

Docker: una solución al transporte de aplicaciones



Mauricio Rigoberto Martínez Romero

Colaborador1

Colaborador2

**Universidad de El Salvador
Facultad Multidisciplinaria Oriental**



Docker: una solución al transporte de aplicaciones



Objetivos del taller

- Conocer Docker y todos los componentes básicos del mismo. Junto con sus Beneficios.
- Implementar los conocimientos obtenidos dentro del taller, para la elaboración de una pequeña aplicación PHP y su implementación con Docker.



¿Qué es Docker?

Docker containers, envuelve una pieza de software en un sistema de archivos completo que contiene todo lo necesario para **funcionar**: código, tiempo de ejecución, herramientas del sistema, las bibliotecas del sistema- cualquier cosa que se puede instalar en un servidor. **Esto garantiza que el software se ejecutará independientemente de su medio ambiente.**



Estos contenedores de Docker podríamos definirlos como máquinas virtuales ligeras, menos exigentes con los chips y memorias de los equipos donde se ejecutarán.

Las características principales de estos contenedores son la portabilidad, la ligereza y la autosuficiencia:

- **Portabilidad**: El contenedor Docker podremos desplegarlo en cualquier otro sistema (que soporte esta tecnología), con lo que nos ahorraremos el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usamos.



- Ligereza:

El peso de este sistema no tiene comparación con cualquier otro sistema de virtualización más convencional que estemos acostumbrados a usar.

Por poner un ejemplo, una de las herramientas de virtualización más extendida es VirtualBox, y cualquier imagen de Ubuntu que queramos usar en otro equipo pesará entorno a 1Gb si contamos únicamente con la instalación limpia del sistema.

En cambio, un Ubuntu con Apache y una aplicación web, pesa alrededor de 180Mb, lo que nos demuestra un significativo ahorro a la hora de almacenar diversos contenedores que podamos desplegar con posterioridad.



- **Autosuficiencia:**

Un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga. Asimismo Docker se encarga de la gestión del contenedor y de las aplicaciones que contenga.

A parte de ofrecernos un entorno similar a Git para, a base de "capas", controlar cada cambio que se haga en la máquina virtual o contenedor.



Para obtener fluidez Docker extiende LXC (Linux Containers), un sistema de virtualización ligero que permite crear múltiples sistemas totalmente aislados entre si sobre la misma máquina o sistema anfitrión.

Y todo dado que no se emula un sistema operativo completo, sólo las librerías y sistemas de archivos necesarios para la utilización de las aplicaciones que tengamos instaladas en cada contenedor.



Conociendo la historia de Docker

Docker comenzó como un proyecto interno dentro **dotCloud**, empresa enfocado a una plataforma como un servicio (PaaS), iniciado por Salomón Hykes.



Salomón Hykes

Con las contribuciones iniciales de otros ingenieros de dotCloud, incluyendo **Andrea Luzzardi** y **Francois-Xavier Bourlet**. **Jeff Lindsay** también participó como colaborador independiente.



Docker fue liberado como código abierto en marzo de 2013. **El 13 de marzo de 2014**, con el **lanzamiento de la versión 0.9**, Docker dejó de utilizar LXC como el entorno de ejecución por defecto y lo reemplazó con su propia librería, **libcontainer**, escrito en **Go** (lenguaje de programación).

El 13 de abril de 2015, el proyecto tenía más de **20.700 estrellas de GitHub** (haciéndolo uno de los proyectos con más estrellas de GitHub, en **20a posición**), más de **4.700 bifurcaciones ("forks")**, y casi **900 colaboradores**.



Principales contribuyentes de Docker:

- El equipo de Docker
- Red Hat (mayores contribuyentes, aún más que el equipo de Docker en sí)
- IBM
- Google
- Cisco Systems
- Amadeus IT Group



Arquitectura de Docker

Docker usa una arquitectura **cliente-servidor**. El **cliente** de Docker habla con el **Daemon** de Docker que hace el trabajo de crear, correr y distribuir los contenedores.

Tanto el cliente como el Daemon pueden ejecutarse en el mismo Sistema, o puede conectar un cliente remoto a un daemon de docker.

- Docker Engine
- Docker Client



Docker Engine

Docker Engine o Demonio Docker es un demonio que corre sobre cualquier distribución de Linux y que expone una API externa para la gestión de imágenes y contenedores (y otras entidades que se van añadiendo en sucesivas distribuciones de docker como volúmenes o redes virtuales).

El usuario nunca interactúa con el demonio directamente, sino que lo hace por medio del cliente.



Es básicamente un motor de contenedores que usa características del Kernel de Linux como espacios de nombres y controles de grupos, para crear contenedores encima del Sistema operativo, y automatizar el despliegue de aplicaciones en estos contenedores.



Entre sus funciones principales:

- Creación de imágenes docker.
- Publicación de imágenes en un Docker Registry o Registro de Docker (otro componente Docker).
- Descarga de imágenes desde un Registro de Docker
- Ejecución de contenedores usando imágenes locales.

Otra función fundamental del Docker Engine es la gestión de los contenedores en ejecución, permitiendo parar su ejecución, rearrancarla, ver sus logs o sus estadísticas de uso de recursos.



