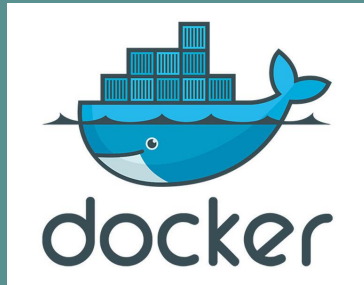


# Docker para desarrolladores



# Previo

- Docker instalado <https://docs.docker.com/install/>
- Repo clonado <https://github.com/mariogalan/AppCoin>



# Presentación

Mario Galán Alonso

Desarrollador, emprendedor (monitorii, brab.io),  
gestor...





# ¿Qué vamos a ver?

1. Introducción a Docker
2. Manejo básico
3. Dockerizar la aplicación Appcoin (3 alternativas)

# Introducción a Docker



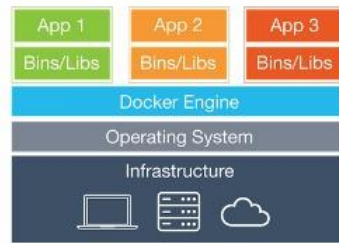


# ¿Qué es Docker?

- **Virtualización ligera** basada en Linux (sutiles diferencias en Windows y Mac)
- Herramienta útil tanto para el **desarrollador**, **testing** como para **sistemas**.
- **Licencia libre** (Apache License 2.0)



Virtual Machines



Containers



# Un poco de historia

- Basado en **chroot** y Linux containers (**LXC**).
- Ideas similares existen en otros SSOO como **Solaris Zones** o **FreeBSD Jails**.
- Docker se comienza a desarrollar en 2008 pero se lanza como software libre en **2013**.
- **Production ready** pero en constante actualización.
- Respaldado por **empresas** como Red Hat, IBM, Google, Cisco y Microsoft.

# Imagen

Paquete que incluye todo lo necesario para ejecutar una aplicación (código, binarios, librerías y variables de entorno)





# ¿De dónde salen las imágenes?

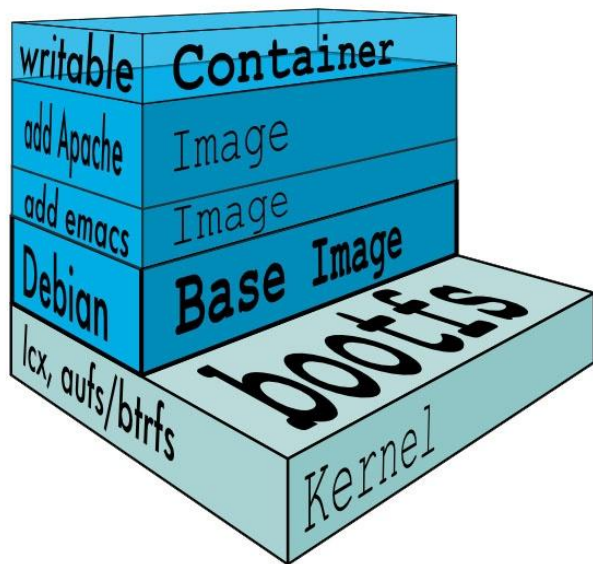
- **Docker hub** u otro registry
- Creación propia con **Dockerfile**
- **Importación** de un tar.gz

# Contenedor

Un contenedor es una **instancia** de una imagen que se encuentra en ejecución



# Arquitectura de capas





# **Docker para desarrolladores**





# ¿Cómo ayuda Docker a los desarrolladores?

- Entornos **iguales** y **repetibles**
- Configuración en **código**
- Mantener la máquina **limpia**
- **Testing** en distintas versiones
- **Unificar** entornos en equipos de desarrollo
- Simplifica **probar** software

