DOCKER Y DOCKER SWARM

ADMINISTRACIÓN DE SISTEMAS INFORMATICOS EN RED

IES GONZALO NAZARENO, DOS HERMANAS (SEVILLA)

EVARISTO GARCÍA ZAMBRANA

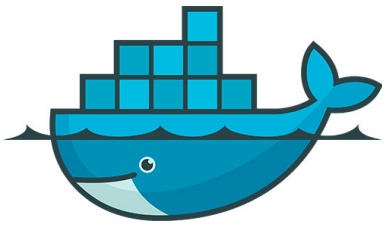
JUNIO 2017

ÍNDICE DE CONTENIDO

1. DOCKER

1.1 ¿QUÉ ES DOCKER?

"Build, Ship, and Run Any App, Anywhere" es el eslogan que Docker ha decido usar para resumir su función. Su significado en español dice: "Construye, envía y ejecuta cualquier aplicación, en cualquier lugar."



Docker es una plataforma de software que permite crear, probar e implementar aplicaciones de manera rápida. Esta plataforma automatiza el despliegue de aplicaciones dentro de contenedores que tienen como principal característica su ligereza y portabilidad.

De esta manera, las aplicaciones pueden ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina anfitriona tenga.

Los despliegues están enfocados a una arquitectura compuesta de microservicios, de esta manera se prevé que cada contenedor sea un servicio. Estos se comunican entre ellos a través de llamadas formando el conjunto de una aplicación.

Esta independencia supone una ventaja a la hora de administrar y escalar la aplicación: un cambio en un contenedor no afectará al resto y podrá escalarse únicamente el servicio que demande más recursos.

1.2 DOCKER COMMUNITY EDITION VS DOCKER ENTERPRISE EDITION

http://babel.es/es/blog/blog/marzo-2017/que-version-de-docker-necesito

De manera casi paralela con el cuarto aniversario de Docker, en marzo de 2017, se anunció la división de Docker en dos plataformas:

- Docker Engine pasaría a llamarse Docker Community Edition (CE)
- Docker Datacenter / Docker Engine CS (Commercial Supported) pasaría a llamarse Docker Enterprise Edition (EE), que consta de tres niveles de soporte.

A su vez, adopta un cambio en versionado de sus productos, que era simplemente numeral y ahora pasa a basarse en el año y mes de lanzamiento de manera similar a como lo hace Canonical en Ubuntu. De esta manera, se pasó de la versión v1.13 de Docker a la versión v17.03 correspondiente al mes de marzo de 2017.

Más que un cambio estético se debe a una asociación con el modelo de soporte y mantenimiento que Docker quiere ofrecer. La versión de la API se mantiene como lo venía haciendo hasta entonces.

La versión CE tiene dos variantes:

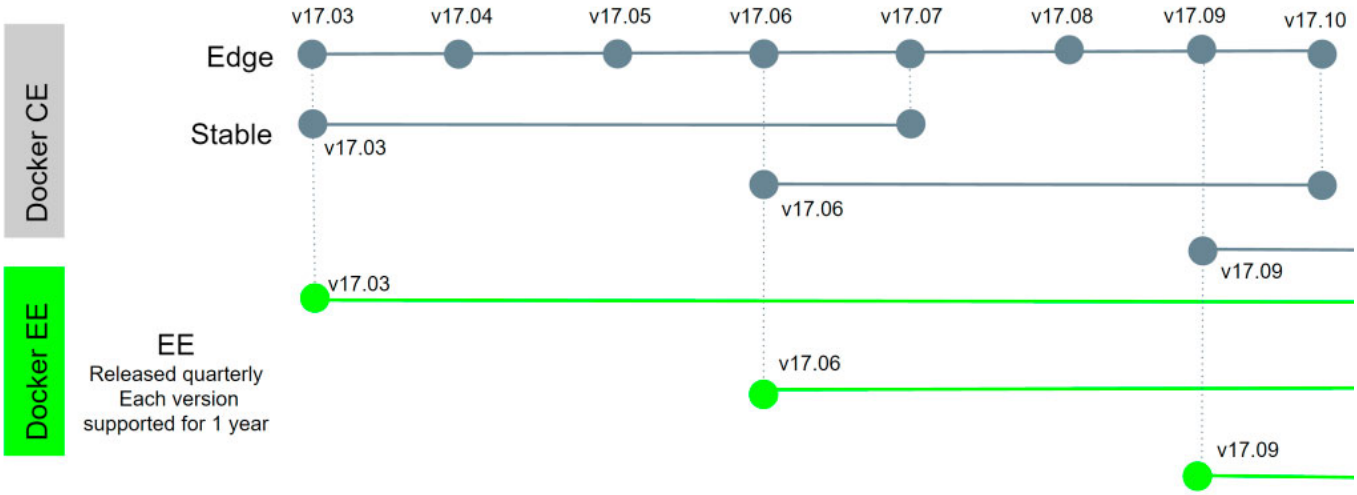
- Edge: Liberación mensual y en la que solo se proporciona soporte a bugs y fallos de seguridad durante el mes en curso.
- Stable: De liberación trimestral con cuatro meses de soporte.

Docker EE también asume el mismo ritmo de liberación que la versión Stable de Docker CE, pero el soporte se garantiza durante un año.

1.2.1 Más sobre Docker Enterprise Edition

Anteriormente el soporte que se ofrecía por parte de Docker estaba demasiado difuso conforme a quien debía darlo: el proveedor, los desarrolladores... Para simplificar, se han diseñado tres planes de soporte: básico, estándar y avanzado.

La licencia de cualquiera de estos soportes es mensual y se factura por nodo.



Además, para obtener la versión Enterprise Edition es necesario que la infraestructura esté dentro de las certificadas. Por lo que existe un programa de certificación para infraestructura, contenedores y plugins. De esta manera se busca asegurar que las herramientas de Docker funcionen sin problemas y que existe una conexión directa entre proveedor y fabricante que permita resolver incidencias y compartir información con agilidad.

1.3 CARACTERÍSTICAS

Docker ofrece a administradores de sistemas y desarrolladores una forma fácil de gestionar el proceso de integración entre entornos. Algunas de sus características más destacadas son:

1.3.1 Ligereza

El tamaño de los contenedores Docker es ligero como ningún otro sistema de virtualización convencional lo es. Un contenedor Docker puede ser funcional ocupando incluso menos de 100MB.

1.3.2 Portabilidad

Un contenedor Docker puede desplegarse en cualquier otro sistema que soporte Docker. Su función permite empaquetar los servicios (incluyendo bibliotecas y dependencias) en contenedores. De esta manera los pases a producción no suponen grandes problemas, ya que utilizándose el esquema de microservicios permite portar un único servicio.

1.3.3 Integración

Docker puede integrarse con múltiples herramientas para infraestructura como servicio (IaaS). Así pues, los principales proveedores de esta tecnología ofrecen una gestión de contenedores Docker que facilita el uso de los mismos en su plataforma, así como los pases a producción.

Estas características, entre otras, hacen que sea más fácil implementar cambios en una aplicación, mejore la productividad de desarrollo, la integración entre varios entornos y la experiencia positiva del equipo de desarrollo, los administradores de sistemas y los clientes y/o usuarios.

1.4 COMPONENTES Y CONCEPTOS DE DOCKER

1.4.1 Docker Engine / Docker CE

Es el demonio de Docker que se ejecuta en el sistema operativo. El usuario nunca interactúa sobre él directamente, si no que lo hace a través de Docker Client, que es quien intermedia entre ambos.

Expone una API para la gestión de imágenes y contenedores, lleva a cabo la creación y/o ejecución de imágenes, publica o descarga desde un Docker Registry, etc.

Además, es el componente encargado de la gestión de contenedores que se encuentran en ejecución.

1.4.2 Docker Client

Podría llamarse cliente Docker a cualquier herramienta que haga uso de la API de Docker Engine, pero normalmente se hace referencia al comando `docker` que interactúa con Docker Engine.

Este cliente permite ser configurado para mandar las interacciones con un Docker Engine local o remoto, siendo posible gestionar un entorno de desarrollo local o servidores de producción.

Dispone de una gran cantidad de comandos que permiten gestionar los contenedores, las imágenes, los volúmenes y las redes: `docker info`, `docker build`, `docker run`, `docker ps`, `docker logs`... son comandos que forman parte de Docker Client.

1.4.3 Docker Compose

Docker Compose permite desplegar y administrar aplicaciones compuestas por varios contenedores relacionados entre sí. Esto se hace definiendo en un único archivo los contenedores que forman parte de una aplicación, permitiendo desplegarla utilizando la arquitectura de despliegue mediante microservicios.

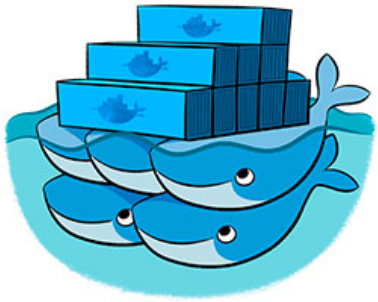
1.4.4 Docker Machine

Este componente ofrece la posibilidad de crear y configurar máquinas virtuales o nodos físicos para alojar dentro de ellas contenedores. Hace posible construir máquinas virtuales en las plataformas de IaaS (Infraestructura como Servicio) más populares cómo AWS, GCE, Azure, DigitalOcean, OpenStack... o bien en plataformas de virtualización de escritorio como VMware Fusion o VirtualBox.

A la hora de crear un nodo instala boot2docker en él, una distribución muy liviana del sistema operativo Linux (Tiny Core Linux) que incluye Docker Engine en su interior. Su tamaño es de ~27MB y además es el encargado de crear las claves SSH, iniciar el nodo, gestionar la red, crear y copiar los certificados TLS... entre otras tareas que realiza por cada nodo.

Así pues, Docker Machine permite utilizar nuestro Docker Client para administrar otros Docker Engine remotos que hayan sido configurados con esta herramienta.

1.4.5 Docker Swarm



Si has escuchado hablar sobre Kubernetes de Google o Mesos de Apache, Docker Swarm es la tecnología nativa que cumple prácticamente con la misma función. Se trata de una herramienta de orquestación de contenedores.

De más reciente creación que Kubernetes, este se trata de un componente que permite realizar un clúster de nodos al que se pueden enviar contenedores Docker para su ejecución. En ocasiones se utiliza conjuntamente con Docker Machine para crear los nodos a utilizar y se combina con Docker Compose para desplegar los contenedores.

Cuando se trata de un servicio se realiza a través de un stack de servicios.

En Docker Swarm existen dos tipos de nodos: Manager y Worker.

1.4.5.1 ¿Qué es un nodo de Swarm?

Un nodo es una instancia de Docker Engine que participa en un Swarm. También puede decirse que se trata de un nodo de Docker.

Estos pueden ejecutarse en nodos físicos o virtualizados, como servidores cloud o plataformas de virtualización de escritorio como VirtualBox.

1.4.5.2 Nodos Manager

Los nodos Manager (también llamados Master) son los encargados de gestionar el clúster. Puede, y es recomendable, que exista más de un nodo Manager. Docker recomienda que en producción, el número de Managers sea impar para soportar una tolerancia a fallos.

La principal tarea de estos tipos de nodos es mantener un estado correcto del clúster, así como de los servicios que existen en cada contenedor, recreando si alguno de ellos deja de funcionar.

También son los encargados de distribuir las tareas entre los nodos Workers, que son quienes reciben estas tareas y las ejecutan.

1.4.5.3 Nodos Worker

Los nodos de trabajo o Workers son instancias de Docker Engine cuyo único propósito es ejecutar contenedores.

Se puede crear un Swarm con un solo nodo Manager, pero no puede ejecutarse un nodo Worker sin al menos un nodo Manager. Por defecto, los Manager también actúan como Worker, aunque esto es configurable para que solo asuman tareas de Manager.

De la misma manera, puede elevar un nodo Worker a Manager o viceversa. Haciendo que los nodos asuman roles distintos a su actual.

1.4.6 Docker Service

Para desplegar una imagen de una aplicación en Docker Swarm, se crea un servicio. De manera común, el servicio es parte de una aplicación más grande, por lo que se considera microservicio y cada uno de ellos será contenedor de Docker.

Cuando desplegamos un servicio en el clúster de Docker Swarm, el Manager acepta la definición del servicio y su estado. Entonces, se programa el despliegue de tareas en nodos, pudiendo ser replicado en un nodo o más.

El servicio se ubica en el Docker Swarm Manager, que lo despliega en nodos a través de tareas, cada una de ellas invoca a un contenedor con una imagen, que será la misma para todos.

Services-diagram.png

Si el contenedor no es capaz de mostrar su estado de salud o termina, la tarea termina, reconociendo el orquestador (el Manager) que esa tarea ha terminado.

Al hablar de servicios, la terminología correcta para referirse a cada réplica no es contenedores si no tareas. Estas tareas son desplegadas en contenedores.

1.4.6.1 Docker Stack

Cuando son varios servicios los que se ejecutan, se denomina un stack. Estos son definidos en un archivo en formato YAML (con extensión .yml) muy similar a un Docker Compose.

La diferencia con Docker Compose es que en él se definen un conjunto de contenedores que forman un servicio, mientras que en un Docker Stack se definen otras secciones como services, networks, volumes, secrets...

Existe un servicio online llamado Stackfiles.io en el que los usuarios almacenan y comparten stacks. Este proyecto fue creado por Tutum, una startup que fue adquirida por Docker y de la que se lanzó Docker Cloud, por ello permite desplegar directamente desde Stackfiles.io a Docker Cloud.

1.4.7 Docker Secrets

Se trata de una función de Docker Swarm que permite manejar datos sensibles en contenedores Docker.

Este tipo de información puede ser, por ejemplo, usuarios y contraseñas, claves SSH privadas, certificados SSL, nombre de una BBDD u otro tipo de información que no debe ser transmitida a través de una red o almacenada sin cifrar en un Dockerfile.

Los secretos de Docker se mantienen de manera cifrada. Cuando se concede el acceso, el secreto descifrado se monta en un sistema de archivos en memoria, volviéndose a desmontar cuando la tarea deja de utilizar ese secreto. Se proporcionan únicamente a los contenedores que los necesitan, enviándolo de forma cifrada a través de la red al nodo en el que se ejecuta ese contenedor.

Se puede referenciar en Docker Swarm o Docker Compose, su contenido puede ser binarios o texto plano siempre que no superen los 500Kb.

Otro caso de uso es en escenarios de desarrollo, prueba y producción. Los contenedores solo necesitan saber el nombre del Docker Secrets para funcionar en los tres entornos.

1.4.8 Docker Hub / Docker Store

Docker Hub es un repositorio de imágenes Docker. Esto permite compartir las imágenes construidas por los usuarios, así como las empresas, lo que facilita el lanzamiento de aplicaciones evitando tener que construir la tuya propia mediante un Dockerfile.

Permite crear repositorios públicos, a los que puede acceder cualquier usuario, o repositorios privados donde alojar de manera privada las imágenes de Docker. Los repositorios públicos son ilimitados, mientras que los repositorios privados son de pago, siendo el primer repositorio privado gratuito.

En él no se alojan los Dockerfile, si no que se hace directamente las imágenes Docker. Aún así, permite construir imágenes a través de Dockerfiles alojados en GitHub o Bitbucket, bien estén alojados en repositorios privados o repositorios públicos. Por lo tanto, es Docker Hub quien se encarga de obtener los Dockerfiles y construir la imagen.

Podemos decir que Docker Hub es el GitHub de los contenedores Docker.

1.4.9 Docker Registry

El Docker Registry es un repositorio de imágenes Docker privado y seguro. Es un Docker Hub que forma parte de nuestra arquitectura como un componente y que normalmente no suele estar en el mismo equipo en el que se está trabajando, si no en un servidor. De esta manera podemos almacenar las imágenes de los contenedores de forma privada.

Puede suponer ventajas para una entidad tanto en privacidad como en seguridad, además de en velocidad a la hora de descargar estos registros o ahorro de costes conforme al precio de repositorios privados.

1.4.10 Docker Network

Docker crea por defecto tres tipos de redes:

- docker0, una interfaz de tipo puente/bridge. Es la red que utilizan por defecto todos los contenedores y visible a través de nuestra máquina con `ifconfig`. Los contenedores recibirán direcciones IP del mismo rango.
- host, para enlazar varios contenedores.
- none, para aislar un contenedor dejándolo sin ninguna red.

Con Docker Network podemos gestionar y administrar las redes de Docker, al igual que crear nuevas redes de diferentes tipos y gestionar su asociación con los contenedores, de manera que puedan aislarse contenedores según su red.

1.4.11 Dockerfile

Dockerfile es un archivo que reconoce Docker y que, a través de una serie de instrucciones, es capaz de automatizar la creación de un contenedor Docker. En este archivo se agrega todo lo que necesitamos en nuestro contenedor, siendo este archivo el que sufra las modificaciones y no el contenedor en sí.

Podemos decir que se trata de un "script" o una "receta" que permite automatizar la creación del contenedor. Igualmente sirve para crear imágenes de Docker, no siendo necesario el archivo para poder hacer uso de estas imágenes y crear contenedores con ellas.

1.4.12 Imagen de Docker

Una imagen de Docker es, por así decirlo, un Dockerfile "compilado". Son utilizadas para crear contenedores y pueden ser compartidas, almacenadas y actualizadas a través Docker Hub o Docker Registry. De esta modo podríamos descargar una imagen preparada para hacer funcionar un servicio en concreto.

Estas imágenes están construidas a través de capas, por lo que su formato de capas construye paso a paso la imagen siguiendo instrucciones: actualizar repositorios y paquetes, ejecutar una instalación, reemplazar un fichero, abrir un puerto...

1.4.13 Docker Cloud

Docker Cloud es un servicio web que permite administrar repositorios de Docker y proporciona la posibilidad de crear y testear fácilmente aplicaciones dockerizadas.

Además, esta herramienta tiene integración con los repositorios de imágenes de Docker Hub así como autobuild al detectar nuevos push en repositorios de GitHub o BitBucket.

En fase beta permite desplegar Docker Swarm en los servicios de Amazon Web Services o Microsoft Azure a través de integración con sus APIs. Esto lo hará con la edición de Docker Community Edition.

Por último, permite crear equipos y organizaciones para que cada miembro de ese grupo tenga distintos roles o niveles de permisos, incluso facturaciones distintas.

Ofrece también un servicio de pago llamado Docker Security Scanning

que escanea las imágenes de repositorios privados para verificar que está libre de vulnerabilidades o riesgos de seguridad, reportando los resultados.

1.4.14 Docker Toolbox

Disponible para Windows y Mac, Docker Toolbox instala Docker Client, Docker Machine, Docker Compose y Kitematic, una interfaz gráfica para Docker.

A su vez instala la distribución boot2docker, que incluirá el núcleo reducido de Linux y Docker Engine sobre el que poder correr contenedores Docker y otras funciones.

1.4.15 Docker Captains

El concepto de Docker Captains no se trata de ningún componente de Docker sino de un grupo de personas que reciben un nombramiento o distinción especial gracias a sus conocimientos de Docker.

Así pues, un Docker Captain es un experto en Docker y líder en comunidades que demuestran un compromiso por compartir sus conocimientos con los demás usuarios de Docker.

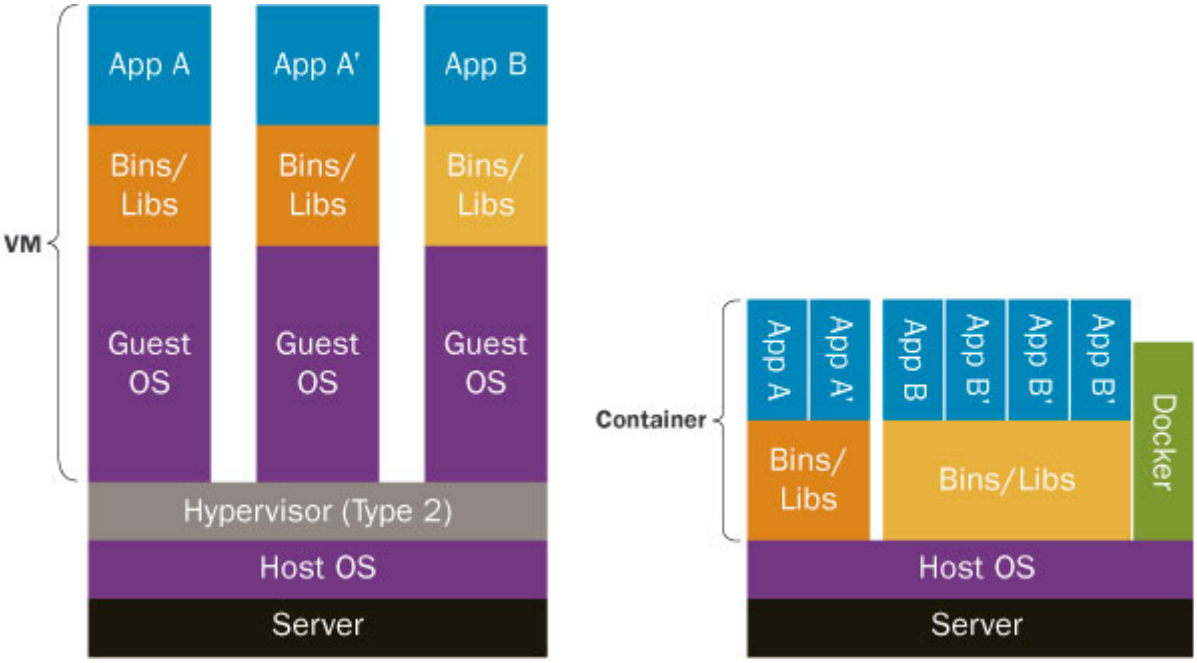
Los capitanes no son empleados de Docker, aún así tienen un gran compromiso e impacto en la comunidad.

Estos publican de forma activa en el blog oficial, hablan sobre Docker en eventos, desarrollan materiales o realizan meetups. También se prestan a dar soporte a miembros de la comunidad a través de sus redes sociales.

1.5 MÁQUINAS VIRTUALES VS DOCKER

Aunque en principio las similitudes pueden ser muchas, las máquinas virtuales y Docker tienen mucho que diferir en cuanto a su forma de trabajar.

