Guía del Curso de Docker

Introducción a Docker

Introducción y fundamentos de Docker

Docker es una plataforma abierta para que desarrolladores y administradores de sistemas desarrollen, envíen y ejecuten aplicaciones distribuidas, ya sea en computadoras portátiles, maquinas virtuales de centros de datos o en la nube.

Docker empaqueta software en **"contenedores"** que incluyen en ellos todo lo necesario para que dicho software se ejecute, incluidas librerías. Con **Docker** se puede implementar y ajustar la escala de aplicaciones de una forma rápida en cualquier entorno con la garantía de que el código se ejecutará.

A primera vista se piensa en **Docker** como una especie de máquina virtual "liviana", pero la verdad no lo es. En **Docker** no existe un **hypervisor** que virtualice hardware sobre el cual corra un sistema operativo completo. En **Docker** lo que se hace es usar las funcionalidades del **Kernel** para encapsular un sistema, de esta forma el proyecto que corre dentro de el no tendrá conocimiento que está en un **contenedor**. Los **contenedores** se encuentran aislados entre sí y se comportaran como máquinas independientes.

Iniciar un **contenedor** no tiene un gran impacto a diferencia de iniciar una máquina virtual ya que no tiene que iniciar un sistema operativo completo (desde cero). Gracias al uso de **contenedores** la demanda de recursos baja

limitándose sólo al consumo de la aplicación que contenga. Un **contenedor** inicia en milisegundos.

