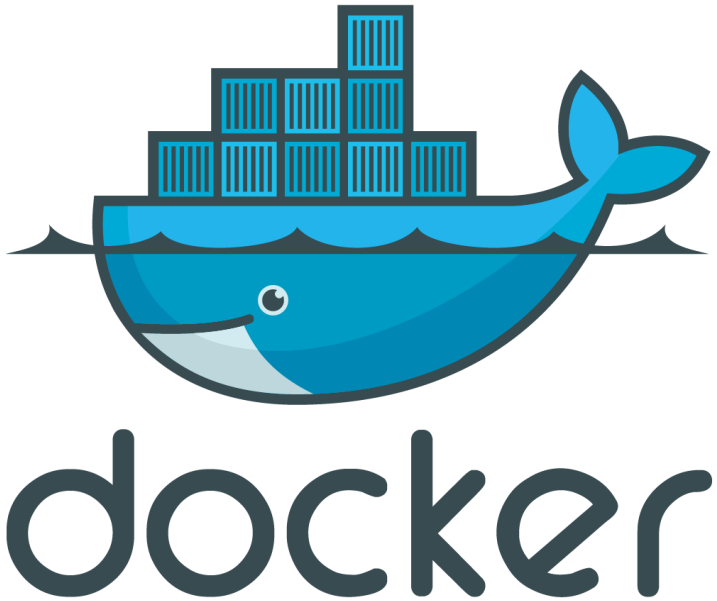
DOCKER - Parte 1



¿Qué es Docker?

Algunas definiciones:

1. Un entorno chroot (jaula de dependencias)
2. Contrato entre *sysadmin* y desarrollador
3. Empaquetador de aplicaciones
4. Un sistema de virtualización

¿Qué es Docker?

1. Un entorno *chroot*

Entorno aislado del sistema, donde se pueden instalar aplicaciones y librerías sin que afecte al sistema principal.

2. Contrato entre *sysadmin* y desarrollador

El administrador solo debe **desplegar** los contenedores, mientras que el desarrollador puede trabajar e instalar con ellos sin poner en riesgo el sistema.

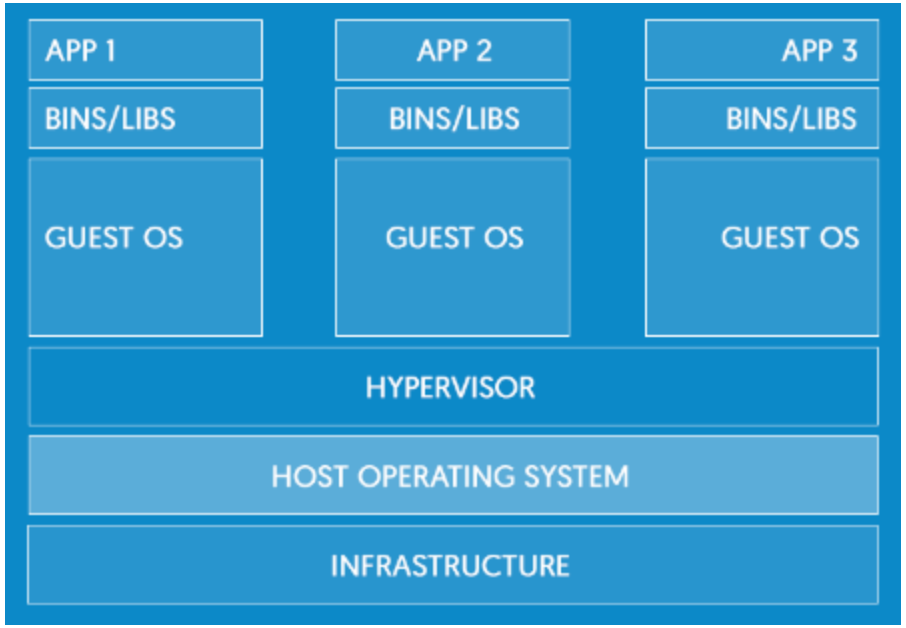
3. Empaquetador de aplicaciones

Se puede crear un *container* para la aplicación, de modo que se ejecuten igual en distintas máquinas.

¿Qué es Docker?