Las máquinas virtuales precisan de un hipervisor que actúe sobre el hardware físico. De este modo, es este hipervisor quien realiza la comunicación con el sistema operativo. Además, necesitan un sistema operativo completo para emularlo, de manera que no comparten nada con el resto de máquinas virtuales y conlleva un mayor gasto de recursos.



En cambio, los contenedores Docker funcionan con un único kérnel (el de la máquina real) y comparten un solo sistema operativo para los contenedores. Sobre esa imagen base se añaden capas con lo necesario para nuestro contenedor: paquetes, librerías, dependencias... Así se genera una imagen de Docker, la cual necesitará la base y las capas cada vez que quiera ejecutarse nuevamente.

De esta manera, el tamaño de los contenedores es mucho menor, haciendo que sea más fácil gestionarlos entre entornos.

Quizás uno de los niveles donde peca Docker es en el nivel de seguridad que puede ofrecer respecto a las máquinas virtuales, pues estas últimas tienen un entorno mucho más aislado conforme al hardware.

1.6 AUTOMATIZACIÓN EN DOCKER

La automatización es un plus a la hora de administrar Docker, ya que permite realizar tareas repetitivas, recrear escenarios con facilidad o realizar tareas de forma automatizada según condiciones.

En un principio puede parecer innecesario el uso de herramientas como Ansible o Puppet, o scripts escritos en Python o Bash. Pero conforme tu entorno va cogiendo más y más magnitud, quizás interese que todo esté automatizado a través de alguna de las herramientas.

Sus usos pueden ser múltiples: desplegar contenedores, desplegar nodos de Docker Machine, desplegar nodos de Docker Swarm, actualizar contenedores, hacer pases a producción...

2. DOCKER ENGINE / DOCKER CE

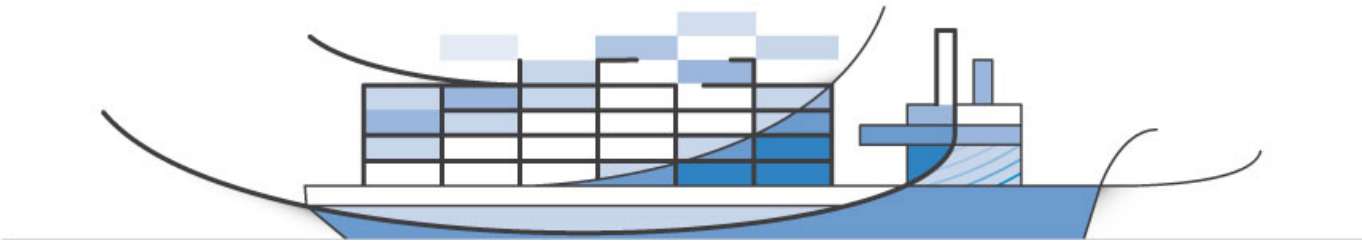
Algunas de las herramientas que voy a utilizar y por tanto, instalar, están incluidas en Docker Toolbox (disponible para Windows y Mac). Este paquete incluye Docker Client, Docker Machine, Docker Compose y Kitematic.

Todas son instalables por separado, que es así como lo haré en mi sistema. Otras, como por ejemplo Docker Swarm, no están disponibles a través de Docker Toolbox.

Como bien he explicado Docker está disponible en dos versiones: Community Edition y Enterprise Edition.

Para instalar Docker en nuestro sistema operativo debemos saber qué versiones están soportadas en él y si cumple con nuestros requisitos o fines de uso. Esto podemos consultarlo en la página web oficial del software: https://docs.docker.com/engine/installation/

En el caso de Debian está disponible Docker Community Edition para sistemas de 32 y 64bits, además de para ARM. Docker Enterprise Edition no está disponible para Debian de momento.



En primer lugar, instalamos los paquetes para poder utilizar repositorios que sirven a través de HTTPS: `apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common`

Añadimos la clave GPG del repositorio con la ejecución del siguiente comando: `curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -`

Y añadimos también el repositorio: `add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"`

Por último, actualizamos los repositorios y ejecutamos su instalación con `apt install docker-ce` que en nuestro caso será la versión 17.03 la que se instale. Comprobamos su correcto funcionamiento lanzando un contenedor de prueba cuya imagen es hello-world.

Comprobamos la versión de Docker CE instalada y como demo, ejecutamos una imagen hello-world.

```
nano@satellite:~$ docker --version
Docker version 17.03.1-ce, build c6d412e
nano@satellite:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

```
Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

2.1 EJECUTAR DOCKER ENGINE CON UN USUARIO SIN PRIVILEGIOS

Si deseamos ejecutar Docker con un usuario sin privilegios, como en el caso del anterior ejemplo, debemos añadir dicho usuario al grupo docker. En nuestro caso se trata del usuario nano y lo hacemos con el lanzamiento del siguiente comando `usermod -aG docker nano`

Una vez añadido, reiniciamos el demonio de Docker con `/etc/init.d/docker restart` para que los cambios surtan efecto en nuestro sistema.

2.2 GESTIÓN GENERAL

Por lo general, muchos de los componentes o funcionalidades tienen opciones en común. Estas pueden ser listar, eliminar, parar, mostrar información...

Igualmente, existen comandos que cumplen la misma función. El ejemplo más claro puede ser la gestión de contenedores, donde el uso de la palabra *container* se suprime, siendo posible ejecutarlos sin él en su mayoría.

Es probable que algunas de las opciones no estén disponibles para todos los componentes, por lo que es recomendable utilizar *--help* para aprovechar las funciones de estos comandos.

2.2.1 Uso de ls

Con *ls* mostraremos la lista y sus columnas de propiedades.

```
nano@satellite:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              a41a7446062d        2 weeks ago         3.97 MB
hello-world         latest              48b5124b2768        5 months ago        1.84 kB
training/webapp     latest              6fae60ef3446        2 years ago         349 MB
nano@satellite:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
50a374a08fe7        bridge              bridge              local
65a3bdd7d613        docker_gwbridge     bridge              local
afcd5ca5b6d4        host                host                local
qfrp240zubh6        ingress             overlay             swarm
b37fc92cd80a        none                null                local
```

Si utilizamos el parámetro *-f* o *--format* podemos realizar filtros de tipo clave=valor, de manera que se muestre el contenido con el formato que deseemos.

```
nano@satellite:~$ docker image ls --format "{{.ID}}: {{.Repository}}"
a41a7446062d: alpine
48b5124b2768: hello-world
6fae60ef3446: training/webapp
nano@satellite:~$ docker network ls --format "{{.ID}} tiene como nombre {{.Name}} y es de
tipo {{.Scope}}"
50a374a08fe7 tiene como nombre bridge y es de tipo local
65a3bdd7d613 tiene como nombre docker_gwbridge y es de tipo local
afcd5ca5b6d4 tiene como nombre host y es de tipo local
qfrp240zubh6 tiene como nombre ingress y es de tipo swarm
b37fc92cd80a tiene como nombre none y es de tipo local
```

Si hacemos uso de *-q* nos mostrará únicamente el ID, de manera que podamos usarlo directamente para gestionar ese elemento.

```
nano@satellite:~$ docker image ls -q
a41a7446062d
48b5124b2768
6fae60ef3446
nano@satellite:~$ docker network ls -q
```

```
50a374a08fe7
65a3bdd7d613
afcd5ca5b6d4
qfrp240zubh6
b37fc92cd80ª
nano@satellite:~$ docker ps -q
1dbfdab75fe6
ad25b1192178
nano@satellite:~$ docker ps -aq
1dbfdab75fe6
ad25b1192178
369bb0151b54
```

2.2.2 Uso de inspect

Inspect es uno de los comandos que más información puede ofrecer de un elemento de Docker. Este muestra, en formato JSON, la información más completa de casi cualquier elemento. Además, permite parsear sus resultados, extrayendo lo que nos resulte importante para tratarlo a través de otra serie de programas.

Para parsear con *inspect* debemos especificar el formato con *–format*

```
nano@satellite:~$ docker inspect -f {{.Config.Image}} pingdocker
alpine
nano@satellite:~$ docker network inspect -f {{.Created}} bridge
2017-06-10 09:07:58.741761207 +0000 UTC
nano@satellite:~$ docker node inspect -f {{.Status.State}} prueba
ready
```

2.2.3 Uso masivo

A través de la combinación de dos comandos, podemos gestionar de forma masiva nuestros componentes o elementos de Docker a través de sus ID que extraemos con *-q*

Por ejemplo, podríamos borrar todos los contenedores, activos e inactivos, con el comando `docker stop $(docker ps -aq)` para parar los contenedores y luego `docker rm $(docker ps -aq)` para eliminarlos.

O borrar todas las imágenes de Docker con el comando `docker rmi $(docker images -q)`

Al igual que también podemos iniciar todos los nodos de Docker Machine con `docker-machine start $(docker-machine ls -q)` o eliminarlos con `docker-machine rm $(docker-machine ls -q)`

2.3 GESTIÓN DE CONTENEDORES

Por defecto, a los contenedores se les asigna un ID de contenedor y un nombre aleatorio, compuesto por dos palabras separadas por un guión bajo como por ejemplo: *pensive_booth*. Esto significa que podemos realizar las ejecuciones de comandos sobre ese contenedor haciendo referencia a él de ambas maneras.

Puede consultar la ayuda de Docker escribiendo únicamente en la terminal: `docker container`

```
nano@satellite:~$ docker container

Usage: docker container COMMAND

Manage containers

Options:
      --help   Muestra su uso

Commands:
  attach      Une a un contenedor que esté corriendo
  commit      Crea una nueva imagen a partir de los cambios de un contenedor
  cp          Copia ficheros/directorios entre el contenedor y el sistema de ficheros local
```

```
  create      Crea un nuevo contenedor
  diff        Muestra los cambios de ficheros/directorios en los sistemas de ficheros del
contenedor.
  exec        Ejecuta un comando en un contenedor que esté corriendo
  export      Exporta el sistema de ficheros de un contenedor a un fichero tar
  inspect     Muestra información detallada de uno o más contenedores
  kill        Mata uno o más contenedores que estén corriendo.
  logs        Busca el log del contenedor
  ls          Lista los contenedores
  pause       Pausa todos los procesos dentro de uno o más contenedores
  port        Lista el mapeo de puertos o un específico mapeo de un contenedor
  prune       Elimina todos los contenedores parados.
  rename      Renombra un contenedor
  restart     Reinicia uno o más contenedores
  rm          Elimina uno o más contenedores
  run         Ejecuta un comando en un nuevo contenedor
  start       Inicia uno o más contenedores parados
  stats       Muestra en estado real las estadísticas de uso de recursos de un contenedor
  stop        Para uno o más contenedores que estén corriendo
  top         Muestra los procesos en ejecución de un contenedor
  unpause     Vuelve a iniciar todos los procesos dentro de uno o más contenedores
  update      Actualiza la configuración de uno o más contenedores
  wait        Espera hasta que uno o más contenedores se paren, luego muestra su código de
salida

Run 'docker container COMMAND --help' for more information on a command.
```

2.3.1 Ejecutar o crear un nuevo contenedor

Lanzar una imagen de Hello-World que muestra una breve info y el correcto funcionamiento de Docker.

```
nano@satellite:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Already exists
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

2.3.1.1 Ejecutar con un nombre definido

La ejecución de un contenedor con un nombre definido nos permitirá gestionar de manera más ágil nuestros contenedores. Aún así, también podremos gestionarlo con el ID o el nombre aleatorio que se le asigna a cada uno de ellos.

```
nano@satellite:~$ docker run --name helloworld hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
```

```
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

2.3.1.2 Ejecutar en segundo plano

La ejecución en segundo plano de un contenedor hará que únicamente se muestre el ID del contenedor, en vez de la salida por pantalla. Este ejemplo es claro con la imagen de Hello-World.

```
nano@satellite:~$ docker run -d hello-world
2920563b75115a423a5489cad0b66a2702ae8bb9aa664e2e95e95a26fc90d72f
```

Es lo habitual cuando se pretende lanzar servicios que van a estar ejecutándose durante toda la vida del contenedor.

2.3.1.3 Ejecutar con exposición de puertos

La exposición de puertos permite que un contenedor exponga un servicio a través de un puerto y lo asocie a Inicia un contenedor con un servidor web Nginx haciendo una exposición del puerto 80 al 8080 de la máquina anfitrión.

```
nano@satellite:~$ docker run -d -p 8080:80 nginx
020cece2c065de3bacf1d25da5170d5bb34da4243f6bc1172575c9988f019f36
```

2.3.1.4 Ejecutar con una terminal interactiva

Lanzar un nuevo contenedor con una imagen de Debian Jessie y acceder a él mediante una terminal interactiva.

```
nano@satellite:~$ docker run -it debian:jessie /bin/bash
Unable to find image 'debian:jessie' locally
jessie: Pulling from library/debian
10a267c67f42: Pull complete
Digest: sha256:476959f29a17423a24a17716e058352ff6fbf13d8389e4a561c8ccc758245937
Status: Downloaded newer image for debian:jessie
root@702137448a18:/#
```

2.3.2 Ver todos los contenedores

Para ver los contenedores activos utilizamos el comando `docker container ls` o su versión corta `docker ps` Si deseamos ver también los que no están en activo debemos utilizar `docker container ls -a` o `docker ps -a`

```
nano@satellite:~$ docker run -d --name pingdocker alpine ping docker.com
5d21d6d9dc47c004b2af4051e49e2d5c599bbe75b8350d575f38e3bb9cc7760b
nano@satellite:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

```
To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

nano@satellite:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
5d21d6d9dc47        alpine              "ping docker.com"   19 seconds ago      Up 19 seconds
pingdocker
nano@satellite:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
92bffc98e5d0        hello-world         "/hello"            6 seconds ago       Exited (0) 6
seconds ago                       kind_mayer
5d21d6d9dc47        alpine              "ping docker.com"   21 seconds ago      Up 21 seconds
pingdocker
```

También podemos utilizar otros argumentos como *-n 5* para ver los últimos cinco contenedores creados o *-l* para ver el último contenedor creado.

2.3.3 Ver información detallada de un contenedor

Con la opción *inspect* se extrae la toda la información de un contenedor. Este comando puede realizarse con `docker container inspect prueba` o `docker inspect prueba`

```
nano@satellite:~$ docker inspect 27b1af977c67
[
    {
        "Id": "27b1af977c67e16bba18d6001b0225686d7f1300656003558a9e19d8b213b15e",
        "Created": "2017-05-11T06:29:29.028880814Z",
        "Path": "/hello",
        "Args": [],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2017-05-11T06:29:29.324610741Z",
            "FinishedAt": "2017-05-11T06:29:29.3940505492"
        },
        "Image": "sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233",
(...)
```

Debido a que *inspect* devuelve el resultado en formato JSON podemos parsearlo y extraer únicamente la información que necesitamos.

```
nano@satellite:~$ docker inspect -f {{.Config}} kind_mayer
{92bffc98e5d0   false true true map[] false false false
[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin] [/hello] <nil> false
hello-world map[]  [] false  [] map[]  <nil> []}
nano@satellite:~$ docker inspect -f {{.Config.Image}} kind_mayer
hello-world
```

2.3.4 Acceder a la terminal de un contenedor

Para ejecutar una terminal interactiva al lanzar un nuevo contenedor Docker lo hacemos con: `docker run -it debian:latest /bin/bash`

Si necesitamos acceder a la terminal de un contenedor podemos hacerlo conla opción *exec*. En nuestro caso: `docker exec pingdocker sh`

```
nano@satellite:~$ docker exec -it pingdocker sh
/ # echo "Esto se ejecuta en el container"
Esto se ejecuta en el container
/ # exit
```

2.3.5 Ejecutar un comando en un contenedor

Una vez que un contenedor está corriendo podemos ejecutar un comando con la misma opción que en el punto anterior. A diferencia, este no abrirá una terminal interactiva, si no que lo hará y saldrá.

```
nano@satellite:~$ docker exec pingdocker touch fichero.txt
nano@satellite:~$ docker exec -it pingdocker sh
/ # ls
bin          etc          home         media        proc         run          srv
tmp          var
dev          fichero.txt  lib          mnt          root         sbin         sys
usr
/ # exit
```

Si deseamos ejecutarlo en segundo plano debemos hacerlo con el parámetro *-d*. En el siguiente ejemplo ejecutamos nuevamente un ping a docker.com, que se ejecutará como root.

```
nano@satellite:~$ docker exec -d pingdocker ping docker.com
nano@satellite:~$ docker exec -it pingdocker sh
/ # ps
PID   USER     TIME   COMMAND
    1 root       0:00 ping docker.com
   21 root       0:00 ping docker.com
   25 root       0:00 sh
   29 root       0:00 ps
/ #
```

2.3.6 Ver ejecución de procesos de un contenedor

Podemos consultar los procesos que se están ejecutando dentro de un contenedor a través de la opción *top* con `docker top pingdocker`

```
nano@satellite:~$ docker top pingdocker
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                6201                6189                0                   16:16
?                   00:00:00            ping docker.com
root                7695                7684                0                   16:45
?                   00:00:00            ping docker.com
```

En este caso, se ve la ejecución de dos procesos en segundo plano realizando un ping a docker.com.

2.3.7 Pausar o reanudar un contenedor

docker pause es un comando que permite suspender o pausar todos los procesos especificados en un contenedor.

El contenedor aparecerá como pausado y no permitirá su acceso.

```
nano@satellite:~$ docker pause pingdocker
pingdocker
nano@satellite:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
6a78fa0fd77e        alpine              "ping docker.com"   14 minutes ago      Up 14 minutes
(Paused)                        pingdocker
nano@satellite:~$ docker exec -it pingdocker sh
Error response from daemon: Container pingdocker is paused, unpause the container before exec
```

Para volver a reanudar el contenedor basta con ejecutar la opción *unpause*.

```
nano@satellite:~$ docker unpause pingdocker
pingdocker
nano@satellite:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
6a78fa0fd77e        alpine              "ping docker.com"   15 minutes ago      Up 15 minutes
```

```
pingdocker
```

2.3.8 Reiniciar un contenedor

Podemos reiniciar un contenedor en concreto con la opción *restart*. Este contenedor no podrá estar pausado, como lo hicimos en el punto *2.3.7*

```
nano@satellite:~$ docker restart pingdocker
Error response from daemon: Cannot restart container pingdocker: Container
5d21d6d9dc47c004b2af4051e49e2d5c599bbe75b8350d575f38e3bb9cc7760b is paused. Unpause the
container before stopping or killing
nano@satellite:~$ docker unpause pingdocker
pingdocker
nano@satellite:~$ docker restart pingdocker
pingdocker
```

Podemos establecer una demora en segundos antes de que se ejecute el reinicio con el parámetro *-t*: `docker restart -t 30 pingdocker`

2.3.9 Matar un contenedor

Usar la opción *kill* hará que el contenedor se pare de manera forzosa.

```
nano@satellite:~$ docker kill pingdocker
pingdocker
nano@satellite:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
6a78fa0fd77e        alpine              "ping docker.com"   21 minutes ago      Exited (137)
8 seconds ago                       pingdocker
```

Aún así, el contenedor seguirá permaneciendo en nuestra lista de contenedores, visible con `docker ps -a` Por lo que no podremos utilizar el mismo nombre de contenedor hasta que no lo eliminemos, pero sí volverlo a iniciar.

```
nano@satellite:~$ docker kill pingdocker
pingdocker
nano@satellite:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
6a78fa0fd77e        alpine              "ping docker.com"   21 minutes ago      Exited (137)
8 seconds ago                       pingdocker
```

2.3.10 Eliminar un contenedor

Para eliminar un contenedor usamos la opción *rm*. No será posible eliminar un contenedor que esté corriendo sin utilizar la opción *-f* o *--force*

```
nano@satellite:~$ docker rm pingdocker
Error response from daemon: You cannot remove a running container
5d21d6d9dc47c004b2af4051e49e2d5c599bbe75b8350d575f38e3bb9cc7760b. Stop the container before
attempting removal or force remove
nano@satellite:~$ docker rm -f pingdocker
pingdocker
nano@satellite:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

Podemos eliminar fácilmente todos los contenedores con la opción `docker rm $(docker ps -aq)`

2.4 GESTIÓN DE IMÁGENES

```
nano@satellite:~$ docker image

Usage: docker image COMMAND

Manage images

Options:
      --help   Muestra su uso

Commands:
  build       Construye una imagen desde un Dockerfile
  history     Muestra la historia de una imagen
  import      Importa el contenido desde un fichero tar para crear una nueva imagen
  inspect     Muestra información detallada de una o más imágenes
  load        Carga una imagen desde un fichero tar o STDIN
  ls          Lista las imágenes
  prune       Elimina las imágenes no usadas
  pull        Descarga una imagen o repositorio de un registry de Docker
  push        Sube una imagen o repositorio de un registry de Docker
  rm          Elimina una o más imágenes
  save        Guarda una o más imágenes en un fichero tar
  tag         Crea una etiqueta de una imagen local

Run 'docker image COMMAND --help' for more information on a command.
```

2.4.1 Construir una imagen desde un Dockerfile

A través de un fichero Dockerfile podemos crear una imagen de Docker de manera automatizada. Esto ofrece ventajas a la hora de realizar despliegues, pues los tiempos de espera se reducen.

En este caso utilizo una imagen base de Ubuntu 17.10, que se descarga desde el Docker Registry, defino que se actualicen repositorios y paquetes para luego crear esa imagen gracias a estas definiciones.

```
FROM ubuntu:17.10
MAINTAINER Evaristo GZ "webmaster@evaristogz.com"

RUN apt-get update && apt-get -y upgrade
```

Con el parámetro *-t* etiquetaremos la nueva imagen. En este caso he puesto que se trata de una versión llamada evaristogz y coja el Dockerfile que se encuentra en el directorio actual.

