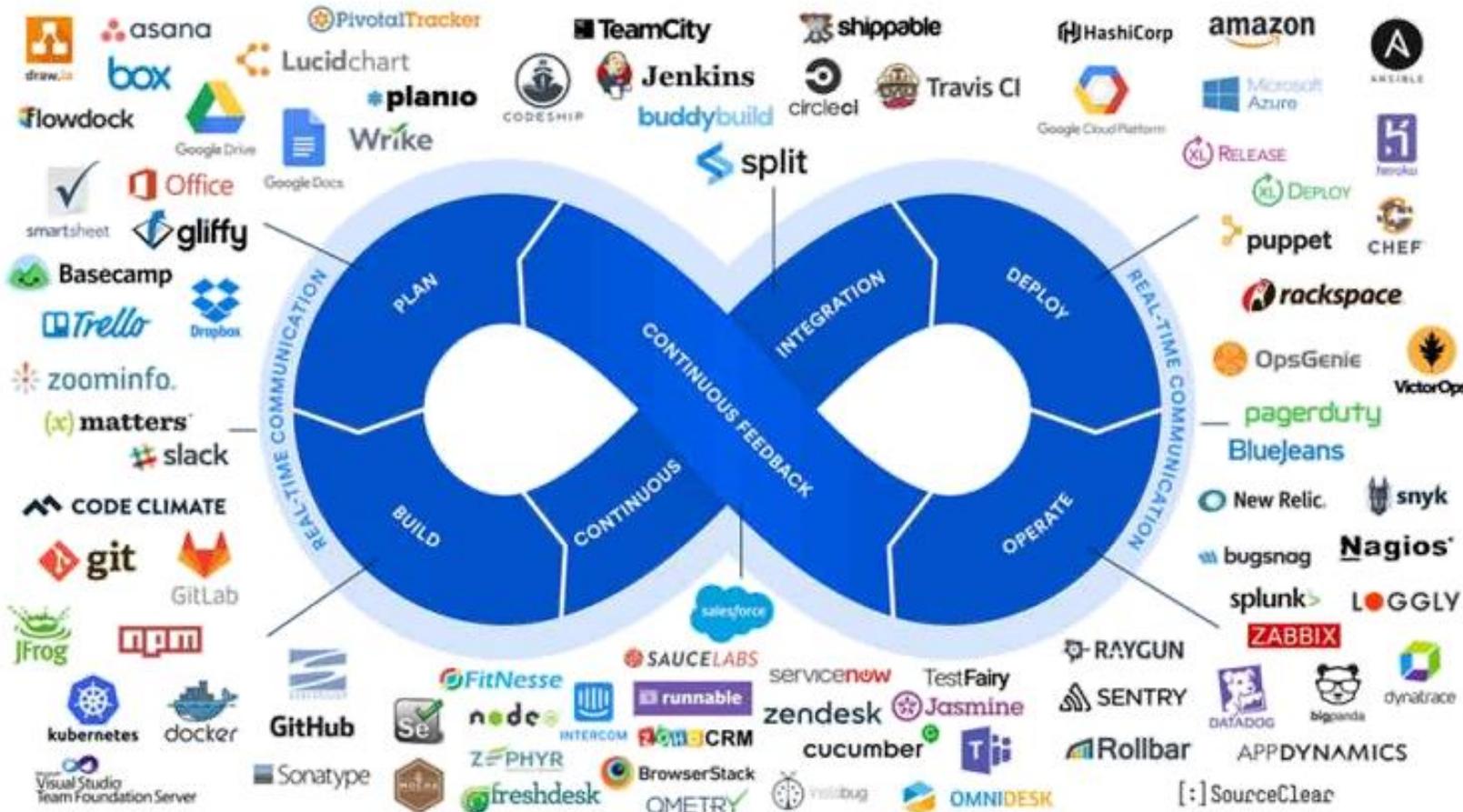


# Docker práctico para entornos DevOps



# Docker

- Introducción
- Instalación de Docker
- Administración básica de contenedores
- Redes en Docker
- Almacenamiento en Docker
- Docker HUB
- Docker File
- Docker Compose

# ¿Qué es Linux Professional Institute?

- LPI Inc. se constituye formalmente como una organización sin ánimo de lucro en Octubre de 1999, con su sede cerca de Toronto, Canadá.
- A LPI se le reconoce en todo el mundo como la primera organización en impulsar y apoyar el uso de Linux, Código Abierto y SW Libre.

# ¿Qué es Linux Professional Institut

LPI se dedica a promover y certificar capacidades esenciales de profesionales y estudiantes en el sistema operativo Linux a través de la creación de exámenes altamente comprensibles e independientes de cualquier distribución.

[www.lpi.org](http://www.lpi.org)

# Certificaciones LPI

- LPI ha diseñado el programa de certificación LPIC (Linux Professional Institute Certification), avalado y reconocido por organizaciones y empresas de todo el mundo.
- El objetivo de las certificaciones LPIC es acreditar la capacitación de los profesionales de las TI en el uso del Sistema Operativo Linux y sus herramientas asociadas.
- Son independientes de cualquier distribución y/o fabricante.

# Certificaciones LPI

## Valor añadido de las certificaciones oficiales LPI:

1. Avance profesional, los individuos que obtienen una certificación LPI logran un mayor reconocimiento y credibilidad.
2. Mayor confianza, al haber acreditado los conocimientos y habilidades necesarias para realizar el trabajo de manera eficiente y efectiva.
3. Diferenciación, en un mercado laboral cada vez más competitivo.

# Certificaciones LPI



**Linux Essentials**



**LPIC 1**



**LPIC 2**



**LPIC 3**

<b>Exámenes</b>	<b>LPI LE</b>	<b>LPI 101</b> <b>LPI 102</b>	<b>LPI 201</b> <b>LPI 202</b>	<b>LPI 300</b> <b>LPI 303</b> <b>LPI 304</b>
<b>Aptitudes</b>	<b>Instalación Escritorio</b>	<b>Administración Servidores Pequeña/Media Empresa</b>	<b>Administración Servidores Mediana/Gran Empresa</b>	<b>Administración Servidores Gran Empresa</b>
<b>Nivel</b>	<b>Introductorio</b>	<b>Junior</b>	<b>Avanzado</b>	<b>Sénior</b>

# Certificaciones LPI



## LPIC 1 (*Junior Level Linux Professional*)

- Ser capaz de trabajar en la línea de comandos
- Realizar tareas sencillas de mantenimiento como ayudar a usuarios, mantenimiento de usuarios, realización de copias de seguridad y restauraciones, paradas y arranque de sistemas.
- Instalación y configuración de un equipo de trabajo y su conectividad a la red.
- Al obtener LPIC1, se habilita:
  - CompTIA Linux +
  - SUSE Certified Linux Administrator



# Certificaciones LPI

## LPIC 2 (*Advanced Level Linux Professional*)

Planificación, mantenimiento, securización de una red mixta (MS, Linux) incluyendo:

- Servidor de red Samba
- Pasarela de Internet: firewall, proxy, correo, news
- Servidores de Internet: Servidores web, FTP
- Supervisión de asistentes
- Aconsejar en automatización y compras



# Certificaciones LPI

## **LPIC 3 (*Senior Level Linux Professional*)**



3 Exámenes Disponibles

**Exam 300**  
*Entorno mixto*

**Exam 303**  
*Seguridad*

**Exam 304**  
*Virtualización y Alta  
Disponibilidad*

# Certificaciones LPI

## Linux Professional Institute DevOps Tools Engineer



<http://www.lpi.org/our-certifications/exam-701-objectives>

Software Development  
Configuration Management  
Log Management  
Virtual Machines  
Container  
scm  
Cloud  
Immutable Servers  
Cloud Services  
Continuous Delivery  
12 Factor App

System Images

# DevOps

**DevOps** es un conjunto de prácticas que automatizan los procesos entre los equipos de desarrollo de software y TI para que puedan compilar, probar y publicar software con mayor rapidez y fiabilidad. El concepto de DevOps se basa en establecer una cultura de colaboración entre equipos que, tradicionalmente, trabajaban en grupos aislados. Entre sus ventajas se incluyen el aumento de la confianza y de la velocidad de publicación de software, la capacidad de solucionar incidencias críticas rápidamente y una mejor gestión del trabajo imprevisto.

## Una revolución en marcha

El software es ahora la pieza clave de todo negocio. Y cuando cualquier empresa es una empresa de software, tus aplicaciones son las que te diferencian de la competencia.

La percepción de lo que es una buena aplicación cambia con el tiempo, especialmente a medida que los mercados evolucionan y los competidores redefinen su estrategia.

Por lo tanto, cuanto más rápido puedas actualizar tus aplicaciones, mejor. Gracias al cloud, la disponibilidad y la escalabilidad de la infraestructura que las soporta ya no es un problema.

Hoy, el modo en que tu equipo técnico colabora con todas las partes implicadas en el desarrollo, diseño, prueba y lanzamiento es completamente diferente.

# DevOps

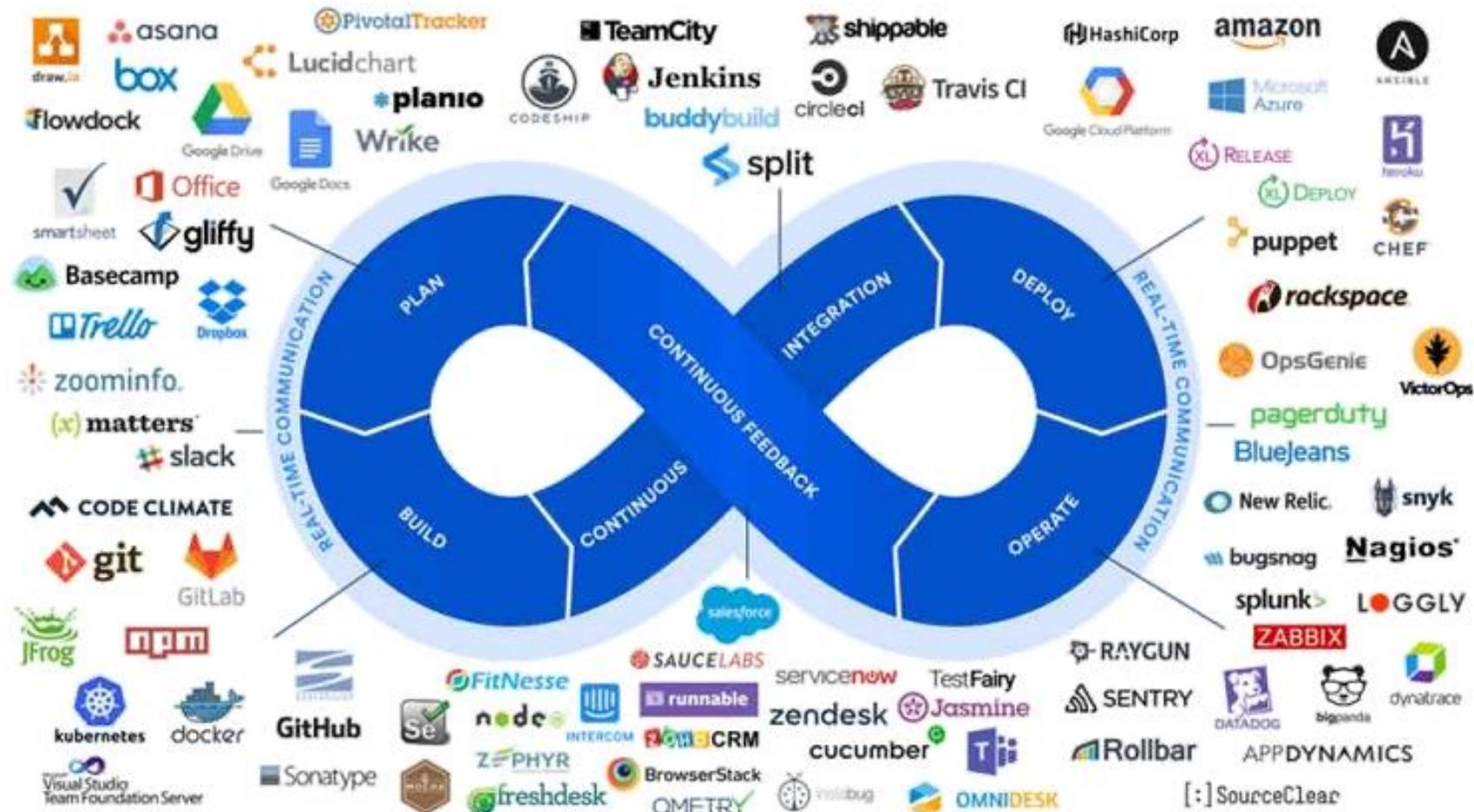
## El papel de DevOps

La finalidad de DevOps es crear las condiciones adecuadas para la colaboración entre los departamentos de desarrollo y de operaciones. Sin embargo, antes de implementar una estrategia DevOps hay que tener varias cosas en cuenta.

