamada **<repo:tag>**
Si no indicamos **<tag>**, será **latest**

Dockerfile

Para diseñar imágenes y distribuirlas

Dockerfile

- Es un fichero de configuración para automatizar la creación de imágenes de Docker
- Actúa como un script de instalación con una sintaxis propia
- Se debe llamar exactamente **Dockerfile**

FROM: Indica una imagen sobre la que trabajar

FROM <image>:<tag>

RUN: Ejecuta un comando para configurar

RUN <command>

Cada comando **RUN** realiza un commit

CMD: Comando para arrancar el contenedor

CMD <command> <args>

o

CMD [<command>, <args>]

Ejecuta el comando cuando se arranca el contenedor.
Puede ser sobrescrito al realizar la llamada.

Ejemplo

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get upgrade
RUN apt-get install -y git
CMD git init --bare
```

Construir la imagen usando el Dockerfile

```
docker build -t <repo>:<tag> <path>
```

Crea una imagen con nombre **<repo>:<tag>** y adjunta los archivos que se encuentran en **<path>** a la imagen

Gestión de contenedores

Ver contenedores

docker ps

Muestra los contenedores en ejecución

docker ps -a

Muestra todos los contenedores (incluso los parados)

Parar un contenedor en background

```
docker stop <container-ID>
```

Volver a arrancar un contenedor

```
docker start <container-ID>
```

Iniciar un proceso en contenedor en ejecución

docker exec <container-ID> <command>

docker exec -ti ubuntu /bin/bash

Inicia un terminal en un contenedor en ejecución

Eliminar un contenedor

```
docker rm <container-ID>
```

Gestión de imágenes ycontenedores

Listar las imágenes disponibles

docker images

Crear una imagen

- Arrancando una imagen de base y hacer `docker commit` tras modificarla
- Construir una imagen utilizando un `Dockerfile` y `docker build`

Eliminar una imagen local

```
docker rmi <image>
```

Renombrar una imagen

docker tag <image>:<tag> <image>:<tag>

Trabajando con Docker Hub

Subir una imagen a Docker Hub

docker push <repo>:<tag>

Descargar una imagen a Docker Hub

docker pull <repo>/<image>:<tag>

Volúmenes

Volúmenes

- Los volúmenes son directorios donde se persisten los datos independientemente del ciclo de vida de un contenedor.
- Puede usarse para mapear una carpeta del host a una carpeta del contenedor.
- Se pueden compartir entre varios contenedores
- Cuando se borra un contenedor, no se elimina el volumen.

Ver contenedores ejecutados (parados)

docker run -v <container-path> <image>

docker run -v <local-path>:<container-path> <image>

Debemos usar rutas absolutas

Volúmenes en Dockerfile

```
VOLUME /www/docker.rocks

VOLUME [“/www/docker.rocks”, “/data”]
```

Pero en Dockerfile no podemos marearlos a directorios locales

¿Cuándo usarlos?

- Separar datos de archivos de aplicación
- Compartir datos entre contenedores
- No es recomendable mapear volúmenes del host en entornos de producción (se pierde portabilidad)

Networking

Networking

- Los contenedores tienen su propia red y dirección IP
- Podemos mapear puertos del contenedor al host

Mapear puertos

docker run -p <host-port>:<container-port> <image>

docker run -P <image>

Y con -P, ¿cómo sabe qué puertos mapear?

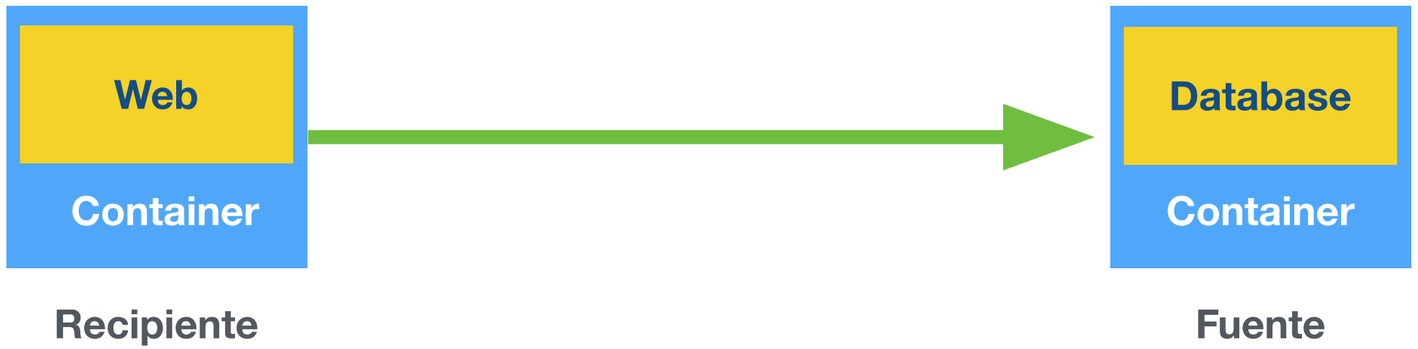
Mapea los puertos que se definen (o exponen) en el Dockerfile que crea la imagen con:

EXPOSE <port-id>

Conexión de contenedores

- La conexión o linking de contenedores permite transferir datos entre ellos sin necesidad de exponer puertos hacia afuera.

- Se establece una política de fuente y recipiente para indicar quién puede acceder a quién.



Crear un enlace

1. Crear el contenedor fuente (con un nombre descriptivo)

3. Crear el contendor recipiente y enlazarlo con **--link**

```
docker run -d --name database mysql

docker run -d -P --name web --link database:db nginx
```

Es un alias, se usa para referenciar la conexión dentro del contenedor recipiente en /etc/hosts

Docker Compose

Docker compose

- Permite crear y gestionar aplicaciones multicontenedor
- Agiliza el trabajo del enlace de los contenedores
- Utiliza un archivo de configuración en formato YAML llamado docker-compose.yml

Configurando Docker compose

```
nodeclient: # a service call “nodeclient”
        build: . # path to Docker file
        command: java Hello. # run java Hello
        link: . # links with
         - database # source container
database: # a service call “database”
        image: redis # use the latest redis image
```

Ejecutar docker-compose

docker-compose up

- Debe estar en el mismo directorio que docker-compose.yml
- Creará las imágenes y arrancará los contenedores

Registros privados

Registros privados

- Nos permiten almacenar nuestras imágenes en nuestros propios servidores en lugar de Docker Hub. 😀
- Para crear nuestro propio registro: tenemos que usar Docker! 😀
- Pero no tiene interfaz web! 😓

```
docker run -d -p 5000:5000 registry:2.0
```

Subir una imagen a Docker Hub

docker tag <image-id> <host:port>/<repo>:<tag>

1) Renombrar la imagen con el host y puerto de nuestro server

docker push <host:port>/<repo>:<tag>

Hacer push a nuestro servidor

Descargar una imagen a Docker Hub

docker pull <host:port>/<image>:<tag>