Docker from scratch

Antonio Jesús Gil

ajgilp@etics.es

```
https://www.docker.com/get-started
```

```
git clone https://github.com/antoniojesusgil/dockerFromScratch.git
```

Intro: Docker from scratch

- ¿Qué es Docker? ¿Qué es un contenedor?
- ¿Por qué aprender Docker?
- Contenedores Vs Máquinas virtuales
- Docker y la infraestructura
- Docker y la comunidad

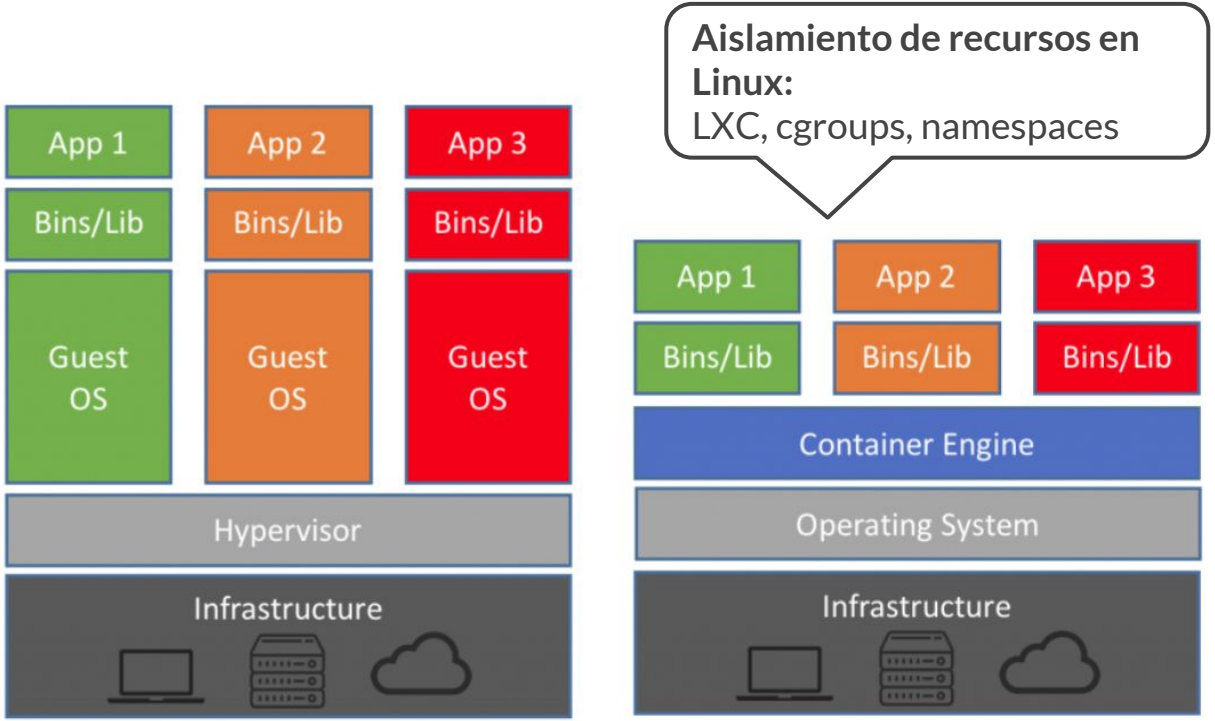
¿Por qué aprender Docker?

- Mejoramos como desarrolladores
- Mejor entorno, menos mantenimiento y menos bugs
- Mayor integración y colaboración entre equipos
- Acceso a infraestructura / contenedores as a Cloud service

¿Qué es ...?

- Un contenedor **encapsula** software en un sistema de ficheros
- Contiene todo lo **necesario para su ejecución**
- Garantiza **estabilidad y portabilidad** a todos los sistemas
- Docker es:
    - **Ligero**, consume pocos recursos
    - **Abierto** y compatible (es software libre)
    - **Seguro**, permite aislamiento entre contenedores

Contenedores Vs Máquinas Virtuales



Docker y la infraestructura

- La compatibilidad depende de tu infraestructura
  - Contenedor Windows Server solo plataforma Windows
  - * Contenedores Linux nativos en windows
  - Arquitectura CPU influye (Raspberry Pi ARM)

