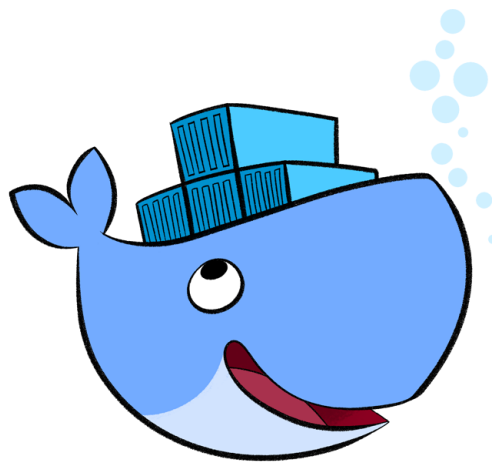


Ingeniería de Servidores (2015-2016)
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS
UNIVERSIDAD DE GRANADA

DOCKER

1 de mayo de 2016



Índice

1. Resumen	1
2. Introducción	1
3. El proyecto Docker	1
3.1. Historia	1
4. ¿Cómo funciona Docker?	2
4.1. Arquitectura	2
4.2. Componentes y conceptos básicos	2
4.3. Docker Engine	3
5. Ventajas y desventajas	4
6. Análisis de prestaciones	5
6.1. Docker vs Máquinas virtuales	5
6.2. Técnicas de evaluación	6
6.3. Presentación de resultados	7
6.3.1. Evaluación en Equipo 1	7
6.3.2. Evaluación en Equipo 2	7
6.4. Interpretación de resultados	7
7. Conclusiones	8
Anexos	10
A. Capturas de pantalla	10
B. Tutorial de uso	12

1. Resumen

La finalidad de este documento es explicar en qué consiste el proyecto Docker, su desarrollo a lo largo del tiempo y las modificaciones que han sido necesarias para llegar a ser el proyecto que actualmente podemos manejar. Algunos de los puntos que se tratarán son:

- Conocer la motivación del proyecto y su evolución a lo largo del tiempo.
- Analizar su estructura interna y elementos necesarios para su funcionamiento.
- Analizar las ventajas y desventajas del uso de Docker.
- Comparar las prestaciones que ofrece frente a otros recursos o alternativas.

2. Introducción

Hoy en día, los usuarios finales se han convertido en una fuente inagotable de expectativas. Quieren aplicaciones más flexibles, más rápidas, siempre disponibles, sin importar hora, lugar, dispositivo desde el que accedan al servicio o lo demandado que esté.

“La solución pasa por lograr la total independencia entre el componente software y la arquitectura subyacente, de modo que cuando las máquinas fallan, o se actualizan,... el servicio siempre esté ahí, independientemente de una máquina” [1]. Estas palabras, pronunciadas por Solomon Hykes en la conferencia FutureStack14, tratan de explicar la principal motivación y clave del éxito del proyecto Docker.

3. El proyecto Docker

Docker es un proyecto open-source que fue creado con la finalidad de “permitir empaquetar una aplicación junto con todas sus dependencias en una unidad estandarizada para el desarrollo software”. [2]

Solomon Hykes, fundador, CTO (“Chief Technical Officer”) y creador de la iniciativa explica la idea que hay detrás de este proyecto usando un ejemplo ilustrativo. En la industria tradicional se transporta la mercancía usando contenedores normalizados, independientemente del tipo de carga. De la misma forma, en la industria del software, el proyecto Docker pretende proporcionar estos contenedores, junto con las herramientas que serán necesarias para manejarlos. [3]

Así, los contenedores Docker incluyen todo lo que una aplicación software necesita para ejecutarse correctamente garantizando que siempre se ejecutará de la misma manera, independientemente del entorno en el que lo haga. [2]

3.1. Historia

Docker comenzó como un proyecto interno de DotCloud, una plataforma como servicio (PaaS) en la que los desarrolladores buscaban construir aplicaciones sin tener que preocuparse de las dependencias y la infraestructura interna. Fue liberado como código abierto en marzo de 2013 y unos meses después, empieza a colaborar con RedHat para solucionar incompatibilidades. [10] En octubre de este mismo año, DotCloud Inc. se convierte en Docker Inc., demostrando las altas expectativas puestas en el proyecto. [11]

En sus primeras versiones, Docker accedía a la virtualización del Kernel de Linux indirectamente, a través de `libvirt` o Linux Containers (LXC), hasta que desarrolló su propia librería,

`libcontainer`, que permite acceder a la virtualización directamente. Así, a partir de Docker 0.9, LXC se convierte en una herramienta opcional, estableciéndose `libcontainer` como librería pre-determinada. Este paso dio un gran impulso a la estabilidad de Docker, y en 2014 se lanza esta versión, considerándose su primera versión estable. [12] Google, Red Hat y Parallels comienzan a colaborar con el proyecto para mantener el núcleo, junto a los ingenieros de Canonical. [13]

En abril de 2015 el proyecto ya contaba con más de 20.700 estrellas de GitHub, situándose en el vigésimo puesto de los proyectos mejor valorados, con más de 4.700 bifurcaciones, y casi 900 colaboradores. [10]

En cuanto a la financiación económica se tiene a Red Hat como su mayor contribuidor, por encima incluso del mismo equipo de Docker, junto con otros colaboradores como IBM, Google, Cisco Systems y Amadeus IT Group. [14]

A día de hoy, en muchos blogs y páginas de noticias sobre software libre se hace referencia a la “Revolución Docker”: en sólo 3 años, cuenta ya con más de 2 billones de descargas de contenedores, más de 300,000 aplicaciones “dockerizadas” disponibles en el registro público y más de 21.000 estrellas de GitHub, entre otros datos. [15]

4. ¿Cómo funciona Docker?

El proyecto Docker nace con la finalidad de separar la ejecución de las aplicaciones de la arquitectura subyacente, facilitando la ejecución de manera aislada y segura de prácticamente cualquier aplicación. Se estudian a continuación algunos conceptos necesarios para comprender la solución que proponen sus desarrolladores.

4.1. Arquitectura

Docker hace uso de la arquitectura cliente-servidor, con la particularidad de que utiliza un mismo ejecutable para los dos componentes. Se pueden distinguir tres elementos en este modelo: [4] [5]

- **Servidor/demonio:** se ejecuta en un host, y hace el trabajo de construir, ejecutar y distribuir los contenedores. El demonio se ejecuta en un segundo plano, y espera a recibir instrucciones del cliente.
- **Cliente:** es el encargado de dirigir toda la comunicación. Los usuarios se comunican directamente con el cliente a través de comandos, y éste dirige al servidor.
- **Registro:** almacena las imágenes e información relacionada con ellas.