También puede utilizarse `docker image build .` que creará la imagen sin nombre de repositorio ni etiqueta.

```
nano@satellite:~$ docker build -t "ubuntu:evaristogz" .
Sending build context to Docker daemon  18.94kB
Step 1/3 : FROM ubuntu:17.10
17.10: Pulling from library/ubuntu
Digest: sha256:52d19954c14bbadf6a0965c4cad3da0bb052b62cae16a108add338e1838cbd72
Status: Downloaded newer image for ubuntu:17.10
 ---> c6cac97ba835
Step 2/3 : MAINTAINER Evaristo GZ "webmaster@evaristogz.com"
 ---> Using cache
 ---> 0574ff178c86
Step 3/3 : RUN apt-get update && apt-get -y upgrade
 ---> Running in 23526b1412fe
Get:1 http://archive.ubuntu.com/ubuntu artful InRelease [237 kB]
Get:2 http://security.ubuntu.com/ubuntu artful-security InRelease [65.5 kB]
Get:3 http://archive.ubuntu.com/ubuntu artful-updates InRelease [65.5 kB]
Get:4 http://archive.ubuntu.com/ubuntu artful-backports InRelease [65.5 kB]
Get:5 http://archive.ubuntu.com/ubuntu artful/universe Sources [10.9 MB]
Get:6 http://archive.ubuntu.com/ubuntu artful/multiverse amd64 Packages [189 kB]
Get:7 http://archive.ubuntu.com/ubuntu artful/universe amd64 Packages [10.8 MB]
Get:8 http://archive.ubuntu.com/ubuntu artful/restricted amd64 Packages [14.3 kB]
Get:9 http://archive.ubuntu.com/ubuntu artful/main amd64 Packages [1531 kB]
Fetched 23.9 MB in 3s (6691 kB/s)
Reading package lists...
```

```
Reading package lists...
Building dependency tree...
Calculating upgrade...
The following packages will be upgraded:
  dpkg libgcrypt20 perl-base
3 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 2882 kB of archives.
After this operation, 47.1 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu artful/main amd64 dpkg amd64 1.18.24ubuntu1 [1140 kB]
Get:2 http://archive.ubuntu.com/ubuntu artful/main amd64 perl-base amd64 5.24.1-3ubuntu1
[1343 kB]
Get:3 http://archive.ubuntu.com/ubuntu artful/main amd64 libgcrypt20 amd64 1.7.6-2 [399 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 2882 kB in 0s (4698 kB/s)
(Reading database ... 4063 files and directories currently installed.)
Preparing to unpack .../dpkg_1.18.24ubuntu1_amd64.deb ...
Unpacking dpkg (1.18.24ubuntu1) over (1.18.23ubuntu7) ...
Setting up dpkg (1.18.24ubuntu1) ...
(Reading database ... 4063 files and directories currently installed.)
Preparing to unpack .../perl-base_5.24.1-3ubuntu1_amd64.deb ...
Unpacking perl-base (5.24.1-3ubuntu1) over (5.24.1-2ubuntu1) ...
Setting up perl-base (5.24.1-3ubuntu1) ...
(Reading database ... 4063 files and directories currently installed.)
Preparing to unpack .../libgcrypt20_1.7.6-2_amd64.deb ...
Unpacking libgcrypt20:amd64 (1.7.6-2) over (1.7.6-1) ...
Setting up libgcrypt20:amd64 (1.7.6-2) ...
Processing triggers for libc-bin (2.24-9ubuntu2) ...
 ---> 946d1267e175
Removing intermediate container 23526b1412fe
Successfully built 946d1267e175
Successfully tagged ubuntu:evaristogz
nano@satellite:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              evaristogz          946d1267e175        15 seconds ago      141MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7MB
```

Si ejecutamos `docker image ls` o `docker images` veremos que se ha descargado también la imagen 17.10 del repositorio de Ubuntu. Esto lo hace porque a raíz de ella se crea nuestra nueva imagen.

```
nano@satellite:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              evaristogz          946d1267e175        15 seconds ago      141MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7MB
```

2.4.2 Ver la historia de una imagen

Con docker history ubuntu:evaristogz o docker image history ubuntu:evaristogz podemos comprobar las acciones que se han realizado sobre una imagen de Docker en concreto, además del impacto en cuanto a tamaño se refiere.

```
nano@satellite:~$ docker history ubuntu:evaristogz
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
0d69fb47ce07        11 minutes ago      /bin/sh -c apt-get update && apt-get -y up...
50.8MB
319e1cdb0a5b        12 minutes ago      /bin/sh -c #(nop)  MAINTAINER Evaristo GZ ...   0B
c6cac97ba835        5 days ago          /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           5 days ago          /bin/sh -c mkdir -p /run/systemd && echo '...   7B
<missing>           5 days ago          /bin/sh -c sed -i 's/^#\s*\(deb.*universe\...
2.76kB
<missing>           5 days ago          /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>           5 days ago          /bin/sh -c set -xe   && echo '#!/bin/sh' >...   745B
<missing>           5 days ago          /bin/sh -c #(nop) ADD file:08785f4740adaeb...
89.7MB
```

2.4.5 Crear nueva versión de una imagen

El comando `docker tag` permite etiquetar con otro nombre y etiqueta una imagen ya creada.

Creará una referencia de la misma imagen con el nuevo nombre, pero sin llegar a tener un duplicado de ellas pues guarda el mismo ID de imagen.

```
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              evaristogz          0d69fb47ce07        21 minutes ago      141MB
ubuntuegz           evaristogz2         0d69fb47ce07        21 minutes ago      141MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7MB
```

2.4.4 Descargar una imagen de Docker

La descarga de una imagen se realiza de manera automática al ejecutar un contenedor, aún así, es posible que deseemos descargar imágenes que sabemos que más tarde utilizaremos para hacerlo en modo sin conexión u otros motivos.

Esta imagen se quedará de manera local en nuestra instalación de Docker. También es posible utilziar `docker image pull nginx`

```
nano@satellite:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
226f4ec56ba3: Pull complete
53d7dd52b97d: Pull complete
Digest: sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
Status: Downloaded newer image for nginx:latest
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              evaristogz          0d69fb47ce07        25 minutes ago      141MB
ubuntuegz           evaristogz2         0d69fb47ce07        25 minutes ago      141MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7MB
nginx               latest              958a7ae9e569        8 days ago          109MB
```

Para descargar un *tag* o versionado en concreto `docker image pull ubuntu:tag`

2.4.5 Subir una imagen a un Registry

Para subir una imagen a un Docker Registry, en concreto el ofrecido por Docker que es gratuito a la vez que público, debemos identificarnos antes en Docker con el comando `docker login`

```
nano@satellite:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker
ID, head over to https://hub.docker.com to create one.
Username: evaristogz
Password:
Login Succeeded
```

Luego debemos taggear la imagen con el nombre de nuestro repositorio, que anteriormente ya habremos creado a través de https://hub.docker.com y subirla con la ejecución del comando `docker push evaristogz/prueba:latest`

```
nano@satellite:~$ docker tag ubuntu:evaristogz evaristogz/prueba
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
evaristogz/prueba   latest              0d69fb47ce07        33 minutes ago      141MB
ubuntu              evaristogz          0d69fb47ce07        33 minutes ago      141MB
ubuntuegz           evaristogz2         0d69fb47ce07        33 minutes ago      141MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7MB
nginx               latest              958a7ae9e569        8 days ago          109MB
nano@satellite:~$ docker push evaristogz/prueba:latest
The push refers to a repository [docker.io/evaristogz/prueba]
6b58b4118996: Pushed
3c92e836185c: Mounted from library/ubuntu
64a57b2ab46c: Mounted from library/ubuntu
bab410172012: Mounted from library/ubuntu
a858a24471f6: Mounted from library/ubuntu
77e40919e694: Mounted from library/ubuntu
latest: digest: sha256:0b0ed7c7dcd2d82251f55ee12fcb427a3b9e923dc54615c702cd5054537cb516 size:
1569
```

2.4.6 Listar imágenes de Docker

Podemos listar todas las imágenes que tenemos descargadas de manera local con el comando `docker images` o `docker image ls`

Podemos observar que las imágenes evaristogz/prueba:latest, ubuntu:evaristogz y ubuntuegz:evaristogz2 tienen el mismo ID, además del mismo tamaño ya que son la misma imagen con distintos repositorios y tags.

```
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
evaristogz/prueba   latest              0d69fb47ce07        1 hours ago         141 MB
ubuntu              evaristogz          0d69fb47ce07        1 hours ago         141 MB
ubuntuegz           evaristogz2         0d69fb47ce07        1 hours ago         141 MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7 MB
nginx               latest              958a7ae9e569        8 days ago          109 MB
```

Si queremos listar las de un repositorio en concreto lo hacemos con `docker images repositorio`

```
nano@satellite:~$ docker images ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              evaristogz          0d69fb47ce07        1 hours ago         141 MB
ubuntu              17.10               c6cac97ba835        5 days ago          89.7 MB
```

2.4.6 Eliminar imágenes

Descargamos dos imágenes de prueba de alpine con `docker pull`, una de ellas la etiquetada como latest y la otra con el tag 3.1.

```
nano@satellite:~$ docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
Digest: sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c
Status: Image is up to date for alpine:latest
nano@satellite:~$ docker pull alpine:3.1
3.1: Pulling from library/alpine
1bf4b7c8ae75: Pull complete
Digest: sha256:c870bf18fb9e3ca8162f1f7c95e924820a1fe3ef6b142d235109853ae68772b4
Status: Downloaded newer image for alpine:3.1
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              a41a7446062d        13 days ago         3.97 MB
alpine              3.1                 820010c31e66        13 days ago         5.05 MB
```

Las formas de borrar una imagen son múltiples: `docker rm prueba`, `docker rmi prueba` o `docker image rm prueba` para eliminar una imagen que se llame *prueba*.

En caso de no especificarse el tag, eliminará la versión *latest* manteniendo el resto. Si no se existe la imagen con la etiqueta *latest* debemos especificar el tag.

```
nano@satellite:~$ docker rmi alpine
Untagged: alpine:latest
Untagged: alpine@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c
Deleted: sha256:a41a7446062d197dd4b21b38122dcc7b2399deb0750c4110925a7dd37c80f118
Deleted: sha256:3fb66f713c9fa9debcdaa58bb9858bd04c17350d9614b7a250ec0ee527319e59
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              3.1                 820010c31e66        13 days ago         5.05 MB
nano@satellite:~$ docker rmi alpine
Error response from daemon: No such image: alpine:latest
```

Utilizando el comando `docker rmi $(docker images -q)` eliminamos todas las imágenes de Docker descargadas de manera local a excepción de aquellas que están siendo usadas por algún contenedor.

2.4.3 Eliminar las imágenes no utilizadas

Con `docker image prune -a` eliminamos todas las imágenes que no son referenciadas por algún contenedor activo.

```
nano@satellite:~$ docker image prune -a
WARNING! This will remove all images without at least one container associated to them.
```

```
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: hello-world:latest
untagged: hello-world@sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
deleted: sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233
deleted: sha256:98c944e98de8d35097100ff70a31083ec57704be0991a92c51700465e4544d08
untagged: evaristogz/prueba:latest
untagged:
evaristogz/prueba@sha256:0b0ed7c7dcd2d82251f55ee12fcb427a3b9e923dc54615c702cd5054537cb516
untagged: ubuntu:evaristogz
untagged: ubuntuegz:evaristogz2
deleted: sha256:0d69fb47ce07b08bdbf62f5b3a95ab463c8269a1fda96070bb4e2da0523f9507
deleted: sha256:ac4a36df04a0ddbf542e6bb6359f7562db50446159ad039b336c262fb6cc8013
deleted: sha256:319e1cdb0a5b26355a1d0b925419e731c1455694fd716a5ab92dfd729d40ba3a
untagged: nginx:latest
untagged: nginx@sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
deleted: sha256:958a7ae9e56979be256796dabd5845c704f784cd422734184999cf91f24c2547
deleted: sha256:13f9b35b2c46beca8711c38cced5e4191a6a7dde0ab63ac1034182886b27e0f0
deleted: sha256:5eba0218753191373b25ba8646462c983391ea0c56f5bd7169d41d183002e49c
deleted: sha256:8781ec54ba04ce83ebcdb5d0bf0b2bb643e1234a1c6c8bec65e8d4b20e58a90d
untagged: ubuntu:17.10
untagged: ubuntu@sha256:52d19954c14bbadf6a0965c4cad3da0bb052b62cae16a108add338e1838cbd72
deleted: sha256:c6cac97ba835a8800292fede7fcb0936aaf045334d35ca0475d4841c9b252629

Total reclaimed space: 160.2 MB
```

Podemos comprobar que la imagen alpine:latest no se ha eliminado con la ejecución de `docker image prune -a` ya que existe un contenedor corriendo con esa imagen actualmente.

```
nano@satellite:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
2cba79e5b8bb        alpine              "ping docker.com"   20 seconds ago      Up 53 seconds
pingdocker
nano@satellite:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              a41a7446062d        13 days ago         3.97 MB
```

2.5 GESTIÓN DE REDES

```
nano@satellite:~$ docker network

Usage: docker network COMMAND

Manage networks

Options:
      --help   Muestra su uso

Commands:
  connect     Conecta un contenedor a una red
  create      Crea una red
  disconnect  Desconecta un contenedor de una red
  inspect     Muestra información detallada de una o más redes networks
  ls          Lista las redes
  prune       Elimina las redes no usadas
  rm          Elimina una o más redes

Run 'docker network COMMAND --help' for more information on a command.
```

2.5.1 Crear una red

Por defecto tenemos tres redes: *bridge*, *host* y *none*. Para crear una nueva red lo hacemos con el comando `docker network create prueba`

```
nano@satellite:~$ docker network create prueba
fe77e2878a7d86936e57f5bfc92a1543e68aaf45b8d3346568521f4e6bec304c
```

2.5.2 Listar redes

La opción *ls* listará las redes Docker disponibles. Se muestra el identificador, el nombre, el driver y el alcance.

```
nano@satellite:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
50a374a08fe7        bridge              bridge              local
afcd5ca5b6d4        host                host                local
b37fc92cd80a        none                null                local
fe77e2878a7d        prueba              bridge              local
```

También podemos hacer uso del parámetro *--filter* para filtrar los resultados y mostrar únicamente las redes que cumplan una condición concreta. En este caso, solo se mostrarán las redes que sean de tipo *bridge*.

```
nano@satellite:~$ docker network ls -f driver=bridge
NETWORK ID          NAME                DRIVER              SCOPE
50a374a08fe7        bridge              bridge              local
fe77e2878a7d        prueba              bridge              local
```

2.5.3 Ver información detallada de una red

A través de la opción *inspect* extraemos en formato JSON la información detallada de una red. En este caso se muestran datos como la dirección IP de la red o los contenedores que están haciendo uso de esta red.

```
nano@satellite:~$ docker network inspect prueba
[
    {
        "Name": "prueba",
        "Id": "fe77e2878a7d86936e57f5bfc92a1543e68aaf45b8d3346568521f4e6bec304c",
        "Created": "2017-06-06T09:19:10.668682101Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

2.5.4 Conectar un contenedor a una red

Conectamos el contenedor *pingdocker* a la red *prueba* que hemos creado anteriormente. Los contenedores también pueden crearse con una red definida.

```
nano@satellite:~$ docker network connect prueba pingdocker
nano@satellite:~$ docker inspect -f {{.NetworkSettings.Networks}} pingdocker
map[bridge:0xc42055e0c0 prueba:0xc42055e180]
```

En este caso, el contenedor *prueba* está conectado a dos redes: *bridge* y *prueba*.

Podemos comprobarlo también utilizando `docker network inspect -f {{.Containers}} prueba`

```
nano@satellite:~$ docker network inspect -f {{.Containers}} prueba
map[b93600ebb844ed3a288f7a50e5896e7b7ed3f87a09d3d222ae431bf8a5a7b127:{pingdocker
eaa8e2860ef566bf69c671c9479135391d89a52ca7e53ff0da294fdb91df76a8 02:42:ac:12:00:02
172.18.0.2/16 }]
```

2.5.5 Desconectar un contenedor de una red

Utilizando la opción *disconnect* podemos desconectar un contenedor de una red de Docker.

```
nano@satellite:~$ docker network disconnect prueba pingdocker
nano@satellite:~$ docker network inspect -f {{.Containers}} prueba
map[]
```

2.5.6 Eliminar redes

Para eliminar una red, es necesario que esa red no esté siendo utilizada por ningún contenedor o servicio. Esto no podrá forzarse con opciones como *-f* que disponen otros comandos de Docker.

```
nano@satellite:~$ docker network rm prueba
Error response from daemon: network prueba id
fe77e2878a7d86936e57f5bfc92a1543e68aaf45b8d3346568521f4e6bec304c has active endpoints
```

Deberemos hacer un *inspect* de la red para ver qué contenedores están utilizando la red y desconectarlos de la red.

```
nano@satellite:~$ docker network inspect -f {{.Containers}} prueba
map[b93600ebb844ed3a288f7a50e5896e7b7ed3f87a09d3d222ae431bf8a5a7b127:{pingdocker
cd3c51f0e8f02a9375a4e5286b55adcecc0160be86fc40152898ed143bd6e7b7 02:42:ac:12:00:02
172.18.0.2/16 }]
nano@satellite:~$ docker network disconnect prueba pingdocker
nano@satellite:~$ docker network rm prueba
prueba
```

También podemos hacerlo parando el contenedor que está haciendo uso de la red. Aún así, ese contenedor no podrá volver a iniciarse debido a que le faltará la red. Por lo que quizás es más recomendable eliminarlo o bien desconectarlo de la red a eliminar como hemos hecho.

2.5.7 Eliminar las redes no utilizadas

Con la opción *prune* de esta comando podemos limpiar las redes que no están siendo usadas por ningún contenedor o servicio. De esta manera eliminamos las redes huérfanas.

```
nano@satellite:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
50a374a08fe7        bridge              bridge              local
afcd5ca5b6d4        host                host                local
b37fc92cd80a        none                null                local
4fd0b909f5b1        prueba              bridge              local
nano@satellite:~$ docker network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
prueba
```

2.6 GESTIÓN DE ALMACENAMIENTO

```
nano@satellite:~$ docker volume

Usage: docker volume COMMAND

Manage volumes

Options:
      --help   Muestra su uso

Commands:
  create      Crea un volumen
  inspect     Muestra información detallada de uno o más volúmenes
  ls          Lista los volúmenes
  prune       Elimina los volúmenes no usados
  rm          Elimina uno o más volúmenes

Run 'docker volume COMMAND --help' for more information on a command.
```

2.6.1 Crear un volumen

docker volume create prueba creará un volumen de nombre prueba con el driver *local*.

```
nano@satellite:~$ docker volume create prueba
prueba
nano@satellite:~$ docker volume ls
DRIVER              VOLUME NAME
local               prueba
```

Los drivers para volúmenes permiten utilizar otros tipos de volúmenes en Docker a través de plugins: Flocker, GlusterFS, Virtuozzo...

2.6.2 Listar volúmenes

Con `docker volume ls` podemos ver los volúmenes que están disponibles en nuestra instalación de Docker.

```
nano@satellite:~$ docker volume ls
DRIVER              VOLUME NAME
local               prueba
```

Este comando también permite la opción *-f* o *--filter* para filtrar los resultados según el driver del volumen, por ejemplo.

2.6.3 Ver información detallada de un volumen

Al igual que el resto de elementos de Docker, puede consultarse su información detallada a través de la opción *inspect* que nos devolverá un resultado en JSON.

```
nano@satellite:~$ docker volume inspect prueba
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/mnt/sda1/var/lib/docker/volumes/prueba/_data",
        "Name": "prueba",
        "Options": {},
        "Scope": "local"
    }
]
```

2.6.4 Eliminar un volumen

Podemos eliminar un volumen que no esté en uso a través del comando `docker volume rm prueba`

```
nano@satellite:~$ docker volume rm prueba
prueba
```

Si el volumen está siendo usado por algún contenedor o servicio deberemos de eliminarlo para poder hacer esta acción.

2.6.5 Eliminar los volúmenes no usados

La ejecución de la opción *prune* eliminará aquellos volúmene huérfanos que no están siendo utilizados por ningún contenedor o servicio.

```
nano@satellite:~$ docker volume prune
WARNING! This will remove all volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
prueba

Total reclaimed space: 0 B
```

2.7 GESTIÓN DE PLUGINS

```
nano@satellite:~$ docker plugin

Usage: docker plugin COMMAND

Manage plugins

Options:
      --help   Print usage

Commands:
  create      Crea un plugin desde rootfs y su configuración. El directorio de datos del
plugin debe contener un config.json y un directorio rootfs
  disable     Desactiva un plugin
  enable      Activa un plugin
  inspect     Muestra información detallada de uno o más plugins
  install     Instala un plugin
  ls          Lista los plugins
  push        Sube un plugin al registry
  rm          Elimina uno o más plugins
  set         Cambia los ajustes de un plugin
  upgrade     Actualiza un plugin existente

Run 'docker plugin COMMAND --help' for more information on a command.
```

2.7.1 Instalar un plugin

```
nano@satellite:~$ docker plugin install tiborvass/sample-volume-plugin
latest: Pulling from tiborvass/sample-volume-plugin
eb9c16fbdc53: Download complete
Digest: sha256:00b42de88f3a3e0342e7b35fa62394b0a9ceb54d37f4c50be5d3167899994639
Status: Downloaded newer image for tiborvass/sample-volume-plugin:latest
Installed plugin tiborvass/sample-volume-plugin
```

2.7.2 Desactivar un plugin

```
nano@satellite:~$ docker plugin disable tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
nano@satellite:~$ docker plugin ls
ID                  NAME                                DESCRIPTION
ENABLED
c74a6b666f0e        tiborvass/sample-volume-plugin:latest   A sample volume plugin for Docker
false
```

2.7.3 Activar un plugin

```
nano@satellite:~$ docker plugin enable tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
nano@satellite:~$ docker plugin ls
ID                  NAME                                DESCRIPTION
ENABLED
c74a6b666f0e        tiborvass/sample-volume-plugin:latest   A sample volume plugin for Docker
true
```

2.7.4 Mostrar información de un plugin

Puede usarse `docker plugin inspect tiborvass/sample-volume-plugin` o `docker inspect tiborvass/sample-volume-plugin`

Esto mostrará datos sobre configuración del plugin.