**¿Cuentas con la infraestructura adecuada para soportar este enfoque?  
¿Y las herramientas? ¿Sabes por dónde empezar?**



# DevOps

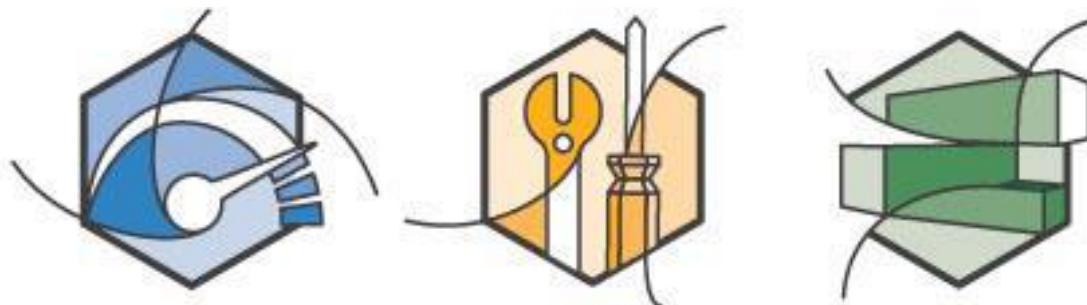


# Introducción

- Docker es un proyecto open source que ha revolucionado la manera de desarrollar software gracias a la sencillez con la que permite gestionar contenedores. Los contenedores LXC (LinuX Containers) son un concepto relativamente antiguo y utilizado desde hace tiempo por grandes empresas como Amazon o Google, pero cuya gestión era complicada. Sin embargo, Docker define APIs y herramientas de línea de comandos que hacen casi trivial la creación, distribución y ejecución de contenedores. De ahí que el lema de Docker sea: “Build, Ship and Run. Any application, Anywhere” (**Construye, envía y ejecuta. Cualquier aplicación, en cualquier lugar**) y se haya convertido en una herramienta fundamental tanto para desarrolladores como para administradores de sistemas.

# Introducción

- Podríamos definir un contenedor Docker como una máquina virtual ligera, que corre sobre un sistema operativo Linux pero con su propio sistema de ficheros, su propio espacio de usuarios y procesos, sus propias interfaces de red... por lo que se dice que son sistemas aislados. Las características principales de Docker son su portabilidad, su inmutabilidad y su ligereza:



# Introducción

- **Portabilidad**

Un contenedor Docker es ejecutado por lo que se denomina el Docker Engine, un demonio que es fácilmente instalable en prácticamente todas las distribuciones Linux. Un contenedor ejecuta una imagen de docker, que es una representación del sistema de ficheros y otros metadatos que el contenedor va a utilizar para su ejecución. Una vez que hemos generado una imagen de Docker, ya sea en nuestro ordenador o vía una herramienta externa, esta imagen podrá ser ejecutada por cualquier Docker Engine, independientemente del sistema operativo y la infraestructura que haya por debajo.

# Introducción

- **Inmutabilidad**

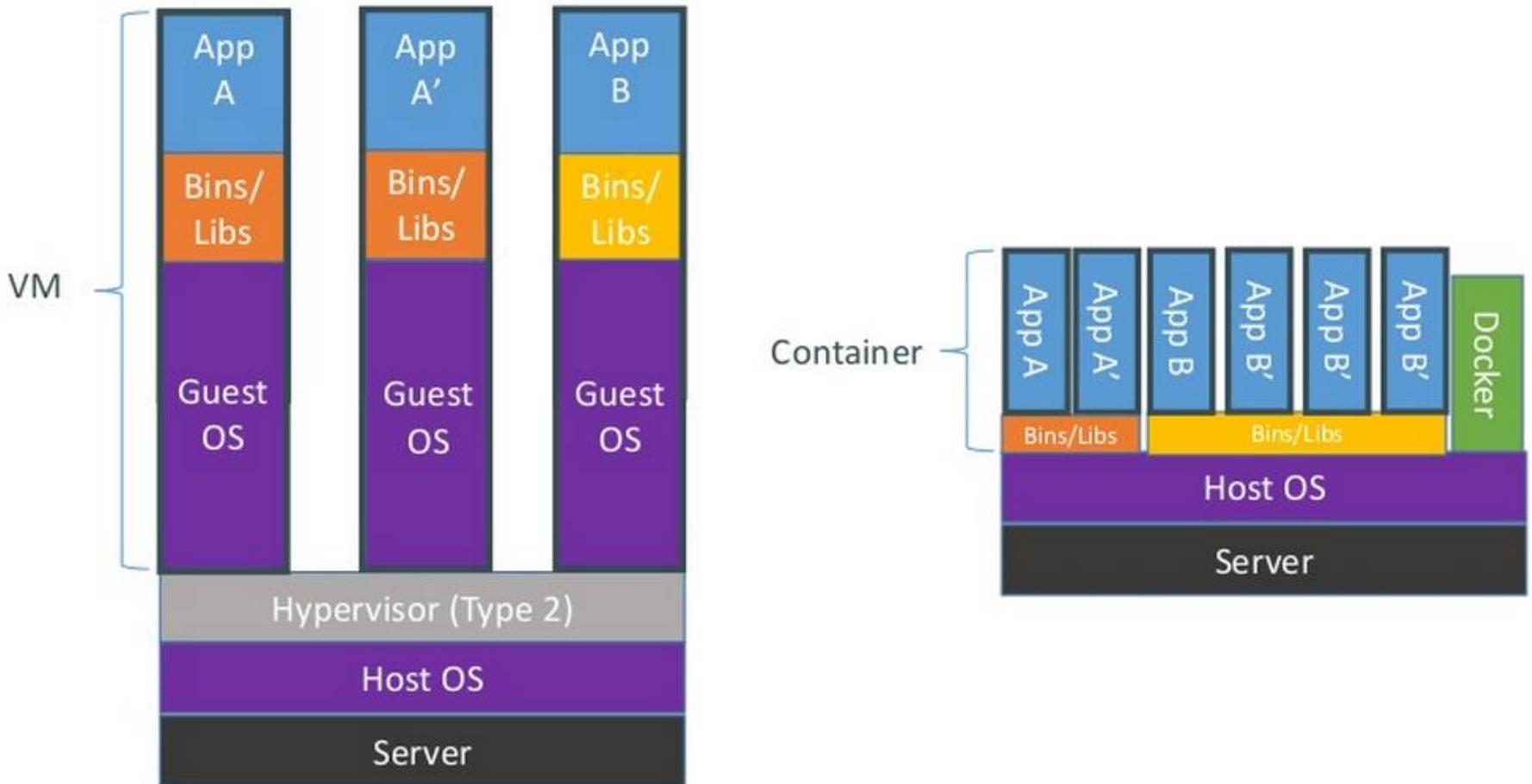
Una aplicación la componen tanto el código fuente como las librerías del sistema operativo y del lenguaje de programación necesarias para la ejecución de dicho código. Estas dependencias dependen a su vez del sistema operativo donde nuestro código va a ser ejecutado, y por esto mismo ocurre muchas veces aquello de que “no sé, en mi máquina funciona”. Sin embargo, el proceso de instalación de dependencias en Docker no depende del sistema operativo, si no que este proceso se realiza cuando se genera una imagen de docker. Es decir, una imagen de docker (también llamada repositorio por su parecido con los repositorios de git) contiene tanto el código de la aplicación como las dependencias que necesita para su ejecución. Una imagen se genera una vez y puede ser ejecutada las veces que sean necesarias, y siempre ejecutará con las misma versión del código fuente y sus dependencias, por lo que se dice que es inmutable. Si unimos inmutabilidad con el hecho de que Docker es portable, decimos que Docker es una herramienta fiable, ya que una vez generada una imagen, ésta se comporta de la misma manera independientemente del sistema operativo y de la infraestructura donde se esté ejecutando.

# Introducción

- **Ligereza**

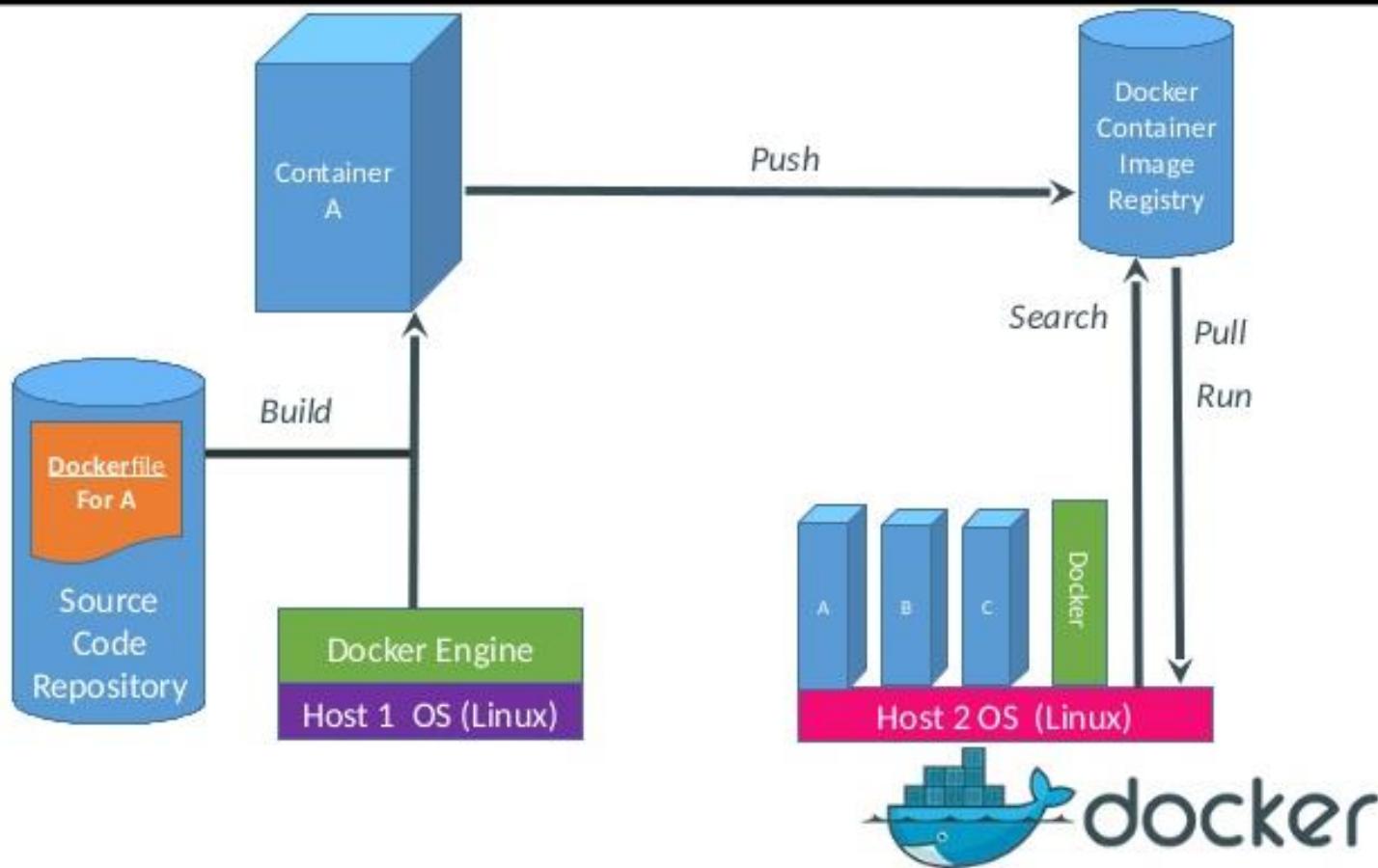
Los contenedores Docker corriendo en la misma máquina comparten entre ellos el sistema operativo, pero cada contenedor es un proceso independiente con su propio sistema de ficheros y su propio espacio de procesos y usuarios (para este fin Docker utiliza cgroups y namespaces, recursos de aislamiento basados en el kernel de Linux). Esto hace que la ejecución de contenedores sea mucho más ligera que otros mecanismos de virtualización. Comparemos por ejemplo con otra tecnología muy utilizada como es Virtualbox. Virtualbox permite del orden de 4 ó 5 máquinas virtuales en un ordenador convencional, mientras que en el mismo ordenador podremos correr cientos de containers sin mayor problema, además de que su gestión es mucho más sencilla.

# Introducción



# Introducción

## Componentes básicos de Docker



# Introducción

## Principales diferencias

Máquinas Virtuales	Contenedores
Más pesadas	Más ligeras
Varios procesos	Un único proceso
Conexión por ssh (aunque esté en local)	Acceso directo al contenedor
Más seguridad porque están más aisladas del host	Potencialmente menor seguridad porque se ejecutan como procesos en el host

# Instalación de Docker

## **DOCKER COMMUNITY EDITION VS DOCKER ENTERPRISE EDITION**

De manera casi paralela con el cuarto aniversario de Docker, en marzo de 2017, se anunció la división de Docker en dos plataformas:

- Docker Engine pasaría a llamarse Docker Community Edition (CE)
- Docker Datacenter / Docker Engine CS (Commercial Supported) pasaría a llamarse Docker Enterprise Edition (EE), que consta de tres niveles de soporte.

A su vez, adopta un cambio en versionado de sus productos, que era simplemente numeral y ahora pasa a basarse en el año y mes de lanzamiento de manera similar a como lo hace Canonical en Ubuntu. De esta manera, se pasó de la versión v1.13 de Docker a la versión v17.03 correspondiente al mes de marzo de 2017

# Instalación de Docker

## DOCKER COMMUNITY EDITION VS DOCKER ENTERPRISE EDITION

**La versión CE tiene dos variantes:**

- **Edge:** Liberación mensual y en la que solo se proporciona soporte a bugs y fallos de seguridad durante el mes en curso.
- **Stable:** De liberación trimestral con cuatro meses de soporte. Docker EE también asume el mismo ritmo de liberación que la versión Stable de Docker CE, pero el soporte se garantiza durante un año.