La arquitectura de Docker puede verse en la figura 4.1.

4.2. Componentes y conceptos básicos

Para comprender cómo funciona Docker es imprescindible entender los siguientes conceptos, a los que se hará referencia con frecuencia a lo largo del documento: [2]

- **Imágenes:** se utilizan para la *construcción*. Una imagen es un fichero de lectura que se usa para crear contenedores. Generalmente se crea una imagen principal con elementos básicos (Java, Ubuntu,...) a la que se le va añadiendo todo aquello que sea necesario, por ejemplo, una aplicación instalada.

Cada imagen está formada por una serie de capas, lo que permite que su funcionamiento sea más rápido y sencillo ya que cuando se modifica una imagen, sólo es necesario añadir una nueva capa, en lugar de reconstruir la imagen al completo.

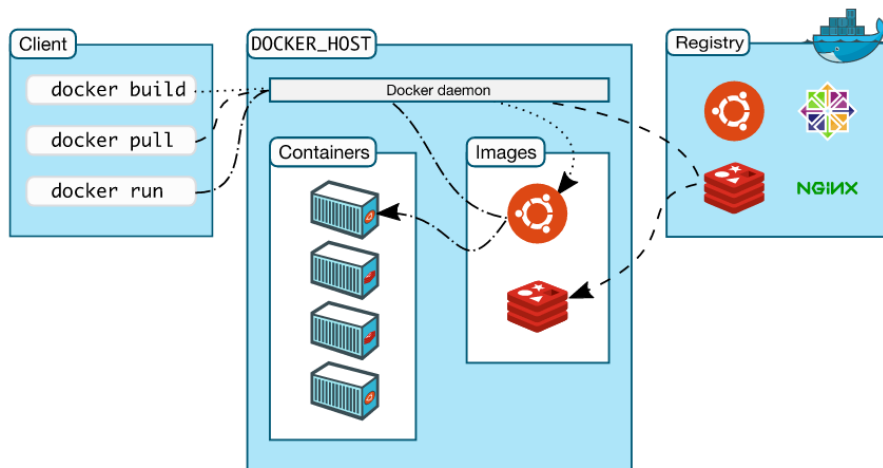


Figura 4.1: Modelo cliente/servidor de Docker [2]

- **Registros:** se utilizan para la *distribución*. Los registros pueden ser públicos o privados y contienen imágenes.
- **Contenedores:** se utilizan para la *ejecución*. Son instancias de una imagen y almacenan todo lo que necesita para que una aplicación se ejecute. Son plataformas aisladas unas de otras.

La herramienta se sostiene gracias a dos componentes claramente diferenciados: el motor (**Docker Engine**) y el registro público (**Docker Hub**).

Cada usuario tiene un registro privado, en el que puede guardar, crear y modificar sus imágenes a nivel local. Existe la opción de subir las imágenes que se han creado al registro público o **Docker Hub**, en el que se almacenan y distribuyen todas las imágenes que quieran ser compartidas. Estas imágenes públicas pueden ser buscadas y descargadas por cualquier usuario de Docker para modificarlas, usarlas como imagen principal o construir contenedores, mientras que las imágenes privadas no aparecen en los resultados de búsqueda.

Las imágenes de Docker se construyen a partir de las imágenes principales que pueden encontrarse en el registro público, ejecutando una serie de instrucciones. Cada instrucción añade una nueva capa a la imagen principal, dando como resultado la imagen final deseada por el usuario. Todas estas instrucciones se almacenan en un script llamado **Dockerfile**. Los pasos necesarios para manejar comandos, construir imágenes o crear un **Dockerfile**, entre otros, pueden consultarse con más detalle en el anexo B, donde se puede encontrar un tutorial de uso. [4]

4.3. Docker Engine

El motor es el núcleo de la plataforma. Cuando se habla de “Docker”, la mayoría de las veces se está haciendo referencia a este **Docker Engine**. Se ejecuta en Linux para crear todo el entorno necesario para las aplicaciones creadas (ver figura 4.2). El software está disponible en GitHub, considerándose uno de los proyectos que crece con más rapidez.[17]

Como el núcleo se ejecuta en Linux, Docker no puede utilizarse de forma nativa en otros sistemas operativos como Mac o Windows: es necesario ejecutar el motor de Docker sobre una máquina

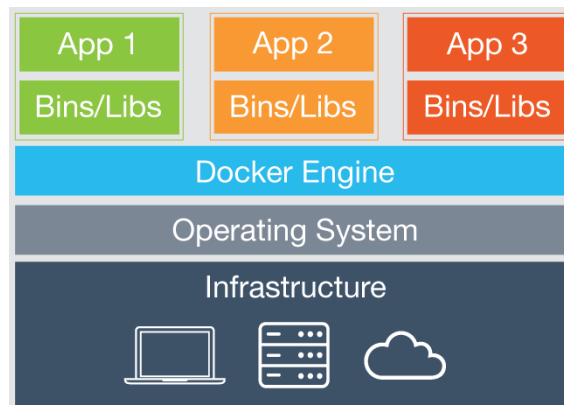


Figura 4.2: Motor de Docker

virtual. Docker Toolbox es un paquete que incluye todo lo necesario para instalar Docker en Mac o Windows, así como facilitar y optimizar su uso. Algunos de las herramientas que incluyen son: [16]

- **Docker Compose:** es una herramienta para la ejecución de aplicaciones que hagan uso de varios contenedores.
- **Docker Machine:** es una herramienta para la administración de los host, locales o remotos, en los que Docker esté instalado. **Docker Machine** crea una máquina virtual, instala el motor en ella y facilita la instalación del cliente y el servidor. Utilizar **Docker Machine** es, por ahora, la única manera de instalar Docker en Mac y Windows. El software que se utiliza para crear esta máquina virtual, y que se descarga de manera automática junto con el paquete, es VirtualBox. [19]
- **Kitematic:** es un proyecto de código abierto construido para simplificar la instalación y uso de Docker. Provee una interfaz gráfica de usuario para manejar contenedores de manera más intuitiva, en lugar de hacerlo desde la terminal. Permite también automatizar características más avanzadas dentro de un contenedor (gestión de puertos, configuración de volúmenes, cambiar variables de entorno,...). [20]

5. Ventajas y desventajas

Las ventajas de usar esta tecnología se desprenden fácilmente de toda la información recogida hasta el momento: [6] [8]

- Es un proyecto de código abierto.
- Facilita el desarrollo de aplicaciones, ya que se eliminan los problemas que conllevan las dependencias software.
- Los contenedores son ligeros, fáciles de crear y borrar, lo que hace que sea un buen entorno de pruebas.
- Existe una gran cantidad imágenes en el registro público de Docker que pueden descargarse y modificarse libremente.
- Pueden ejecutarse varios contenedores en un mismo host.