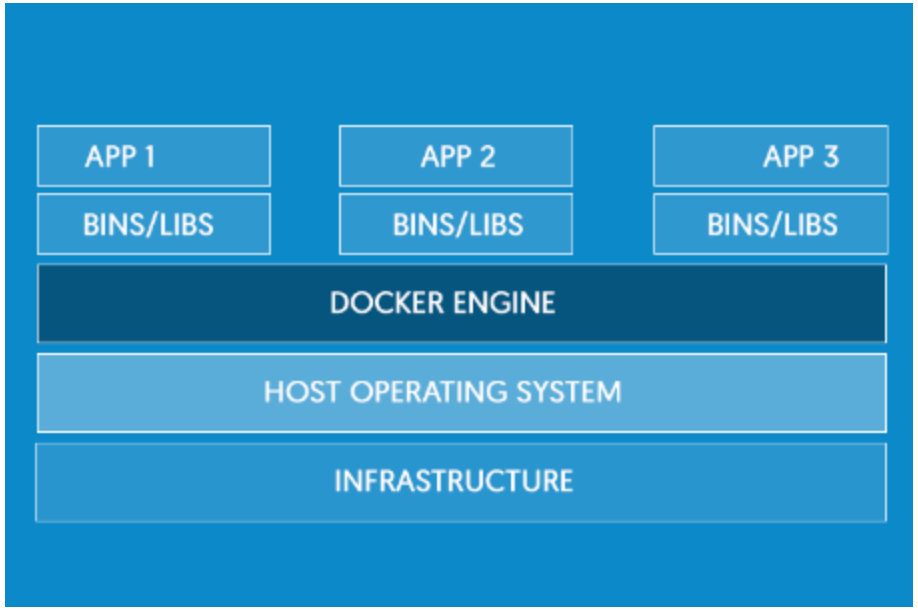
4. Un sistema de virtualización

Virtual machine: *Incluye la aplicación, las librerías y archivos necesarios, y un sistema operativo completo, lo que puede llegar a ocupar varios GBs.*



Virtualización

Contenedor: *Incluye la aplicación y todas sus dependencias, pero comparte el kernel con otros contenedores, ejecutándose como procesos aislados en el espacio de usuario del sistema operativo anfitrión. Los contenedores de Docker no están atados a ninguna infraestructura específica: se ejecutan en cualquier ordenador, en cualquier infraestructura y en cualquier nube.*



Virtualización de Docker

- Docker no solo virtualiza el hardware, también el sistema operativo.
- Docker es una tecnología de *código abierto* para crear, ejecutar, probar e implementar aplicaciones distribuidas dentro de **contenedores** de software
- Docker está basado en el sistema de virtualización de **Linux**

Contenedores Docker

¿Qué es un contenedor?

- Los contenedores crean un **entorno virtual** para las aplicaciones
- Ocupan menos **espacio** que una máquina virtual al no tener que almacenar el sistema completo.
- El tiempo de **arranque** de un container es inferior a 1 segundo.
- Para **integrar** máquinas virtuales en un host, debemos establecer la red. En los contenedores de Docker la integración es directa.

Ventajas para desarrolladores

- **Dependencias**: Docker permite a los desarrolladores entregar servicios aislados, gracias a la eliminación de problemas de dependencias de ejecución del software.
- **Productividad**: Se reduce el tiempo empleado en configurar los entornos o solucionar problemas relacionados con los mismos.
- **Despliegue**: Las aplicaciones compatibles con Docker se pueden desplegar desde equipos de desarrollo a versiones de producción.

Instalación de Docker

- Trabajaremos con Ubuntu, aunque es posible en cualquier sistema operativo
- Docker solo trabaja con sistemas de 64 bits

Instalación (Ubuntu)

Instrucciones de instalación:

https://docs.docker.com/engine/install/ubuntu/

Comprobar que se está ejecutando:

```
sudo docker --version
sudo service docker status
```

Instalación (Windows)

- Docker está ya integrado en Windows 10, haciendo uso de Hyper-V
- Cuando se usa Docker no se puede utilizar **Virtual Box**, lo que puede significar un problema
- Tenemos la opción de deshabilitar Hyper-V. Para ello debemos ejecutar como administrador:

Apagar

```
bcdedit /set  hypervisorlaunchtype off
```

Encender

```
bcdedit /set hypervisorlaunchtype auto
```

Usos sin necesidad de *sudo* (recomendable)

Para ejecutar el daemon de Docker y los contenedores sin ser superusuarios:

https://docs.docker.com/engine/security/rootless/

Creamos un grupo docker

```
sudo groupadd docker
```

Añadimos el usuario al grupo

```
sudo usermod -aG docker $USER
```

Cerramos sesión y volvemos a entrar

Comprobamos que se está ejecutando

```
sudo service docker status
```

Comprobar la instalación

Vemos con qué versión estamos trabajando

```
docker --version
```

Lista de comandos

```
docker
```

Información del sistema

```
docker info
```

Descarga y ejecución de un contenedor básico

```
docker run hello-world
```

Docker Hub

https://hub.docker.com/

Es un repositorio donde los usuarios de Docker pueden compartir las imágenes de los contenedores que han creado.