# Instalación de Docker

## En Ubuntu/Debian:

- Crear el fichero /etc/apt/sources.list.d/docker.list
- Contenido:

deb https://apt.dockerproject.org/repo distro main

## • Ejecutar:

apt-get update

apt-get install docker-engine

## • Comprobar:

systemctl status docker

docker info

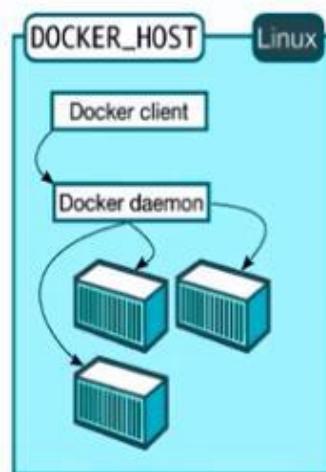
docker versión

```
root@debian:/etc/apt/sources.list.d# systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled)
  Active: active (running) since Sat 2016-02-20 19:55:07 CET; 14s ago
    Docs: https://docs.docker.com
   Main PID: 27148 (docker)
     CGroup: /system.slice/docker.service
             └─27148 /usr/bin/docker daemon -H fd://

Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.527340735+01:00" level=warni...und"
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.528168171+01:00" level=warni...iod"
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.528380217+01:00" level=warni...tas"
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.533598886+01:00" level=info ...rt."
Feb 20 19:55:07 debian docker[27148]: .....
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.567016564+01:00" level=info ...ne."
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.567116348+01:00" level=info ...ion"
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.567491097+01:00" level=info ...10.1
Feb 20 19:55:07 debian systemd[1]: Started Docker Application Container Engine.
Feb 20 19:55:07 debian docker[27148]: time="2016-02-20T19:55:07.679717000+01:00" level=info ...ock"
Hint: Some lines were ellipsized, use -l to show in full.
root@debian:/etc/apt/sources.list.d#
```

# Instalación de Docker

## Instalación centos/RHEL7/fedora:



• En centos / RHEL7 / Fedora:

→ Crear el fichero /etc/yum.repos.d/docker.repo

→ Con el contenido

→ Ejecutar:

• centos / RHEL7:

yum install docker-engine

• Fedora:

dnf install docker-engine

→ Comprobar:

systemctl status docker-engine // /etc/init.d/docker status

docker info

docker version

[docker repo]  
name = Docker Repository  
baseurl = https://yum.dockerproject.org/repo/main/

enabled = 1

gpgcheck = 1

gpgkey = https://yum.dockerproject.org/yum

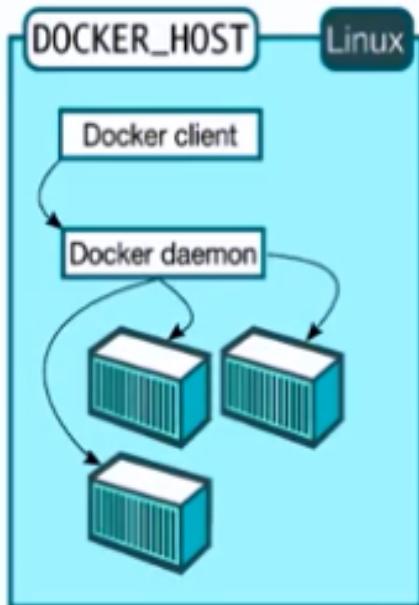
CentOS: centos/greasever

RHEL7: centos/7

Fedora: fedora/greasever

# Instalación de Docker

## Otras distribuciones Linux:



- Arch Linux
- CRUX Linux
- FrugalWare
- Gentoo
- Oracle Linux
- openSUSE / SUSE Linux Enterprise

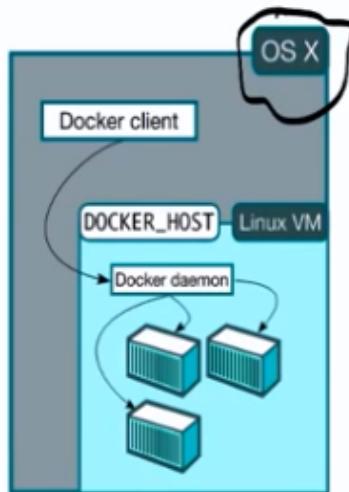
· INSTRUCCIONES:

<https://docs.docker.com/engine/installation/linux/>

# Instalación de Docker

## Instalación en Mac/OS X/ Windows:

### • Instalación en Mac OS X / Windows



→ En Windows comprobar si la virtualización  
está activada.

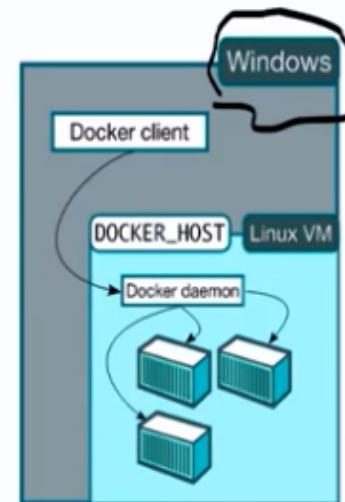
→ Instalar Docker toolbox

→ Ejecutar Docker Quickstart Terminal

→ Comprobar:

`docker info`

`docker version`



# Docker

- La empresa Docker Inc. ofrece muchos **servicios** y **herramientas** relacionados con Docker



**Docker Datacenter**

An on-premises / VPC CaaS platform that integrates with enterprise infrastructure to build, ship and run distributed applications anywhere.

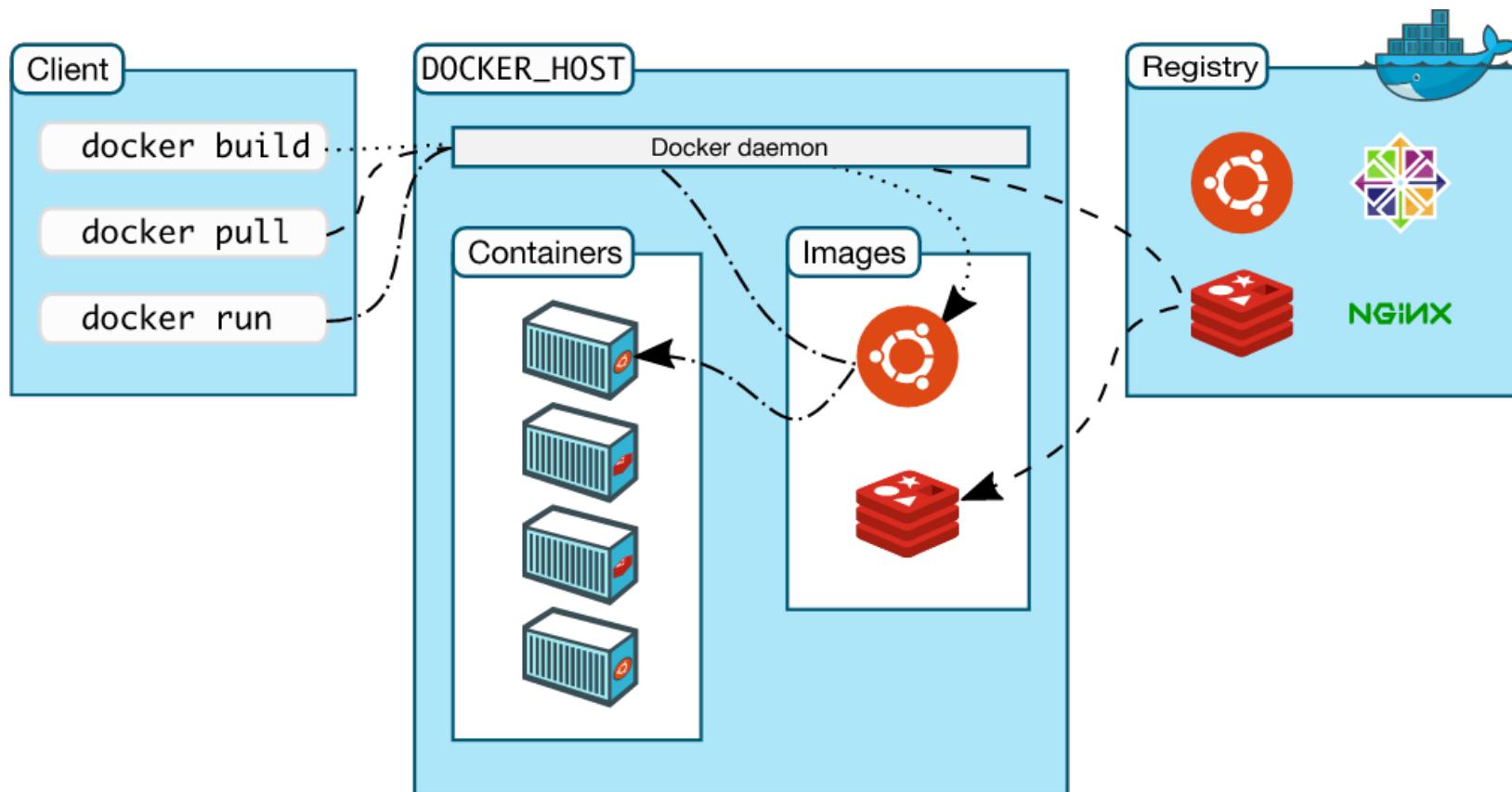


**Docker Cloud**

A SaaS CaaS service to build, ship and run distributed applications anywhere

# Docker concepts

## •Conceptos básicos docker



# Docker concepts

## Docker Engine / Docker CE

Es el demonio de Docker que se ejecuta en el sistema operativo. El usuario nunca interactúa sobre él directamente, si no que lo hace a través de Docker Client, que es quien intermedia entre ambos.

Expone una API para la gestión de imágenes y contenedores, lleva a cabo la creación y/o ejecución de imágenes, publica o descarga desde un Docker Registry, etc.

Además, es el componente encargado de la gestión de contenedores que se encuentran en ejecución.

## Docker Client

Podría llamarse cliente Docker a cualquier herramienta que haga uso de la API de Docker Engine, pero normalmente se hace referencia al comando docker que interactúa con Docker Engine.

Este cliente permite ser configurado para mandar las interacciones con un Docker Engine local o remoto, siendo posible gestionar un entorno de desarrollo local o servidores de producción.

Dispone de una gran cantidad de comandos que permiten gestionar los contenedores, las imágenes, los volúmenes y las redes: docker info, docker build, docker run, docker ps, docker logs... son comandos que forman parte de Docker Client.

# Docker concepts

## Docker Compose

Docker Compose permite desplegar y administrar aplicaciones compuestas por varios contenedores relacionados entre sí. Esto se hace definiendo en un único archivo los contenedores que forman parte de una aplicación, permitiendo desplegarla utilizando la arquitectura de despliegue mediante microservicios.

## Docker Machine

Este componente ofrece la posibilidad de crear y configurar máquinas virtuales o nodos físicos para alojar dentro de ellas contenedores. Hace posible construir máquinas virtuales en las plataformas de IaaS (Infraestructura como Servicio) más populares como AWS, GCE, Azure, DigitalOcean, OpenStack... o bien en plataformas de virtualización de escritorio como VMware Fusion o VirtualBox.

A la hora de crear un nodo instala boot2docker en él, una distribución muy liviana del sistema operativo Linux (Tiny Core Linux) que incluye Docker Engine en su interior. Su tamaño es de ~27MB y además es el encargado de crear las claves SSH, iniciar el nodo, gestionar la red, crear y copiar los certificados TLS... entre otras tareas que realiza por cada nodo.

Así pues, Docker Machine permite utilizar nuestro Docker Client para administrar otros Docker Engine remotos que hayan sido configurados con esta herramienta.

# Docker concepts

## Docker Swarm

Si has escuchado hablar sobre Kubernetes de Google o Mesos de Apache, Docker Swarm es la tecnología nativa que cumple prácticamente con la misma función. Se trata de una herramienta de orquestación de contenedores.

De más reciente creación que Kubernetes, este se trata de un componente que permite realizar un clúster de nodos al que se pueden enviar contenedores Docker para su ejecución. En ocasiones se utiliza conjuntamente con Docker Machine para crear los nodos a utilizar y se combina con Docker Compose para desplegar los contenedores.

# Docker concepts

## Docker Hub / Docker Store

Docker Hub es un repositorio de imágenes Docker. Esto permite compartir las imágenes construidas por los usuarios, así como las empresas, lo que facilita el lanzamiento de aplicaciones evitando tener que construir la tuya propia mediante un Dockerfile.

Permite crear repositorios públicos, a los que puede acceder cualquier usuario, o repositorios privados donde alojar de manera privada las imágenes de Docker. Los repositorios públicos son ilimitados, mientras que los repositorios privados son de pago, siendo el primer repositorio privado gratuito.



# Docker concepts

## Docker Registry

El Docker Registry es un repositorio de imágenes Docker privado y seguro. Es un Docker Hub que forma parte de nuestra arquitectura como un componente y que normalmente no suele estar en el mismo equipo en el que se está trabajando, si no en un servidor. De esta manera podemos almacenar las imágenes de los contenedores de forma privada.

Puede suponer ventajas para una entidad tanto en privacidad como en seguridad, además de en velocidad a la hora de descargar estos registros o ahorro de costes conforme al precio de repositorios privados.



# Docker concepts

## Dockerfile

Dockerfile es un archivo que reconoce Docker y que, a través de una serie de instrucciones, es capaz de automatizar la creación de un contenedor Docker. En este archivo se agrega todo lo que necesitamos en nuestro contenedor, siendo este archivo el que sufra las modificaciones y no el contenedor en sí.

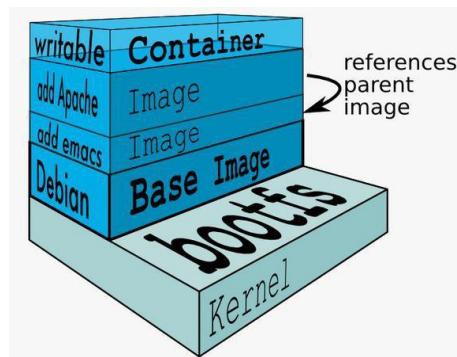
Podemos decir que se trata de un “script” o una “receta” que permite automatizar la creación del contenedor. Igualmente sirve para crear imágenes de Docker, no siendo necesario el archivo para poder hacer uso de estas imágenes y crear contenedores con ellas.