Docker Client

Binario que constituye la interfaz de usuario entre el cliente y el demonio del servicio. Acepta y procesa las acciones del cliente a través de una serie de comandos.

Es cualquier herramienta que hace uso de la api remota del Docker Engine, pero suele hacer referencia al comando docker que hace las veces de herramienta de línea de comandos (cli) para gestionar un Docker Engine.

La cli de docker se puede configurar para hablar con un Docker Engine local o remoto, permitiendo gestionar tanto nuestro entorno de desarrollo local, como nuestros servidores de producción.



Gráfico de la arquitectura de Docker

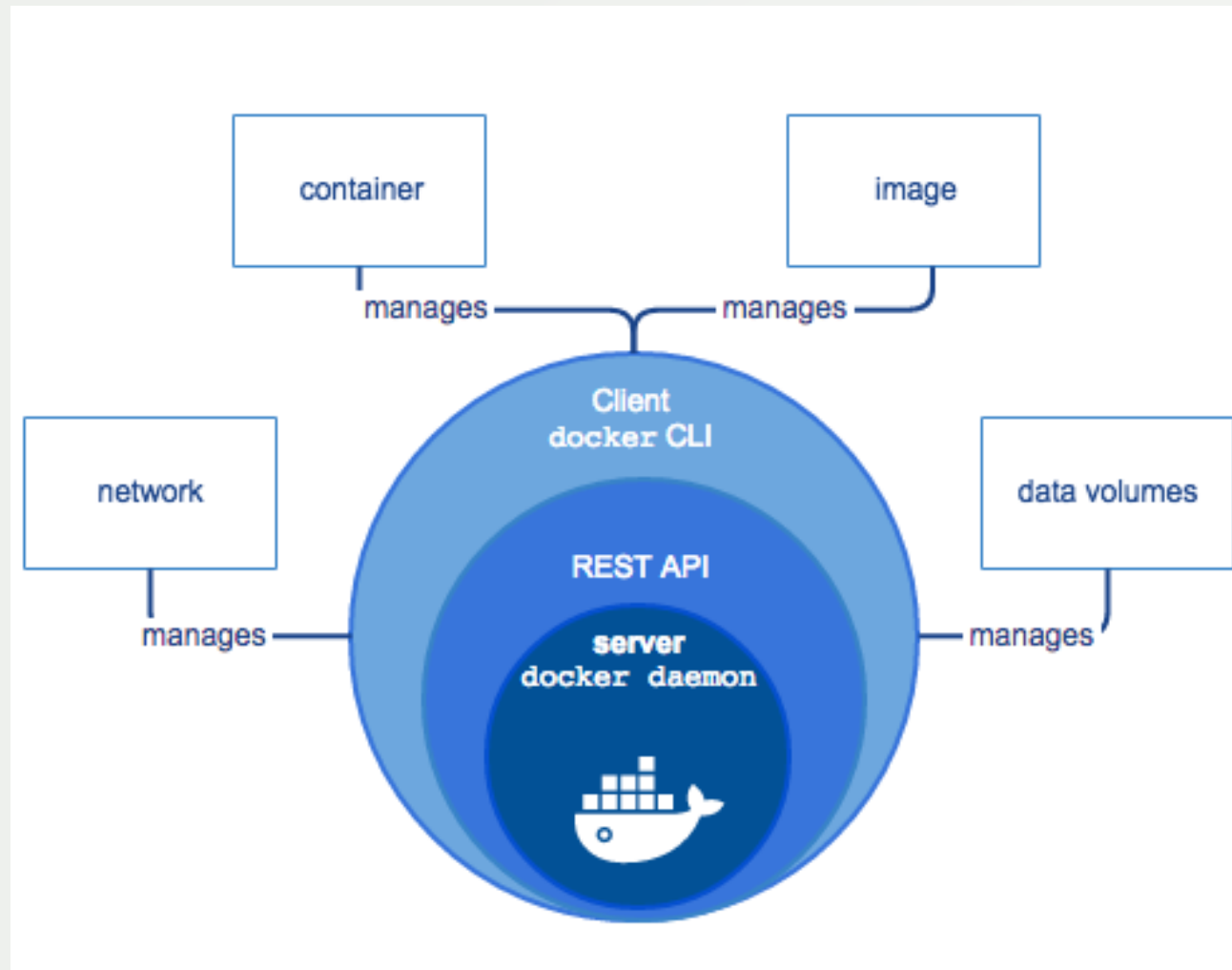
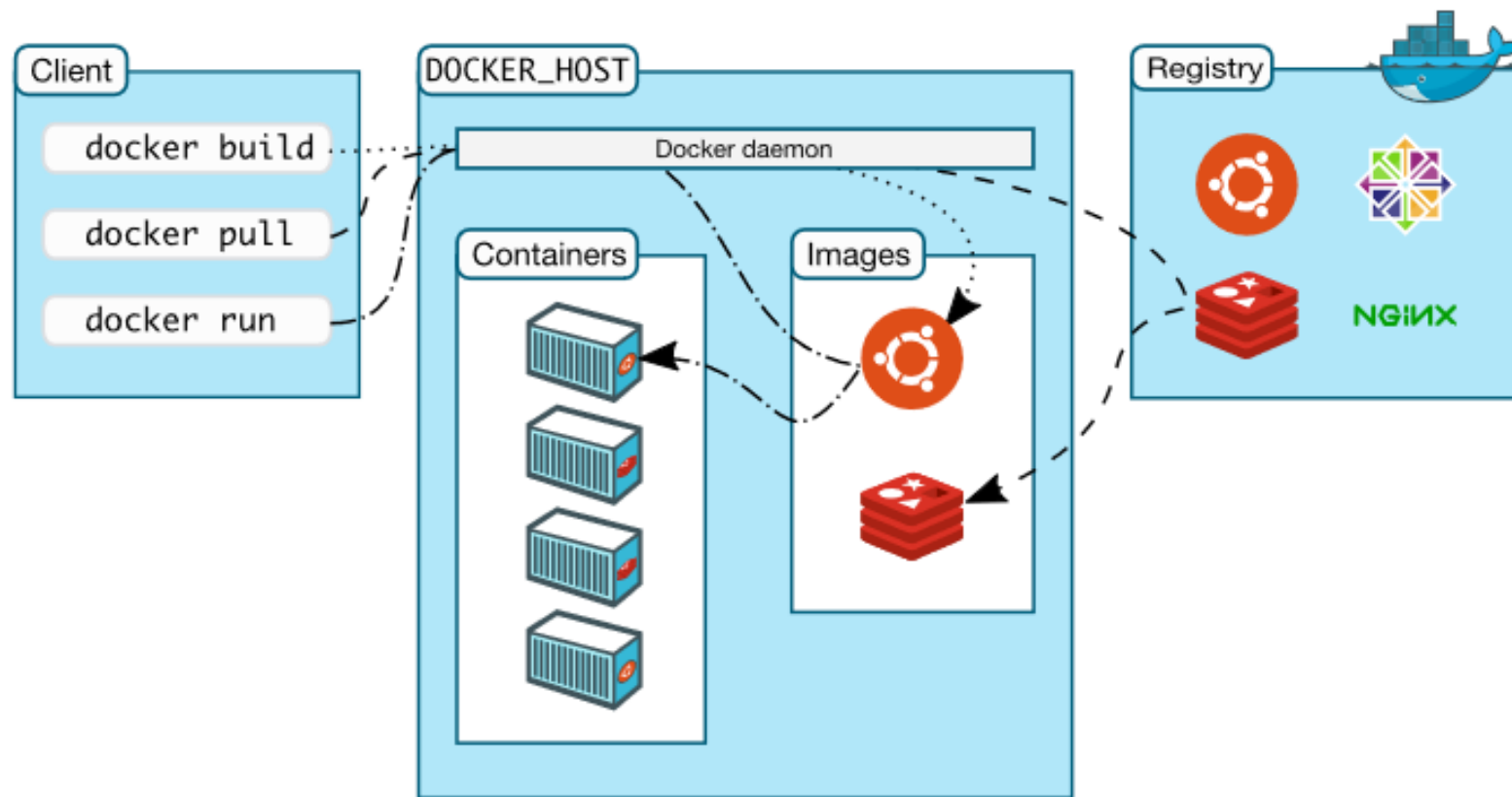