* Docker nativo desde 2016 para Windows 10 y Mac OS

Docker & comunidad

- Adoptado por las grandes empresas, completamente estandarizado.
- Comparte tu código + Dockerfile para que otros puedan colaborar o desplegar con mayor facilidad
- Gran catálogo de imágenes y recetas disponibles
- Libre

Índice: Docker from scratch, conceptos básicos

Conceptos

- Docker Engine / Daemon
- Container
- Image
- Registry
- DockerHub
- Network & Volumes
- Dockerfile
- Compose

Docker Engine & CLI

Docker Engine CLI

- CLI como cliente principal
- Dispone un API remota
- Demonio docker se ejecuta en segundo plano y gestiona la ejecución de los contenedores

```
$ docker
    | info
    | run
    | kill
    | stop
    | start
    | ps
    | logs
    | rm
    | images
    | build
    | rmi
    | network
    | volumes
    | ...
```

Lab 0 - Engine & CLI

Instalando docker

docker CLI

Hello-world

```
docker info
```

Muestra el estado del servicio

```
docker run hello-world
```

Contenedor de test

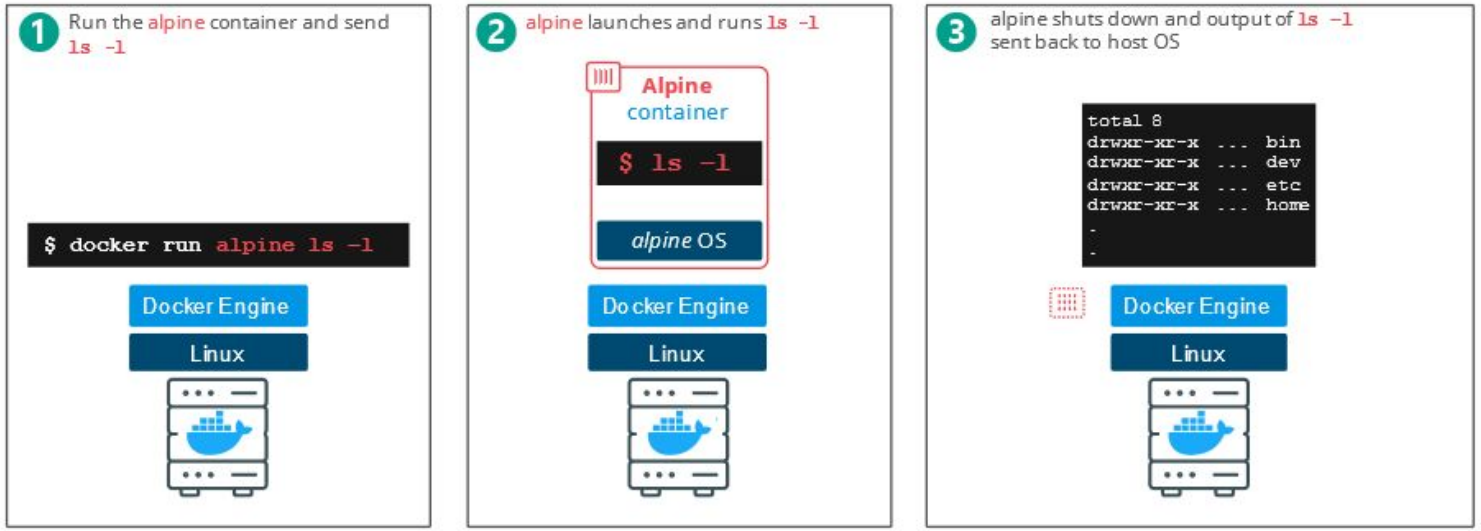
```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Contenedor real e interactivo

