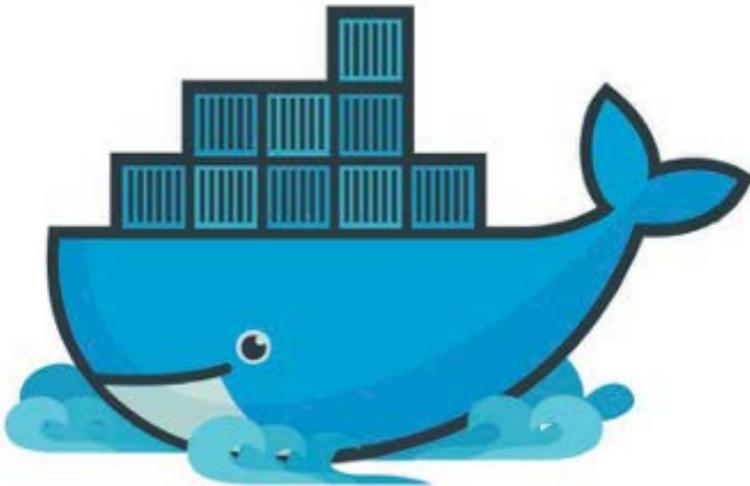


DOCKER

Guía práctica



Alberto González

Alfaomega

libros

DOCKER

Guía práctica

Alberto González Rodríguez

 **Alfaomega**



Dirección de colección y pre-impresión: Grupo RC

Diseño de cubierta: Cuadratin

Datos catalográficos

González, Alberto
Docker. Guía práctica
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-622-963-7

Formato: 17 x 23 cm

Páginas: 256

Docker. Guía práctica

Alberto González Rodríguez

ISBN: 978-84-944650-8-6 edición original publicada por RC Libros, Madrid, España.

Derechos reservados © 2017 RC Libros

Primera edición: Alfaomega Grupo Editor, México, julio 2017

© 2017 Alfaomega Grupo Editor, S.A. de C.V.

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720, Ciudad de México.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro No. 2317

Pág. Web: <http://www.alfaomega.com.mx>

E-mail: atencionalcliente@alfaomega.com.mx

ISBN: 978-607-622-963-7

Derechos reservados:

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera darsele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México. Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. San costo: 01-800-020-4396 E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia. Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile. Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215 piso 10, C.P. 1055, Buenos Aires, Argentina. – Tel./Fax: (54-11) 4811-0887 y 4811-7183 – E-mail: ventas@alfaomegageditor.com.ar

ÍNDICE

Prefacio.....	XI
Capítulo 1: Introducción	1
Contenedores vs virtualización	2
Docker al rescate	4
El futuro.....	6
Contenido de un contenedor	7
Imágenes.....	9
Componentes.....	10
Capítulo 2: Instalación.....	13
Instalación en Linux (Versión comunitaria)	13
Instalación automática	13
Ubuntu.....	14
Debian	15
CentOS y Red Hat Enterprise Linux.....	17
Fedora.....	19
Instalación en Linux (Versión empresarial)	20
Red Hat Enterprise Linux	20
SUSE Linux Enterprise	21
Habilitar e iniciar servicio	23
Instalación en OS X.....	23
Instalación en Windows (Version comunitaria).....	25
Instalación en Windows Server (Version empresarial).....	27
docker info	28
Capítulo 3: Primeros pasos	29
Ejecutar primer contenedor	29
Inspeccionar un contenedor.....	32
Contenedor en modo interactivo	33
Contenedor en segundo plano	34
Detener contenedor	36
Iniciar contenedor	36
Exponer puertos	37

Capítulo 4: Contenedores	41
Crear contenedores sin ejecutarlos.....	41
Acciones básicas sobre contenedores	42
start [<i>opciones</i>] contenedor.....	43
attach [<i>opciones</i>] contenedor	43
stop [<i>opciones</i>] contenedor	44
restart [<i>opciones</i>] contenedor	44
rename contenedor nuevo_nombre	44
ps [<i>opciones</i>].....	44
logs [<i>opciones</i>] contenedor.....	45
pause/unpause contenedor.....	46
kill [<i>opciones</i>] contenedor.....	46
top contenedor [<i>opciones ps</i>]	47
rm [<i>opciones</i>] contenedor.....	47
exec [<i>opciones</i>] contenedor comando	47
run/create [<i>opciones</i>] imagen comando	48
Copia de seguridad.....	50
Restaurar	50
Inspeccionando contenedores	50
Obteniendo estadísticas de uso.....	56
Obteniendo eventos desde el servidor	56
Capítulo 5: Imágenes	59
Introducción	59
Listar imágenes	59
Buscar imágenes en el repositorio oficial	60
Descargar imagen.....	62
Historial de una imagen	63
Copia de seguridad.....	64
Restaurar	65
Eliminar una imagen	65
Convertir un contenedor en una imagen	66
Etiquetado de imágenes.....	68
Publicar una imagen en el repositorio oficial	69
Repositorio local.....	70
Configurar repositorio	74
Capítulo 6: Generar imágenes.....	79
Dockerfile	79
FROM	83
MAINTAINER	83
RUN	83
CMD	84
EXPOSE	85
ADD.....	85

COPY	87
ENTRYPOINT	87
VOLUME	88
USER	89
WORKDIR	90
SHELL	90
LABEL	91
ENV	91
ARG	92
HEALTHCHECK	93
Ejemplo.....	93
Capítulo 7: Redes	95
Introducción	95
Redes predefinidas	95
Listar redes.....	96
Crear red	97
Crear red con rango autogenerado.....	97
Crear red con rango específico	98
Crear red sin acceso al exterior.....	98
Especificar red al crear un contenedor	98
Inspeccionar red	99
Conectar y desconectar contenedor a/de una red	100
Eliminar una red.....	102
Capítulo 8: Almacenamiento	103
Introducción	103
AUFS.....	106
OverlayFS/OverlayFS2.....	109
Device mapper.....	111
Brtrfs	117
ZFS.....	121
Volúmenes	124
<i>Plugins para volúmenes</i>	129
Capítulo 9: Etiquetas	133
Introducción	133
Contenedores	133
Imágenes.....	135
Volúmenes	137
Redes	137
Capítulo 10: Limitar recursos	139
Introducción	139
Memoria	139
Procesador	143

Almacenamiento: Entrada/Salida (I/O)	145
docker update.....	148
Capítulo 11: Registros.....	151
Logging drivers	151
json-file	153
syslog	154
journald	156
gelf	157
fluentd	158
awslogs	160
splunk	161
etwlogs	162
gcplogs.....	162
Capítulo 12: Docker Compose	165
Introducción.....	165
Instalación	167
Primeros pasos	168
Acciones básicas	171
up [<i>opciones</i>] [<i>servicio</i>].....	171
ps [<i>servicio</i>].....	172
down [<i>opciones</i>] [<i>servicio</i>].....	172
start [<i>servicio</i>]	172
stop [-t <i>timeout</i>] [<i>servicio</i>].....	172
Sintaxis plantilla.....	173
<i>version</i> : <i>versión</i>	173
build: directorio	173
context: directorio o url.....	174
dockerfile: fichero.....	174
args: argumentos.....	174
command: comando.....	174
entrypoint: comando.....	174
container_name: nombre.....	175
depends_on: servicios.....	175
environment: valores.....	175
env_file: fichero.....	175
expose: puertos	176
imagen: imagen	176
labels: <i>etiquetas</i>	176
logging: <i>configuración</i>	176
network_mode: <i>red</i>	176
networks: <i>redes</i>	177
ports: <i>puertos</i>	177
volumes: volúmenes.....	177

Ejemplo	177
Acciones	179
<i>pause [servicio]</i>	179
<i>unpause [servicio]</i>	180
<i>build [opciones] [servicio]</i>	180
<i>config [opciones]</i>	181
<i>create [opciones] [servicio]</i>	182
<i>events [opciones] [services]</i>	183
<i>exec [opciones] servicio comando [argumentos]</i>	183
<i>kill [opciones] [servicio]</i>	183
<i>logs [opciones] [servicio]</i>	184
<i>port [opciones] servicio puerto</i>	185
<i>pull [--ignore-pull-failures] [servicio]</i>	185
<i>push [--ignore-push-failures] [servicio]</i>	185
Capítulo 13: Docker Cloud	187
Introducción	187
Terminología	187
Instalar nuestro propio nodo	188
Desplegar un servicio	191
Crear un <i>stack</i>	194
Sintaxis plantilla <i>stack</i>	196
Escalar un servicio	197
<i>Triggers</i> (disparadores)	198
Repositorios	199
Conectar a Github	201
Autotest	204
Cliente Docker Cloud	206
Instalación	206
Acciones	208
<i>action [subacción]</i>	208
<i>container [subacción]</i>	209
<i>event</i>	210
<i>node [subacción]</i>	210
<i>nodecluster [subacción]</i>	212
<i>repository [subacción]</i>	213
<i>run [opciones]</i>	213
<i>service [subacción]</i>	215
<i>stack [subacción]</i>	217
<i>trigger [subacción]</i>	218
<i>up [opciones]</i>	218
<i>docker-cloud stack up</i>	218

Capítulo 14: Docker Hub	219
Introducción	219
Organizaciones	219
Repositorios.....	221
Automatizar creación	221
Webhooks.....	224
Capítulo 15: Swarm	227
Introducción	227
Terminología	228
Arquitectura	228
Crear un <i>Swarm</i>	229
Añadir los <i>nodos de trabajo</i>	231
Desplegar un servicio	232
Escalar un servicio	233
Eliminar un servicio	235
Publicar puertos	235
Eliminar nodo	237
docker service.....	237
ls <i>[opciones]</i>	237
inspect <i>[opciones]</i> servicio.....	238
ps <i>[opciones]</i> servicio.....	239
scale <i>servicio=replicas</i>	239
create <i>[opciones]</i> <i>imagen</i>	239
update <i>[opciones]</i> servicio	241
docker node.....	243
ls <i>[opciones]</i>	243
ps <i>[opciones]</i> <i>[nodo]</i>	244
inspect <i>[opciones]</i> <i>[nodo]</i>	244
update <i>[opciones]</i> <i>[nodo]</i>	245
Índice analítico	247

PREFACIO

En estos tiempos tan cambiantes en el mundo de la tecnología, hay muchos productos que nacen en el mercado muy prometedores pero que desaparecen tan rápido como han aparecido.

En el mundo de la virtualización han surgido durante los últimos años diferentes soluciones que han servido para ahorrar costes a las empresas y mejorar sus prestaciones. Pero ante las nuevas necesidades, principalmente por el denominado *Big Data*, una solución ha aparecido con fuerza con el apoyo de las empresas más importantes en el ámbito tecnológico: **Docker**.

Docker se ha convertido en uno de los productos más solicitados y admirados de la actualidad. Este libro cubre todo lo necesario, desde la instalación hasta la administración más avanzada, todo acompañado con una gran variedad de ejemplos.

SOBRE ESTE LIBRO

Este libro está dirigido a todas las personas que quieran iniciarse en **Docker** o a aquella que tenga actualmente conocimientos y quiera profundizar en las tareas a realizar. No es necesario tener un dominio de un sistema operativo particular, pero se recomienda conocer la consola de **Linux** para utilizar los clientes de **Docker** de una forma más fluida.

La instalación se explica tanto para sistemas **Linux**, **Windows** y **OSX**. Aunque se recomienda el uso de una distribución reciente de **Linux**; el uso de los otros sistemas operativos son válidos para seguir el temario de este libro.

Una vez finalizada la lectura de este libro, al ser una guía práctica, el lector estará preparado para la administración de **Docker** y el despliegue de aplicaciones a través de *contenedores*.

Al ser un libro práctico, todos los comandos y las salidas que genera se insertan en un recuadro con el siguiente formato:

```
# docker --version  
Docker version 1.13.0, build 49bf474
```

SOBRE EL AUTOR

Alberto González trabaja actualmente como *Cloud Consultant* en la empresa *Red Hat*. Anteriormente trabajó como *Administrador de Sistemas Senior* en la empresa *IBM*. En sus 15 años de experiencia ha centrado sus conocimientos en tecnologías principalmente de código abierto, centrándose en **Linux** y virtualización. Posee conocimientos de programación y base de datos.

Además, es profesor en una plataforma en línea ofreciendo cursos de calidad y prácticos de diferentes tecnologías (mayoritariamente de código abierto). La página web de sus cursos es <https://www.oforte.net>.

AGRADECIMIENTOS

El mayor agradecimiento es a mi sobrina Leire, que hace que quiera ser mejor persona cada día y ser un ejemplo para ella.

También mi agradecimiento a mi buen amigo Jonathan Veites Penedo, por acompañarme y apoyarme en la mayoría de los proyectos que he iniciado, e igualmente a todos mis amigos de la infancia, que a pesar de tantos años alejados siguen estando allí como si no hubiese distancia. Gracias a mis familiares por sentirse orgullosos de mis progresos. Gracias a Míriam Pérez por ayudarme a revisar el libro.

Por último, agradezco a todas las empresas y compañeros con los que he trabajado en estos 15 años, donde he crecido como persona y como profesional.

1 INTRODUCCIÓN

Docker se ha convertido en uno de los proyectos más populares en la actualidad. Grandes empresas tecnológicas han apoyado este proyecto en los últimos años, ayudando a su desarrollo y a su evolución. Empresas como *Red Hat*, *Google*, *IBM* o *Microsoft* no solo han colaborado económicamente, sino que también han proporcionado código y soporte para solucionar determinados errores.

Docker en sí no es una tecnología, sino la forma de acceder a ella. Dicha tecnología se conoce con el nombre de contenedores y fue adoptada por el núcleo de Linux recientemente. La finalidad de Docker es facilitar la creación y manipulación de los contenedores.

La tecnología de contenedores no es algo nuevo, es una forma histórica de intentar aislar recursos tanto a nivel de usuario como a nivel de aplicación. En *Linux*, el primer acercamiento previo a los contenedores se realizó a través de la operación "*chroot*", también conocida como "jaula"; que consistía en aislar aplicaciones y usuarios entre sí, con la limitación de no poder aislar recursos físicos (memoria, procesador o dispositivos). Las primeras implementaciones de "*chroot*" datan de principios de los años 80.

A partir del año 2000, aparecen las nuevas implementaciones para aislar los recursos, ya no solo para *Linux* sino para sistemas *UNIX* (Sistemas *BSD*, *Solaris*, *AIX*) y *Windows*.

Virtuozzo fue la empresa pionera en desarrollar un software para el aislamiento de recursos a nivel de sistema operativo. Su aplicación era de código cerrado, es decir, no disponible al público general. La aplicación, llamada también *Virtuozzo*, era capaz de aislar los recursos entre usuarios, limitaba el acceso a los dispositivos y separaba el acceso de las aplicaciones entre ellas. En el año 2005 la empresa lanzó, esta vez como

código abierto, el software *OpenVZ*. Este software todavía es muy popular siendo utilizado por pequeñas y grandes empresas, sobre todo aquellas que ofrecen *VPS* (servidores privados virtualizados) a bajo precio.

Entre los años 2000 y 2005, en sistemas UNIX, la tecnología avanzó en el ámbito del aislamiento de recursos. *FreeBSD* (un sistema operativo *UNIX*) implementó “*FreeBSD jail*” similar a “*chroot*” pero logrando aislar recursos. En 2001, aparece *Linux-VServer* como alternativa gratuita a *Virtuozzo* siendo considerado uno de los predecesores de los contenedores actuales. En el año 2004 aparece una de las mejores tecnologías para aislar recursos: las zonas de *Solaris*. Las zonas son consideradas una de las mejores implementaciones del aislamiento de recursos hasta la aparición de **Docker**. Las zonas de *Solaris* lograron reducir costes a empresas y reducir la infraestructura física de empresas que utilizaban servidores *Sun*.

Hasta la aparición de **Docker** en el año 2013, caben destacar las distintas implementaciones de aislamiento de recursos (conocidas como contenedores):

- *Workload partitions (WPARs)* para el sistema operativo de *IBM AIX*, en el año 2007.
- *HP-UX Containers* para el sistema operativo *HP-UX*, en el año 2007.
- *Linux containers (LXC)*, en el año 2008.

En los primeros años de vida de **Docker** (2013 y 2014), este utilizaba *LXC* para el uso de contenedores. A partir de la versión 0.9, empezó a utilizar su propia librería (*libcontainer*) para la manipulación de los contenedores.

Contenedores vs virtualización

Una de las primeras dudas que surgen con Docker es saber diferenciar claramente las ventajas y desventajas que existen entre el uso de contenedores y la virtualización.

Antes de introducirnos en el contexto de las ventajas y desventajas, debemos revisar qué es la virtualización y los diferentes tipos que existen para evitar la confusión con los contenedores.

La virtualización consiste en añadir una capa de abstracción a los recursos físicos con el objetivo de mejorar el uso de los recursos del sistema. En el pasado, cada ele-

mento físico ejecutaba un recurso. Con la introducción de la virtualización es posible crear varios entornos simulados (máquinas virtuales) para diversos recursos. Es decir, en el caso de un servidor, gracias a la virtualización, podemos ejecutar varios sistemas dentro del mismo con diferentes entornos. Por ejemplo, un servidor puede ejecutar múltiples máquinas virtuales con diferentes sistemas operativos para diversos propósitos.

Los diferentes tipos de virtualización:

- **Virtualización completa:** la máquina virtual no tiene acceso directo a los recursos físicos y requiere de una capa superior para acceder a ellos.

Algunos ejemplos:

- VirtualBox
- QEMU
- Hyper-V
- VMware ESXi

- **Virtualización asistida por hardware:** es el *hardware* el que facilita la creación de la máquina virtual y controla su estado. Algunos ejemplos:

- KVM
- Xen
- VMWare fusión

- **Virtualización a nivel de sistema operativo:** aquí incluimos los contenedores. Es el sistema operativo, y no el *hardware*, el encargado de aislar los recursos y proporcionar las herramientas para crear, manipular o controlar el estado de los contenedores (término utilizado en lugar de máquina virtual).

Para entender con más detalle cómo funciona **Docker**, le dedicaremos el último capítulo del libro para explicar cómo se integra con el sistema operativo y los “espacios de nombre” para aislar los recursos.

Docker al rescate

Hasta ahora hemos visto de forma resumida la historia de los contenedores y las ventajas que presenta frente a la virtualización tradicional. Pero aún no hemos respondido al porqué **Docker** es tan popular tanto para administradores de sistema como para desarrolladores.

La virtualización fue un soplo de aire fresco para la tecnología en términos de reducción de costes e infraestructura. Además, agilizó el trabajo en equipo entre los administradores de sistema y los desarrolladores. Un nuevo puesto de trabajo apareció con fuerza: la persona encargada de los entornos virtuales, ya sea *Xen* o *VMWare*, liberando la carga a los administradores de sistema que a partir de ese momento dejaban de encargarse de la parte física de los servidores.

El administrador de virtualización proporcionaba una máquina virtual con los requisitos previamente establecidos y el administrador de sistema era el encargado de instalar y configurar el sistema operativo. Una vez finalizada su tarea, por ejemplo instalar una base de datos o un servidor web, daba acceso a los desarrolladores para que pudieran desplegar sus aplicaciones.

Esta solución resultó válida durante varios años; sin embargo, en el ámbito de la tecnología que avanza a una velocidad vertiginosa, empezaron a surgir diversos problemas:

- Los desarrolladores aún estaban limitados: sus entornos estaban administrados por terceras personas.
- Los administradores de sistemas no controlaban los servidores: los desarrolladores manipulaban los sistemas y era difícil cumplir todos sus requisitos.
- Los administradores de virtualización se quejaban de los administradores de sistemas y de los desarrolladores: cada vez solicitaban cosas más inusuales.

Esto se debía a que las nuevas tecnologías requieren de entornos dinámicos, que sean fáciles de crear, destruir y transferir entre diferentes plataformas. Las aplicaciones ya no debían regirse a un tipo de *hardware*, a un sistema operativo o incluso a una versión de una distribución específica. Era necesaria la creación de una manera de programar más ágil sin tener en cuenta qué había por debajo de la aplicación.

Y de repente aparecieron los contenedores. Aunque habíamos comentado anteriormente que los contenedores llevaban tiempo en el mercado, **Docker** apareció en el momento preciso convirtiéndose en la solución del futuro.

Este nuevo proyecto solventó los siguientes problemas:

- Los desarrolladores ya no requerían de los administradores de sistemas o de terceras personas. Solo necesitaban un sistema operativo para ejecutar **Docker**, el cual era el encargado de lanzar un entorno virtual con una distribución requerida por el desarrollador, sin importar el sistema operativo o la distribución del sistema padre.
- Los administradores de sistemas ya solo requerían solicitar un sistema (físico o virtual) con unas prestaciones no demasiado grandes donde instalar y ejecutar los contenedores **Docker**. El uso de memoria y procesador de la nueva solución lograba reducir los problemas que surgían de la virtualización estándar.
- Los administradores de virtualización ya no necesitaban tanto esfuerzo o tanta dedicación a todas las peticiones que venían indirectamente de los desarrolladores. Ahora, solo era necesario mantener un sistema con unas prestaciones moderadas. Cabe destacar que en muchos casos Docker forma parte de sistemas físicos y no virtuales, requiriendo en algunos casos menos personal para administrar los entornos físicos.

Como observamos, **Docker** ha solventado, al menos temporalmente, muchos de los problemas que habían surgido con las nuevas tecnologías. En cambio no ha solucionado un asunto: el trabajo entre administradores de sistemas y desarrolladores. Con tal motivo, surge una nueva posición laboral: los *DevOps*.

Seguramente habréis notado que en los portales de búsqueda de empleo han empezado a solicitar más y más *DevOps*. ¿Pero qué es esta posición? ¿Y por qué hay otra llamada *SysOps*? Resumamos estas posiciones:

- *SysOps*: son los clásicos administradores de sistemas. Se encargan del sistema operativo y de la monitorización de los servidores. Normalmente son los encargados de instalar software, configurar el sistema y mantener las aplicaciones ejecutándose.
- *DevOps*: este nuevo rol consiste en un puente entre los desarrolladores y los administradores de sistema. Su fuerte es la administración de sistemas pero tienen un amplio conocimiento en los entornos de desarrollo: saben cómo funcionan los controles de versión, la integración entre diversos entornos, el desarrollo continuo y

sobre todo, solventan las peticiones de los desarrolladores sin necesidad de un procedimiento largo.

Como observamos, **Docker** se ha posicionado en el mercado de una forma muy clara. Los desarrolladores pueden lanzar los entornos necesarios, desarrollar sus aplicaciones y desplegarlas dentro de un contenedor, el cual se convertirá en una imagen para su uso en diversos entornos.

Serán los *DevOps* o los *SysOps* los encargados de ejecutar esa imagen (con todo lo necesario para ejecutar la aplicación) en los diversos entornos: desde el entorno de pruebas hasta el entorno de producción. Y todo eso sin preocuparse si el sistema contiene las versiones correctas de librerías o la distribución apropiada.

El futuro

En el mundo de la tecnología, es prácticamente imposible predecir el futuro. Ya no solo es difícil saber qué pasará, sino también conocer si ciertas tecnologías o *software* van a sobrevivir en un entorno tan cambiante y con tanta demanda de nuevos requisitos. Solo hay que ver empresas como *Heroku*, que surgió con gran popularidad para solventar los problemas de los desarrolladores y que ha perdido fuerza con la aparición de **Docker**.

El presente y el futuro próximo están ligados a los *microservicios*: aplicaciones que se ejecutan sin necesidad de alojar datos en disco, facilitando el despliegue de nuevos servidores para soportar la demanda de carga y ahorrar costes. Con los *microservicios* es posible tener una infraestructura dinámica: la carga de usuarios o de operaciones marcará el número de servidores necesarios.

Como en toda innovación, siempre hay críticos y personas que ponen en duda diversas soluciones. **Docker** no ha sido una excepción y ha recibido las siguientes acusaciones:

- **No está preparado para producción:** esta es la crítica que surge desde diferentes sectores, sobre todo de los administradores de sistemas y administradores de virtualización. Cabe decir que **Docker** ha evolucionado mucho, como es lógico debido al apoyo de grandes empresas, en la estabilidad y en la madurez del producto. En las primeras versiones era habitual un bajo rendimiento en aplicaciones con gran uso de entrada y salida, tanto en disco como en procesador.

- **No hay alta disponibilidad:** otra de las críticas que surgen, quizá relacionada con la anterior, es que no fue inicialmente diseñado para la alta disponibilidad. Nuevas soluciones aparecieron para solventarlo, la más popular es *Kubernetes*. En la actualidad, **Docker** ofrece un componente llamado *Swarm*.
- **Falta de seguridad:** una de las demandas más comunes es la posibilidad de aislar los contenedores entre sí a nivel de red. Las nuevas versiones de Docker permiten crear redes virtuales para aislar contenedores entre ellos. Además, la seguridad de las imágenes estuvo en entredicho hasta la introducción de un control de veracidad llamada *digest*.
- **Inestabilidad en los “drivers”:** la diversidad de *drivers* utilizados por Docker para sus sistemas de ficheros ha sido la gran odisea a solventar, ya que cada uno de ellos contiene ventajas y desventajas. Lo veremos con más calma en el capítulo 8 de Almacenamiento: *Drivers*.

El producto **OpenShift**, a través de su versión *community* llamada **Openshift Origin** y la versión empresarial **OpenShift Container Platform** (anteriormente conocido como **OpenShift Enterprise**), ha copado el mercado de los contenedores a nivel profesional al ser una de las mejores herramientas para el desarrollo de aplicaciones de forma acelerada y para *DevOps*. Utiliza *Kubernetes* como administrador de cluster de contenedores y simplifica la creación de un **PaaS** (*Platform As a Service*) dentro de la infraestructura de la empresa. Este producto de **Red Hat** también se ofrece en las siguientes opciones:

- **OpenShift Online:** Multi-tenant, plataforma de contenedores basado en la nube.
- **OpenShift Dedicated:** Single-tenant, plataforma de contenedores basado en la nube. Disponible en *Amazon Web Services* (AWS) y *Google Cloud Platform* (GCP)

Contenido de un contenedor

Un contenedor está compuesto de todo lo necesario para ejecutar una o varias aplicaciones. Su contenido es el siguiente:

- Librerías del sistema operativo: como comentábamos anteriormente, dentro de un contenedor tenemos todo lo necesario de una distribución para aislarlo del

sistema que ejecuta el servicio de **Docker**. Un ejemplo son las librerías para SSL cuya finalidad es que el servidor web pueda aceptar conexiones seguras.

- Herramientas del sistema: existe una gran diversidad de herramientas dentro de cada contenedor. Algunos ejemplos:

- Editores de texto (VI, Emacs).
- Monitorización (Nagios, Check-MK).
- Herramientas de registros (Elasticsearch, Splunk).

- Runtime: es el software que se necesita para ejecutar la aplicación dentro del contenedor. Puede ser:

- Lenguajes interpretados: el intérprete para el código de fuente, como puede ser PHP, Perl o Python.
- Máquina virtual Java.
- El programa compilado y autoejecutable.
- En el caso de lenguajes interpretados, el código fuente (por ejemplo, ficheros .php, .py o .pl) o en el caso de máquina virtual los ficheros que contienen la aplicación (por ejemplo .jar o .war).

Dentro del contenedor tendremos las utilidades, la distribución que hemos elegido y utilizando los gestores de paquete (por ejemplo, APT o YUM) podremos instalar software de los repositorios de la misma.

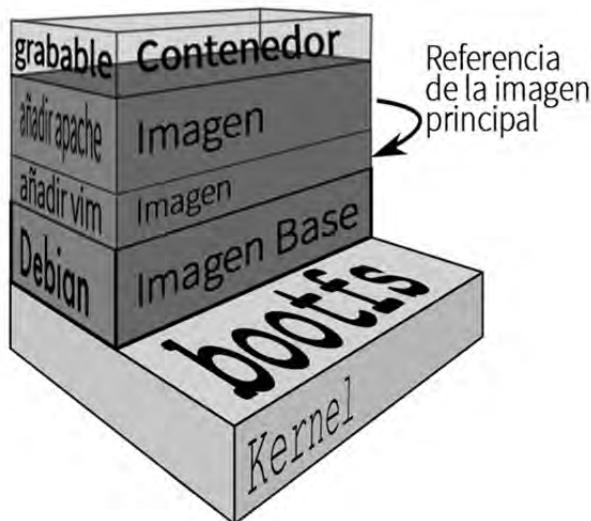
Un contenedor puede ejecutar varias aplicaciones. No obstante, se recomienda separarlas en varios contenedores debido a que estos son muy ligeros. De esta manera, las tareas de mantenimiento serán más fáciles. Algunos ejemplos:

- Monitorizar las aplicaciones individualmente monitorizando el uso del contenedor.
- Limitar los recursos de las aplicaciones.
- Reinicio de aplicaciones (ya que no afectamos a las demás).
- Copia de seguridad y restauración.

Imágenes

Antes de profundizar en **Docker**, es necesario definir un nuevo concepto: las imágenes. Hasta ahora hemos visto qué es un contenedor y sus ventajas. Pero seguramente haya surgido ya la pregunta: ¿De dónde surge el contenido de los contenedores? Y la respuesta es: de una imagen.

Una imagen contiene distintas capas de datos (la distribución, diferente software, librerías y la personalización).



Como vemos en esta representación, tenemos una imagen base, que contiene la distribución Debian y a la que se le ha añadido diferentes capas:

- Instalación del editor Emacs
- Instalación del servidor web Apache

Una vez ejecutado un contenedor basándonos en esta imagen, tendremos instalados el editor y el servidor web y no tendremos que hacerlo manualmente.

Todo contenedor puede ser convertido en una imagen empleando las utilidades de Docker. Estas imágenes pueden ser transferidas a otro servidor para ejecutar un contenedor basado en ellas y pueden hacerse copias de seguridad. Más adelante, veremos cómo utilizar imágenes dentro del repositorio público de Docker y cómo alojar nuestras propias imágenes en el mismo.

En Docker, las imágenes contienen un historial y un control de versiones, lo que facilita listar los cambios y poder restaurar la anterior.