Gráfico de la arquitectura de Docker



Componentes de Docker

- Imágenes de Docker (Docker Images)
- Contenedores de Docker (Docker Containers)
- Registros de Docker (Docker Registries)



Imágenes de Docker (Docker Images)

Una imagen de Docker es una plantilla de lectura. Por ejemplo, una imagen puede contener un sistema operativo Ubuntu además de Apache instalado junto con su aplicación web. Las imágenes se utilizan para crear contenedores Docker.

Docker provee una forma de construir imágenes, o actualizar una existente, además de poder hacer uso de otras imágenes creadas por otras personas.

Las imágenes son uno de los componentes de construcción de Docker.



Imágenes de Docker (Docker Images)

Las imágenes se utilizan para crear contenedores, y nunca cambian.

Hay muchas imágenes públicas con elementos básicos como Java, Ubuntu, Apache...etc, que se pueden descargar y utilizar.



Imágenes de Docker (Docker Images)

Normalmente cuando creas imágenes, partimos de una imagen padre a la que le vamos añadiendo cosas (p.e: una imagen padre con Ubuntu y Apache, que hemos modificado para instalar nuestra aplicación).

Las imágenes se identifican por un ID, y un par nombre-versión, por ejemplo: ubuntu:latest, django:1.6, etc.



Contenedores de Docker (Docker Containers)

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.



Contenedores de Docker (Docker Containers)

Podemos trazar un paralelismo entre el contenedor y una máquina virtual: ambos son sistemas autocontenidos que, en realidad, utilizan un sistema superior para ejecutar sus trabajos. La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor aprovecha el sistema operativo sobre el cual se ejecuta.



Contenedores de Docker (Docker Containers)

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.



Contenedores de Docker (Docker Containers)

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.



Registros de Docker (Docker Registries)

Los registros de Docker guardan las imágenes, estos son repos públicos o privados donde podemos subir o descargar imágenes. El registro público lo provee el Hub de Docker que sirve una colección de imágenes para nuestro uso. Los registros de dockers básicamente son el componente de Distribución de Docker.