```
docker run -it ubuntu /bin/bash
```

¿Qué ha sucedido?

Detalle


1
Run the alpine container and send ls -l
$ docker run alpine ls -l
Docker Engine
Linux
2
alpine launches and runs ls -l
Alpine
container
$ ls -l
alpine OS
Docker Engine
Linux
3
alpine shuts down and output of ls -l sent back to host OS
total 8
drwxr-xr-x ... bin
drwxr-xr-x ... dev
drwxr-xr-x ... etc
drwxr-xr-x ... home
.
.
Docker Engine
Linux

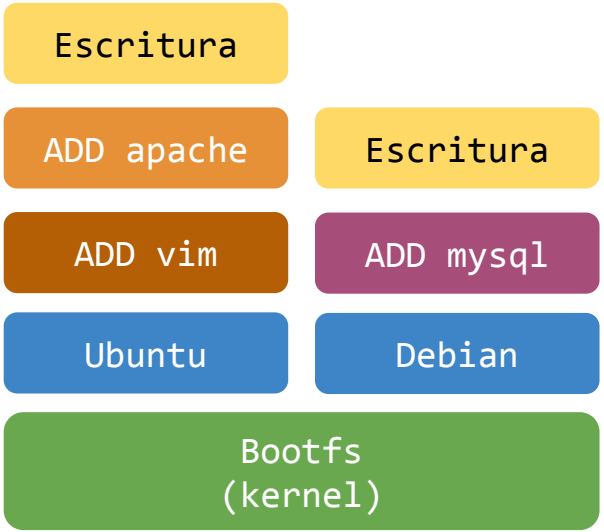
Containers & Images

Contenedores

- Se inician desde una imagen y creando su estructura de ficheros
- Pueden encontrarse en **ejecución** o **parados**
- **Persisten los cambios** realizados sobre la imagen base
- Suelen ejecutarse como **servicios**, se puede acceder de manera interactiva
- Docker puede configurar redes y volúmenes compartidos entre contenedores y host

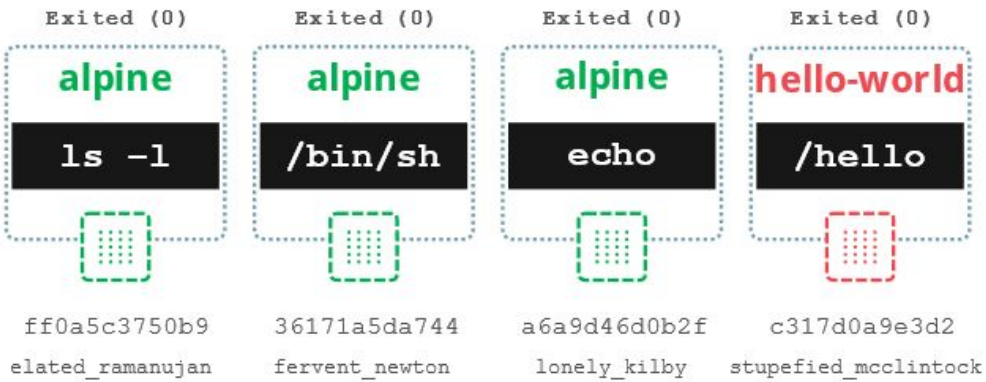
Imágenes

- Las utilizaremos como punto de inicio de nuestros contenedores
- Definen una serie de capas que añaden funcionalidad a partir de una **imagen base** (SO)
- Podemos **gestionar** imágenes:
  - Definidas por nosotros (Dockerfile)
  - Descargadas desde un **registro**



Instancias

- Una instancia es un contenedor de una imagen
- Servicio confiable y **aislado**
- Pueden estar en ejecución o terminadas