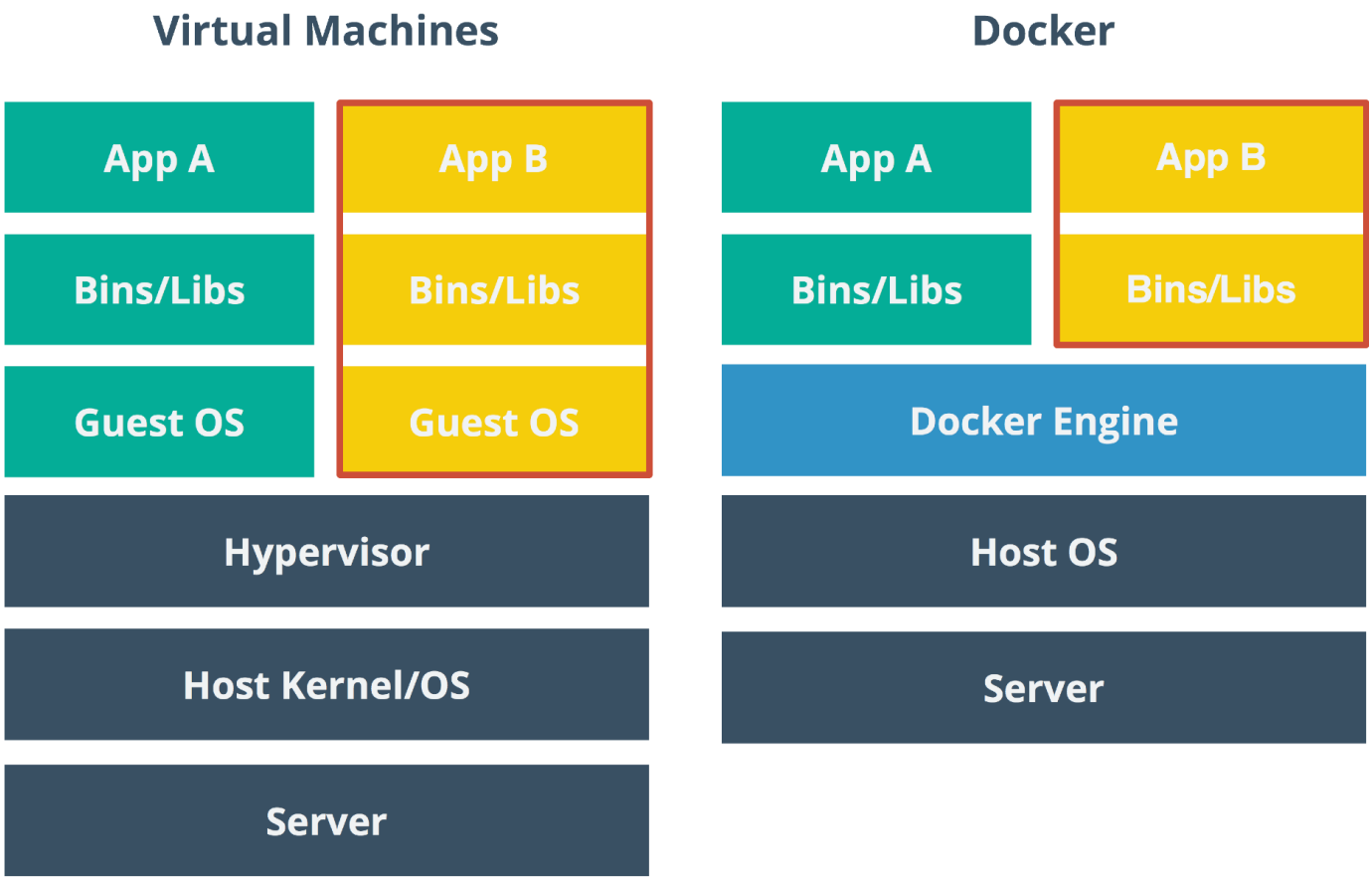
Contenedores y VMs

Anteriormente hablamos de manera de introducción sobre los **contenedores** pero ahora definamos en sí este concepto, **Docker** trabaja con algo que se llama **"contenedores de Linux"** estos son un conjunto de tecnologías que juntas forman un **contenedor** (de **Docker**), este conjunto de tecnologías se llaman:

- **Namespaces:** Permite a la aplicación que corre en un **contenedor** de **Docker** tener una vista de los recursos del sistema operativo.
- **Cgroups:** Permite limitar y medir los recursos que se encuentran disponibles en el sistema operativo.
- **Chroot:** Permite tener en el **contenedor** una vista de un sistema "falso" para el mismo, es decir, crea su propio entorno de ejecución con su propio **root** y home.

Algunas de las características más notables de un **contenedor** son:

- Los contenedores son más livianos (ya que trabajan directamente sobre el **Kernel**) que las maquinas virtuales.
- No es necesario instalar un sistema operativo por **contenedor**.
- Menor uso de los recursos de la máquina.
- Mayor cantidad de **contenedores** por equipo físico.
- Mejor portabilidad.