Contenedores vs Máquinas Virtuales

Cuando se trata de comparar los dos tipos de tecnologías se podría decir que Docker y sus contenedores tienen mucho más potencial que las máquinas virtuales.

Los aspectos fuertes y débiles al usar Docker vs VM, distinguiendo categorías como rendimiento, rapidez, portabilidad, seguridad y administración.



IMAGEN DE LA DIFERENCIA ENTRE ESTAS DOS COSAS



Rapidez

Docker y sus contenedores son capaces de compartir un solo núcleo y compartir bibliotecas de aplicaciones, esto ayuda a que los contenedores presenten una carga más baja de sistema que las máquinas virtuales.

En comparación con las máquinas virtuales, los contenedores pueden ser más rápidos y consumirán menos recursos siempre que el usuario está dispuesto a pegarse a una única plataforma para proporcionar el sistema operativo compartido.



Una máquina virtual puede tardar hasta varios minutos para crearse y poner en marcha mientras que un contenedor puede ser creado y lanzado sólo en unos pocos segundos.

Para dar un ejemplo claro, los tiempos de inicio y detención de Docker son menores a 50ms, mientras que las máquinas virtuales inician en 30-45 segundos, y se detienen en 10 segundos o menos.



Portabilidad

Todas las aplicaciones tienen sus propias dependencias, que incluyen tanto los recursos de software y hardware. Los Contenedores Docker aportarán numerosos beneficios en comparación con las tecnologías existentes.

En términos de tecnología, es bastante interesante en escenarios donde ayuda en la promoción de la portabilidad de la nube mediante la ejecución de las mismas aplicaciones en diferentes entornos virtuales esto es muy útil en el ciclo de vida para el desarrollo de software.



Docker es una plataforma abierta para desarrolladores, es un mecanismo que ayuda a aislar las dependencias por cada aplicación mediante la creación de contenedores. Los contenedores son escalables y seguros si los comparamos con el enfoque anterior del uso de máquinas virtuales.



Seguridad

Una de las ventajas de la utilización de máquinas virtuales es la abstracción a nivel de hardware físico que se traduce en kernels individuales, que limitan la superficie de ataque al hipervisor (el monitor o núcleo de la virtualización).

En teoría, las vulnerabilidades particulares en las versiones de sistemas operativos no se pueden aprovechar para poner en peligro otras máquinas virtuales que se ejecutan en la misma máquina física.



Administración

Soluciones tales como Docker hacen más fácil la gestión de contenedores, pero muchos clientes todavía encuentran la gestión de contenedores más un arte que una ciencia. Para varios usuarios que han estado trabajando con Docker recientemente y comparten su experiencia y su frustración de la gestión del Docker en un entorno de producción.



Datos Adicionales

La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor aprovecha el sistema operativo sobre el cual se ejecuta.

Por ejemplo, virtualizar una base de datos sobre una máquina virtual, requiere:

- Una máquina física que aporte el hardware
- Un sistema operativo “Host” sobre esta máquina física
- Un sistema de virtualización o hypervisor que gestione las peticiones al hardware virtual y las ejecute sobre el real
- Un sistema operativo “Guest” bajo el hypervisor. Este sistema debe ser completo ya que no puede obtener recursos del kernel de su Host
- Instalar bajo el sistema “Guest” el motor de base de datos y todas sus dependencias



Datos Adicionales

En cambio, virtualizar la misma base de datos sobre Docker requiere:

- Una máquina física o virtual
- Un sistema operativo sobre esta máquina
- El motor de Docker instalado en esta máquina
- Un contenedor basado en una imagen que contenga el motor de base de datos y todas sus dependencias



Datos Adicionales

Las diferencias importantes son:

- El sistema de contenedores no exige una máquina física, mientras que no tiene sentido instalar un hypervisor bajo una máquina virtual (el rendimiento se verá penalizado en extremo). Por lo tanto ganamos versatilidad.
- Cada máquina virtual conlleva virtualizar todo un sistema operativo. En el ejemplo anterior, poner en marcha una segunda base de datos como réplica implicaría destinar recursos a un segundo sistema operativo, mientras que con Docker simplemente lanzaremos un segundo contenedor mucho más ligero. Por lo tanto aprovechamos mejor los recursos.



Datos Adicionales

- Arrancar un contenedor cuesta, en tiempo, lo mismo que arrancar un proceso en el sistema. En cambio, al arrancar una máquina virtual debemos esperar a que arranque también el sistema operativo subyacente. Por lo tanto ganamos agilidad.
- También es importante destacar el tamaño de las imágenes que podemos conseguir. La imagen oficial de Debian 7.7 en Docker Hub ocupa 85.1 MB. La imagen de Nginx ocupa 91.4 MB. ¿Cuánto ocuparían las imágenes de Virtualbox correspondientes? Nos podemos hacer una idea de lo ágil que resulta realizar cualquier operación de aprovisionamiento o recolocación de contenedores.



Ventajas de Docker

- La principal ventaja del uso de Docker es que los desarrolladores pueden centrarse en su código sin tener que preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.
- Por ejemplo, si un desarrollador tiene en su ordenador instalado una versión de JAVA 8 y programa una función específica para esa versión, y otro desarrollador tiene instalado en su máquina la versión JAVA 7, cuando el primer desarrollador quiera probar su programa en la máquina del desarrollador que tiene una versión inferior, la aplicación fallará.