# Docker concepts

## Imagen de Docker

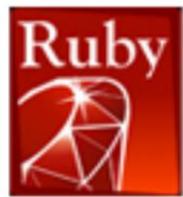
Una imagen de Docker es, por así decirlo, un Dockerfile “compilado”. Son utilizadas para crear contenedores y pueden ser compartidas, almacenadas y actualizadas a través Docker Hub o Docker Registry. De esta modo podríamos descargar una imagen preparada para hacer funcionar un servicio en concreto.

Estas imágenes están construidas a través de capas, por lo que su formato de capas construye paso a paso la imagen siguiendo instrucciones: actualizar repositorios y paquetes, ejecutar una instalación, reemplazar un fichero, abrir un puerto...



# Docker concepts

- Popular repositories in Docker Hub



redis

NGINX

node



# First steps with docker

- **Hands on...**

- Official beginners tutorial

<https://github.com/docker/labs/tree/master/beginner>

- Install docker

<https://docs.docker.com/engine/installation/>

# Docker documentation

- **Official documentation**

- <https://docs.docker.com/>

- **Advanced tutorials**

- <https://docs.docker.com/engine/tutorials/dockerizing/>
  - <https://docs.docker.com/engine/tutorials/usingdocker/>
  - <https://docs.docker.com/engine/tutorials/dockerimages/>
  - <https://docs.docker.com/engine/tutorials/networkingcontainers/>
  - <https://docs.docker.com/engine/tutorials/dockervolumes/>
  - <https://docs.docker.com/engine/tutorials/dockerrepos/>

- **Cheat Sheet**

- <https://github.com/wsargent/docker-cheat-sheet/blob/master/README.md>

# Administración básica de contenedores

- Testing if docker is correctly installed

```
$ docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
```

```
03f4658f8b78: Pull complete
```

```
a3ed95caeb02: Pull complete
```

```
Digest:
```

```
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc
```

```
72074cc1ca36966a7
```

```
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker.

This message shows that your installation appears to be working correctly.

...

# Primeros pasos con docker

- **Running your first container**

```
$ docker run alpine ls -l
total 48
drwxr-xr-x    2 root      root            4096 Mar  2 16:20 bin drwxr-xr-x    5 root
root          360 Mar 18 09:47 dev drwxr-xr-x   13 root      root            4096
Mar 18 09:47 etc drwxr-xr-x    2 root      root            4096 Mar  2 16:20
home drwxr-xr-x    5 root      root            4096 Mar  2 16:20 lib
.....
```

# Primeros pasos con docker

- Running your first container

```
$ docker run alpine ls -l
```



**Command “run”**

Creates a new  
container and start it

# Primeros pasos con docker

- Running your first container

```
$ docker run alpine ls -l
```

Image name

alpine is a minimal linux system (4.8Mb). The image is downloaded if not stored in local machine

# Primeros pasos con docker

- **Running your first container**

```
$ docker run alpine ls -l
```

**Command “ls -l”**

This command will be  
executed inside the  
running container

# Primeros pasos con docker

- **Inspecting the downloaded images**

```
$ docker images
```

REPOSITORY	CREATED	TAG	VIRTUAL SIZE	IMAGE ID
alpine	4 weeks ago	latest	1.109 MB	c51f86c28340
hello-world	5 months ago	latest	960 B	690ed74de00f

List all images stored in the system

# Primeros pasos con docker

- **Executing a container**

```
$ docker run alpine echo "hello from alpine"  
hello from alpine
```

Execute the command “echo” inside the container

# Primeros pasos con docker

## • Inspecting containers

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	PORTS
CREATED	STATUS		
NAMES			
a6a9d46d0b2f	alpine	"echo 'hello from alp"	6 minutes ago
Exited (0) 6 minutes ago		lonely_kilby	
ff0a5c3750b9	alpine	"ls -l"	8 minutes ago
minutes ago		elated_ramanujan c317d0a9e3d2	Exited (0) 8
"/hello"	34		hello-world
seconds ago	Exited (0) 12 minutes ago		
stupefied_mcclintock			

It shows containers in the system.  
All of them has STATUS Existed. These  
containers are not currently executing

# Primeros pasos con docker

- Interactive commands in containers

```
$ docker run -it alpine /bin/sh
/ # ls
bin      dev      etc      home      lib      linuxrc  media    mnt      proc root  run      sbin
sys      tmp      usr      var
/ # uname -a
Linux 97916e8cb5dc 4.4.27-moby #1 SMP Wed Oct 26 14:01:48 UTC 2016 x86_64 Linux
/ # exit
$
```

To execute an interactive command it is necessary to use the option “-it” to connect the console to the container command

# Primeros pasos con docker

- Managing containers lifecycle

```
$ docker run -d seqvence/static-site
```

Option “-d”

Executes the container  
in background



# Primeros pasos con docker

## • Managing containers lifecycle

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	NAMES
a7a0e504ca3e	seqvence/static-site	/bin/sh -c 'cd /usr/'	
28 seconds ago	Up 26 seconds	PORTS	

CREATED

a7a0e504ca3e

28 seconds ago

IMAGE  
STATUS

seqvence/static-site  
Up 26 seconds

COMMAND  
PORTS

"/bin/sh -c 'cd /usr/'

NAMES

Container id is  
**a7a0e504ca3e**

This id is used to refer  
to this container

STATUS is UP



# Primeros pasos con docker

- **Managing containers lifecycle**

- Stop running container

```
$ docker stop a7a0e504ca3e
```

- Delete files of the container

```
$ docker rm      a7a0e504ca3e
```

# Primeros pasos con docker

## • Resumen Iniciar contenedor y ejecución

- Iniciar container

```
$ docker run -dti --name centos6 b45f1e1c24ef
```

- Conectarnos a un container

```
$ docker exec -ti centos6 /bin/bash
```

# Primeros pasos con docker

- Start container with public port

```
docker run --name static-site \
-e AUTHOR="Your Name" -d \
-p 9000:80 seqvence/static-site
```

# Primeros pasos con docker

- Start container with public port

```
docker run --name static-site \
-e AUTHOR="Your Name" -d \
-p 9000:80 seqvence/static-site
```

**--name static-site**

Specifies the unique name of  
the container

# Primeros pasos con docker

- Start container with public port

```
docker run --name static-site \  
-e AUTHOR="Your Name" -d \  
-p 9000:80 seqvence/static-site
```

**-e AUTHOR="Your Name"**

Set the environment variable  
AUTHOR to value "Your Name"

# Primeros pasos con docker

- Start container with public port

```
docker run --name static-site \  
-e AUTHOR="Your Name" -d \  
-p 9000:80 seqvence/static-site
```

-d

Execute container as deamon

# Primeros pasos con docker

- Start container with public port

```
docker run -dtip 8080:80 --name intranet agarciacf/intranet
```



**-p 8080:80**

Connects the local port 8080 to  
the port 80 in the container

# Primeros pasos con docker

## • Use the service

Open <http://127.0.0.1:8080>

-  **joomla! spanish**

**MENÚ PRINCIPAL**

- Inicio
- Visión general
- Licencia
- Más sobre Joomla!
- FAQ
- Noticias
- Enlaces
- Servidor de noticias

**RECURSOS**

- Joomla!
- Foros Joomla!
- Ayuda Joomla!
- OSM
- Administración
- Joomla! Spanish

**CONCEPTOS CLAVE**

**Bienvenidos a la portada**

Bienvenido a Joomla!

Jueves, 12 de Octubre de 2006 10:00 administrador

Powered by **Joomla!** version 1.5 Joomla! proporciona una interfaz fácil de usar que simplifica la administración y publicación de grandes volúmenes de contenido incluyendo HTML, documentos, y multimedia. Joomla! es usado por organizaciones de todas las clases para sitios web públicos, intranets, extranets y con soporte de una comunidad con miles de usuarios.

Última actualización el Jueves, 08 de Noviembre de 2007 12:06

[LEER MÁS...](#)

**Somos voluntarios**

Miércoles, 07 de Julio de 2004 09:54 administrador

El equipo de Joomla! consiste en un grupo de desarrolladores voluntarios, así como diseñadores, administradores y gestores que, siendo dedicados miembros de la comunidad, juntos y con una larga andadura en el grupo de trabajo, han llevado a Joomla! a nuevas alturas dentro de su relativamente breve período de

**Sobre el inicio del código!**

Miércoles, 07 de Julio de 2004 12:00 administrador

Para asegurar esta liberación de código, Wilco Jansen solo tuvo que enviarle ron a Johan Janssens (el cual siente especial inclinación por esta bebida desde que vio la película "Los piratas del caribe").. el cual picó el anchoa mezclándolo con ron ron. Y aníjate estamos con

**ENCUESTAS**

¿Para qué usas Joomla? :

- Comunidades-Grupos
- Sitios públicos
- Comercio electrónico
- Blogs
- Intranets
- Fotos y sitios multimedia
- Para todo lo anterior!

**PATROCINADORES**

[Web Empresa Hosting Joomla!](#)  
Servicios Profesionales para Joomla:  
Hosting, Servicio Técnico, Formación y  
JoomlaFácil.

[CompluSoft](#)  
Empresa de informática situada en la  
zona este de la Comunidad de Madrid  
presta sus servicios en las más

# Primeros pasos con docker

- **Use the service**

- If you are using **Docker for Mac** or **Docker for Windows**

```
$ docker-machine ip default  
192.168.99.100
```

- Then you have to open  
<http://192.168.99.100:9000/> in the browser

# Primeros pasos con docker

## • Container management

- Stop and remove the container

```
$ docker stop static-site  
$ docker rm static-site
```

- Stop and remove a running container

```
$ docker rm -f static-site
```

- Remove all containers

```
$ docker rm -f $(docker ps -a -q)
```

# Primeros pasos con docker

## • Container management

- Parar todos los contenedores

```
$ docker stop $(docker ps -a -q)
```

- Remove all containers

```
$ docker rm $(docker ps -a -q)
```

# Managing docker images

Las imágenes de **Docker** son esencialmente una instantánea de un contenedor. Las imágenes se crean con el comando ***build***, que crean un contenedor cuando se inicia con ***run***. Una vez creada una imagen, se pueden almacenar en el **Hub Docker**, o en local.

## Comprendiendo las Capas

Docker gestiona imágenes usando un controlador de almacenamiento back-end. Hay varios controladores soportados como AUFS, BTRFS, y overlays. Las imágenes están hechas de capas ordenadas. Puedes pensar en una capa como un conjunto de cambios en el sistema de archivos. Cuando tomas todas las capas y las apilas, obtienes una nueva imagen que contiene todos los cambios acumulados.

La parte ordenada es importante. Si agregas un archivo en una capa y lo eliminás en otra capa, mejor lo harías en el orden correcto. Docker mantiene un seguimiento de cada capa. Una imagen puede componerse de docenas de capas (el límite actual es 127). Cada capa es muy ligera. El beneficio de las capas es que las imágenes pueden compartir capas.

Si tienes muchas imágenes basadas en capas similares, como Sistema Operativo base o paquetes comunes, entonces todas éstas capas comunes serán almacenadas solo una vez, y el overhead por imagen serán las capas originales de esa imagen.

# Managing docker images

## **Copy on Write (Copiar en Escritura)**

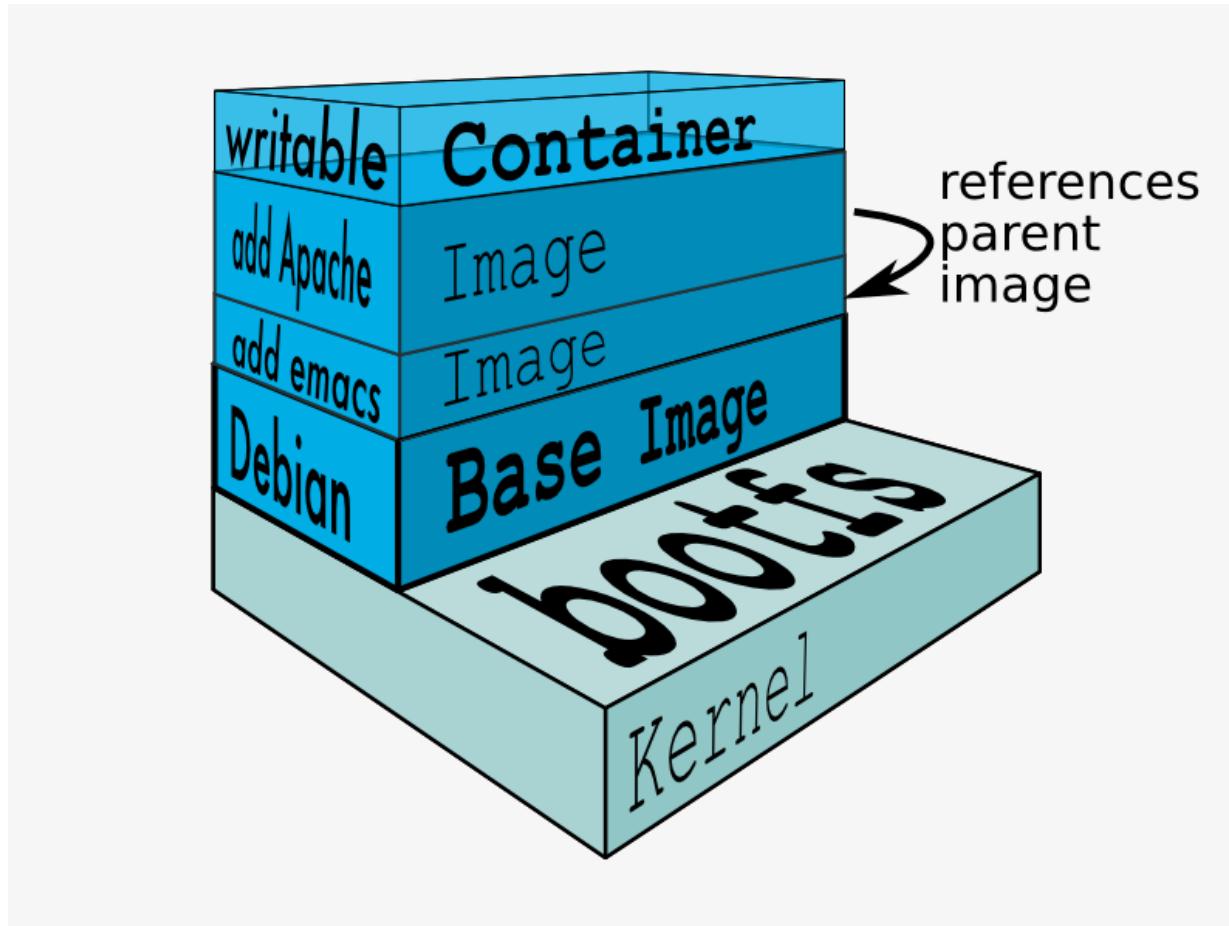
Cuando un nuevo contenedor es creado desde una imagen, todas las capas de la imagen son únicamente de lectura y una delgada capa lectura-escritura es agregada arriba. Todos los cambios efectuados al contenedor específico son almacenados en esa capa.

Ahora, no importa que el contenedor no pueda modificar los archivos desde su capa de imagen. Definitivamente puede. Pero creará una copia en su capa superior, y desde ese punto en adelante, cualquiera que trate de acceder al archivo obtendrá la copia de la capa superior.

Cuando los archivos o directorios son eliminados de capas inferiores, pasan a estar ocultos. Las capas originales de la imagen son identificadas por un hash criptográfico basado en el contenido. La capa lectura-escritura del contenedor es identificada por un UUID (Identificador Único Universal).

Esto permite una estrategia copia-en-escritura para ambas imágenes y contenedores. Docker reutiliza los mismos elementos tanto como sea posible. Solamente cuando un elemento es modificado Docker creará una nueva copia.

# Managing docker images



# Managing docker images

- List images

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
seqvence/static-site	latest	92a386b6e686	2 hours ago	190.5 MB nginx
latest	af4b3d7d5401	190.5 MB python	3 hours ago	2.7
1c32174fd534	14 hours ago	676.8 MB postgres		9.4
88d845ac7a88	14 hours ago	263.6 MB Containous/traefik	latest	27b4e0c6b2fd 4
days ago	20.75 MB			
...				

Tag is like “version”. Latest is... the latest ;)

# Managing docker images

- Buscar imágenes
- \$ docker search