Componentes

En marzo de 2017, la compañía *Docker Inc* anuncia la división en dos variantes de la aplicación:

- Edición comunitaria (CE, *Community Edition*). Contiene dos versiones:
 - *Stable*: Actualizaciones cada trimestre.
 - *Edge*: Actualizaciones cada mes.
- Edición empresarial (EE, *Enterprise Edition*). Diseñada para empresas y equipos que ofrecen aplicaciones en negocios con necesidades críticas. Contiene tres ediciones:
 - Básica: incluye soporte para infraestructura certificada, plugins e ISV (*Independent Software Vendors*).
 - Estándar: incluye el soporte de la versión básica incluyendo soporte para la administración de imágenes y aplicaciones en los contenedores.
 - Avanzada: esta versión añade a la versión estándar el escaneo de imágenes en búsqueda de problemas de seguridad.

Gracias a su popularidad, **Docker** ha progresado y se ha desarrollado en numerosas áreas, por lo que actualmente, ya no solo consiste en un software para manejar contenedores. Se ha diversificado en los siguientes términos:

INSTALACIÓN

2

Instalación en Linux

Docker está incluido en la mayoría de las distribuciones de Linux. En esta sección veremos los pasos a seguir para la instalación en las distribuciones más populares. Los requisitos para el uso de Docker son los siguientes: un procesador de 64 bits y un núcleo (Kernel) versión 3.10 o superior. Para comprobar su versión del núcleo ejecute el siguiente comando:

```
# uname -r  
4.4.0-59-generic
```

INSTALACIÓN AUTOMÁTICA

Es posible instalar Docker automáticamente utilizando este comando (con el usuario *root* o utilizando *sudo*) en cualquiera de las distribuciones más populares:

```
# curl -fsSL https://get.docker.com/ | sh  
[omitido]  
+ sh -c docker version  
Client:  
Version: 1.13.0  
API version: 1.25  
Go version: go1.7.3  
Git commit: 49bf474  
Built: Tue Jan 17 09:58:26 2017  
OS/Arch: linux/amd64  
  
Server:  
Version: 1.13.0
```

```
API version: 1.25 (minimum version 1.12)
Go version: go1.7.3
Git commit: 49bf474
Built: Tue Jan 17 09:58:26 2017
OS/Arch: linux/amd64
Experimental: false
```

UBUNTU

Las versiones soportadas para el uso de Docker en la distribución Ubuntu son las siguientes:

- Yakkety 16.10
- Xenial 16.04 (LTS)
- Trusty 14.04 (LTS)



Para preparar la instalación, en un terminal (consola) deberemos ejecutar los siguientes comandos para añadir el repositorio oficial:

Actualizar repositorios.

```
$ sudo apt-get update
Hit:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Hit:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Reading package lists... Done
```

Instalar, o actualizar, las utilidades para confiar en el certificado de Docker.

```
$ sudo apt-get install -y apt-transport-https ca-certificates
Reading package lists... Done
Building dependency tree
ca-certificates is already the newest version (20160104ubuntu2).
[omitido]
Setting up apt-transport-https (1.2.19) ...
```

Confiar en el certificado del repositorio de Docker.

```
$ curl -fsSL https://yum.dockerproject.org/gpg | sudo apt-key add -
OK
```

Añadir el repositorio oficial de Docker para Ubuntu.

```
$ sudo add-apt-repository "deb https://apt.dockerproject.org/repo/ubuntu-$(lsb_release -cs) main"
```

Una vez configurado el repositorio oficial, es posible proceder a la instalación de Docker que incluye el servidor y el cliente.

Actualizar repositorios para obtener la lista de paquetes del repositorio Docker.

```
$ sudo apt-get update
[omitido]
Get:5 https://apt.dockerproject.org/repo ubuntu-xenial InRelease [30.2 kB]
Get:6 https://apt.dockerproject.org/repo ubuntu-xenial/main amd64 Packages [3,480 B]
```

Instalar el paquete docker-engine.

```
$ sudo apt-get install -y docker-engine
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
aufs-tools cgroupfs-mount libltdl7
Suggested packages:
mountall
The following NEW packages will be installed:
aufs-tools cgroupfs-mount docker-engine libltdl7
0 upgraded, 4 newly installed, 0 to remove and 28 not upgraded.
[omitido]
Setting up cgroupfs-mount (1.2) ...
Setting up libltdl7:amd64 (2.4.6-0.1) ...
Setting up docker-engine (1.13.0-0~ubuntu-xenial) ...
Processing triggers for libc-bin (2.23-0ubuntu5) ...
Processing triggers for systemd (229-4ubuntu13) ...
Processing triggers for ureadahead (0.100.0-19) ...
```

DEBIAN

Las siguientes versiones de la distribución *Debian* están soportadas:

- Stretch (testing)
- Jessie 8.0 (LTS)
- Wheezy 7.7 (LTS)



Para *Debian 7.7 Wheezy* debemos habilitar el repository "backports" debido a que la versión del núcleo de *Linux* incluida es 3.2.

```
$ sudo echo "deb http://http.debian.net/debian wheezy-backports main" >> /etc/apt/sources.list
```

Para preparar la instalación, en un terminal (consola) deberemos ejecutar los siguientes comandos para añadir el repositorio oficial:

Actualizar repositorios.

```
$ sudo apt-get update
Hit http://security.debian.org jessie/updates InRelease
Hit http://security.debian.org jessie/updates/main Sources
Hit http://security.debian.org jessie/updates/main amd64 Packages
Ign http://httpredir.debian.org jessie InRelease
Hit http://httpredir.debian.org jessie Release.gpg
Hit http://httpredir.debian.org jessie Release
Hit http://httpredir.debian.org jessie/main Sources
Hit http://httpredir.debian.org jessie/main amd64 Packages
Reading package lists... Done
```

En Jessie o Stretch: Instalar, o actualizar, las utilidades para confiar en el certificado de Docker.

```
$ sudo apt-get install curl apt-transport-https ca-certificates software-properties-common
```

Wheezy: Instalar, o actualizar, las utilidades para confiar en el certificado de Docker.

```
$ sudo apt-get install curl apt-transport-https ca-certificates python-software-properties
```

Confiar en el certificado del repositorio de Docker.

```
$ curl -fsSL https://yum.dockerproject.org/gpg | sudo apt-key add -
OK
```

Añadir el repositorio oficial de Docker para Debian.

```
$ sudo add-apt-repository "deb https://apt.dockerproject.org/repo/debian-$(lsb_release -cs) main"
```

Una vez configurado el repositorio oficial, es posible proceder a la instalación de Docker que incluye el servidor y el cliente.

Actualizar repositorios para obtener la lista de paquetes del repositorio Docker.

```
$ sudo apt-get update
Get:1 https://apt.dockerproject.org debian-jessie InRelease [30.2 kB]
Hit http://security.debian.org jessie/updates/main amd64 Packages
Get:2 https://apt.dockerproject.org debian-jessie/main amd64 Packages [6,153 B]
[omitted]
```

Instalar el paquete docker-engine.

```
$ sudo apt-get install -y docker-engine
[omitido]
Setting up libnih1 (1.0.3-4.3) ...
Setting up libnih-dbus1 (1.0.3-4.3) ...
Setting up mountall (2.54) ...
Setting up aufs-tools (1:3.2+20130722-1.1) ...
Setting up cgroupfs-mount (1.1) ...
Setting up libapparmor1:amd64 (2.9.0-3) ...
Setting up docker-engine (1.13.0-0~debian-jessie) ...
Setting up liblrror-perl (0.17-1.1) ...
Setting up git-man (1:2.1.4-2.1+deb8u2) ...
Setting up git (1:2.1.4-2.1+deb8u2) ...
```

CENTOS Y RED HAT ENTERPRISE LINUX

A partir de la versión 7 de CentOS y Red Hat Enterprise Linux 7 es posible instalar y ejecutar Docker.



Para preparar la instalación, en un terminal (consola) deberemos ejecutar los siguientes comandos para añadir el repositorio oficial:

Instalar, si es necesario, el paquete yum-utils que incluye la utilidad yum-config-manager.

```
$ sudo yum install -y yum-utils
[omitido]
Installed:
  yum-utils.noarch 0:1.1.31-40.el7
```

Configurar repositorio.

```
$ sudo yum-config-manager \
> --add-repo https://docs.docker.com/engine/installation/linux/repo_files/centos/docker.repo
Loaded plugins: fastestmirror
```

```
adding repo from: https://docs.docker.com/engine/installation/linux/repo_files/centos/docker.repo
grabbing file https://docs.docker.com/engine/installation/linux/repo_files/centos/docker.repo to
/etc/yum.repos.d/docker.repo
repo saved to /etc/yum.repos.d/docker.repo
```

Una vez configurado el repositorio oficial, es posible proceder a la instalación de Docker que incluye el servidor y el cliente.

Actualizar la lista de paquetes.

```
$ sudo yum makecache fast
Loaded plugins: fastestmirror
base                                         | 3.6 kB  00:00:00
docker-main                                    | 2.9 kB  00:00:00
extras                                         | 3.4 kB  00:00:00
updates                                         | 3.4 kB  00:00:00
docker-main/primary_db                         | 28 kB   00:00:00
Loading mirror speeds from cached hostfile
 * base: centos.trisect.eu
 * extras: centos.trisect.eu
 * updates: centos.mirrors.ovh.net
Metadata Cache Created
```

Instalar el paquete docker-engine.

```
$ sudo yum install -y docker-engine
[omitted]
Installed:
  docker-engine.x86_64 0:1.13.0-1.el7.centos
```

Dependency installed:

<i>auditlibs-python.x86_64 0:2.6.5-3.el7</i>	<i>checkpolicy.x86_64 0:2.5-4.el7</i>	<i>docker-engine-</i>
<i>selinux.noarch 0:1.13.0-1.el7.centos</i>	<i>libcgroup.x86_64 0:0.41-11.el7</i>	<i>libseccomp.x86_64</i>
<i>0:2.3.1-2.el7</i>		
<i>libsemanage-python.x86_64 0:2.5-5.1.el7_3</i>	<i>libtalloc-ltdl.x86_64 0:2.4.2-21.el7_2</i>	<i>policycore-</i>
<i>reutils-python.x86_64 0:2.5-11.el7_3</i>	<i>python-ipynote.noarch 0:0.75-6.el7</i>	<i>setools-libs.x86_64</i>
<i>0:3.3.8-1.1.el7</i>		

Dependency Updated:

<i>libsemanage.x86_64 0:2.5-5.1.el7_3</i>	<i>policycoreu-</i>
<i>libs.x86_64 0:2.5-11.el7_3</i>	

Complete!

FEDORA

Docker está soportado en Fedora a partir de la versión 24 y posteriores.



Para preparar la instalación, en un terminal (consola) deberemos ejecutar los siguientes comandos para añadir el repositorio oficial:

Instalar, si es necesario, el paquete `dnf-plugins-core` que incluye la utilidad `config-manager`.

```
$ sudo dnf install -y dnf-plugins-core
[omitiendo]
Installed:
  dnf-plugins-core.noarch 0.1.21-4.fc25           python3-dnf-plugins-core.noarch 0.1.21-4.fc25
Complete!
```

Configurar repositorio.

```
$ sudo dnf config-manager \
> --add-repo https://docs.docker.com/engine/installation/linux/repo_files/fedora/docker.repo
Adding repo from: https://docs.docker.com/engine/installation/linux/repo_files/fedora/docker.repo
```

Una vez configurado el repositorio oficial, es posible proceder a la instalación de Docker que incluye el servidor y el cliente.

Actualizar la lista de paquetes.

```
$ sudo dnf makecache fast
Metadata cache created.
```

Instalar el paquete docker-engine.

```
$ sudo dnf install -y docker-engine
[omitiendo]
Installed:
  audit-lbs-python.x86_64 2.6.7-1.fc25 docker-engine.x86_64 1.13.0-1.fc25 docker-engine-
selinux.noarch 1.13.0-1.fc25 iptables.x86_64 1.6.0-2.fc25 libnetfilter_conntrack.x86_64 1.0.4-
6.fc24 libnftnl.x86_64 1.0.1-8.fc24 libselinux-python.x86_64 2.5-12.fc25 libsemanage-
```

```
python.x86_64 2.5-8.fc25 libtool-ltdl.x86_64 2.4.6-13.fc25 policycoreutils-python.x86_64 2.5-17.fc25 python-iPy.noarch 0.81-16.fc25 python-libs.x86_64 2.7.13-1.fc25
Complete!
```

OPENSUSE Y SUSE LINUX ENTERPRISE

Las siguientes versiones y posteriores de la distribución *openSUSE* y *SLES* están soportadas:

- OpenSuSE Leap 42.x
- SLES 12.x



Para preparar la instalación, en un terminal (consola) deberemos ejecutar los siguientes comandos para añadir el repositorio oficial:

Añadir el repositorio oficial de Docker para Debian.

```
$ sudo zypper addrepo \
> https://yum.dockerproject.org/repo/main/opensuse/13.2/ \
> docker-main
Adding repository 'docker-main' ..... [done]
Repository 'docker-main' successfully added
Enabled : Yes
Autorefresh : No
GPG Check : Yes
URI : https://yum.dockerproject.org/repo/main/opensuse/13.2/
```

Una vez configurado el repositorio oficial, es posible proceder a la instalación de Docker que incluye el servidor y el cliente.

Actualizar repositorios para obtener la lista de paquetes del repositorio Docker.

```
$ sudo zypper refresh
Retrieving repository 'docker-main' metadata ..... [U]
File 'repomd.xml' from repository 'docker-main' is signed with an unknown key 'F76221572C52609D'.
Continue? [yes/no] (no): yes
Retrieving repository 'docker-main' metadata ..... [done]
Building repository 'docker-main' cache ..... [done]
Repository 'openSUSE-42.1-0' is up to date.
Building repository 'openSUSE-42.1-0' cache ..... [done]
Retrieving repository 'openSUSE-Leap-42.1-Oss' metadata ..... [done]
```

```

Building repository 'openSUSE-Leap-42.1-Oss' cache.....[done]
Retrieving repository 'openSUSE-Leap-42.1-Update' metadata .....[done]
Building repository 'openSUSE-Leap-42.1-Update' cache .....[done]
Retrieving repository 'openSUSE-Leap-42.1-Update-Non-Oss' metadata .....[done]
Building repository 'openSUSE-Leap-42.1-Update-Non-Oss' cache .....[done]
All repositories have been refreshed.

```

Instalar el paquete docker-engine.

```
$ sudo zypper install docker-engine
```

Loading repository data...

Reading installed packages...

Resolving package dependencies...

The following 10 NEW packages are going to be installed:

```

docker-engine iptables libcgroup1 libiptc0 libltdl7 libnetfilter_conntrack3 libnftnl0 tar tar-lang
xtables-plugins
[omitido]

```

Checking for file conflicts:[done]

```
{ 1/10} Installing: libcgroup1-0.41-4.2
```

[done]

```
{ 2/10} Installing: libiptc0-1.4.21-4.1
```

[done]

```
{ 3/10} Installing: libltdl7-2.4.2-16.6
```

[done]

```
{ 4/10} Installing: libnftnl0-1.0.1-9.2
```

[done]

```
{ 5/10} Installing: libnetfilter_conntrack3-1.0.4-5.1
```

[done]

```
{ 6/10} Installing: xtables-plugins-1.4.21-4.1
```

[done]

```
{ 7/10} Installing: iptables-1.4.21-4.1
```

[done]

```
{ 8/10} Installing: tar-1.27.1-12.1
```

[done]

```
{ 9/10} Installing: tar-lang-1.27.1-12.1
```

[done]

```
{10/10} Installing: docker-engine-1.13.0-
```

[done]

HABILITAR E INICIAR SERVICIO

Después de la instalación de Docker, debemos habilitar e iniciar el servicio para que al reiniciar el servidor se arranque automáticamente. Para ello seguiremos estos pasos:

Habilitar servicio de Docker

```
$ sudo systemctl enable docker
```

Arrancar el servicio de Docker

```
$ sudo systemctl start docker
```

Comprobar el estado de Docker

```
$ sudo systemctl status docker
```

```
docker.service - Docker Application Container Engine
```

```
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled)
```

```
   Active: active (running) since Mon 2017-01-30 22:53:34 GMT; 36s ago
```

```
     Docs: https://docs.docker.com
```

```
Main PID: 2606 (dockerd)
```

```
CGroup: /system.slice/docker.service
```

```
   └─2606 /usr/bin/dockerd
```

```
      └─2609 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout 2m --state-dir /var/run/docker/libcontainerd/containerd --shim docker-containerd-shim --runtime dock...
```

[omitido]

Instalación en OS X

Los requisitos para la instalación son los siguientes:

- Un modelo de Mac del año 2010 o posterior con procesador Intel.
- OS X Yosemite 10.10.3 o versiones más recientes. Aunque es muy recomendado utilizar OS X El Capitan 10.11 o posterior.
- Al menos 4GB de memoria RAM.
- Virtualbox anterior a 4.3.30 no debe estar instalado.

El instalador de **Docker for MAC** se descarga desde la página oficial: <https://download.docker.com/mac/stable/Docker.dmg>, haciendo doble clic sobre el instalador se mostrará la siguiente ventana.



Al mover la aplicación (**Docker.app**) hacia la carpeta, la instalación comenzará. Para ello, es necesario permisos de administrador (solicitará la contraseña para las tareas necesarias). Una vez instalado podemos ejecutar la aplicación:



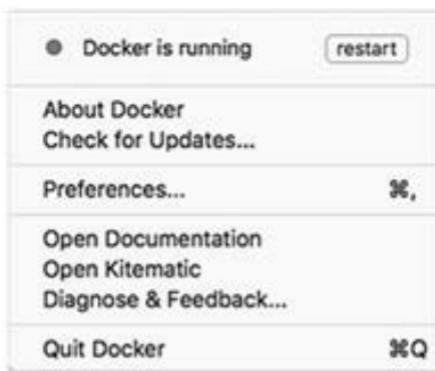
En la barra superior, se mostrará una ballena indicando que **Docker** está en funcionamiento.



En la primera ejecución después de la instalación, también se mostrará la siguiente ventana de bienvenida:



Una vez instalado, podemos abrir un terminal para ejecutar los comandos de Docker para continuar las instrucciones de este libro. Se puede comprobar, en cualquier momento el estado, pulsando sobre la ballena y se mostrará si está en funcionamiento:



Instalación en Windows

Los requisitos para Docker for Windows son los siguientes:

- Las versiones soportadas son las siguientes:

- Windows 10 64bits. Profesional, Enterprise o Educational.
- Windows 2016.
- Microsoft Hyper-V.
- Virtualización habilitada.



El instalador de **Docker for Windows** se descarga desde la página oficial: <https://download.docker.com/win/stable/InstallDocker.msi>, haciendo doble clic sobre el instalador y siguiendo los pasos para aceptar la licencia, se instalará todo lo necesario para utilizar **Docker** y se mostrará la siguiente ventana (Comprobar que *Launch Docker* está seleccionado).



Una vez ejecutado **Docker** aparecerá una ballena en la barra de tareas indicando que se está ejecutando y es accesible. Después de la instalación, por primera vez, aparecerá la siguiente ventana:



Ahora es posible utilizar la consola de Windows, ya sea el histórico *cmd.exe* o *PowerShell* y ejecutar comandos de **Docker**.

docker info

A través de la acción `info` podemos mostrar la información del servidor de Docker. La información mostrada es la siguiente:

```
# docker info
Containers: 5
Running: 1
Paused: 0
Stopped: 4
Images: 234
Server Version: 1.12.5
Storage Driver: aufs
Root Dir: /var/lib/docker/aufs
Backing Filesystem: xfs
Dirs: 303
Dirperm1 Supported: true
Logging Driver: json-file
```

```
Cgroup Driver: cgroups
Plugins:
Volume: local
Network: host bridge overlay null
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Security Options:
Kernel Version: 3.16.0-4-amd64
Operating System: Debian GNU/Linux 8 (jessie)
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 15.7 GiB
Name: cloud.oforte.net
ID: S5BA:6BWG:33OQ:MYEL:NWJ7:HM67:WIOA:3TPJ:3U2G:TQ75:7IUA:B07U
Docker Root Dir: /var/lib/docker
Debug Mode {client}: false
Debug Mode {server}: false
Registry: https://index.docker.io/v1/
Insecure Registries:
127.0.0.0/8
```


3

PRIMEROS PASOS

Ejecutar primer contenedor

Una vez realizada la instalación en el sistema operativo elegido, ya estamos preparados para ejecutar nuestro primer contenedor. Para ello en nuestra consola (terminal) introduciremos el siguiente comando con el usuario root:

```
# docker run -ti debian cat /etc/debian_version
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
5040bd298390: Pull complete
Digest: sha256:abbe80c8c87b7e1f652fe5e99ff1799cdf9e087&c7009035afe1bccac129cad8
Status: Downloaded newer image for debian:latest
8.7
```

Donde los elementos del comando son los siguientes:

- **docker** es el comando de acceso como cliente al servicio de **Docker**.
- **run** es la acción que vamos a ejecutar con el cliente, en este caso significa ejecutar un nuevo contenedor.
- Opciones **-ti**:
 - **-t** (*pseudo TTY*): indica iniciar el contenedor con la posibilidad de acceder al terminal del mismo.
 - **-i**: indica iniciar el contenedor en modo interactivo, la entrada de texto será posible al trabajar con el contenedor. Por defecto la entrada de texto (STDIN) no es posible.

- *debian* es la distribución con la que queremos ejecutar nuestro contenedor.
- *cat /etc/debian_version* es el comando que se ejecutará dentro del contenedor. Este comando sirve para visualizar la versión de la distribución en sistemas basados en *Debian* (por ejemplo *Ubuntu*).

Al ejecutar este comando, se producirán las tareas siguientes:

- El cliente *docker* intentará conectarse al servicio de *Docker* utilizando sockets. Si el usuario que ejecuta el cliente no tiene permiso para acceder, surgirá un error. El usuario *root* y los usuarios dentro del grupo *docker* son los únicos que pueden acceder al servicio de *Docker*.
- Una vez que la conexión a través de sockets se establece, comprobará si la imagen llamada *debian* ha sido descargada previamente. En caso contrario, como es nuestra primera vez, lo descargará del repositorio oficial.
- Una vez descargada y alojada en el directorio apropiado, ejecutará el contenedor con un nombre aleatorio.
- Una vez ejecutado el contenedor, dentro de él ejecutará el comando que le hemos indicado (*cat /etc/debian_version*) y como este comando termina su ejecución al visualizar la versión de la distribución, el contenedor se detendrá.

Es posible listar los contenedores que se están ejecutando actualmente, es decir, que tiene un proceso que no se ha detenido, utilizando el comando:

```
# docker ps
```

En el caso de querer listar los contenedores que están detenidos añadiremos la opción *-a*.

```
# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0e9470b0c6f5	debian	"cat /etc/debian_v..."	2 minutos ago	Exited (0) 2 minutos ago		sleepy_banach

Si queremos solo visualizar el último creado sin importar su estado (ejecutándose, detenido o error) indicaremos la opción *-l*.

Este comando nos devolverá la siguiente información:

- Identificador del contenedor corto de 12 caracteres. El identificador real contiene 64 caracteres.
- Imagen en la que se ha basado el contenedor.
- El comando que se ha ejecutado. En el caso no especificado se ejecutará el comando por defecto de la imagen, en el caso de las distribuciones será /bin/sh o /bin/bash. En el caso de imágenes específicas para aplicaciones, el comando de entrada será el ejecutable de la aplicación en primer plano para que no se detenga el contenedor.
- Momento de su creación. Indicado en meses, semanas, días o en formato horario (horas, minutos y/o segundos)
- El estado del contenedor:
 - Si se está ejecutando, incluye el tiempo que lleva ejecutándose (puede ser diferente al tiempo de cuando fue creado, ya que los contenedores se pueden detener e iniciar tantas veces como fuese necesario).
 - Si fue detenido, incluye el estado de salida (0 indica que finalizó correctamente u otro número en caso de error) o si se produjo algún error (incluyendo el mensaje del mismo). Además incluirá el tiempo que estuvo ejecutándose.
 - Puede estar vacío en el caso de los contenedores que fueron creados pero no ejecutados.
- Los puertos que se redirigen desde el servidor al contenedor.
- El nombre del contenedor. Este nombre, si no es especificado con las opciones, será autogenerado.

Para ver los registros del contenedor, es decir, la salida de la aplicación siempre registre su contenido a la salida estándar (STDOUT) podemos utilizar el siguiente comando:

- `docker logs contenedor`

Podemos especificar el contenedor al cual queremos acceder a sus registros de tres distintas maneras:

- Con el *identificador* corto de 16 caracteres.
- Con el *identificador* largo de 64 caracteres.
- Con el nombre del *contenedor*.

Las opciones que podemos indicar para la acción `logs` se muestran en la siguiente tabla:

Ejemplo:

```
# docker logs 0e9470b0c6f5  
8.7
```

INSPECCIONAR UN CONTENEDOR

Cuando un contenedor es ejecutado, el servidor Docker realiza diferentes tareas para ponerlo en funcionamiento:

- Genera un *identificador* único y un nombre si no es especificado.
- Descarga, si no existe, la *imagen* indicada para ejecutar el contenedor.
- Crea los ficheros o los sistemas de ficheros necesarios para el *contenedor*.
- Asocia una dirección IP para el *contenedor* y redirige, si es necesario, los puertos desde el servidor al *contenedor*.
- Asocia los límites de recursos.

Utilizando el comando:

- `docker inspect contenedor`

Igual que con la opción `logs` podemos especificar el contenedor a través del identificador (corto o largo) o el nombre.

Ejemplo:

```
# docker inspect 0e9470b0c6f5
{
  {
    "Id": "0e9470b0c6f58568ec0fc323f760403d5a01110892b75857ef17353bf638e219",
    "Created": "2017-01-30T20:43:12.710236071Z",
    "Path": "cat",
    "Args": [
      "/etc/debian_version"
    ],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2017-01-30T20:43:13.091441004Z",
      "FinishedAt": "2017-01-30T20:43:13.150994637Z"
    },
    [omitido]
  }
}
```

Entraremos en más detalle en la información que nos muestra `inspect` y cómo filtrar esta salida.

CONTENEDOR EN MODO INTERACTIVO

Hasta ahora hemos visto solo como ejecutar una aplicación y visualizar su salida, ya sea después de ejecutarse o a la través de la acción `logs`. Sin embargo, en muchos casos también querremos ejecutar un contenedor y trabajar interactivamente. Para que un contenedor no se detenga al ejecutarse, debemos indicarle un comando con el que podamos trabajar de manera interactiva.

En las imágenes de distribuciones, como pueden ser `debian` o `centos`, su comando por defecto es el intérprete de órdenes `/bin/bash`. Ejecutando el siguiente comando:

- `docker run -ti debian`

Nos aparecerá la consola de nuestro contenedor. Una vez ejecutado el comando veremos que nuestro prompt (indicador que la consola está lista para introducir comandos) ha cambiado.

```
servidor:~# docker run -ti debian
root@6e603710d02b:/#
```

Vemos que el *hostname* del contenedor ha sido autogenerado y veremos que coincide con el nombre del mismo. En este *prompt* podremos ejecutar los comandos que deseemos dentro del contenedor. Hasta que cerramos la sesión (*logout* o pulsando *Control-D*) el contenedor seguirá ejecutándose y todo comando que introduzcamos será ejecutado dentro del contenedor.

En el caso de querer salir del contenedor sin detenerlo tendremos que presionar una combinación de teclas: *Control-P* y a continuación *Control-Q*. Veremos que el *prompt* de nuestro servidor vuelve a aparecer. Ejecutando el siguiente comando vemos el estado del último contenedor creado.

# docker ps -l						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6e603710d02b	debian	"/bin/bash"	About a minute ago	Up About a minute		zen_edison

Como indicábamos anteriormente podemos utilizar la acción logs para observar las tareas que hemos realizado y su salida.

CONTENEDOR EN SEGUNDO PLANO

Hasta ahora hemos visto dos maneras de ejecutar un contenedor:

- Lanzando un comando y esperando su salida y la finalización del contenedor.
- En modo interactivo accediendo a la consola.

Sin embargo, Docker nos proporciona otra variante, que es la más utilizada: ejecutar un contenedor en segundo plano mientras ejecuta una aplicación en primer plano. Es decir, nosotros ejecutamos un contenedor que se estará ejecutando incluso si nosotros cerramos la sesión y no tenemos que acceder de modo interactivo para ello. La aplicación dentro del contenedor se ejecutará en primer plano y mientras esto suceda el contenedor se mantendrá ejecutándose.

Por ejemplo, podremos ejecutar un contenedor que dentro lanza un servidor web. Mientras el proceso del servidor web esté en ejecución, el contenedor estará en funcionamiento.

Para ello utilizaremos el siguiente comando:

- `docker run -dti imagen comando`

Donde la opción `-d` (del inglés *detach*) indica que se ejecutará el segundo plano. Como ya hemos indicado, podremos acceder a los registros a través de la acción `logs`.

Un ejemplo simple de un contenedor que ejecutará una tarea infinitamente (hasta que lo detengamos) es el siguiente:

```
# docker run -dti debian /bin/bash -c "while true; do date; sleep 5; done"
87c3b5bb26bfbe204cc44e4ed3fdbf61b6a2daa0fdb52a32bd02c3498514d94e
```

Este comando realizará:

- La ejecución de un contenedor utilizando la imagen *debian*.
- El contenedor estará en segundo plano, el comando devolverá el identificador largo del mismo.
- El comando indicado es un bucle infinito que mostrará la fecha y la hora cada 5 segundos.
- Hasta que se detenga el contenedor, la tarea se ejecutará.

Podemos utilizar:

```
# docker logs -f 87c3b5bb26bfbe204cc44e4ed3fdbf61b6a2daa0fdb52a32bd02c3498514d94e
Mon Jan 30 21:04:57 UTC 2017
Mon Jan 30 21:05:02 UTC 2017
Mon Jan 30 21:05:07 UTC 2017
Mon Jan 30 21:05:12 UTC 2017
```

Se observa el progreso de la tarea, cada 5 segundos veremos que se actualizan los registros generados por el contenedor. A través de la acción `ps` podemos listar los contenedores que están en ejecución actualmente y el comando de entrada.

DETENER CONTENEDOR

En la sección anterior hemos visto cómo lanzar un contenedor en segundo plano con una aplicación en funcionamiento. En el caso de querer pararse el contenedor, utilizaremos el siguiente comando:

- `docker stop contenedor`

Con la acción `stop`, Docker esperará 10 segundos a que la aplicación se detenga. En el caso de no pararse en ese periodo, el contenedor será detenido a la fuerza. Podemos indicar el número de segundos a esperar utilizando la opción `-t`.