```
Docker container ls -a
```

Lab 1 - Containers & Images

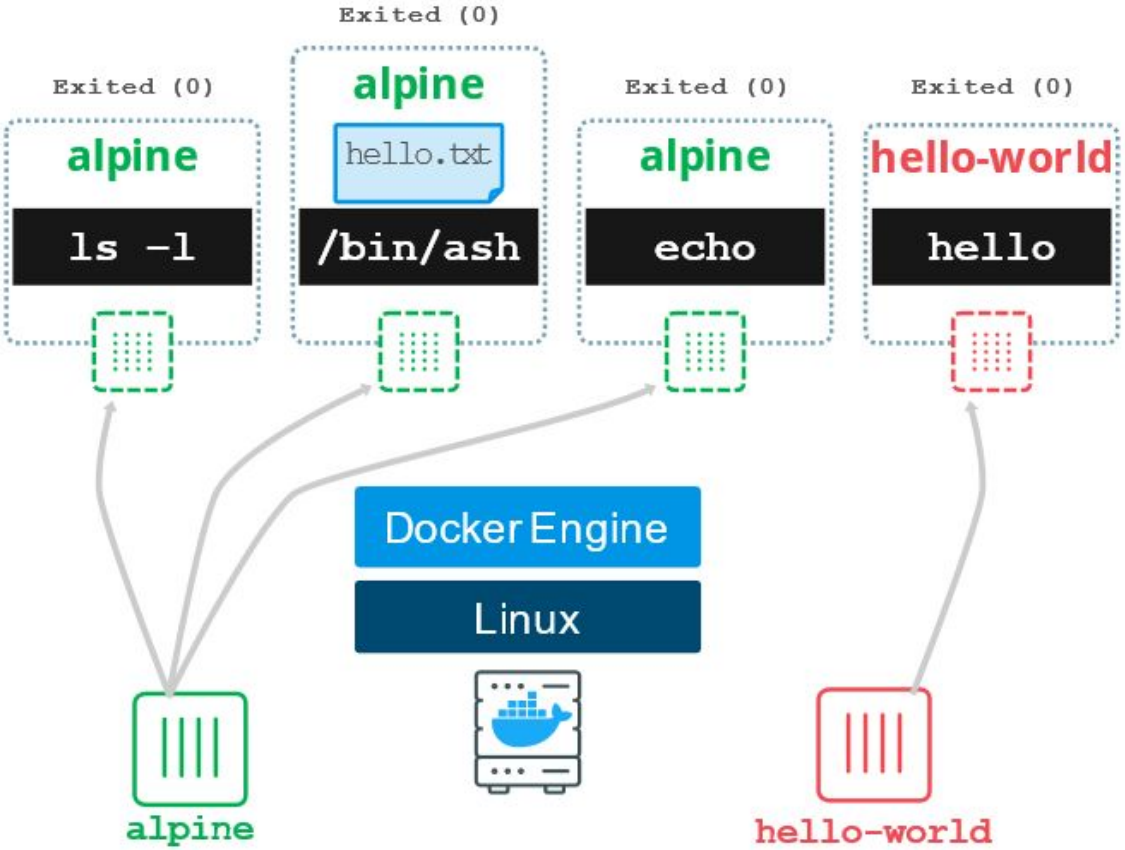
Isolation

Ubuntu

Más docker CLI

Instancias

Imágenes



```
docker ps
```

```
docker ps -a
```

Listar contenedores en ejecución

-a contenedores parados

```
docker run -rm -t -i <image> <command>
```

Alterar el estado de contenedores

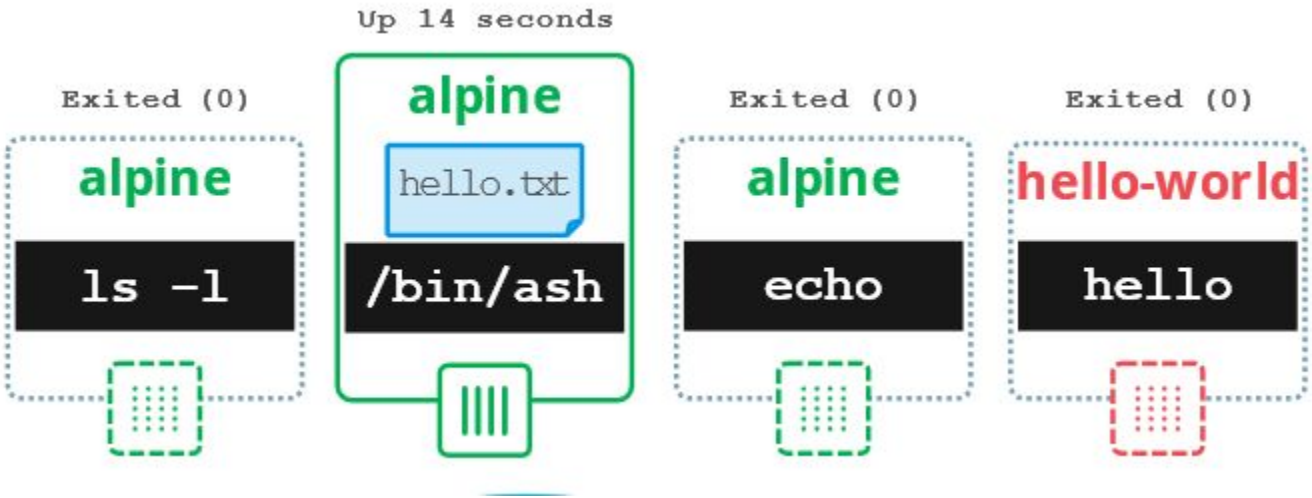
```
docker stop <container>
```