Existe una opción de pago para registro privado.

Tiene servicios automatizados webhooks y se puede integrar con Github y BitBucket.

Docker Hub

Darse de alta como usuario

https://hub.docker.com/

Iniciar sesión desde el terminal

`docker login`

Ver los repositorios locales descargados

`docker images` / `docker image ls`

> Tendremos el hello-world que hemos utilizado para comprobar que el servicio funcionaba

Descargando de repositorios

Buscar un repositorio

```
docker search ubuntu
```

Descargar la versión oficial

```
docker pull ubuntu
```

> Podemos usar opciones como `ubuntu:14.04`, `ubuntu:latest`, etc.

Ejecutamos el contenedor

> `run [nombre_imagen] [comando]`

```
docker run ubuntu /bin/echo "Pues parece que funciona"
```

> Al ejecutar `run`, si la imagen no existiera en el repositorio local, se descarga antes

Repositorios en ejecución

Ver los contenedores en ejecución

```
docker ps
```

> `CONTAINER ID` es el identificador del contenedor
> `IMAGE` es la imagen usada para crearlo
> `NAME` -- si no se especifica, Docker crea un nombre aleatorio

Ver los contenedores que se han creado

```
docker ps -a
```

Ver el último contenedor creado

```
docker ps -l
```

Borrar una imagen

```
docker rmi [nombre_imagen]
```

Trabajando en los contenedores

Poner nombre a un contenedor

```
docker run -t -i --name myUbuntu ubuntu /bin/bash
```

`-t` incluye terminal dentro del contenedor

`-i` se puede trabajar de manera interactiva

Se pueden poner unidos y/o en diferente orden: `-it`

Para salir del terminal

```
exit
```

Trabajando en los contenedores en ejecución

```
docker run --name myUbuntu ubuntu /bin/bash \
    -c "while true; do echo hola mundo; sleep 1; done"
```

> `-c` ejecuta un comando en el contenedor

Vemos los contenedores que se están ejecutando

```
docker ps
```

Detenemos el contenedor

```
docker stop [nombre contenedor]
```

Borrar un contenedor

```
docker rm [nombre_contenedor]
```

Contenedores ejecutando en segundo plano

```
docker run -d --name myUbuntu ubuntu /bin/bash \
    -c "while true; do echo hola mundo; sleep 1; done"
```

