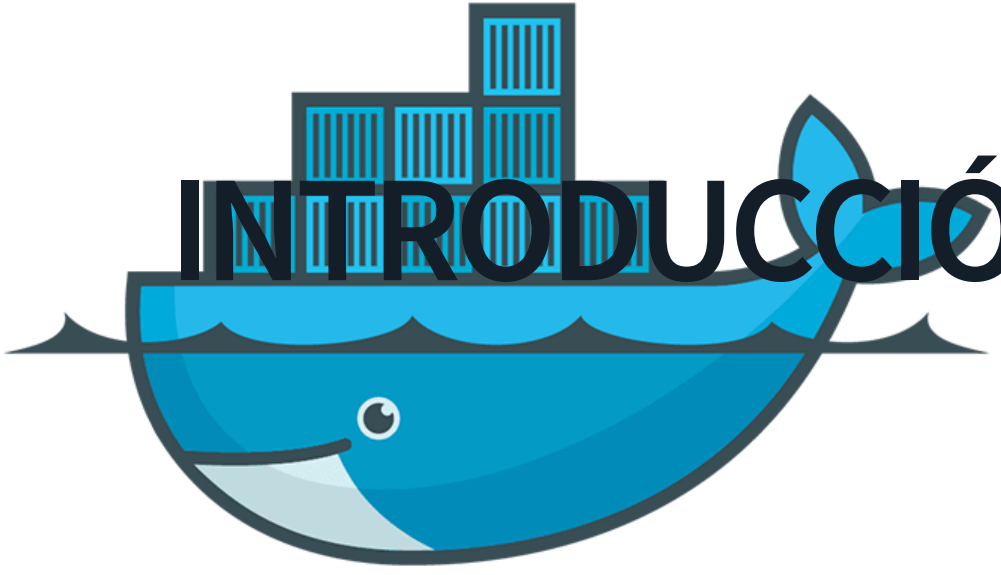


INTRODUCCIÓN A DOCKER

docker

QUIÉNES SOMOS

- Trabajamos en **Cosmomedia**
- Desarrollamos con Symfony desde 2010
- Twitter: @webcome y @canonale

CON LA COLABORACIÓN DE

¿Alguien conoce este botón?

LO QUE EL JEFE QUIERA, ES:

La adaptación y la rápidez

SISTEMAS ÁGILES

DOCKER TE AYUDA

- Alguien tiene un script para instalar automaticamente Wordpress
- Habéis tenido que llevar una aplicación rápidamente a otro servidor
- Habéis necesitado escalar un servicio.

COMO ENCONTRÉ DOCKER

COMO ENCONTRÉ DOCKER

- Equipos de desarrollo con la misma configuración.
- Tener distintas versiones de una app en el mismo servidor
- Desplegar rápidamente y en cualquier servidor cualquier **stack**

ANTES PROBAMOS

- Había probado con Virtualbox. Demasiados recursos
- Probamos con Vagrant. Demasiado manual
- Usamos Vagrant + Puppet. Complejo y tedioso.
- El stack se había hecho muy complejo