```
nano@satellite:~$ docker inspect tiborvass/sample-volume-plugin
[
    {
        "Config": {
            "Args": {
                "Description": "",
                "Name": "",
```

```
                "Settable": null,
                "Value": null
            },
            "Description": "A sample volume plugin for Docker",
            "Documentation": "https://docs.docker.com/engine/extend/plugins/",
            "Entrypoint": [
                "/usr/bin/sample-volume-plugin",
                "/data"
            ],
        (...)
        "Enabled": true,
        "Id": "c74a6b666f0e1777561aace67892eb02109835ce0e5574e6c62209f4d6ca1232",
        "Name": "tiborvass/sample-volume-plugin:latest",
        "PluginReference": "tiborvass/sample-volume-plugin:latest",
        "Settings": {
            "Args": [],
            "Devices": [],
            "Env": [
                "DEBUG=0"
            ],
            "Mounts": []
        }
    }
]
```

También es posible extraer solo ciertos parámetros del JSON que devuelve, parseando su resultado.

```
nano@satellite:~$ docker plugin inspect -f {{.Config.Description}} tiborvass/sample-volume-plugin
A sample volume plugin for Docker
```

2.7.5 Cambiar configuración de un plugin

Consultamos el parámetro que queremos cambiar con la opción *-f* de *inspect*

```
nano@satellite:~$ docker plugin inspect -f {{.Settings.Env}} tiborvass/sample-volume-plugin
[DEBUG=0]
```

Intentamos cambiar su configuración, pero no puede realizarse con un plugin que se encuentra activo.

Por lo tanto, desactivamos el plugin, cambiamos el valor con `docker plugin set tiborvass/sample-volume-plugin DEBUG=1` activamos el plugin y comprobamos el valor de su configuración.

```
nano@satellite:~$ docker plugin set tiborvass/sample-volume-plugin DEBUG=1
Error response from daemon: cannot set on an active plugin, disable plugin before setting
nano@satellite:~$ docker plugin disable tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
nano@satellite:~$ docker plugin set tiborvass/sample-volume-plugin DEBUG=1
nano@satellite:~$ docker plugin enable tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
nano@satellite:~$ docker plugin inspect -f {{.Settings.Env}} tiborvass/sample-volume-plugin
[DEBUG=1]
```

2.7.6 Eliminar un plugin

Para eliminar un plugin es necesario que este no esté activado.

```
nano@satellite:~$ docker plugin disable tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
nano@satellite:~$ docker plugin rm tiborvass/sample-volume-plugin
tiborvass/sample-volume-plugin
```

2.8 GESTIÓN DE SECRETOS

```
nano@satellite:~$ docker secret

Usage: docker secret COMMAND

Manage Docker secrets

Options:
      --help   Print usage

Commands:
  create      Create a secret from a file or STDIN as content
  inspect     Display detailed information on one or more secrets
  ls          List secrets
  rm          Remove one or more secrets

Run 'docker secret COMMAND --help' for more information on a command.
```

2.8.1 Crear un secreto

Un secreto debe ser pasado al manager de Swarm a través de un fichero anteriormente creado y que puede ser borrado tras su creación. En él se incluyen las declaraciones que son privadas, como puede ser un archivo de configuración con datos de conexión a una base de datos que se guardarán cifradas en el clúster.

En este caso se trata del fichero *secretos.conf* que incluye las cadenas *usuario* y *contraseña*

```
nano@satellite:~$ docker secret create prueba secretos.conf
n5d36bv5cnktzzfmp4geyo77q
nano@satellite:~$ docker secret ls
ID                          NAME                CREATED             UPDATED
n5d36bv5cnktzzfmp4geyo77q   prueba              6 seconds ago       6 seconds ago
```

Con esto está creado el secreto, de manera que pueda utilizarse para crear un servicio. Esto también es útil para tener configuraciones de varios entornos, teniendo que cambiar únicamente la referencia del secreto.

A continuación, para lanzar un servicio con el secreto debemos indicar su nombre con *--secret*

```
nano@satellite:~$ docker service create --name pingdocker --secret="prueba" alpine ping
docker.com
uvd71rz546v0o92s0x15iban7
nano@satellite:~$ docker service ls
ID            NAME        MODE        REPLICAS  IMAGE
uvd71rz546v0  pingdocker  replicated  1/1       alpine:latest
```

2.8.1.1 Comprobar un secreto creado

El contenido de un secreto creado y asociado a un servicio es consultable a través de la ruta */run/secrets* del contenedor.

Consultamos las tareas desplegadas por el servicio, comprobando que en este caso se trata del mismo nodo manager. Si esta tarea fuese ejecutada en otro nodo distinto deberíamos conectar nuestra CLI de Docker con ese nodo de Swarm a través de `eval $(docker-machine env prueba02)`

```
nano@satellite:~$ docker service ps pingdocker
ID            NAME          IMAGE          NODE    DESIRED STATE  CURRENT STATE
ERROR  PORTS
qrlmy22aozzo  pingdocker.1  alpine:latest  prueba  Running        Running 3 minutes ago
```

Vemos las tareas que se están ejecutando en los contenedores con el comando docker ps

```
nano@satellite:~$ docker ps
CONTAINER ID        IMAGE
COMMAND             CREATED             STATUS              PORTS               NAMES
3a092e062dc0
alpine:latest@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c   "ping
docker.com"   3 minutes ago       Up 3 minutes
pingdocker.1.qrlmy22aozzo4vwc30sar0q99
```

Con el ID del contenedor, ejecutamos en él un *ls -l* del directorio */run/secrets/* que nos mostrará los secretos de ese contenedor. Con *cat* podremos ver su contenido igualmente.

```
nano@satellite:~$ docker exec 3a092e062dc0 ls -l /run/secrets
total 4
-r--r--r--    1 root     root            34 Jun 3 17:48 prueba
nano@satellite:~$ docker exec 3a092e062dc0 cat /run/secrets/prueba
usuariosecreto
contraseñasecreta
```

2.8.2 Listar secretos

Los secretos solo podrán ser listados desde un nodo Manager y nunca desde un nodo Worker. Además, tampoco podrá verse el contenido de un secreto, a no ser que se asocie a un servicio.

```
nano@satellite:~$ docker secret ls
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to
view or modify cluster state. Please run this command on a manager node or promote the
current node to a manager.
```

Por lo tanto, debemos conectarnos a un nodo que con rol Manager y ejecutar `docker secret ls`

```
nano@satellite:~$ eval $(docker-machine env prueba)
nano@satellite:~$ docker secret ls
ID                          NAME                CREATED             UPDATED
n5d36bv5cnktzzfmp4geyo77q   prueba              13 minutes ago      13 minutes ago
```

2.8.3 Mostrar información detallada de un secreto

Es posible consultar la información sobre un secreto a través de la opción *inspect*. Esto mostrará distintos parámetros como el ID, la fecha de creación, la fecha de actualización o el nombre.

```
nano@satellite:~$ docker secret inspect prueba
[
    {
        "ID": "n5d36bv5cnktzzfmp4geyo77q",
        "Version": {
            "Index": 26
        },
        "CreatedAt": "2017-06-03T17:45:40.243647204Z",
        "UpdatedAt": "2017-06-03T17:45:40.243647204Z",
        "Spec": {
            "Name": "prueba",
            "Labels": {}
        }
    }
]
```

2.8.4 Eliminar un secreto

Si deseamos eliminar un secreto, es necesario que es no esté siendo utilizado por ningún servicio activo.

En este ejemplo recreamos el borrado del secreto con nombre *prueba* que está siendo usado por el servicio *pingdocker*.

```
nano@satellite:~$ docker secret rm prueba
Error response from daemon: rpc error: code = 3 desc = secret 'prueba' is in use by the
following service: pingdocker
nano@satellite:~$ docker service rm pingdocker
pingdocker
nano@satellite:~$ docker secret rm prueba
n5d36bv5cnktzzfmp4geyo77q
```

2.9 GESTIÓN DE SWARM

```
nano@satellite:~$ docker swarm

Usage: docker swarm COMMAND

Manage Swarm

Options:
      --help   Muestra su uso

Commands:
  init         Inicia un Swarm
  join         Unir nodo a un Swarm y/o Manager
  join-token   Administra los tokens de unión
  leave        Sale del Swarm
  unlock       Desbloquea Swarm
  unlock-key   Administra la clave de desbloqueo
  update       Actualiza el Swarm

Run 'docker swarm COMMAND --help' for more information on a command.
```

En los siguientes ejemplos utilizo una serie de nodos de prueba creados en VirtualBox con Docker Machine.

Lo hago con la siguiente batería de comandos:

```
nano@satellite:~$ docker-machine create -d virtualbox prueba
Running pre-create checks...
Creating machine...
(prueba) Copying /home/nano/.docker/machine/cache/boot2docker.iso to
/home/nano/.docker/machine/machines/prueba/boot2docker.iso...
(prueba) Creating VirtualBox VM...
(prueba) Creating SSH key...
(prueba) Starting the VM...
(prueba) Check network to re-create if needed...
(prueba) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
machine, run: docker-machine env prueba
```

Accedemos a él con `docker-machine ssh prueba`

2.9.1 Iniciar un Swarm

El comando `docker swarm init` permitirá iniciar un Docker Swarm. Sin embargo, en la mayoría de las ocasiones será necesario especificar un dirección IP o tarjeta de red para que el clúster pueda iniciarse debido a que se disponga de varias interfaces de red.

```
docker@prueba:~$ docker swarm init
Error response from daemon: could not choose an IP address to advertise since this system has
multiple addresses on different interfaces (10.0.2.15 on eth0 and 192.168.99.100 on eth1) -
specify one with --advertise-addr
```

Así pues, podemos hacerlo mediante la especificación de la tarjeta de red con `docker swarm init --advertise-addr eth1` o la dirección IP del nodo a través del parámetro *--advertise-addr*

```
docker@prueba:~$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (tpznikyxkv27wyu7bzvblfoyd) is now a manager.
```

```
To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r91c-
6arxrguisp3wa7zmubzms8bqn \
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

2.9.1.1 Iniciar un Swarm con autolock

Autolock es una opción que permite cifrar Swarm a través de un token. Esto se realiza pasándole el parámetro al iniciar el nodo de Swarm, no puede hacerse a posteriori.

Aquí ejecutamos un nuevo Swarm con autolock y utilizando el nombre de la interfaz de red y no la dirección IP como lo hacemos en el punto *2.9.1*.

```
docker@prueba:~$ docker swarm init --autolock --advertise-addr eth1
Swarm initialized: current node (m226dmt32l13h9i5nzq5ai62o) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-1dabx0g90mdmgak3kyrat74ivmf5ileypbb51tl1msoo5g8ypx-
476s0g29uxdzmvf724o55djm7 \
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.

To unlock a swarm manager after it restarts, run the `docker swarm unlock`
command and provide the following key:

    SWMKEY-1-0A2xk5lptIcO3N+xKTYSfRNJl/aM9A7sK4pU3z4Ex7Y

Please remember to store this key in a password manager, since without it you
will not be able to restart the manager.
```

Para que los cambios surtan efecto es necesario que se reinicie el servicio de Docker Swarm. En nuestro caso lo hemos realizado reiniciando el nodo de VirtualBox completo.

```
docker@prueba:~$ exit
exit status 1
nano@satellite:~$ docker-machine restart prueba
Restarting "prueba"...
(prueba) Check network to re-create if needed...
(prueba) Waiting for an IP...
Waiting for SSH to be available...
Detecting the provisioner...
Restarted machines may have new IP addresses. You may need to re-run the `docker-machine env`
command.
```

Al acceder al nodo e intentar utilizar cualquier comando de Docker Swarm obtendremos el mensaje de que Swarm está cifrado y es necesario desbloquearlo.

```
nano@satellite:~$ docker-machine ssh prueba
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \     \         __/
              \____\_______/

 _                 _   ____     _            _
| |__   ___     ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 17.05.0-ce, build HEAD : 5ed2840 - Fri May  5 21:04:09 UTC 2017
Docker version 17.05.0-ce, build 89658be
```

```
docker@prueba:~$ docker swarm join-token manager
Error response from daemon: Swarm is encrypted and needs to be unlocked before it can be
used. Please use "docker swarm unlock" to unlock it.
```

Tampoco podríamos ejecutar comandos desde nuestro host anfitrión, haciendo uso del Docker Engine de *prueba*.

2.9.2 Desbloquear un Swarm

En el punto *2.9.1.1* hemos iniciado un Swarm con el parámetro *--autolock*, esto hace que su contenido esté cifrado y no sea accesible tras un reinicio sin desbloquearlo.

```
docker@prueba:~$ docker swarm join-token manager
Error response from daemon: Swarm is encrypted and needs to be unlocked before it can be
used. Please use "docker swarm unlock" to unlock it.
docker@prueba:~$ docker swarm unlock
Please enter unlock key:
docker@prueba:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-1dabx0g90mdmgak3kyrat74ivmf5ileypbb51tl1msoo5g8ypx-
8ny7qh2wcul4vtr1qqqz1i5y7 \
    192.168.99.100:2377
```

Lógicamente, si no conocemos el token de desbloqueo no podremos administrar el Swarm. Este token podemos consultarlo a través de `docker swarm unlock-key -q` y no puede ser cambiado.

Sí podremos hacer que el nodo abandone el Swarm con `docker swarm leave --force`

2.9.3 Administrar tokens de unión

Los tokens son los que permitirán a otros nodos unirse al Docker Swarm. Estos son proporcionados por un nodo manager y deben ejecutarse dentro del nodo que queremos añadir al Docker Swarm.

Aquí vemos un ejemplo de qué ocurriría si solicitásemos los tokens a un nodo que no es Manager o a un nodo que no forma parte del Swarm.

```
docker@prueba02:~$ docker swarm join-token worker
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to
view or modify cluster state. Please run this command on a manager node or promote the
current node to a manager.
docker@prueba02:~$ docker swarm leave
Node left the swarm.
docker@prueba02:~$ docker swarm join-token worker
Error response from daemon: This node is not a swarm manager. Use "docker swarm init" or
"docker swarm join" to connect this node to swarm and try again.
```

Con `docker swarm join-token manager` obtendremos el comando a ejecutar para añadir otro nodo manager. Mientras que `docker swarm join-token worker` nos daría el resultado a ejecutar en otro nodo que quisiéramos añadir como worker.

```
docker@prueba:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r9lc-
9cbal67yfh46urqrcw20r65kd \
    192.168.99.100:2377

docker@prueba:~$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r9lc-
6arxrguisp3wa7zmubzms8bqn \
    192.168.99.100:2377
```

Podemos obtener únicamente el token con el parámetro *-q*, esto permitiría tratar el token con procesos de automatización.

```
docker@prueba:~$ docker swarm join-token worker -q
SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r9lc-6arxrguisp3wa7zmubzms8bqn
```

2.9.4 Unir un nodo a un Swarm

Para unir un nodo a un Swarm necesitamos los tokens que proporciona el nodo Manager.

En este caso añadimos el nodo *prueba02* como Manager del Swarm.

```
docker@prueba02:~$ docker swarm join \
>     --token SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r9lc-
9cbal67yfh46urqrcw20r65kd \
>     192.168.99.100:2377

This node joined a swarm as a manager.
```

Ahora, también será posible consultar los tokens en este nodo, ya que cumple la función de Manager.

```
docker@prueba02:~$ docker swarm join-token worker -q
SWMTKN-1-278pxc9p8ybm68xo0kuchigijtrugn1q80w5js5ae4i0n0r9lc-6arxrguisp3wa7zmubzms8bqn
```

2.9.5 Salir de un Swarm

Podemos hacer que un nodo abandone el Swarm, dejando de prestar servicio en el clúster de Docker. Si se trata de un nodo Manager deberemos forzar la acción, ya que esto haría que el resto del clúster no funcionase.

```
docker@prueba02:~$ docker swarm leave
Error response from daemon: You are attempting to leave the swarm on a node that is
participating as a manager. Removing the last manager erases all current state of the swarm.
Use `--force` to ignore this message.
docker@prueba02:~$ docker swarm leave --force
Node left the swarm.
```

Esto puede ser una opción para aprovechar recursos de máquinas en picos de demanda y que luego seguirán ejecutando su función normal, sin formar parte de un clúster de Docker.

2.10 GESTIÓN DE NODOS

```
nano@satellite:~$ docker node

Usage: docker node COMMAND

Manage Swarm nodes

Options:
      --help   Muestra su uso

Commands:
  demote      Quita uno o más nodos como Manager en el Swarm
  inspect     Muestra información detallada de uno o más nodos
  ls          Lista los nodos en el Swarm
  promote     Promueve uno o más nodos como Manager en el Swarm
  ps          Lista las tareas corriendo en uno o más nodos, por defecto en el nodo actual
  rm          Elimina uno o más nodos del Swarm
  update      Actualiza un nodo

Run 'docker node COMMAND --help' for more information on a command.
```

2.10.1 Promover nodo a Manager

Desde un nodo Manager es posible gestionar los roles del resto de nodos de Swarm, de manera que podemos promover un nodo Worker a Manager o quitarle ese rol.

```
docker@prueba:~$ docker node ls
ID                           HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
m226dmt32l13h9i5nzq5ai62o *  prueba              Ready               Active
Leader
```

```
zc0suyc5ompi8agkt5tpg4bkd     prueba02            Ready               Active
docker@prueba:~$ docker node promote prueba02
Node prueba02 promoted to a manager in the swarm.
docker@prueba:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
m226dmt32l13h9i5nzq5ai62o *   prueba              Ready               Active
Leader
zc0suyc5ompi8agkt5tpg4bkd     prueba02            Ready               Active
Reachable
```

El nodo *prueba02* pasará a ser Manager, pero el nodo *prueba* seguirá cumpliendo las funciones de *Leader*. Aún así, *prueba02* puede gestionar los roles de la misma manera que *prueba*, por lo que podría quitar el rol de Manager a cualquier otro nodo.

También podemos hacerlo con el comando `docker node update --role manager prueba02`

2.10.2 Degradar nodo a Worker

Siguiendo con el ejemplo expuesto en el punto *2.10.1*, ahora quitaremos el rol de Manager al nodo *prueba* (que concedió rol de Manager a *prueba02*).

De esta manera comprobamos cómo el nodo *prueba02* puede gestionar el rol del nodo Leader (*prueba*) y como pasa a asumir el estado de Leader.

```
docker@prueba02:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
m226dmt32l13h9i5nzq5ai62o     prueba              Ready               Active
Leader
zc0suyc5ompi8agkt5tpg4bkd *   prueba02            Ready               Active
Reachable
docker@prueba02:~$ docker node demote prueba
Manager prueba demoted in the swarm.
docker@prueba02:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
m226dmt32l13h9i5nzq5ai62o     prueba              Ready               Active
zc0suyc5ompi8agkt5tpg4bkd *   prueba02            Ready               Active
Leader
```

También puede hacerse con el propio nodo.

```
docker@prueba02:~$ docker node promote prueba
Node prueba promoted to a manager in the swarm.
docker@prueba02:~$ docker node demote prueba02
Manager prueba02 demoted in the swarm.
```

O con el comando `docker node update --role manager prueba02`

2.10.3 Listar los contenedores de un nodo

Para listar los contenedores que se están ejecutando en un nodo utilizamos el comando `docker node ps` desde un nodo Manager.

Si lo hacemos sin parámetros se mostrará los contenedores que se ejecutan en el nodo actual, que mostraría un resultado similar a ejecutar `docker ps`

```
docker@prueba:~$ docker node ps
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE            ERROR               PORTS
tg6cpua4pya0        registry.1          registry:latest     prueba              Running
Running 22 seconds ago
5u8bzvwf2yus        registry.3          registry:latest     prueba              Running
Running 22 seconds ago
docker@prueba:~$ docker node ps prueba02
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE           ERROR               PORTS
my5rr4qae5yo        registry.2          registry:latest     prueba02            Running
Running 57 seconds ago
```

```
feyjwb8zacj3        registry.4          registry:latest     prueba02            Running
Running 57 seconds ago
483yupyxd5hb        registry.5          registry:latest     prueba02            Running
Running 57 seconds ago
```

Un nodo Worker no puede ejecutar `docker node ps` ni en su propio nodo. Para consultar los contenedores que están corriendo en un nodo Worker debemos hacerlo desde un Manager.

2.10.4 Cambiar nodo a solo Manager y no Manager+Worker

Por defecto, un nodo Manager también actúa como nodo Worker debido a que su disponibilidad es *Active*. Así pues, todos los nodos de un Swarm ejecutarán en él tareas desplegadas por un servicio y, por lo tanto, ejecutará en él contenedores.

Este estado se indica en el valor *Availability*.

Para que un nodo Manager no ejecute ninguna tarea debemos cambiar su disponibilidad a *Drain*. Lo hacemos con la ejecución `docker node update --availability drain prueba` en nuestro nodo Manager.

```
docker@prueba:~$ docker node update --availability drain prueba
prueba
docker@prueba:~$ docker node ps
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE              ERROR               PORTS
tg6cpua4pya0        registry.1          registry:latest     prueba              Shutdown
Shutdown 31 seconds ago
5u8bzvwf2yus        registry.3          registry:latest     prueba              Shutdown
Shutdown 31 seconds ago
docker@prueba:~$ docker node ps prueba02
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE            ERROR               PORTS
wbp9zkr01dms        registry.1          registry:latest     prueba02            Running
Running 42 seconds ago
my5rr4qae5yo        registry.2          registry:latest     prueba02            Running
Running 11 minutes ago
n3ybk7qbljhj        registry.3          registry:latest     prueba02            Running
Running 42 seconds ago
feyjwb8zacj3        registry.4          registry:latest     prueba02            Running
Running 11 minutes ago
483yupyxd5hb        registry.5          registry:latest     prueba02            Running
Running 11 minutes ago
docker@prueba:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
m226dmt32l13h9i5nzq5ai62o *   prueba              Ready               Drain
Leader
zc0suyc5ompi8agkt5tpg4bkd     prueba02            Ready               Active
```

El valor de la columna "Availability" pasará de *Active* a *Drain* y los contenedores que se estaban ejecutando en *prueba* se ejecutarán ahora en *prueba02*, "vaciando" de contenedores el nodo Manager.