```
docker kill <container>
```

```
docker rm <container>
```

```
docker start <container>
```

```
docker exec <container> ls /home
```



Registro & DockerHub

Registro

- **Repositorios** a los subimos imágenes para utilizarlas nosotros, colaboradores o terceros

```
docker push
```

```
docker pull
```

- Las imágenes están **disponibles** a través del cliente docker.
- Permite **control de versiones**, **etiquetado**...

```
<image-name>:<tag>
```

```
ubuntu:latest
```

```
ubuntu:xenial
```

Dockerhub

- https://hub.docker.com
- Registro **oficial** y **gratuito** de docker
- +400k repositorios
- Imágenes oficiales de las plataformas más importantes: distros linux, servidores BBDD, web, librerías…
- Posibilidad de cuenta **privada** para empresas

Lab 2 - Dockerhub

Sing Up

Explorar imágenes base

Visita y crea una cuenta en
https://hub.docker.com

```
docker login
```

Inicia sesión con tus credenciales

No te olvides de hacer logout
cuando te vayas

```
docker logout
```

```
docker run --rm -ti python:3.6-slim
```

```
docker run --rm -ti node:8-slim
```

```
docker run --rm nginx
```

```
Docker images ls
```

```
docker images rm / docker rmi
```

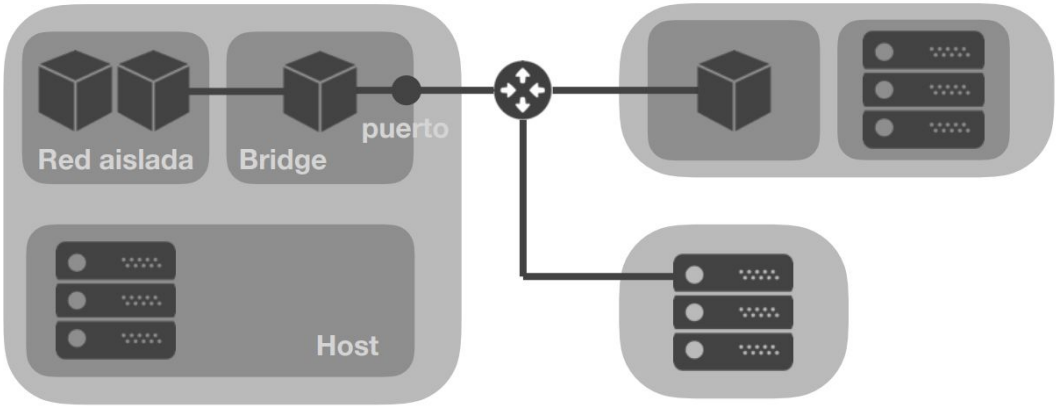
Listar / borrar imágenes en nuestro sistema local

Importante mantener nuestro docker limpio

Network & Volumes

Network: Comunicación entre contenedores

- Los contenedores pueden conectarse entre ellos o a otros sistemas mediante redes
- Podemos crear múltiples adaptadores y controlar los puertos accesibles a cada contenedor para crear diversas arquitecturas



Persistencia: Volúmenes

- Docker puede crear **volúmenes de datos persistentes** que se montan sobre el sistema de ficheros de un contenedor.
- Los volúmenes pueden **compartirse** entre contenedores para establecer **comunicación**
- Podemos montar un **directorio local** como volumen de datos

Lab 3 - Redes & Volúmenes

Binding puertos y directorios

Servidores Web

Ejecutar código desarrollo

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

Bind de un puerto del contenedor al host

```
docker run -p 80:80 --rm nginx
```

Podemos desplegar un servidor web en nuestro loop local
Visita http://localhost:80

```
docker run -p 8080:80 --rm nginx
```

Despliegue en un puerto diferente

```
docker run -p 80:80 -d nginx
```

Despliegue en modo detach

```
docker stop
```

```
docker kill
```

```
docker run -v <local-dir>:<remote-dir> <image>
```

Bind de un directorio local a un contenedor