Ventajas de Docker

- Y aquí es donde aparece Docker, el primer desarrollador creará un contenedor de Docker con la aplicación, la versión 8 de Java y el resto de recursos necesarios, se lo pasa al usuario con la versión inferior, y éste con el Docker instalado en su máquina, podrá ejecutar la aplicación a través del contenedor sin tener que instalar absolutamente nada más.
- Estos contenedores tienen unas características principales:
- El contenedor se puede extender en cualquier sistema que soporte la tecnología.



Ventajas de Docker

- Es ligero. Ocupa mucho menos que cualquier otro sistema de virtualización.
- Los contenedores que se ejecutan en una misma máquina comparten Sistema Operativo.
- Además, existen dos componentes esenciales:
- Imágenes: Plantillas con unos sistemas y configuraciones que podríamos utilizar de base para fabricar nuevos contenedores o entornos.
- Repositorios: Contienen imágenes creadas por los usuarios y pueden ser públicos o privados.



Ventajas de Docker

- En resumen, si utilizamos Docker en nuestros obtendremos estos beneficios, algo que, sin duda, cualquier desarrollador agradecerá en el día a día de sus proyectos:
- Se acelera el proceso de mantenimiento y desarrollo, de manera que realizar una copia de los sistemas que están en producción y ejecutarlos en otro equipo, sea una tarea sencilla.
- Las aplicaciones se ejecutan tal y como fueron creadas y no importa el equipo ni el ambiente (pruebas o producción).



- Ventajas de Docker

- En un escenario típico el cliente instalaría y configuraría MySQL y posteriormente la aplicación. Con Docker, ejecutando el contenedor estaría todo listo.
- En DevOps, tanto los desarrolladores como los administradores de sistema pueden probar aplicaciones en un entorno seguro y exactamente igual en todos los casos.



¡¡A INSTALAR!!



Instalacion de docker

```
$ wget -qO- https://get.docker.com/ | sh  
sudo usermod -aG docker $(whoami)
```



PRIMEROS PASOS



Principales comandos de Docker



`docker pull [imagen]:[version]`

Esto nos descargará la imagen con la versión que se ha seleccionado. En caso de no indicar la versión nos descargará la imagen con etiqueta (tag) por defecto. Por ejemplo, si descargamos Debian nos descargará la versión latest.



`docker run [imagen]:[version]`

Este comando nos permite crear un contenedor a través de una imagen.

El comando run permite pasarle una serie de parámetros con los que le podemos asignar memoria, cpu, variables de entorno, etc...



¡Mi primer contenedor con Docker!
:D

docker run hello-world



```
# docker run -i -t ubuntu /bin/echo  
“Este es mi segundo contenedor de  
Docker usando la imagen de Ubuntu”
```

¿Qué hace este comando?*



Parámetros de interés

Si queremos ver la lista completa de parámetros que acepta el comando run ejecutamos:

- # `docker run -h`

Unos cuantos parámetros de uso común al ejecutar el comando `run`:

- `-rm`: elimina el contenedor una vez haya finalizado su ejecución.
- `-t`: asigna un emulador de terminal al comando.



Parámetros de interés

- **-d**: ejecuta el contenedor en modo background.
- **-i**: hace que la ejecución de los comandos sea interactiva, mostrando por pantalla los resultados de la ejecución del comando sobre el contenedor.
- **-P**: cuando pasamos este parámetro al comando run, Docker mapea todos los puertos expuestos en nuestra imagen a nuestro host.



Comandos básicos de docker



docker version

Nos permite ver la versión instalada de Docker en nuestro sistema.

docker info

Este comando nos muestra información acerca de la cantidad de contenedores e imágenes que se encuentran actualmente en nuestra maquina como así también la versión de docker que está ejecutando.



docker images

Este comando nos muestra información acerca de cada una de las imágenes que se encuentran en nuestra maquina (nombre, id, espacio que ocupa, fecha de creación).

docker ps

Este comando nos indicará que contenedores están actualmente ejecutándose en nuestra maquina. Este comando nos mostrará la información de todos los contenedores que existen actualmente, en que estado se encuentran así como cuando fueron apagados.



docker search [busqueda]

Este comando nos permite buscar imágenes en el repositorio oficial de Docker. Para no tener que ir buscando en la web, también podemos buscar imágenes desde la Terminal.

docker rm [imagen]:[version]

Este comando nos permite eliminar una imagen. En caso de no informar la versión/tag procederá a borrar todas las que tengan el mismo nombre sin importar la versión.



`docker commit [imagen]:[version]`

Aquí se pone interesante esto. Cuando montamos un contenedor con una imagen, podemos configurar dicha imagen y para evitar que estos cambios se pierdan, podemos “guardarlos” haciendo un commit.

`docker save [imagen] > [archivo].tar`

Si queremos exportar una imagen a otra imagen podremos hacerlo con este comando. Este comando lo que hace es exportar la imagen que indicamos en un archivo tar.



`docker load < [archivo].tar`

Si antes podíamos exportar, este comando permite hacer lo contrario. Cargar la imagen que se encuentra dentro del fichero tar para luego ser usado en nuestro entorno de Docker.