```
# docker stop 87c3b5bb26bfbe204cc44e4ed3fdbf61b6a2daa0fdb52a32bd02c3498514d94e
87c3b5bb26bfbe204cc44e4ed3fdbf61b6a2daa0fdb52a32bd02c3498514d94e
```

A través de la acción `ps` es posible observar el código de salida.

# docker ps -l						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
87c3b5bb26bf	debian	"/bin/bash -c 'whi.."	5 minutes ago	Exited (137) About a minute		admiring_mahavira

En el caso de que el código de salida es mayor que 128, significa que ha salido inesperadamente. En este caso $128+9 = 137$ indica que fue detenido a la fuerza (`SIGKILL` tiene el código 9).

Es importante tener en cuenta que, una vez se detiene un contenedor, su dirección IP queda libre para el uso de cualquier otro.

INICIAR CONTENEDOR

Un contenedor que ha sido creado previamente y detenido posteriormente, ya sea manualmente o porque la aplicación que estaba ejecutándose ha fallado o ha terminado su tarea, nosotros podemos volver a lanzar el mismo contenedor otra vez.

Este contenedor mantendrá las mismas propiedades (nombre de contenedor, identificador, cambios realizados dentro, etc.).

Para ello utilizaremos el siguiente comando:

- `docker start contenedor`

Una vez iniciado, obtendrá una nueva dirección IP de la lista de direcciones disponibles. Un contenedor que ha sido inicializado con un comando no podrá arrancarse con un comando distinto. Para realizar esta acción, es necesario convertir el contenedor a una imagen e iniciar un nuevo contenedor usando la misma. Explicaremos este caso cuando tratemos la creación de *imágenes*.

```
# docker start 87c3b5bb26bf
87c3b5bb26bf
# docker ps -l
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS     NAMES
87c3b5bb26bf        debian      "/bin/bash -c 'whi..."   11 minutes ago   Up 30 seconds           admiring_mahavira
```

EXPONER PUERTOS

Cuando nosotros ejecutemos un contenedor que contenga una aplicación que debe ser accesible externamente, como podría ser un servidor web, nosotros por defecto solo podremos acceder a la misma desde el servidor de Docker a la dirección privada del contenedor.

Además, un contenedor que esté en la misma red de otro contenedor, podrá acceder también a la dirección privada donde la aplicación se está ejecutando. Un ejemplo común es un contenedor que está ejecutando un servidor web y la aplicación que sirve se conecta a la base de datos que está alojada en otro contenedor.

Un ejemplo utilizando la imagen oficial del servidor web Apache es el siguiente:

- `docker run -dti httpd`

La imagen `httpd` es proporcionada por el repositorio oficial de Docker al igual que las imágenes `debian` o `centos`. Este contenedor ejecutará el servidor web Apache y escuchará en el puerto 80 local.

```
# docker run -dti httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
5040bd298390: Already exists
63408554ba61: Pull complete
4dfe9e0af52: Pull complete
84871cd4d3da: Pull complete
b376204f8aa6: Pull complete
2a8d452c2c14: Pull complete
```

```
7cfa9c4a8891: Pull complete
Digest: sha256:1407eb0941b010035f86dfe8b7339c4dd4153e2f7b855ccc67dc0494e9f2756c
Status: Downloaded newer image for httpd:latest
a77ac589f3fb3e0354a792f9cde2c1088aece5728a532339e0ff45c2ad9d3736
```

A través de la conocida acción `inspect` podremos obtener qué dirección IP está utilizando.

```
# docker inspect a77ac589f3fb | grep -m1 "IPAddress"
"IPAddress": "172.17.0.4",
```

Utilizando esa IP, es posible acceder desde el servidor Docker o desde otro contenedor al contenedor recién creado. En el caso de que la dirección IP del contenedor fuese 172.17.0.4, podemos acceder al servidor web utilizando el comando desde nuestro servidor web o desde otro contenedor.

```
# curl http://172.17.0.4
<html><body><h1>It works!</h1></body></html>
```

Y nos mostrará el contenido por defecto del servidor web que se está ejecutando dentro del contenedor. Podremos utilizar un navegador para acceder al contenido de forma más accesible (utilizando Chromium, Chrome, Firefox o similar).

No obstante, en la mayoría de los casos, queremos acceder a las aplicaciones del contenedor desde nuestra red y no solo desde el servidor que está ejecutando el servicio de Docker.

Para ello, Docker nos proporciona lo denominado “exposición de puertos” que consiste en reservar un puerto del servidor de Docker con el fin de redirigir las peticiones a un puerto específico del contenedor. Si nosotros queremos ejecutar un servidor web dentro de un contenedor y queremos que sea accesible a través de la IP del servidor de Docker, podremos redirigir las comunicaciones al puerto 80 del servidor al puerto 80 del contenedor.

Para realizar esta tarea, durante la creación del contenedor indicaremos a través de la opciones `-p` o `-P` el puerto o los puertos que deseemos redirigir.

La diferencia entre las dos opciones se muestra a continuación:

- `-P (--publish-all)`: selecciona un puerto libre aleatorio del servidor donde se van a escuchar las peticiones a redirigir.

- **-p {--publish-value}**: debemos especificar un puerto manualmente donde deseamos realizar la escucha. Si ese puerto está en uso, la creación del contenedor fallará.

En el caso de utilizar la opción **-P**, Docker revisará los puertos que se han configurado en la imagen para ser expuestos y por cada uno de ellos generará uno aleatorio. Podemos utilizar la acción **ps** para obtener dichos puertos.

- **docker run -dtiP httpd**

En el siguiente ejemplo observamos que el puerto autogenerado ha sido 32768:

```
# docker run -dtiP httpd
4eac3254a41dc7e499d9b6f2559462b7da66ebccf3b920c40aa9d5aa558b02c
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
4eac3254a41d httpd "httpd-foreground" 5 seconds ago Up 4 seconds 0.0.0.0:32768->80/tcp unruuf-
file_euler
```

Este puerto está a la escucha en el servidor Docker, accediendo a él desde el propio servidor accederemos al servidor web alojado en el *contenedor*.

```
# curl http://localhost:32768
<html><body><h1>It works!</h1></body></html>
```

O desde otro ordenador de nuestra red podremos acceder usando un navegador a la dirección <http://ip.servidor:32768>

En el caso de utilizar la opción **-p** debemos elegir manualmente cuál es puerto del servidor que utilizaremos para la redirección, la sintaxis es la siguiente:

- **docker run -dti -p redirección contenedor [comando]**

La *redirección* tiene tres formatos distintos posibles:

Formato	Descripción
<i>puerto_servidor:puerto_contenedor</i>	Redirige el puerto <i>puerto_servidor</i> al <i>puerto_contenedor</i> de la IP con la que contenedor se está ejecutando. La escucha en el servidor será en todas las interfaces de red e IPs (0.0.0.0).

Formato	Descripción
<i>IP:puerto_servidor:puerto_contenedor</i>	Al igual que el formato anterior, se redirigirá el puerto_servidor al puerto_contenedor pero esta vez solo se mantendrá la escucha en la IP especificada.
<i>IP::puerto_contenedor</i>	En este caso, el puerto_contenedor será escuchado en el servidor Docker en la IP especificada. Es igual que el formato anterior pero ahorramos indicar puerto_servidor si coincide con el puerto del contenedor.

Ejemplo:

```
# docker run -dti -p 80:80 httpd
4e4c8f4a592804546993ed9a686be23d6cb011138b9d446fb8ce76a37d85fd5e
```

En este caso, si en nuestro servidor Docker no tenemos ninguna aplicación a la escucha del puerto 80 (TCP), Docker redirigirá las peticiones HTTP al contenedor. Podremos acceder a través del servidor usando curl:

```
# curl http://localhost
<html><body><h1>It works!</h1></body></html>
```

O a través de nuestra red con un navegador web usando <http://ip.servidor/>

En cualquier de los dos casos, usando -p o -P podremos listar los puertos redirigidos utilizando la acción port especificándole el contenedor del que queremos obtener la información:

```
# docker port 4eac3254a41d
80/tcp -> 0.0.0.0:32768
```

IPTABLES

La redirección de la comunicación entre el servidor y el contenedor se realiza utilizando iptables (permite actuar como firewall y cómo redireccionar de tráfico a través de reglas). Podemos listar las reglas con el siguiente comando:

```
# iptables -t nat -L DOCKER -v -n
Chain DOCKER (2 references)
pkts bytes target  prot opt in  out  source        destination
  0   0 RETURN  all  --  docker0 *    0.0.0.0/0      0.0.0.0/0
  0   0 DNAT   tcp  --  docker0 *    0.0.0.0/0      0.0.0.0/0      tcp dpt:32768 to:172.17.0.5:80
  0   0 DNAT   tcp  --  docker0 *    0.0.0.0/0      0.0.0.0/0      tcp dpt:80 to:172.17.0.6:80
```

CONTENEDORES

Crear contenedores sin ejecutarlos

En el capítulo anterior hemos visto cómo se ejecutan los contenedores: primero son creados y una vez finalizada dicha creación, son ejecutados.

En algunas situaciones nos resulta útil poder crear un contenedor en avance y ejecutarlo en un momento determinado. Para ello utilizaremos la acción create y las mismas opciones que run.

- `docker create -[opciones] imagen [comando]`

La acción create realizará las siguientes tareas:

- Descargará, si no ha sido descargada anteriormente o si hay una nueva versión disponible, la *imagen* especificada.
- Creará los ficheros o sistemas de ficheros requeridos.
- Establecerá las propiedades del contenedor.
- Devolverá el identificador del contenedor recién creado.

Algunas de las razones para crear un contenedor sin la necesidad de ejecutarlo se listan a continuación:

- Reducir el tiempo de parada de una aplicación en caso de actualización: con create podemos crear un contenedor con la nueva versión:

- Utilizar stop al contenedor que ejecuta la aplicación antigua.
- Utilizar start del nuevo *contenedor* creado.
- Probar las opciones y atributos que se establecerán a un contenedor futuro sin tener que ejecutarlo en dicho momento.
- Iniciar volúmenes (lo detallaremos en el capítulo de almacenamiento) con anterioridad.

Ejemplo:

```
# docker create -ti -p 8080:80 httpd
fdff6c0d9a99706f86266d6e51349021f283eb900ffc26297213f6144bbd195c
# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS              NAMES
fdff6c0d9a99        httpd              "httpd-foreground"   2 seconds ago    Created
                                         ports
                                         names
                                         mus-
                                         ing_babbage
```

ACCIONES BÁSICAS SOBRE CONTENEDORES

Hasta ahora hemos visto las tareas más básicas que se pueden realizar con los contenedores. En este capítulo vamos a profundizar más en esas tareas y ver todas las opciones que están disponibles antes de continuar con tareas más complicadas.

Para listar y obtener la ayuda de las acciones disponibles de Docker podemos ejecutar el siguiente comando:

- `docker --help`

Además de las acciones disponibles, nos mostrará las opciones que podemos especificarle al cliente `docker` para cada una de esas acciones.

- Opción `-v` (`--version`) nos mostrará la versión del cliente.
- Con la opción `-H` podemos conectarnos a un servidor de Docker diferente al de `localhost` (en el capítulo del configuración del servidor, veremos cómo habilitar las comunicaciones externas).
- Con la opción `-l` (`--log-level`) podemos indicar el nivel de registro que queremos mostrar: `debug`, `info`, `warn`, `error` o `fatal`. Por defecto el nivel es `info`.

```
# docker --version
Docker version 1.13.0, build 49bf474
```

Podemos acceder a las opciones que podemos indicar para cada una de las acciones a través de la sintaxis:

- **docker acción --help**

Las acciones utilizadas en las tareas cotidianas con contenedores se listan a continuación con las opciones posibles (todas las acciones, a no ser que se especifique lo contrario, tienen un solo argumento, que es el identificador o el nombre del contenedor a tratar).

start [opciones] contenedor

Iniciar un contenedor detenido o creado previamente. Las opciones son:

- **-a (–attach)**: para acceder a la consola (si es posible) del contenedor.
- **-i (interactive)**: para poder trabajar dentro del mismo.

Iniciar un contenedor detenido y acceder a su consola:

```
servidor:~# docker start -ai 6e603710d02b
root@6e603710d02b:/#
```

attach [opciones] contenedor

Accede a la consola, siempre que sea posible, de un contenedor que se está ejecutando en segundo plano. Las opciones son:

- **--detach-keys teclas**: indica la combinación de teclas a usar para salir de la consola.
- **--no-stdin**: acceder en modo visor, no se permitirá ejecutar comandos dentro.

Acceder a la consola de un contenedor, para salir pulsar Contrl-P y Contrl-Q:

```
servidor:~# docker attach 6e603710d02b
root@6e603710d02b:/#
```

stop [opciones] contenedor

Detiene un contenedor que está ejecutándose. La opción posible es:

- -t (--time): podemos indicar el número de segundos a esperar antes de que el contenedor sea parado a la fuerza.

Detener un contenedor previamente iniciado:

```
# docker stop 6e603710d02b
6e603710d02b
```

restart [opciones] contenedor

Detiene e inicia un contenedor que estaba en ejecución. La opción posible es:

- -t (--time): podemos indicar el número de segundos a esperar antes de que el contenedor sea parado a la fuerza.

Reiniciar un contenedor previamente iniciado:

```
# docker restart 6e603710d02b
6e603710d02b
```

rename contenedor nuevo_nombre

Renombra un contenedor. Este comando no tiene opciones, en el siguiente ejemplo renombramos un nombre autogenerado a uno específico.

```
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
fdff6c0d9a99 httpd "httpd-foreground" 17 minutes ago Created
musing_babbage
# docker rename fdff6c0d9a99 servidor_web
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
fdff6c0d9a99 httpd "httpd-foreground" 17 minutes ago Created
servidor_web
```

ps [opciones]

Lista los contenedores. Las opciones posibles son:

- -a, --all: muestra todos los contenedores independientemente de su estado.

- **-f/—filter filtro:** filtra la salida a través del *filtro* especificado.
- **--format formato:** formatea la salida con el *formato especificado*.
- **-n/-last num:** muestra los últimos *contenedores creados* limitándolo al número especificado.
- **-l/—latest:** muestra el último *contenedor creado*.
- **--no-trunc:** no limita el número de caracteres a mostrar.
- **-q/—quiet:** solo muestra los identificadores.
- **-s/—size:** muestra el tamaño total.

Filtrar el listado buscando por nombre.

```
# docker ps -f name=servidor_web -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
faff6c0d9a99 httpd "httpd-foreground" 20 minutes ago Created
servidor_web
```

logs [opciones] contenedor

Muestra el registro generado dentro de un contenedor. Las opciones son:

- **--details:** muestra información más detallada.
- **-f/—follow:** espera por nuevos registros.
- **--since fecha:** muestra registros desde la fecha especificada.
- **--tail número:** el número de líneas a mostrar o la palabra *all* para todas.
- **-t/—timestamps:** muestra la fecha y la hora.

Para mostrar los registros de un contenedor incluyendo la fecha.

```
# docker logs --timestamps 4e4c8f4a5928
2017-01-20T21:23:06.563Z001282 AH00558: http: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.6. Set the 'ServerName' directive globally to suppress this message
```

```

2017-01-30T21:23:06.572488601Z AH00551: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.6. Set the 'ServerName' directive globally to suppress this message
2017-01-30T21:23:06.577390720Z [Mon Jan 30 22:33:06.574151 2017] [mpm_event:notice] [pid 2:tid 140121985746816] AH00488: Apache/2.4.25 (Ubuntu) configured -- resuming normal operations
2017-01-30T21:23:06.577390720Z [Mon Jan 30 22:33:06.574251 2017] [core:notice] [pid 2:tid 140121985746816] AH00094: Command line: "httpd -D FOREGROUND"
2017-01-30T21:23:42.2503814027 172.17.0.1 - - [30/Jan/2017:21:23:42 +0000] "GET / HTTP/2.1" 200 45

```

pause/unpause contenedor

Pausa o reanuda los procesos dentro del contenedor especificado. Este comando no tiene opciones. Ejemplo:

```

# docker pause servidor_web
servidor_web
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f0ff6c0d9a99 httpd "httpd-foreground" 37 minutes ago Up 5 secs [Paused] 0.0.0.0:8080->80/tcp servidor_web
# docker unpause servidor_web
servidor_web
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f0ff6c0d9a99 httpd "httpd-foreground" 37 minutes ago Up 22 seconds 0.0.0.0:8080->80/tcp servidor_web

```

kill [opciones] contenedor

Detiene a la fuerza un contenedor. La opción posible es:

- **-s (--signal)** se puede indicar la señal que se mandará, por defecto "SIGKILL".

Ejemplo de detención a la fuerza:

```

# docker kill servidor_web
servidor_web
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f0ff6c0d9a99 httpd "httpd-foreground" 42 minutes ago Exited (137) 1 second ago servidor_web

```

top contenedor [opciones *ps*]

Muestra la lista de procesos con información detallada dentro de un contenedor. Se pueden incluir después del *identificador* o del *nombre del contenedor* las opciones del comando *ps*:

# docker top servidor_web								
USER	PID	PPID	C	STIME	TTY	TIME	CMD	
root	8196	8173	2	22:25	pts/0	00:00:00	httpd -DDETACHED	
daemon	8225	8196	0	22:25	pts/0	00:00:00	httpd -DDETACHED	
daemon	8226	8196	0	22:25	pts/0	00:00:00	httpd -DDETACHED	
# docker top servidor_web -u								
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START TIME COMMAND
root	8196	0.0	0.4	7720K	4604	pts/0	S+.	22:25 0:00 httpd -
DETACHED								

rm [opciones] contenedor

Elimina un contenedor. Por defecto, el contenedor debe estar detenido antes de ser eliminado. Las opciones posibles son:

- **-f/—force**: fuerza la eliminación de un contenedor en ejecución.
- **-v/—volumes**: elimina los volúmenes asociados al contenedor.

Eliminar un contenedor previamente creado:

```
# docker rm d1ae49d695e3
d1ae49d695e3
```

exec [opciones] contenedor comando

Ejecuta un comando dentro de un contenedor en funcionamiento. Las opciones son:

- **-d/—detach**: ejecuta el comando en segundo plano.
- **--detach-keys teclas**: la combinación de teclas para salir de la consola.
- **-i/—interactive**: modo interactivo para poder introducir los comandos.
- **-t/—tty**: adjunta un terminal virtual.

- **-u/-user usuario:** especifica el usuario para ejecutar el comando.

Ejecutar comando dentro del contenedor, en este caso grep:

```
# docker exec servidor_web grep '^DocumentRoot /usr/local/apache2/conf/httpd.conf'
DocumentRoot "/usr/local/apache2/htdocs"
```

Acceder a la consola de un contenedor (similar a attach pero sin riesgo de detenerlo al salir de él).

```
servidor:~# docker exec -ti servidor_web /bin/bash
root@fdff6c0d9a99:/usr/local/apache2# ps -ef
UID      PID  PPID  C STIME TTY      TIME CMD
root      1     0 22:25 ?    00:00:00 httpd -DFOREGROUND
daemon    9     1 22:25 ?    00:00:00 httpd -DFOREGROUND
daemon   10     1 22:25 ?    00:00:00 httpd -DFOREGROUND
daemon   11     1 22:25 ?    00:00:00 httpd -DFOREGROUND
root    173     0 22:42 ?    00:00:00 /bin/bash
root    185     0 22:42 ?    00:00:00 /bin/bash
root    189   185 22:42 ?    00:00:00 ps -ef
```

run/create [opciones] imagen comando

La acción run crea y lanza un contenedor, create solo crea el contenedor. Las opciones son:

- **-a/-attach valor:** adjunta STDIN, STDOUT o STDERR.
- **-d, --detach:** ejecuta el contenedor en segundo plano.
- **--dns servidores:** establece los servidores DNS.
- **--dns-search dominios:** establece los dominios de búsqueda DNS.
- **-h/-hostname nombre:** especifica el nombre de host para el contenedor.
- **-i/-interactive:** permite el modo interactivo para introducir comandos.
- **--mac-address dirección:** especifica la dirección MAC.
- **--name nombre:** asigna nombre al contenedor.

- **-p/-publish puerto:** publica el puerto especificado.
- **-P/-publish-all:** publica todos los puertos a puertos aleatorios.
- **--read-only:** ejecuta el contenedor en modo solo lectura.
- **--restart política:** la política de reinicio a establecer cuando el contenedor es detenido. Opciones:
 - **off:** no inicia el contenedor nunca. Solo manual.
 - **on-failure:** reinicia el contenedor en caso de error.
 - **always:** reinicia siempre que sea parado.
 - **unless-stopped:** reinicia a no ser que fuese parado con stop.
- **--rm:** automáticamente elimina el contenedor cuando es detenido.
- **-t, --tty:** adjunta una consola virtual.
- **-u/--user usuario:** el usuario para ejecutar los comandos en el contenedor.
- **-w/--workdir directorio:** el directorio de trabajo.

Ejecutar un contenedor con un nombre específico y definir el nombre de host.

```
# docker run --name servidor1 --hostname servidor1 debian uname -a
Linux servidor1 4.4.0-59-generic #80-Ubuntu SMP Fri Jan 6 17:47:47 UTC 2017 x86_64 GNU/Linux
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f9c8b9f84df4 debian "uname -a" 3 seconds ago Exited (0) 2 seconds ago
servidor1
```

COPIA DE SEGURIDAD

Es posible hacer una copia de seguridad de un contenedor, ya esté en ejecución o detenido. A través de la acción export, empaquetará el contenido y generará un fichero tar. Las sintaxis posibles son las siguientes:

- `docker export [-o fichero.salida.tar] contenedor`
- `docker export contenedor > contenedor.tar`

Ejemplo de creación de una copia de seguridad de un *contenedor*:

```
# docker export servidor1 > servidor1.tar
# ls -lh servidor1.tar
-rw-r--r-- 1 root root 123M Jan 31 16:29 servidor1.tar
```

RESTAURAR

No es posible restaurar un *contenedor* de forma automática, la restauración consiste en la creación de una *imagen* basada en el contenido de la copia de seguridad del *contenedor*. Es posible hacer la restauración en un servidor diferente al que ha sido realizada la copia. La sintaxis es la siguiente:

- `docker import [-m Mensaje importación] fichero nombre[:etiqueta]`

Ejemplo de importación de una copia de seguridad a una *imagen* y ejecutar un *contenedor* basado en ella.

```
# docker import servidor1.tar servidor1:20170131
sha256:f3cefd9ac1a10e1440dc1e8e5f4508b37cc0e918d21893365c6276638bad7a64
# docker run -ti servidor1:20170131 cat /etc/debian_version
8.?
```

INSPECCIONANDO CONTENEDORES

A través de la acción `Inspect` podemos inspeccionar distintos objetos de Docker que iremos viendo más adelante (*imágenes*, *redes*) pero en esta sección veremos la información que nos proporciona sobre un *contenedor*.

Elemento	Descripción	Ejemplo
Id	Identificador largo del contenedor.	c24d11528f754c3754da26c35[...]
Created	Fecha de creación del contenedor.	2016-12-11T18:22:36.861772177Z
Path	Comando de entrada en la ejecución del contenedor.	/bin/bash
Args	Argumentos del comando indicado anteriormente.	-c "ls"

Elemento	Descripción	Ejemplo
Image	Identificador de la imagen utilizada.	sha256:9a02494beff8d0d088ee[...]
ResolvConfPath	Ruta en el servidor Docker para los ficheros del contenedor:	"ResolvConfPath":
HostnamePath	/etc/resolv.conf	"/var/lib/docker/containers/c24d1152bf754c3754da28c35[...]/resolv.conf"
HostsPath	/etc/hostname	"HostnamePath":
LogPath	/etc/hosts	"/var/lib/docker/containers/c24d1152bf754c3754da28c35[...]/hostname"
	Y fichero logs.	"HostPath":
		"/var/lib/docker/containers/c24d1152bf754c3754da28c35[...]/hosts"
		"LogPath":
		"/var/lib/docker/containers/c24d1152bf754c3754da28c35[...]/c24d1152bf754c3754da28c3543ba8217409811fb56bd47b268a46ab865d729.json.log".
Name	Nombre del contenedor	/evil_terni
RestartCount	Veces que el contenedor fue reiniciado.	0
Driver	Driver usado para el sistema de ficheros.	aufs
HostConfig	Muestra configuraciones del servidor Docker, cuotas y límites.	<pre> "HostConfig": ["Binds": null, "ContainerIDFile": "", "LogConfig": { "Type": "json-file", "Config": {} }, "NetworkMode": "default", "PortBindings": {}, "RestartPolicy": { "Name": "no", "MaximumRetryCount": 0 }, "AutoRemove": false, "VolumeDriver": "", "VolumesFrom": null, "CapAdd": null, "CapDrop": null, "Ons": [], "OnsOptions": [], "OnsSearch": []] </pre>

Elemento	Descripción	Ejemplo
		<pre> "ExtraHosts": null, "GroupAdd": null, "IpcMode": "", "Cgroup": "", "Links": null, "DomSpecAdd": 0, "PubMode": "", "Privileged": false, "PublishAllPorts": false, "ReadonlyRootfs": false, "SecurityOpt": null, "UTSMode": "", "UsernsMode": "", "ShmSize": 67108864, "Runtime": "runc", "ConsoleSize": [0, 0], "Isolation": "", "CpuShares": 0, "Memory": 0, "NanoCpus": 0, "CgroupParent": "", "BlkioWeight": 0, "BlkioWeightDevice": null, "BlkioDeviceReadBps": null, "BlkioDeviceWriteBps": null, "BlkioDeviceReadIOPS": null, "BlkioDeviceWriteIOPS": null, "CpuPeriod": 0, "CpuQuota": 0, "CpuRealtimePeriod": 0, "CpuRealtimeRuntime": 0, "CpusetCpus": "", "CpusetMems": "", "Devices": [], "DiskQuota": 0, "KernelMemory": 0, "MemoryReservation": 0, </pre>

Elemento	Descripción	Ejemplo
		<pre>"MemorySwap": 0, "MemorySwappiness": -3, "OomKillDisable": false, "PidLimit": 0, "Ulimits": null, "CpuCount": 0, "CpuPercent": 0, "IOMaximumIops": 0, "IOMaximumBandwidth": 0 },</pre>
GraphDriver	El driver para el sistema de ficheros usado.	<pre>"GraphDriver": { "Name": "overlay2", "Data": { "LowerDir": "/var/lib/docker/overlay2/33ece775[...]-", "InitDiff": "/var/lib/docker/overlay2/c7b728[...]/diff", "MergedDir": "/var/lib/docker/overlay2/33ece775[...]/merged", "UpperDir": "/var/lib/docker/overlay2/33ece775[...]/diff", "WorkDir": "/var/lib/docker/overlay2/33ece775[...]/work" } },</pre>
Mount	Puntos de montaje especificados desde el servidor Docker al contenedor.	<pre>"Mounts": [{ "Type": "bind", "Source": "/web", "Destination": "/var/www/html", "Mode": "", "RW": true, "Propagation": "" }],</pre>
Config	Incluye configuraciones del contenedor, por ejemplo su <code>hostname</code> , el nombre de la <code>imagen</code> utilizada y los parámetros especificados (<code>tty</code> , modo interactivo).	<pre>"Config": { "Hostname": "c24d31528f75", "Domainname": "", "User": "", "AttachStdin": true,</pre>

Elemento	Descripción	Ejemplo
		<pre> "AttachStdout": true, "AttachStderr": true, "Tty": true, "OpenStdin": true, "StdinOnce": true, "Env": null, "Cmd": ["/bin/bash"], "Image": "debian", "Volumes": null, "WorkingDir": "", "Entrypoint": null, "OnBuild": null, "Labels": {}] </pre>
NetworkSettings	Todas las configuraciones y opciones relacionadas con la red del contenedor.	<pre> "NetworkSettings": { "Bridge": "", "ContainerID": "bc91828c287b1b4164[...]", "HairpinMode": false, "LinkLocalIPv6Address": "", "LinkLocalIPv6PrefixLen": 0, "Ports": {}, "SandboxKey": "/var/run/docker/netns/bc91828c287b", "SecondaryIPAddresses": null, "SecondaryIPv6Addresses": null, "EndpointID": "5d8a5ee922dc1aebb8[...]", "Gateway": "172.17.0.1", "GlobalIPv6Address": "", "GlobalIPv6PrefixLen": 0, "IPAddress": "172.17.0.2", "IPPrefixLen": 16, "IPv6Gateway": "", "MacAddress": "02:42:ac:11:00:02", "Networks": { "bridge": { "IPAMConfig": null, "Links": null, "Aliases": null } } } </pre>

Elemento	Descripción	Ejemplo
		<pre>"NetworkID": "b0da4662072eb86[...]", "EndpointID": "5d8a5ee92dc1ae8[...]", "Gateway": "172.17.0.1", "IPAddress": "172.17.0.2", "IPPrefixLen": 16, "IPv6Gateway": "", "GlobalIPv6Address": "", "GlobalIPv6PrefixLen": 0, "MacAddress": "02:42:ac:11:00:02"</pre>

Podemos filtrar la salida de la acción `inspect` a través de la opción `-f` (`--format`), algunos ejemplos se muestran a continuación:

Mostrar el hostname de un contenedor.
<code># docker inspect -f "{{.Config.Hostname }}" servidor_web fdf76c0d9a99</code>
Obtener la dirección IP de un contenedor.
<code># docker inspect -f "{{.NetworkSettings.IPAddress }}" servidor_web 172.17.0.7</code>
Obtener la dirección MAC de un contenedor.
<code># docker inspect -f "{{.NetworkSettings.MacAddress }}" servidor_web 02:42:ac:11:00:07</code>
Mostrar el estado de un contenedor.
<code># docker inspect -f "{{.State.Status }}" servidor_web running</code>

Como observamos en estos ejemplos, podemos acceder a distintos elementos utilizando el formato: `{} .Categoria.Elemento {}`

OBteniendo estadísticas de uso

Docker nos proporciona la posibilidad de obtener estadísticas de uso del servidor a través del cliente `docker`. Para ello utilizaremos la siguiente sintaxis:

- **docker stats [-a o --all] [--no-stream] [lista de contenedores]**

Donde las opciones tienen el siguiente significado:

- **-a (-all):** muestra todos los contenedores si no especificamos una lista de ellos. Por defecto solo muestra los contenedores que se encuentran en ejecución, con esta opción también se muestran los detenidos.
- **--no-stream:** no muestra la información en tiempo real, solo lo muestra una vez y el cliente se cierra.

Ejemplo (docker stats -a)

CONTAINER	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
PIDS					
fbf9f22d0f6f	--	-- / --	--	-- / --	--
c24d11528f75	0.00%	0 B / 0 B	0.00%	690 B / 648 B	0 B / 0 B
f371644ef900	--	-- / --	--	-- / --	--
64bf24ed0fdf	--	-- / --	--	-- / --	--
c39422d76aa4	2.24%	0 B / 0 B	0.00%	2.446 MB / 43.45 MB	1.936 GB / 2.137 GB

OBTENIENDO EVENTOS DESDE EL SERVIDOR

Docker nos ofrece una opción de visualizar todos los eventos que se han generado o ver en tiempo real los eventos que se van generando. La sintaxis es la siguiente:

- **docker events [opciones]**

Donde las opciones disponibles son las siguientes:

- **-f/-filter filtro:** filtra la salida con el filtro especificado.
- **--since fecha:** muestra registros desde la fecha especificada.
- **--until fecha:** muestra registros hasta la fecha especificada.

El formato para --since y --until puede ser de dos tipos:

- Relativo: 10m (10 minutos), 1h (1 hora), 1d (1 día)

- Entero: en formato RFC3399: 2016-01-30T15:20:59, 2016-01-30T07:00, 2016-01-30 ó 2016-01-30T15:20:59.999999999,

En el caso de no especificar ninguna opción de filtrado, `docker events` esperará a nuevos eventos que surjan y los mostrará en pantalla. Los filtros actualmente aceptados para la opción `--filter` son los siguientes:

- `container (container=nombre o identificador)`: filtra por contenedor.
- `event (event=acción)`: filtra por un tipo de evento.
- `image (image=etiqueta o identificador)`: filtra por imagen.
- `label (label=clave o label=clave=valor)`: filtra por una etiqueta.
- `type (type=container o image o volume o network o daemon)`: filtra por el tipo de objeto que genera el evento (contenedor, imagen, volumen, red o servicio).
- `volume (volume=nombre o identificador)`: filtra por un volumen.
- `network (network=nombre o identificador)`: filtra por una red interna de Docker.
- `daemon (daemon=nombre o identificador)`: filtra por un servicio.

En el caso de especificar dos veces un mismo filtro (`--filter container=contenedor1 --filter=contenedor2`) el filtro será de tipo OR, es decir, mostrará eventos del `contenedor1` o del `contenedor2`.

En el caso de especificar distintos filtros (`--filter container=contenedor1 --filter event=restart`) el filtro será de tipo AND, es decir, mostrará los eventos que coincidan con ambos filtros. En este caso, filtrará por el contenedor `contenedor1` y el evento `restart`.

En el siguiente ejemplo se visualizan los eventos mostrados al reiniciar un contenedor utilizando la acción `restart`.