- Un mismo contenedor funciona en cualquier máquina Linux.
- Consume pocos recursos hardware.

Sin embargo, aunque muchos desarrolladores presenten Docker como la alternativa definitiva de abstracción, pueden encontrarse algunas desventajas e ideas erróneas que se han creado a partir del éxito y rápido crecimiento que ha tenido este proyecto [7] [9]:

- **Docker no es una máquina virtual.** Hay muchos usuarios y desarrolladores que confunden Docker con una máquina virtual un poco más ligera. Sin embargo, pueden encontrarse diferencias importantes entre ambos. Se verá este tema con más detalle en la subsección 6.1.
- Al ser relativamente nuevo, es posible encontrar errores en algunas versiones.
- **Docker no es independiente del sistema operativo de la máquina anfitriona.** Una de las creencias más comunes es que Docker permite ejecutar cualquier aplicación en cualquier sistema operativo. Como se ha visto en la subsección 4.3, aunque **Docker Machine** facilite la instalación en Mac y Windows, el motor de Docker presenta dependencia del kernel de Linux.
- Para Linux sólo soporta arquitecturas de 64 bits.
- **Docker aumenta la seguridad al ejecutar las aplicaciones en contenedores.** Docker es una tecnología de empaquetado de contenedores, pero no hay que olvidar que se trata de contenedores software: se ejecuta sobre el host, por lo que éste es igualmente vulnerable.
- **Docker no tiene overhead.** El overhead es menor al no contar con el sistema operativo invitado, pero esto no significa que no exista. Es menor que en el de las máquinas virtuales, pero no despreciable. Este aspecto servirá de base para la realización del experimento en la subsección 6.2.

6. Análisis de prestaciones

Intuitivamente, el rendimiento de una aplicación no puede ser el mismo ejecutándola en un contenedor de Docker que haciéndolo en un host en el que pueda ejecutarse sin problemas de dependencias. Aún así, Docker se presenta como alternativa preferible a otras tecnologías tales como las máquinas virtuales, Vagrant, Puppet,...

¿Hasta qué punto es preferible utilizar Docker frente a otras alternativas?

6.1. Docker vs Máquinas virtuales

A partir de toda la información recogida hasta ahora, puede deducirse que los conceptos de Docker y de máquina virtual no son tan distantes como pudiera parecer en un principio, y por eso es frecuente que Docker se confunda con una máquina virtual ligera. Recordemos que una máquina virtual es un software que puede simular hardware suficiente en un host como para permitir ejecutar una o varias instancias de un sistema operativo funcionen de manera aislada. [21]

La tecnología de contenedores y las máquinas virtuales presentan algunas diferencias importantes. [2] [9]

- Docker utiliza los binarios y las librerías directamente para ejecutar la aplicación. Por otra parte, las máquinas virtuales necesitan tener, además de la aplicación, los binarios y librerías necesarias, un sistema operativo invitado, consumiendo así recursos hardware (memoria RAM, procesador, etc) y minimizando aquellos que podría utilizar la aplicación.

- El aislamiento de recursos en Docker es mucho menor, ya que por el diseño de la plataforma existen algunos recursos compartidos. De esta forma si un contenedor consume todos estos recursos compartidos, otros procesos que se estén ejecutando en el host tendrán que esperar a la liberación de éstos.
- La estructura en capas de las imágenes permite actualizar una imagen rápidamente simplemente añadiendo una capa más, algo imposible en una máquina virtual, en la que se tendría que reconstruir por completo.

Para mayor claridad, en el anexo A, la imagen A ofrece una comparación entre máquinas virtuales y Docker.

6.2. Técnicas de evaluación

Para la evaluación del rendimiento se utilizará un script muy simple, escrito en bash, que utiliza un contador para calcular los cuadrados de los números desde el cero hasta un límite establecido. Se instalará Docker en dos ordenadores distintos: uno de ellos tendrá instalado un sistema operativo basado en Linux, y en el otro será necesario la instalación de la máquina virtual **Docker Machine**.

En cada uno de estos dos equipos habrá a su vez una máquina virtual, que consuma una cantidad considerable de recursos en función de las características de cada uno de ellos. El experimento consistirá en ejecutar este script en la terminal de Docker, en la máquina virtual y en la máquina anfitriona.

```
#!/bin/bash
#Script simple

contador=0
maximo=100

time while [ $contador -lt $maximo ]
do
    let contador+=1
    resultado=$contador
    let resultado*=$resultado
    echo $resultado
done
```

El hardware y el software utilizado para la evaluación serán los siguientes:

- **Equipo 1.** *Procesador Intel Core i3. Frecuencia 1,8 GHz. Memoria 4 GB. Sistema operativo Ubuntu 14.04*
- **Equipo 2.** *Procesador Intel Core i5. Frecuencia 1,3 GHz. Memoria 8 GB. Sistema operativo OS X Yosemite*
- **Máquina virtual 1.** se utilizará como software de virtualización VirtualBox y como anfitrión el **equipo 1**. Tendrá 2 GB de memoria RAM y sistema operativo Ubuntu 14.04.
- **Máquina virtual 2.** se utilizará como software de virtualización VirtualBox y como anfitrión el **equipo 2**. Tendrá 4 GB de memoria RAM y sistema operativo Ubuntu 14.04.

Se han de tener en cuenta los siguientes puntos para el estudio:

- Se utilizará como medida para la evaluación el tiempo de ejecución medido en segundos.
- Para ejecutar el script en Docker se ha descargado y ejecutado una imagen base con Ubuntu 14.04.

- Para que las medidas sean lo más objetivas posible, no se ha ejecutado ningún otro proceso durante la ejecución del script.

6.3. Presentación de resultados

Una vez ejecutado el script bajo las condiciones anteriores, se han obtenido los siguientes resultados.

6.3.1. Evaluación en Equipo 1

Ver tabla 6.1 y figura 6.1.