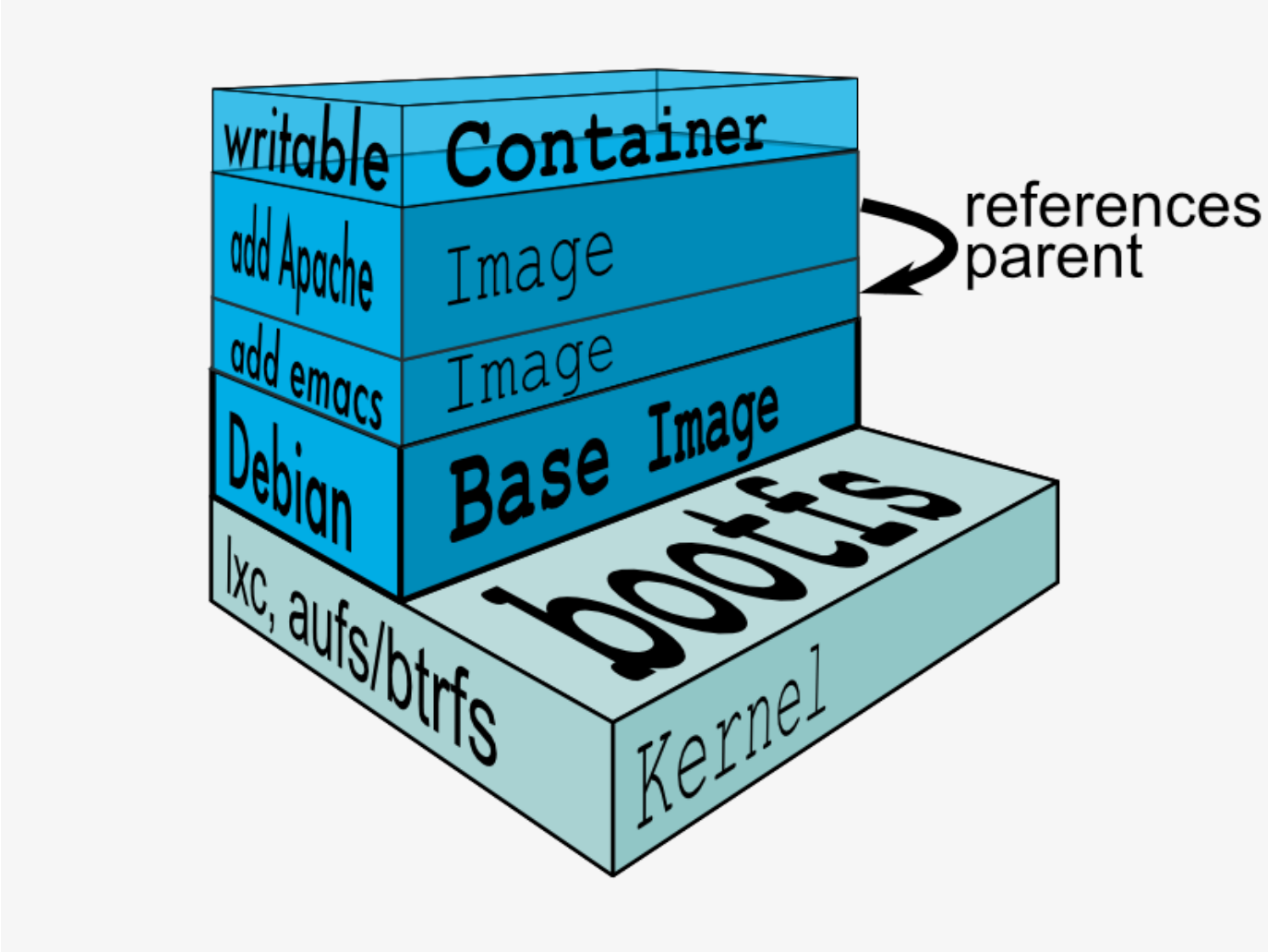
Instalación

En este caso veremos como instalar Docker en Linux, es importante saber que se debe tener un sistema operativo de 64 bits, los pasos son los siguientes:

1. Acceder a get.docker.com (https://get.docker.com/)
2. Copiar el comando para ejecutar el script.
3. Pegar el comando y ejecutar el script de instalación de Docker.
4. Verificar si Docker esta instalado con los comando **`docker info`** o **`docker version`**
5. Crear un "Hello World" con Docker usando el comando **`docker hello-world`**

Docker Engine e Imágenes

Una imagen según Docker es una plantilla de solo lectura vacía o con un aplicación pre-instalada para la creación de contenedores, estas pueden ser creadas por nosotros o terceros. Se pueden guardar en un registro interno o público, puedes encontrar imágenes en hub.docker.com (https://hub.docker.com/)



Para descargar imágenes de un repositorio externo, se utiliza en comando **`docker pull`**

Cuando se ejecuta un contenedor con el comando **`docker run`** las imágenes son descargadas automáticamente.

Algunos comandos útiles para el manejo de imágenes y **contenedores** en **Docker** son los siguientes:

- **`docker images`** Lista las imágenes locales.
- **`docker pull [nombre de la imagen]`** Descarga una imagen de **Docker**.
- **`docker pull [nombre de la imagen]:[tag]`** Descarga una imagen de **Docker** con un tag en especifico.
- **`docker ps`** Lista los contenedores que se encuentran en ejecución.
- **`docker ps -a`** Lista todos los contenedores que estén o no en ejecución.