```
2016-12-13T [...] container kill c24d11528 [...] (image=debian, name=evil_fermi, signal=15)
2016-12-13T [...] container die c24d11528 [...] (exitCode=0, image=debian, name=evil_fermi)
2016-12-13T [...] network disconnect bcda [...] (container=c24d11528 [...], name=bridge, type=bridge)
2016-12-13T [...] container stop c24d11528 [...] (image=debian, name=evil_fermi)
2016-12-13T [...] network connect bcda [...] (container=c24d11528 [...], name=bridge, type=bridge)
2016-12-13T [...] container start c24d11528 [...] (image=debian, name=evil_fermi)
2016-12-13T [...] container restart c24d11528 [...] (image=debian, name=evil_fermi)
```

Los eventos que un *contenedor* puede generar son los siguientes:

attach, commit, copy, create, destroy, detach, die, exec_create, exec_detach, exec_start, export, health_status, kill, oom, pause, rename, resize, restart, start, stop, top, unpause, update.

Los eventos que una *imagen* puede generar son los siguientes:

delete, import, load, pull, push, save, tag, untag

En el caso de *volumenes*:

create, mount, unmount, destroy

En el caso de redes internas de Docker:

create, connect, disconnect, destroy

El servicio (*daemon*) de Docker solo genera un evento: *reload*. Este evento solo es generado en el caso de releer la configuración de Docker.

5 IMÁGENES

Introducción

Hasta hora hemos visto cómo trabajar con contenedores. Desde la creación, pasando por las tareas más habituales, hasta la eliminación. Pero como ya sabemos, un contenedor depende de una imagen para su funcionamiento. Cuando creamos un contenedor se basará en el contenido de una imagen, una vez realizadas diferentes tareas y modificaciones en el contenedor, esos cambios solo se verán reflejados en el mismo y no en la imagen original.

En el caso de querer guardar los cambios generados en un contenedor para utilizarlo en otro, será necesario convertirlo en una imagen. En este capítulo veremos las tareas que podremos realizar con contenedores, desde la descarga de una imagen sin necesidad de crear un contenedor hasta la creación de nuestra propia imagen basándonos en los cambios realizados en un contenedor.

LISTAR IMÁGENES

Para listar las imágenes que disponemos actualmente en nuestro servidor Docker, utilizaremos el siguiente comando:

- `docker images [opciones] [nombre imagen]`

Las opciones para la acción `images` son las siguientes:

- `-a/--all`: muestra todas las imágenes.
- `-f, --filter filtro`: filtra la salida con el filtro especificado.

- ***-format formato***: formatea la salida con el formato especificado.
- ***--no-trunc***: no trunca el tamaño de las columnas.
- ***-q/-quiet***: muestra solo los identificadores.

Un ejemplo de un listado de *ímágenes* mostradas en un servidor Docker:

# docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	debian	7	26f8900cfb86	17 hours ago	85.26 MB
	debian	latest	9a02f494bef8	10 months ago	125.1 MB
	mongo	latest	81af31244bed	13 months ago	261.5 MB
	centos	centos6	6f95786cfb8	18 months ago	203.1 MB
	centos	latest	67898f1f8ef2	18 months ago	172.2 MB
	ubuntu	latest	b77d6a00f049	19 months ago	188.3 MB

La información que se visualiza en columnas es la siguiente:

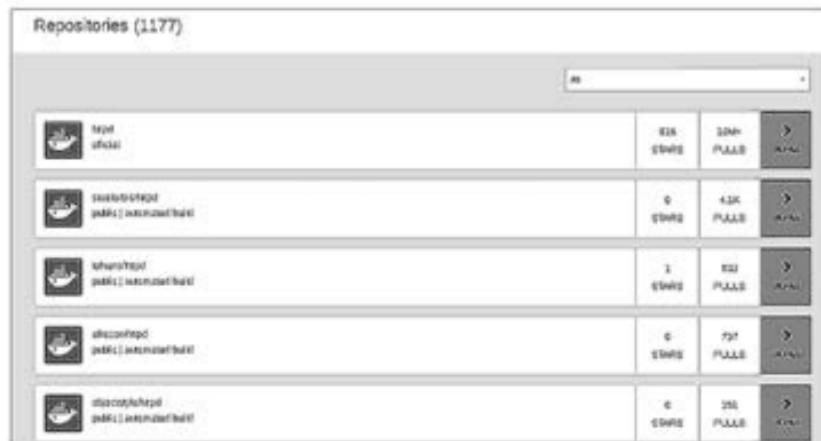
- Nombre de la imagen (y su repositorio, veremos en este capítulo lo que significa).
- La etiqueta (o versión). En este caso "latest" significa la última versión disponible. Podemos tener diferentes versiones de una distribución o aplicación.
- Identificador de la imagen. Por defecto muestra el identificador corto.
- El tiempo que ha pasado desde la creación de la imagen en el sistema.
- El tamaño de la imagen descomprimida en el sistema.

Las imágenes en los repositorios (oficiales o propios como veremos más adelante) estarán comprimidas. Una vez descargada se descomprimirá en el sistema para su uso.

BUSCAR IMÁGENES EN EL REPOSITORIO OFICIAL

Docker nos ofrece un repositorio oficial (**Docker Hub**) donde están disponibles imágenes oficiales para distribuciones (debian, ubuntu, centos, etc.) y para aplicaciones (httpd, nginx, mariadb, etc.). Además, en este repositorio, los usuarios tienen la opción de subir sus propias imágenes que han generado.

La dirección para el repositorio oficial de Docker es la siguiente: <https://hub.docker.com/>. En esta dirección es posible buscar imágenes, ver cómo utilizar la imagen, distinguir las diferentes versiones (*tags*) y diversa información de gran utilidad.



El cliente de Docker también nos permite hacer búsquedas en el repositorio oficial sin tener que abrir un navegador para ello. Para ello utilizaremos el comando:

- **`docker search`** *búsqueda [opciones]*

Las opciones disponibles son las siguientes:

- **`-f, --filter`** *filtro*: filtra la salida con el filtro especificado.
- **`--limit`** *número*: limita el número de resultados (por defecto 25).
- **`--no-trunc`**: no trunca el tamaño de las columnas.

Un ejemplo buscando *httpd* y limitando la salida a 5 resultados.

```
# docker search --limit 5 httpd
NAME          DESCRIPTION           STARS      OFFICIAL AUTOMATED
```

httpd	<i>The Apache HTTP Server Project</i>	819	[OK]
centos/httpd		10	[OK]
lolhens/httpd	<i>Apache httpd 2 Server</i>	1	[OK]
efrecon/httpd	<i>A micro-sized httpd. Start serving files i...</i>	0	[OK]
objectstyle/httpd	<i>ObjectStyle HTTPD Image</i>	0	[OK]

Donde observamos:

- Nombre de la Imagen. Si no es una imagen oficial, mostrará el usuario que la ha creado.
- Una breve descripción.
- Las valoraciones (STARS) de la imagen.
- Si es oficial o no.
- Si es una imagen automatizada.

DESCARGAR IMAGEN

Para descargar o actualizar una imagen desde el repositorio oficial, utilizaremos el siguiente comando:

- **docker pull *Imagen[:tag]***

Esta acción realizará los siguientes pasos:

- Comprobar si en el repositorio existe dicha *Imagen*.
- Comprobar si la imagen y la versión (la etiqueta especificada con *tag* o *latest* si no se especifica) existen actualmente en nuestro servidor.
- En el caso de que exista una versión más reciente, actualizará la imagen y la versión indicada (o *latest* si no se especifica ninguna). En el caso de que no existiera previamente en el servidor, descargará la imagen nueva.
- Descomprimirá la imagen recién descargada.

- Incluirá la imagen a la lista de imágenes accesibles y le asociará un identificador corto y uno largo de 64 caracteres basados en el algoritmo *sha256*. Este identificador largo es importante para verificar la identidad de la imagen.

Obtener Imagen



Un ejemplo de la descarga de la *imagen* oficial para el servidor web *nginx*:

```
# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
5040bd298390: Already exists
333547110842: Pull complete
4df1e44d2a7a: Pull complete
Digest: sha256:f2d384a6ca8ada733df555be3edc427f2e5f285ebf468aae940843de8cf74645
Status: Downloaded newer image for nginx:latest
```

HISTORIAL DE UNA IMAGEN

Una imagen contiene distintas capas de las modificaciones que se han ido realizando a una imagen *base*. Las imágenes creadas entre la imagen *base* y la imagen *definitiva* se llaman imágenes *intermedias*. Un ejemplo es una imagen con una distribución que contiene los elementos básicos. Posteriormente, en esa imagen se han instalado algunas utilidades, una aplicación y sus librerías. Para finalizar, se establece un software de monitorización para dicha aplicación. En este caso dispondremos:

- Una imagen *base*: contiene lo básico del sistema operativo.
- Dos imágenes *intermedias*:

- La primera después de la instalación de las utilidades.
- La segunda después de la instalación de la aplicación y sus librerías.
- Una **Imagen final**: es la unión del resto de las imágenes.

Nosotros podemos ver el historial de una imagen utilizando el comando:

- **docker history imagen**

Y nos listará el contenedor *base* y los contenedores *intermedios* que han llevado a tener la imagen *final* indicada.

Un ejemplo del historial de la imagen *nginx*:

# docker history nginx	IMAGE	CREATED	CREATED BY	SIZE	COM-
	MENT				
cc1b61406712	6 days ago	/bin/sh -c #(nop) CMD ["nginx" "-g" "daemon..."		0 B	
<missing>	6 days ago	/bin/sh -c #(nop) EXPOSE 443/tcp 80/tcp		0 B	
<missing>	6 days ago	/bin/sh -c ln -sf /dev/stdout /var/log/nginx/error.log		22 B	
<missing>	6 days ago	/bin/sh -c apt-key adv --keyserver hkp://p.../etc/apt/trusted.gpg.d/docker-archive-keyring.gpg		58.8 MB	
<missing>	6 days ago	/bin/sh -c #(nop) ENV NGINX_VERSION=1.11.10-0~stretch1		0 B	
<missing>	13 days ago	/bin/sh -c #(nop) MAINTAINER NGINX Dockerfile <nginx-devel@lists.debian.org>		0 B	
<missing>	2 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]		0 B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:89ecb642d662ee7... /etc/nginx		123 MB	

COPIA DE SEGURIDAD

Es posible hacer una copia de seguridad de una *imagen*. A través de la acción save, empaquetará el contenido y generará un fichero *tar*. Las sintaxis posibles son las siguientes:

- **docker save [-o fichero.salida.tar] imagen**
- **docker save imagen > contenedor.tar**

Ejemplo de creación de una copia de seguridad de un *contenedor*:

```
# docker save -o nginx.latest.tar nginx
# ls -lh nginx.latest.tar
-rw----- 1 root root 182M Jan 31 16:50 nginx.latest.tar
```

RESTAURAR

Es posible restaurar una copia de seguridad previamente creada, ya sea en el mismo servidor o en el otro. La sintaxis es la siguiente:

- `docker import [-i] fichero.tar]`
- `docker import < fichero.tar`

Un ejemplo de importación de una copia de seguridad de una *imagen*:

```
# docker load -i nginx.latest.tar
Loaded image: nginx:latest
```

ELIMINAR UNA IMAGEN

Cuando explicamos las tareas más comunes a realizar sobre un contenedor, veímos que los contenedores podían eliminarse. En el caso de las imágenes también podemos eliminarlas de nuestro servidor. El único requisito es que no exista ningún contenedor que haga referencia a la *imagen* que deseamos eliminar.

El comando es el siguiente:

- `docker rmi imagen [opciones]`

Las opciones son las siguientes:

- `-f`/`--force`: fuerza la eliminación de una imagen.
- `--no-prune`: no elimina imágenes padre sin etiquetar.

En el caso de querer eliminar una versión específica y no la última (*latest*) debemos especificar a la imagen la versión que deseamos eliminar:

- `docker rmi imagen:version [opciones]`

Ejemplo:

```
# docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:f2d384a6ca8ada733df555be3edc427f2e5f285ebf48aoe940843de8cf74645
Deleted: sha256:cc1b614067128cd2f5cdafb258b0a4dd25760f14562bcc516c13f760c3b79c4
Deleted: sha256:a92b2e17cab27c7e920fb7d683b402e0ab99b4658faecd0d889cd98007193f
Deleted: sha256:db2175a8ce095b094fc38584f8fd47298de0720ae8b29125e9befb8e56912
```

CONVERTIR UN CONTENEDOR EN UNA IMAGEN

En una gran variedad de casos existirán cambios que hemos realizado en un contenedor que nos gustaría mantener. Es el caso de la instalación de una aplicación específica, la configuración de una aplicación o distintas configuraciones dentro del contenedor. Además, es habitual querer transferir esos cambios a otros sistemas Docker para que se puedan ejecutar en el contenedor con el mismo contenido.

La solución más fácil, sin tener que hacer una copia de seguridad y restaurar el contenedor, es convertir dicho contenedor en una imagen. Esta imagen contendrá todos los cambios realizados a la imagen base. Además, podemos definir el nombre de la imagen a nuestro gusto y crear versiones en el tiempo. Esto nos facilitará desplegar aplicaciones, mantenerlas y actualizarlas.

Imaginemos que tenemos un servidor Docker para los desarrollos. Proporcionamos un contenedor a los desarrolladores, donde desplegarán la aplicación que están desarrollando. Una vez probada, nos pide a nosotros (DevOps) que transfiramos este contenedor al servidor de pruebas (Testing) para que esta versión sea evaluada por el equipo de calidad (QA / Quality Analysis).

Una vez evaluada y aprobada en el entorno de pruebas (Testing), la aplicación está lista para ser desplegada en el servidor de carga (Load) para la simulación de usuarios reales. Una vez superada esta fase, es posible que la aplicación sea transferida a un entorno de pre-producción (Preprod) para finalmente en su ciclo de vida pasar a producción (Prod).

Como vemos en este ejemplo, cada entorno podría tener diferentes versiones de una aplicación, ya que los desarrolladores irán desplegando diferentes versiones para su comprobación y las distintas tareas de prueba.

Docker nos ofrece la posibilidad de convertir un contenedor a una imagen a través del comando:

- `docker commit contenedor usuario/imagen[:versión]`

Además, con las opciones `-a` para indicar el autor y `-m` para indicar la modificación realizada podemos mantener un buen control de versiones de nuestras propias imágenes. Aunque esta tarea no requiere que el contenedor esté detenido, se recomienda detener el contenedor después de las modificaciones realizadas antes de convertirlo en una imagen. Esto nos garantizará que todo está en orden y no hay procesos o ficheros abiertos.

Ejecutar un contenedor basado en la imagen debian

```
# docker run --name convertir -ti debian
9fe5d15f60fbab49c833f40d30cf885667ddee274861b8608a89af964944c106
```

Actualizar la lista de repositorios dentro del contenedor.

```
# docker exec -ti convertir apt-get -q update
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://deb.debian.org jessie InRelease
Get:2 http://security.debian.org jessie/updates/main amd64 Packages [436 kB]
Get:3 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:4 http://deb.debian.org jessie Release.gpg [2373 B]
Get:5 http://deb.debian.org jessie-updates/main amd64 Packages [17.6 kB]
Get:6 http://deb.debian.org jessie Release [148 kB]
Get:7 http://deb.debian.org jessie/main amd64 Packages [9049 kB]
Fetched 9861 kB in 5s (1874 kB/s)
Reading package lists...
```

Instalar el editor VIM

```
# docker exec -ti convertir apt-get install vim
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
  libgpm2 vim-common vim-runtime
Suggested packages:
  gpm ctags vim-doc vim-scripts
The following NEW packages will be installed:
  libgpm2 vim vim-common vim-runtime
[omitido]
Unpacking vim (2:7.4.488-7+deb8u1) ...
Setting up libgpm2:amd64 (1.20.4-6.1+b2) ...
Setting up vim-common (2:7.4.488-7+deb8u1) ...
Setting up vim-runtime (2:7.4.488-7+deb8u1) ...
[omitido]
```

Convertir contenedor en imagen

```
# docker commit -a "Alberto" -m "Instalar VIM" convertir debianvim
sha256:c881f4aa0ca3e0dd7b810c1516fe13f22728018fe1cb5eb3e901ce5070b2bf41
```

Una vez finalizada la tarea, podremos listar las imágenes de nuestro servidor Docker para comprobar nuestra imagen creada y su tamaño con el comando `docker images`.

docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debianvim	latest	c881f4aa0ca3	29 seconds ago	162 MB
httpd	latest	67a5f36fd844	2 weeks ago	176 MB
debian	latest	e5599115b6a6	2 weeks ago	123 MB

Además, utilizando `docker history` podremos listar las modificaciones que hemos realizado a la imagen base hasta llegar a la imagen que hemos creado.

docker history debianvim

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
c881f4aa0ca3	6 seconds ago	/bin/bash	39.1 MB	Instalar
VIM				
e5599115b6a6	2 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:89ecb642...	123 MB	

Observamos que el tamaño se ha ido modificando en cada capa que se ha añadido a la *imagen base*.

ETIQUETADO DE IMÁGENES

Observamos que con Docker podemos mantener un historial de versiones para las imágenes. Esto se realiza con etiquetas (tags). En esta sección vamos a ver la acción `tag` que nos permitirá modificar las etiquetas de una imagen.

La sintaxis es la siguiente:

- `docker tag Imagen Imagen:tag`

Resultará útil en el caso de que tengamos una aplicación en alguno de nuestros entornos. La versión que se está ejecutando utilizada la etiqueta por defecto `latest` y nosotros si queremos actualizar a una nueva versión, podemos añadir una etiqueta a la imagen actual (por ejemplo, `Imagen:0.8`) y descarga la nueva imagen desde el repositorio (ya sea el oficial o uno propio) con la etiqueta `latest`. En el caso de que la nueva versión no funcione correctamente, nosotros siempre podremos ejecutar un contenedor con la versión anterior (en nuestro ejemplo, `Imagen:0.8`) ahorrando tiempo en las tareas de restauración.

docker tag debianvim debianvim:0.1# docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debianvim	0.1	c881f4aa0ca3	3 minutes ago	162 MB

debianvim	latest	c881f40a0ca3	29 seconds ago	162 MB
httpd	latest	67a5f36fd844	2 weeks ago	176 MB
debian	latest	e5599115b6a6	2 weeks ago	123 MB

PUBLICAR UNA IMAGEN EN EL REPOSITORIO OFICIAL

En la web oficial del repositorio de Docker: <https://hub.docker.com/> es posible registrarse de forma gratuita para alojar las imágenes que hemos creado. Es importante tener en consideración que las imágenes creadas pueden ser accesibles públicamente, por lo cual no debemos alojar datos comprometidos.

Publicar Imagen



Una vez registrado con un usuario y una contraseña (llamado **Docker ID**), para almacenar las imágenes dentro del repositorio nosotros debemos nombrar a la imagen con el formato: *usuario/imagen[:version]*. Esta tarea puede realizarse con la acción tag.

Antes de realizar la acción de publicar la *imagen*, debemos autenticarnos con la cuenta previamente creada. Para ello utilizaremos la acción login.

```
# 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: oforte

Password:

Login Succeeded

Para publicar nuestra imagen, utilizaremos la siguiente sintaxis:

- `docker push usuario/imagen[:version]`

Esta acción también será utilizada cuando creamos nuestro repositorio privado.

```
# docker push oforte/debianvim
The push refers to a repository [docker.io/oforte/debianvim]
a2ae92ffcd29: Pushed
latest: digest: sha256:887c83ddcfe66c669759e49881cd4871d03de765cfe72bc90450046513d934ea
size: 529
```

Los datos de *login* en Linux se almacenarán en el siguiente fichero: `$HOME/.docker/config.json`. En el caso de Windows utilizará la siguiente ruta: `%HOMEP%\docker\config.json` (ruta utilizando *cmd*), en el caso de utilizar *PowerShell* la ruta será la siguiente: `$env:Home\docker\config.json`

Una vez almacenados los datos, es posible utilizar `docker push` sin necesidad de introducir nuestras credenciales de nuevo.

Además, es posible utilizar `docker logout` para cerrar la sesión en Docker Hub.

```
# docker logout
Removing login credentials for https://index.docker.io/v1/
```

REPOSITORIO LOCAL

Como es comprensible, algunas de las *imágenes* que creamos no deben ser accesibles públicamente. En otros casos, no tendremos acceso a *Internet* desde diferentes redes. La solución es desplegar un repositorio local donde alojaremos nuestras *imágenes* y serán accesibles a diferentes servidores Docker.

Para desplegar un repositorio local, ejecutaremos un contenedor usando una imagen oficial llamada *registry:2*. El puerto por defecto para la comunicación a un registro Docker es 5000/tcp. El comando para ejecutar esta imagen y escuchar a dicho puerto es el siguiente:

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
b7f33cc0b48e: Pull complete
46730e1e05c9: Pull complete
```

```
458210699647: Pull complete
0cf045fea0fd: Pull complete
b78a03aa98b7: Pull complete
Digest: sha256:0e40793ad06ac099ba63b5a8fae7a83288e64b50fe2ea0a2b59741de85fd3b97
Status: Downloaded newer image for registry:2
f638ea907783b33796b61e0301e66f6e5b8727788d65266fc7d5e9f3ed3cd05a
```

Una vez ejecutado nuestro repositorio local, podemos etiquetar las imágenes con el formato *servidor:5000/nombre* (por ejemplo: *localhost:5000/imagen*). Se muestran a continuación las acciones con commit y con push:

```
# docker commit ubuntu1 localhost:5000/ubuntuapache2
sha256:369365cc939653681d5dcc83c7b6bfca57acdcaf66192058deb5fa80beba31e
# docker push localhost:5000/ubuntuapache2
The push refers to a repository [localhost:5000/ubuntuapache2]
45219016209: Pushed
Seb5bd4c5014: Pushed
d195a7a18c70: Pushed
af605e724c5a: Pushed
59f161c3069d: Pushed
4f03495a4d7d: Pushed
latest: digest: sha256:6afbdaf573708ad4878ae6a95aa868e367944be740b1970833da12855c751115
size: 1569
```

Las *imágenes* publicadas con push se almacenarán dentro del contenedor que ejecuta el repositorio local. Se recomienda utilizar un *volumen* para almacenarlo externamente al *contenedor*, porque permitirá ser reutilizado en otro servidor en caso de problema con el actual y nos dará más flexibilidad a la hora de actualizar el repositorio. Para ello utilizaremos la opción *-v* para especificar un directorio del servidor Docker para enlazar con el directorio */var/lib/registry* dentro del contenedor que es donde se alojan las imágenes.

```
# docker stop registry
dockregistry
# docker rm registry
registry
# mkdir /registro
# docker run -d -p 5000:5000 --restart=always --name registry -v /registro:/var/lib/registry registry:2
2c9f1e3f0803927d0a78f4cb10dc5465a88e31c35154ce3e3e679ab699172b61
```

Una vez vuelto a ejecutar nuestro registro local, debemos volver a publicar la *imagen* previamente etiquetada, posteriormente podemos ver en el directorio del servidor Docker el contenido.

```
# docker push localhost:5000/ubuntuapache2
The push refers to a repository [localhost:5000/ubuntuapache2]
4521f9016209: Pushed
Seb5bd4c5014: Pushed
d195a7a18c70: Pushed
af605e724c5a: Pushed
59f161c3069d: Pushed
latest: digest: sha256:6afbda573708ad4878ae6a95aa868e367944be740b1970833da12855c751115
size: 1569
# ls -l /registro/docker/registry/v2/repositories/
total 4
drwxr-xr-x 5 root root 4096 Jan 23 22:03 ubuntuapache2
```

En este caso vemos que accedemos al registro local desde el propio servidor Docker donde está alojado. En el caso de querer acceder desde un servidor externo, debemos configurar el servicio Docker para permitir las conexiones a dicho repositorio. Para ello configuraremos (o crearemos) el fichero */etc/docker/daemon.json* con el siguiente contenido:

```
{ "insecure-registries": ["servidor:5000"] }
```

Después de reiniciar nuestro servicio (*systemctl restart docker*), podremos acceder a las imágenes de un servidor remoto.

```
# docker pull 192.168.56.104:5000/ubuntuapache2
Using default tag: latest
Error response from daemon: Get https://192.168.56.104:5000/v1/_ping: http: server gave HTTP re-
sponse to HTTPS client
# echo '{ "insecure-registries": ["192.168.56.104:5000"] }' > /etc/docker/daemon.json
# systemctl restart docker
# docker pull 192.168.56.104:5000/ubuntuapache2
Using default tag: latest
latest: Pulling from ubuntuapache2
e2d7e96004fd: Pull complete
1bacd3c8ccb1: Pull complete
869dsd3f92fb: Pull complete
f8a4e25b40ce: Pull complete
```

```
f40e1388890a: Pull complete
5a1df36d1695: Pull complete
Digest: sha256:6afbdaf573708ad4878ae6a95aa868e367944be740b1970833da12855c751115
Status: Downloaded newer image for 192.168.56.104:5000/ubuntuapache2:latest
```

Al habilitar la dirección IP o el nombre DNS a la lista de registros inseguros (*insecure-registries*) podemos descargar imágenes desde ese servidor que previamente fueron públicas. Además, desde el servidor que hemos configurado también podremos publicar imágenes utilizando push.

6

GENERAR IMÁGENES

Dockerfile

En el capítulo anterior habíamos visto cómo crear imágenes a partir del contenido de un contenedor con el que habíamos trabajado previamente. En este capítulo veremos cómo simplificar y automatizar este proceso.

Para ello **Docker** nos ofrece la posibilidad de indicar las instrucciones requeridas para crear una nueva imagen. Estas instrucciones se incluyen dentro de un fichero llamado **Dockerfile**. La sintaxis para las instrucciones son muy simples y podemos comenzar viendo un ejemplo:

```
FROM debian:latest
RUN apt-get update && apt-get install -y vim
CMD /bin/bash
```

Donde las expresiones tienen el siguiente significado:

- **FROM**: indica cuál es la imagen base para crear la nueva.
- **RUN**: los comandos a ejecutar dentro del contenedor con la imagen base para generar la imagen final.
- **CMD**: el comando por defecto a ejecutar cuando utilicemos esta imagen. Es el punto de entrada a la imagen si no especificamos otro comando con run o create.

Los pasos para crear una imagen a partir de un fichero **Dockerfile** son los siguientes:

- Crear un nuevo directorio que contenga el fichero *Dockerfile* y otros ficheros que fuesen necesarios dentro del contenedor.
- Crear el contenido para el fichero *Dockerfile*.
- Ejecutar *docker* con la acción **build**.

La sintaxis para la acción **build** es la siguiente:

```
docker build [opciones] directorio
```

Donde las opciones más comunes se muestran a continuación:

- **-t nombre[:etiqueta]**: crea una imagen con el nombre y la etiqueta especificados a partir de las instrucciones de *Dockerfile*. Es muy recomendable utilizar esta opción.
- **--no-cache**: por defecto, Docker guarda en memoria caché las acciones realizadas recientemente. Si nosotros ejecutamos *docker build* varias veces, Docker comprobará si el fichero *Dockerfile* contiene las mismas instrucciones y en caso afirmativo no generará una nueva imagen. Para generar siempre una nueva imagen sin hacer caso a la memoria caché utilizaremos esta opción.
- **--pull**: por defecto, Docker solo descargará la imagen especificada en la expresión *FROM* si no existe. Para forzar que descargue la nueva versión de la imagen utilizaremos esta opción.
- **--quiet**: por defecto se muestra todo el proceso de creación, los comandos ejecutados y su salida. Utilizando esta opción solo mostrará el identificador de la imagen creada.

Una vez que hemos creado un directorio y alojado nuestro fichero *Dockerfile* podemos crear nuestro primer contenedor utilizando la acción **build**.