Conociendo mejor las imágenes de docker

Construir imágenes desde Dockerfiles



¿Qué es un Dockerfile?

Un Dockerfile no es más que un archivo que reconoce Docker y que contiene una serie de instrucciones para automatizar el proceso de creación de un contenedor. En pocas palabras en este archivo vamos a agregar todo lo que necesitamos en nuestro contenedor para que cada vez que queramos correr el script de construcción obtengamos un contenedor completamente fresco y actualizado.



¿Cómo creo un Dockerfile?

Para crear una imagen a partir de un Dockerfile es necesario tener una carpeta base en donde se almacenará el archivo para luego compilarlo.

```
# mkdir taller_Docker && cd taller_Docker
```

Luego tenemos que crear el archivo Dockerfile

```
# nano Dockerfile
```



¿Cómo construyo el Dockerfile?

Para construir un Dockerfile, Docker tiene ciertas instrucciones a seguir, una de ellas son;

- FROM:

Se especifica que imagen se ocupará como base.

- MAINTAINER:

La persona que crea la imagen.



¿Cómo construyo el Dockerfile?

- RUN:

El comando RUN ejecuta directamente comandos dentro de el contenedor, y luego persiste los cambios en el contenedor una vez persistidos los cambios continua al siguiente paso (línea) que se encuentre en el Dockerfile.

- ADD:

El comando ADD es utilizado para copiar archivos hacia el contenedor, por lo tanto utiliza dos argumentos que son la fuente y el destino, si la fuente es un URL Docker se encarga de bajar el archivo de Internet y copiarlo al destino descrito.



¿Cómo construyo el Dockerfile?

- EXPOSE:

el comando EXPOSE lo utilizamos para asociar puertos, permitiéndonos exponer un contenedor con el mundo (la computadora anfitrión).

- CMD:

El comando CMD es de cierta manera muy similar al comando RUN con la ligera diferencia que no se ejecuta el comando descrito cuando se corre el comando build sino cuando instanciamos (ponemos a funcionar) el contenedor.



¿Cómo construyo el Dockerfile?

- VOLUME:

El comando VOLUME permite al contenedor crear un punto de montura mediante un nombre, es decir si escribimos `VOLUME ["/var/www"]` estaremos creando un punto de montura en el directorio especificado y esto permite compartir dicho punto de montura con otros contenedores o con la máquina anfitrión.

- WORKDIR:

El comando WORKDIR es bastante sencillo, el mismo nos permite especificarle a Docker en que directorio va a ejecutar un CMD. Puede ser algo cómo `WORKDIR /var/www`



```
FROM debian
MAINTAINER Participate Taller "taller.docker.2016@gmail.com"

RUN apt-get update && apt-get install -y apache2 && apt-get clean
&& rm -rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

EXPOSE 80
ADD ["index.html", "/var/www/html/"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```



En este caso utilizamos la imagen base de debian, instalamos el servidor web apache2, para reducir el tamaño, borramos la caché de paquetes apt y la lista de paquetes descargada, creamos varias variables de entorno (en este ejemplo no se van a utilizar, pero se podrían utilizar en cualquier fichero del contexto, por ejemplo para configurar el servidor web), exponemos el puerto http TCP/80, copiamos el fichero index.html al DocumentRoot y finalmente indicamos el comando que se va a ejecutar al crear el contenedor, y además, al usar el comando ENTRYPOINT, no permitimos ejecutar ningún otro comando durante la creación.



```
# docker build -t {participanteTaller}/docker-compdes2016:1.0
```

Generamos la nueva imagen con el comando docker build con la opción -t indicamos el nombre de la nueva imagen (para indicar el nombre de la imagen es recomendable usar nuestro nombre de usuario en el registro docker hub, para posteriormente poder guardarlas en el registro), mandamos todos los ficheros del contexto (indicado con el punto).

Podemos comprobar que tenemos generado la nueva imagen:

```
# docker images
```



A continuación podemos crear un nuevo contenedor a partir de la nueva imagen:

```
# docker run -p 80:80 --name servidor_web  
{participanteTaller}/docker-compdes2016:1.0
```

Comprobamos que el contenedor está creado:

```
# docker ps
```

¿Cómo tengo acceso al contenedor desde el navegador?



Modificando nuestra imagen

Luego de haber instalado creado nuestra imagen con Apache, llegó la hora de mostrar, como usar una imagen existente y añadirle PHP5.

Primero paramos el contenedor.

```
# docker stop [container ID]
```

Luego eliminamos el contenedor

```
# docker rm [container ID]
```



Modificando nuestra imagen

Creamos carpeta para el Dockerfile

```
# mkdir php && cd php
```

Creamos un archivo php

```
# echo "<?php echo phpinfo();?>">index.php
```



Modificando nuestra imagen

Y ahora creamos nuestro archivo Dockerfile que contendrá lo siguiente;

nano Dockerfile



```
FROM {participanteTaller}/docker-compdes2016:1.0
MAINTAINER Participate Taller "tallerdocker-2016@gmail.com"

RUN apt-get update && apt-get install -y php5 && apt-get
clean && rm -rf /var/lib/apt/lists/*

EXPOSE 80
ADD ["index.php", "/var/www/html/"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```



Compilamos:


```
# docker build -t {participanteTaller}/docker-compdes2016:2.0
```