> **-d** significa que se trabaje en segundo plano

Vemos los contenedores que se están ejecutando: `docker ps`

Podemos ver la salida de nuestro contenedor: `docker logs myUbuntu`

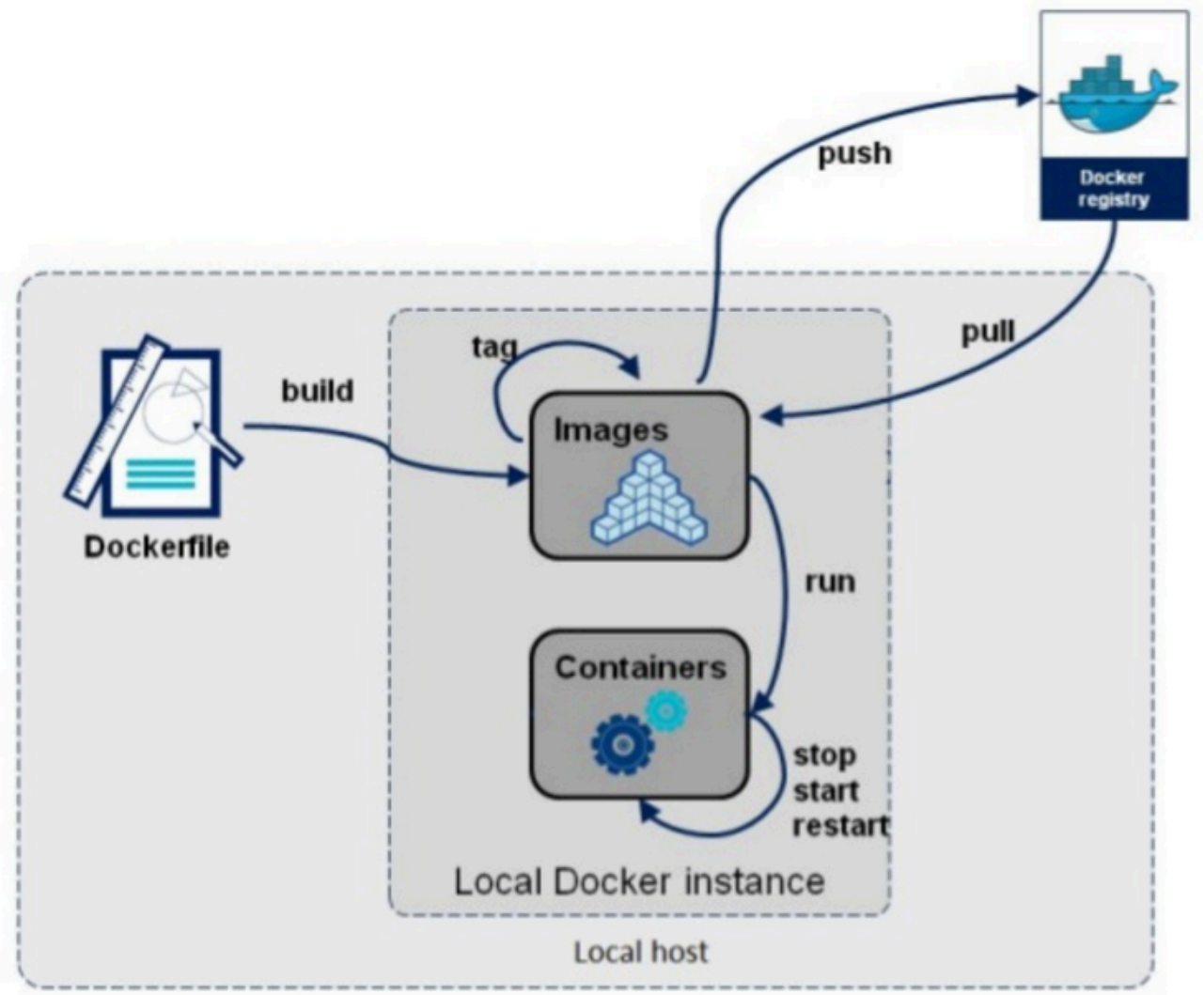
Detenemos el contenedor: `docker stop [nombre contendedor]`

Iniciar un contenedor existente: `docker start [nombre contendedor]`

Abrir terminal de un determinado contenedor

`docker exec -it myUbuntu /bin/bash`

Trabajando con Docker



Opciones

Exportar un contenedor

```
docker export myUbuntu > myUbuntu.tar
```

Importar desde un fichero local

```
docker import /path/to/latest.tar
```

Ejemplo: contenedor con servidor nginx

Ejecutamos nginx (descarga automática si no está en el repositorio local)

```
docker run --name myNginx -d -P nginx
```

> **-d** trabajar en segundo plano
> **-P** mapea los puertos a nuestro host para poder ver la aplicación

No necesitamos especificar comando, ya que la imagen de nginx tiene un comando por defecto

Vemos los puertos usados: `docker ps -a`

> La opción **-P** redirecciona los puertos del contenedor de la imagen (en este caso el 80) a un puerto de nuestro host local: `[0.0.0.0:XXXXX->80/tcp]`

Abrir en el navegador http://localhost:XXXXX/

Otras opciones de ejecución

Especificando un puerto en el host

```
docker run --name myNginx2 -d -p 2000:80 nginx
```

> **-p** mapea el puerto 2000 del host local al 80 del contenedor

Abrir en el navegador http://localhost:2000/

Podemos ver los puertos asociados a nuestro contenedor

```
docker port [nombre_contenedor] [puerto_contenedor]
```

```
docker port myNginx 80
```

Podemos ver los logs de nuestra aplicación

```
docker logs -f myNginx
```

> **-f** muestra los logs de manera continua

Ver procesos que se están ejecutando en el contenedor

```
docker top myNginx
```

Inspeccionar la máquina

```
docker inspect myNginx
```

Detener el contenedor

```
docker stop myNginx
```

Reiniciar un contenedor

```
docker start myNginx
```

Creando nuestros contenedores

- Creación **desde imagen**: Partimos de una imagen de un contenedor para realizar modificaciones y crear el nuestro
- Creación desde cero: Partimos de un **fichero de configuración** para crear una imagen para un contenedor