```
# mkdir debianvim2
# cd debianvim2/
# nano Dockerfile <- Copiar contenido dentro. (Elegir el editor preferido).
# docker build -t debianvim2.
```

En la acción **build** especificamos el punto (.) como el directorio actual como argumento. Podriamos utilizar el directorio absoluto (por ejemplo,

/home/agonzalez/debianvim2/) o relativo (por ejemplo, ~/debianvim2/). La salida del comando, como ejemplo, se muestra a continuación:

```

Sending build context to Docker daemon 2.048 kB
Step 1 : FROM debian:latest
latest: Pulling from library/debian
Digest: sha256:f7062cf040f67f0c26ff46b3b44fe036c29468a7e69d8170f37c57f2eec1261b
Status: Image is up to date for debian:latest
    --> 19134a8202e7
Step 2 : RUN apt-get update && apt-get install -y vim
    --> Running in 85f37b639158
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
[omitiendo]
Fetched 9871 kB in 1s (5532 kB/s)
[omitiendo]
Unpacking vim (2:7.4.488-7+deb8u1) ...
Setting up libgpm2:amd64 (1.20.4-6.1+b2) ...
Setting up vim-common (2:7.4.488-7+deb8u1) ...
Setting up vim-runtime (2:7.4.488-7+deb8u1) ...
Processing /usr/share/vim-addons/doc
Setting up vim (2:7.4.488-7+deb8u1) ...
[omitiendo]
Processing triggers for libc-bin (2.19-18+deb8u6) ...
--> 0f1a46e7588b
Removing intermediate container 85f37b639158
Step 3 : CMD /bin/bash
    --> Running in 6c893c113b47
    --> cbb5a0feed34
Removing intermediate container 6c893c113b47
Successfully built cbb5a0feed34

```

Con la salida de la acción `build` podemos distinguir distintas características:

- La expresión `FROM` comprobará si existe la imagen y si es reciente.
- Para la expresión `RUN` se creará un *contenedor* intermedio para realizar las acciones. Si las acciones son completadas, se eliminarán del sistema (por defecto).
- Igual que con `RUN`, para la expresión `CMD` se creará un *contenedor* intermedio con el contenido después de las acciones ejecutadas para comprobar que el comando especificado es válido. También se eliminará después de la comprobación.

- Una vez finalizadas todas las acciones y eliminados los *contenedores intermedios*, se creará la imagen definitiva con el nombre y la etiqueta especificados.

Ahora podemos comprobar que nuestra imagen ha sido creada y ejecutar un contenedor basado en la misma.

```
# docker images debianvim2
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
debianvim2      latest   cbb5a0feed34   3 minutes ago   161.6 MB
# docker run -ti debianvim2 dpkg -l vim
[omitido]
ii  vim          2:7.4.488-7+ amd64    Vi iMproved - enhanced vi editor
```

Con el primer comando comprobamos que la imagen ha sido creada correctamente. Con el segundo comando ejecutamos un contenedor basado en la *Imagen* recién creada para comprobar que el software indicado en nuestro *Dockerfile* ha sido instalado.

Además, podemos utilizar `docker history` para ver las acciones ejecutadas para crear la imagen basada en el fichero *Dockerfile*

```
# docker history debianvim2
IMAGE      CREATED      CREATED BY      SIZE      COMMENT
cbb5a0feed34  6 minutes ago  /bin/sh -c #(nop) CMD ["/bin/sh -c *"] /bin/  0 B
0f1045e7588b  6 minutes ago  /bin/sh -c apt-get update && apt-get install  38.55 MB
19134a8202e7  4 weeks ago   /bin/sh -c #(nop) CMD ["/bin/bash"]       0 B
<missing>    4 weeks ago   /bin/sh -c #(nop) ADD file:1d214d2782eaccc743  123.1 MB
```

Como observamos, con *Dockerfile* podemos de una forma fácil y simple generar nuestras imágenes. Además de las siguientes ventajas:

- Es posible reutilizar las plantillas para nuevas imágenes.
- Su sintaxis es sencilla tanto para administradores como para desarrolladores.
- Utilizando un control de versiones (git, svn) es posible mantener versiones de estas plantillas.
- En Internet hay infinidad de repositorios que se basan en *Dockerfile* para ejecutar contenedores para multitud de aplicaciones. Simplemente clonando el repositorio estamos listos para utilizar `build` después de mínimas configuraciones.

A continuación veremos todas las expresiones que podemos utilizar dentro de un fichero *Dockerfile* y ejemplos de cada una de ellas.

FROM

Tiene que ser la primera línea del fichero *Dockerfile* (aunque puede aparecer varias veces a continuación para crear diversas imágenes). Sus posibles sintaxis se muestran a continuación:

- **FROM *Imagen*** -> Por defecto obtendrá la última versión (*latest*)
- **FROM *Imagen:etiqueta*** -> Especifica una versión distinta a *latest*.

Algunos ejemplos:

- **FROM debian**
- **FROM debian:latest** -> Equivalente a la anterior.
- **FROM debian:wheezy**

MAINTAINER

Con esta instrucción especificará el autor indicado a las imágenes generadas. Su sintaxis es la siguiente:

- **MAINTAINER <name>**

Ejemplo:

- **MAINTAINER Alberto Gonzalez <alberto@oforte.net>**

RUN

Ejecuta comandos sobre la imagen especificada para poder generar la definitiva deseada. Tiene dos sintaxis posibles:

- **RUN comando**
- **RUN ["comando", "parametro 1", "parametro 2"]**

Esta expresión puede aparecer tantas veces como sea necesaria dentro de nuestro fichero *Dockerfile* pero es recomendable intentar agrupar las acciones en el mínimo número de expresiones RUN. Esto es debido, como hemos visto, a que cada acción RUN genera un contenedor intermedio para realizar las acciones. En el caso de tener, por ejemplo, 5 veces la expresión, generará 5 contenedores que causarán que la creación sea mucho más lenta.

Tenemos dos opciones para agrupar comandos:

- Utilizar la expresión `&&` (AND) para agrupar comandos como hemos visto en nuestro *Dockerfile* de ejemplo: comando1 `&&` comando2 `&&` comando3 ...
- Crear un fichero *script* que contenga todas las acciones, copiarlo al contenedor y ejecutarlo. Veremos la instrucción para copiar ficheros al contenedor a continuación.

Algunos ejemplos de esta instrucción se muestran a continuación:

- `RUN apt-get install -y apache2`
- `RUN apt-get update && apt-get install -y apache2`
- `RUN ["apt-get", "install", "-y", "vim"]`

CMD

La expresión **CMD**, como se explicó previamente, sirve para proporcionar un comando por defecto al crear un contenedor basado en esa imagen. Esta expresión solo puede estar especificada una vez dentro del fichero *Dockerfile*. Las sintaxis posibles son las siguientes:

- `CMD ["ejecutable", "parametro 1", "parametro 2"]`
- `CMD comando parametro1 parametro2`
- `CMD ["parametro 1", "parametro2"]` -> Usado en conjunción con la expresión **ENTRYPOINT**.

Algunos ejemplos se muestran a continuación:

- `CMD cat /etc/debian_version`

- **CMD** [“cat”, “/etc/debian_version”]
- **CMD** [“/etc/debian_version”]

Es importante recordar que el comando especificado en **CMD** solo es para la entrada y no se ejecutará durante la creación de la *imagen* para que sus cambios sean incluidos.

EXPOSE

La expresión **EXPOSE** es utilizada solo para especificar los puertos que estarán a la escucha dentro del *contenedor* basado en esta *imagen*. En ningún caso se expondrá el puerto automáticamente, para ello tendremos que utilizar las opciones que conocemos para la acción **run**: -P para exponer todos los puertos especificados en la imagen o -p para especificar manualmente los puertos.

La sintaxis es la siguiente:

- **EXPOSE** puerto1 [puerto2 ..]

Y un ejemplo para una *imagen* que contiene un servidor web:

- **EXPOSE** 80 443 8080

Podemos listar los puertos expuestos por una *imagen* utilizando el siguiente comando:

```
# docker inspect -f "{{ .Config.ExposedPorts }}* debianapache2
map[443/tcp:{} 80/tcp:{} 8080/tcp:{}]
```

ADD

La expresión **ADD** sirve para copiar ficheros desde el servidor donde estamos creando la *imagen* a la misma. Es muy útil cuando creamos nuestras *imágenes* con aplicaciones, poder copiar los ficheros de configuración o ficheros que son necesarios para nuestra aplicación, como podría ser el contenido de una página web o la copia de seguridad de una base de datos a importar dentro del *contenedor*. Hay dos sintaxis posibles:

- **ADD** origen1 [origen2] destino

- ADD ["origen1","origen2.."],"destino" -> Sintaxis necesaria si los nombres contienen espacios.

Es posible copiar ficheros y directorios especificando su nombre o a través de expresiones. El destino especificado debe ser un directorio, especificado con una ruta absoluta o relativa. Todos los ficheros y directorios serán asignados al UID (identificador de usuario) y GID (identificador de grupo) del usuario root (0, cero). Todos los ficheros o directorios a copiar deben estar dentro del mismo directorio que el fichero Dockerfile. Algunos ejemplos:

- ADD 004-miweb.conf /etc/apache2/sites-available/
- ADD miweb/ /var/www/html/
- ADD *.conf /etc/apache2/

En el caso de copiar un directorio solo su contenido (ficheros y subdirectorios) se rá copiado al contenedor y no el directorio padre.

Una nota importante: si copiamos un fichero comprimido (tar, gzip, xz o bzip2) será extraído automáticamente y el fichero no se conservará dentro del contenedor, solo su contenido. En caso de no querer eso, debemos utilizar la expresión COPY.

Por ejemplo:

Crear fichero empaquetado.

```
# tar -cvvf motd.tar /etc/motd
```

Añadir a Dockerfile: ADD motd.tar /tmp/, y crear imagen.

```
# docker build -t debianvim2 .
```

Comprobar que el fichero ha sido automáticamente descomprimido.

```
# docker run -ti debianvim2 ls -l /tmp/etc/motd
-rw-r--r-- 1 root root 180 Dec 29 18:14 /tmp/etc/motd
```

Con la expresión ADD es posible especificar como origen un fichero alojado en una web, siempre que no requiera autenticación.

- ADD https://dl.bintray.com/consul/0.5.0_linux_amd64.zip /tmp/consul.zip

Los ficheros comprimidos copiados desde un origen web no serán descomprimidos.

COPY

Al igual que ADD copia ficheros dentro de la *Imagen* a crear. Es la expresión recomendada para copiar ficheros al contenedor, excepto que tengamos la necesidad de extraer ficheros comprimidos automáticamente. Su sintaxis es igual que la de ADD:

- **COPY origen1 [origen2] destino**
- **COPY ["origen1","origen2"],"destino"** -> Sintaxis necesaria si los nombres contienen espacios.

Los mismos ejemplos anteriormente indicados pero con COPY:

- **COPY 004-miweb.conf /etc/apache2/sites-availables/**
- **COPY miweb/ /var/www/html/**
- **COPY *.conf /etc/apache2/**

Otra diferencia con ADD, es que la expresión COPY no permite especificar como origen un fichero alojado en un servidor web.

ENTRYPOINT

Por defecto, las imágenes son creadas para que todos los comandos indicados se ejecuten utilizando */bin/sh*. A través de la instrucción ENTRYPOINT podemos cambiar el comportamiento por defecto. Las posibles sintaxis son las siguientes:

- **ENTRYPOINT ["comando", "parámetro 1", "parámetro 2"]** -> Recomendado.
- **ENTRYPOINT comando parametro1 parametro2**

Por ejemplo, si especificamos como punto de entrada el comando cat:

- **ENTRYPOINT ["cat"]**

Cuando ejecutemos un contenedor, el argumento que le indicamos al contenedor a ejecutar será un parámetro para el comando cat.

```
# docker run -ti debian cat /etc/debian_version
8.6
# docker run -ti debian cat /etc/hostname
59ec3cbc2543
# docker run -ti debian cat ls
cat: ls: No such file or directory
```

Como observamos en este ejemplo, al crear una imagen cambiando el ENTRYPPOINT por defecto a un comando distinto a `/bin/sh`, todos los argumentos pasados a la imagen serán tratados como argumentos al comando especificado.

Cuando veíamos la sintaxis de la expresión CMD había una alternativa que era solo especificar argumentos.

```
ENTRYPOINT ["cat"]
CMD ["/etc/debian_version"]
```

El comando que se ejecutará por defecto en este caso será:

```
cat /etc/debian_version
```

VOLUME

La expresión VOLUME crea un punto de montaje que podrá ser accesible por otros contenedores o enlazar un directorio del servidor al contenedor basado en esta imagen. Las posibles sintaxis son las siguientes:

- **VOLUME directorio1 directorio2**
- **VOLUME ["directorio1", "directorio2"]**

Por ejemplo, podemos definir un volumen y realizar acciones dentro:

```
RUN mkdir /datos && date > /datos/fecha.txt
VOLUME /datos
```

Al crear un contenedor basado en esta *imagen* podemos ver las acciones que se realizan:

```
# docker run --name debianvol -ti debianvol
Sat Jan 14 15:18:45 UTC 2017
# docker inspect -f "[{.Mounts}]" debianvol
[{"Id": "d1ad16077b57[...]", "Path": "/var/lib/docker/volumes/d1ad16077b57f[...]/_data", "Type": "volume"}, {"Id": "d1ad16077b57[...]", "Path": "/var/lib/docker/volumes/d1ad16077b57[...]/_data/fecha.txt", "Type": "file"}]
# cat /var/lib/docker/volumes/d1ad16077b57[...]/_data/fecha.txt
Sat Jan 14 15:18:45 UTC 2017
```

Vemos que se ha creado un volumen y todas las modificaciones hechas dentro del directorio especificado se guardarán dentro de él y es accesible a través de `/var/lib/docker/volumes/`.

USER

Por defecto, todos los comandos son ejecutados con el usuario `root` (uid 0). A través de esta expresión podemos cambiar el comportamiento. Su sintaxis:

- USER usuario
- USER uid

Un ejemplo de cada una de las sintaxis:

- USER nobody
- USER 65534

En este ejemplo podemos ver la salida del comando `id` al especificar un usuario distinto con `USER`.

```
FROM debian:latest
USER nobody
CMD id
# docker run -ti debian id
uid=0(root) gid=0(root) groups=0(root)
# docker build -q --no-cache --pull -t debianuser .
# docker run -ti debianuser
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
```

Observamos que en la imagen base los comandos se ejecutan con el usuario **root** pero en nuestra imagen al cambiar su comportamiento con **USER** se ejecuta con dicho usuario.

WORKDIR

El directorio por defecto de trabajo es la raíz (/). A través de esta expresión podemos especificar un directorio distinto para trabajar. Sintaxis:

- **WORKDIR /directorio**

El directorio de trabajo es utilizado por **RUN**, **CMD**, **ENTRYPOINT**, **ADD** y **COPY**. Se puede especificar tantas veces como sea necesario. Ejemplo:

Contenido fichero Dockerfile

```
FROM debian:latest
WORKDIR /tmp
RUN echo "test" > test.txt
WORKDIR /var/tmp
RUN echo "test2" > test2.txt
CMD ls -l /tmp/test.txt /var/tmp/test2.txt
```

Construir imagen.

```
# docker build -q --no-cache --pull -t debianworkdir .
# docker run -ti debianworkdir
-rw-r--r-- 1 root root 5 Jan 14 15:41 /tmp/test.txt
-rw-r--r-- 1 root root 6 Jan 14 15:41 /var/tmp/test2.txt
```

Como observamos en este ejemplo podemos especificar varias veces con **WORKDIR** antes de las acciones para cambiar el directorio en el que estamos trabajando. El último directorio especificado será el directorio por defecto de trabajo:

```
# docker run -ti debianworkdir pwd
/vor/tmp
```

SHELL

Como indicábamos, el punto de entrada para los contenedores Linux es el comando **/bin/sh** y para ser más precisos utiliza **/bin/sh -c** para ejecutar los comandos especificados en **CMD** o los comandos especificados en línea de comandos para la acción **run**. La sintaxis es la siguiente:

- SHELL ["comando", "argumento 1", "argumento 2"]

Por ejemplo si queremos añadir la opción -x a /bin/sh que hará que nos muestre qué comando se ejecuta:

Contenido fichero Dockerfile

```
FROM debian:latest
SHELL ['/bin/sh', "-x", "-c"]
CMD date
```

Construir imagen.

```
# docker build -q --no-cache --pull -t debianshell .
# docker run -ti debianshell
+ date
Sat Jan 14 16:13:38 UTC 2017
```

LABEL

La expresión LABEL sirve para añadir metadatos a una imagen. Estos metadatos son muy útiles para añadir información importante como puede ser la versión de una aplicación o una descripción detallada. La sintaxis es la siguiente:

- LABEL clave=valor [clave2=valor2 clave3=valor3]

Por ejemplo si queremos añadir un metadato incluyendo la versión y una descripción:

- LABEL version="1.0" description="Ejemplo de metadato como prueba"

Podemos acceder a los metadatos de una imagen a través de la acción inspect:

```
# docker inspect -f "{{.Config.Labels}}" debianlabel
map[description:Ejemplo de metadato como prueba version:1.0]
```

ENV

La expresión ENV nos permite establecer variables de entorno dentro del contenedor que use la imagen que estamos creando. Las sintaxis posibles son las siguientes:

- ENV clave valor

- ENV clave=valor [clave2=valor2 clave3=valor3]

Un ejemplo es cambiar la localización del sistema a través de la variable de entorno *LC_ALL*:

Contenido fichero Dockerfile

```
FROM debian:latest
RUN apt-get update && apt-get install -y locales locales-all
ENV LC_ALL="es_ES.UTF-8" LANG="es_ES.UTF-8"
CMD date
```

Construir la imagen.

```
# docker build -q --no-cache --pull -t debianenv .
# docker run -ti debianenv
sáb ene 14 16:51:39 UTC 2017
```

Observamos que se nos muestra la fecha en español.

ARG

Hasta ahora hemos visto que nuestro fichero *Dockerfile* contenía todas las instrucciones necesarias para crear la *imagen*. Pero en algunas situaciones deseamos tener una plantilla y poder, a través de *docker build*, pasarle argumentos para distintos propósitos. La sintaxis para **ARG** es la siguiente:

- ARG nombre
- ARG nombre=valorpordefecto

Una vez indicado en *Dockerfile* los argumentos podemos utilizar la opción *--build-arg* para la acción *build* para indicarle el valor que deseamos. En nuestra plantilla podremos utilizar el nombre del argumento precedido por un dólar (\$) para acceder a su valor. En el siguiente ejemplo indicamos un usuario por defecto para la imagen pero que puede ser sobreescrito:

Contenido fichero Dockerfile

```
FROM debian:latest
ARG user=nobody
USER $user
CMD id
```

Construir imagen.

```
# docker build -q --no-cache --pull -t debianarg .

# docker run -ti debianarg
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)

# docker build -q --build-arg user=bin --no-cache --pull -t debianarg .^
# docker run -ti debianarg
uid=2(bin) gid=2(bin) groups=2(bin)
```

Observamos, en este ejemplo, que en la primera creación de la imagen no especificamos ningún argumento y el comando *id* es ejecutado con el usuario por defecto especificado *nobody*. En la segunda creación utilizamos la opción **--build-arg** para especificar el usuario *bin*.

HEALTHCHECK

Indica el comando para comprobar si el estado del *contenedor* es correcto. La comprobación puede ser por ejemplo comprobar si un servidor web acepta conexiones al puerto 80. Se pueden especificar las siguientes opciones antes del comando:

- **--interval=duración:** indica el intervalo de comprobación, por defecto 30s.
- **--timeout=duración:** indica el tiempo de espera para la comprobación, por defecto también 30s.
- **--retries=intentos:** el número de intentos antes de declarar un contenedor como fallido.

```
HEALTHCHECK --interval=2m --timeout=3s CMD curl -f http://127.0.0.1/ || exit 1
```

Ejemplo

En el siguiente ejemplo creamos una imagen para ejecutar *Apache2*.

```
FROM debian:latest
MAINTAINER Alberto Gonzalez <alberto@oforte.net>
# Instalar apache2 y configurar locales
RUN apt-get update && apt-get install -y locales locales-all apache2
RUN locale-gen es_ES.UTF-8
```

```

EXPOSE 80
VOLUME /var/www
# Variables para Apache 2
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_LOG_DIR /var/log/apache
RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR
# Contenido web
COPY index.html /var/www/html/
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]

```

Los comentarios dentro de un fichero *Dockerfile* se especifican empezando la linea con el simbolo almohadilla (#). Los pasos para crear la imagen y ejecutar un contenedor que tendrá un servidor web que muestre el contenido deseado se detallan a continuación:

```

# echo "Hola desde el contenedor" > index.html
# docker build --no-cache --pull -t debianapache2 .
# docker run --name servweb -P debianapache2
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
fc5071472b38 debianapache2 "/usr/sbin/apache2 -D" 5 secs Up 4 seconds 0.0.0.0:32772->80/tcp
servweb
# curl http://localhost:32772
Hola desde el contenedor

```

Como observamos en este ejemplo, primero creamos el fichero *index.html* que se utilizará con la expresión **COPY**. Una vez creada la imagen ejecutamos un **contenedor** con la opción **-P** para exponer un puerto aleatorio en nuestro servidor, con la acción **ps** o la acción **ports** podemos obtener dicho puerto. Finalmente con la aplicación **curl** podemos acceder al servidor web y su contenido.

7 REDES

Introducción

En este capítulo hablaremos de las redes internas dentro de **Docker Engine**. Veremos las redes creadas automáticamente por el instalador, cómo crear nuestras propias redes y cómo conectar contenedores existentes a diferentes redes.

REDES PREDEFINIDAS

Durante la instalación de Docker Engine se crean tres redes, estas son:

- **bridge**: red por defecto. En Linux durante la instalación se crea una nueva interfaz de red virtual llamada *docker0*. Cuando ejecutamos un contenedor, por defecto, utilizará esta red a no ser que especifiquemos lo contrario.

```
# ifconfig docker0
docker0 Link encap:Ethernet HWaddr 02:42:77:aa:97:cd
          inet addr:172.17.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              RX packets:29657660 errors:0 dropped:0 overruns:0 frame:0
              TX packets:52071060 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:7686781007 (7.1 GiB) TX bytes:96895976119 (90.2 GiB)
```

- **none**: utilizando esta red, el contenedor no tendrá asociada ninguna interfaz de red, solo la de *loopback* (*lo*).
- **host**: utilizando esta red, el contenedor tendrá la misma configuración que el servidor *Docker Engine* donde se está ejecutando.

A excepción de la red **bridge**, no es necesario interaccionar con las otras dos. No es posible eliminar las redes predefinidas creadas por el instalador ya que son necesarias para el servicio. Es por ello que veremos cómo crear nuestras propias redes, dónde tendremos más flexibilidad y será posible eliminarlas si fuese necesario.

LISTAR REDES

A través de la acción **network** del cliente **docker**, podemos listar, crear, modificar o borrar redes. Además, podremos conectar un contenedor a una red existente. Para listar las redes existentes utilizaremos:

- **docker network ls [opciones]**
- **docker network list [opciones]**

Ejemplo:

# docker <u>network ls</u>			
NETWORK ID	NAME	DRIVER	SCOPE
608aa8899b15	bridge	bridge	local
a84d74e72982	host	host	local
0259eda18f84	none	null	local

Las columnas mostradas son las siguientes:

- Identificador de la red.
- Nombre de la red.
- Driver (manejador) de la red.
- Ámbito.

Opciones posibles para el listado:

- **-f/-filter filtro**: filtra la salida a través del filtro especificado.
- **-q**: muestra solo los identificadores.

CREAR RED

Cuando trabajamos con distintos entornos, como puede ser desarrollo, test, producción, vamos a necesitar separar los contenedores en diferentes redes. Docker Engine nos permite crear redes que permitirán aislar las comunicaciones entre ellos.

La sintaxis es la siguiente:

- `docker network create [opciones] nombre`

Las opciones que se pueden especificar son las siguientes:

- `-d/--driver valor`: el *driver* a utilizar, por defecto, es *bridge*.
- `--gateway valor`: la dirección IP para la *puerta de enlace*.
- `--internal`: restringe el acceso al exterior en esta red.
- `--ip-range valor`: el rango de red a utilizar.
- `--ipv6`: habilita IPv6.
- `--subnet valor`: define la máscara de red.

CREAR RED CON RANGO AUTOGENERADO

Si la opción `--subnet` no es especificada, Docker Engine buscará un rango libre y lo asignará a esta red. La red por defecto bridge es 172.17.0.1/16. La primera red personalizada creada (sin especificar `--subnet`) será con la subnet 172.18.0.1/16 y la siguiente 172.19.0.1/16 y así sucesivamente.

En el servidor se creará una interfaz de red virtual *bridge* con el formato *br-identificador*

```
# docker network create desarrollo
f4ee75e3fc389d0c0f746a91f8563af01d3bb17e31572157c9de1b71cff9172
# ip a show br-f4ee75e3fc38
65: br-f4ee75e3fc38: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:f7:99:1e:85 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 scope global br-f4ee75e3fc38
```

CREAR RED CON RANGO ESPECÍFICO

A través de la acción `network create` podemos especificar diferentes valores para personalizar nuestra red. En este ejemplo creamos una red con un rango específico, limitando las direcciones que se pueden utilizar y con una IP específica como puerta de enlace.

```
# docker network create --subnet 192.168.100.1/24 --ip-range 192.168.100.100/30 \
> --gateway 192.168.100.100 produccion
2bdffb2dace8a:94b6cabbb00165f36d453e9a06c53cf04fe078ca11461b9289d
# ip a show br-2bdffb2dace8a
66: br-2bdffb2dace8a: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:35:f7:4b:b9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.100/24 scope global br-2bdffb2dace8a
        valid_lft forever preferred_lft forever
```

CREAR RED SIN ACCESO AL EXTERIOR

En algunas situaciones, por motivos de seguridad, deseamos que nuestros contenedores no tengan acceso al exterior, especialmente Internet. Para ello especificaremos la opción `--internal`

```
# docker network create --internal interna
e6fc9044a4d3a691e75930b309b5c88f96c53a02d9f4cb7b57461273e7a7290c
```

ESPECIFICAR RED AL CREAR UN CONTENEDOR

Al crear un contenedor, ya sea a través de la acción `run` o con `create`, podemos especificar la red donde va a formar parte. Para ello utilizaremos la opción `--network`.

```
# docker run --network desarrollo -ti ubuntu ip a show eth0
73: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 scope global eth0
# docker run --network produccion -ti ubuntu ip a show eth0
75: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:c0:a8:64:65 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.101/24 scope global eth0
```

Además, podemos comprobar que en la red creada como *internal* no podemos acceder al exterior.

```
# docker run --network desarrollo -ti ubuntu ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=4.59 ms
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.597/4.597/4.597/0.000 ms
# docker run --network interna -ti ubuntu ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

INSPECCIONAR RED

Al igual que los contenedores con la acción inspect es posible inspeccionar una red utilizando en este caso la acción network inspect. La sintaxis es la siguiente:

- docker network inspect [-f formato] nombre o identificador.

Ejemplo:

```
# docker network inspect interna
"Config": [
    {
        "Subnet": "172.18.0.0/16",
        "Gateway": "172.18.0.1/16"
    }
],
"Internal": false,
"Containers": {
    "9d198cff34fed64f40c68035c4f98171728ed2fc5c0c745b35d8a5a60f7fe77": {
        "Name": "hungry_wright",
        "EndpointID": "26dbd08f0bbd5620919771990137fa9573eb1266269b27f00fb4eb282d37db11",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
    }
},
```

```

    "Options": {},
    "Labels": {}
}
]

```

Como observamos, si un contenedor o varios contenedores están en ejecución usando la red que inspeccionamos, mostrará por cada uno de ellos:

- Su nombre.
- Su identificadores.
- Su dirección MAC.
- Su dirección IP (IPv4 e IPv6)

Si queremos ver las redes a las que el *contenedor* está conectado, utilizaremos el comando:

```
# docker inspect -f "[.NetworkSettings.Networks] {{.Name}}={{.IPAMConfig}}={{.Links}}"
map[desarrollo:{{"IPAMConfig": {"IPv4Config": {"IP": "172.17.0.2", "SubnetMask": "255.255.0.0"}, "IPv6Config": {"IP": "2001:db8:2039:1::2", "SubnetMask": "ffff:ffff:ffff:ffff::0"}, "Gateway": "172.17.42.1"}, "Links": []}]}]
```

CONECTAR Y DESCONECTAR CONTENEDOR A/DE UNA RED

Hasta ahora hemos visto que en la fase de creación del *contenedor* es posible elegir la red en la que queremos incluirlo. Una vez creado, *Docker Engine* nos permite conectarlo a otra red, creando una nueva interfaz de red virtual dentro del *contenedor* utilizando la red especificada. Para ello utilizaremos la siguiente sintaxis:

- **`docker network connect [opciones] red contenedor`**

Donde *red* es el nombre o el identificador de una red ya existente y *contenedor* es el nombre o el identificador de un contenedor en ejecución. No es posible conectar un contenedor detenido a una red.