- **`docker ps -a -q`** Lista y muestra sólo los ID de todos los contenedores que estén o no en ejecución.
- **`docker ps -a --no-trunc`** Muestra el formato extendido del ID de los contenedores.
- **`docker ps --filter="[tipo de filtrado]"`** Filtra los contenedores en función del código de salida, del estado...
- **`docker ps --help`** Muestra un menú de ayuda.
- **`docker attach [ID/nombre del contenedor]`** Permite ingresar nuevamente a un contenedor creado anteriormente.
- **`docker run --name [nombre del contenedor] [imagen]`** Permite iniciar un contenedor con un nombre.
- **`docker rm [ID/nombre del contenedor]`** Permite eliminar un contenedor.
- **`docker rm [ID/nombre...] [ID/nombre...] [ID/nombre...]`** Permite eliminar más de un contenedor.
- **`docker rmi [nombre de la imagen]`** Permite eliminar una imagen.
- **`docker logs [ID/nombre del contenedor]`** Muestra los logs de un contenedor.
- **`docker logs -f [ID/nombre del contenedor]`** Muestra los logs en tiempo real de un contenedor.
- **`docker exec [opciones] [ID/nombre del contenedor] [comando]`** Permite ejecutar nuevas acciones en un contenedor.
- **`docker start -a [ID/nombre del contenedor]`** Permite iniciar nuevamente un contenedor.
- **`docker pause [ID/nombre del contenedor]`** Permite pausar un contenedor en ejecución.
- **`docker unpause [ID/nombre del contenedor]`** Permite reanudar un contenedor pausado.
- **`docker stop [ID/nombre del contenedor]`** Permite detener un contenedor en ejecución.
- **`docker kill [ID/nombre del contenedor]`** Permite matar (detener forzosamente) un contenedor en ejecución.
- **`docker inspect`** Muestra información útil de un contenedor.

- **`docker history [ID/nombre del contenedor]`** Muestra las capas que conforman una imagen con información referente a ella.

Con el comando **`docker run [opciones] [imagen] [comando] [args]`** se creará un contenedor a partir de una imagen. Si Docker no consigue la imagen en el área local la descargará.

Por medio de las opciones que se les pasa a los contenedores, estos pueden ser interactivos. Algunas muy importantes de conocer son:

- **`-i`** Le indica a Docker que se utilizará el STDIN del contenedor.
- **`-t`** Le indica a Docker que se requiere una pseudo-terminal en el contenedor.
- **`-d`** Le indica a Docker que el contenedor correrá en el background.
- **`-P`** Expone los puertos utilizados por un contenedor.
- **`-p [puerto "del contenedor":puerto "de la aplicación"]`** Permite establecer un puerto.

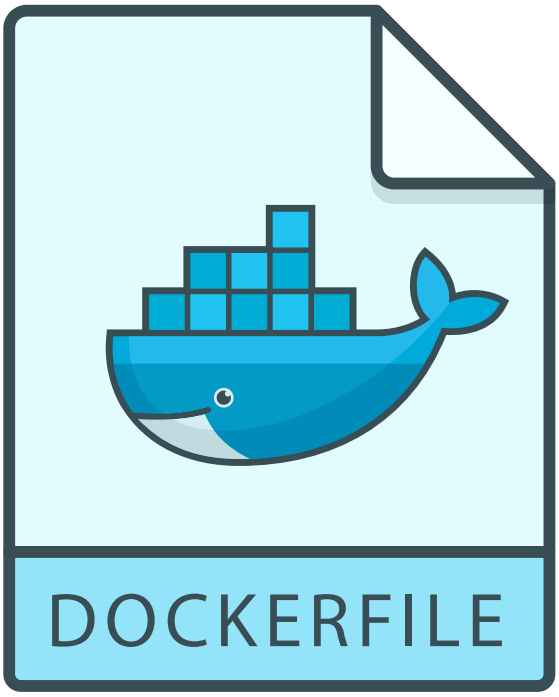
Con las combinaciones de teclas **`CNTRL + PQ`** el servicio de Docker quedará corriendo en el background.