Máximo	Equipo 1	Docker	Máquina virtual 1
100	0,019	0,0244	0,0169
1000	0,139	0,1482	0,2093
10000	1,1362	1,3036	2,1676
100000	10,8797	12,7948	21,2975
1000000	112,3853	167,9897	207,1008

Tabla 6.1: Comparativa en Ubuntu

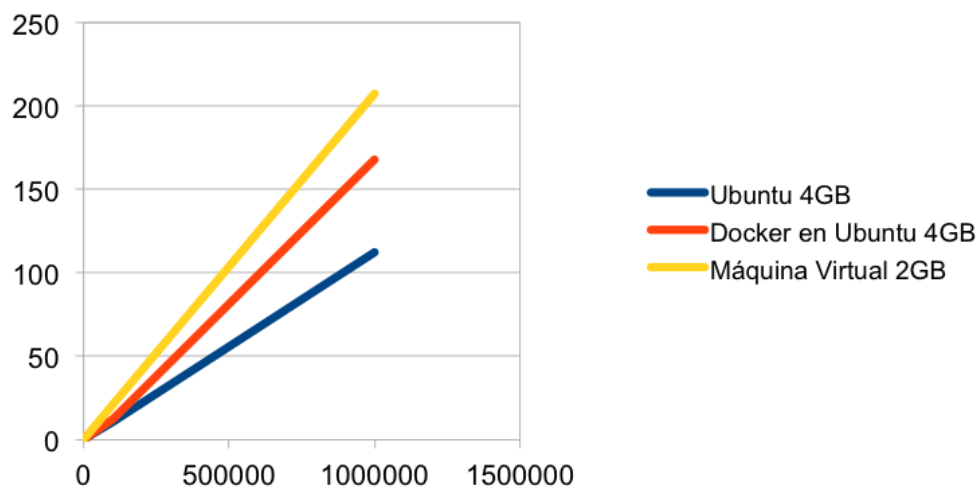


Figura 6.1: Gráfico comparativo: Ubuntu

6.3.2. Evaluación en Equipo 2

Ver tabla 6.2 y figura 6.2.

6.4. Interpretación de resultados

Las conclusiones que pueden extraerse de este experimento son las siguientes:

- En Linux, Docker es 51.89 % más lento que el host, pero 45.67 % más rápido que la máquina virtual.

Máximo	Equipo 2	Docker	Máquina virtual 2
100	0,0162	0,0372	0,0197
1000	0,1531	0,3634	0,1778
10000	1,5818	3,6926	1,9252
100000	15,9377	36,8415	18,7984
1000000	160,4102	333,4249	181,1432

Tabla 6.2: Comparativa en OS X

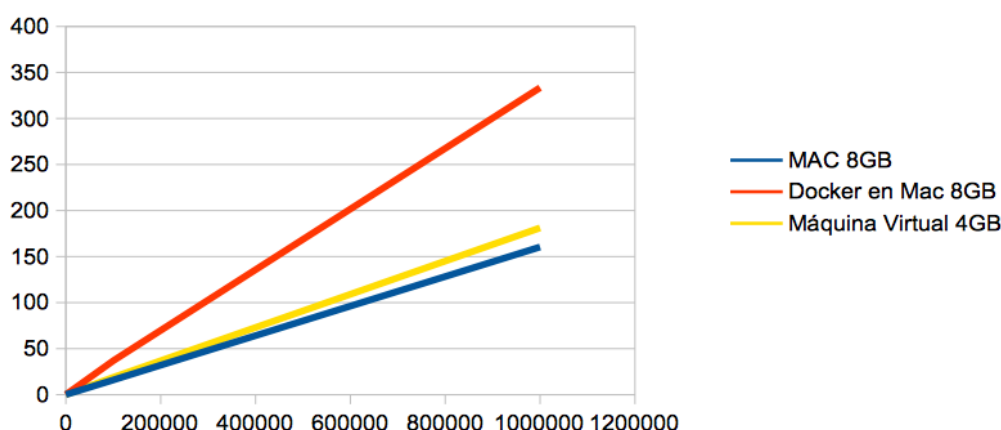


Figura 6.2: Gráfico comparativo: OS X

- En otros sistemas operativos, Docker es 33.101 % más lento que la máquina virtual y además 23.289 % más lento que el host.

7. Conclusiones

Como conclusión, puede extraerse que en Linux, donde Docker puede instalarse de forma nativa, resulta ventajoso frente a otras herramientas más conocidas como son las máquinas virtuales. Además, la diferencia en el tiempo de ejecución con respecto al host puede estar justificada por la portabilidad que proporciona la tecnología de los contenedores.

Sin embargo, para el resto de sistemas operativos, aunque existen algunas facilidades, como la interfaz gráfica, Docker está en período de pruebas. Es necesario el apoyo de otras herramientas que hacen que los tiempos de ejecución sean mucho más altos, tanto que puede no estar compensada por su portabilidad, dependiendo del uso que se le quiera dar a la herramienta.

Para finalizar este pequeño análisis, puede decirse que Docker ha dado un gran salto con su “Revolución Docker”, pero sus desarrolladores están todavía lejos de conseguir esa utopía buscada que es la completa independencia entre el software y la arquitectura del computador. Pero, del mismo modo que no hay que dejarse llevar por el éxito y la gran cantidad de críticas positivas que está recibiendo, no hay que quitar mérito a todo lo que ha conseguido este proyecto: Docker ha convertido la tecnología de los contenedores, ya existente pero difícil de manejar, en un recurso mucho más organizado, rápido y sencillo de utilizar y comprender; todo esto siendo una herramienta que lanzó su primera versión hace tan sólo tres años. Distinguir si Docker se ajusta o no a las necesidades de

un usuario concreto es cuestión de mantener una perspectiva objetiva acerca de las posibilidades de esta herramienta.