En el caso de no especificar ninguna opción, obtendrá una dirección IP libre dentro de la red especificada. Las opciones posibles para network connect son las siguientes:

- **`--ip dirección`**: especifica una dirección IP estática.
- **`--ip6 dirección`**: especifica una dirección IPv6 estática.

Como vemos en el siguiente ejemplo, lanzamos un contenedor dentro de una red previamente creada y una vez que está en ejecución lo conectamos a otra red.

Ejecutar contenedor conectado solo a una red.

```
# docker run --name dosredes --network desarrollo -ti debian  
4fff9b6edcd0c0dbd46cf5ca2c218349d46f7ed5996cd254cb054669640b43c
```

Listar redes del contenedor previamente creado.

```
# docker inspect -f "{{.NetworkSettings.Networks }}" dosredes  
map[desarrollo:{"MacAddress": "0xc820396000"}]
```

Conectar contenedor a una nueva red.

```
# docker network connect interna dosredes
```

Listar interfaces e IPs dentro del contenedor para comprobar ambas redes.

```
# docker exec -ti dosredes ip a  
[omitido]  
94: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff  
      inet 172.18.0.3/16 scope global eth0  
        valid_lft forever preferred_lft forever  
      inet6 fe80::42:acff:fe12:3/64 scope link  
        valid_lft forever preferred_lft forever  
96: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:ac:13:00:02 brd ff:ff:ff:ff:ff:ff  
      inet 172.19.0.2/16 scope global eth1  
        valid_lft forever preferred_lft forever  
      inet6 fe80::42:acff:fe13:2/64 scope link  
        valid_lft forever preferred_lft forever
```

En el caso de querer desconectar un contenedor de una red, utilizaremos la siguiente sintaxis:

- `docker network disconnect [-force] red contenedor`

Es posible desconectar el contenedor tanto de la red conectada utilizando `network connect` como de la red especificada durante la creación con `run` o `create`.

Listar las redes actuales del contenedor.

```
# docker inspect -f "{{.NetworkSettings.Networks}}" dasredes
map[desarrollo:0xc4203f20c0 interna:0xc4203f2180]
```

Desconectar contenedor de una red.

```
# docker network disconnect desarrollo dasredes
```

Comprobar la desconexión.

```
# docker inspect -f "{{.NetworkSettings.Networks}}" dasredes
map[interna:0xc4200e4900]
```

ELIMINAR UNA RED

Al igual que con otros elementos de *Docker Engine*, una vez creada una red es posible eliminarla. El único requisito es que esa red no esté utilizada por ninguno de los contenedores que actualmente están en ejecución. La sintaxis es la siguiente:

- `docker network rm red`
- `docker network remove red`

En el caso de que la red sea eliminada, el comando nos devolverá en la salida el nombre de la misma. En caso de que la red esté siendo utilizada por uno o varios contenedores, se mostrará el siguiente error:

```
# docker network remove interna
Error response from daemon: network interna has active endpoints
```

8

ALMACENAMIENTO

Introducción

Como hemos visto en los capítulos anteriores, los *contenedores* y las *imágenes* se almacenan en disco directamente. **Docker** posee una arquitectura de almacenamiento flexible que permite elegir entre diversos *drivers* dependiendo del sistema operativo y de las necesidades del entorno. Cada *driver* de almacenamiento de **Docker** se basa en sistema de ficheros Linux o un administrador de volúmenes (como *LVM*).

En este capítulo veremos las ventajas y desventajas de cada una de las opciones que nos proporciona **Docker** y veremos cómo configurar el servicio para utilizarlo. Es importante saber que cada *driver* administra los datos de forma particular y los *contenedores* e *imágenes* no son compatibles entre ellos. Es decir, si creamos *imágenes* y *contenedores* en un *driver* y decidimos utilizar otro, no serán accesibles en este último. Si deseamos utilizar los *contenedores* e *imágenes* de un *driver* a otro debemos hacer una copia de seguridad en el antiguo y restaurarlo en el nuevo.

La lista de *drivers* de almacenamiento para **Docker** se muestra a continuación:

Tecnología	Nombre <i>driver</i> de almacenamiento
OverlayFS	overlay or overlay2
AUFS	aufs
Btrfs	btrfs
Device Mapper	devicemapper
VFS	vfs
ZFS	zfs

Dependiendo del sistema operativo que utilicemos y su versión, Docker elegirá por defecto el *driver* más óptimo y estable para él mismo. Podemos comprobar el *driver* actual a través del siguiente comando ya conocido:

```
# docker info
Containers: 100
Running: 3
Paused: 0
Stopped: 97
Images: 409
Server Version: 1.12.5
Storage Driver: aufs
Root Dir: /var/lib/docker/aufs
Backing Filesystem: xfs
```

En este ejemplo, en un sistema *Debian* el *driver* elegido por defecto es *aufs*. Además, podemos observar que el directorio raíz es */var/lib/docker/aufs* y donde está alojado utiliza el sistema de ficheros *xfs*.

Driver de almacenamiento	Usado frecuentemente con los sistemas de ficheros:
overlay	ext4 y xfs
overlay2	ext4 y xfs
aufs	ext4 y xfs
btrfs	Sólo btrfs
devicemapper	direct-lvm
vfs	Sólo utilizado para pruebas.
zfs	Sólo zfs

Las ventajas y desventajas de cada uno de los *drivers* de almacenamiento se muestran a continuación:

Aufs	estable	buena para producción	bueno para la memoria	experiencia buena en Docker	alta actividad de escritura	entorno Prod.
Devicemapper (loop)	estable	en el kernel principal		experiencia buena en Docker	producción	rendimiento
Devicemapper (direct-lvm)	estable		bueno para producción	en el kernel principal	experiencia buena en Docker	entorno Prod.
Btrfs	en el kernel principal	alta actividad de escritura	contenedor inestable	crear pools		
Overlay	estable	buen uso de memoria	en el kernel principal	contenedor inestable	alto de pruebas	
ZFS native (ZFS)		entorno Prod.				
ZFS FUSE	estable	alto de pruebas	producción			

Clave:

Tiene un atributo	atributo
Si es bueno para el caso de uso	caso de uso
Si es malo para el caso de uso	caso de uso

Un resumen de los *drivers* recomendados por el soporte de **Docker** (*en su versión comercial*) en cada distribución se muestra a continuación:

Distribución	Driver de almacenamiento
RHEL 7.0, 7.1, 7.2	devicemapper
Ubuntu	aufs3
CentOS 7.1-1503, 7.2-1511	devicemapper
SLES	btrfs

Si deseamos utilizar un *driver* distinto en nuestro servidor **Docker** debemos configurar el servicio a través de la opción `--storage-driver=driver`.

A continuación detallaremos cada uno de los *drivers* para describir cómo funcionan, las ventajas y las desventajas.

Aufs

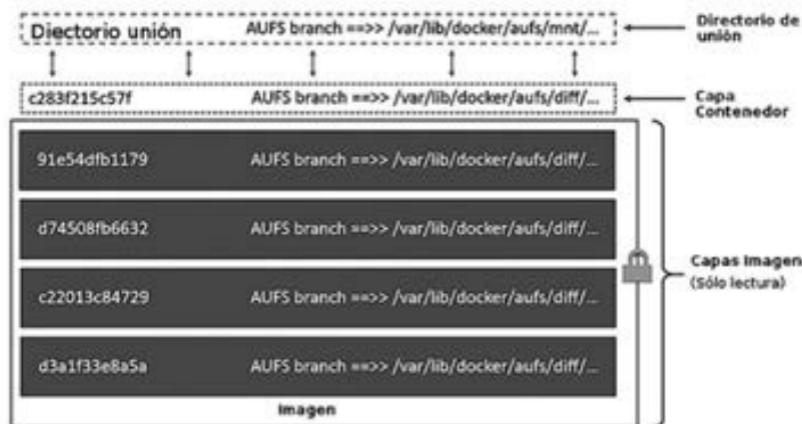
Aufs (*Advanced multi-layered Unification FileSystem*) fue el primer *driver* disponible en **Docker**. Por ello es parte importante de la historia de los *contenedores* y es uno de los *drivers* más estables y utilizado en entornos de producción en diversos entornos. Sus prestaciones son:

- Eficiencia en el uso de almacenamiento.
- Eficiencia en el uso de memoria.

- Tiempo de ejecución de un contenedor bajo, es decir, desde la creación a la ejecución es más rápido que otros *drivers*.

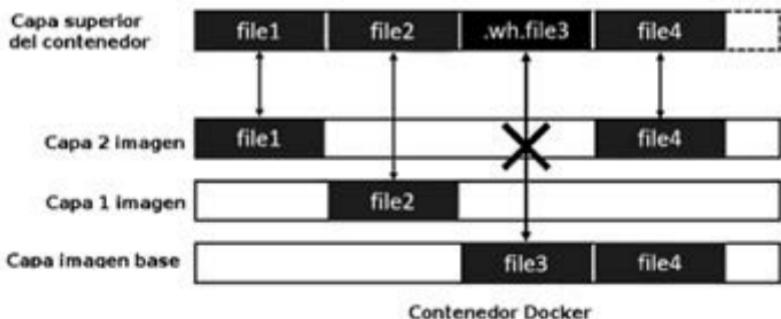
A pesar de que AUFS es estable y utilizado generalmente, hay algunas distribuciones que no lo soportan debido a que no utilizan versiones recientes del núcleo de Linux (*Kernel*).

AUFS utiliza la técnica de *union mount*, esto significa que un contenedor no duplica el contenido de una imagen, sino que a través de este método solo los ficheros nuevos creados en el contenedor serán parte del mismo. En el caso de tener varios contenedores basados en la misma imagen, solo las modificaciones dentro de los mismos se almacenarán dentro de `/var/lib/docker/`.



En esta imagen podemos observar que la *imagen* y sus *capas* son de solo lectura. Los cambios realizados dentro de contenedor se *unen* para generar un directorio de trabajo para él mismo (*Union mount*).

En el caso de modificación o creación de ficheros, simplemente se aloja dentro del directorio propio del *contenedor* y en el caso de querer acceder a él accederemos al fichero nuevo y no al de la *imagen*. El dilema sucede cuando nosotros borramos un fichero en el contenedor, para ello AUFS creará un fichero para ocultar el original. Este método en inglés es denominado *whiteout file*.



Si eliminamos el fichero *file3*, **AUFS** generará un fichero para enmascarar el fichero original y que no podamos acceder a él.

El directorio de *unión* para el contenedor está almacenado en el siguiente directorio: */var/lib/docker/aufs/mnt/<identificador>*. El directorio que contiene las diferencias entre el *contenedor* y la *imagen base e intermedias* es el siguiente: */var/lib/docker/aufs/diff/<identificador>*.

El directorio de unión solo será accesible si el *contenedor* está en ejecución. En el caso de que esté detenido, el directorio existirá pero estará vacío. Para obtener el identificador del directorio que contiene los datos debemos acceder al siguiente fichero: */var/lib/docker/image/aufs/layerdb-mounts/<idcontenedor>/mount-id*.

Veamos un ejemplo para clarificar todos estos directorios y detallar el funcionamiento de **AUFS**.

```
# docker run -dti debian
da14792c2217d7375cbba6f7e01b746031bdd39f7a11776980a093b20cc93e7b
```

Una vez que el contenedor está en ejecución y tengamos su identificador, podemos acceder a su contenido. Primero creamos un fichero de ejemplo dentro del *contenedor*.

```
# docker exec -ti da14792c2217 [...] /bin/bash
root@da14792c2217:/# echo "Prueba" > aufsprueba.txt
root@da14792c2217:/# exit
```

Después debemos obtener el identificador AUFS para el directorio `union`. Para ello podemos ejecutar el siguiente comando:

```
# cat /var/lib/docker/image/aufs/layerdb-mounts/da14792c2217[...]/mount-id
77c64bc1bd2975c16c2c5f09650092ed1c48ba183d6be4da4fe2bcf1dd2771b8
```

Obtenido el identificador de AUFS podemos acceder tanto a las diferencias como al contenido del contenedor que se ha unido con la imagen:

Listar las diferencias dentro del contenedor.

```
# ls -l /var/lib/docker/aufs/diff/77c64bc1bd2975[...]/
total 4
-rw-r--r-- 1 root root 7 Jan 14 23:01 aufsprueba.txt
drwx----- 2 root root 26 Jan 14 23:01 root
```

Listar el contenido unido, el contenido de la *imagen* unido a las diferencias del contenedor.

```
# ls -l /var/lib/docker/aufs/mnt/77c64bc1bd2975[...]/
total 12
-rw-r--r-- 1 root root 7 Jan 14 23:01 aufsprueba.txt
drwxr-xr-x 2 root root 4096 Dec 13 20:09 bin
drwxr-xr-x 2 root root 6 Sep 12 06:09 boot
drwxr-xr-x 4 root root 40 Jan 14 22:57 dev
drwxr-xr-x 41 root root 62 Jan 14 22:57 etc
drwxr-xr-x 2 root root 6 Sep 12 06:09 home
drwxr-xr-x 9 root root 105 Nov 27 2014 lib
drwxr-xr-x 2 root root 33 Dec 13 20:08 lib64
drwxr-xr-x 2 root root 6 Dec 13 20:07 media
drwxr-xr-x 2 root root 6 Dec 13 20:07 mnt
drwxr-xr-x 2 root root 6 Dec 13 20:07 opt
drwxr-xr-x 2 root root 6 Dec 13 20:09 proc
drwx----- 2 root root 26 Jan 14 23:01 root
drwxr-xr-x 3 root root 28 Dec 13 20:07 run
drwxr-xr-x 2 root root 4096 Dec 13 20:09 sbin
drwxr-xr-x 2 root root 6 Dec 13 20:07 srv
drwxr-xr-x 2 root root 6 Apr 6 2015 sys
drwxrwxrwt 2 root root 6 Dec 13 20:09 tmp
drwxr-xr-x 10 root root 97 Dec 13 20:07 usr
drwxr-xr-x 11 root root 128 Dec 13 20:07 var
#
```

En este ejemplo podemos observar que con **AUFS** ahorraremos espacio en disco, ya que solo los nuevos ficheros son generados en disco y el uso de memoria es eficiente al tener que utilizar menos ficheros en acciones de lectura y escritura. En otros drivers todo el contenido de la *imagen* es clonado en cada creación de un *contenedor*, lo que significa la necesidad de más espacio.

Las desventajas de **AUFS** surgen en entornos con alto volumen de escritura debido a los cálculos que debe realizar para saber si el fichero existe previamente, los cálculos para relacionar el fichero al *contenedor* y el fichero debe ser copiado a la capa del *contenedor*.

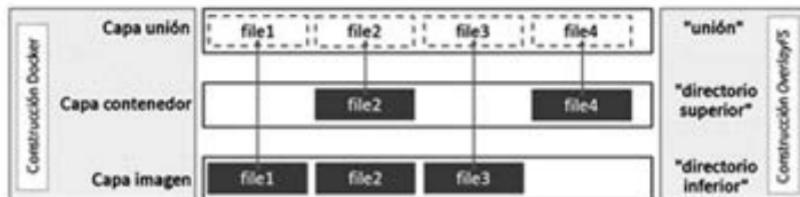
OVERLAYFS/OVERLAYFS2

OverlayFS es el driver moderno para *union mount*. Es similar a **AUFS** pero con estas ventajas:

- Un diseño más simple.
- En el núcleo de Linux (*Kernel*) desde la versión 3.18.
- Más rápido.

Aunque sus ventajas son claras, es menos maduro que **AUFS**. Desde la versión 1.12 de **Docker** y utilizando el núcleo de Linux 4.0 y posteriores, es posible utilizar **OverlayFS2** que es mucho más eficiente en el uso de i-nodos (índice para un fichero dentro de un sistema de ficheros).

Al igual que con **AUFS** tendremos un directorio *union* con las capas de la *imagen* y del *contenedor*.



El directorio utilizado para *overlay* es el siguiente: `/var/lib/docker/overlay/`

Para habilitar el driver *overlay2* (en el caso de tener *Kernel 4.X*) utilizaremos las siguientes instrucciones:

- Detener el servicio Docker: `systemctl stop docker`
- Eliminar el directorio raíz: `rm -rf /var/lib/docker`
- Configurar o añadir la opción `--storage-driver` dentro del fichero `/etc/default/docker` al valor (ejemplo):

```
DOCKER_OPTS="--storage-driver=overlay2"
```

- Iniciar el servicio: `systemctl start docker`

Una vez iniciado el servicio, si no ha fallado la inicialización, podemos obtener la información del servidor:

```
$ sudo docker info
[...]
Server Version: 1.12.1
Storage Driver: overlay2
Backing Filesystem: extfs
```

Podemos observar qué sucede cuando descargamos una *imagen* desde el repositorio oficial:

```
# docker pull debian
Using default tag: latest
latest: Pulling from library/debian
75a822cd7888: Pull complete
Digest: sha256:f7062cf040f67fc26ff46b3b44fe036c29468a7e69d8170f37c57f2eec1261b
Status: Downloaded newer image for debian:latest
# ls -l /var/lib/docker/overlay2/
total 8
drwx----- 3 root root 4096 Jan 15 14:33
4a986cdef4efa407e0a431c09f759e83b85005a1427c9784dc3d746c97fab22
drwx----- 2 root root 4096 Jan 15 14:33 |
```

Observamos que un directorio se ha creado dentro de `/var/lib/docker/overlay2` que hace referencia a la *Imagen* descargada. En el caso de ejecutar un *contenedor*, el contenido del directorio será el siguiente:

```
# ls -l /var/lib/docker/overlay2/865d29df6d6[...]/
total 20
drwxr-xr-x 2 root root 4096 Jan 15 14:52 diff
-rw-r--r-- 1 root root  26 Jan 15 14:52 link
-rw-r--r-- 1 root root  57 Jan 15 14:52 lower
drwxr-xr-x 1 root root 4096 Jan 15 14:52 merged
drwx----- 3 root root 4096 Jan 15 14:52 work
```

El directorio *merged* es el directorio donde se unen los ficheros de la *Imagen base* y sus distintas capas. Dentro del directorio *diff* se alojarán los ficheros propios del *contenedor*. Ejemplo:

```
# docker exec -ti 3b922284e821 touch /root/test.txt
# ls -l /var/lib/docker/overlay2/865d29df6d6[...]/diff/root/
total 0
-rw-r--r-- 1 root root 0 Jan 15 15:00 test.txt
```

En este ejemplo hemos creado un fichero vacío dentro del *contenedor* y observamos que se aloja dentro del subdirectorio llamado *diff* dentro del directorio de *overlay2*. Dentro del directorio *merged* se muestra el contenido de la *Imagen* unida con todas sus capas:

```
# ls -l /var/lib/docker/overlay2/865d29df6d6[...]/merged/
total 76
drwxr-xr-x 2 root root 4096 Dec 13 19:09 bin
drwxr-xr-x 2 root root 4096 Sep 12 04:09 boot
drwxr-xr-x 1 root root 4096 Jan 15 14:52 dev
drwxr-xr-x 1 root root 4096 Jan 15 14:52 etc
drwxr-xr-x 2 root root 4096 Sep 12 04:09 home
drwxr-xr-x 9 root root 4096 Nov 27 2014 lib
drwxr-xr-x 2 root root 4096 Dec 13 19:08 lib64
[...]
```

DEVICE MAPPER

Este *driver* fue desarrollado por la compañía Red Hat debido a que en su sistema operativo, *Red Hat Enterprise Linux*, incluía una versión del núcleo de Linux sin

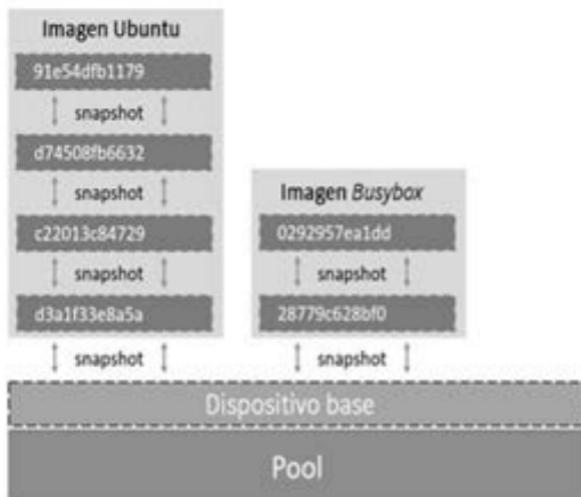
soporte para **AUFS**. Aunque la compañía contempló incluir el soporte **AUFS**, decidieron utilizar el *Device Mapper* por su estabilidad y su madurez.

Con *devicemapper* cada *imagen* y cada *contenedor* se alojan dentro de su propio dispositivo virtual. La gran diferencia de este *driver* es que funciona a nivel de bloque y no a nivel de fichero como veíamos anteriormente. Para ahorrar espacio y operaciones utiliza la técnica de *snapshot* (instantánea) para manipular *imágenes* y *contenedores*.

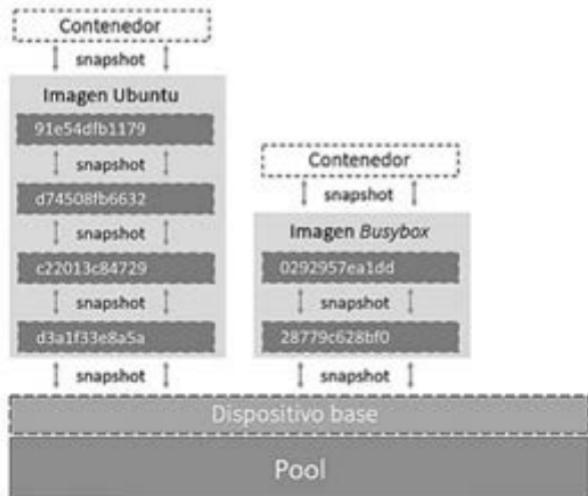
Al elegir *devicemapper* como *driver* tendremos dos opciones:

- Crear un dispositivo virtual llamado *loop*, que será un fichero alojado dentro de `/var/lib/docker/devicemapper/devicemapper`, que será utilizado como *pool*.
- Utilizar un dispositivo físico (un disco, una partición) para ser utilizado como *pool*.

En este caso, *pool* es un contenedor de las *imágenes* y *contenedores*, además de sus instantáneas derivadas.



En el caso de los contenedores simplemente creamos una *instantánea* de la *imagen* como se muestra a continuación:



Las distribuciones que soportan *devicemapper* en la actualidad son las siguientes:

- *RHEL/CentOS/Fedora*
- *Ubuntu 12.04*
- *Ubuntu 14.04*
- *Debian*
- *Arch Linux*

En el caso de utilizar este *driver*, la información mostrada por `docker info` será similar a la siguiente:

```

# docker info
[omitido]
Storage Driver: devicemapper
  
```

```
Pool Name: docker-253:0-15385-pool
Pool Blocksize: 65.54 kB
Base Device Size: 10.74 GB
Backing Filesystem: xfs
Data file: /dev/loop0
Metadata file: /dev/loop1
[...]
Data loop file: /var/lib/docker/devicemapper/devicemapper/data
WARNING: Usage of loopback devices is strongly discouraged for production use. Either use '--storage-opt dm.thinpooldev' or use '--storage-opt dm.no_warn_on_loop_devices=true' to suppress this warning.
Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
Library Version: 1.02.135-RHEL7 (2016-09-28)
```

Como observamos, por defecto utiliza lo denominado *loop* que es un fichero alojado en disco y tratado como un dispositivo para ser usado con *LVM*. Los ficheros son los siguientes:

- */var/lib/docker/devicemapper/devicemapper/data* -> Contiene el contenido de *ímágenes y contenedores*.
- */var/lib/docker/devicemapper/devicemapper/metadata* -> Contiene la información de metadatos necesarios para el funcionamiento.

Al utilizar en modo *loop*, Docker nos indicará con un aviso que no es recomendado para el uso en producción. La recomendación es utilizar un dispositivo separado para el uso de *devicemapper*. Antes de cambiar del modo *loop* al modo *direct-lvm* (utilizando dispositivos) es necesario hacer una copia de seguridad de *ímágenes y contenedores*, ya que el cambio de *driver* conlleva la pérdida de todos los datos.

Los pasos para utilizar un dispositivo se detallan a continuación para *CentOS/Red Hat*:

- Detener Docker: `systemctl stop docker`

- `rm -rf /var/lib/docker` -> Eliminar los datos antiguos, recordar hacer una copia de seguridad antes.
- Configurar el fichero `/etc/sysconfig/docker-storage-setup` para la opción `DEVS` para apuntar a nuestro dispositivo físico (disco o partición) y el nombre del grupo de volúmenes.
- Ejemplo:

```
DEVS="/dev/sdb"
VG="curso-docker-vg"
```

- Nota: todas las opciones que son configurables se pueden ver en el fichero `/usr/lib/docker-storage-setup/docker-storage-setup`
- Ejecutar el comando `docker-storage-setup`, la salida del comando se muestra a continuación:

```
# docker-storage-setup
[amitido]
Logical volume "docker-pool" created.
Logical volume curso-docker-vg/docker-pool changed.
```

El comando `docker-storage-setup` realizará las siguientes tareas:

- Añadir una firma al dispositivo por motivos de seguridad.
- Particionar el disco.
- Crear un grupo de volúmenes (*Volume Group*) y crear un volumen lógico (*Logical Volume*) -> Por defecto con el 40% del espacio libre.
- Configurar el fichero `/etc/sysconfig/docker-storage` para utilizar el dispositivo:

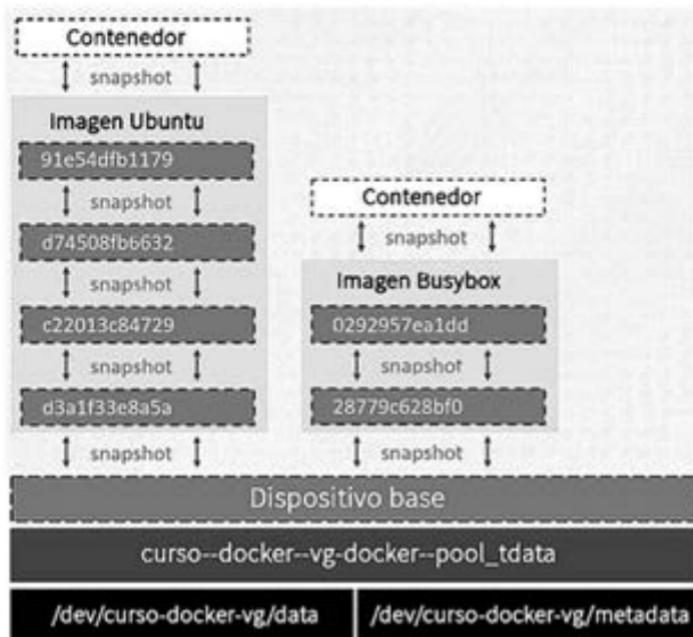
Contenido fichero `/etc/sysconfig/docker-storage`

```
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper --storage-opt dm.fs=xfs --
storage-opt dm.thinpooldev=/dev/mapper/curso-docker-vg-docker-pool --storage-
opt dm.use_deferred_removal=true"
```

En la salida del comando detectamos que un nuevo grupo de volúmenes (*Volume Group*) ha sido creado con el nombre “*curso-docker-vg*” y un volumen lógico (*Logical Volume*) con el nombre “*docker-pool*”. Después de iniciar **Docker** con el comando: `systemctl start docker`, podemos obtener la información del servicio:

```
# docker info
[omitido]
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: VolGroup00-docker--pool
 Pool Blocksize: 524.3 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
```

Al configurar nuestro dispositivo físico en *devicemapper* nos llevará a la siguiente imagen al trabajar con *índices* y *contenedores* en **Docker**:



Aunque *devicemapper* es uno de los *drivers* más fáciles de utilizar y más flexibles, tiene las siguientes desventajas:

- El modo por defecto de *devicemapper* es el modo *loop*, que tiene bajo rendimiento y no debe ser usado en producción. Para la producción es necesario utilizar *direct-lvm* como observábamos en esta sección.
- El uso de la memoria no es el más óptimo, por ello debemos monitorizarlo y alojarlo en un servidor con buenos recursos.
- Debido a que el rendimiento de disco, al trabajar con bloques, no es el mejor, solo se recomienda utilizar discos de alta velocidad: *SSD* o almacenamiento *SAN/NAS*.

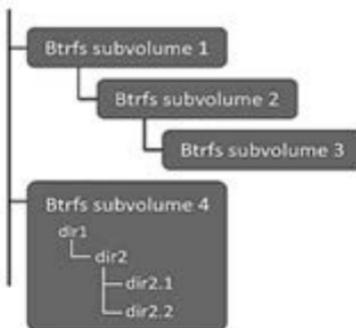
Una de las recomendaciones al utilizar *devicemapper*, es que los directorios que posean un alto uso de operaciones de escritura serán alojados en un *volumen* de Docker. Los *volumenes*, al estar separados del resto del contenedor, tienen un mejor rendimiento a la hora de escribir en ellos utilizando este *driver*.