También podemos evitar que ciertos servicios se ejecuten en un nodo concreto, en este caso en el nodo manager, a través del uso del parámetro *constraint* de *docker service*.

2.10.5 Cambiar disponibilidad de un nodo

En el anterior punto se editaba la disponibilidad (valor *Availability*) para que el nodo Manager no ejecutara ninguna tarea, estableciendo el valor a *Drain*.

Esta columna puede tener dos valores distintos más: *Active* y *Pause*.

Active es el valor por defecto para todos los nodos, hará que ese nodo reciba tareas del Swarm para ejecutar en él. Mientras que *Pause* hará que el nodo deje de recibir nuevas tareas, pero manteniendo las que se ejecutaban.

En el siguiente ejemplo cambiaremos la disponibilidad del nodo *prueba* a *Pause*. Este nodo cuenta con dos tareas desplegadas, invocadas por un servicio que requiere tres réplicas.

```
docker@prueba:~$ docker service create --name registry --publish 5000:5000 --replicas=3
registry
jxtj9uknr1gjsezi8uodcl7mz
docker@prueba:~$ docker node ps
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE           ERROR               PORTS
ib7f1vhri27b        registry.1          registry:latest     prueba              Running
Preparing 7 seconds ago
y0210ih2vc4d        registry.3          registry:latest     prueba              Running
Preparing 7 seconds ago
docker@prueba:~$ docker node ps prueba02
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE               ERROR               PORTS
24329ftb6bvk        registry.2          registry:latest     prueba02            Running
Preparing 16 seconds ago
docker@prueba:~$ docker node update --availability pause prueba
docker@prueba:~$ docker node ls
ID                           HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS
e7bb4o42sm9o3h2h2ek4i9pvi    prueba02            Ready               Active
z5jnl56d54vb53a538r1jc23i *  prueba              Ready               Pause
Leader
```

Una vez que cambiamos su disponibilidad escalamos el servicio a diez réplicas para comprobar que no se desplegará ninguna tarea más en el nodo que tiene disponibilidad *Pause*.

```
docker@prueba:~$ docker service scale registry=10
registry scaled to 10
docker@prueba:~$ docker node ps
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE                ERROR               PORTS
ib7f1vhri27b        registry.1          registry:latest     prueba              Running
Preparing about a minute ago
y0210ih2vc4d        registry.3          registry:latest     prueba              Running
Preparing about a minute ago
docker@prueba:~$ docker node ps prueba02
ID                  NAME                IMAGE               NODE                DESIRED STATE
CURRENT STATE                ERROR               PORTS
24329ftb6bvk        registry.2          registry:latest     prueba02            Running
Preparing about a minute ago
dqux3t4nz6q6        registry.4          registry:latest     prueba02            Running
Preparing 16 seconds ago
lwbhwqkl8boe        registry.5          registry:latest     prueba02            Running
Preparing 16 seconds ago
urajgkztn036        registry.6          registry:latest     prueba02            Running
Preparing 6 seconds ago
qioxfkzoq3l4        registry.7          registry:latest     prueba02            Running
Preparing 6 seconds ago
n5kbregu92j9        registry.8          registry:latest     prueba02            Running
Preparing 6 seconds ago
9f41fx628d4j        registry.9          registry:latest     prueba02            Running
Preparing 6 seconds ago
98y5ice3xaic        registry.10         registry:latest     prueba02            Running
Preparing 6 seconds ago
```

2.10.5 Etiquetar un nodo

Con la opción *update* podemos poner etiquetas a nuestro nodo mediante clave=valor, de manera que sean consultables en la información detallada del nodo a través del parámetro *inspect*.

Esto sirve para añadir especificaciones o información adicional, como puede ser el entorno (desarrollo/producción), el nombre del administrador del nodo o cualquier otro tipo de información que necesitemos almacenar.

Un ejemplo más útil sería añadir una etiqueta a los nodos que queremos que permanezcan siempre activos, tratándolos con un script para eliminar aquellos que no pertenezcan a esa etiqueta.

```
docker@prueba:~$ docker node update --label-add permanente=si prueba
prueba
```

```
docker@prueba:~$ docker node inspect prueba | grep permanente
                "permanente": "si"
docker@prueba:~$ docker node inspect prueba
[
    {
        "ID": "wlc3c0ivyatvwdcoz6zlel3pu",
        "Version": {
            "Index": 61
        },
        "CreatedAt": "2017-05-22T23:17:10.101776653Z",
        "UpdatedAt": "2017-05-22T23:39:53.519842362Z",
        "Spec": {
            "Labels": {
                "permanente": "si"
            },
            "Role": "manager",
            "Availability": "drain"
        },
        (...)
]
docker@prueba:~$ docker node inspect -f {{.Spec.Labels.permanente}} prueba
si
```

Para consultar todas las etiquetas de un nodo debemos parsear el JSON con el siguiente comando: `docker node inspect -f {{.Spec.Labels}} prueba`

2.10.6 Eliminar etiqueta de un nodo

Podemos eliminar las etiquetas de un nodo con el parámetro *--label-rm* de *update*

```
docker@prueba:~$ docker node inspect -f {{.Spec.Labels.permanente}} prueba
si
docker@prueba:~$ docker node update --label-rm permanente prueba
prueba
docker@prueba:~$ docker node inspect -f {{.Spec.Labels.permanente}} prueba
<no value>
```

2.11 GESTIÓN DE SERVICIOS

```
nano@satellite:~$ docker service

Usage: docker service COMMAND

Manage services

Options:
      --help   Muestra su uso

Commands:
  create      Crea un nuevo servicio
  inspect     Muestra información detallada de uno o más servicios
  ls          Lista los servicios
  ps          Lista las tareas de un servicio
  rm          Elimina uno o más servicios
  scale       Escala uno o múltiples servicios replicados
  update      Actualiza un servicio

Run 'docker service COMMAND --help' for more information on a command.
```

2.11.1 Crear un servicio

La manera más simple de crear un servicio es ejecutando el comando `docker service create alpine ping docker.com` donde *alpine* es la imagen a utilizar y *ping docker.com* un comando que ejecutará.

Esto generará un servicio con un nombre aleatorio, sin réplicas (aunque sí en modo replicado) en un nodo sin especificar.

Sin embargo, lo común es especificar al menos el nombre del servicio que creamos a través del

parámetro *--name*

```
nano@satellite:~$ docker service create --name pingdocker alpine ping docker.com
mzx6srrvjsgdf1iev6fepw8zc
```

De esta manera, podría gestionarse el servicio a través del nombre *pingdocker*.

2.11.1.1 Crear un servicio con especificaciones

A un servicio se le puede pasar múltiples parámetros antes de crearlo. Por ejemplo, el nombre de host que tendrá el contenedor donde se ejecute las tareas del servicio. En este caso, queremos que sea *contenedor*.

```
nano@satellite:~$ docker service create --name pingdocker --hostname contenedor --replicas=5
alpine ping docker.com
ko18t7ehjcqavuocbpk89iepz
```

Lanzamos cinco réplicas de manera que encontraremos tareas ejecutándose en cualquiera de los nodos. Podemos comprobar el hostname accediendo al nodo de Docker Swarm y conectándonos con una terminal interactiva con el comando `docker exec -it 07eed65b5573 sh` donde *07eed65b5573* es el identificador del contenedor.

También podemos hacerlo a través de la información detallada ofrecida por el parámetro *inspect*: `docker inspect -f {{.Config.Hostname}} 07eed65b5573`

2.11.1.2 Crear un servicio en modo replicado o global

Al crear un servicio, por defecto se realiza en modo réplica. Aunque no se haya definido ninguna réplica o se haya establecido una única, por lo que hará posible que un servicio pueda ser escalado.

Aún así, existe otro modo llamado *global* que permite desplegar un servicio con una tarea o réplica en cada nodo de Swarm.

Esto quiere decir que tendremos tantas réplicas como nodos pertenezcan a Docker Swarm. En nuestro caso, que tenemos *prueba* y *prueba02* tendremos dos réplicas del servicio que lancemos con modo global.

```
nano@satellite:~$ docker service create --name pingdocker --mode global alpine ping
docker.com
medda7er8vmxu84zd6szqf09y
nano@satellite:~$ docker service ls
ID            NAME        MODE    REPLICAS  IMAGE
medda7er8vmx  pingdocker  global  2/2       alpine:latest
nano@satellite:~$ docker service ps pingdocker
ID            NAME                                    IMAGE          NODE      DESIRED STATE
CURRENT STATE           ERROR  PORTS
sfwli7j7uuor  pingdocker.fe7yo3nl30hco6qoshbv7k4ie  alpine:latest  prueba    Running
Running 18 seconds ago
6udirbss3tpt  pingdocker.0tptmeffn3u3qp2v69e2n20v2  alpine:latest  prueba02  Running
Running 18 seconds ago
```

2.11.1.3 Crear un servicio con exposición de puertos

La exposición de puertos hace posible que se asocie un puerto de un servicio a un puerto de nuestro nodo, siendo posible acceder a él desde fuera de la red establecida.

Para ello, tanto la configuración del contenedor como la configuración de la aplicación que corre dentro del contenedor deberá estar correctamente definida. Es decir, no podemos hacer una asociación de puertos de un servidor web desde 8080 del servicio al 8080 del nodo si el servidor web no está correctamente configurado para servir por el 8080.

Sí podemos hacer que el servidor web (que sirve por el puerto 80) se asocie con el puerto 8080 del servicio de Docker. Pudiendo acceder a través de la dirección IP del nodo y el puerto 8080.

Para ello, utilizamos una imagen de nginx y ejecutamos el siguiente comando: `docker service create --name servidorweb --publish 8080:80 nginx`

```
nano@satellite:~$ docker service create --name servidorweb --publish 8080:80 nginx
ljxrjcqk27uqppnarf1wvzvfn
nano@satellite:~$ docker service ps servidorweb
ID            NAME           IMAGE         NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
raa3ac4annqo  servidorweb.1  nginx:latest  prueba02  Running        Running 11 minutes ago
nano@satellite:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0tptmeffn3u3qp2v69e2n20v2 *  prueba    Ready   Active        Leader
fe7yo3nl30hco6qoshbv7k4ie    prueba02  Ready   Active
nano@satellite:~$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
ERRORS
prueba     *        virtualbox   Running   tcp://192.168.99.100:2376           v17.05.0-ce
prueba02   -        virtualbox   Running   tcp://192.168.99.101:2376           v17.05.0-ce
```

Consultamos el nodo donde se está ejecutando la única tarea del servicio *servidorweb*. Luego listamos los nodos de VirtualBox para ver la dirección IP a la que debemos acceder desde un navegador web, aunque Docker Swarm utiliza routing mesh.

```
nano@satellite:~$ docker service ps servidorweb
ID            NAME           IMAGE         NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
raa3ac4annqo  servidorweb.1  nginx:latest  prueba02  Running        Running 11 minutes ago
nano@satellite:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0tptmeffn3u3qp2v69e2n20v2 *  prueba    Ready   Active        Leader
fe7yo3nl30hco6qoshbv7k4ie    prueba02  Ready   Active
nano@satellite:~$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
ERRORS
prueba     *        virtualbox   Running   tcp://192.168.99.100:2376           v17.05.0-ce
prueba02   -        virtualbox   Running   tcp://192.168.99.101:2376           v17.05.0-ce
```

En este caso podemos ver la pantalla de "Welcome to nginx!" accediendo a cualquier nodo de Docker Swarm a través del puerto 8080. Esto es debido al routing mesh que Docker Swarm, un mecanismo que permite que un servicio sea accesible por el mismo puerto en todos los nodos, incluso si el nodo no tiene el servicio desplegado en él.

2.11.1.4 Crear un servicio con variables de entorno

Al crear un servicio es posible establecer unas variables para todas las tareas de ese servicio. Estas pueden contener múltiples valores como por ejemplo la configuración de un proxy.

```
nano@satellite:~$ docker service create --name pingdocker --env
HTTP_PROXY=http://proxy.iesgn.org:8000 --env HTTPS_PROXY=http://proxy.iesng.org:8000 alpine
ping docker.com
lkz4ej2r5g2yikk4cw31aso0a
```

Como se ve, es necesario utilizar *--env* para cada variable de entorno que queremos establecer.

Puede consultarse a través de la información detallada que nos devuelve la opción *inspect*.

```
nano@satellite:~$ docker inspect -f {{.Spec.TaskTemplate.ContainerSpec.Env}} pingdocker
[HTTP_PROXY=http://proxy.iesgn.org:8000 HTTPS_PROXY=http://proxy.iesng.org:8000
```

2.11.1.6 Crear un servicio con constraints

Al crear un servicio, podemos especificar constraints o restricciones principalmente sobre los nodos donde se ejecutarán las tareas.

Por ejemplo, podemos indicar que un servicio nunca se ejecute tareas en un nodo específico, bien sea por seguridad, por motivos de rendimiento o por cualquier otra razón. O que se ejecute todas las tareas de ese servicio en un mismo nodo.

Para que nunca se ejecute en un nodo específico lo hacemos indicando la negación con *!=* de manera que quedaría un comando así `docker service create --name pingdocker --constraint 'node.hostname != prueba' alpine ping docker.com`

Las tareas del servicio *pingdocker* nunca se ejecutarían en el nodo *prueba*, aunque este se escalase.

En el siguiente ejemplo se ejecutará un servicio con nombre *pingdocker*, con 15 réplicas en el nodo de Docker Swarm con hostname *prueba02*

```
nano@satellite:~$ docker service create --name pingdocker --constraint 'node.hostname == prueba02' --replicas=15 alpine ping docker.com
pcxazkpumn7m52xkp1sjd7rhi
nano@satellite:~$ docker service ls
ID            NAME        MODE        REPLICAS  IMAGE
pcxazkpumn7m  pingdocker  replicated  15/15     alpine:latest
nano@satellite:~$ docker service ps pingdocker
ID            NAME            IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
ut3x76al6dwc  pingdocker.1    alpine:latest  prueba02  Running        Running 17 seconds ago
1qijlprxkb97  pingdocker.2    alpine:latest  prueba02  Running        Running 17 seconds ago
a0sr68jzt3mt  pingdocker.3    alpine:latest  prueba02  Running        Running 18 seconds ago
0cw0ns12cghn  pingdocker.4    alpine:latest  prueba02  Running        Running 18 seconds ago
a4whj9frs74s  pingdocker.5    alpine:latest  prueba02  Running        Running 17 seconds ago
z7zt6c00ihn7  pingdocker.6    alpine:latest  prueba02  Running        Running 17 seconds ago
gkzzciejl2ig  pingdocker.7    alpine:latest  prueba02  Running        Running 18 seconds ago
gr1qylnaey0f  pingdocker.8    alpine:latest  prueba02  Running        Running 18 seconds ago
khv265lxanqi  pingdocker.9    alpine:latest  prueba02  Running        Running 18 seconds ago
3htn0pdqqgtc  pingdocker.10   alpine:latest  prueba02  Running        Running 17 seconds ago
tittb54fupnk  pingdocker.11   alpine:latest  prueba02  Running        Running 17 seconds ago
k008j0jb42ae  pingdocker.12   alpine:latest  prueba02  Running        Running 17 seconds ago
u3qs2zocloo4  pingdocker.13   alpine:latest  prueba02  Running        Running 18 seconds ago
rfhvktswbao1  pingdocker.14   alpine:latest  prueba02  Running        Running 18 seconds ago
mhi2d84tdpnz  pingdocker.15   alpine:latest  prueba02  Running        Running 18 seconds ago
nano@satellite:~$ docker node ls
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0tptmeffn3u3qp2v69e2n20v2 *   prueba    Ready   Active        Leader
fe7yo3nl30hco6qoshbv7k4ie     prueba02  Ready   Active
```

Podemos comprobar que el servicio está replicado 15/15 en el nodo *prueba02*.

2.11.1.7 Crear un servicio con especificaciones de update

Es posible especificar cómo se comporta un servicio a la hora de realizar una actualización de este. Por ejemplo, podemos definir cuánto tiempo ha de pasar entre actualización de una tarea y otra, o en qué cantidad se realiza (de uno en uno, de cinco en cinco...)

```
nano@satellite:~$ docker service create --name pingdocker --update-delay 30s --update-parallelism 2 alpine ping docker.com
jcl81hgsx6vw3vnzh8prtuy61
```

De esta manera se actualizarían dos tareas cada 30 segundos. Es decir, pararía dos tareas del servicio, las iniciaría con la nueva configuración del update y pasado 30 segundos volvería a hacerlo con las otras dos tareas siguientes.

2.11.2 Listar servicios

Con la ejecución de `docker service ls` podremos ver qué servicios se están ejecutando en nuestro Swarm, así como otro tipo de información como el modo (*replicated* o *global*) o el número de réplicas.

```
nano@satellite:~$ docker service ls
ID            NAME        MODE        REPLICAS  IMAGE
syp2nt42zx5t  pingdocker  replicated  3/3       alpine:latest
vj40hbkxzw98  registry    replicated  1/1       registry:latest
```

Con la opción *--filter* podemos filtrar el listado mostrado a través de clave=valor. Puede filtrarse según el id, nombre, modo o labels en caso de haber usado alguno al crear el servicio. En este ejemplo muestro los servicios que están en modo réplica.

```
nano@satellite:~$ docker service ls --filter mode=replicated
ID            NAME        MODE        REPLICAS  IMAGE
syp2nt42zx5t  pingdocker  replicated  3/3       alpine:latest
vj40hbkxzw98  registry    replicated  1/1       registry:latest
```

Si utilizamos el parámetro *-q* nos mostrará solo el ID del servicio.

```
nano@satellite:~$ docker service ls -q
syp2nt42zx5tf6zgkryeb4pt6
vj40hbkxzw9832e5wrbmnkkfl
```

2.11.3 Listar tareas de un servicio

Un servicio está compuesto por tareas. Docker Swarm despliega tareas de este servicio, estas tareas se ejecutan en contenedores. Estos contenedores reciben el nombre del servicio y una numeración incremental según el número de réplicas.

Consultamos los servicios activos, en este caso un ping a docker.com con cinco réplicas.

```
nano@satellite:~$ docker service ls
ID            NAME        MODE        REPLICAS  IMAGE
nheealov3hm4  pingdocker  replicated  5/5       alpine:latest
```

Seguidamente consultamos con `docker service ps pingdocker` las tareas que ejecuta ese servicio.

```
nano@satellite:~$ docker service ps pingdocker
ID            NAME          IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
kpmw7s1j1c07  pingdocker.1  alpine:latest  prueba    Running        Running 39 seconds ago
rhahazoq6c07  pingdocker.2  alpine:latest  prueba02  Running        Running 39 seconds ago
zccmotnfgmq2  pingdocker.3  alpine:latest  prueba02  Running        Running 39 seconds ago
e54lld0du4r9  pingdocker.4  alpine:latest  prueba    Running        Running 39 seconds ago
ilbigwihw9k3  pingdocker.5  alpine:latest  prueba02  Running        Running 39 seconds ago
```

2.11.4 Acceder al contenedor que ejecuta una tarea de un servicio

Aunque acceder a contenedores no es una tarea habitual, es posible que deseéis acceder a un contenedor que ejecuta una tarea.

Para hacerlo, debemos listar las tareas de un servicio con `docker service ps pingdocker`

```
nano@satellite:~$ docker service ps pingdocker
ID            NAME          IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
zbejdmsdr91x  pingdocker.1  alpine:latest  prueba    Running        Running 19 minutes ago
5lvo8xxfdtz6  pingdocker.2  alpine:latest  prueba    Running        Running 19 minutes ago
vw0y68kn4d4o  pingdocker.3  alpine:latest  prueba02  Running        Running 19 minutes ago
ovgckl6v1cic  pingdocker.4  alpine:latest  prueba02  Running        Running 19 minutes ago
n3rcis5t2b90  pingdocker.5  alpine:latest  prueba02  Running        Running 19 minutes ago
```

Luego acceder al nodo que ejecuta esa tarea y ejecutar un `docker ps` para ver el contenedor que la ejecuta.

```
nano@satellite:~$ eval $(docker-machine env prueba02)
nano@satellite:~$ docker ps
CONTAINER ID        IMAGE
COMMAND             CREATED             STATUS              PORTS               NAMES
83d8dfca01fe
alpine:latest@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c   "ping
docker.com"   20 minutes ago      Up 20 minutes
pingdocker.3.vw0y68kn4d4o5ffn29pc4fl7o
e397df06d71e
alpine:latest@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c   "ping
docker.com"   20 minutes ago      Up 20 minutes
pingdocker.4.ovgckl6v1cic6vovu9uqpkgsm
4fb752459551
alpine:latest@sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c   "ping
docker.com"   20 minutes ago      Up 20 minutes
pingdocker.5.n3rcis5t2b90f4pkfy2si79qm
```

Entonces ejecutamos `docker exec -it 83d8dfca01fe sh`

2.11.5 Escalar un servicio

Para escalar un servicio de docker basta con usar el parámetro *scale* y en clave=valor poner el nombre del servicio y el número de réplicas a las que se desea escalar.

```
nano@satellite:~$ docker service scale pingdocker=15
pingdocker scaled to 15
nano@satellite:~$ docker service ls
ID            NAME            MODE        REPLICAS  IMAGE
nheealov3hm4  pingdocker      replicated  15/15     alpine:latest
nano@satellite:~$ docker service ps pingdocker
ID            NAME            IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
kpmw7s1j1c07  pingdocker.1    alpine:latest  prueba    Running        Running 11 minutes ago
rhahazoq6c07  pingdocker.2    alpine:latest  prueba02  Running        Running 11 minutes ago
zccmotnfgmq2  pingdocker.3    alpine:latest  prueba02  Running        Running 11 minutes ago
e54lld0du4r9  pingdocker.4    alpine:latest  prueba    Running        Running 11 minutes ago
ilbigwihw9k3  pingdocker.5    alpine:latest  prueba02  Running        Running 11 minutes ago
nebsnelo8ydi  pingdocker.6    alpine:latest  prueba02  Running        Running 9 seconds ago
jhs9xn3hyge1  pingdocker.7    alpine:latest  prueba    Running        Running 9 seconds ago
umkn89bglckt  pingdocker.8    alpine:latest  prueba    Running        Running 9 seconds ago
a555lghvuydf  pingdocker.9    alpine:latest  prueba    Running        Running 9 seconds ago
pif9dwmfgrgs  pingdocker.10   alpine:latest  prueba02  Running        Running 9 seconds ago
w47url63i1pu  pingdocker.11   alpine:latest  prueba02  Running        Running 9 seconds ago
w8sxdzbiybe2  pingdocker.12   alpine:latest  prueba02  Running        Running 9 seconds ago
zf554xcx58ds  pingdocker.13   alpine:latest  prueba    Running        Running 9 seconds ago
r0n712nso4qm  pingdocker.14   alpine:latest  prueba    Running        Running 9 seconds ago
keebp9ozgokp  pingdocker.15   alpine:latest  prueba02  Running        Running 9 seconds ago
```

Este proceso no podría hacerse con servicios desplegados en modo global.