Anexos

A. Capturas de pantalla

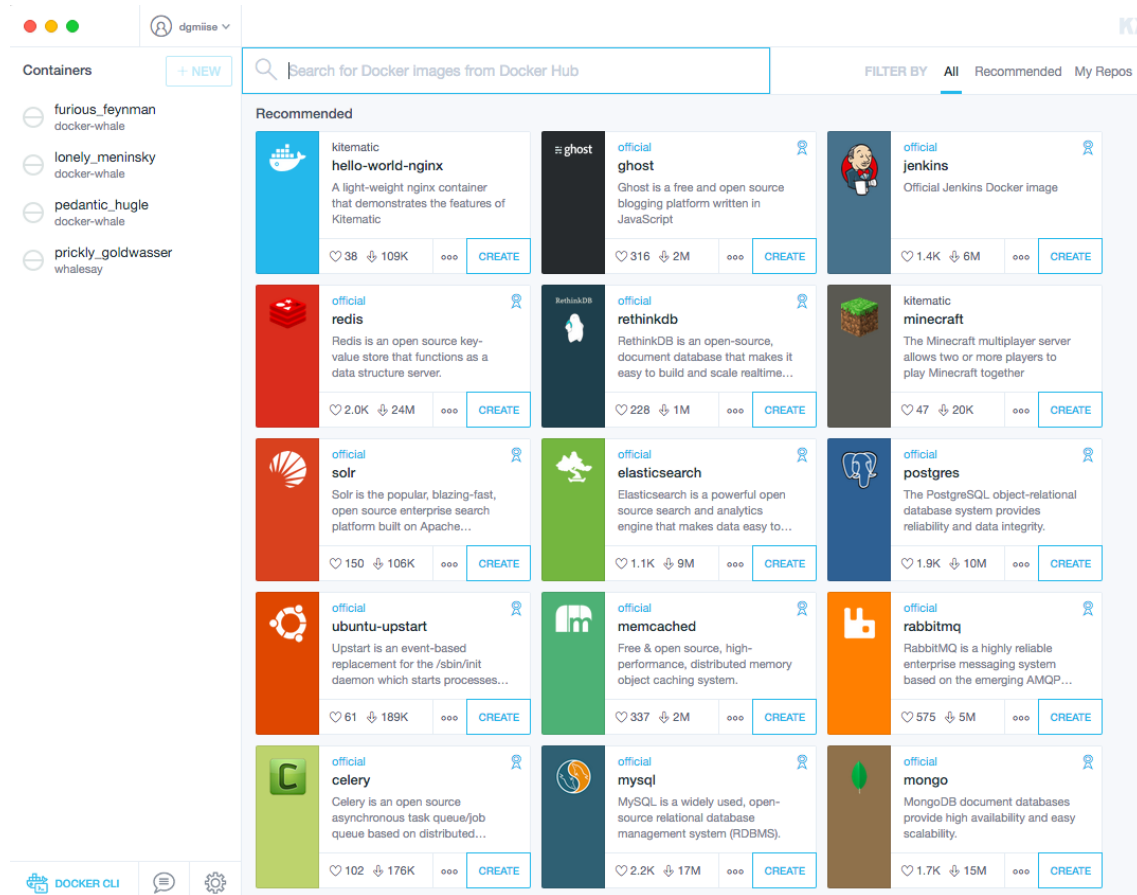


Figura A.1: Interfaz Kitematic

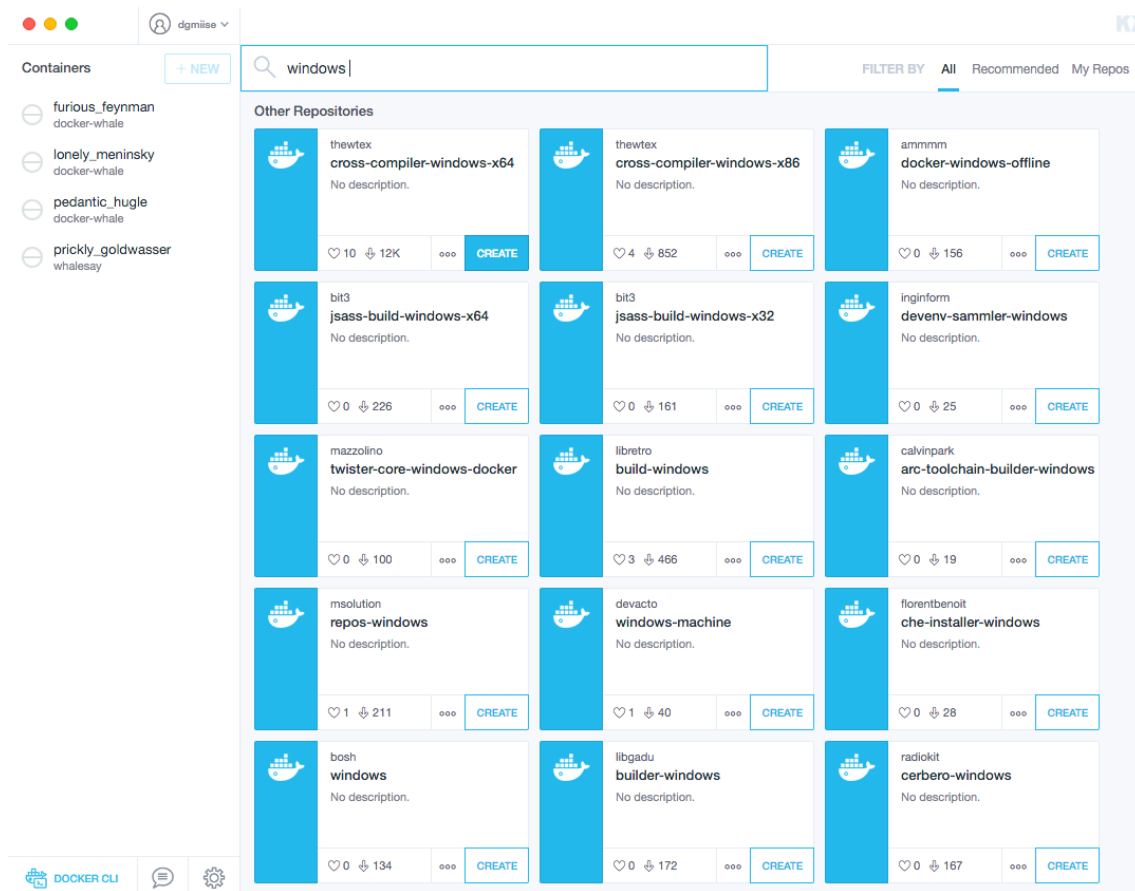


Figura A.2: Buscar y descargar contenedores

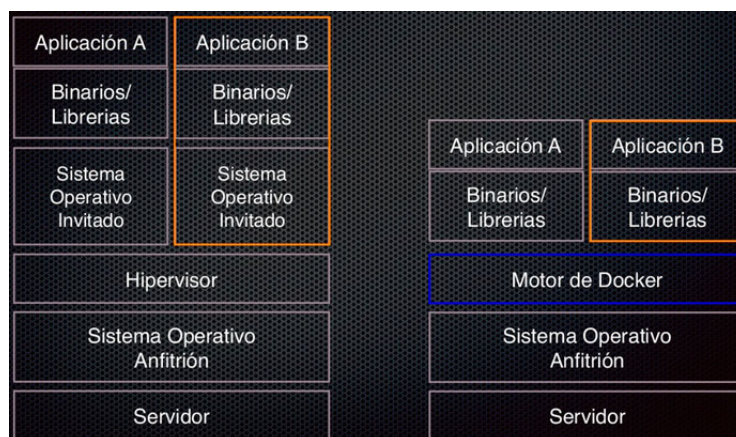


Figura A.3: Máquinas virtuales vs Docker[6]