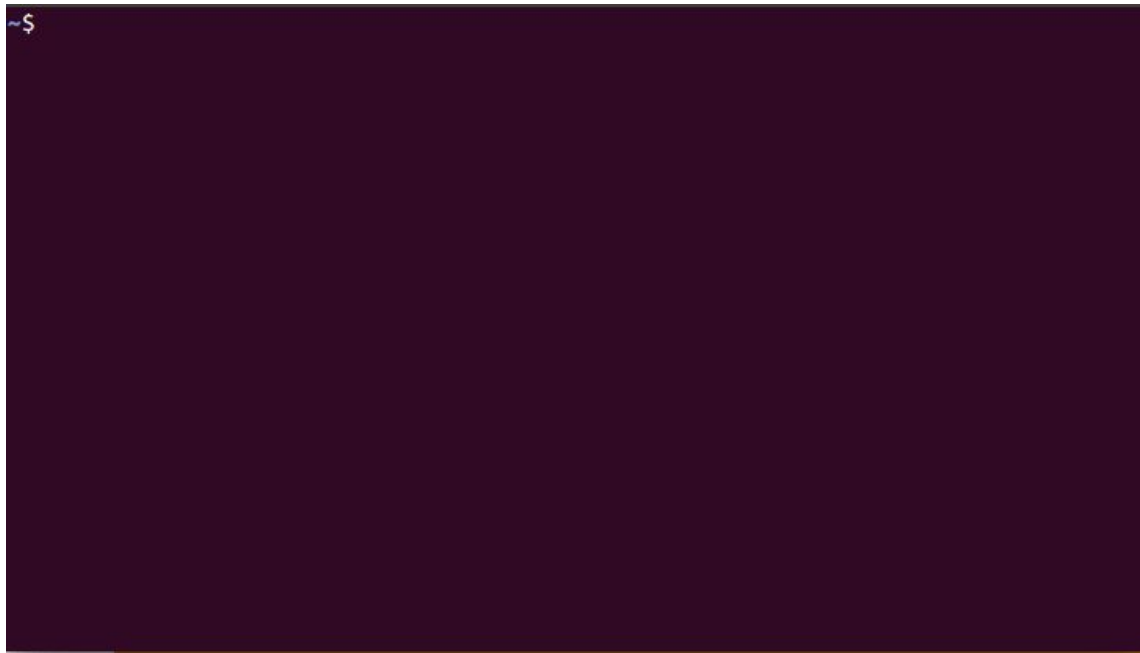


**Vamos al lío**





**\$ docker run hello-world**





# Comandos útiles I

Crea un contenedor a partir de una imagen

```
$ docker run nombre-imagen [comando]
```

```
$ docker container run nombre-imagen [comando]
```

Listar las imágenes

```
$ docker image ls
```

```
$ docker images
```

```
~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	f2a91732366c	3 months ago	1.85kB





# Comandos útiles II

Listar contenedor en funcionamiento

```
$ docker container ls
```

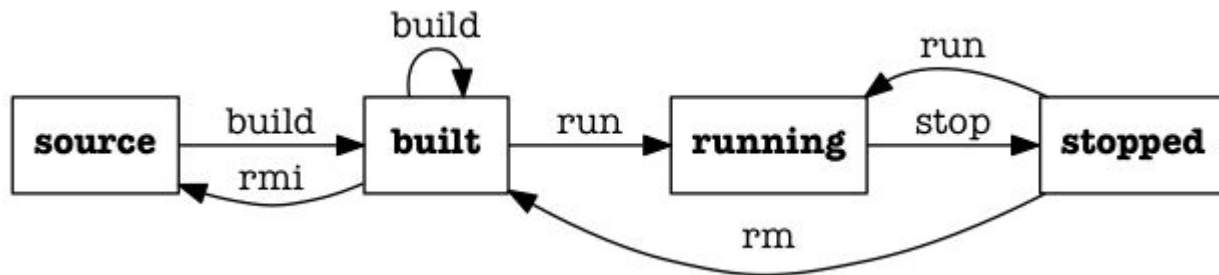
Lista todos los contenedores

```
$ docker container ls -a
```

Eliminar un contenedor

```
$ docker container rm id|nombre
```

# Estados de un contenedor





# Ejemplo de contenedores

Ubuntu

```
$ docker run -it --name my-ubuntu ubuntu:latest bash
```

Centos

```
$ docker run -it --name my-centos centos:latest bash
```

Drupal

```
$ docker run --rm -p 8000:80 drupal
```



# Login y logout de un contenedor

Ejecutar un shell en un contenedor en marcha:

```
$ docker exec -it ID|nombre bash
```

Salir y terminar el contenedor

Escribir “exit” o Control + D

Salir sin terminar un contenedor

Control + P + Q



# Ejecución en una sola línea

Última versión de PHP:

```
$ docker run -it --rm --name my-running-script -v "$PWD":/src -w /src php:cli php hello.php
```

Con la versión 5.6:

```
$ docker run -it --rm --name my-running-script -v "$PWD":/src -w /src php:5.6-cli php hello.php
```

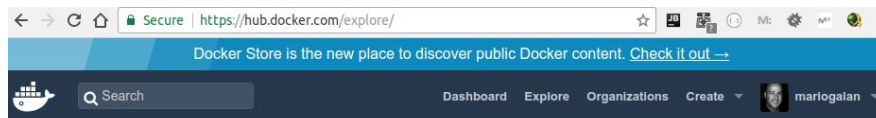
Composer:

```
$ docker run --rm -it --volume $PWD:/app -u $UID composer install
```









# Buscando imágenes en Docker Hub

```
$ docker search imagen-a-buscar
```



## Explore Official Repositories

 <b>alpine</b> official	3.3K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS
 <b>nginx</b> official	8.1K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS
 <b>httpd</b> official	1.6K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS
 <b>redis</b> official	4.9K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS
 <b>busybox</b> official	1.2K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS
 <b>ubuntu</b> official	7.4K STARS	10M+ PULLS	<a href="#">➤</a> DETAILS



# **Networks - Comunicación por red**





# A grandes rasgos

- Una aplicación desplegada en Docker **no tiene conciencia** que está dentro de un contenedor.
- Tampoco sabe si los servicios de los que depende están dockerizados o no.
- Docker **asigna una IP** a cada contenedor que levantamos. Podemos ver la IP de un contenedor (y mucha más información) con:  
\$ docker container inspect ID|nombre





# Network drivers

- Bridge:
  - Driver **por defecto**. Se usa para aplicaciones que funcionan en contenedores standalone y necesitan comunicarse.
  - Bridge es para contenedores funcionando en el **mismo host**
  - No recomendado para **producción**
- Host:
  - Para contenedores standalone pero se usa la red del host directamente.
- Overlay, macvlan, none, etc...



# Publicando un puerto en el host

- Lanzar un contenedor y publicar puertos en el host (Ej):  
`$ docker run -d --name my-apache-app -p 9090:80 httpd:2.4`
- Es necesario cuando utilizamos Docker en Windows o Mac.



# Comunicando contenedores entre sí

- Por defecto, los contenedores se pueden comunicar a través de la IP.
- Pero las IPs van cambiando al lanzar los contenedores.
- Para comunicar contenedores usando hostnames debemos utilizar el parámetro `--link` (deprecated).
- Por ejemplo:

```
$ docker run -d -e POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo --name db postgres:9.4
$ docker run -v /path/to/config:/etc/odoo -p 8069:8069 --name odoo --link db:db -t odoo
```



# **Persistiendo información en Docker**



**Los contenedores deberían  
ser efímeros e  
intercambiables**

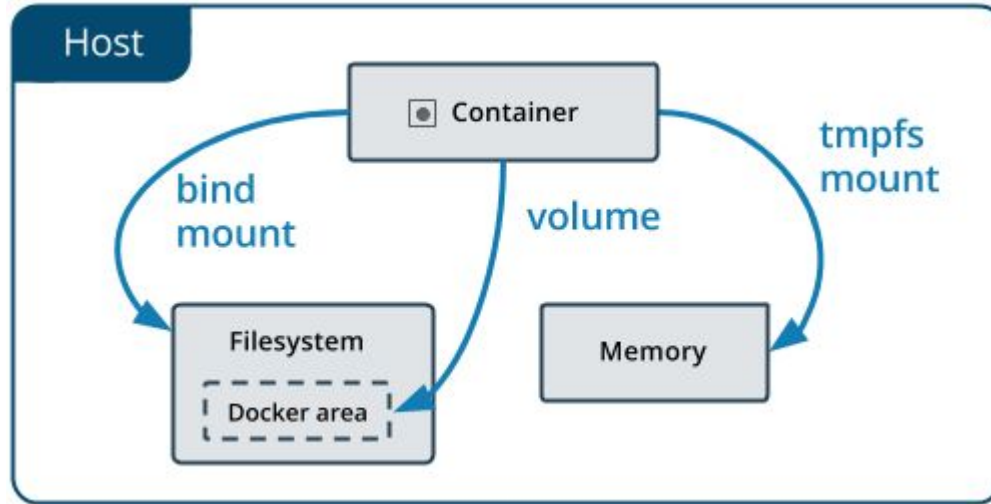




# Capa escribible en el contenedor

- Ventajas
  - No hay nada que configurar
- Inconvenientes
  - Difícil acceder a la información si el contenedor no está levantado
  - Los datos están ligados al host donde se ejecuta el contenedor. Complicado moverlos a otro entorno.
  - Menor rendimiento que otras alternativas
- Casos de uso
  - Pruebas a contenedores que no se vayan a compartir

# Información fuera del contenedor





# tmpfs

- Ventajas
  - Máxima seguridad. Los datos no se escriben en disco.
- Inconvenientes
  - Sólo utilizable para información efímera..
  - No se puede manejar desde el docker CLI.
- Casos de uso
  - Cuando por seguridad o rendimiento no se quiera guardar información a disco





# Bind mounts

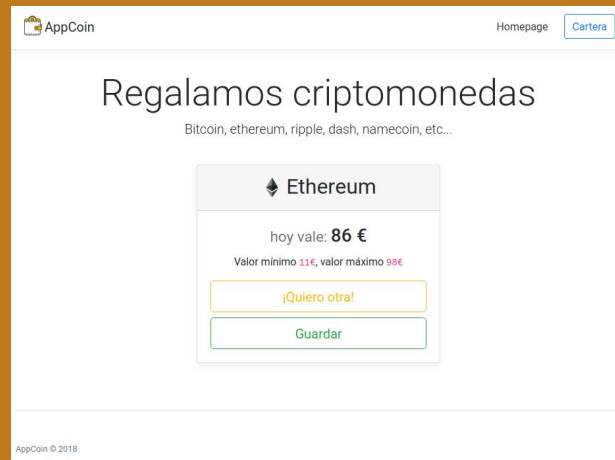
- Uso (ej: `docker run -v dir-local:dir-contenedor ...`)
- Ventajas
  - Buen rendimiento
  - Se puede modificar la información desde fuera del contenedor
- Inconvenientes
  - Se puede modificar la información desde fuera del contenedor
  - Los datos están ligados al host donde se ejecuta el contenedor. Complicado moverlos a otro entorno.
  - No se puede manejar desde el docker CLI.
  - Problemas en Windows
- Casos de uso
  - Compartir ficheros de configuración
  - Compartir el código fuente o artefactos entre el contenedor y el host
  - Cuando la estructura de directorios en el host es conocida y consistente con lo que espera el contenedor



# Volumes

- Uso (docker volume ...)
- Ventajas
  - Se pueden manejar desde el docker CLI.
  - La información no se puede modificar desde el host.
  - Fácil de compartir entre contenedores
  - Fácil de mover entre entornos
- Inconvenientes
  - Un poco más engorroso de usar que bind
- Casos de uso
  - BBDD
  - Logs
  - Más utilizado en entornos de producción que en desarrollo

# Dockerizando AppCoin: Tres opciones





# Requisitos de Appcoin

- PHP 7.1.3+ con las extensiones DOM, Zip y MySQL.
- Servidor web
- Servidor MySQL



**Primera opción:  
Una sólo imagen y un  
sólo contenedor**





**Creando una imagen**



# Primera opción (docker commit)

- Creamos un contenedor
- Nos metemos dentro y hacemos las modificaciones
- Salimos y utilizamos docker commit para crear una imagen con nuestras modificaciones
- `$ docker container commit CONTAINER [REPOSITORY:TAG]`



## Segunda y mejor opción Dockerfile

- El Dockerfile son las instrucciones para la creación de una imagen.
- Una imagen se genera a partir del Dockerfile y del contexto en el que se encuentra.
- Las instrucciones se ejecutan de manera secuencial e independiente (ojo con `RUN cd /tmp`).





# Buenas prácticas con Dockerfile

- Un contenedor debe ser efímero
- No instalar paquetes innecesarios
- Un contenedor para cada función (pero con tranquilidad)
- Ordenar los argumentos y separar los comandos con '\'.



# Probando la alt1

- Generando la imagen:  
`$ docker build -f ./docker/alt1/Dockerfile -t mariogalan/alt1 .`
- Ejecutando:  
`$ docker run -it -p 8000:8000 mariogalan/alt1`
- La base de datos habría que configurarla de forma independiente.



# Compartiendo las imágenes en Docker Hub





# Compartiendo la imagen en Docker Hub

- `$ docker login`
- `$ docker tag image username/repository:tag`
- `$ docker push username/repository:tag`



# **Segunda opción**

## **Múltiples imágenes y contenedores**

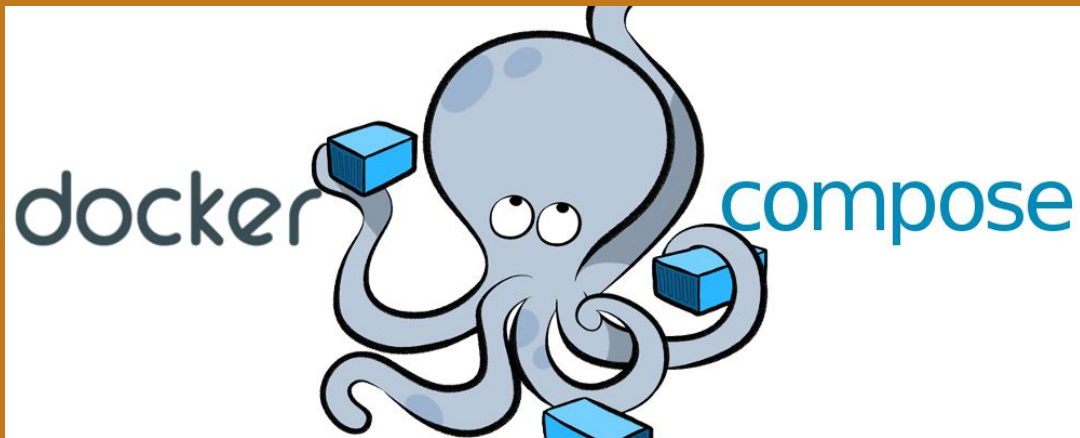




# Pensado en la arquitectura

- Servidor web: Nginx
- Intérprete script: PHP
- Base de datos: Mysql

# Docker compose







# Utilizando el alt2

- Levantamos los servicios definidos en docker-compose.yml:  
`$ docker-compose up`
- Generamos el esquema de la BBDD:  
`$ docker exec -it appcoin_php bin/console doctrine:schema:create`





# **Tercera opción: Swarms, services y stacks**





# Docker Services





# Docker Services

- En una aplicación distribuida, las distintas partes son conocidas como servicios.
- En Docker un servicio es un contenedor en producción.
- Los servicios se manejan con:  
\$docker service comando
- Ejemplo:  