2.11.8 Actualizar un servicio

Una de las actualizaciones que podemos realizar es el tiempo de espera entre actualización y actualización de cada contenedor. En este caso establecemos esta opción a un minuto.

```
nano@satellite:~$ docker service update --update-delay 1m pingdocker
pingdocker
```

Al ejecutar cualquier otra actualización del servicio se esperará un minuto para reiniciar la tarea y volverla a lanzar (no necesariamente en el mismo nodo). En este ejemplo ha comenzado con la tarea *pingdocker.2*

```
nano@satellite:~$ docker service update --hostname=container pingdocker
pingdocker
nano@satellite:~$ docker service ps pingdocker
ID            NAME              IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
jw4d1cjlaorw  pingdocker.1      alpine:latest  prueba02  Running        Running 51 seconds
ago
d0s3iyjlc52i  pingdocker.2      alpine:latest  prueba    Ready          Ready 3 seconds ago
4zcbvbky69r5   \_ pingdocker.2  alpine:latest  prueba02  Shutdown       Running 3 seconds ago
urbot3qyildj  pingdocker.3      alpine:latest  prueba    Running        Running 52 seconds
ago
kco36opobjmx  pingdocker.4      alpine:latest  prueba02  Running        Running 51 seconds
ago
gsbz60km1b3x  pingdocker.5      alpine:latest  prueba    Running        Running 52 seconds
ago
```

Quiere decir que si el servicio requiere de cinco réplicas, habrá cinco tareas y por lo tanto cuatro minutos para que se termine el proceso de actualización, a no ser que se indique una condición de paralelismo. El paralelismo indica cuántas tareas actualizará a la vez y se especifica con *--update-parallelism*

También existe la posibilidad de que el servicio ya esté creado con estas especificaciones de actualización.

2.11.7 Rollback de un servicio

El rollback de un servicio permite deshacer una actualización efectuada en un servicio, siguiendo las mismas condiciones de actualización que en el punto *2.11.8*

Con `docker service update --rollback pingdocker` las tareas de nuestro servicio volverán a ejecutarse en contenedores que tendrán como hostname *contenedor*

```
nano@satellite:~$ docker service update --rollback pingdocker
pingdocker
nano@satellite:~$ docker service ps pingdocker
ID             NAME               IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
jfp12srlywll   pingdocker.1       alpine:latest  prueba    Ready          Ready 4 seconds ago
jm15zr5fgzt9    \_ pingdocker.1   alpine:latest  prueba    Shutdown       Running 4 seconds ago
jw4d1cjlaorw    \_ pingdocker.1   alpine:latest  prueba02  Shutdown       Shutdown 8 minutes
ago
d0s3iyjlc52i   pingdocker.2       alpine:latest  prueba    Running        Running 10 minutes
ago
4zcbvbky69r5    \_ pingdocker.2   alpine:latest  prueba02  Shutdown       Shutdown 10 minutes
ago
nt1kdv1z656p   pingdocker.3       alpine:latest  prueba    Running        Running 6 minutes ago
urbot3qyildj    \_ pingdocker.3   alpine:latest  prueba    Shutdown       Shutdown 6 minutes
ago
nriflgmbjntt   pingdocker.4       alpine:latest  prueba02  Running        Running 7 minutes ago
kco36opobjmx    \_ pingdocker.4   alpine:latest  prueba02  Shutdown       Shutdown 7 minutes
ago
xidjz06sgw7l   pingdocker.5       alpine:latest  prueba02  Running        Running 9 minutes ago
gsbz60km1b3x    \_ pingdocker.5   alpine:latest  prueba    Shutdown       Shutdown 9 minutes ag
```

Al finalizar el rollback tendremos una tarea más por cada réplica.

```
nano@satellite:~$ docker service ps pingdocker
ID             NAME               IMAGE          NODE      DESIRED STATE  CURRENT STATE
ERROR  PORTS
jfp12srlywll   pingdocker.1       alpine:latest  prueba    Running        Running 17 minutes
ago
jm15zr5fgzt9    \_ pingdocker.1   alpine:latest  prueba    Shutdown       Shutdown 17 minutes
ago
jw4d1cjlaorw    \_ pingdocker.1   alpine:latest  prueba02  Shutdown       Shutdown 25 minutes
ago
h630i36uj66u   pingdocker.2       alpine:latest  prueba    Running        Running 12 minutes
ago
d0s3iyjlc52i    \_ pingdocker.2   alpine:latest  prueba    Shutdown       Shutdown 12 minutes
ago
4zcbvbky69r5    \_ pingdocker.2   alpine:latest  prueba02  Shutdown       Shutdown 28 minutes
ago
33olfu0am21u   pingdocker.3       alpine:latest  prueba    Running        Running 15 minutes
ago
nt1kdv1z656p    \_ pingdocker.3   alpine:latest  prueba    Shutdown       Shutdown 15 minutes
ago
urbot3qyildj    \_ pingdocker.3   alpine:latest  prueba    Shutdown       Shutdown 23 minutes
ago
9liuc0yux01p   pingdocker.4       alpine:latest  prueba02  Running        Running 16 minutes
ago
nriflgmbjntt    \_ pingdocker.4   alpine:latest  prueba02  Shutdown       Shutdown 16 minutes
ago
kco36opobjmx    \_ pingdocker.4   alpine:latest  prueba02  Shutdown       Shutdown 24 minutes
ago
ylu19vgk7iyx   pingdocker.5       alpine:latest  prueba02  Running        Running 13 minutes
ago
xidjz06sgw7l    \_ pingdocker.5   alpine:latest  prueba02  Shutdown       Shutdown 13 minutes
ago
gsbz60km1b3x    \_ pingdocker.5   alpine:latest  prueba    Shutdown       Shutdown 27 minutes
ago
```

2.11.8 Eliminar servicios

De la misma forma que borramos un contenedor, utilizamos la opción *rm* para borrar un servicio. Los servicios no tienen estados (start/stop), por lo que el servicio será borrado aunque se encuentre en ejecución.

```
nano@satellite:~$ docker service rm pingdocker
pingdocker
```

Para eliminar todos los servicios podemos ejecutar `docker service rm $(docker service ls -q)`

2.12 GESTIÓN DE STACKS

```
nano@satellite:~$ docker stack

Usage: docker stack COMMAND

Manage Docker stacks

Options:
      --help   Muestra su uso

Commands:
  deploy      Despliega un nuevo stack o actualiza un stack existente
  ls          Lista los stacks
  ps          Lista las tareas de un stack
  rm          Elimina el stack
  services    Lista los servicio de un stack

Run 'docker stack COMMAND --help' for more information on a command.
```

2.12.1 Desplegar o actualizar un stack

Para desplegar un nuevo stack de servicios en Docker Swarm podemos utilizar el comando `docker stack up -c stackprueba.yml prueba` o `docker stack deploy -c stackprueba.yml prueba`

```
nano@satellite:~$ docker stack deploy -c stackprueba.yml prueba
Creating network prueba_default
Creating network prueba_backend
Creating network prueba_frontend
Creating service prueba_vote
Creating service prueba_result
Creating service prueba_worker
Creating service prueba_visualizer
Creating service prueba_redis
Creating service prueba_db
```

Si deseamos actualizar el stack con modificaciones en el archivo YML podemos hacerlo ejecutando el mismo comando. Los cambios solo afectarán al servicio modificado.

2.12.2 Listar stacks

Podemos gestionar los stacks que tenemos corriendo en nuestro Docker Swarm con la opción *ls* del comando *docker stack.*

```
nano@satellite:~$ docker stack ls
NAME    SERVICES
prueba  6
```

En este caso son seis servicios los que se están ejecutando a través de este stack.

2.12.3 Listar los servicios de un stack

Para ver los servicios pertenecientes a un stack ejecutamos `docker stack services prueba`

```
nano@satellite:~$ docker stack services prueba
ID            NAME               MODE        REPLICAS  IMAGE
3m803gub3fus  prueba_redis       replicated  2/2       redis:alpine
8dwvg5n4vxmg  prueba_result      replicated  1/1
dockersamples/examplevotingapp_result:before
iynush5rtjdx  prueba_worker      replicated  1/1
dockersamples/examplevotingapp_worker:latest
nyq747flbec6  prueba_visualizer  replicated  1/1       dockersamples/visualizer:stable
p5dheej36nwt  prueba_vote        replicated  2/2
dockersamples/examplevotingapp_vote:before
zf7if0ijl5eh  prueba_db          replicated  1/1       postgres:9.4
```

2.12.4 Listar las tareas de un stack

Si deseamos ver las tareas que están siendo ejecutadas a través de un stack de servicios debemos lanzar el comando `docker stack ps prueba`

```
nano@satellite:~$ docker stack ps prueba
```

```
ID            NAME                  IMAGE                                          NODE
DESIRED STATE  CURRENT STATE               ERROR                         PORTS
g07pxbodkyi8  prueba_db.1           postgres:9.4                                   prueba
Running        Running about a minute ago
u33gb672ebz5  prueba_redis.1        redis:alpine                                   prueba02
Running        Running about a minute ago
oc44mp7q9zwf  prueba_visualizer.1   dockersamples/visualizer:stable                prueba
Running        Running about a minute ago
ycx23nvdzwmt  prueba_worker.1       dockersamples/examplevotingapp_worker:latest   prueba
Running        Running about a minute ago
q35tyawp8ia5   \_ prueba_worker.1   dockersamples/examplevotingapp_worker:latest   prueba
Shutdown       Failed about a minute ago   "task: non-zero exit (1)"
la7mcbis4hbh  prueba_result.1       dockersamples/examplevotingapp_result:before   prueba
Running        Running about a minute ago
qd4zwe43csp2  prueba_vote.1         dockersamples/examplevotingapp_vote:before     prueba
Running        Running about a minute ago
4vsqjq0sb9m5  prueba_redis.2        redis:alpine                                   prueba
Running        Running about a minute ago
xabhg7dqckje  prueba_vote.2         dockersamples/examplevotingapp_vote:before     prueba02
Running        Running about a minute ago
```

2.12.5 Eliminar un stack

Podemos eliminar un stack de servicios con la opción *rm* de *docker stack*.

```
nano@satellite:~$ docker stack rm prueba
Removing service prueba_redis
Removing service prueba_result
Removing service prueba_worker
Removing service prueba_visualizer
Removing service prueba_vote
Removing service prueba_db
Removing network prueba_default
Removing network prueba_frontend
Removing network prueba_backend
```

2.13 INFORMACIÓN DEL SISTEMA

```
nano@satellite:~$ docker system

Usage: docker system COMMAND

Manage Docker

Options:
      --help   Muestra su uso

Commands:
  df          Muestra el disco usado por Docker
  events      Muestra los eventos del servidor Docker en tiempo real
  info        Muestra información del todo sistema
  prune       Elimina datos no usados

Run 'docker system COMMAND --help' for more information on a command.
```

2.13.1 Mostrar espacio de disco usado por Docker

Muestra el espacio usado por Docker, detallando cuánto es ocupado por las imágenes de Docker, cuánto por los contenedores y poor los volúmenes. Incluye el número total de elementos y los activos actualmente.

```
nano@satellite:~$ docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              4                   3                   248.4 MB            15.49 MB (6%)
Containers          10                  0                   11 B                11 B (100%)
Local Volumes       20                  0                   625.6 MB            625.6 MB
(100%)
```

2.13.2 Mostrar los eventos de Docker en tiempo real

Muestra en tiempo real los eventos que se ocurren en la instalación de Docker. En este caso un extracto tras la ejecución de `docker system prune`

```
nano@satellite:~$ docker system events
2017-05-16T11:06:55.312285709+02:00 container destroy
18a5b622035f92456711e50ee5bf417050b4478c81437444264b87643e55a97a (image=hello-world,
name=prueba)
2017-05-16T11:06:55.273533792+02:00 container destroy
dc41ea7b360599745a2c812f05f2b846b65c697f30db5a43b1e3d213c861e377 (image=hello-world,
name=dreamy_cray)
2017-05-16T11:06:55.351868008+02:00 container destroy
020cece2c065de3bacf1d25da5170d5bb34da4243f6bc1172575c9988f019f36 (image=nginx,
name=modest_shockley)
(...)
```

2.13.3 Mostrar información de la instalación de Docker

Con la ejecución de este comando podemos ver toda la información del sistema Docker.

Muestra información variada de Docker (número de contenedores y imágenes, versión de Docker, número de nodos de Swarm...) así como información propia del sistema host (sistema operativo, versión del kérnel, CPU's, memoria RAM...)

```
nano@satellite:~$ docker info
Containers: 10
 Running: 0
 Paused: 0
 Stopped: 10
Images: 4
Server Version: 17.03.1-ce
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: xfs
 Dirs: 35
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: active
 NodeID: 3xwj9vrt255deqistbwcebdk2
 Is Manager: true
 ClusterID: zur9v9u86u1jwzfr1bg5483gj
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
 Node Address: 192.168.1.11
 Manager Addresses:
  192.168.1.11:2377
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 4ab9917febca54791c5f071a9d1f404867857fcc
runc version: 54296cf40ad8143b62dbcaa1d90e520a2136ddfe
init version: 949e6fa
Kernel Version: 3.16.0-4-amd64
Operating System: Debian GNU/Linux 8 (jessie)
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 7.746 GiB
```

```
Name: satellite
ID: DZT3:IZQZ:J235:AVEA:A6KM:YUDK:PKYZ:A67I:FZOW:SVB2:NGKP:JO4J
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Username: evaristogz
Registry: https://index.docker.io/v1/
WARNING: No memory limit support
WARNING: No swap limit support
WARNING: No kernel memory limit support
WARNING: No oom kill disable support
WARNING: No cpu cfs quota support
WARNING: No cpu cfs period support
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

2.13.4 Eliminar elementos que no estén en uso

Unifica todas las opciones prune de contenedores, volúmenes, redes e imágenes, borrando todas aquellas que no estén en uso.

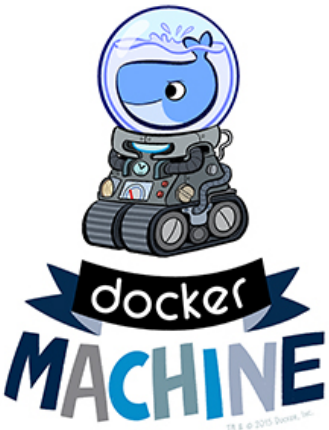
```
nano@satellite:~$ docker system prune
WARNING! This will remove:
       - all stopped containers
       - all volumes not used by at least one container
       - all networks not used by at least one container
       - all dangling images
Are you sure you want to continue? [y/N] y
Deleted Containers:
dc41ea7b360599745a2c812f05f2b846b65c697f30db5a43b1e3d213c861e377
18a5b622035f92456711e50ee5bf417050b4478c81437444264b87643e55a97a
020cece2c065de3bacf1d25da5170d5bb34da4243f6bc1172575c9988f019f36
(...)

Deleted Volumes:
04a60c7c7125f72e35b138da4af3a1634e9262caf92ccb6a06cdab7e9ac7aa7c
66c142b61eb1376e7a55f9c35c60ee6e3356880869d2cc42894a8c99634fb75c
8967c318cf59bb1a9435330bb371af0c9f16abf7c6af1c3abe61ac4037d54d61
(...)

Deleted Networks:
prueba

Total reclaimed space: 625.6 MB
nano@satellite:~$ docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              4                   0                   248.4 MB            248.4 MB
(100%)
Containers          0                   0                   0 B                 0 B
Local Volumes       0                   0                   0 B                 0 B
```

3. DOCKER MACHINE



Esta herramienta permite crear, configurar y administrar nodos físicos o virtuales a través de drivers o conectores. Instala en los nodos una distribución de Tiny Core Linux llamada boot2docker, que incluye un Docker Engine. De manera que es posible conectar nuestro Docker Client con el Docker Engine de ese nodo.

Existen multitud de drivers o conectores para interactuar con máquinas de las principales plataformas de IaaS como Amazon Web Service, Microsoft Azure, Google Compute Engine, Digital Ocean o plataformas de virtualización de escritorio como VMware Fusion, VMware vSphere o Oracle VirtualBox.

En nuestro caso utilizaremos VirtualBox, pero si estás interesado en utilizar otro servicio, puedes ver la lista de los catorce conectores disponibles y su documentación en:
https://docs.docker.com/machine/drivers/

3.1 INSTALACIÓN DE DOCKER MACHINE

Podemos instalar Docker Machine a través de los binarios. Esta herramienta está disponible para macOS, Windows y Linux: https://docs.docker.com/machine/install-machine/

En nuestro caso, la instalación se ejecuta descargando el paquete, ortogándole permisos y ubicando sus ficheros en el directorio correspondiente.

Por ello, con usuario con privilegios de root lanzamos la siguiente ejecución:

```
curl -L https://github.com/docker/machine/releases/download/v0.10.0/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine && chmod +x /tmp/docker-machine && sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

```
root@satellite:/home/nano# curl -L
https://github.com/docker/machine/releases/download/v0.10.0/docker-machine-`uname -s`-`uname
-m` >/tmp/docker-machine &&
>   chmod +x /tmp/docker-machine &&
>   sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   600    0   600    0     0    985      0 --:--:-- --:--:-- --:--:--   985
100 24.1M  100 24.1M    0     0  1972k      0  0:00:12  0:00:12 --:--:-- 3844k
root@satellite:/home/nano# docker-machine --version
docker-machine version 0.10.0, build 76ed2a6
```

Ayuda del comando *docker-machine*

```
nano@satellite:~$ docker-machine
Usage: docker-machine [OPTIONS] COMMAND [arg...]

Create and manage machines running Docker.

Version: 0.10.0, build 76ed2a6

Author:
  Docker Machine Contributors - <https://github.com/docker/machine>

Options:
  --debug, -D                                   Activa el modo debug
  --storage-path, -s "/home/nano/.docker/machine"    Configura la ruta de almacenamiento
[$MACHINE_STORAGE_PATH]
  --tls-ca-cert                                 Certifica de la CA [$MACHINE_TLS_CA_CERT]
  --tls-ca-key                                  Clave privada para generar certificados
[$MACHINE_TLS_CA_KEY]
  --tls-client-cert                                   Certificado del cliente para usar TLS
[$MACHINE_TLS_CLIENT_CERT]
  --tls-client-key                              Clave privada usada en el cliente TLS
[$MACHINE_TLS_CLIENT_KEY]
```

```
  --github-api-token                                  Token para usar peticiones a la API de
Github [$MACHINE_GITHUB_API_TOKEN]
  --native-ssh                                        Usa la implemntación nativa de SSH (Go-
based) [$MACHINE_NATIVE_SSH]
  --bugsnag-api-token                                 Token para la API de BugSnag API sobre
errores [$MACHINE_BUGSNAG_API_TOKEN]
  --help, -h                                  Muestra la ayuda
  --version, -v                               Muestra la versión

Commands:
  active            Muestra qué máquina está activa
  config            Muestra la configuración de conexión para una máquina
  create            Crea una máquina
  env               Muestra los comandos para configurar el entorno para Docker Client
  inspect           Muestra información sobre una máquina
  ip                Muestra la dirección IP de una máquina
  kill              Mata una máquina
  ls                Lista las máquinas
  provision         Reprovisiona las máquinas existentes
  regenerate-certs  Regenera los certificados TLS para una máquina
  restart           Reinicia una máquina
  rm                Elimina una máquina
  ssh               Log into or run a command on a machine with SSH.
  scp               Copia ficheros entre máquinas
  start             Inicia una máquina
  status            Muestra el estado de una máquina
  stop              Para una máquina
  upgrade           Actualiza una máquina a la última versión de Docker
  url               Muestra la URL de una máquina
  version           Muestra la versión de Docker Machine
  help              Muestra una lista de comandos o ayuda para un comando

Run 'docker-machine COMMAND --help' for more information on a command.
```

3.2 CREAR UN NUEVO NODO

Docker Machine permite el uso de distintos conectores para realizar los nodos sobre los que queremos trabajar. En nuestro caso, la conexión se hará con el gestor de máquinas virtuales VirtualBox, por lo que se sobreentiende que esto se realizará para entornos de desarrollo.

Para ello nos bastará con la ejecución del comando: `docker-machine create --driver virtualbox prueba` que creará una máquina virtual con el nombre *prueba* e instalará en él una distribución de Linux muy reducida en la que se incluye Docker Engine (boot2docker).

```
nano@satellite:~$ docker-machine create -d virtualbox prueba
Running pre-create checks...
(dev) You are using version 4.3.36_Debianr105129 of VirtualBox. If you encounter issues, you
might want to upgrade to version 5 at https://www.virtualbox.org
Creating machine...
(dev) Copying /home/nano/.docker/machine/cache/boot2docker.iso to
/home/nano/.docker/machine/machines/prueba/boot2docker.iso...
(dev) Creating VirtualBox VM...
(dev) Creating SSH key...
(dev) Starting the VM...
(dev) Check network to re-create if needed...
(dev) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
machine, run: docker-machine env prueba
```

Como vemos en la salida, su ejecución realiza varias tareas: crea la máquina virtual, crea el par de claves SSH, inicia la máquina virtual, comprueba la red, espera a que se le asigne una dirección IP, copia los certificados en el directorio local y en la máquina remota, configura el demonio de Docker...