Existen tres formas de crear imágenes:

1. Hacer commit de los contenidos de un contenedor: **`docker commit [ID/nombre...] [nombre para la imagen]:[tag]`**
2. Construir una imagen basada en un Dockerfile.
3. Importar un archivo Tar a Docker con el contenido de un imagen.

Dockerfiles

Un Dockerfile es un documento de texto que contiene todos los comandos que un usuario puede llamar en la línea de comando para armar una imagen.



Instrucciones que podría contener un archivo **Dockerfile**:

```
FROM [imagen]:[tag]
MAINTAINER joaquinaraujojs@gmail.com
LABEL "[clave]": "[valor]"
WORKDIR /[directorio raíz del proyecto]
COPY [ruta-relativa-del-archivo/carpeta] [dest
ADD [URL del archivo ha descargar]
RUN [comando] && [comando] && ... && \
CMD [comando]
ENTRYPOINT [comando]
EXPOSE [puerto]
ENV [variable de entorno]
```

Los comandos a continuación permiten la construcción y configuración de una imagen a partir de un documento **Dockerfile**:

- **`docker build [ruta del dockerfile]`** Inicializa la constricción de una imagen a partir de un archivo **Dockerfile**.

- **`docker build -t [nombre-de-la-imagen:tag] [ruta...]`** Permite establecer un nombre y un tag a la imagen a generar.

Volúmenes

Un volumen es un directorio designado en el contenedor en el cual la información persiste independientemente del ciclo de vida de dicho contenedor.

Algunos comando para gestionar volúmenes en Docker

- **`docker volume create --name [nombre]`** Permite crear un volumen y asignarle un nombre.
- **`docker volume ls`** Lista los volúmenes existentes.
- **`docker run -it -v [nombre del volumen]:/[punto/directorio de montaje] [imagen] [comando]`** Permite correr un contenedor de forma interactiva que contendrá un volumen.
- **`docker volumen inspect [nombre del volumen]`** Muestra información de interés relacionada al volumen.
- **`docker volumen rm [nombre del volumen]`** Permite eliminar un volumen.

Aspectos de seguridad en Docker

Pasos para configurar Docker con TLS:

- **`mkdir docker-ca`**
- **`cd docker-ca`**
- **`openssl genrsa -aes256 -out ca-key.pem 2048`**
- **`openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem`**
- **`openssl genrsa -out server-key.pem 2048`**
- **`openssl req -subj "/CN=[hostname]" -new -key server-key.pem -out server.csr`**
- **`echo "subjectAltName = IP:[IP/127.0.0.1]" > extfile.cnf`**

- **openssl x509 -req -days 365 -in server.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out server-cert.pem -extfile extfile.cnf**
- **openssl genrsa -out client-key.pem 2048**
- **openssl req -subj "/CN=client" -new -key client-key.pem -out client.csr**
- **echo "extendedKeyUsage = clientAuth" > extfile.cnf**
- **openssl x506 -req -days 365 -in client.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out client-cert.pem -extfile extfile.cnf**

Pasos para asegurar las claves:

1. Asegurar las claves del cliente y el servidor de modo que solo puedan ser leídas por el usuario actual: **chmod -v 0400 ca-key.pem client-key.pem server-key.pem**
2. Remover el acceso de escritura a todos los certificados: **chmod -v 0444 ca.pem client-cert.pem server-cert.pem**
3. Crear la carpeta /etc/docker en caso de que no exista.
4. Cambiar los permisos de la carpeta /etc/docker: **sudo chown <username>:docker /etc/docker sudo chmod 700 /etc/docker**
5. Copiar las claves del servidor a la nueva carpeta: **sudo cp ~/docker-ca/{ca, server-key, server-cert}.pem /etc/docker**