Ejecutamos el contenedor

```
# docker run -p 80:80 --name servidor_web  
{participanteTaller}/docker-compdes2016:2.0
```



Docker Compose

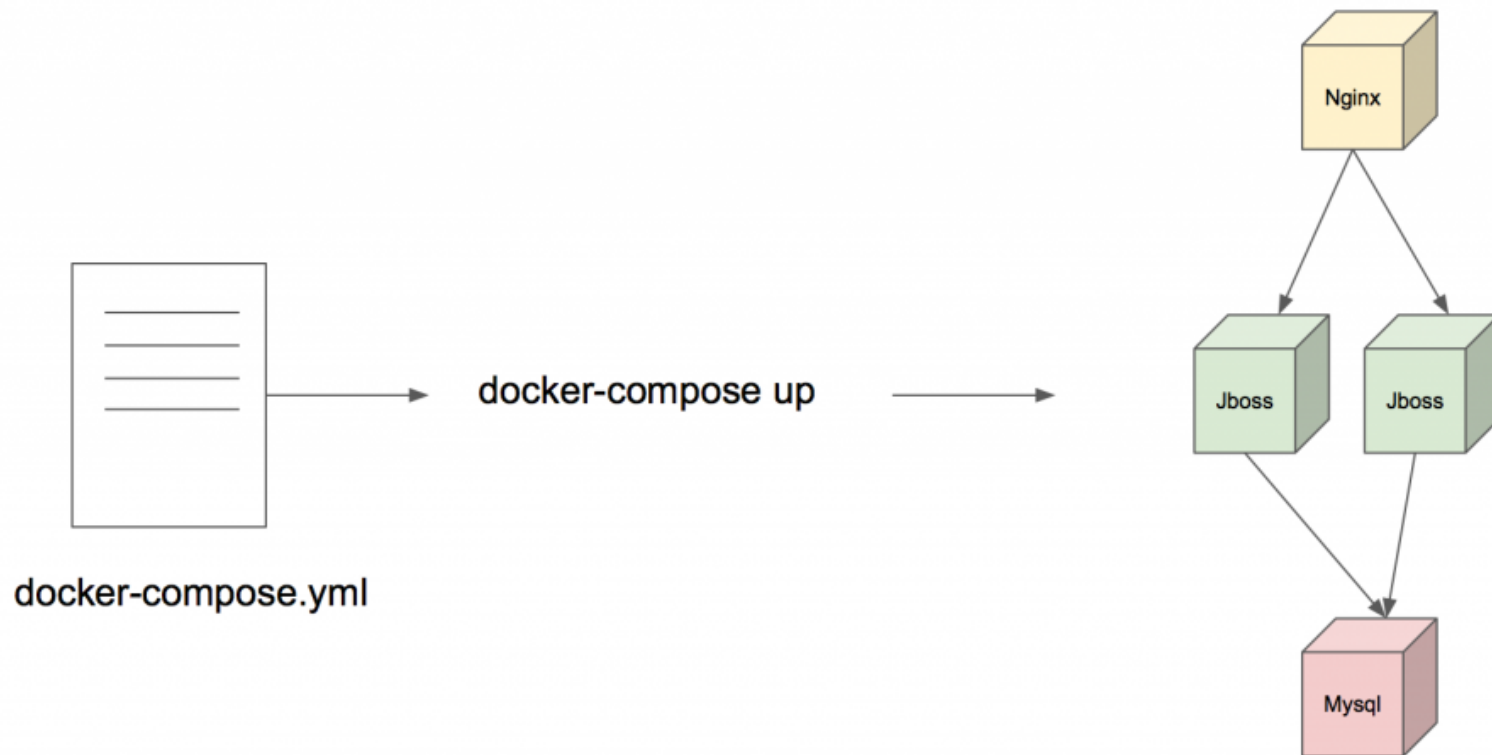
The bottom left corner of the slide features a decorative graphic composed of several overlapping triangular and quadrilateral shapes. These shapes are colored in a gradient of purple, ranging from a dark, almost black shade to a lighter, medium purple. The shapes are arranged in a way that creates a sense of depth and movement, extending from the bottom left towards the center of the slide.

Docker Compose

El descriptor de los contenedores a usar con Docker Compose es un archivo de texto con formato [yaml](#) en la que especificamos los diferentes contenedores y sus propiedades, básicamente podemos indicar las mismas propiedades que indicamos arrancando los contenedores individualmente con el comando docker run.



Docker Compose



Instalación de Docker Compose

```
$ apt-get -y install python-pip
```

```
$ pip install docker-compose
```



Docker Compose

EJEMPLO DE DOCKE COMPOSE



Docker y el Cloud Computing



Docker y el Cloud Computing

Amazon Web Services y Microsoft están efectuando cambios rápidamente para lograr que los contenedores Docker se incluyan en sus respectivos hosts de nube. Los contenedores, en ocasiones descritos como virtualización ligera, prometen desplazar el software más fácilmente y nivelar el terreno de juego entre nubes.

En el futuro, se espera que los contenedores estén anidados. Un componente de software que conforma una capa en un contenedor puede ser llamado por otro situado en una localización remota. Resulta posible pasar una actualización de la misma capa sobre cualquier otro contenedor que utilice el mismo componente.