A través de Docker Machine podemos especificar las distintas características que ha de tener ese nodo de VirtualBox como el número de CPU's, la memoria RAM, el tamaño de disco duro, la dirección IP... Por defecto la máquina virtual se crea con 1 CPU, 1GB de memoria RAM y 20GB de almacenamiento.

Estos parámetros dependen de cada conector de Docker Machine, siendo posible en conectores como AWS elegir la región del nodo, la dirección privada o las claves, por ejemplo.

En la documentación del driver de VirtualBox podemos consultar todas las opciones que permite: https://docs.docker.com/machine/drivers/virtualbox/

3.2.1 Crear un nuevo nodo en Amazon Web Services

Siendo Amazon Web Service otro de los servicios más populares hemos realizado una prueba en esta plataforma PaaS.

Lo primero será acceder a nuestra cuenta de AWS a través de https://aws.amazon.com y hacer click en nuestro nombre situado en la parte superior derecha. Pulsamos sobre *My Security Credentials* y hacemos click en el botón *Continue to Security Credentials*. Allí desplegamos el apartado *Access Keys (Access Key ID and Secret Access Key)*

Pulsamos en el botón azul *Create New Access Key* y descargamos el fichero que contiene el *Access Key ID* y la *Secret Access Key*.

```
nano@satellite:~$ docker-machine create -d amazonec2 --amazonec2-access-key
AKIAJD57233YLAWOJTJQ --amazonec2-secret-key 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
pruebaAWS
Running pre-create checks...
Creating machine...
(pruebaAWS) Launching instance...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with ubuntu(systemd)...
Installing Docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
machine, run: docker-machine env pruebaAWS
nano@satellite:~$ docker-machine ls
NAME        ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
ERRORS
prueba      -        virtualbox   Running   tcp://192.168.99.100:2376           v17.05.0-ce
prueba02    -        virtualbox   Running   tcp://192.168.99.101:2376           v17.05.0-ce
pruebaAWS   -        amazonec2    Running   tcp://54.175.92.114:2376            v17.05.0-ce
nano@satellite:~$ docker-machine ssh pruebaAWS
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-57-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

137 packages can be updated.
49 updates are security updates.


*** System restart required ***
```

```
ubuntu@pruebaAWS:~$ docker --version
Docker version 17.05.0-ce, build 89658be
ubuntu@pruebaAWS:~$ exit
logout
nano@satellite:~$ docker-machine rm pruebaAWS
About to remove pruebaAWS
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed pruebaAWS
nano@satellite:~$ docker-machine inspect pruebaAWS
{
    "ConfigVersion": 3,
    "Driver": {
        "IPAddress": "54.158.117.176",
        "MachineName": "pruebaAWS",
        "SSHUser": "ubuntu",
        "SSHPort": 22,
        "SSHKeyPath": "/home/nano/.docker/machine/machines/pruebaAWS/id_rsa",
        "StorePath": "/home/nano/.docker/machine",
        "SwarmMaster": false,
        "SwarmHost": "tcp://0.0.0.0:3376",
        "SwarmDiscovery": "",
        "Id": "dde91dc94836d9e67ef67e1fd3672e4c",
        "AccessKey": "AKIAJD57233YLAWOJTJQ",
        "SecretKey": "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",
        "SessionToken": "",
        "Region": "us-east-1",
        "AMI": "ami-9dcfdb8a",
        "SSHKeyID": 0,
        "ExistingKey": false,
        "KeyName": "pruebaAWS",
        "InstanceId": "i-0b541248d5dcaeaaf",
        "InstanceType": "t2.micro",
        "PrivateIPAddress": "172.31.34.114",
        "SecurityGroupId": "",
        "SecurityGroupIds": [
            "sg-30b34741"
        ],
        "SecurityGroupName": "",
        "SecurityGroupNames": [
            "docker-machine"
        ],
        "OpenPorts": null,
        "Tags": "",
        "ReservationId": "",
        "DeviceName": "/dev/sda1",
        "RootSize": 16,
        "VolumeType": "gp2",
        "IamInstanceProfile": "",
        "VpcId": "vpc-1508ab6c",
        "SubnetId": "subnet-9313b4c9",
        "Zone": "a",
        "RequestSpotInstance": false,
        "SpotPrice": "0.50",
        "BlockDurationMinutes": 0,
        "PrivateIPOnly": false,
        "UsePrivateIP": false,
        "UseEbsOptimizedInstance": false,
        "Monitoring": false,
        "SSHPrivateKeyPath": "",
        "RetryCount": 5,
        "Endpoint": "",
        "DisableSSL": false,
        "UserDataFile": ""
    },
    "DriverName": "amazonec2",
    "HostOptions": {
        "Driver": "",
        "Memory": 0,
        "Disk": 0,
        "EngineOptions": {
            "ArbitraryFlags": [],
            "Dns": null,
            "GraphDir": "",
```

```
            "Env": [],
            "Ipv6": false,
            "InsecureRegistry": [],
            "Labels": [],
            "LogLevel": "",
            "StorageDriver": "",
            "SelinuxEnabled": false,
            "TlsVerify": true,
            "RegistryMirror": [],
            "InstallURL": "https://get.docker.com"
        },
        "SwarmOptions": {
            "IsSwarm": false,
            "Address": "",
            "Discovery": "",
            "Agent": false,
            "Master": false,
            "Host": "tcp://0.0.0.0:3376",
            "Image": "swarm:latest",
            "Strategy": "spread",
            "Heartbeat": 0,
            "Overcommit": 0,
            "ArbitraryFlags": [],
            "ArbitraryJoinFlags": [],
            "Env": null,
            "IsExperimental": false
        },
        "AuthOptions": {
            "CertDir": "/home/nano/.docker/machine/certs",
            "CaCertPath": "/home/nano/.docker/machine/certs/ca.pem",
            "CaPrivateKeyPath": "/home/nano/.docker/machine/certs/ca-key.pem",
            "CaCertRemotePath": "",
            "ServerCertPath": "/home/nano/.docker/machine/machines/pruebaAWS/server.pem",
            "ServerKeyPath": "/home/nano/.docker/machine/machines/pruebaAWS/server-key.pem",
            "ClientKeyPath": "/home/nano/.docker/machine/certs/key.pem",
            "ServerCertRemotePath": "",
            "ServerKeyRemotePath": "",
            "ClientCertPath": "/home/nano/.docker/machine/certs/cert.pem",
            "ServerCertSANs": [],
            "StorePath": "/home/nano/.docker/machine/machines/pruebaAWS"
        }
    },
    "Name": "pruebaAWS"
}
```


3.3 CREAR UN NUEVO NODO CON ESPECIFICACIONES Y CONSULTARLAS

Como ejemplo, crearemos una máquina virtual con especificaciones antes de instanciar.

Detallaremos la memoria RAM (2GB), el número de CPU's (2 CPU's), el disco duro (10GB) y la versión 1.12.6 de boot2docker con el comando: `docker-machine create -d virtualbox --virtualbox-memory "2048" --virtualbox-cpu-count "2" --virtualbox-disk-size "10000" --virtualbox-boot2docker-url="https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso" prueba`

```
nano@satellite:~$ docker-machine create -d virtualbox --virtualbox-memory "2048"
--virtualbox-cpu-count "2" --virtualbox-disk-size "10000" --virtualbox-share-folder
"/home/nano:satellite" --virtualbox-boot2docker-
url="https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso"
prueba02
Running pre-create checks...
(prueba) Boot2Docker URL was explicitly set to
"https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso" at
create time, so Docker Machine cannot upgrade this machine to the latest version.
Creating machine...
(prueba) Boot2Docker URL was explicitly set to
```

```
"https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso" at
create time, so Docker Machine cannot upgrade this machine to the latest version.
(prueba) Downloading /home/nano/.docker/machine/cache/boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso...
(prueba) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
(prueba) Creating VirtualBox VM...
(prueba) Creating SSH key...
(prueba) Starting the VM...
(prueba) Check network to re-create if needed...
(prueba) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
machine, run: docker-machine env prueba
```

Podemos ver la información de ese nodo a través del parámetro *inspect* que devuelve los datos en formato JSON:

- `docker-machine inspect prueba`
- `docker-machine inspect --format='{{prettyjson .Driver}}' prueba`
- `docker-machine inspect --format='{{.Driver.Memory}}' prueba`

```
nano@satellite:~$ docker-machine inspect --format='{{prettyjson .Driver}}' prueba
{
    "Boot2DockerImportVM": "",
    "Boot2DockerURL":
"https://github.com/boot2docker/boot2docker/releases/download/v1.12.6/boot2docker.iso",
    "CPU": 2,
    "DNSProxy": true,
    "DiskSize": 10000,
    "HostDNSResolver": false,
    "HostInterfaces": {},
    "HostOnlyCIDR": "192.168.99.1/24",
    "HostOnlyNicType": "82540EM",
    "HostOnlyNoDHCP": false,
    "HostOnlyPromiscMode": "deny",
    "IPAddress": "192.168.99.100",
    "MachineName": "prueba",
    "Memory": 2048,
    "NatNicType": "82540EM",
    "NoShare": false,
    "NoVTXCheck": false,
    "SSHKeyPath": "/home/nano/.docker/machine/machines/prueba/id_rsa",
    "SSHPort": 35481,
    "SSHUser": "docker",
    "ShareFolder": "/home/nano:satellite",
    "StorePath": "/home/nano/.docker/machine",
    "SwarmDiscovery": "",
    "SwarmHost": "tcp://0.0.0.0:3376",
    "SwarmMaster": false,
    "UIType": "headless",
    "VBoxManager": {}
}
```

3.4 LISTAR NODOS

Para ver los nodos que nuestro Docker Machine puede gestionar, lo hacemos a través de la ejecución de `docker-machine ls`

```
nano@satellite:~$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
ERRORS
```

```
prueba    -        virtualbox   Running   tcp://192.168.99.100:2376           v17.05.0-ce
prueba02  -        virtualbox   Running   tcp://192.168.99.101:2376           v17.05.0-ce
```

En este caso aparece el nodo creado con especificaciones estándar y el nodo creado con especificaciones detalladas.

3.5 MOSTRAR CONFIGURACIÓN DE CONEXIÓN

Podemos consultar las rutas de los certificados y claves, además como la dirección IP de una máquina con el parámetro *config*

```
nano@satellite:~$ docker-machine config prueba
--tlsverify
--tlscacert="/home/nano/.docker/machine/machines/prueba/ca.pem"
--tlscert="/home/nano/.docker/machine/machines/prueba/cert.pem"
--tlskey="/home/nano/.docker/machine/machines/prueba/key.pem"
-H=tcp://192.168.99.100:2376
```

3.6 MOSTRAR LA DIRECCIÓN IP DE UN NODO

Podemos extrar la dirección IP a través del parámetro *inspect* y parseando el JSON que devuelve su ejecución o haciendo uso de `docker-machine ip prueba`

```
nano@satellite:~$ docker-machine ip prueba
192.168.99.100
nano@satellite:~$ docker-machine inspect -f {{.Driver.IPAddress}} node-01
192.168.99.100
```

3.7 MOSTRAR NODO EN EL QUE NOS ENCONTRAMOS

El parámetro *active* permite extraer el nombre del nodo de Docker Machine al que estamos conectados.

Su ejecución es simple `docker-machine active`

```
nano@satellite:~$ docker-machine active
prueba
```

3.8 MOSTRAR ESTADO DE UN NODO

Con el comando `docker-machine status` prueba podemos saber el estado en el que se encuentra un nodo en concreto.

```
nano@satellite:~$ docker-machine status prueba
Running
```

Puede conocerse igualmente con `docker-machine ls` que mostrará el listado de todos los nodos y distintos valores.

3.9 CONECTAR DOCKER CLIENT CON EL DOCKER ENGINE DE UN NODO

Para conectar el cliente de Docker con el Docker Engine que se ejecuta en el nodo debemos de setear las variables, las cuales consultamos con el comando `docker-machine env prueba` y estableemos con `eval $(docker-machine env prueba)`

```
nano@satellite:~$ docker-machine env prueba
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/home/nano/.docker/machine/machines/prueba"
export DOCKER_MACHINE_NAME="dev"
# Run this command to configure your shell:
# eval $(docker-machine env dev)
nano@satellite:~$ eval $(docker-machine env prueba)
nano@satellite:~$ env | grep DOCKER_*
```

```
DOCKER_HOST=tcp://192.168.99.100:2376
DOCKER_MACHINE_NAME=prueba
DOCKER_TLS_VERIFY=1
DOCKER_CERT_PATH=/home/nano/.docker/machine/machines/prueba
```

De este modo, nuestro cliente está conectado con la VM con nombre *prueba*, que ejecuta dentro un Docker Engine. Por lo tanto, si ejecutamos `docker run hello-world` se hará en el nodo, al igual que cualquier otro comando de Docker Client.

3.10 COPIAR FICHEROS O DIRECTORIOS ENTRE ANFITRIÓN Y NODO

Con la función scp es posible copiar ficheros entre la máquina anfitriona donde se ejecuta Docker Machine (o Docker Client) y el nodo de Docker Machine.

ficheroanfitrion.txt indica la ruta del fichero anfitrión y a continuación el nombre del nodo y el fichero o ruta de destino.

```
nano@satellite:~$
nano@satellite:~$ docker-machine scp -r /ruta/ficheroanfitrion.txt prueba:ficheronodo.txt
ficheroanfitrion.txt                                          100%   32     0.0KB/s   00:00
```

También es posible hacerlo con directorios utilizando el parámetro que lo hará de forma recursiva *-r*

3.11 ACCEDER A UN NODO MEDIANTE SSH

Si por alguna razón queremos acceder a un nodo controlado por Docker Machine, podemos hacerlo con la ejecución de `docker-machine ssh prueba`

También lo podemos hacer mediante el comando ssh, con usuario docker y contraseña tcuser: `ssh docker@192.168.99.100`

Aún así, al igual que en los contenedores Docker, no es recomendable realizar configuraciones manuales dentro de la máquina virtual o nodo.

```
nano@satellite:~$ docker-machine ssh prueba
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 17.05.0-ce, build HEAD : 5ed2840 - Fri May  5 21:04:09 UTC 2017
Docker version 17.05.0-ce, build 89658be
docker@prueba:~$
```

3.12 INICIAR Y PARAR UN NODO

Podemos arrancar un nodo de Docker Machine con el comando `docker-machine start prueba`
Como bien reza en la ejecución, la máquina se iniciará con una nueva dirección IP y deberemos de consultar las variables del nodo nuevamente para poder conectar.

```
nano@satellite:~$ docker-machine start prueba
Starting "prueba"...
(prueba) Check network to re-create if needed...
(prueba) Waiting for an IP...
Machine "prueba" was started.
Waiting for SSH to be available...
Detecting the provisioner...
Started machines may have new IP addresses. You may need to re-run the `docker-machine env`
command.
```

Para parar una máquina virtual o nodo lo hacemos con la opción *stop* del comando *docker-machine*.

```
nano@satellite:~$ docker-machine stop prueba
Stopping "prueba"...
Machine "prueba" was stopped.
```

Con `docker-machine start $(docker-machine ls -q)` o `docker-machine stop $(docker-machine ls -q)` iniciamos o paramos todos los nodos de los que disponga ese Docker Machine. El problema que puede causar iniciar todos los nodos a la vez es que se asignen direcciones IP distintas a las que se le asignaron por primera vez, teniendo que regenerar los certificados de cada nodo de nuevo.

También existe la opción de matar un nodo, que parará un nodo de Docker Machine de manera forzosa. Podemos hacerlo con la ejecución de `docker-machine kill prueba`

3.13 BORRAR NODOS

Si deseamos borrar uno de nuestros nodos podemos hacerlo con `docker-machine rm prueba`

Esto eliminará el nodo y todo su contenido, independientemente de si este se encuentra en ejecución o no.

```
nano@satellite:~$ docker-machine rm prueba
About to remove prueba
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed prueba
```

Con `docker-machine rm $(docker-machine ls -q)` eliminamos todos los nodos de los que disponga ese Docker Machine.

```
nano@satellite:~$ docker-machine rm $(docker-machine ls -q)
About to remove prueba, prueba02
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed prueba
Successfully removed prueba02
```

3.14 REGENERAR CERTIFICADOS DE UN NODO

En ocasiones, es posible que al iniciar un nodo con el comando `docker-machine start prueba` dé error el certificado que utiliza para la conexión entre el nodo y nuestra instalación de Docker Machine.

Esto se debe a que la dirección IP asignada a ese nodo ha cambiado y el certificado es válido para una dirección IP distinta a la asignada actualmente.

Se soluciona regenerando el certificado de dicho nodo con `docker-machine regenerate-certs prueba`

```
nano@satellite:~$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
ERRORS
prueba    -        virtualbox   Running   tcp://192.168.99.100:2376           Unknown
Unable to query docker version: Get https://192.168.99.101:2376/v1.15/version: x509:
certificate is valid for 192.168.99.108, not 192.168.99.100
nano@satellite:~$ docker-machine env prueba
Error checking TLS connection: Error checking and/or regenerating the certs: There was an
error validating certificates for host "192.168.99.100:2376": x509: certificate is valid for
192.168.99.108, not 192.168.99.100
You can attempt to regenerate them using 'docker-machine regenerate-certs [name]'.
Be advised that this will trigger a Docker daemon restart which might stop running
containers.

nano@satellite:~$ docker-machine regenerate-certs prueba
Regenerate TLS machine certs?  Warning: this is irreversible. (y/n): y
Regenerating TLS certificates
```

```
Waiting for SSH to be available...
Detecting the provisioner...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
nano@satellite:~$ docker-machine env prueba
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/home/nano/.docker/machine/machines/prueba"
export DOCKER_MACHINE_NAME="prueba"
# Run this command to configure your shell:
# eval $(docker-machine env prueba)
```

3.15 ERROR DE ADAPTADORES DE VIRTUALBOX

Docker Machine crea un adaptador de VirtualBox con la dirección IP 192.168.99.1 Al realizar pruebas, es posible que no permita crear los nodos pues VirtualBox encuentra otros adaptadores con la misma dirección IP.

```
nano@satellite:~$ docker-machine create -d virtualbox prueba
Running pre-create checks...
(prueba) You are using version 4.3.36_Debianr105129 of VirtualBox. If you encounter issues,
you might want to upgrade to version 5 at https://www.virtualbox.org
Error with pre-create check: "VirtualBox is configured with multiple host-only adapters with
the same IP \"192.168.99.1\". Please remove one"
```

Para solucionar este problema podemos configurar nuevamente los adaptadores de red de VirtualBox o directamente eliminarlos. VirtualBox gestiona las interfaces de red desde Archivo>Preferencias...>Red y Redes solo-anfitrión.

4. DESPLIEGUE DE APLICACIÓN DE EJEMPLO EN DOCKER SWARM

El escenario se comprende de tres nodos virtualizados en VirtualBox, cuyo anfitrión es una máquina Debian Jessie 8.8. Estos se despliegan a través de Docker Machine con un script en Bash que permite hacerlo de manera automatizada.

A su vez, inicializa un Docker Swarm en estos nodos de Docker Machine. La ejecución de esta aplicación se realiza a través de un stack de servicios, que despliega hasta un total de seis servicios en los distintos nodos del clúster.

La aplicación consta de tres servicios accesibles a través del navegador web: una aplicación web para votar, una aplicación web para ver los resultados y una herramienta web para visualizar los nodos que forman el clúster y los servicios desplegados dentro de ellos.

La aplicación que se despliega está alojada en un repositorio GitHub llamado example-voting-app. (Ver diagrama de la arquitectura)

Todo el proceso explicado en los siguientes puntos está automatizado a través de scripts interactivos escritos en bash que muestran información sobre el proceso. Estos están alojados en un repositorio GitHub llamado tutorial-Docker-Swarm.

4.1 CREAR UN CLÚSTER DE TRES NODOS

A través de Docker Machine, crearemos un clúster de tres nodos (un manager y dos workers) sobre VirtualBox. Lo haré de manera automatizada a través de un script en bash, que realizará el proceso de manera independiente.

Lo primero que hará será crear tres nodos a través de un bucle: *manager-01*, *nodo-02* y *nodo-03* a través del comando `docker-machine create -d virtualbox nombrenodo`

Luego consultará la dirección IP del nodo Manager y la guardará en una variable (*$MANAGER_IP*). Seguidamente establecerá las variables de entorno del nodo llamado *manager-01* para conectar la CLI de Docker con el Docker Engine de ese nodo. Allí iniciará el Swarm con `docker swarm init --advertise-addr $MANAGER_IP` y añadirá un label al nodo con el nombre del administrador del Swarm.

Almacenará los tokens de unión en variables (*$MANAGER_TOKEN* y *$WORKER_TOKEN*). Estos permitirán unir un nodo como Manager o como Worker al Swarm.

Mediante un bucle accederá a los nodos *nodo-02* y *nodo-03*, guardará la dirección IP de ese nodo en la variable *$WORKER_IP*. Luego establece las variables de entorno y conecta el cliente de Docker con el Docker Engine del nodo de Docker Machine. Utilizando las variables *$WORKER_TOKEN*, *$WORKER_IP* y *$MANAGER_IP* ejecutará el comando de unión al clúster de Swarm.

Por último, conectará el Docker Client con el Docker Engine de *manager-01* y ejecutará `docker-machine ls` para ver los nodos de Docker Machine (en este caso de VirtualBox) y los nodos de Swarm así como su información con `docker node ls`

4.2 DESPLEGAR UN STACK DE SERVICIOS

Con el fichero *docker-stack.yml* desplegamos un stack que contiene seis servicios:

- ***redis***: ejecuta una imagen de redis, expone el puerto 6379 y está conectado a la red *frontend*.
- ***db***: ejecuta una imagen de Postgres 9.4, asocia un volumen para los datos de PostgreSQL y está conectado a la red *backend*.
- ***vote***: ejecuta una imagen para votar de la aplicación de ejemplo y expone el puerto 80 del servicio al 5000 del nodo. Se conecta a la red *frontend* y depende del servicio *redis*.