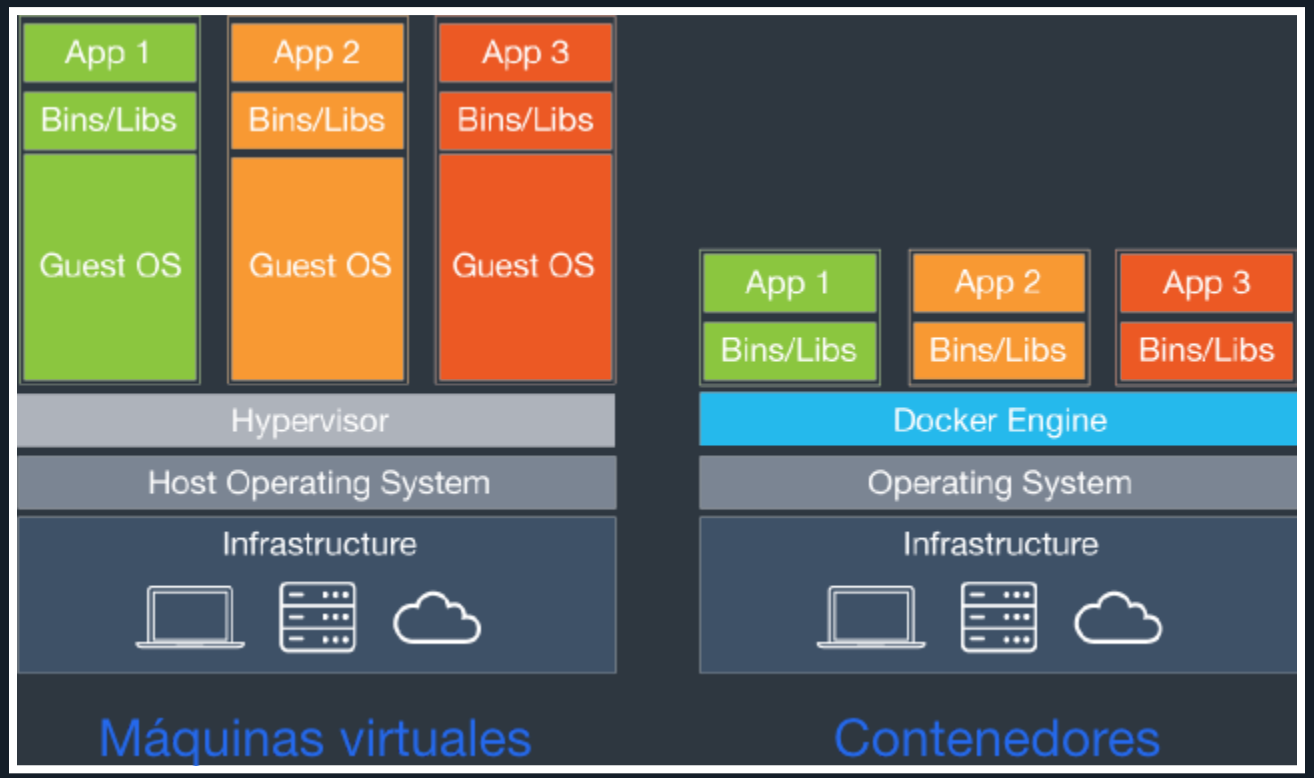
Y ENCONTRAMOS DOCKER

QUE ES DOCKER

Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries #@!#*

Docker website

DOCKER VS VM


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
Máquinas virtuales
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure
Contenedores

PERO QUE ES DOCKER

- Usa la tecnología **LXC** como sistema de virtuacización
- Se apoya en **cgroups**
- También usa **AUFS**, para el sistema de ficheros
- Y Chroot

REQUISITOS DOCKER

- Funciona Nativamente en Linux
- Se requiere de un kernel 3.10 o superior
- En Mac y Windows usa una máquina virtual.

INSTALAR DOCKER

Ejecutar la orden:

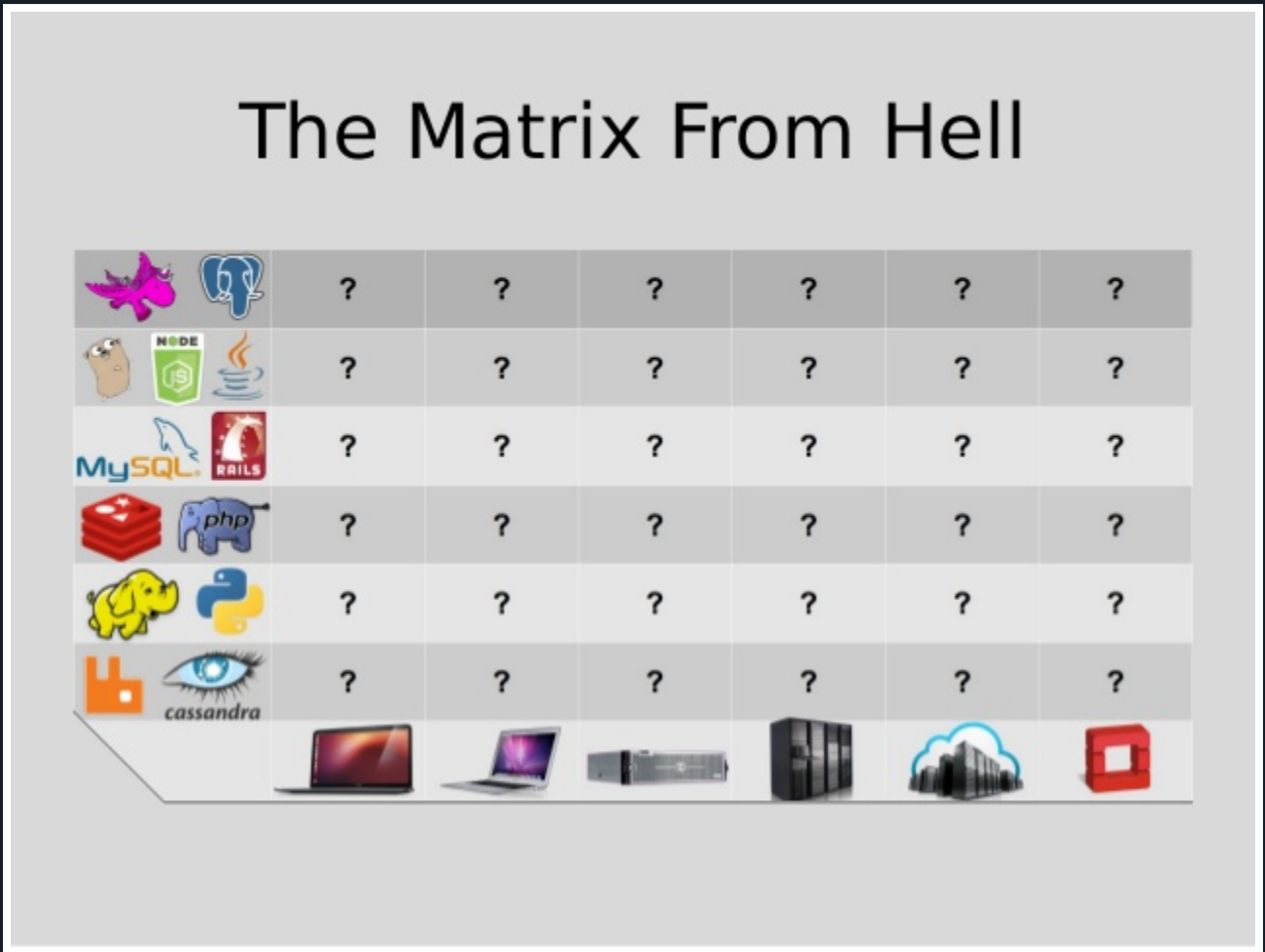
```
>_ curl -sSL https://get.docker.com/ | sh
```

Si no usas el usuario root:

```
>_ sudo groupadd docker
>_ sudo gpasswd -a ${USER} docker
>_ sudo service docker restart
```