Pasos para usar Docker con TLS:

1. Iniciar el servicio con los siguientes parámetros:

```
DOCKER_OPTS="-H tcp://0.0.0.0:2376 --tlsver
```

2. Reiniciar el servicio de Docker en caso de ser necesario: **sudo service docker restart**
3. Utilizar las credenciales correspondientes en el cliente (carpeta docker-ca):

```
docker --tlsverify --tlscacert=./ca.pem --t
```

Redes

Por defecto, al instalar **Docker** en **Linux** se crea una interfaz de red virtual llamada **docker0** en el equipo host.

El comando **`docker network`** permite interactuar con las redes de **Docker** y los contenedores dentro de ellas, unos comandos importantes a tomar en cuenta son:

- **`docker network create [nombre de la red]`** Permite crear una red.
- **`docker network connect [nombre de la red][nombre del contenedor]`** Permite conectar un contenedor de una red.
- **`docker network ls`** Lista las redes existentes.
- **`docker network rm`** Permite eliminar una red.
- **`docker network disconnect`** Permite desconectar un contenedor de una red.
- **`docker network inspect`** Muestra información de interés sobre las redes de **Docker**.
- **`--net [nombre de la red]`** Utilizar esta bandera al crear un **contenedor** permite especificar la red en la cual estará dicho **contenedor**.
- **`--link [nombre del contenedor]`** Permite enlazar dos **contenedor** y así comunicarse entre ellos por medio de su nombre.

La red **bridge** es la que utilizan todos los **contenedores** por defecto, la **host** permite que los **contenedores** compartan la misma IP que la del equipo host.

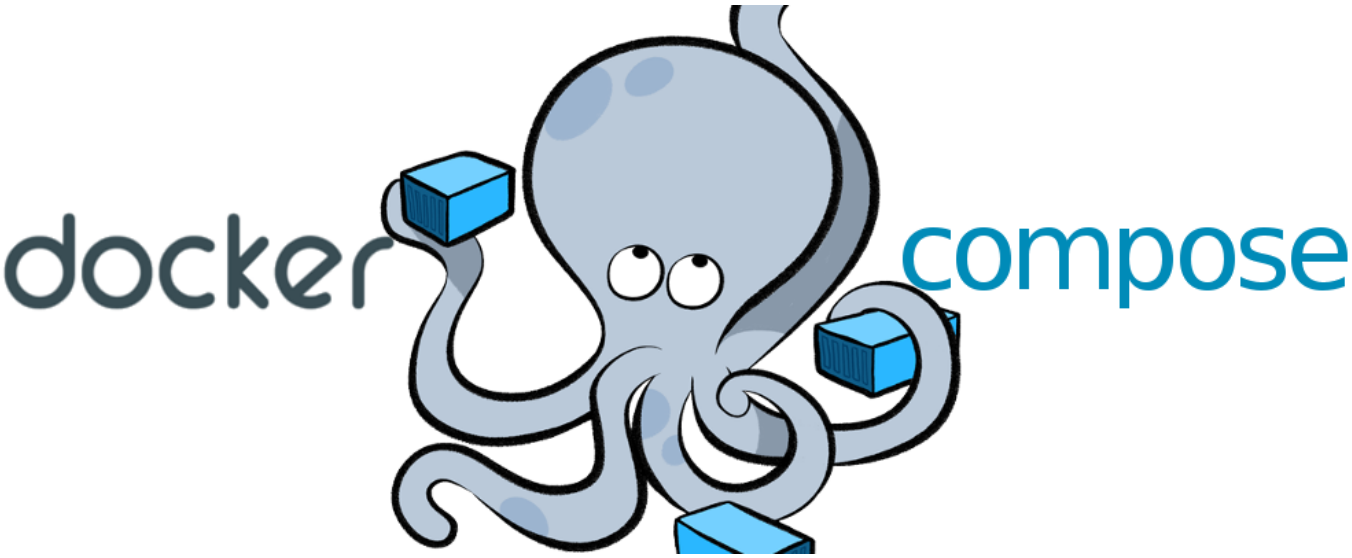
Docker Machine

Machine (https://github.com/docker/machine/releases) es una herramienta para provisionar hosts de **Docker** y configurar el Engine en los mismos.