```
docker run -v $(pwd):/root --rm --ti ubuntu
```

Montar la carpeta actual en el directorio /root del contenedor
Aviso: $(pwd) puede no funcionar en tu shell
La ruta debe ser absoluta

Anexo: volúmenes en Windows

- **$(pwd) --> %cd%**

```
“%cd%/webB:/usr/share/nginx/html:ro”
```

- También ruta absoluta entre comillas
- Otra opción:

```
/C/Users/casa/Documents/docker-from-scratch/webB:/usr/share/nginx/html:ro
```

```
cd Lab3
```

```
docker run --rm -d -p 80:80 \
-v $(pwd)/webA:/usr/share/nginx/html:ro nginx
```

La imagen nginx viene preparada para servir desde /usr/share/nginx/html

```
docker run --rm -d -p 8000:80 \
-v $(pwd)/webB:/usr/share/nginx/html:ro nginx
```

Podemos desplegar más servidores.
Probad a modificar el código de una web

```
docker run --rm -v $(pwd):/root python3.6-slim python3
/root/very-useful.py docker-from-scratch
```

Podemos utilizar volúmenes para compartir y ejecutar código dentro de un contenedor

Dockerfile

Dockerfile

- Un script que define la construcción (**build**) de una imagen docker

Dockerfile.dockerfile

```
1   FROM ubuntu:14.04
2
3   LABEL maintainer=”ajgil”
4
5   RUN apt-get update && \
6       apt-get install -y apache2
7
8   COPY index.html /var/www/
9
10  EXPOSE 80
11
12  CMD [“/usr/sbin/apache2”, “-D”, “FOREGROUND”]
```

```
Docker build -t <namespace>/<image-name>:<tag> <dir>
```

Dockerfile: Comandos

- Cada comando en crea una **capa** en un contenedor
- Cada capa se almacena en una **caché** para ser reutilizada
- Las imágenes y contenedores pueden **compartir capas**

Dockerfile: Comandos

`FROM`	Añade las capas de una imagen base
`RUN`	Ejecuta un comando dentro del contenedor
`COPY/ADD`	Copia ficheros desde nuestro sistema
`USER`	Cambia al usuario que ejecuta el contenedor
`WORKDIR`	Cambia el directorio desde el que se ejecuta
`ENV`	Declara variables de entorno
`CMD`	Modifica el comando por defecto
`EXPOSE`	Especifica los puertos disponibles (no hace bind)

Dockerfile: Buenas prácticas

- Diseña cada contenedor para una **función específica**
- Los contenedores deben ser **efímeros**
- Minimiza el número de capas y su **tamaño** (caché)
- Cuidado con la **seguridad**, evita usar **root** y vigila los puertos
- Cuidado con la **privacidad**, ¡todas las capas están disponibles!

Lab 4- Dockerfiles

Desplegando web estática

Desplegando aplicación python

Creando entorno desarrollo Data science

```
cd Lab4/webserver
```

```
docker build -t <dockerhub-user>/webserver .
```

Hacemos un build del servidor y el código

```
docker run -p 80:80 --rm <dockerhub-user>/webserver
```

Ahora podemos desplegar el servidor en cualquier parte

```
cd Lab4/python-useful
```

```
docker build -t <dockerhub-user>/python-useful .
```

Hacemos un build del código y su entorno

```
docker run --rm <dockerhub-user>/python-useful
```

Ahora nuestro código es portable

```
docker run --rm -e ARGS=”scratch” \
   <dockerhub-user>/python-useful
```

Hemos parametrizado la interfaz con variables de entorno

Entorno de trabajo para Data Scientist

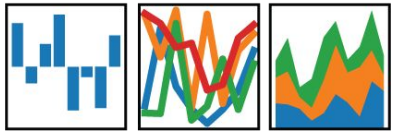
Requiere instalar dependencias python

Podemos instalarlo en local, no filosofía docker

NumPy

pandas

$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



matplotlib