B. Tutorial de uso

En este anexo se expondrá el modo de uso de Docker de una manera bastante básica.¹

En primer lugar se tratará un ejemplo sencillo, lanzar un contenedor ya creado, el Hello World. Para ello, bastará con introducir el comando: `docker run ubuntu /bin/echo 'Hello world' Hello world`, donde `docker run` ejecuta un contenedor, `ubuntu` es la imagen que se ejecuta y `/bin/echo` es el comando que se ejecuta dentro del contenedor. Cuando se especifica una imagen, en este caso `ubuntu`, Docker la busca primero a nivel local y, si no existe, la busca en las imágenes públicas del Docker Hub. La salida por pantalla puede verse en la figura B.1, donde se ve que no encuentra la imagen a nivel local y pasa a descargarla.

```
sagrario — bash --login — 80x28
```

```
##  
## ## ##  
## ## ## ## ##  
.....  
{ ~~~~~ / ===  
| o |  
|   |  
|___|
```

`docker` is configured to use the `default` machine with IP `192.168.99.100`
For help getting started, check out the docs at <https://docs.docker.com>

```
[cvihs215064:~ sagrario$ docker run ubuntu /bin/echo 'Hello world'  
Unable to find image 'ubuntu:latest' locally  
latest: Pulling from library/ubuntu  
  
759d6771041e: Downloading 49.72 MB/65.69 MB  
759d6771041e: Pull complete  
8836b825667b: Pull complete  
c2f5e51744e6: Pull complete  
a3ed95caeb02: Pull complete  
Digest: sha256:b4dbab2d8029edddfe494f42183de20b7e2e871a424ff16ffe7b15a31f102536  
Status: Downloaded newer image for ubuntu:latest  
Hello world  
cvihs215064:~ sagrario$
```

Figura B.1: docker run Hello world

El siguiente paso es crear un demonio, para lo que hay que ejecutar: `docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"`, donde la bandera `-d` hace que el contenedor se ejecute en segundo plano y, en este caso, el comando a ejecutar en el contenedor es: `/bin/sh -c "while true; do echo hello world; sleep 1; done"`.

En la figura B.2 se ve que la salida por pantalla de este comando es la ID del contenedor, una larga cadena de caracteres bastante difícil de manejar. A continuación se verá que Docker brinda una solución a este problema.

¹El usuario que aparecerá a lo largo de este tutorial en las capturas de pantalla es un usuario inventado. Cualquier parecido con la realidad es pura coincidencia.

Para ver información útil sobre el contenedor que se acaba de ejecutar, se usa el comando **docker ps**. Analizando la salida por pantalla, que se ve en la figura B.2, se tiene: [23]

- **CONTAINER ID**: la variante más corta de la ID del contenedor.
- **IMAGE**: nombre de la imagen que se ejecuta.
- **COMMAND**: comando que es está ejecutando en el contenedor.
- **STATUS**: indica el estado del contenedor, si está ejecutándose, pausado, abortado, etc. En este caso indica que está activo hace aproximadamente un minuto.
- **NAMES**: indice el nombre que Docker asigna automáticamente a los contenedores que se ejecutan, en este caso es **silly_cray**. Esta es la cómoda alternativa que Docker brinda a la tediosa ID del contenedor, permitiendo utilizar este nombre en su lugar.

Por otra parte, para ver lo que está ocurriendo dentro del contenedor se puede ejecutar **docker logs name-container**. En este caso, name-container será **silly_cray**. En la figura B.2 se ve salida por pantalla, donde, como cabe esperar, es una larga serie de “Hello World” impresas en pantalla.

```

cvihs215064:~ sagrario$ docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1;done"
96f4df68fe30ad2512a3ce352b2cac08f08d366e3006b750f1ae0906e5fda031
cvihs215064:~ sagrario$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
96f4df68fe30       ubuntu             "/bin/sh -c 'while tr"  17 minutes ago     Up About a minute   0.0.0.0:80->80/tcp   silly_cray
cvihs215064:~ sagrario$ docker logs silly_cray
hello world
hello world
hello world
hello world
hello world
hello world

```

Figura B.2: docker ps y docker logs

Para concluir este ejemplo, hay que parar el demonio que hemos creado ejecutando **docker stop silly_cray**. Para comprobar que ha funcionado, se vuelve a introducir el comando **docker ps** y, como se ve en la figura B.3, ya no aparece información sobre el contenedor. B.3.

```

cvihs215064:~ sagrario$ docker stop silly_cray
silly_cray
cvihs215064:~ sagrario$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
cvihs215064:~ sagrario$

```

Figura B.3: docker stop y docker ps

Por otro lado, para descargar un contenedor basta con introducir el comando **pull nombre**. En este ejemplo se va a descargar CentOS, como se ve en la figura B.4. Además, se puede acceder al interior del contenedor a través del comando **docker run -t -i ubuntu /bin/bash** Donde la bandera **-t** asigna una terminal dentro del contenedor, **-i** permite realizar una conexión interactiva por la entrada estándar (STDIN) del contenedor y **/bin/bash** lanza un intérprete de comandos en el contenedor.

En la figura B.4 puede verse un ejemplo en el que se navega por algunas carpetas del contenedor, y en B.5 se muestra que, efectivamente, el sistema operativo en el que se encuentra es CentOS.

Otro comando útil es el comando **docker search cadena**, que permite buscar, en Docker Hub las imágenes que contengan “cadena” en su nombre. En la figura B.6 puede verse un ejemplo en que se busca imágenes de windows.

```
sagrario — @771f1bc0bb0f:/ — docker run -t -i centos /bin/bash — 93x23
[cvihs215064:~ sagrario$ docker pull centos
Using default tag: latest
latest: Pulling from library/centos

a3ed95caeb02: Pull complete
5989106db7fb: Pull complete
Digest: sha256:1b9adf413b3ab95ce430c2039954bb0db0c8e2672c48182f2c5b3d30373d5b71
Status: Downloaded newer image for centos:latest
[cvihs215064:~ sagrario$ docker run -t -i centos /bin/bash
[root@5ba499fd85e6 /]# ls
anaconda-post.log  dev  home  lib64      media  opt  root  sbin  sys  usr
bin               etc  lib   lost+found mnt    proc run  srv   tmp  var
[root@5ba499fd85e6 /]# cat /etc/issue
\S
Kernel \r on an \m

[root@5ba499fd85e6 /]# cat /etc/redhat-release
.dockervenv      home/          opt/           sys/
.dockervinit     lib/           proc/          tmp/
anaconda-post.log lib64/         root/          usr/
bin/             lost+found/    run/           var/
dev/             media/         sbin/
etc/             mnt/          srv/
```

Figura B.4: pull CentOS

```
sagrario — @771f1bc0bb0f:/ — bash --login — 93x23
[cvihs215064:~ sagrario$ docker run -t -i centos /bin/bash
[root@771f1bc0bb0f /]# cat /etc/redhat-release
CentOS Linux release 7.2.1511 (Core)
[root@771f1bc0bb0f /]# exit
exit
cvihs215064:~ sagrario$
```

Figura B.5: docker en CentOS

Por último, se verá como crear una imagen propia en un contenedor Docker. En primer lugar habrá que analizar el funcionamiento de la imagen ya creada, para ello se ejecuta el comando **docker run Docker/whalesay cowsay boo**, como se ve en la figura B.7. En este ejemplo se modificará esta imagen para que la ballena hable por sí sola diciendo frases “aleatorias”.