```
root@ubuntu:~# docker search --filter=stars=5 nginx
NAME                           DESCRIPTION                               STARS   OFFICIAL  AUTOMATED
nginx                          Official build of Nginx.          6132    [OK]
jwilder/nginx-proxy            Automated Nginx reverse proxy for docker c... 1044    [OK]
richarvey/nginx-php-fpm        Container running Nginx + PHP-FPM capable ... 382     [OK]
jrcs/letsencrypt-nginx-proxy-companion LetsEncrypt container to use with nginx as... 187     [OK]
webdevops/php-nginx            Nginx with PHP-FPM                         78      [OK]
million12/nginx-php            Nginx + PHP-FPM 5.5, 5.6, 7.0 (NG), CentOS... 77      [OK]
bitnami/nginx                  Bitnami nginx Docker Image                   29      [OK]
evild/alpine-nginx             Minimalistic Docker image with Nginx           16      [OK]
funkygibbon/nginx-pagespeed   nginx + ngx_pagespeed + openssl on docker-... 11      [OK]
webdevops/nginx                Nginx container                           7       [OK]
blacklabelops/nginx            Dockerized Nginx Reverse Proxy Server.       5       [OK]
root@ubuntu:~#
```

# Managing docker images

- **Managing versions**

- Download a concrete version

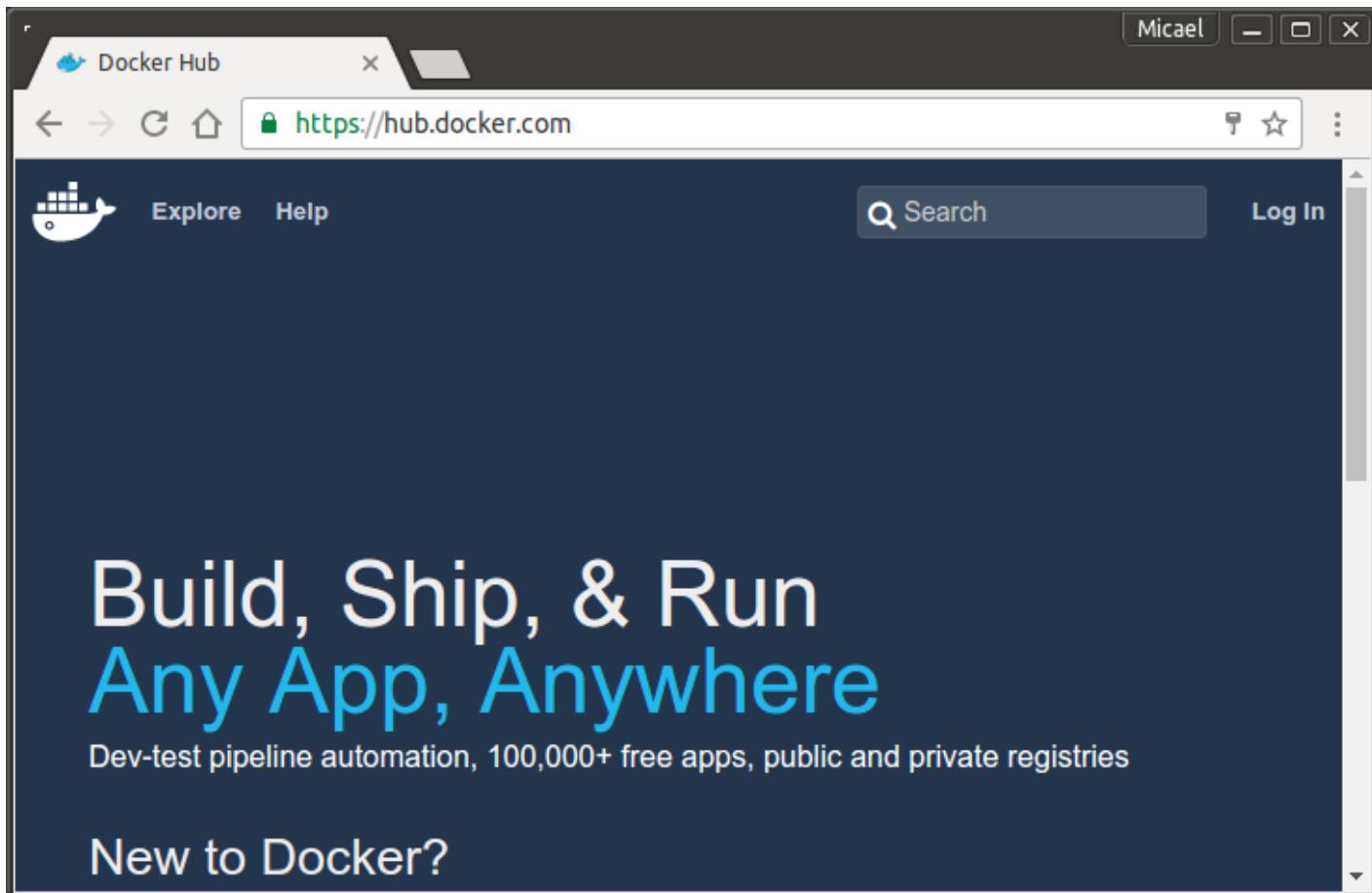
```
$ docker pull ubuntu:12.04
```

- Download latest version

```
$ docker pull ubuntu
```

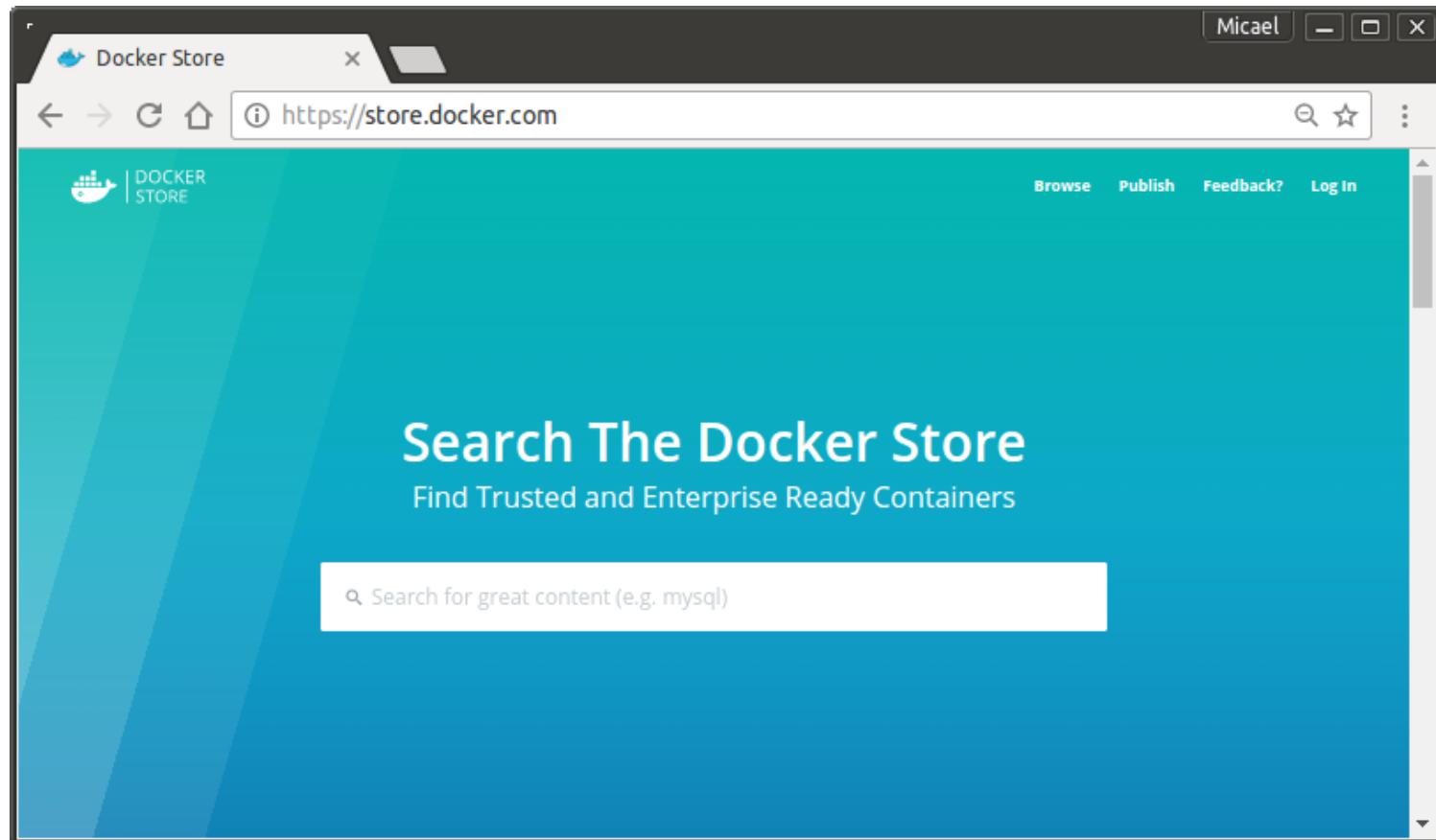
# Managing docker images

- Searching images in DockerHub



# Managing docker images

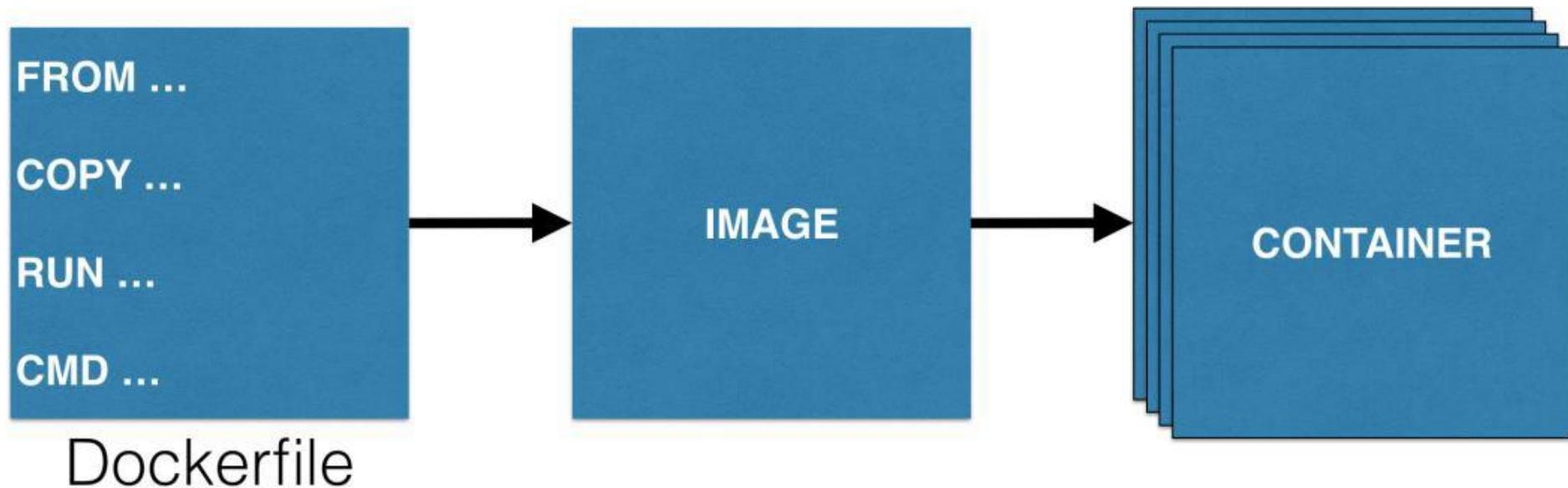
- Searching images in Docker Store



# Managing docker images

- **Image types**
  - **Base images**
    - Images without a parent image
    - Examples: Ubuntu, debian, alpine...
  - **Child images**
    - Base images plus some additional software
    - Examples: Nginx, Apache, MySQL...

# DockerFile



# DockerFile

Docker nos ofrece la posibilidad de indicar las intrusiones requeridas para crear una nueva imagen. Estas instrucciones se incluyen dentro de un fichero llamado Dockerfile.

**Los pasos para crear una imagen a partir de un fichero Dockerfile son las siguientes:**

- Crear un nuevo directorio que contenga un fichero Dockerfile y otros ficheros que fuesen necesarios dentro del contenedor.
- Crear el contenido de para el fichero Dockerfile.
- Ejecutar docker con la acción buil.

## Dockerfile reference

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

# DockerFile

## Ejemplo:

```
mkdir /debianvim2  
cd /debianvim2
```

```
[root@docker debianvim2]# vi Dockerfile  
FROM debian:latest  
RUN apt-get update && apt-get install -y vim  
CMD /bin/bash
```

Donde las expresiones tienen el siguiente significado:

- **FROM**: indica cual es la imagen base para crear la nueva.
- **RUN**: los comandos a ejecutar dentro del contenedor con la imagen base para generar la imagen final.
- **CMD**: el comando por defecto a ejecutar cuando utilicemos esta imagen. Es el punto de entrada a la imagen si no especificamos otro comando con run o créate

En la acción build especificamos el (.) como el directorio actual como argumento, también se podría utilizar el directorio absoluto.

```
[root@docker debianvim2]# docker build -t debianvim2 .
```

# DockerFile

Una vez finalizada todos las acciones y eliminados los contenedores intermedios, se creara la imagen definitiva con el nombre y la etiqueta especificados:

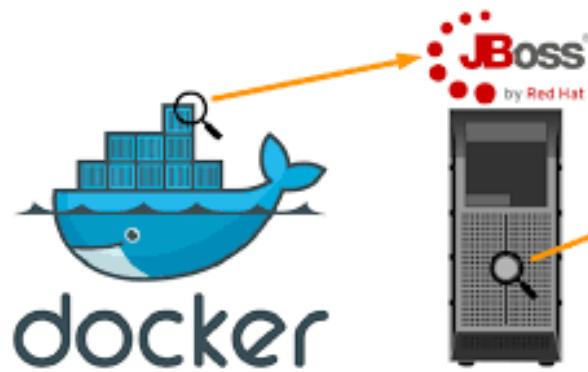
```
[root@docker debianvim2]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
debianvim2 latest aee276dba7e7 22 seconds ago 146 MB
```

Como podemos observar, con Dockerfile podemos de una forma fácil y simple generar nuestras imágenes. Tendremos las siguientes ventajas:

- Es posible reutilizar las plantillas para nuevas imágenes.
- Su sintaxis es sencilla tanto para administradores como para desarrolladores
- Utilizando un control de versiones (git,svn) es posible mantener versiones de estas plantillas.

# DockerFile Demostración

**En este punto realizaremos demostraciones sobre dockerfile para la instalación de un servidor de aplicaciones de Jboss en un contenedor bajo centos7.**



```
# git clone https://github.com/agarciafer/jboss-docker.git
```

**Construimos la imagen, terminamos la orden con un .:**

```
# docker build --rm -f Dockerfile -t jboss-lab .
```

# Redes en Docker

## Redes Predefinidas

Durante la instalación de Docker Engine se crean tres redes, estas son:

- **bridge:** red por defecto. En Linux durante la instalación se crea una nueva interfaz de red virtual llamada docker0. Cuando ejecutamos un contenedor, por defecto, utiliza esta red a no ser que especifiquemos lo contrario.

```
# ifconfig docker0
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
```

# Redes en Docker

## Redes Predefinidas

- **none:** Utilizando esta red, el contenedor no tendrá asociada ninguna interfaz de red, solo la de loopback (lo).
- **host:** utilizando esta red, el contenedor tendrá la misma configuración que el servidor Docker Engine donde se este ejecutando.

A excepción de la red bridge, no es necesario interaccionar con las otras dos. No es posible eliminar las redes predefinidas creadas por el instalador ya que son necesarias para el servicio. Es por ello que veremos como crear nuestras propias redes, donde tendremos mas flexibilidad y será posible eliminarlas si fuera posible.

# Redes en Docker

## Redes Predefinidas

- **none:** Utilizando esta red, el contenedor no tendrá asociada ninguna interfaz de red, solo la de loopback (lo).
- **host:** utilizando esta red, el contenedor tendrá la misma configuración que el servidor Docker Engine donde se este ejecutando.

A excepción de la red bridge, no es necesario interaccionar con las otras dos. No es posible eliminar las redes predefinidas creadas por el instalador ya que son necesarias para el servicio. Es por ello que veremos como crear nuestras propias redes, donde tendremos mas flexibilidad y será posible eliminarlas si fuera posible.

# Redes en Docker

## Listar redes

A través de la acción network del cliente Docker, podremos listar, crear, modificar o borrar redes. Ademas, podremos conectar un contenedor a una red existente, para listar las redes externas utilizaremos:

```
#docker network ls [opciones]  
# docker network list [opciones]
```

## # docker network ls

NETWORK ID	NAME	DRIVER	SCOPE
3fb2d3b156fb	bridge	bridge	local
1aa8d6b12369	host	host	local
d6e5abc27f84	none	null	local

# Redes en Docker

## Crear redes

Cuando trabajamos con distintos entornos, como puede ser desarrollo, test, producción, vamos a necesitar separar los contenedores en diferentes redes. Docker engine nos permite crear redes que permiten aislar las comunicaciones entre ellos.

## La sintaxis es la siguiente:

*#docker network create [opciones] nombre*

-attachable	false	Enable manual container attachment
--aux-address	map[]	Auxiliary IPv4 or IPv6 addresses used by Network driver
--driver, -d	bridge	Driver to manage the Network
--gateway		IPv4 or IPv6 Gateway for the master subnet
--internal	false	Restrict external access to the network
--ip-range		Allocate container ip from a sub-range
--ipam-driver	default	IP Address Management Driver
--ipam-opt	map[]	Set IPAM driver specific options
--ipv6	false	Enable IPv6 networking
--label		Set metadata on a network

# Redes en Docker

## Crear red con Rango Especificado

A través de la acción network create podemos especificar diferentes valores para personalizar nuestra red. En este laboratorio crearemos una red con un rango específico, limitando las direcciones que se pueden utilizar a una ip específica como puerta de enlace.

```
# docker network create --subnet 192.168.100.1/24 --ip-range 192.168.100.100/30 --gateway  
192.168.100.100 produccion
```

2a7b1ed7c6e2b409a7d96d09db30667efc42171016096286bf6fb48abe9703b9

```
# ip a show br-2a7b1ed7c6e2
```

```
9: br-2a7b1ed7c6e2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN  
link/ether 02:42:5c:33:08 brd ff:ff:ff:ff:ff:ff  
inet 192.168.100.100/24 scope global br-2a7b1ed7c6e2  
    valid_lft forever preferred_lft forever
```

# Redes en Docker

## Crear red con Rango Autogenerado

Si la opción `--subnet` no se especifica, Docker Engine buscara un rango libre y lo asignara a esta red.

*La red por defecto bridge es 172.17.0.1/16.*

La primera red personalizada creada (sin especificar `--subnet`) será con la subnet 172.18.0.1/16 y la siguiente 172.19.0.1/16 y asi sucesivamente.

En el servidor se creará una intefaz virtual bridge con el formato br-identificador :

**# docker network create desarrollo**

```
e342b97fe80943490a99828c61e8b9b80bb6a391e882f0b34d095c9c28a4fc97
```

**# ip a show br-e342b97fe809**

```
8: br-e342b97fe809: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
link/ether 02:42:4c:03:6a:d7 brd ff:ff:ff:ff:ff:ff
inet 172.18.0.1/16 scope global br-e342b97fe809
    valid_lft forever preferred_lft forever
```

# Redes en Docker

## Inspecionar la Red

Al igual que con los contenedores con la acción inspect es posibles inspeccionar una red utilizando en este caso la acción network inspect

**La sintaxis es la siguiente:**

# docker network inspect [-f formato] nombre o identificador

**# docker network ls**

*NETWORK ID NAME DRIVER SCOPE*

*3fb2d3b156fb bridge bridge local*

*e342b97fe809 desarrollo bridge local*

*1aa8d6b12369 host host local*

*b1f8f3717a46 interna bridge local*

*d6e5abc27f84 none null local*

*2a7b1ed7c6e2 produccion bridge local*

# Redes en Docker

## Inspeccionar la Red

Al igual que con los contenedores con la acción inspect es posible inspeccionar una red utilizando en este caso la acción network inspect

**La sintaxis es la siguiente:**

# docker network inspect [-f formato] nombre o identificador

**# docker network ls**

*NETWORK ID NAME DRIVER SCOPE*

*3fb2d3b156fb bridge bridge local*

*e342b97fe809 desarrollo bridge local*

**# docker network inspect desarrollo**

# Almacenamiento en Docker

## Almacenamiento

Los contenedores y las imágenes se almacenan en disco directamente. Docker pose una arquitectura de almacenamiento flexible que permite elegir entre diversos drivers dependiendo del sistema operativo y las necesidades del entorno. Cada driver de almacenamiento Docker se basa en el sistema de ficheros Linux o un administrador de volúmenes como (LVM).

### Linux distribution

Docker CE on Ubuntu

16.04 or later), overlay, zfs

Docker CE on Debian

Docker CE on CentOS

Docker CE on Fedora  
overlay (experimental)

### Supported storage drivers

aufs, devicemapper, overlay2 (Ubuntu 14.04.4 or later,  
zfs

aufs, devicemapper, overlay2 (Debian Stretch), overlay  
devicemapper

devicemapper, overlay2 (Fedora 26 or later, experimental),

### Storage driver      Supported backing filesystems

overlay, overlay2 ext4, xfs

aufs ext4, xfs

devicemapper direct-lvm

btrfs btrfs

Zfs zfs

# Almacenamiento en Docker

## Almacenamiento

Dependiendo del sistema operativo que utilicemos y su versión de Docker elegirá por defecto el driver mas optimo y estable, podemos comprobarlo con el comando:

```
[root@docker ~]# docker info
Server Version: 1.12.6
Storage Driver: devicemapper
Pool Name: docker-253:0-3485556-pool
Pool Blocksize: 65.54 kB
Base Device Size: 10.74 GB
Backing Filesystem: xfs
```

# Almacenamiento en Docker

## Almacenamiento

Las ventajas y desventajas de cada uno de los drivers de almacenamiento:

AUFS	stable	production-ready	good memory use	smooth Docker experience	high write activity	PaaS-type work
Devicemapper (loop)	stable	in mainline kernel	smooth Docker experience	production	performance	lab testing
Devicemapper (direct-lvm)	stable	production-ready	in mainline kernel	smooth Docker experience	PaaS-type work	
Btrfs	in mainline kernel	high write activity	container churn	build pools		
Overlay	stable	good memory use	in mainline kernel	container churn	lab testing	
ZFS native (ZoL)		PaaS-type work				
ZFS FUSE	stable	lab testing	production			

### Key

Has attribute	attribute
If good for use case	use case
If bad for use case	use case

# Volúmenes de datos

## Volumenes

Un volumen en Docker es un directorio especial asignado a un contenedor y que aloja los datos dentro del directorio /var/lib/docker/volumes.

- Las ventajas de utilizar volúmenes son las siguientes:
- Se inicializan durante la creación del contenedor.
- Se pueden compartir entre diferentes contenedores.

Las modificaciones realizadas inmediatamente a disco sin pasar primero por cache (con la posibilidad de que esto pueda significar perdida de datos).

- Los cambios en el volumen no se verán afectados en el caso de que se actualice la imagen base del contenedor.

<https://docs.docker.com/engine/tutorials/dockervolumes/>

# Volúmenes de datos

**Para crear un contenedor durante las acciones run o créate, utilizaremos la opción -v (--volume) y el nombre del directorio como se muestra en el ejemplo:**

```
# docker run -dtiP --name centos6-lamp -v /var/www/html docker.io/nickistre/centos-lamp
```

**Al igual que con otras configuraciones relacionadas con los contenedores, es posible utilizar la acción inspect:**

```
# docker inspect -f "{{ .Mounts }}" centos6-lamp
[{"Target": "/var/www/html", "Type": "bind", "Source": "/var/lib/docker/volumes/0353ca0338ba0c02b8d6a42d940dac27dae4b909fb6d724ee8adbeb8ba46eb0b/_data", "Readonly": false, "Sticky": false, "BindOptions": ""}]
```

**Podemos manipular los ficheros de configuración del volumen:**