Docker Compose

Compose (https://github.com/docker/compose/releases) es una herramienta para crear y administrar aplicaciones multi-contenedor.

Los microservicios son aplicaciones individuales que tiene un comportamiento independiente y hablan un protocolo en común, por ejemplo, JSON sobre HTTP.



En una arquitectura de microservicios existen muchos servicios, por tal motivo tiene múltiples componentes para correr.

docker-compose.yml define los servicios que componeme la aplicación y cada servicio contiene las instrucciones para construir y ejecutar el contenedor.

```
    web:
        build: [ubicación del archivo Dockerfile]
        links:
            - [nombre del contenedor a enlazar]
        ports:
    #   expose:
    #       - "[puerto en el que escucha la aplica
            - "[puerto en el que escuchará el cont
    redis:
        image: [nombre de la imagen]
    # lb:
    #     image: dockercloud/haproxy:latest
    #     links:
    #         - web
    #     ports:
    #         - "80:80"
    #     environment:
    #         - BACKEND_PORT=5000
    #         - BALANCE=roundrobin
```

Algunos comandos para gestionar de Docker-compose son:

- **`docker-compose up`** Inicia los microservicios definidos en **docker-compose.yml**.
- **`docker-compose up -d`** Inicia los microservicios definidos en **docker-compose.yml** y correr los **contenedores** en el background.
- **`docker-compose build`** Construye desde 0 los microservicios si se realizó algún cambio.
- **`docker-compose logs`** Muestra todos los logs de todos los servicios.
- **`docker-compose logs [nombre del microservicio]`** Muestra los logs de un servicio en particular.

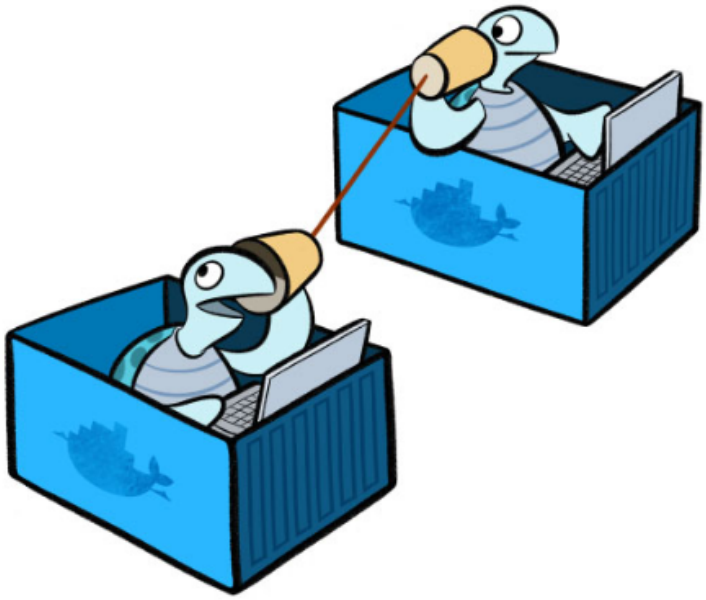
- **`docker-compose rm [microservicio]`** Permite eliminar un servicio, siempre y cuando este se encuentre detenido.
- **`docker-compose stop`** Permite detener todos los microservicios que se encuentren en ejecución.
- **`docker-compose down`** Detiene todos los microservicios y los elimina que se encuentren en ejecución.
- **`docker-compose help`** Muestra un menú de ayuda con todos los comandos de **`docker-compose`**
- **`docker-compose -f [docker-compose].yml [comando]`** Permite usar un archivo de configuración de docker-compose.yml con otro nombre.