Para crear una imagen propia hay que crear un Dockerfile, que debe tener la siguiente estructura: [24]

- FROM <image>o FROM <image>:<tag>o FROM <image>@<digest>. Debe ser la primera instrucción del Dockerfile. Establece la imagen base para las siguientes instrucciones. Pueden haber múltiples FROM en un mismo Dockerfile, la finalidad sería crear múltiples imágenes. Los valores de <tag>y <digest>son opcionales, si se omiten se asume que se toma la ultima versión de la imagen.
- RUN <comando>o RUN [ejecutable, parametro1, parametro2, ...] Las instrucciones que aparezcan aquí se ejecutarán en una nueva capa sobre la imagen actual. La imagen resultante de estas operaciones será la que se ejecute en el siguiente paso del Dockerfile.


```

[cvihs215064:~ sagrario$ docker search windows
NAME                                DESCRIPTION
STARS      OFFICIAL    AUTOMATED
suchja/wine                                Running Windows applications in a Docker c...
33                                     [OK]
thewtex/cross-compiler-windows-x64        64-bit Windows cross-compiler based on MXE...
10                                     [OK]
thewtex/cross-compiler-windows-x86        32-bit Windows cross-compiler based on MXE...
4                                     [OK]
libretro/build-windows                    Build image for libretro windows
3
msolution/repos-windows
1
nsgb/konstrukt-build-image-windows
1
vimagick/samba                            The standard Windows interoperability suit...
1                                     [OK]
florentbenoit/che-installer-windows        Allow to get installer dependencies and co...
0                                     [OK]
bosh/windows
0
radiokit/cerbera-windows
0
libgadu/builder-windows
0
gplugin/builder-windows
0

```

Figura B.6: docker search windows

- `CMD [ejecutable, parametro1, parametro2, ...]` o `CMD [parametro1, parametro2, ...]` o `CMD comando parametro1, parametro2, ...` Solo puede haber una instrucción `CMD` por `Dockerfile`, en el caso de que se hubieran más solo se tendría en cuenta el último. Se utiliza para proporcionar comandos por defecto para ser ejecutados en un contenedor.

Puede verse un ejemplo concreto en la figura B.8, donde: [25]

- `FROM Docker/whalesay:latest`: Indica a Docker que la imagen que crearemos está basada en la imagen Whalesay existente. La etiqueta `latest` indica que se debe tomar la última versión existente de esta imagen, podríamos haber prescindido de ella y lograr el mismo efecto.
- `RUN apt-get -y update && apt-get install -y fortunes`: Se añade el programa “fortunes” a la imagen existente.
- `CMD /usr/games/fortune -a | cowsay`: Se indica el software a ejecutar cuando la imagen sea cargada. De esta forma se indica al programa “fortunes” que envíe su salida al programa `cowsay`.

Ahora, para construir esta nueva imagen se escribirá el comando: `docker build -f ruta -t nombre_imagen ..` Donde: `docker build` construye un contenedor de un archivo Docker, a partir del `Dockerfile`, la bandera `-t` especifica el repositorio o directorio donde se desea guardar la nueva imagen y la bandera `-f` especifica la ruta donde se encuentra el `Dockerfile`. [24]. La salida por pantalla se ve en las figuras B.9 y B.10.

Si ahora se ejecuta el comando `docker images` se ve que, efectivamente, la imagen se ha creado. Ver figura B.11.

Por último ejecutamos la imagen que se ha creado, como se ve en la figura B.12.


```

[cvi070140:ejemplo sagrario$ docker build -t docker-whale .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM Docker/whalesay:latest
invalid reference format: repository name must be lowercase
[cvi070140:ejemplo sagrario$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker/whalesay     latest             6b362a9f73eb       10 months ago      247 MB
[cvi070140:ejemplo sagrario$ open -e Dockerfile
[cvi070140:ejemplo sagrario$ docker build -t docker-whale .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM docker/whalesay:latest
----> 6b362a9f73eb
Step 2 : RUN apt-get -y update && apt-get install -y fortunes
----> Running in 537a281e5cfe
Ign http://archive.ubuntu.com trusty InRelease
Get:1 http://archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:2 http://archive.ubuntu.com trusty-security InRelease [65.9 kB]
Hit http://archive.ubuntu.com trusty Release.gpg
Hit http://archive.ubuntu.com trusty Release
Get:3 http://archive.ubuntu.com trusty-updates/main Sources [344 kB]
Get:4 http://archive.ubuntu.com trusty-updates/restricted Sources [5217 B]
Get:5 http://archive.ubuntu.com trusty-updates/universe Sources [192 kB]
Get:6 http://archive.ubuntu.com trusty-updates/main amd64 Packages [948 kB]
Get:7 http://archive.ubuntu.com trusty-updates/restricted amd64 Packages [23.5 kB]
Get:8 http://archive.ubuntu.com trusty-updates/universe amd64 Packages [462 kB]
Get:9 http://archive.ubuntu.com trusty-security/main Sources [141 kB]
Get:10 http://archive.ubuntu.com trusty-security/restricted Sources [3920 B]
Get:11 http://archive.ubuntu.com trusty-security/universe Sources [40.9 kB]
Get:12 http://archive.ubuntu.com trusty-security/main amd64 Packages [570 kB]
Get:13 http://archive.ubuntu.com trusty-security/restricted amd64 Packages [20.2 kB]
Get:14 http://archive.ubuntu.com trusty-security/universe amd64 Packages [165 kB]
Hit http://archive.ubuntu.com trusty/main Sources
Hit http://archive.ubuntu.com trusty/restricted Sources
Hit http://archive.ubuntu.com trusty/universe Sources
Hit http://archive.ubuntu.com trusty/main amd64 Packages
Hit http://archive.ubuntu.com trusty/restricted amd64 Packages
Hit http://archive.ubuntu.com trusty/universe amd64 Packages
Fetched 3047 kB in 22s (133 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following extra packages will be installed:
  fortune-mod fortunes-min librecode0
Suggested packages:
  x11-utils bsdmainutils
The following NEW packages will be installed:
  fortune-mod fortunes fortunes-min librecode0
0 upgraded, 4 newly installed, 0 to remove and 80 not upgraded.
Need to get 1961 kB of archives.
After this operation, 4817 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main librecode0 amd64 3.6-21 [771 kB]
Get:2 http://archive.ubuntu.com/ubuntu/ trusty/universe fortune-mod amd64 1:1.99.1-7 [39.5 kB]
Get:3 http://archive.ubuntu.com/ubuntu/ trusty/universe fortunes-min all 1:1.99.1-7 [61.8 kB]
Get:4 http://archive.ubuntu.com/ubuntu/ trusty/universe fortunes all 1:1.99.1-7 [1089 kB]