PORQUE UTILIZAR DOCKER

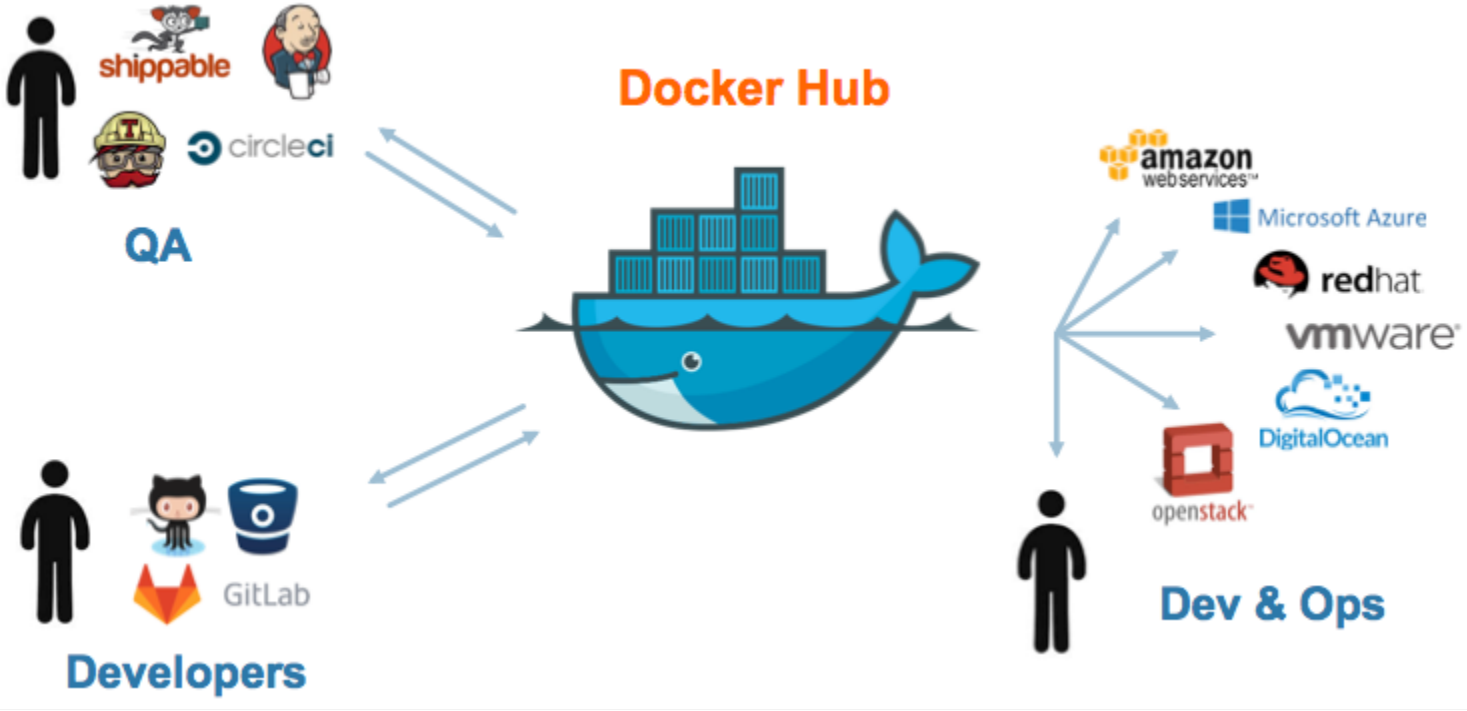
Nuestro stack funcionará en cualquier entorno, el despliegue es instantáneo y nos permite escalar servicios



PORQUE UTILIZAR DOCKER

Dispondremos de una autentica estructura orientada a los servicios.

TRABAJO CON CONTENEDORES



DOCKER: IMAGES

Listar las imágenes instaladas

```
>_ docker images
REPOSITORY  TAG     IMAGE ID      CREATED      VIRTUAL SIZE
python-bce  latest  e6c6b9f243b3  9 days ago   692.5 MB
node-gulp   latest  af8ac3d2b121  10 days ago  661.7 MB
puppet      v1      6fac737d4ff8  13 days ago  337.2 MB
```

DOCKER: PULL

Pull como en git sirve para bajar imagenes:

```
>_ docker pull httpd
```

Esto guarda localmente todas las capas de la imagen de apache.

DOCKER: PULL

A veces las imágenes tienen diferentes versiones de un mismo servicio.

Podemos bajar solamente la versión que no interese de una imagen, descartando capas que no nos valen

```
>_ docker pull httpd:2.2
```

DOCKER: RUN

El comando `run` ejecuta la imagen escogida. Toda imagen necesita ejecutar un comando y cuando ese comando termine la máquina se apaga.

```
>_ docker run -ti busybox echo "hola mundo"
hola mundo
```

DOCKER: RUN

Podemos mantener la maquina en segundo plano, siempre y cuando ejecuto un comando que no termina:

```
>_ docker run -d busybox tail -f /proc/swaps
97819261ac2
```

Generará una máquina con un nombre aleatorio y el id que se muestra

DOCKER: COMMIT

Docker no guarda persistencia en los cambios. Si queremos modificar una imagen y guardar esos cambios tenemos que hacer un commit, con estos pasos

1. Creamos la imagen

```
>_ docker run -ti busybox touch myfile
```

2. Cogemos el ID de la ultima máquina creada

```
>_ docker ps -lq
b8679bcd41dc
```

3. Hacemos el commit

```
>_ docker commit b8679bcd41dc busybox:2
```

4. Verificamos la disposición

```
>_ docker images
  REPOSITORY   TAG      IMAGE ID       CREATED          VIRTUA
  busybox:2    latest   4e4d7142d4cf   20 seconds ago   1.109
```

ORDENES DE CONTROL DE DOCKER

Maquinas en funcionamiento

```
>_ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED
6ab5c7de440a   node-gulp:latest     "/entrypoint.sh gulp "   1 week

STATUS         PORTS      NAMES
Up 4 days      3000/tcp   frontend_dockregistry_1
```

Parar una máquina

```
>_ docker stop id-maquina/nombre-maquina
```

Arrancar una máquina **parada**

```
>_ docker start id-maquina/nombre-maquina
```

Eliminar una máquina. (Parada)

```
>_ docker rm id-maquina/nombre-maquina
```

DOCKER PUSH

Guardar y distribuir nuestras imagenes ya creadas.