BRTFS

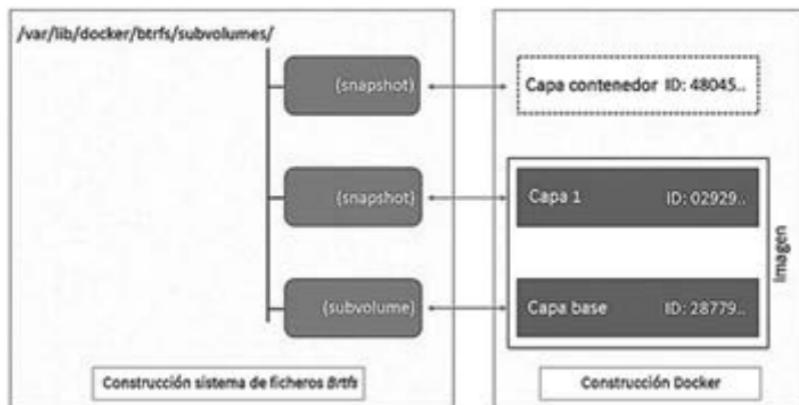
El sistema de fichero *brtfs* y su *driver* para Docker es una de las grandes apuestas para el futuro debido a sus ventajas y sus prestaciones. Forma parte del núcleo de Linux (*Kernel*) y se considera estable, pero como está bajo desarrollo intensivo tiene muchas modificaciones en poco tiempo.

El soporte comercial de Docker, actualmente, no da soporte para ninguna distribución para el uso de *brtfs*. Este *driver* se considera como el reemplazo natural de *devicemapper* (pero este es mucho más maduro, estable y probado en la actualidad), pero solo deberíamos utilizarlo en un entorno previamente probado antes de utilizarlo en producción.

Docker utiliza *subvolúmenes* y *snapshots* de *brtfs* para manejar las distintas capas de imágenes y contenedores. Un *subvolumen* tiene la misma estructura que un sistema de ficheros en Linux. Los *subvolúmenes* pueden estar anidados, y utilizan la técnica *copy-on-write* para distribuir los datos. En la siguiente imagen vemos cómo los tres primeros *subvolúmenes* están anidados y el último muestra la estructura de directorios.



El directorio donde se alojan los *subvolúmenes* de este driver es el siguiente: `/var/lib/docker/btrfs/subvolumes` y la composición de un contenedor se muestra en la imagen siguiente:



Antes de utilizar **btrfs**, debemos comprobar si nuestro sistema lo soporta. Para ello ejecutaremos el siguiente comando:

```
# grep -c btrfs /proc/filesystems
1
```

En el caso de que la búsqueda devuelva 1, significa que podremos usar este sistema de ficheros y el *driver* de almacenamiento relacionado para **Docker**. En el caso de

que el disco o partición que vayamos a utilizar esté en uso por Docker como parte de otro tipo de almacenamiento, debemos detener el servicio y eliminar el directorio `/var/lib/docker`. Recordad que todas las imágenes y contenedores se eliminarán.

```
# systemctl stop docker
# rm -rf /var/lib/docker/
# mkdir /var/lib/docker
```

Una vez detenido el servicio, podemos crear el sistema de ficheros y montar el disco o partición dentro del directorio `/var/lib/docker`.

Formatear el dispositivo a utilizar con el sistema de ficheros btrfs.

```
# mkfs.btrfs -f /dev/sdc
```

`btrfs-progs v4.4`

See <http://btrfs.wiki.kernel.org> for more information.

```
Label:      (null)
UUID:      e04f0622-d01a-4f02-90d2-2c8849a45cec
Node size:   16384
Sector size: 4096
Filesystem size: 8.00GiB
Block group profiles:
  Data:    single     8.00MiB
  Metadata: DUP     417.56MiB
  System:  DUP     12.00MiB
SSD detected: no
Incompat features: extref, skinny-metadata
Number of devices: 1
Devices:
  ID  SIZE PATH
  1  8.00GiB /dev/sdc
```

Montar el sistema de ficheros btrfs en el directorio `/var/lib/docker`

```
# mount /dev/sdc /var/lib/docker
```

Una vez realizadas dichas tareas, estamos listos para ejecutar el servicio de Docker utilizando el *driver* de btrfs como se muestra a continuación:

- Editar el fichero `/etc/default/docker` para indicar el *driver* de almacenamiento.

```
# grep DOCKER_OPTS /etc/default/docker
DOCKER_OPTS="--storage-driver=btrfs"
```

- Iniciar el servicio y comprobar el *driver* utilizado:

```
# systemctl start docker
# docker info
[...]
Server Version: 1.12.1
Storage Driver: btrfs
Build Version: Btrfs v4.4
Library Version: 101
```

Una vez descargada una *imagen* y ejecutado un contenedor basado en ella, veremos que el directorio `/var/lib/docker/btrfs/subvolumes/` se llena con subdirectorios que se enlazan entre ellos:

```
# ls -1 /var/lib/docker/btrfs/subvolumes/
d68bb63dcc5d3bb5e82c0c8e4affb80fe16053abdf6bf16eb47881a25b528bbd
f2bdce3e87690723c79ed46a7bb093a4aa58b291a38768bfce3264d137f3b2ff
f2bdce3e87690723c79ed46a7bb093a4aa58b291a38768bfce3264d137f3b2ff-init
```

El primer directorio listado consiste en la *imagen*, el segundo directorio es el relativo al *contenedor* y el último con la palabra “*-init*” es el contenido del contenedor al iniciarse. Al iniciar el contenedor, los ficheros hacen referencia a los ficheros originales de la imagen:

```
# ls -1i /var/lib/docker/btrfs/subvolumes/* /etc/passwd
540 /var/lib/docker/btrfs/subvolumes/d68bb63dcc5[...]/etc/passwd
540 /var/lib/docker/btrfs/subvolumes/f2bdce3e8769[...]/etc/passwd
540 /var/lib/docker/btrfs/subvolumes/f2bdce3e8769[...]-init/etc/passwd
```

En la salida de este comando, observamos que el *i-nodo* (la posición dentro de la partición) coincide en los tres subdirectorios de btrfs tanto para la *imagen* como para el *contenedor*. En el caso de modificar dicho fichero (creando un usuario, cambiando

la contraseña, etc.) veremos que el fichero es modificado solo dentro del directorio del contenedor y no en los otros dos:

```
# ls -li /var/lib/docker/btrfs/subvolumes/* /etc/passwd
540 /var/lib/docker/btrfs/subvolumes/d68bbcd63dcc5[...]/etc/passwd
8549 /var/lib/docker/btrfs/subvolumes/f2bdce3e8769[...]/etc/passwd
540 /var/lib/docker/btrfs/subvolumes/f2bdce3e8769[...].init/etc/passwd
```

Se puede observar que el directorio de la *Imagen*, como era de esperar, está intacto y el directorio con el nombre "*-init*" que contiene el estado inicial del contenedor, también.

Las desventajas del uso de **btrfs** se resumen a continuación:

- Escrituras a disco de tamaños pequeños tienen mal rendimiento.
- No soporta *caché* compartido, sino individual. Esto conlleva que el acceso a un mismo fichero desde varios contenedores conlleva crear copias individuales para el *caché*.
- Las escrituras en modo secuencial tienen bajo rendimiento debido al uso de *journaling*.
- Es necesario rebalancear los dispositivos periódicamente, se recomienda leer la documentación para programar esta tarea automáticamente.

ZFS

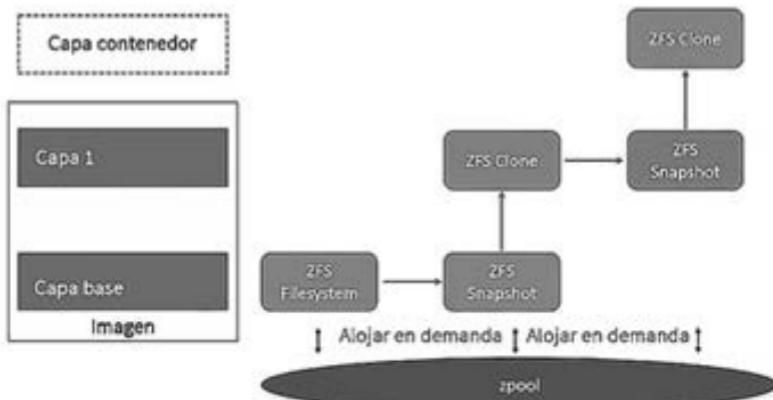
El sistema de ficheros **ZFS** es uno de los sistemas de ficheros más potentes y avanzados que existe. Creado por *Sun Microsystems* (ahora parte de *Oracle*), como código propietario para *Solaris* y posteriormente liberado como *código abierto* para la comunidad pero con limitación de licencia para incluirlo en el núcleo de *Linux*. Existe un proyecto llamado *ZFS On Linux* (*ZoL*) que provee todo lo necesario para habilitar este sistema de ficheros en *Linux*.

Además, existe un proyecto dentro de *FUSE (Filesystem in Userspace)*, que aunque funciona con *Docker*, no es recomendado su uso. La recomendación, en caso de querer utilizar **ZFS**, es seguir el procedimiento descrito por *ZoL*.

ZFS trabaja con tres elementos importantes:

- Sistemas de ficheros.
- Instantáneas (*snapshots*).
- Clones.

En la siguiente imagen podemos ver un diagrama del funcionamiento con *imágenes y contenedores*.



Veamos un ejemplo dentro de un sistema Ubuntu 16.04 LTS, con un disco separado para el uso de Docker: `/dev/sdc`. Primero instalamos el software proporcionado por ZoL.

```
# apt-get install -y zfs
[...]
Setting up zfs-zed (0.6.5.6-0ubuntu15) ...
zed.service is a disabled or a static unit, not starting it.
Processing triggers for libc-bin (2.23-0ubuntu5) ...
Processing triggers for systemd (229-4ubuntu13) ...
Processing triggers for ureadahead (0.100.0-19)
```

Una vez instalado, debemos comprobar que el modo `zfs` ha sido cargado correctamente por nuestro sistema *Linux*.

```
# lsmod | grep zfs
zfs           2813952  3
```

zunicode	331776	1 zfs
zcommon	57344	1 zfs
znvpair	90112	2 zfs, zcommon
spf	102400	3 zfs, zcommon, znvpair
zavl	16384	1 zfs

A continuación, con el servidor Docker detenido, configuraremos el pool de ZFS para el uso dentro de Docker. Las instrucciones se muestran a continuación:

```
# zpool create -f zpool-docker /dev/sdc
# zfs list
NAME      USED  AVAIL REFER MOUNTPOINT
zpool-docker  55K  7.69G  19K  /zpool-docker
# zfs create -o mountpoint=/var/lib/docker zpool-docker/docker
# zfs list -t all
NAME      USED  AVAIL REFER MOUNTPOINT
zpool-docker    95K  7.69G  19K  /zpool-docker
zpool-docker/docker  19K  7.69G  19K  /var/lib/docker
```

En el primer comando hemos creado nuestro pool, utilizando el disco alojado en `/dev/sdc`. Una vez creado nuestro pool, crearemos un volumen que será montado en el directorio `/var/lib/docker` (el directorio principal de Docker). Una vez iniciado el servicio, podemos ejecutar `docker info` para ver los detalles:

```
# systemctl start docker
# docker info
[omitido]
Storage Driver: zfs
Zpool: zpool-docker
Zpool Health: ONLINE
Parent Dataset: zpool-docker/docker
Space Used By Parent: 19456
Space Available: 8256390144
Parent Quota: no
Compression: off
```

Una vez configurado nuestro servidor Docker para utilizar ZFS, podemos descargar una *imagen* para ver el funcionamiento de este *driver*:

```
# docker pull debian
Using default tag: latest
```

```
latest: Pulling from library/debian
75a822cd7888: Pull complete
Digest: sha256:f7062cf040f67f0c26ff46b3b44fe036c29468a7e69d8170f37c57f2eec1261b
Status: Downloaded newer image for debian:latest

# zfs list -t all


| NAME                                       | USED | AVAIL | REFER | MOUNTPOINT      |
|--------------------------------------------|------|-------|-------|-----------------|
| zpool-docker                               | 137M | 7.56G | 19K   | /zpool-docker   |
| zpool-docker/docker                        | 137M | 7.56G | 688K  | /var/lib/docker |
| zpool-docker/docker/80dbf5945b3d588429f..J | 136M | 7.56G | 136M  | legacy          |


```

En cada creación de una *imagen* o *contenedor*, se creará un nuevo volumen dentro de ZFS. Observamos que tenemos un tamaño global (7.56G) en este caso, que será compartido por todos los volúmenes. En el caso de eliminación de un objeto de Docker, también se eliminará en ZFS:

```
# docker rmi debian
Untagged: debian:latest
Untagged: de-
bian@sha256:f7062cf040f67f0c26ff46b3b44fe036c29468a7e69d8170f37c57f2eec1261b
Deleted: sha256:19134a8202e737105fb53da5749afada404c8926eccfcf3dad2d6866d6d60c
Deleted: sha256:b6ca02dfe5e62c58dacb1dec16eb42ed35761c15562485f9da9364bb7c90b9b3
# zfs list -t all


| NAME                | USED | AVAIL | REFER | MOUNTPOINT      |
|---------------------|------|-------|-------|-----------------|
| zpool-docker        | 792K | 7.69G | 19K   | /zpool-docker   |
| zpool-docker/docker | 685K | 7.69G | 685K  | /var/lib/docker |


```

VOLÚMENES

Un *volumen* en Docker es un directorio especial asignado a un contenedor y que aloja los datos dentro del directorio `/var/lib/docker/volumes`. Las características y ventajas de utilizar volúmenes son las siguientes:

- Se inicializan durante la creación del contenedor.
- Se pueden compartir entre diferentes contenedores.
- Las modificaciones son realizadas inmediatamente a disco sin pasar primero por caché (con la posibilidad de que eso puede significar la pérdida de datos).

- Los cambios en el volumen no se verán afectados en el caso de que se actualice la *Imagen base del contenedor*.
- La información dentro del *volumen* permanece incluso después de la eliminación del *contenedor*.

Para crear un contenedor durante las acciones `run` o `create`, utilizaremos la opción `-v (--volume)` y el nombre del directorio como se muestra en el siguiente ejemplo:

```
# docker run --name pruebavolume -v /var/www debian df
Filesystem      1K-blocks  Used Available Use% Mounted on
overlay        10098468 4933376 5148708 49% /
[omitido]
/dev/sda1     10098468 4933376 5148708 49% /var/www
```

Al igual que con otras configuraciones relacionadas con *contenedores*, es posible utilizar la acción `inspect` para listar los volúmenes asignados.

```
# docker inspect -f "{{ .Config.Volumes }}" pruebavolume
map[/var/www:{}]
# docker inspect -f "{{ .Mounts }}" pruebavolume
[{"volume": "58eb8726e339597ce24984db5b54d03a39fb5d963d27696f3291abc50ab4e408",
 "/var/lib/docker/volumes/58eb8726e339597ce24984db5b54d03a39fb5d963d27696f3291abc50ab4e408/_data /var/www local true}]
```

En el apartado `.Mounts` es posible obtener el directorio local del servidor Docker asociado para dicho volumen. En él podemos acceder a los ficheros relativos al *contenedor*, añadir nuevos o manipular los existentes.

Ejecutar un contenedor utilizando un *volumen*.

```
# docker run -dti --name pruebavolume2 -v /var/www debian
27fc3988e1d70a16992df01c2da3634fb0aa5e002c79a244d15e3aa2ac254630
```

Inspeccionar el contenedor y listar el contenido del volumen desde el servidor Docker.

```
# docker inspect -f "{{ .Mounts }}" pruebavolume2
[{"volume": "3c52dc12279cd20404ef8953e5f6ff11e5b7fade0f9ac736fd20bec775379837"}]
```

```
/var/lib/docker/volumes/3c52dc12279cd20404ef8953e5f6ff11e5b7fade0f9ac736fd20bec775379837/_data /var/www local true ]
```

```
# ls -1 /var/lib/docker/volumes/3c52dc12279[...]/_data
```

```
index.html
```

Desde el servidor Docker, manipular el fichero del volumen.

```
# echo "Prueba desde el servidor" > /var/lib/docker/volumes/3c52dc12279[...]/_data/index.html
```

Comprobar desde el contenedor que el fichero ha sido modificado.

```
# docker exec -ti pruebavolume2 cat /var/www/index.html
```

Prueba desde el servidor

En el ejemplo anterior observamos que un *volumen* actúa como un directorio normal dentro del servidor Docker, haciendo ideal su uso para realizar copias de seguridad o compartir datos entre contenedores o desde el propio servidor.

La opción `-v` (`--volume`) admite otra sintaxis para compartir un directorio del servidor Docker al contenedor. La sintaxis es la siguiente:

- `-v|--volume directorio_servidor:directorio_contenedor`

En el caso de tener, por ejemplo, nuestra aplicación, página web o base de datos alojada o accesible desde el servidor, es posible hacerla accesible para la aplicación en la que se está ejecutando dentro del *contenedor*.

Preparar directorio en el servidor Docker para servir un index.html básico.

```
# mkdir /web
```

```
# echo "Prueba montar directorio" > /web/index.html
```

Ejecutar un contenedor utilizando un volumen con directorio del servidor.

```
# docker run --name pruebaweb -dti -p 4488:80 -v /web:/usr/local/apache2/htdocs httpd  
b03ceacb4b04de6f184d245ac15eeef289e600e8b578f246ba633cbff38bad669
```

Comprobar el contenido servidor por el servidor web del contenedor.

```
# curl http://localhost:4488
```

Prueba montar directorio

El cliente docker a través de la acción volume nos permite realizar las siguientes tareas:

ls [opciones]

Lista los volúmenes, las opciones son las siguientes:

- **-f/-filter filtro:** filtra la salida a través del filtro especificado.
- **-q:** muestra solo los identificadores.

Ejemplo:

```
# docker volume ls
DRIVER      VOLUME NAME
local
113ce4cd1411ff7fb85a51e77ed60b64ac6dff117311c546c19f88dbd17e5458
local
13ddc95b2de73712cd96d4cbd615186087f0c2450c02630f1e994cab18f7e83d
```

inspect [opciones] volumen

Muestra información detallada de un volumen. La opción aceptada es la siguiente:

- **-f/-format formato:** formatea la salida mostrada.

Ejemplo:

```
# docker inspect
113ce4cd1411ff7fb85a51e77ed60b64ac6dff117311c546c19f88dbd17e5458
|
{
  "Driver": "local",
  "Labels": null,
  "Mountpoint":
  "/var/lib/docker/volumes/113ce4cd1411ff7fb85a51e77ed60b64ac6dff117311c546c19f88dbd17e5458/_data",
```

```

    "Name":  

    "113ce4cd1411ff7fb85a51e77ed60b64ac6dff117311c546c19f88dbd17e5458",  

    "Options": {},  

    "Scope": "local"  

}  

]

```

create [opciones] [nombre]

Crea un volumen nuevo a utilizar, las opciones son las siguientes:

- **-d/–driver driver:** especifica el driver a utilizar (por defecto local).
- **-o/–opt:** opciones para el driver especificado.

Una vez creado un volumen manualmente podemos utilizar la opción -v/–volume durante la creación para asignarlo. Ejemplo:

```

Crear volumen con un nombre específico.  

# docker volume create pruebavol  

Crear fichero desde el servidor Docker dentro del volumen.  

# echo "Prueba precreacion" > /var/lib/docker/volumes/pruebavol/_data/ejemplo.txt  

Arrancar un contenedor utilizando dicho volumen y comprobar el contenido.  

# docker run -dti --name pruebacreacion -v pruebavol:/prueba debian  

026414927d1f78193603d28fb3cce3786ef704a8a8a39022d156d45c480fa13c  

# docker exec -ti pruebacreacion cat /prueba/ejemplo.txt  

Prueba precreacion

```

prune [opciones]

Elimina los volúmenes sin uso al menos por un contenedor, la opción aceptada es la siguiente:

- **-f:** no pregunta por confirmación.

Ejemplo:

```
# 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:
113ce4cd1411ff7fb85a51e77ed60b64ac6dff117311c546c19f88dbd17e5458
63e18a4c66a1de166c2efd62416a895936ae602989f2503f9e52dcad8b012f6c
7027441e73c6807a798e3d80054c3038f8ab62fa38c4bc47cf5f9750f286592e

Total reclaimed space: 29.97 MB
```

`rm [opciones] volumen`

Elimina el *volumen* especificado, la opción aceptada es la siguiente:

- **-f:** fuerza la eliminación incluso si está en uso.

Ejemplo:

```
# docker volume rm -f pruebavol
pruebavol
```

PLUGINS PARA VOLÚMENES

A partir de la versión 1.8.0 de Docker, es posible integrar *plugins* externos para soportar diversos almacenamientos externos a utilizar por los *volúmenes*. A través de estos almacenamientos externos podemos tener nuestros datos de forma persistente independientemente del estado del servidor Docker. La lista de *plugins* más importantes y soportados se lista a continuación:

Plugin	Descripción
Azure File Storage	Permite montar Microsoft Azure File Store como volúmenes.
Contiv Volume	Permite montar nodos CEPH y NFS.
Convey	Permite montar dispositivos diversos como: Device Mapper, Virtual File System/Network File System y Amazon Elastic Block (EBS)
DRBD	Permite montar almacenamiento proporcionado por DRBD
Flocker	Permite montar almacenamiento proporcionado por Flocker.
HPE 3Par	Permite montar almacenamiento HPE 3Par y StoreVirtual SCSI.
NetApp (nDVP)	Permite montar almacenamiento proporcionado por NetApp.
NetShare	Permite montar almacenamiento proporcionado por NFS 3/4, AWS EFS (Elastic Block Filesystem) y CIFS (Common Internet File System).
Vmware Vsphere	Permite montar almacenamiento proporcionado por vSphere.

Las instrucciones de instalación y de uso se detallan en la siguiente dirección web: https://docs.docker.com/engine/extend/legacy_plugins/. Como ejemplo, instalaremos el plugin llamado *NetShare* para montar un directorio desde un servidor NFS.

Instalar plugin en Ubuntu e iniciar el servicio.

```
# wget https://github.com/ContainX/docker-volume-netshare/releases/download/v0.20/docker-volume-netshare_0.20_amd64.deb
# dpkg -i docker-volume-netshare_0.20_amd64.deb
# service docker-volume-netshare start
```

Ejecutar contenedor utilizando el driver y para montar un directorio NFS.

```
# docker run -dit --volume-driver=nfs -v 192.168.56.104:/data debian /bin/bash
079c95f5839ffe48f60066c25d73611f775f5a1b41829a6095f438ae62163348
# docker volume ls -f driver=nfs
DRIVER      VOLUME NAME
nfs         192.168.56.104/
```

A partir de la versión 1.12 de Docker, los *plugins* cambian su arquitectura y se incluye la acción *plugin* al cliente. En el momento de escribir este libro, los *plugins* actuales para dicha versión son escasos y sin documentación. La página web oficial que describe el cambio de arquitectura es la siguiente: <https://docs.docker.com/engine/extend/>. Mostramos un ejemplo utilizamos el plugin *sshfs*, que nos permite montar un directorio utilizando SSH.

Instalación del plugin, utilizando la acción `plugin install`

```
# docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
- network: [host]
- device: [/dev/fuse]
- capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
ca06593c3b54: Download complete
Digest: sha256:660b5302a69510a1c47c3042ca288d16973f236adbd2f77fc5571f164143adf5
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
```

Listar los plugins instalados actualmente.

```
# docker plugin ls
ID          NAME      DESCRIPTION      ENABLED
3f551c6fbab5  vieux/sshfs:latest  sshFS plugin for Docker true
```

Crear manualmente un volumen utilizando el driver instalado previamente.

```
# docker volume create -d vieux/sshfs --name volumenssh \
> -o sshcmd=root@192.168.56.101:/docker -o password=contrasena
volumenssh
```

Ejecutar contenedor utilizando el volumen previamente creado.

```
# docker run -v volumenssh:/data debian ls /data
ejemplo.txt
```


9 ETIQUETAS

Introducción

A todos los objetos de Docker se les puede asignar etiquetas para facilitar la búsqueda de ellos. En esta sección vamos a ver cómo asignar etiquetas a cada uno de los objetos y cómo utilizar el filtrado.

Algunos ejemplos de uso de etiquetas:

- Definir diferentes entornos: desarrollo, pruebas, producción.
- Definir versiones de aplicaciones: 0.1, 1.0, 2.1.
- Definir la versión usada desde el sistema de versión de control.
- Relación entre objetos.
- Información sobre las redes.

CONTENEDORES

Al crear un contenedor con las acciones `create` o `run` podremos especificar etiquetas con dos opciones:

- `-l/---label`: especifica una etiqueta al contenedor. El formato de la etiqueta es: `clave=valor`. Si el `valor` no es especificado será establecido a valor vacío. Esta opción se puede especificar tantas veces como sea necesaria.

- **--label-file:** lee un fichero que contiene las etiquetas con el mismo formato visto anteriormente: *clave=valor*.

Ejemplos:

```
# docker run -dti -l "entorno=dev" -l "version=2.2" httpd:2.2
82bc8e3f6f824440c7c4bfa8cdece562f814f885755c5576d9cb8e9a1e71ed26
# docker run -dti -l "entorno=qa" -l "version=2.4" httpd:2.4
737bc4636c9338090b05ad4187639f307ab049b29fc6000ddbb27cf758f4fc34
# docker run -dti -l "entorno=dev" -l "version=2.4" httpd:2.4
6cbda8c2901dfe4b900968oe079eb17b0b9b053ff63c930bc2deea56e7d0f91b
```

Es posible listar todas las etiquetas de un contenedor utilizando la acción inspect, como se muestra a continuación:

```
# docker inspect -f "[.Config.Labels]" 82bc8e3f6f82
map[entorno:dev version:2.2]
```

Pero el uso de etiquetas es para facilitar el filtrado a la hora de listar los contenedores, para ello utilizaremos la opción **-f** y el filtro: *label=clave[=valor]*. Observamos distintos ejemplos:

Listar contenedores del entorno qa

```
# docker ps -f "label=entorno=dev"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
6cada8c2901d httpd:2.4 "httpd-foreground" 46 seconds ago Up 46 seconds 80/tcp berserk_swartz
82bc8e3f6f82 httpd:2.2 "httpd-foreground" 11 minutes ago Up 11 minutes 80/tcp desper-ate_iskov
```

Listar contenedores con una versión específica.

```
# docker ps -f "version=2.2"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
82bc8e3f6f82 httpd:2.2 "httpd-foreground" 12 minutes ago Up 12 minutes 80/tcp desper-ate_iskov
```

Filtrar por entorno y versión.

```
# docker ps -f "label=entorno=qa" -f "label(version=2.4"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

737bc4636c93	httpd:2.4	"httpd-foreground"	7 minutes ago	Up 7 minutes	80/tcp	zen_meitner
Listar todos los contenedores con una etiqueta especificada asignada pero sin filtrar por valor.						
# docker ps -f "label=entorno"						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6cbda8c2901d	httpd:2.4	"httpd-foreground"	11 minutes ago	Up 11 minutes	80/tcp	berserk_swartz
737bc4636c93	httpd:2.4	"httpd-foreground"	16 minutes ago	Up 16 minutes	80/tcp	zen_meitner
82bc8e3f5f82	httpd:2.2	"httpd-foreground"	22 minutes ago	Up 22 minutes	80/tcp	desper-
ate_riskov						

En el caso de tener un fichero con las etiquetas, usaremos la opción previamente vista: **--label-file**.

```
# cat etiquetas.txt
entorno=prod
version=2.4

# docker run -dti --label-file etiquetas.txt httpd:2.4
fe0127df3165691c6c163c731808367ba5e193519b1960163ee5ad4277f5ad29

# docker inspect -f "[.Config.Labels]" fe0127df3165
map[entorno:prod version:2.4]
```

IMÁGENES

Al igual que los *contenedores*, las *imágenes* también pueden tener asignadas etiquetas. Para ello hemos visto en el capítulo 6 de *Dockerfile* el uso de la instrucción **LABEL**.

Contenido fichero Dockerfile
FROM debian:7.4
LABEL entorno=dev version=7.4 distro=debian
Crear imagen basada en el fichero Dockerfile.
docker build -t debian74 .
<i>Sending build context to Docker daemon 2.048 kB</i>
<i>Step 1 : FROM debian:7.4</i>

```
--> a2b98fb66f92
Step 2 : LABEL entorno dev version 7.4 distro debian
--> Using cache
--> 7e6b2301b6d3

Successfully built 7e6b2301b6d3
```

(La misma operación fue realizada para *debian:7.5*).

Después de la creación de la *Imagen* con la acción **build**, podemos listar sus etiquetas con la acción **inspect**. Además, al igual que con los *contenedores*, podemos utilizar la opción **-f** para filtrar imágenes con la acción **images**.

Mostrar todas las etiquetas de una *imagen*.

```
# docker inspect -f "[.Config.Labels]" debian74
map[version:7.4 distro:debian entorno:dev]
```

Filtrar *imágenes* por una etiqueta.

```
# docker images -f "label=entorno=dev"
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian74	latest	7e6b2301b6d3	About an hour ago	115 MB
debian75	latest	286ede46b7f8	About an hour ago	85.18 MB

Filtrar *imágenes* con dos filtros diferentes.

```
# docker images -f "label=entorno=dev" -f "label(version)=7.5"
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian75	latest	286ede46b7f8	About an hour ago	85.18 MB

Un *contenedor* hereda las etiquetas de una *Imagen*. Como hemos visto anteriormente podemos añadir nuevas etiquetas en el tiempo de ejecución (**run** o **create**).

```
# docker run -dt --name prueba_debian74 debian74
b2754e15c654bf3135acc7a0249c3725673f32d050b6786b0ffda818a95c0045
# docker inspect -f "[.Config.Labels]" prueba_debian74
map[entorno:dev version:7.4 distro:debian]
```

VOLÚMENES

Al crear un volumen manualmente a través de la acción `volume create`, podemos especificarle las etiquetas a través de la opción `--label`.

```
# docker volume create --name libro --label entorno=dev
libro
```

Al igual que con los objetos mostrados previamente, podemos filtrar el listado utilizando la opción `-f`, en este caso para la acción `volume ls`. En el caso de querer mostrar la lista de etiquetas, utilizaremos `volume inspect`.

Listar etiquetas para un volumen.

```
# docker volume inspect -f "{{ .Labels }}" libro
map[entorno:dev]
```

Filtrar por etiqueta (No funciona en versiones anteriores a Docker 1.13).

```
# docker volume ls -f "label=entorno=dev"
DRIVER      VOLUME NAME
local      libro
```

REDES

La misma opción, `--label`, podemos utilizarla para `volume create`.