```

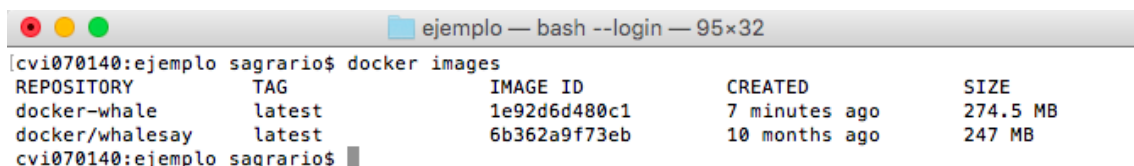
Figura B.9: docker build-1

```

debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Fetch 1961 kB in 16s (115 kB/s)
Selecting previously unselected package librecode0:amd64.
(Reading database ... 13116 files and directories currently installed.)
Preparing to unpack .../librecode0_3.6-21_amd64.deb ...
Unpacking librecode0:amd64 (3.6-21) ...
Selecting previously unselected package fortune-mod.
Preparing to unpack .../fortune-mod_1%3a1.99.1-7_amd64.deb ...
Unpacking fortune-mod (1:1.99.1-7) ...
Selecting previously unselected package fortunes-min.
Preparing to unpack .../fortunes-min_1%3a1.99.1-7_all.deb ...
Unpacking fortunes-min (1:1.99.1-7) ...
Selecting previously unselected package fortunes.
Preparing to unpack .../fortunes_1%3a1.99.1-7_all.deb ...
Unpacking fortunes (1:1.99.1-7) ...
Setting up librecode0:amd64 (3.6-21) ...
Setting up fortune-mod (1:1.99.1-7) ...
Setting up fortunes-min (1:1.99.1-7) ...
Setting up fortunes (1:1.99.1-7) ...
Processing triggers for libc-bin (2.19-0ubuntu6.6) ...
--> ba6b0655095e
Removing intermediate container 537a281e5cfe
Step 3 : CMD /usr/games/fortune -a | cowsay
--> Running in fdcd7c57a48b
--> 1e92d6d480c1
Removing intermediate container fdcd7c57a48b
Successfully built 1e92d6d480c1
cvi070140:ejemplo sagrario$

```

Figura B.10: docker build-2



```

[cvi070140:ejemplo sagrario$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker-whale         latest              1e92d6d480c1        7 minutes ago      274.5 MB
docker/whalesay      latest              6b362a9f73eb        10 months ago      247 MB
cvi070140:ejemplo sagrario$

```

Figura B.11: docker images2

Referencias

- [1] <https://soundcloud.com/thenewstackmakers/solomon-hykes-creator-of-the-docker-open-source-initiative> Audio de entrevista a Solomon Hykes.
- [2] <https://www.docker.com/what-docker> Web oficial de Docker.
- [3] DOCKER CONTAINERS Cristopher Negus, 2015.
- [4] <https://docs.docker.com/engine/understanding-docker> Documentación oficial de Docker, arquitectura.
- [5] DOCKER: UP & RUNNING Sean P. Kane, Karl Matthias, 2015.
- [6] <http://koldohernandez.com/docker-que-es-y-caracteristicas-principales/> Blog Koldo Hernández.
- [7] <http://vagrantanddocker.blogspot.com.es/2015/06/cuales-son-sus-ventajas-y-desventajas.html>
- [8] <http://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html> Blog Javier Garzas, introducción a Docker
- [9] <http://blog.takipi.com/ignora-la-alharaca-5-ideas-erroneas-sobre-docker-que-los-desarrolladores-java-deben-tener-claras/> Blog de Hosh Dreyfuss, ideas erróneas sobre Docker
- [10] http://coast.pink/docker-software_1283959.html
- [11] <https://blog.docker.com/2013/10/dotcloud-is-becoming-docker-inc/> Blog de Docker.
- [12] <https://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/> Blog de Docker.
- [13] <http://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/> Artículo en zdnet.
- [14] http://www.infoworld.com/article/2925484/application-virtualization/look-whos-helping-build-docker-besides-docker-itself.html?utm_content=buffer5d7&utm_medium=social&utm_source=linkedin.com&utm_campaign=buffer Revista digital InfoWorld
- [15] <https://www.docker.com/company> Web oficial de Docker, acerca de Docker.
- [16] <https://www.docker.com/products/docker-toolbox> Documentación oficial de Docker, Toolbox
- [17] <https://www.docker.com/products/docker-engine> Documentación oficial de Docker, Docker Engine.
- [18] <https://docs.docker.com/compose/overview/> Documentación oficial de Docker, Docker Compose
- [19] <https://docs.docker.com/machine/> Documentación oficial de Docker, Docker Machine
- [20] <https://docs.docker.com/kitematic/userguide/> Documentación oficial de Docker, Kitematic.

- [21] <http://www.gonzalonazareno.org/cloud/material/IntroVirtualizacion.pdf> Blog IES Gonzalo Zareno
- [22] <https://docs.docker.com/engine/userguide/> Documentación oficial de Docker, guía de usuario.
- [23] <https://docs.docker.com/engine/reference/commandline/ps/> Documentación oficial de Docker, comando ps.
- [24] <https://docs.docker.com/engine/reference/builder/> Documentación oficial de Docker, comando build.
- [25] <http://www.javiergarzas.com/2015/11/para-los-que-empiezan-crear-y-ejecutar-una-imagen-propia-en-un-contenedor-docker-22.html> Blog Javier Garzas, tutorial Docker.