Las imagenes en **docker.com son siempre públicas**

IMAGENES PRIVADAS

Hay servicios que dan la posibilidad de tener imágenes privadas

- 

DonDocker.com
- hub.docker.com
- quay.io

DOCKER PUSH

El proceso tiene 4 pasos:

1. Loguearnos en el servicio (DonDocker)
2. Creamos el contenedor.
3. Hacemos un commit de la imagen
4. Hacemos el push de la imagen

Tutorial de DonDocker

NUESTRO PRIMER DOCKER. SERVIDOR APACHE

Vamos a buscar la imagen a hub.docker.com. *(Se recomienda usar las imágenes oficiales)* Bajamos la imagen **httpd**

```
>_ docker pull httpd:2.2
```

Tras tener la imagen, vamos a desplegar la máquina con el código que tenemos en nuestra carpeta `~/app/` así:

```
>_ docker run -d --name apache -p 8888:80 \
-v ~/app:/usr/local/apache2/htdocs/ httpd:2.2
```

Ahora podemos abrir la aplicación:

```
>_ xdg-open http://localhost:8888/
```

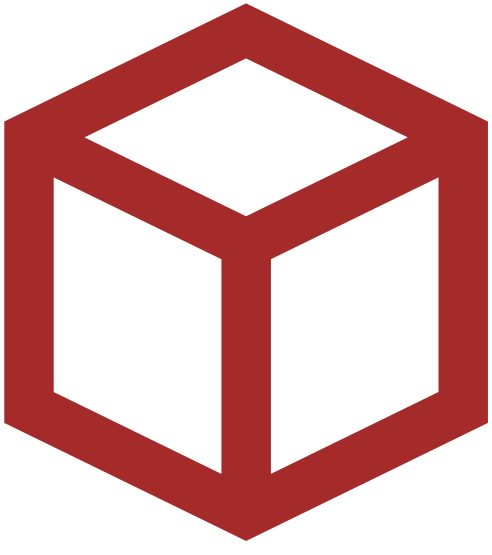
NUESTRO PRIMER DOCKER. SERVIDOR APACHE

1. Hemos creado una máquina en background con `-d`
2. Hemos mapeado el puerto 8888 con el 80 con `-p`
3. Compartimos la carpeta `app` con `-v`

DOCKERFILE

Dockerfile

build



Image

DOCKERFILE

Es un fichero con instrucciones para crear máquinas partiendo desde una imagen. Podemos modificar una existente

DOCKERFILE: FROM, ENV, RUN

- *FROM* es la orden que indica la base sobre la que partimos

```
FROM debian:jessie
```

- *ENV* se usa para generar variables de entorno que luego estarán disponibles

```
ENV PHP_VERSION 5.6.2
  ENV PATH $PATH:$HTTPD_PREFIX/bin
```

- *RUN* es cualquier orden que ejecutaríamos en la consola. Para que en la máquina conste menos pasos se suelen agrupar todas las ordenes

```
RUN apt-get update
  #UNA SOLA ORDEN
  RUN apt-get update \
    && apt-get install -y \
      libapr1 \
      libaprutil1 \
      libapr1-dev \
      libaprutil1-dev \
      libpcre++0 \
      libssl1.0.0 \
    && rm -r /var/lib/apt/lists/*
```

DOCKERFILE: WORKDIR, ADD, COPY, EXPOSE, VOLUME

- *COPY* sirve para añadir scripts o ficheros a la imagen durante la creación

```
COPY httpd-foreground /usr/local/bin/
```

- *ADD* es similar a copy pero puede interpretar cualquier uri

```
ADD http://gits.github.com/saujhf22 /local/
```