```
cd Lab4/jupyter
```

```
docker build -t <dockerhub-user>/jupyter .
```

```
docker run --rm -p 8888:8888 \
   <dockerhub-user>/jupyter
```

Hemos creado un entorno portable

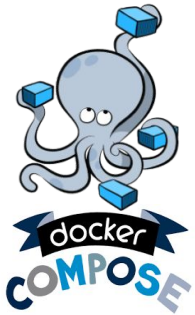
Podemos compartirlo con colaboradores

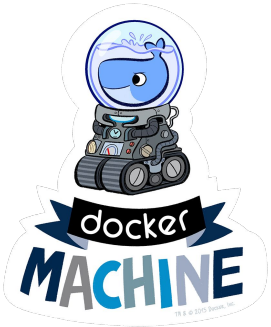
```
docker push <dockerhub-user>/jupyter
```

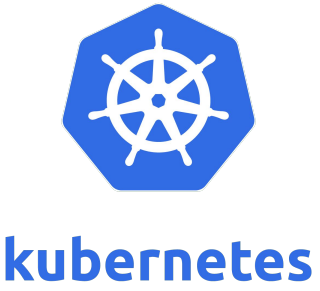
Compose

Ecosistema y herramientas

- Es una de las tecnologías de mayor y más rápida adopción en el mundo del desarrollo de software.
- Ha permitido un nuevo nivel de **desarrollo ágil** y es el soporte de la **computación en la nube**

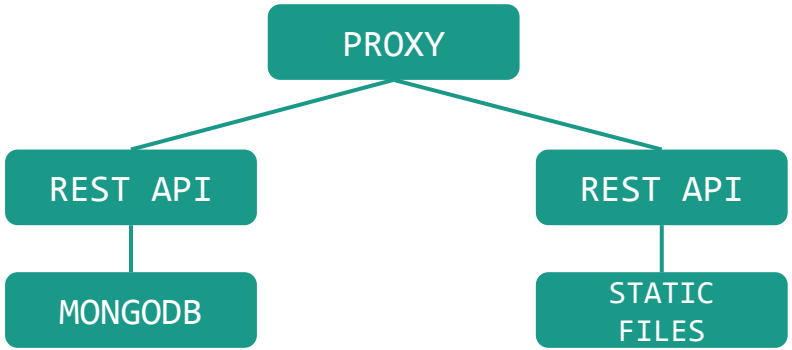






Caso de uso: Microservicios

- Elimina las restricciones de la infraestructura subyacente
- Agiliza la orquestación del sistema completo
- Independiza cada componente del sistema



Caso de uso: Microservicios

- Un fichero de configuración que nos permite definir contenedores a modo de servicios y sus recursos compartidos:


```yaml
services:
    webserver:
        image: ajgil/scratch-web
        ports: 80:80
        volumes:
          - app:/var/www/
    database:
        image: mysql/mysql
    environment:
        MYSQL_USER: usario
        MYSQL_PASSWORD: unbreakablepass
        MYSQL_DATABASE: scratch
```

```
docker-compose up
```

```
docker-compose down
```

```
docker-compose run
```

Lab 5- Docker-compose

Primera aplicación compose

API REST con MongoDB

Wordpress

```
cd Lab5/rest-tutorial/flask1
```

App.py es una aplicación que despliega una API REST en python

Para desplegar en modo host podemos instalar los requirements sobre un virtualenv

```
python3 -m venv venv
```

```
source venv/bin/activate
```

```
pip install -r requirements.txt
```

```
python app.py
```

```
cd Lab5/rest-tutorial/flask2-mongo
```

Esta versión utiliza mongodb

```
docker run --rm -name mongo -p 27017:27017 mongo
```

```
python app.py
```

Nuestra app se conecta por localhost a mongodb. ¿Y si queremos desplegar en docker?
En lugar de usar dos imágenes por separado, utilizamos docker-compose

```
cd Lab5/rest
```

Esta versión es una app docker-compose, toda la información está en el archivo *.yml

```
docker-compose build
```

```
docker-compose up
```

Desplegamos nuestros microservicios

Eliminamos los recursos creados

```
docker-compose down
```

```
cd Lab5/wordpress
```

```
docker-compose up
```

Desplegamos nuestros microservicios

Eliminamos los recursos creados

```
docker-compose down
```

Lab 6- Reto

Stream & Business Intelligence visual tool

¡Gracias!

Antonio Jesús Gil

ajgilp@etics.es