```
# ls -l
/var/lib/docker/volumes/0353ca0338ba0c02b8d6a42d940dac27dae4b909fb6d724ee8adbeb8ba46eb0b/_data
total 8
-rw-r--r-- 1 root root 159 jun 26 13:10 index.php
-rw-r--r-- 1 root root 22 may 18 2016 phpinfo.php
```

# Volúmenes de datos

La opción **-v** (**--volume**) **admite otra sintaxis** para compartir un directorio del servidor Docker al contenedor, la sintaxis es la siguiente:

**-v/volumen directorio del servidor:directorio contenedor.**

En el siguiente ejemplo queremos un directorio llamado **/web**, en el servidor de Docker, y este directorio lo hacemos accesible para los apaches que se ejecuten en los contenedores, con los queremos compartir:

```
[root@docker ~]# mkdir /web
```

Copiamos un archivo **index.html** a el directorio **/web**

Ejecutamos un contenedor utilizando un volumen con el directorio del servidor, si accedemos al apache de los dos contenedores tendrán la misma web:

```
# docker run -dtiP --name centos6-prueba-web -v /web:/var/www/html  
docker.io/nickistre/centos-lamp
```

```
# docker run -dtiP --name centos6-prueba-web2 -v /web:/var/www/html  
docker.io/nickistre/centos-lamp
```

# Volúmenes de datos

- En Windows y Mac con Docker Toolbox (no con la última versión) puede haber limitaciones al montar una carpeta
  - En Mac sólo están disponibles por defecto las carpetas dentro de /Users
  - En Windows sólo están disponibles por defecto las carpetas de C:\Users

```
$ docker run -v //c/<path>:/<container path>
```

- Para montar nuevas carpetas es necesario que estén accesibles por la máquina virtual VirtualBox

<https://docs.docker.com/engine/tutorials/dockervolumes/>

# Volúmenes de datos

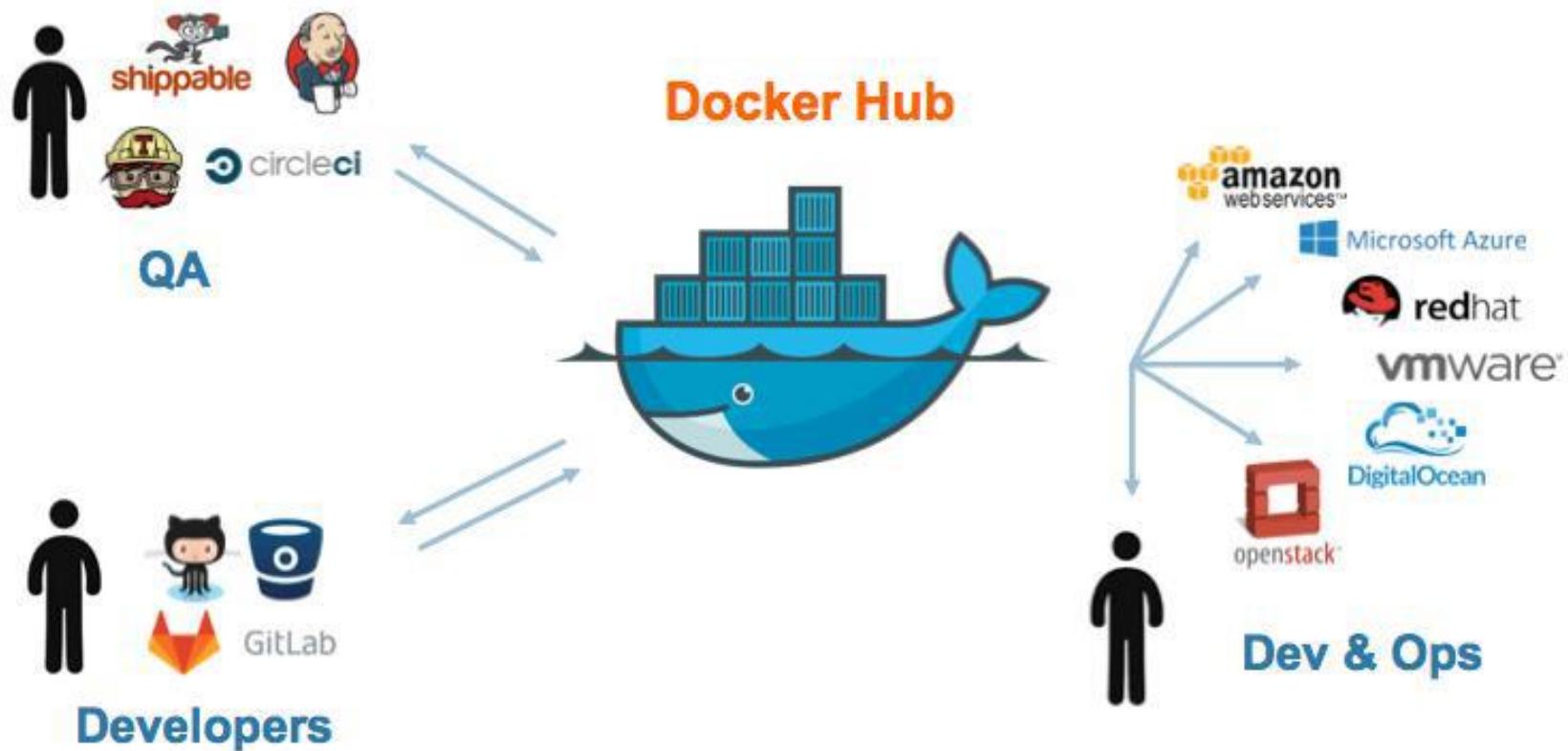
- Por ejemplo, si queremos **servir por http** un contenido estático que tengamos en el disco duro, podemos usar la **imagen oficial de NGINX**

```
docker run -d -p 9000:80 -v $PWD:/usr/share/nginx/html nginx
```

- **\$PWD** se usa para obtener la ruta del directorio actual
- **/usr/share/nginx/html** es la carpeta que publica por http el NGINX instalado en el contenedor

[https://hub.docker.com/\\_/nginx/](https://hub.docker.com/_/nginx/)

# Docker Hub



# Docker Hub

Docker Hub que permite a los usuarios compartir las imágenes construidas, se podría decir que es el GitHub de los contenedores docker y quizá por ello el paralelismo en el nombre entre ambos. Docker Hub permite subir imágenes o usar las imágenes oficiales de postgresql, redis, mysql, ubuntu, rabbitmq, ... y otra multitud de proyectos.

En la web oficial del repositorio de Docker

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

Es posible registrarse de forma gratuita para alojar las imágenes que hemos creado. Es importante tener en consideración que las imágenes creadas pueden ser accesible públicamente, por lo cual no debemos de alojar datos comprometidos.

# Docker Hub

Una vez registrado con un usuario y una contraseña (llamada Docker ID), para almacenar imágenes dentro del repositorio nosotros debemos nombrar a la imagen con el formato:

**usuario/imagen:versión**

Esta tarea puede realizarse con la acción tag.

```
# docker tag intranet agarciaf/intranet
```

# Docker Hub

Antes de realizar la acción de publicar la imagen debemos de autenticarnos con la cuenta previamente creada. Para ello utilizaremos la acción login.

## # docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

Username:

Password:

# Docker Hub

Para publicar una imagen utilizaremos la siguiente sintaxis:

```
# docker push usuario/imagen[:versión]
```

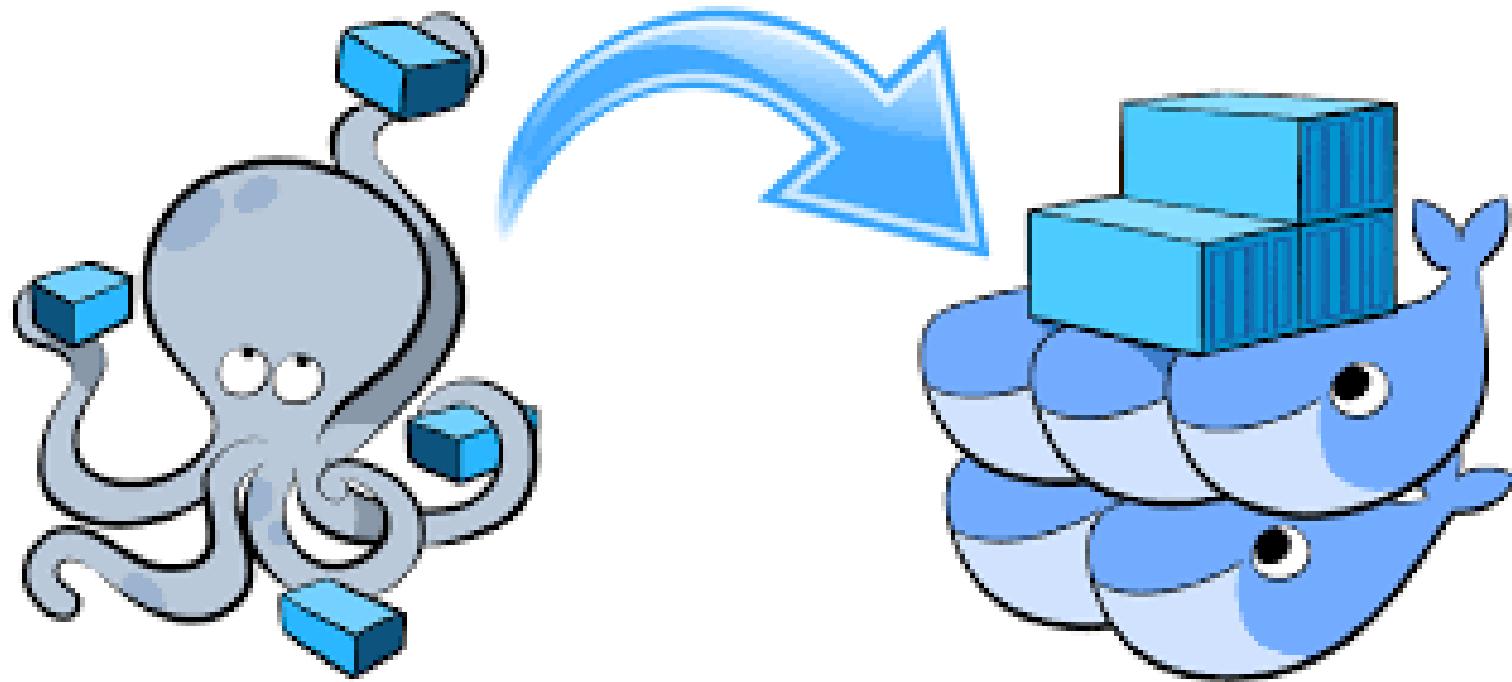
```
# docker push agarciaf/intranet
```

Ahora podremos buscar las imágenes creadas por en nombre del usuario, o bien por el nombre de la imagen:

```
# docker search agarciaf
```

INDEX	NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
docker.io	docker.io/agarciaf/centos6-joomla	0	0		
docker.io	docker.io/agarciaf/debian-vim	0	0		
docker.io	docker.io/agarciaf/intranet		0		

# Usos de los contenedores



# Usos de los contenedores

- Los contenedores se suelen usar de dos formas diferentes
  - **Contenedores para servicios**
    - Se ejecutan en segundo plano (-d)
    - Se consumen sus servicios vía red
    - Ejemplos: base de datos, servidor web...
  - **Contenedores para comandos**
    - Se ejecutan para que se borren al terminar (--rm)
    - Se comunican con el host mediante volúmenes y parámetros del comando que se ejecuta
    - Ejemplos: Compiladores, ffmpeg, generadores de sitios web

# Usos de los contenedores

- **Contenedores para comandos**

- Jekyll es una herramienta que genera un sitio web partiendo de ficheros de texto (Markdown)
- Es el sistema que usa GitHub para sus gh-pages
- Jekyll se puede usar desde un **contenedor** sin tener que instalar nada en el host



# Usos de los contenedores

- **Contenedores para comandos**
  - Descargar contenido de ejemplo

```
$ git clone https://github.com/henrythemes/jekyll  
minimaltheme
```

- Ejecutar el contenedor para generar el sitio web en la carpeta descargada

```
$ docker run rm v "$PWD:/src" grahamc/jekyll build
```

- El resultado se genera en la carpeta `_site`

<https://hub.docker.com/r/grahamc/jekyll/>

# Aplicaciones multicontenedor

- Docker recomienda que cada contenedor se use para **un único proceso**
- Pero es habitual que las aplicaciones tengan **varios procesos (web + bd)**
- **Docker-compose** nos permite definir un conjunto de contenedores que colaboran entre si formando una aplicación

<https://docs.docker.com/compose/>

# Aplicaciones multicontenedor

## Docker Compose

Con Dockerfile podemos usar Docker Engine para crear imágenes y posteriormente desplegar contenedores con ellas. Docker nos proporciona otro componente llamado Compose para definir y ejecutar contenedores basados en plantillas

**Esta plantilla contendrá:**

- Lista de imágenes a utilizar para ejecutar contenedores.
- La ruta de los ficheros Dockerfile que crearan las imágenes previamente especificando si furea necesario.
- Los puertos a exponer para acceder a dicho contenedor
- Los volúmenes a utilizar
- Las variables necesarias para ejecutar las aplicaciones

# Aplicaciones multicontenedor

## Las principales características de este componente son:

- Compose fue diseñado para ejecutar contenedores en un solo servidor y no en alta disponibilidad.
- Mas centrado en entornos de desarrollo que de producción.

## Los casos de uso mas común de Docker Compose:

- Entornos de desarrollo: simplifica todo el proceso de creación de imágenes, de ejecución de contenedores y comunicación entre ellos.
- Automatización de test a aplicaciones: es posible utilizar lo denominado continuous integration (**CI**), integración continua, para desplegar aplicaciones que están bajo desarrollo dentro de un contenedor y comprobar su estado y sus dependencias.
- En el caso de solo poseer un único servidor Docker, es posible utilizarlo como herramienta para desplegar aplicaciones.

Para entornos de producción con diferentes servidores, se recomienda utilizar Docker Swarm, kubernetes para crear un cluster que contendrá los contenedores en alta disponibilidad.

# Aplicaciones multicontenedor

## Algunos comandos docker compose

- **docker-compose build**: Lanza el proceso de generación de imágenes docker de los servicios definidos en docker-compose.yml
- **docker-compose create**: Lanza el proceso de creación de los servicios indicado en el archivo docker-compose.yml
- **docker-compose start**: Inicia el servicio indicado
- **docker-compose stop**: Detiene el servicio indicado
- **docker-compose rm**: Elimina el servicio, no la imagen, indicado
- **docker-compose exec**: Ejecuta un comando en el container del servicio indicado
- **docker-compose up**: Crea imágenes, crea los servicios y los inicia según el docker-compose.yml
- **docker-compose down**: Detiene y elimina servicios

# Demostración Aplicaciones multicontenedor

En esta demo de Docker Compose instalaremos Wordpress en un entorno aislado construido mediante contenedores Docker.



# Demostración Aplicaciones multicontenedor

## docker-compose.yml