- *WORKDIR* es la ruta base sobre la que se ejecutará todo

```
WORKDIR /local/
```

- *EXPOSE* sirve para dar acceso a un puerto o rango de puertos

```
EXPOSE 80
```

- *VOLUME* se usa para dar persistencia a los datos, suele darse a carpetas que contienen la aplicación a ejecutar o los datos da la base de datos.

```
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]
```

DOCKERFILE: CMD, ENTRYPOINT

- *CMD* este comando se ejecuta cuando se arranca la máquina. Suele usarse para ejecutar servicios que funcionan constantemente como apache

```
CMD ["httpd-foreground"]
```

- *ENTRYPOINT* se ejecuta cada vez que se ejecuta la máquina. Suele usarse para preparar la máquina o descargar librerias

```
ENTRYPOINT bower install -g
```

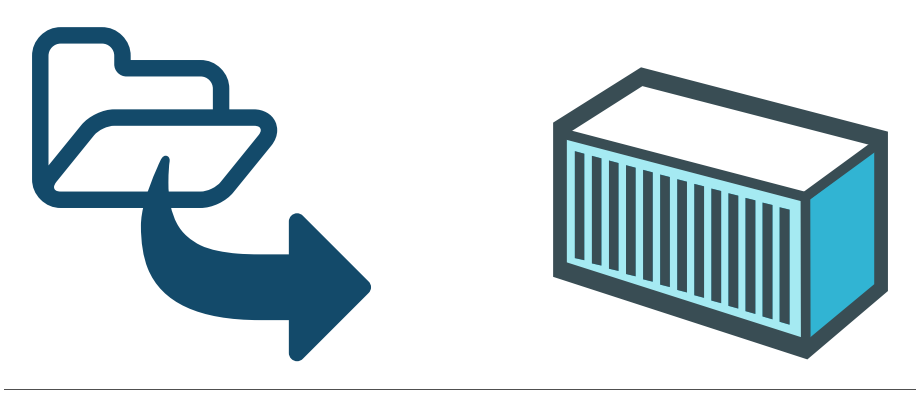
CONSTRUIR LA MÁQUINA:

```
>_ docker build -t my-app .
```

Verificamos que la máquina funciona

```
>_ docker run --name alpache -d -p 80:80 my-app
59a4a65442c92c3856 #id-maquina
>_ docker ps
```

COMPARTIR CARPETAS



Resulta de mucha utilidad para cuando estamos desarrollando

Se pueden montar varias carpetas incluso ficheros individuales

```
>_ docker run -v $PWD/app:/var/www/html httpd:2
>_ docker run --volume $PWD/app:/var/www/html httpd:2
```

Es necesario usar rutas absolutas, por eso se usa **$PWD**

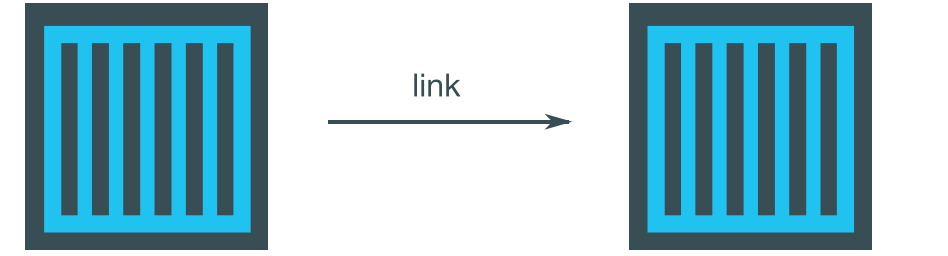
MONTAR VOLUMES DE OTRO DOCKER

```
>_ docker run --volumes-from mysql httpd:2
```

No se necesitan que otras máquinas estén ejecutandose

Montará aquellas carpetas definidas como volumenes, bien en el Dockerfile al crearse o al errancar la máquina

STACK WORDPRESS



MySQL

IP: 172.17.1.9
Puerto: 3306/tcp
Name: Mysql
Volume: /var/lib/mysql

Wordpress

IP: 172.17.1.10
Puerto: 80/tcp
Name: Wordpress
Volume: /var/www/html

PÁSO PARA CREAR EL STACK

1. Definir variables de la base de datos.
2. Definir variables de la aplicación
3. Arrancar la máquina con MySQL
4. Arrancar Wordpress y enlazar a la base de datos

VARIABLES DE MYSQL

Nombre de la base de datos

El usuario que se va a conectar

El password del usuario

Hay otras variables que se pueden definir

VARIABLES DE WORDPRESS

Definimos lo que tiene que ver con la base de datos.