```
$ docker service create --name redis --replicas=5 redis:3.0.6
```



# Docker Swarms



# Docker Swarms

- Un swarm (enjambre) es un conjunto de 1 o más nodos que están ejecutando el Docker engine y se han añadido a un cluster.
- Crearemos un swarm en local con:  
\$ docker swarm init



# Docker stacks





# Docker stacks

- Equivalente a docker-compose pero integrado
- Opción de futuro
- Compatible con docker-compose v3



## **Fig -> docker compose -> docker stacks**

- Docker stack es la nueva forma de hacer despliegues multicontenedor en Swarms
- Docker compose es probable que sea discontinuado en el futuro
- Docker stacks sólo soporta docker-compose.yml en version 3. Docker compose soporta 2 y 3.
- No toda la funcionalidad de Docker compose está integrada todavía en Docker stacks.





# Ejecutando la alt3

- Alta  
\$ docker swarm init  
\$ docker stack deploy -c docker-stack.yml appcoin
- Actualización  
\$ docker stack deploy -c docker-stack.yml appcoin
- Baja  
\$ docker stack rm appcoin  
\$ docker swarm leave --force





# Contacto

Mario Galán Alonso

[mario.galan@gmail.com](mailto:mario.galan@gmail.com)

<http://mariogalan.com>

[@poquitopaquito](#)

<https://www.linkedin.com/in/mariogalanalonso/>