Docker y el Cloud Computing

La contenización se va a convertir en un elemento atractivo a ojos de la próxima generación de desarrolladores, en parte porque puede ajustarse de cualquier manera mediante una sofisticada gestión y herramientas de virtualización . Existen pruebas de IBM indicando que los contenedores se despliegan más rápidamente y se ejecutan de manera más eficiente que las máquinas virtuales. Asimismo, pueden empaquetarse en los servidores alcanzando mayor densidad. En la nube esto es un elemento muy positivo, ya que la eficiencia global constituye la prueba decisiva que indica quién sobrevivirá y quién no.



Amazon y Docker

Amazon es uno de los mas interesados en Docker ya que puede abrir un nuevo mundo para su empresa y los desarrolladores. Amazon describe a Docker de esta manera **“La ejecución de Docker en AWS es una manera muy fiable y económica de crear, ejecutar, probar e implementar aplicaciones distribuidas de forma rápida y a cualquier escala. AWS proporciona soporte técnico para las soluciones de código abierto o comerciales de Docker, dentro de los servicios de AWS.”**



Amazon y Docker

Los contenedores son un método de virtualización del sistema operativo que le permiten ejecutar una aplicación y sus dependencias en procesos de recursos aislados. Los contenedores también le permiten empaquetar con facilidad el código de una aplicación, sus configuraciones y sus dependencias en bloques de construcción de uso sencillo que aportan uniformidad de entorno, eficacia operativa, productividad para los desarrolladores y control de versiones.



Amazon y Docker

La ejecución de contenedores en la nube de AWS le permite crear aplicaciones y servicios sólidos y escalables al aprovechar los beneficios de la nube de AWS, como la elasticidad, disponibilidad, seguridad y economía de escala. Asimismo, solo paga por los recursos que utiliza.



Amazon y Docker

Productividad para los desarrolladores

Los contenedores incrementan la productividad de los desarrolladores al eliminar las dependencias y los conflictos entre servicios. Cada componente de una aplicación se puede dividir entre varios contenedores que ejecutan un microservicio distinto. Los contenedores están aislados entre sí, de manera que no tiene que preocuparse de que las bibliotecas o dependencias estén sincronizadas para cada servicio. Los desarrolladores pueden actualizar cada uno de los servicios de manera independiente, ya que no existen conflictos de biblioteca.



Donde puedo utilizar docker y el cloud

Explicacion del cloud



Dockerhub

The bottom left corner of the slide features a series of overlapping geometric shapes. These include a dark grey triangle, a black triangle, and two large purple triangles that overlap each other and the other shapes, creating a modern, abstract design.

Docker Hub

Docker Hub es un servicio de registro basado en la nube que permite insertar un enlace de repositorios de código, construye sus imágenes y prueba de ellos, empujando manualmente imágenes y enlaces a la nube por lo que pueden desplegar imágenes a sus anfitriones. Proporciona un recurso centralizado para el descubrimiento de la imagen de contenedores, distribución y gestión de cambios, el usuario y la colaboración en equipo, y la automatización del flujo de trabajo a lo largo de la línea de desarrollo.



Docker Hub

Primer paso hay que crear una cuenta em Docker Hub.

Luego de tener un identificador de la siguiente forma...

IddeDockerHub/imagen:version

ID Docker Hub: **tallerdocker**



Subir nuestra imagen com Apache y PHP a Docker Hub

Una vez hecho, para subir la nueva imagen creada a partir del Dockerfile ejecutamos el siguiente comando:

```
$ docker push {participanteTaller}/docker-compdes2016:2.0
```

Nos lanzará un error ya que nuestro identificador no es {participanteTaller}, para ello debemos de renombrar la imagen com una etiqueta com el siguiente comando:

```
$ docker tag {participanteTaller}/docker-compdes2016:2.0  
tallerdocker/{participanteTaller}-docker-compdes2016:2.0
```



Subir nuestra imagen con Apache y PHP a Docker Hub

Donde IDdockerhub es el nombre de usuario en Docker Hub. Ahora no deberíamos tener problemas a la hora de subir la imagen al repositorio:

```
$ docker push tallerdocker/{participanteTaller}-docker-compdes2016:2.0
```

Es probable que cuando ejecutes docker images veas tanto la imagen tagueada incorrectamente como la nueva que has subido al repositorio. Es recomendable que elimines la imagen incorrecta para evitar futuros errores:

```
$ docker rmi {participanteTaller}/docker-compdes2016:2.0
```



Ejercicio Práctico



<http://www.hoplasoftware.com/productos/docker-toolbox/docker-engine/>
https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%205/docker_architecture.html

<http://helloit.es/2016/01/introduccion-a-docker/>
<http://www.nacionrosique.es/2016/06/componentes-de-dockers.html>

<https://openwebinars.net/academia/18/curso-de-docker/870/componentes-de-docker/preview/>
https://jsitech1.gitbooks.io/meet-docker/content/mejores_practicas_dockerfiles.html

<http://www.hipertexto.info/documentos/namespaces.htm>

<http://7542.fi.uba.ar/tecnica/docker-para-virtualizacion-de-trabajos/>



<http://blog.crespo.org.ve/2016/02/compartir-imagenes-docker-por-medio-de.html>