Creación desde una imagen

Ejecutar un nuevo contenedor nginx

```
docker run --name myNginx3 -ti -d -p 3000:80 nginx
```

Abrir un terminal del contenedor

```
docker exec -t -i myNginx3 /bin/bash
```

Instalar nano, personalizar bienvenida y hacer `docker commit`

```
apt-get update
apt-get install nano
nano /usr/share/nginx/html/index.html
docker commit -m "Modificando saludo" -a "Nombre Autor" myNginx3 \
    usuario/mynginx:v2
```

> - Especificar el nombre o id del contenedor en ejecución a modificar
> - `usuario` debe coincidir con nuestro usuario de Docker Hub
> - El nombre de la imagen `mynginx` debe ir en minúsculas

Vemos nuestra nueva imagen

`docker images`

Ejecutar nuestra imagen

```
docker run --name myNewNginx -ti -d -p 3500:80 \
    usuario/mynginx:v2
```

Creación de un *Dockerfile*

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Comandos básicos de Dockerfile

FROM: para definir la imagen base
MAINTAINER: nombre o correo del mantenedor
COPY: copiar un fichero o directorio a la imagen
ADD: para copiar ficheros desde urls
RUN: ejecutar un comando dentro del container
CMD: comando por defecto cuando ejecutamos un container
ENV: variables de entorno
EXPOSE: para definir los puertos del contenedor
VOLUME: para definir directorios de datos que quedan fuera de la imagen
ENTRYPOINT: comando a ejecutar de forma obligatoria al correr una imagen
USER: Usuario para RUN, CMD y ENTRYPOINT
WORKDIR: directorio para ejecutar los comandos

Ejemplo de Dockerfile

Creamos un directorio: `mkdir myapacheweb && cd myapacheweb`

Creamos un fichero llamado Dockerfile: `touch Dockerfile`

Editamos el fichero con el siguiente contenido:

```
FROM ubuntu:14.04
RUN apt-get update && \
apt-get install -y apache2 && \
apt-get clean && \
rm -rf /var/lib/apt/lists/*
RUN echo "<h1>Apache with Docker</h1>" > /var/www/html/index.html
ENV APACHE_RUN_USER=www-data
ENV APACHE_RUN_GROUP=www-data
ENV APACHE_LOG_DIR=/var/log/apache2
EXPOSE 80
ENTRYPOINT ["apache2ctl", "-D", "FOREGROUND"]
```

Construcción de nuestra imagen

```
docker build -t usuario/myserver .
```

Ejecutamos nuestra imagen

```
docker run -ti -p 3333:80 --name myserver usuario/myserver:latest
```

Probar el contenedor y abrir http://localhost:3333/

Colocamos nuestra imagen en Docker Hub

```
docker push usuario/myserver
```

Averiguar el tamaño de las imágenes

```
docker history usuario/myserver
```

Imagen de Ubuntu

La imagen de ubuntu no trae muchas de las herramientas necesarias para comprobar el funcionamiento del contenedor. Se recomienda instalar las siguientes herramientas.

```
apt-get update
apt-get install nano
apt-get install -y net-tools
apt-get install telnet
apt-get install iputils-ping
apt-get install curl
```

Tutorial de Docker

https://docs.docker.com/get-started/

Ejercicio 1: Apache server

Tareas:

1. Crear un contenedor con Apache Server 2 (buscar en Docker Hub)
2. Personalizar el contenedor. El servidor por defecto muestra en la página principal "It works!". Modificar este mensaje para que muestre un saludo personal: "Hello + (tu nombre y apellidos)!".
3. Configurarlo para acceder mediante el puerto 8082.
4. Subir la imagen del contenedor creado a Docker Hub. La imagen debe llamarse `apacheserver_p1`.

Ejercicio 2: Crear un contenedor con Nginx y un fichero HTML

Tareas:

1. Crear un directorio llamado `my-html-folder`.
2. Dentro de este directorio, crear un fichero llamado `index.html` con cualquier contenido HTML.
3. Crear un fichero llamado `Dockerfile` con las instrucciones necesarias para crear una imagen de Nginx que muestre el contenido del fichero `index.html` (Consultar la documentación de Nginx en Docker Hub).
4. Construir la imagen y ejecutar un contenedor con ella.
5. Configurarlo para acceder mediante el puerto 1234.
6. Subir la imagen del contenedor creado a Docker Hub. La imagen debe llamarse `nginxserver_p1`.