En una arquitectura de microservicios, generalmente tenemos la ventaja de poder escalar un servicio específico para soportar la carga.

- **`docker-compose scale [servicio]=[instancias de escalar]`** Permite escalar servicios (mapeados en puertos distintos).

Redes multi-host y Docker Swarm

Los contenedores que corren en diferentes hosts no pueden comunicarse entre sí a menos que expongan sus puertos mediante el host. Multi-host networking habilita a que contenedores en diferentes hosts puedan comunicarse sin necesidad de exponer sus puertos.



El **Docker** engine permite realizar multi-hosts networking nativamente mediante el **driver overlay** de red.

Docker swarm permite formar clusters de **Docker** hosts y decide dónde lanzar los **contenedores** (scheduler).

Algunas de sus características son:

- Presenta varios hosts de **Docker** como si fueran uno solo.
- Permite distribuir **contenedores** por varios equipos dentro del cluster.
- Utiliza la API standard de **Docker**.
- Incluye una política simple de programación de **contenedores** y descubrimiento.

Cuando se crea una nueva red en una maquina que no pertenece a un **swarm** la red por defecto es **bridge**, en cambio cuando la maquina pertenece a un **swarm** la red por efecto es **overlay**.

Nombre

Erik Ochoa • hace 10 meses

Que buen resumen amigo, me lo guardo.
Buen trabajo!

1 • Responder • Compartir ›

Joaquin Araujo Moderador → Erik Ochoa • hace 10 meses

Me alegro que te haya servido, disfrutala, compartela y si ves algo que se puede mejorar no dudes en mandarme tu pull request.

• Responder • Compartir ›

Jeison Higuita Sanchez • hace 2 meses

bro, esta muy bueno, para mis marcadores

• Responder • Compartir ›

Oscar Henríquez • hace 8 meses

excelente material Joaquin! gracias por compartirlo y sigue así.

• Responder • Compartir ›

Luis Ramon Garcia Meneses • hace 10 meses

Voy a buscar un tutorial que me actualize sobre el concepto de docker, ando bien perdido, pero como dice el dicho nadie nacio aprendido, te agradezco Juaquin si puedes sumininistrarme un tutorial muy sencillo para iniciarme, dias atras no entendia nada de deploy, hoy ya manejo algo el tema, espero en pocos dias avanzar en docker, de momento tome el curso introductorio de linux y luego voy a ver sobre administracion de sistemas linux, Soy de Venezuela Juaquin espero tu ayuda

• Responder • Compartir ›

Joaquin Araujo Moderador → Luis Ramon Garcia Meneses • hace 10 meses

Hola Luis, Docker no es dificil solo que al principio es un poco "complicado de entender", te animo a que hagas el curso de CLI y Administración de Servidores Linux y luego entres con el de Docker, ya verás que cuando domines Docker lo comenzarás a implementar en todos tus proyectos. Aquí estará la guía para que repases y recuerdes los comandos más puntuales.

• Responder • Compartir ›

TAMBIÉN EN **JOAQUIN ARAUJO**

Acerca del autor

Hola, mi nombre es **Joaquin Araujo**, soy Venezolano, natal de la cálida y soleada ciudad de **Maracaibo**. Soy de los que apuesta por **JavaScript** porque sencillamente **¡JavaScript se comió al mundo!** Me enfoco más al área del backend (con **Nodejs**).

No me limito a aprender cosas nuevas ya que siempre ando en constante aprendizaje. **Me gusta compartir lo que aprendo** es por ello que realicé esta guía.

Si quieres comunicarte conmigo puedes hacerlo a través de mi cuenta en twitter **@JoaquinAraujoJS (https://twitter.com/JoaquinAraujoJS)**. Estoy abierto a cualquier proyecto **¡con ganas de cambiar al mundo!**

Esta guía fue realizada por Joaquin Araujo (https://twitter.com/JoaquinAraujoJS) y es un resumen del Curso de Docker (https://platzi.com/cursos/docker/) de Platzi (https://platzi.com)