```
1  version: '2'
2  services:
3    db:
4      image: mysql:5.7
5      volumes:
6        - "./.data/dbwordpress:/var/lib/mysql"
7      restart: always
8      environment:
9        MYSQL_ROOT_PASSWORD: wordpress
10       MYSQL_DATABASE: wordpress
11       MYSQL_USER: wordpress
12       MYSQL_PASSWORD: wordpress
13
14    wordpress:
15      depends_on:
16        - db
17      image: wordpress:latest
18      links:
19        - db
20      ports:
21        - "8000:80"
22      restart: always
23      environment:
24        WORDPRESS_DB_HOST: db:3306
25        WORDPRESS_DB_PASSWORD: wordpress
```

# Demostración Aplicaciones multicontenedor

Construyendo el proyecto:

Ahora vamos a ejecutar el comando **docker-compose up -d** estando situados dentro del directorio del proyecto.

Este comando descargará ( pull ) las imágenes necesarias y iniciará el contenedor de Wordpress y de la base de datos.

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

Con esto ya tendremos creados y corriendo dos contenedores:

- **wordpress\_db\_1** : Es el contenedor MySQL para nuestra base de datos de Wordpress.
- **wordpress\_wordpress\_1** : Es el contenedor que contiene nuestra instancia de WordPress además del servidor Web apache.

\* El directorio **./.data/db** será automáticamente creado en el directorio del proyecto, a la misma altura que el fichero `docker-compose.yml` , y contendrá los datos persistentes de la base de datos de nuestro WordPress.

# Demostración Aplicaciones multicontenedor

Construyendo el proyecto:

Ahora vamos a ejecutar el comando **docker-compose up -d** estando situados dentro del directorio del proyecto.

Este comando descargará ( pull ) las imágenes necesarias y iniciará el contenedor de Wordpress y de la base de datos.

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

Con esto ya tendremos creados y corriendo dos contenedores:

- **wordpress\_db\_1** : Es el contenedor MySQL para nuestra base de datos de Wordpress.
- **wordpress\_wordpress\_1** : Es el contenedor que contiene nuestra instancia de WordPress además del servidor Web apache.

\* El directorio **./.data/db** será automáticamente creado en el directorio del proyecto, a la misma altura que el fichero `docker-compose.yml`, y contendrá los datos persistentes de la base de datos de nuestro WordPress.

# Docker en cluster

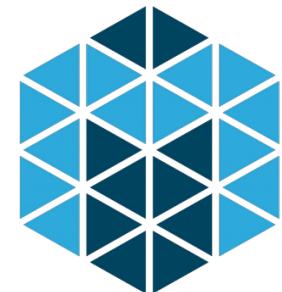
- Orquestadores de contenedores



<https://docs.docker.com/engine/swarm/>



<http://kubernetes.io/>



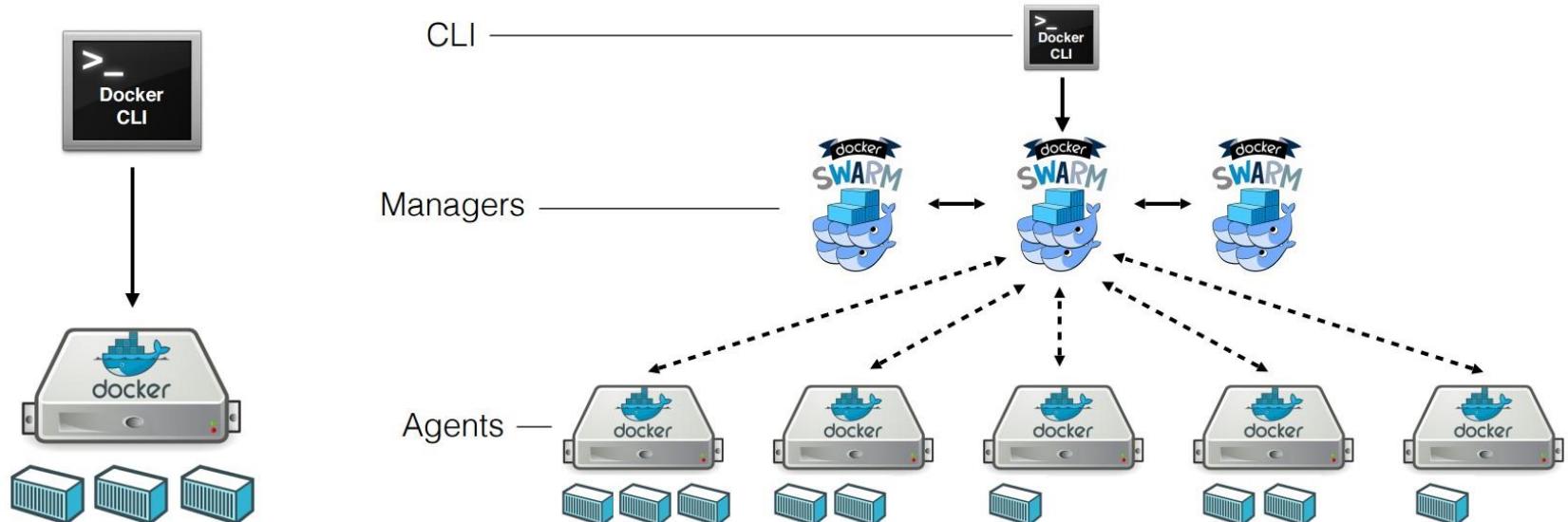
<http://mesos.apache.org/>

# Docker en cluster

- **Orquestadores de contenedores**
  - Software encargado de gestionar la ejecución de contenedores en un **cluster** de máquinas
  - Pueden tratar **varios contenedores** como una única unidad lógica (**aplicación**)
- **Servicios ofrecidos**
  - **Redes aisladas** para contenedores de la misma app
  - Políticas de **reinicio** en caso de **caída** del servicio
  - Políticas de **replicación** de contenedores por carga

# Docker en cluster

- Docker Swarm



Docker en un servidor

Docker en un cluster

# Docker en cluster

- **Kubernetes**

- Desarrollado principalmente por **Google** basado en sus sistemas internos de orquestación (Borg)
- Muy **maduro**. El más utilizado y completo
- Disponible de forma sencilla en Google Cloud Platform y se puede instalar en local, AWS, etc
- Conceptos y comandos diferentes a docker en local

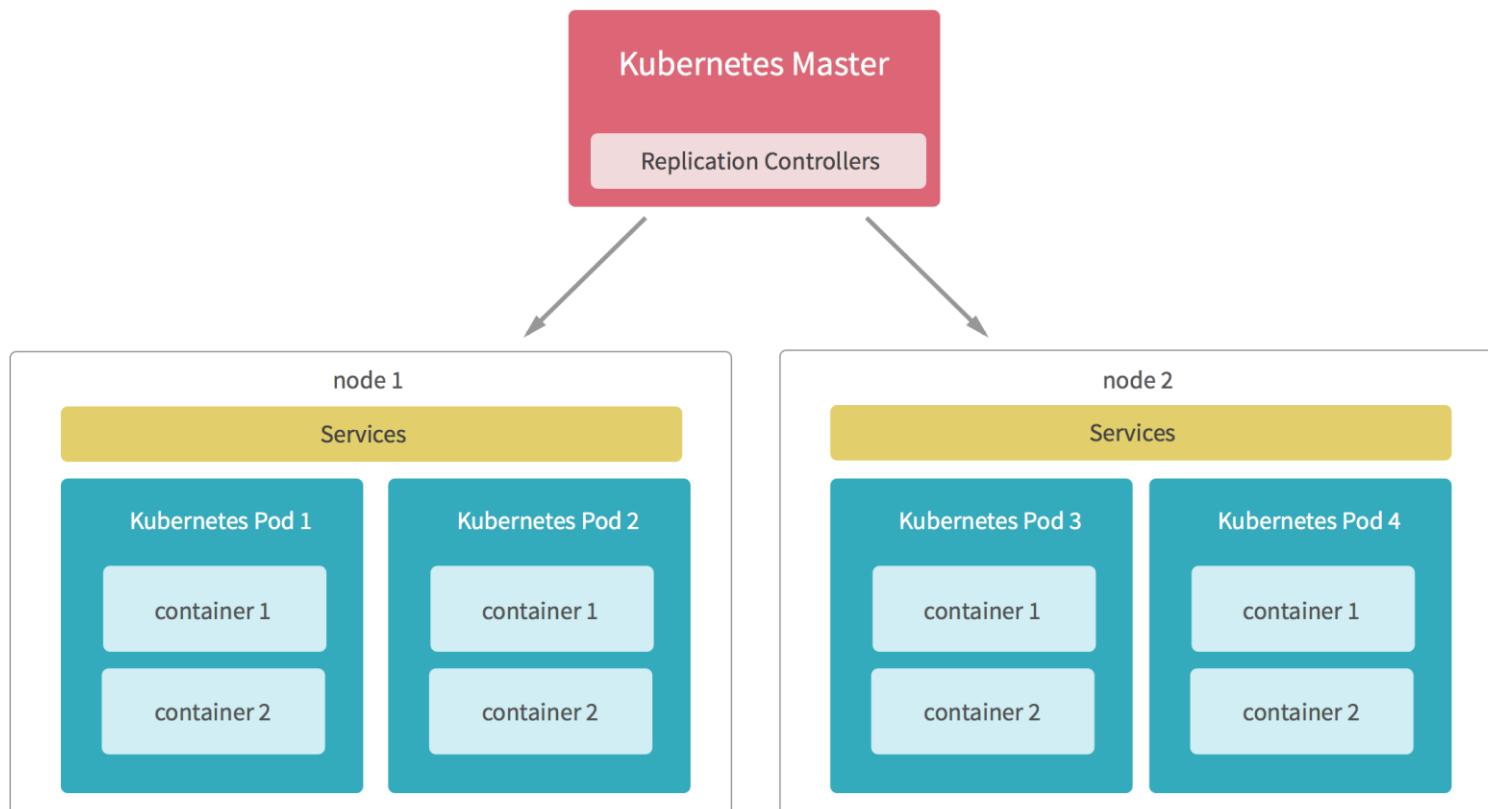


<http://kubernetes.io/>

<https://www.cncf.io/>

# Docker en cluster

- **Kubernetes**



# Docker en cluster

MESOS

- **Apache Mesos**

- Desarrollado en la fundación Apache
- Se usa sobre todo como una plataforma para gestionar las máquinas del cluster como una unidad
- Permite ejecutar Hadoop, Kafka, Spark en el mismo cluster y también **contenedores**
- Cada vez se parece más a kubernetes

<http://mesos.apache.org/>

# Docker en cluster

- Existen diversas formas de **ejecutar contenedores** en un proveedor cloud
  - Ejecución manual de contenedores en una VM (IaaS)
  - Instalación de un orquestador de contenedores en un cluster de VMs
  - Uso de servicios de orquestación ofrecidos por el proveedor cloud

# Docker en cluster

- Instalación de un orquestador de contenedores en un cluster de VMs

- **Kubernetes**

- Se puede instalar en AWS y Azure

- <http://kubernetes.io/docs/getting-started-guides/aws/>

- <http://kubernetes.io/docs/getting-started-guides/azure/>

- Se ofrece como servicio en Google Cloud

- **Docker Swarm**

- Se puede instalar de forma sencilla en AWS y Azure

- <https://docs.docker.com/docker-for-aws/>

- <https://docs.docker.com/docker-for-azure/>

# Docker en cluster

- Muchos proveedores PaaS están ofreciendo la posibilidad de ejecutar la app como un contenedor docker

Basadas en kubernetes

## Orquestación propietaria



Amazon EC2  
Container Service  
(ECS)



Google Container  
Engine (GKE)



**GRACIAS**

**ARIGATO**

**SHUKURIA**

**JUSPAXAR**

**GOZAIMASHITA**

**EFCHARISTO**

**KOMAPSUNNIDA**

**MAAKE**

**LAH**

**FAKAUAE**

**TASHAKKUR ATU**

**SUKSAMA**

**MEHRBANI**

**PALDIES**

**GRAZIE**

**MEHRBANI**

**PALDIES**

**BOLZİN**

**TINGKI**

**HATUR GUI**

**EKOU**

**SIKOMO**

**MINMONCHAR**

**MERCY**

**SHUKRIA**

**DANKSCHEEN**