- ***result***: ejecuta una imagen para mostrar resultados de la aplicación de ejemplo y expone el puerto 80 del servicio al 5001 del nodo. Se conecta a la red *backend* y depende del servicio *db*.
- ***worker***: ejecuta una imagen que escribe los votos de *redis* en *db*.
- ***visualizer***: ejecuta la imagen Visualizer, que permite consultar los nodos de Docker Swarm a través de un navegador web.

También creará dos redes:
- ***frontend***: a la que se conectan los servicios *redis*, *vote* y *worker*
- ***backend***: a la que se conectan los servicios *db* y *worker*.

Por último, creará un volumen llamado *db-data* que se asocia al servicio *db*

Este fichero YML contiene además una serie de cláusulas de despliegue: número de réplicas, paralelismo, política de reinicio, constraints... y lo lanzamos con `docker stack deploy -c example-voting-app/docker-stack.yml` encuesta desde el nodo *manager-01*.

Tras el despliegue del stack se consulta los servicios del stack mediante el comando `docker stack services encuesta` en él se ve que no ha terminado de desplegar todas las réplicas.

Nuevamente se comprueba el número de réplicas de los servicios desplegados por el stack *encuesta*.

Mediante un bucle, se accederá a todos los nodos (*manager-01*, *nodo-02* y *nodo-03*) para ejecutar docker ps dentro de ellos y ver las tareas que se están ejecutando en cada uno de ellos.

Las tareas son invocadas por el Manager Docker Swarm al crear un servicio y estas se despliegan en contenedores. No es correcto llamar contenedores cuando estamos hablando de tareas o réplicas de un servicio.

Por lo tanto, tienen esta "jerarquía" por así decirlo: manager > servicios > tareas > contenedores.

4.3 LISTAR TAREAS DEL STACK

Utilizando un bucle, accedemos a todos los nodos para listar las tareas de encuesta_vote que son las que reciben las peticiones a través del puerto 5000 del nodo.

Lo hacemos utilizando un filtro de formato, de manera que muestra el ID de la tarea y su nombre (que llevará una numeración según el número de réplica que sea). Este comando es: `docker ps --format "{{.ID}} {{.Names}}" --filter Name=encuesta_vote`

Se informa que se puede acceder a los servicios:

- Vote a través de la dirección IP y puerto: http://192.168.99.100:5000
- Result a través de la dirección IP y puerto: http://192.168.99.100:5001
- Visualizer a través de la dirección IP y puerto: http://192.168.99.100:8080

Igualmente, se puede acceder a través de la dirección IP de cualquier nodo funcionando la resolución gracias al routing mesh que realiza Docker Swarm.

4.4 CAMBIAR DISPONIBILIDAD DEL MANAGER A PAUSE

El parámetro Availability puede tener tres valores: *Active*, *Pause* o *Drain*. *Active* es el valor por defecto, que hará que ese nodo reciba nuevas tareas de un servicio para ejecutarlas. Es decir, actuará como un worker.

Pause hará que ese nodo de Swarm no esté disponible para ejecutar nuevas tareas en él, manteniendo las que estén corriendo actualmente. Mientras que *Drain* hará que el nodo deje de ejecutar cualquier tarea y pasen a ser ejecutadas en otros nodos del clúster.

En el nodo *manager-01* listamos los nodos que componen el clúster de Swarm, para comprobar el

valor Availability de ellos.

Filtramos las tareas que se están ejecutando en *manager-01* con la ejecución de *docker ps* con un filtro: `docker ps --filter name=encuesta_`

A continuación, cambiamos la disponibilidad del propio nodo manager-01 actualizando el parámetro *availability*: `docker node update --availability pause manager-01`

Volvemos a comprobar el valor *Availability* de los nodos y comprobamos qué las tareas que se estaban ejecutando en el nodo *manager-01* siguen ejecutándose en él sin problema.

4.5 ACTUALIZAR EL STACK Y ESCALAR EL SERVICIO VOTE

Para actualizar el stack, hemos creado una copia nueva en la que se define el servicio *vote* con 15 réplicas en vez de 2.

Fragmento del fichero docker-stack-scale.yml

```
(...)
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 15
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
(...)
```

Su actualización se realiza de la misma manera que un despliegue nuevo, de manera que se sobreescribirán los cambios actualizando servicio a servicio.

Esta actualización solo puede realizarse desde un nodo manager, como en este caso *manager-01*.

Primero mostramos el número de réplicas de servicios del stack *encuesta* ejecutando el comando `docker stack services encuesta` donde se ve que el servicio *encuesta_vote* tiene dos réplicas desplegadas en el clúster.

Luego, ejecutamos el comando `docker stack deploy -c example-voting-app/docker-stack-scale.yml encuesta` que actualizará el stack *encuesta* con unas nuevas cláusulas.

Esperamos 30 segundos y pasamos a comprobar cuántas réplicas de los servicios se han desplegado. Esto lo realizamos ejecutando `docker stack services encuesta` Por lo general, 30 segundos son suficientes para que el servicio se escale a 15 réplicas.

Vuelve a ejecutarse el script número 3 que hará que muestre las tareas del servicio *encuesta_vote* que se están ejecutando en todos los nodos.

A través de la página web podremos ver el ID de los contenedores o tareas que están procesando la petición en ese momento.

4.6 COMPROBAR LA DISPONIBILIDAD DE MANAGER Y CAMBIAR A DRAIN

Tras escalar el servicio *encuesta_vote* a 15 réplicas, pasaremos a comprobar que el nodo *manager-01* con valor *Pause* en el parámetro *Availability* no ha ejecutado ninguna tarea más.

Para ello accedemos al nodo de Docker Machine con `eval $(docker-machine env manager-01)` y ejecutamos un listado de los contenedores que están ejecutando las tareas de encuesta con el comando `docker ps --filter name=encuesta_`

También comprobamos el número de réplicas de los servicios del stack, para ver que siguen siendo 15 en el caso de *encuesta_vote*.

Volvemos a comprobar la disponibilidad de los nodos, en este caso para corroborar que *manager-01* está en *Pause*.

A continuación, cambiamos la disponibilidad del nodo *manager-01* a *Drain* de manera que deje de ejecutar todas las tareas invocadas por los servicios. Volvemos a comprobar el campo Availability de *manager-01* para ver que ha cambiado a *Drain*.

Ahora, comprobamos las réplicas de los servicios desplegados. En un primer momento se ven afectados los servicios *encuesta_db* y *encuesta_worker*, más tarde dejará de estar disponible el servicio *encuesta_visualizer*, que también se ejecutaba en el nodo *manager-01* pero que tenía un periodo de gracia de un minuto y medio.

Fragmento del fichero docker-stack-scale.yml

```
(...)
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
(...)
```

Podemos comprobar que esos tres servicios no volverán a estar disponibles pase el tiempo que pase. Esto se debe a que tienen una cláusula que obliga a desplegar el servicio en un nodo con rol de manager, como en este caso el nodo *manager-01* no ejecuta tareas de contenedores, no podrá levantar ninguna réplica.

4.7 CAMBIAR CONSTRAINT DE LOS SERVICIOS DEL STACK

Al utilizar los servicios *encuesta_db*, *encuesta_worker* y *encuesta_visualizer* restricciones en cuanto al tipo de nodo en el que se despliega las tareas la opción es o bien crear otro nodo con rol de manager, o modificar el stack de servicios para que no tengan esa constraint.

Fragmento del fichero docker-stack-constraint.yml

```
(...)
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
(...)
  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
```

```
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]
(...)
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
(...)
```

Comprobamos las tareas que se ejecutan en *manager-01* y que pertenecen al stack de servicios *encuesta* ejecutando `docker ps --filter name=encuesta_`

Igualmente, comprobamos el número de réplicas de los servicios del stack observando que los tres servicios continúan con 0 réplicas.

Ahora, utilizando el nuevo fichero YML donde se definen los servicios del stack, actualizamos el stack de servicios desplegado en el clúster de Docker Swarm: `docker stack deploy -c example-voting-app/docker-stack-constraints.yml encuesta`

Esperamos 30 segundos para que las tareas puedan desplegarse en los nodos y comprobamos el stack de servicios con `docker stack services` encuesta

Comprobamos dónde se están ejecutando las tareas de encuesta_worker y encuesta_db con la ejecución de `docker service ps encuesta_worker` y `docker service ps encuesta_db`

4.8 ELIMINAR CONTENEDORES QUE EJECUTAN TAREAS PARA COMPROBAR EL CLÚSTER

Queremos recrear problemas de funcionamiento en el clúster de Docker Swarm, para ello eliminaremos todos los contenedores del *nodo-02*, los cuales ejecutan tareas de nuestro stack de servicios.

Antes, para verlo de manera más clara, escalamos el servicio de manera manual. Esto lo haremos sin actualizar el stack, sino atacando al servicio *encuesta_vote* en sí. Recordamos que este servicio es el que muestra la página web a través del puerto 5000.

Para escalarlo, ejecutamos `docker service scale encuesta_vote=30`

Tras una espera de 30 segundos, comprobamos el número de réplicas de los servicios del stack con docker stack services encuesta En este caso, ya están desplegadas las 30 réplicas del servicio de manera correcta.

Ahora, accedemos al *nodo-02*, listamos los contenedores del *nodo-02* y los borramos masivamente con `docker rm -f $(docker ps -aq)`

4.9 LISTAR LOS CONTENEDORES QUE EJECUTAN TAREAS DE ENCUESTA_VOTE

Accediendo al nodo *manager-01* listamos todas las tareas *encuesta_vote* que se ejecutan en los nodos, así como las que han sido paradas. Se observa multitud de tareas y estados (*Running*/*Shutdown*) al ejecutar `docker service ps encuesta_vote` en *manager-01*.

Continuamos consultando las tareas del servicio *encuesta_vote* que ejecuta el *nodo-03* que suele ser gran parte de ellas, quedando algunas en el *nodo-02* (en este caso solo 4 de 30). Ejecutamos `docker service ps -f node=nodo-03 encuesta_vote` para mostrar esta información.

4.10 ROLLBACK DE UN SERVICIO

Podemos volver a un estado anterior de un servicio a través del comando `docker service update --rollback encuesta_vote` Esto puede hacerse con un servicio en concreto, pero no con un stack de servicios.

Al realizar un rollback, devolvemos un servicio a su estado anterior. En este caso, el servicio *encuesta_vote* pasaría de las 30 réplicas a las 15 réplicas que se le asignaron anteriormente.

Al finalizar el rollback ejecutamos `docker stack services encuesta` para ver el número de réplicas.

4.11 DESPLEGAR UN SERVICIO DE REGISTRY Y PORTAINER

Los nodos managers suelen contener servicios más específicos, como pueden ser una base de datos o managers de otras aplicaciones como pueden ser de monitorización o centralización de logs.

En este caso, nuestro *manager-01* se encuentra aún en *Drain* por lo que no acepta despliegue de nuevos servicios o tareas. Comenzamos cambiando su disponibilidad por *Active* que permitirá que corra nuevos servicios y tareas: `docker node update --availability active manager-01`

4.11.1 Desplegar un servicio de Registry

Desde el nodo *manager-01* lanzamos un servicio para crear un Docker Registry. Esto permitirá tener nuestra imágenes alojadas en un entorno aislado y local, por lo que supondrá mejoras de seguridad (al ser nosotros quienes custodiamos esos datos) y mejoras en la velocidad (al formar parte de nuestra misma red).

Para lanzar este servicio utilizamos el siguiente comando, que incluye saltos de líneas para hacerlo más legible:

```
docker service create \
--name registry --constraint \
'node.role == manager' \
--publish 5050:5000 registry
```

Esto hará que se cree un servicio con el nombre *registry*, se ejecute en un nodo con rol manager, exponga el puerto 5000 del servicio al 5050 del nodo y lo haga con la imagen *registry*.

Utilizamos el puerto 5050 para exporte el servicio a través de la dirección IP del nodo porque el 5000 está siendo utilizado por el servicio *encuesta_vote*.

4.11.2 Desplegar un servicio de Portainer

Portainer es una simple interfaz gráfica para manejar Docker. En principio no tiene mucha integración con Docker Swarm, pero permite conocer multitud de características a base de clicks.

Lanzamos el servicio de Portainer con la ejecución de:

```
docker service create \
--name portainer \
--constraint 'node.role == manager' \
--publish 9000:9000 \
--mount type=bind,src=//var/run/docker.sock,dst=/var/run/docker.sock \
portainer/portainer \
-H unix:///var/run/docker.sock
```

Creará un servicio llamado *portainer*, se deba ejecutar en un nodo que cumpla las funciones de manager, puentea el puerto 9000 del servicio al 9000 del nodo, monte el socket de Docker para que puedan ejecutarse comando dentro de él y lo hace con la imagen portainer/portainer.

El acceso de este servicio estará disponible a través del navegador web con la dirección URL: http://192.168.99.100:5000

4.12 CAMBIAR ROLES DE NODOS Y ABANDONAR UN SWARM

Los nodos pueden tener dos roles distintos: *manager* y *worker*. Actualmente existen dos nodos como workers (*nodo-02* y *nodo-03*) y uno como manager (*manager-01*).

Para gestionar estos roles debemos acceder a un nodo manager: `eval $(docker-machine env manager-01)`

Promovemos el nodo-02 a manager a través del comando `docker node promote nodo-02` y con `docker node ls` comprobamos que la columna *Manager* tendrá el valor *Reachable*.

Ahora, conectamos al *nodo-02* con `eval $(docker-machine env nodo-02)` El nodo-02 podrá actuar sobre cualquier otro manager, en este caso *nodo-02* degradará a worker el nodo *manager-01* con `docker node demote manager-01`

Con la ejecución de `docker node ls` vemos que el valor de la columna *Manager* tendrá como *Leader* al *nodo-02*, estando vacía para los dos nodos restantes (*manager-01* y *nodo-03)*.

Por último, accedemos al nodo manager-01 para que abandone el clúster de Docker Swarm. Primero ejecutamos eval $(docker-machine env manager-01) y luego docker swarm leave

Para listar los nodos del clúster debemos conectar al nodo que cumple las funciones de manager actualmente: `eval $(docker-machine env nodo-02)` y allí ejecutar `docker node ls`

El estado de *manager-01* pasará a *Down*.

4.13 ELIMINAR CLÚSTER

Si deseamos eliminar el escenario recreado basta con eliminar las máquinas virtuales de VirtualbBox con `docker-machine rm manager-01 && docker-machine rm nodo-02 && docker-machine rm nodo-03`

El comando `docker-machine rm $(docker-machine ls -q)` eliminará todos los Docker Machine creados, incluso aunque no pertenezcan al clúster que hemos tratado en este escenario.

Si queremos eliminar el stack creado: `docker stack rm encuesta`

Si queremos eliminar todos los servicios creados: `docker service rm $(docker service ls -q)`

5. DESPLEGAR WORDPRESS EN UN CLÚSTER DE DOCKER SWARM

Con la ayuda de un script, ejecutamos el despliegue para la instalación del CMS WordPress. Este script despliega dos servicios: uno con el sistema de gestión de base de datos MariaDB y otro con WordPress 4.8.

Este ejemplo no sigue la arquitectura de microservicios, ya que la misma imagen de WordPress trae consigo PHP 7.0 y Apache 2.4, además de WordPress 4.8.

Comenzamos generando una password para la BBDD. Esto se realiza con `openssl rand -base64 20` y será almacenada en un secreto de Docker.

Para ello debemos ejecutar el comando `openssl rand -base64 20 | docker secret create root_db_pass -` que generará un secreto con nombre *root_db_pass* utilizado para la contraseña root de MariaDB.

Los secretos de Docker se almacenan de forma cifrada en el Swarm. No son consultables desde ningún nodo y se montan en el interior de los contenedores (tareas) que hagan uso de él. De esta manera, si accedemos a un contenedor que esté ejecutando una tarea con secreto, podremos consultarlo con `cat /var/run/secrets/nombresecreto`

Igualmente, creamos otro secreto con el nombre *wp_db_pass* para la contraseña de la base de datos de WordPress: `openssl rand -base64 20 | docker secret create wp_db_pass -`

Podemos consultar los secretos con la ejecución de `docker secret ls` pero no su contenido.

Creamos un volumen para almacenar el contenido del servicio *mariadb* que crearemos posteriormente. De la misma manera creamos otro para el contenido del servicio *wordpress*.

Los creamos con `docker volume create dbcontent` y `docker volume create wwwcontent`

A continuación, creamos el servicio *mariadb* con el siguiente comando:

```
docker service create \
--name mariadb \
--replicas 1 \
--constraint=node.role==manager \
--network wp \
--mount type=volume,source=dbcontent,destination=/var/lib/mysql
--secret source=root_db_pass,target=root_db_pass \
--secret source=wp_db_pass,target=wp_db_pass \
-e MYSQL_ROOT_PASSWORD_FILE=/run/secrets/root_db_pass \
-e MYSQL_PASSWORD_FILE=/run/secrets/wp_db_pass \
-e MYSQL_USER=wp \
-e MYSQL_DATABASE=wp \
mariadb:10.3.0
```

Esto creará un servicio con el nombre *mariadb*, una única réplica, en un nodo con rol manager, asociado a la red *wp*, con un volumen montado que guardará el contenido de */var/lib/mysql*, con un secreto llamado *root_db_pass* y otro llamado *wp_db_pass*. Tomará las variables de entorno *MYSQL_ROOT_PASSWORD_FILE* y *MYSQL_PASSWORD_FILE* desde los secretos de Docker. Las variables *MYSQL_USER* y *MYSQL_DATABASE* también se pasan en la ejecución de este servicio. Como base, tomará la imagen de MariaDB taggeada como 10.3.0.

Ahora, ejecutamos la creación del servicio *wordpress* que contiene además PHP 7.0 y Apache 2.4:

```
docker service create \
--name wordpress \
--replicas 1 \
--constraint=node.role==worker \
--network wp \
--publish 80:80 \
--mount type=volume,source=wwwcontent,destination=/var/www/html \
--secret source=wp_db_pass,target=wp_db_pass,mode=0400 \
-e WORDPRESS_DB_USER=wp \
-e WORDPRESS_DB_PASSWORD_FILE=/run/secrets/wp_db_pass \
-e WORDPRESS_DB_HOST=mariadb \
-e WORDPRESS_DB_NAME=wp \
wordpress:4.8.0-php7.0-apache
```

Esta ejecución creará un servicio con nombre *wordpress*, una única réplica, en un nodo que tenga el rol worker, conectado a la red *wp*, puenteando el puerto 80 del servicio al 80 del nodo, con un volumen montado que guardará el contenido de la ruta */var/www/html*. Además cargará un secreto llamado *wp_db_pass* y pasará a través de variables *WORDPRESS_DB_USER*, *WORDPRESS_DB_PASSWORD_FILE* (que lo tomará del secreto montado), *WORDPRESS_DB_HOST* y *WORDPRESS_DB_NAME*. Esto lo hará tomando una imagen de WordPress taggeada como *4.8.0-php7.0-apache*

Hay que tener en cuenta que esta aplicación no estaría correctamente desplegada para un entorno de alta disponibilidad o elástico. La modificación simultánea de la base de datos o los archivos de WordPress podrían suponer problemas de corrupción de datos.

Se trata de un ejemplo sencillo que utiliza algunas de las funciones de Docker y Docker Swarm.

6. ALGO MÁS SOBRE PORTAINER

Portainer es una UI (User Interface) para mejorar y facilitar gestión de Docker o Docker Swarm a través de una interfaz gráfica accesible vía navegador web.

Está disponible para Linux, Windows y OSX.

En esta tabla podemos ver algunas de sus características, a la vez que se compara con algunos de sus competidores más similares como son Shipyard o Panamax.

	Portainer	Shipyard	Panamax
Gestionar contenedores de Docker	✔	✔	✔
Acceder a la consola del contenedor	✔	✔	
Gestionar imágenes de Docker	✔	✔	✔
Etiquetar y subir imágenes Docker	✔	✔	
Gestionar redes de Docker	✔		
Gestionar volúmenes de Docker	✔		
Navegar por los eventos de Docker	✔	✔	
Preconfigurar templates de contenedores	✔		
Vista de clúster con Swarm	✔	✔	

Para ejecutar Portainer en una instalación de Docker podemos hacerlo ejecutando un contenedor: `docker run -d -p 9000:9000 portainer/portainer`

Sin embargo, si queremos hacerlo en un Docker Swarm, lo hacemos creando un servicio con el comando: `docker service create --name portainer --publish 9000:9000 --constraint 'node.role == manager' --mount type=bind,src=//var/run/docker.sock,dst=/var/run/docker.sock portainer/portainer -H unix:///var/run/docker.sock`

Existen otras herramientas más complejas y con más funcionalidades que Portainer, como es el caso de Rancher, una potente aplicación que permite incluso gestionar nodos en otras plataformas como AWS o DigitalOcean.

7. REFERENCIAS

- Docker Documentation
- ¿Qué es Docker?
- ¿Qué versión de Docker necesito?
- Clúster de Docker con Swarm Mode
- Formación Docker: Creando clusters de servidores con Docker Swarm
- Información sensible en los contenedores con Docker Secrets
- Montando un docker registry "Como Dios Manda ™"
- Desplegando un registro de Docker privado y seguro
- Docker Cloud Docs
- Docker Captains
- Docker, qué es y sus principales características
- Máquinas virtuales VS Contenedores
- Docker Documentation
- Docker Cheat Sheet by Docker
- Iniciar un stack de servicios en un cluster de Docker Swarm
- Cómo guardar secretos con Docker
- Creando servidores docker con Docker Machine
- Usar docker con Docker Machine en Linux, Windows o Mac
- 3 Node Swarm Cluster in 30 seconds (Docker 1.12)
- Deploy and Scale WordPress with Docker Cloud Swarm Mode
- Architecting a Highly Available and Scalable Wordpress Using Docker Swarm, Traefik & GlusterFS
- Docker Swarm: Setting Up a Scalable System

Imágenes de Docker Inc. y Amazon Web Services.

Realizado en Debian Jessie 8.8 con VirtualBox 5.1.22 y Docker 17.03
Dos Hermanas, Junio 2017