Usuario

Contraseña

Nombre de la base de datos

Prefijo de la base de datos

Opciones de autenticación

ARRANCAMOS MYSQL

```
>_ docker run --name mysql --env-file .vars -d mysql
```

ARRANCAMOS WORDPRESS

```
>_ docker run --name wordpress --link mysql:mysql \
   --env-file .vars \
   -p 88:80 -d wordpress
```

FICHERO .VARS

```
#MYSQL
MYSQL_ROOT_PASSWORD=123456
MYSQL_DATABASE=database
MYSQL_USER=usuario
MYSQL_PASSWORD=password
#WORDPRESS
WORDPRESS_DB_HOST=mysql
WORDPRESS_DB_USER=usuario
WORDPRESS_DB_PASSWORD=password
WORDPRESS_DB_NAME=database
WORDPRESS_TABLE_PREFIX=clk_
```

INSPECT

Herramienta que nos proporciona información acerca de las máquinas en funcionamiento. Ejemplo:

```
>_ docker inspect my_machine
```

Devuelve un JSON con datos de la máquina: IP, imagen, volumenes, puertos...

INSPECT


```
[{
  "Config": {
   "Cmd": ["mysqld"],
   "Entrypoint": ["/entrypoint.sh"],
   "Env": [],
   "ExposedPorts": {},
   "Image": "mysql:latest",
   "Volumes": {
    "/var/lib/mysql": {}
   },
   "WorkingDir": ""
  },
  "Id": "5d37f187b2ce57....",
  "Image": "9eefddb060bc647...",
  "Name": "/piwik_db_1",
  "NetworkSettings": {
```

INSPECT: FILTROS

Como *inspect* devuelve muchos datos, la opción *-f* nos permite filtrar como lo harías en un JSON. Para saber la IPv4 de una máquina:

```
>_ docker inspect -f '{{.NetworkSettings.IPAddress}}' my_machine
```

El resultado de inspect -f no es json para ello tenemos que poner el modificador delante del filtro: -f '{{json .Config.Env}}'

HERRAMIENTAS PARA DOCKER

DOCKER-COMPOSE

- Se usa para controlar un stack entero
- Toda la configuración en un solo fichero **docker-compose.yml**
- Facilita el escalado de las máquinas

DOCKER-COMPOSE DE WORDPRESS

```
wordpress:
  image: wordpress
  links:
    - mysql:mysql
  env_file:
    - ./.vars
  ports:
    - "88:80"
mysql:
  image: mysql
  env_file:
    - ./.vars
```

ORDENES BÁSICAS DE DOCKER-COMPOSE

Inicializar el stack:

```
>_ docker-compose up -d
```

Parar el stack:

```
>_ docker-compose stop
```

Volver a arrancar

```
>_ docker-compose start
```

DOCKER-MACHINE

Gestor de maquinas virtuales que nos abstrae del tipo de máquina y que instala lo imprescindible para usar docker. Para sistemas operativos no linux y cuando no quieres guarrear tu máquina.

ORDENES BÁSICAS

Ver las máquinas creadas.

```
>_ docker-machine ls
```

Arrancar una máquina.

```
>_ docker-machine start [nombre-maquina]
```

Parar el stack:

```
>_ docker-compose stop [nombre-maquina]
```

CREAR UNA MÁQUINA

docker-machine permite crear una máquina virtual local o en diferentes plataformas

```
>_ docker-machine create \
   --generic-ip-address 192.168.99.111 \
   --virtualbox-boot2docker-url  http://192.168.99.100:4444/boot2doc
   --virtualbox-cpu-count "1" \
   --virtualbox-disk-size "20000" \
   --virtualbox-memory "1024" \
   -d  virtualbox \
   mi-maquina
```

Conectar con la máquina sin usar ssh

```
>_ eval $(docker-machine env mi-maquina)
```


FIN
AWESOME!

¿PREGUNTAS?

j.mp/ENCUESTADOCKER