```
# docker network create --label "entorno=dev" dev_192_168_20
fc96516e2180b41a2d08a06f43611aea54a652f6376a9cc12f9666da9c1191ff
```

Al igual que con los volúmenes, el filtro utilizado para la acción `network inspect` es `"{{ .Labels }}"`, y la opción para filtrar el listado para la acción `network ls` es `-f`:

Listar etiquetas de una red existente.

```
# docker network inspect -f "{{ .Labels }}" dev_192_168_20
map[entorno:dev]
```

Filtrar listado de redes utilizando una etiqueta.

```
# docker network ls -f "label=entorno=dev"
```

NETWORK ID	NAME	DRIVER	SCOPE
fc96516e2180	dev_192_168_20	bridge	local

10

LIMITAR RECURSOS

Introducción

Por defecto, un **contenedor** puede utilizar todos los recursos disponibles del servidor donde está alojado. Esto puede causar que una aplicación ejecutándose dentro de uno de ellos pueda afectar al resto de ellos e incluso al propio servidor. **Docker** proporciona la opción de limitar los recursos que un **contenedor** puede utilizar: memoria, procesador o entrada/salida de disco.

Para ello, utilizaremos diferentes opciones que nos proporciona **Docker** tanto para la acción run o create.

MEMORIA

Aunque los **contenedores** no usan demasiada memoria, las aplicaciones que alojan pueden utilizar más memoria de la esperada. Ya sea por su uso o por errores en el código que deriven en un consumo abusivo. A través de las siguientes opciones podemos limitar su uso:

Opción	Descripción	Ejemplo
Opciones para limitar la memoria al ejecutar un contenedor. Las cantidades indicadas aceptan las letras: b, k, m o g, al final de un número para indicar bytes, kilobytes, megabytes o gigabytes.		
-m --memory	Limita el uso de memoria que un contenedor puede utilizar.	-m 128m
--memory-swap	Si se limita la memoria (-m, --memory) pero no se limita la memoria swap, se establecerá automáticamente al tamaño doble.	-m 256m --memory-swap 1G <i>Nota: esto indica que se limita la memoria a 128 MB y la memoria swap a 768MB. Al especificar</i>

Opción	Descripción	Ejemplo
	En el caso de especificar ambos valores, <code>--memory-swap</code> especifica el total de memoria. Si el valor especificado es -1, se indica que puede utilizar memoria swap ilimitada.	ambos valores, <code>--memory-swap</code> indica la memoria total.
<code>--memory-swappiness</code>	Define la frecuencia que almacena datos en la memoria swap. El valor de <code>swappiness</code> es establecido por defecto al mismo valor que el servidor Docker (normalmente 60). El valor que se puede especificar es desde 0 (no utilizar swap) a 100 (utilizar siempre swap).	<code>--memory-swappiness 10</code>
<code>--memory-reservation</code>	Reserva una cantidad de memoria para dicho contenedor. Útil en un entorno con poca memoria y es necesario reservar memoria para que no sea consumida por otros procesos/contenedores. La cantidad especificada debe ser menor que la indicada a <code>-m/-memory</code> .	<code>--memory-reservation 1G</code>
<code>--kernel-memory</code>	La máxima cantidad de memoria Kernel que puede utilizar.	<code>--kernel-memory 128m</code>
<code>--oom-kill-disable</code>	Cuando el servidor Docker no tiene suficiente memoria para los procesos de alta prioridad, detendrá los contenedores que consuman mucha memoria. Esta opción deshabilita OOM Killer (Out-of-Memory Killer), es muy recomendable solo utilizarla en conjunción a la opción <code>-m/-memory</code> .	<code>-m 256m --oom-kill-disable</code>

Cuando limitamos la memoria que puede utilizar el contenedor, es importante saber que desde dentro del contenedor no es posible ver esa limitación. El comando `free` u otros comandos como `top`, mostrarán la memoria del servidor Docker y no la propia del mismo.

```
# docker run -m 128m -ti debian free -m
total    used     free   shared  buffers   cached
Mem:      992     465     526      4      17     304
```

Para comprobar la limitación de memoria de un contenedor desde el servidor Docker podemos utilizar la acción inspect, las claves relacionadas al límite de uso de memoria son las siguientes:

- *Memory*: la cantidad de memoria máxima que puede utilizar y especificada con la opción *-m*.
- *KernelMemory*: memoria de *Kernel* limitada.
- *MemoryReservation*: la memoria reserva para el contenedor específico.
- *MemorySwap*: la cantidad de memoria *swap* máxima que puede utilizar.
- *MemorySwappiness*: el valor de *swappiness* para el contenedor.
- *OOMKilled*: indica si está deshabilitado *OOM Killer*.

Obtener la limitación de memoria para el contenedor:

```
# docker inspect -f "[.HostConfig.Memory]" 8bb4481252af
134217728
```

Además, podemos acceder a los valores a través de los ficheros especiales de *cgroups* en el directorio */sys/fs/cgroup/memory/docker/<identificador>*:

- *memory.limit_in_bytes*: la cantidad de memoria máxima que puede utilizar y especificada con la opción *-m*.
- *memory.memsw.limit_in_bytes*: la cantidad de memoria *swap* máxima que puede utilizar.
- *memory.swappiness*: el valor de *swappiness* para el contenedor.
- *memory.oom_control*: indica si OOM Killer está habilitado para este contenedor.

- *memory.kmem.limit_in_bytes*: la cantidad de memoria Kernel máxima que puede utilizar.
- *memory.soft_limit_in_bytes*: la memoria reservada con la opción *--memory-reservation*

Ejemplo accediendo información a
/sys/fs/cgroup/memory/docker/<identificador>:

```
# docker run -m 128m -dti debian
1c3598d4a5e10b7efbb99d8d2d66c615dc5764a6dac0804620980bedefd36141
# cat /sys/fs/cgroup/memory/docker/1c3598d4a5e10b7[...]/memory.limit_in_bytes
134217728
# cat /sys/fs/cgroup/memory/docker/1c3598d4a5e10b7[...]/memory.swappiness
60
# cat /sys/fs/cgroup/memory/docker/1c3598d4a5e10b7[...]/memory.memsw.limit_in_bytes
268435456
# cat /sys/fs/cgroup/memory/docker/1c3598d4a5e10b7[...]/memory.oom_control
oom_kill_disable 0
under_oom 0
```

En este ejemplo, hemos ejecutado un contenedor limitando la memoria a 128MB, observamos en el fichero *memory.limit_in_bytes* la memoria limitada especificada y en el fichero *memory.memsw.limit_in_bytes* la cantidad doble de la misma (comportamiento por defecto).

En el caso de que un contenedor intente utilizar más memoria de la permitida, será detenido por *OOM Killer* en caso de que no hayamos habilitado la correspondiente opción de no detener a la fuerza el contenedor. Podemos ver en los logs generados por Docker la detención a la fuerza de dicho contenedor:

```
# docker run -m 4m -dti tomcat
58963b9e1f17aa67ebd72e6de93bc242e801bb260fea683be1e75e8ec2c1ff19
# journalctl | tail
Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: Task in /docker/58963b9e1f17[...] killed as a result of limit of /docker/58963b9e1f17aa67ebd72e6de93bc242e801bb260fea683be1e75e8ec2c1ff19
Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: memory: usage 4096kB, limit 4096kB, failcnt 873
```

```

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: memory+swap: usage 8128kB, limit 8192kB, failcnt 199

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: kmem: usage 0kB, limit 9007199254740988kB, failcnt 0

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: Memory cgroup stats for /docker/58963b9e1f17[...]: cache:8KB rss:4088KB rss_huge:2048KB mapped_file:0KB dirty:0KB writeback:0KB swap:4032KB inactive_anon:3476KB active_anon:612KB inactive_file:4KB active_file:4KB unevictable:0KB

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: [ pid ] uid tgid total_vm rss nr_ptes nr_pmds swapents oom_store_adj name
Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: [ 5664] 0 5664 263955 3992 42 4 1034 0 java

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: Memory cgroup out of memory: Kill process 5664 [java] score 2476 or sacrifice child

Jan 21 18:54:26 ubuntu-xenial.localdomain kernel: Killed process 5664 [java] total-vm:1055820kB, anon-rss:3852kB, file-rss:12116kB

```

Se observa en este ejemplo que hemos intentado ejecutar un contenedor con la aplicación Tomcat limitando la memoria a lo mínimo, que es 4MB. Debido a que dicha aplicación requiere mucha más memoria, al intentar utilizarla el contenedor es detenido por querer acceder a más recursos de los permitidos.

En el caso de querer restringir el uso de memoria pero no detenerlo si intenta sobrepasarlo, sino simplemente limitarlo, utilizaremos la opción indicada anteriormente `-oom-kill-disable`.

PROCESADOR

Al igual que con la memoria, el uso de procesador por un contenedor es ilimitado. A través de las siguientes opciones es posible limitar tanto los procesadores a utilizar, como los ciclos:

Opciones para limitar el uso de CPU al ejecutar un contenedor.		
Opción	Descripción	Ejemplo
<code>--cpus</code>	Especifica cuánto, del total disponible del procesador, puede utilizar. Disponible a partir de Docker 1.13.	<code>--cpus "1.5"</code> -> indica que tiene el acceso garantizado a los recursos de un procesador y a la mitad de otro.

--cpu	Prioridad relativa, acepta los valores desde 2 a 1024 (valor por defecto).	--cpu-shares 512
--cpu-period	Especifica el periodo de uso, con conjunción a --cpu-quota. Por defecto 1 segundo.	--cpu-period=100000
--cpu-quota	Especifica la cuota que puede utilizar del total disponible.	--cpu-period=150000
--cpuset-cpus	Límite el uso de CPUs/cores que puede utilizar. Los formatos posibles son: n -> Un único procesador. n-m -> Rango de procesadores. n,m -> Lista de procesadores.	--cpuset-cpus 0 --cpuset-cpus 0-2 -> Utilizará los CPUs 0,1,2 --cpuset-cpus 0,2 -> Utilizará los CPUs 0 y 2.

En el caso de utilizar Docker 1.13 o posterior, es recomendable utilizar --cpus. En el caso de versiones anteriores aquí se indican los equivalentes como ejemplo:

Versión >=1.13	Versión < 1.13
--cpus .5	--cpu-period=100000 --cpu-quota=50000
--cpus 1.5	--cpu-period=100000 --cpu-quota=250000

Podemos comprobar la limitación de uso a través de docker inspect a través de las claves:

- *NanoCpus* -> Valor especificado con la opción --cpus
- *CpuPeriod* -> Valor especificado con la opción --cpu-period
- *CpuQuota* -> Valor especificado con la opción --cpu-quota
- *CpusetCpus* -> Los procesados elegidos con la opción --cpuset-cpus

Mostrar las limitaciones de CPU a un contenedor:

```
# docker inspect 1da539dc9f606 | grep -E "(CpuPeriod|CpuQuota|CpusetCpus)"
"CpuPeriod": 100000,
```

```
"CpuQuota": 50000,
"CpusetCpus": "1",
```

Además, al igual que con la memoria, podemos utilizar los ficheros especiales de cgroup para obtener la misma información, en este caso en el directorio: /sys/fs/cgroup/cpu/docker/<idcontenedor>/ donde los ficheros principales son los siguientes:

- cpu.cfs_period_us: contiene el valor de la opción especificada con --cpu-period
- cpu.cfs_quota_us: contiene el valor de la opción especificada con --cpu-quota

```
# cat /sys/fs/cgroup/cpu/docker/1da539dc9f6062[...]/cpu.cfs_period_us
100000
# cat /sys/fs/cgroup/cpu/docker/1da539dc9f6062[...]/cpu.cfs_quota_us
50000
```

Para acceder a la información especificada con --cpuset-cpus, utilizaremos el siguiente fichero especial: /sys/fs/cgroup/cpuset/docker/<idcontenedor>/cpuset_cpus

```
# cat /sys/fs/cgroup/cpuset/docker/1da539dc9f6062[...]/cpuset_cpus
1
```

ALMACENAMIENTO: ENTRADA/SALIDA (I/O)

Al igual que con la memoria y los procesadores, es posible limitar el uso de entrada y salida a dispositivos de almacenamiento. Las distintas opciones para restringir dicho uso se especifican en la siguiente tabla:

Opciones para limitar el uso de entrada/salida a dispositivos.		
Opción	Descripción	Ejemplo
--blkio-weight	Especifica la prioridad para las acciones de IO, entre 10 a 1000, o 0 para deshabilitarlo. Mayor número, mayor prioridad.	--blkio-weight 600
--blkio-weight-device	Dispositivo a limitar por la prioridad (peso).	--blkio-weight-device /dev/sda1

--device-read-bps	Limita la velocidad de lectura (bytes por segundo) desde un dispositivo. Formato: dispositivo:velocidad	--device-read-bps /dev/sda:10mb
--device-read-iops	Limita la velocidad de lectura (operaciones por segundo) desde un dispositivo. Formato: dispositivo:operaciones	--device-read-iops /dev/sda:100
--device-write-bps	Limita la velocidad de escritura (bytes por segundo) sobre un dispositivo. Formato: dispositivo:velocidad	--device-write-bps /dev/sda:10mb
--device-write-iops	Limita la velocidad de escritura (operaciones por segundo) sobre un dispositivo. Formato: dispositivo:operaciones	--device-write-iops /dev/sda:100

Si ejecutamos dos contenedores con diferente prioridad realizando la misma tarea, veremos que los de mayor prioridad accederán al disco de una forma más rápida. Veamos este ejemplo:

```
# docker run -dti --blkio-weight 300 --name prueba1 debian dd if=/dev/zero \
> of=/root/prueba bs=10M count=100
c4738b4b88f90855f34b45296ec23467e4de7134b8458b7a4d242d27f38e819f
# docker run -dti --blkio-weight 600 --name prueba2 debian dd if=/dev/zero \
> of=/root/prueba bs=10M count=100
9ccf80791473c934f74cbd6e758eac682977963a26c9385a61fc6e0b137e825d
```

En este caso ejecutamos dos contenedores, *prueba1* y *prueba2*, con diferentes valores de prioridad (peso) con `--blkio-weight`. Veamos la salida del contenedor para observar la velocidad de cada uno de ellos.

```
# docker logs prueba1
100+0 records in
100+0 records out
1048576000 bytes (1.0 GB) copied, 3.2183 s, 326 MB/s
# docker logs prueba2
100+0 records in
100+0 records out
1048576000 bytes (1.0 GB) copied, 1.72361 s, 608 MB/s
```

Como podemos observar en la salida del comando *dd*, el contenedor con mayor prioridad (*prueba2*) ejecuta la acción más rápido debido a que la velocidad de acceso al dispositivo es más veloz. En el caso de querer limitar la velocidad de escritura a un dispositivo, podemos utilizar la opción *--device-write-bps*:

```
# docker run -dti --device-write-bps /dev/sda:10mb --name prueba4 debian dd if=/dev/zero \
> of=/root/prueba bs=10M count=100 oflag=direct
0173e44165fb13c71a00f693c77257d83440f510078ef540ed66f3dcc56ae38
# docker logs prueba4
100+0 records in
100+0 records out
1048576000 bytes (1.0 GB) copied, 136.475 s, 7.7 MB/s
```

En este caso limitamos la velocidad de escritura al dispositivo */dev/sda* a 10MB/s, utilizamos el comando *dd* con las opciones *oflag=direct* para un acceso directo al disco para poder observar la limitación.

Si en cambio queremos restringir el número de operaciones por segundo, utilizaremos la opción *--device-write-iops*:

```
# docker run -dti --device-write-iops /dev/sda:100 --name prueba4b debian dd if=/dev/zero \
> of=/root/prueba bs=10M count=100 oflag=direct
031729a95e364040620df592ab69c368b668b8ae2a8b5f0899770328b1166d29
# docker logs prueba4b
100+0 records in
100+0 records out
1048576000 bytes (1.0 GB) copied, 17.8841 s, 58.6 MB/s
```

En este ejemplo, escribimos 100 veces un fragmento de 10MB. En este caso vemos que la velocidad se redujo debido a la limitación indicada. Si en vez de escribir 10MB, escribimos 1MB pero 1000 veces podemos comprobar lo que sucede:

```
# docker run -dti --device-write-iops /dev/sda:100 --name prueba4c debian dd if=/dev/zero \
> of=/root/prueba bs=1M count=1000 oflag=direct
dadebd09bf7d7824100b03b0686aed15f2184a2d4adb97412a8416abf5d4d4d48
# docker logs prueba4c
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB) copied, 49.8598 s, 21.0 MB/s
```

Al necesitar más operaciones (1000) la velocidad se reduce a más de la mitad, ya que hemos restringido el número de operaciones por segundo.

Como observamos con las opciones que nos proporciona Docker, podemos restringir tanto la velocidad como el número de operaciones por segundo, tanto para la escritura como para la lectura.

DOCKER UPDATE

A través de la acción update es posible establecer nuevas restricciones a un contenedor que está en ejecución. La sintaxis es la siguiente:

- **`docker update opciones contenedor`**

Las opciones que podemos utilizar se muestran en la siguiente tabla:

Opción	Descripción
<code>--blkio-weight</code>	Especifica la prioridad (peso) para las acciones de IO, entre 10 a 1000, o 0 para deshabilitarlo. Mayor número, mayor prioridad.
<code>-c</code>	Prioridad relativa, acepta los valores desde 2 a 1024 (valor por defecto).
<code>--cpu-shares</code>	
<code>--cpu-period</code>	Especifica el periodo de uso, con conjunción a <code>--cpu-quota</code> . Por defecto 1 segundo.
<code>--cpu-quota</code>	Especifica la cuota que puede utilizar del total disponible.
<code>--cpuset-cpus</code>	Límite el uso de CPUs/cores que puede utilizar. Los formatos posibles son: n -> Un único procesador. n-m -> Rango de procesadores. n,m -> Lista de procesadores.
<code>--kernel-memory</code>	La máxima cantidad de memoria Kernel que puede utilizar.
<code>--m</code>	Límite el uso de memoria que un contenedor puede utilizar
<code>--memory</code>	

--memory-reservation	Reserva una cantidad de memoria para dicho contenedor. Útil en un entorno con poca memoria y es necesario reservar memoria para que no sea consumida por otros procesos/contenedores. La cantidad especificada debe ser menor que la indicada a <code>-m/-memory</code> .
--memory-swap	Si se limite la memoria (<code>-m, --memory</code>) pero no se limita la memoria swap, se establecerá automáticamente al tamaño doble. En el caso de especificar ambos valores, <code>--memory-swap</code> especifica el total de memoria. Si el valor especificado es <code>-1</code> , se indica que puede utilizar memoria swap ilimitada.
--restart	Directiva sobre si el contenedor debe ser reiniciado en caso de que el servicio Docker es reiniciado. Las opciones son las siguientes: <ul style="list-style-type: none"> • <code>no</code> • <code>on-failure[:max-intentos]</code> • <code>always</code> • <code>unless-stopped</code>

Si ejecutamos un **contenedor**, con o sin restricciones, podemos establecerle valores a restringir y serán aplicados inmediatamente sin necesidad de detenerlo. Se muestran a continuación las acciones para aumentar la restricción de memoria de un **contenedor**:

Ejecutar contenedor con límite de uso de memoria y comprobarlo con inspect.

```
# docker run -m 128m -dti debian
8bb4481252af914d760df4b3c8880c27ba00dd5814b1ca75d1f22e30f7f8093c
# docker inspect -f "[.HostConfig.Memory]" 8bb4481252af
134217728
```

Actualizar contenedor con nuevo límite de uso de memoria y comprobarlo con inspect.

```
# docker update -m 156m 8bb4481252af
8bb4481252af
# docker inspect -f "[.HostConfig.Memory]" 8bb4481252af
163577856
```

En este ejemplo hemos ejecutado un **contenedor** limitando el uso de memoria a 128MB, una vez en ejecución, a través de la acción update, actualizamos esta

restricción a 156MB. A través de la acción inspect podemos comprobar en cualquier momento qué memoria tiene limitada dicho contenedor.

La opción --restart permite crear una directiva para saber qué sucederá con el contenedor en el caso de que el servicio (o el servidor) Docker es reiniciado por algún motivo. Por defecto, ningún contenedor se ejecutará al iniciar el servicio, pero en el caso de cambiar la directiva restart, podemos indicarle que se inicie. Veamos un ejemplo:

Reiniciar servicio de Docker y comprobar que ningún contenedor está en ejecución.

```
# systemctl restart docker
```

```
# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Actualizar la política de reinicio para el contenedor, indicando que se inicie siempre.

```
# docker update --restart always 8bb4481252af
```

```
8bb4481252af
```

Reiniciar servicio de Docker y comprobar que el contenedor está en ejecución.

```
# systemctl restart docker
```

```
# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8bb4481252af	debian	"/bin/bash"	6 minutes ago	Up 1 second		naughty_darwin

Los contenedores que posean la directiva restart establecida a always siempre se iniciarán al iniciarse el servicio Docker.

11 REGISTROS

Logging drivers

Por defecto, la salida que produce un *contenedor* es almacenada internamente por el servicio de **Docker**. Como ya hemos descrito, a través de la acción `logs` podemos acceder a dichos registros.

```
# docker run -dti --name servweb httpd
a2bc0d44113c10409ae003b9b5b4c52a0c70a7a4e0c4db19db817fdc8c79395f
# docker logs servweb
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.4. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.4. Set the 'ServerName' directive globally to suppress this message
[Sun Jan 22 12:44:31.025355 2017] [mpm_event:notice] [pid 1:tid 139968353429376] AH00489:
Apache/2.4.25 (Unix) configured -- resuming normal operations
[Sun Jan 22 12:44:31.025471 2017] [core:notice] [pid 1:tid 139968353429376] AH00094: Command
line: 'httpd -D FOREGROUND'
```

Esta solución está limitada por los siguientes motivos:

- Requiere que el servicio **Docker** esté en funcionamiento.
- Los registros no son permanentes, se pierden en caso de reinicio.
- No se almacenan de forma centralizada, en caso de tener diversos servidores **Docker**, es necesario acceder al apropiado.

Docker a través de los *logging drivers* nos permite cambiar el comportamiento por defecto y almacenar los registros generados por los *contenedores* en diversas opciones que nos proporciona el mercado. A continuación listamos los *drivers* soportados:

Driver	Descripción
none	Ningún registro estará disponible.
json-file	Es el <i>driver</i> por defecto, accesible a través de <code>docker logs</code> .
syslog	Envía los registros a un servidor syslog (local o remoto).
journald	Envía los registros a <i>journald</i> (servidores usando <i>systemd</i>). También accesible a través de <code>docker logs</code> .
gelf	Envía los registros con formato Grey Extended Log Format (<i>gelf</i>), usado en servidores como <i>GrayLog</i> o <i>Logstash</i> .
fluentd	Envía los registros a un servicio <i>fluentd</i> (Colector de registros).
awslogs	Envía los registros a Amazon CloudWatch Logs.
splunk	Envía los registros a un servidor <i>splunk</i> utilizando HTTP Event Collector.
etwlogs	Envía los registros a Event Tracing para Windows (<i>ETW</i>). Solo disponible en entornos Windows.
gcplogs	Envía los registros a Google Cloud Platform (GCP) Logging.

La configuración del *driver* de registros puede ser configurado a nivel de servicio y a nivel individual por cada uno de los *contenedores*. Tanto a nivel de servicio (opciones para `dockerd`) como a nivel de *contenedor* (con las acciones `run` o `create`) las opciones para especificar un nuevo *driver* y sus opciones son las siguientes:

- `--log-driver driver`: indica el *driver* a utilizar a nivel global.
- `--log-opt opciones`: establece diferentes opciones del *driver*, como pueden ser el servidor a conectar, el puerto, o diferentes parámetros.

A través de la acción `info`, como hemos visto previamente, podemos ver las opciones del servidor Docker y entre ellas podemos observar el *driver* actual:

```
# docker info | grep Logging
```

```
Logging Driver: json-file
```

En el caso de un contenedor, utilizaremos la acción `inspect` para obtener el *driver* con el que se ha ejecutado.

```
# docker inspect -f "{{.HostConfig.LogConfig.Type }}" servweb
json-file
```

A continuación detallaremos cada *driver* y las opciones que pueden ser indicadas.

json-file

El *driver* por defecto de Docker posee las siguientes opciones a utilizar con `-log-opt`:

- *max-size*: indica el tamaño máximo del registro. Los nuevos registros que sobrepasen dicha cantidad sobrescribirán registros antiguos. Se especifica con las letras *k*, *m* o *g* para *kilobytes*, *megabytes* o *gigabytes*.
- *max-file*: número máximo de ficheros a conservar, solo efectivo al utilizar la opción *max-size*.

Los registros son alojados dentro de ficheros dentro del directorio `/var/lib/docker/containers/<idcontenedor>` con el nombre `<idcontenedor>-json.log` y en el caso de especificar la opción *max-file* incluirá al final del fichero un número (`<idcontenedor>-json.log.1`).

```
# docker run -dit --log-driver=json-file --log-opt max-size=512 \
> --log-opt max-file=3 --name pruebabson httpd
ef63eeb9f3a9366aea1e0321c1a5494c0bbaf3c2db9284452231875c2c097d08
# docker logs pruebabson
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.7. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.7. Set the 'ServerName' directive globally to suppress this message
[Sun Jan 22 13:43:58.880133 2017] [mpm_event:notice] [pid 1:tid 140500016867200] AH00489:
Apache/2.4.25 (Unix) configured -- returning normal operations
[Sun Jan 22 13:43:58.880260 2017] [core:notice] [pid 1:tid 140500016867200] AH00094: Command
line: 'httpd -D FOREGROUND'
# ls -l /var/lib/docker/containers/ef63eeb9f3a9366[...]/ef63eeb9f3a9366[...].json.log*
```

```
----- 1 root root 193 Jan 22 13:43 /var/lib/docker/containers/ef63[...]/ef63[...].json.log
-rw-r-- 1 root root 708 Jan 22 13:43 /var/lib/docker/containers/ef63[...]/ef63[...].json.log.1
```

En este caso hemos limitado el tamaño máximo (aproximado para no cortar líneas) y se han creado dos ficheros. El contenido de estos ficheros es en formato **JSON (JavaScript Object Notation)** que incluye la fecha en que el registro fue generado, el tipo de salida (estándar o de error) y el registro en sí.

```
# cat /var/lib/docker/containers/ef63eeb9f3a9366[...]/ef63eeb9f3a9366[...].json.log
{"log": "[Sun Jan 22 13:43:58.880260 2017] [core:notice] [pid 1:tid 140500016867200] AH00094:
Command line: 'httpd -D FOREGROUND'\r\n","stream": "stdout","time": "2017-01-
22T13:43:58.880669406Z"}
```

syslog

El *driver* de *syslog* es uno de los más utilizados a la hora de utilizar Docker, ya que la mayoría de empresas suelen poseer un servidor centralizado. Las opciones que podemos indicarle a este *driver* son las siguientes:

Opción	Descripción
syslog-address	Indica el servidor donde está alojado el servidor de syslog. El formato es uno de los siguientes: <ul style="list-style-type: none"> • [tcp udp tcp-tls]://servidor:puerto -> En caso de no especificar el puerto en el primer formato, por defecto es 514. • unix://path • unixgram://path
syslog-facility	Especifica el recurso (<i>facility</i>) del registro. Los más populares son: <ul style="list-style-type: none"> • kern • user • daemon • auth • syslog • local0 / local1 / .. / local7
tag	Añade una etiqueta al registro para facilitar su filtrado.

Opción	Descripción
syslog-format	El formato de registro para syslog, las opciones son: <ul style="list-style-type: none">• rfc3164• rfc5424• rfc5424micro
En el caso de utilizar el protocolo tcp+tls para syslog estas opciones están disponibles.	
syslog-tls-ca-cert	La ruta al fichero que contiene el certificado CA (Certificate Authority)
syslog-tls-cert	La ruta al fichero que contiene el certificado TLS.
syslog-tls-key	La ruta al fichero que contiene la clave para el certificado TLS.
syslog-tls-skip	Indica si debe verificar la autenticidad del certificado. <i>True/false</i> .

En el ejemplo siguiente, ejecutamos un contenedor y sus registros serán enviados al servidor syslog que está en ejecución en el mismo servidor Docker.

```
# docker run --log-driver=syslog --log-opt tag=pruebasyslog \
--log-opt syslog-facility=daemon \
--log-opt syslog-address=unixgram://run/systemd/journal/syslog \
-dti --name pruebasyslog httpd
4f3ef6f6e96f6e513b661402df3ab734cf920be1e1b8a3207c6047f722b67a7f
```

En el caso utilizamos el fichero especial `/run/systemd/journal/syslog` para grabar los registros del contenedor. Podemos acceder a los logs de `syslog` en el fichero `/var/log/syslog`.

```
# tail -4 /var/log/syslog
```

```
Jan 22 14:18:33 ubuntu-xenial pruebasyslog[4181]: AH00558: httpd: Could not reliably determine the
server's fully qualified domain name, using 172.17.0.8. Set the 'ServerName' directive globally to sup-
press this message#015
```

```
Jan 22 14:18:33 ubuntu-xenial pruebasyslog[4181]: AH00558: httpd: Could not reliably determine the
server's fully qualified domain name, using 172.17.0.8. Set the 'ServerName' directive globally to sup-
press this message#015
```

```
Jan 22 14:18:33 ubuntu-xenial pruebasyslog[4181]: [Sun Jan 22 14:18:33.302463 2017]
[mpm_event:notice] [pid 1:tid 140660388505472] AH00489: Apache/2.4.25 (Unix) configured -- re-
suming normal operations#015
```

```
Jan 22 14:18:33 ubuntu-xenial pruebasyslog[4181]: [Sun Jan 22 14:18:33.302575 2017] [core:notice]
[pid 1:tid 140660388505472] AH00094: Command line: 'httpd -D FOREGROUND'#015
```

Observamos que los registros generados por nuestro *contenedor*, ahora son alojados en los registros del sistema. Si intentamos acceder a ellos a través del comando que utilizamos hasta ahora, `docker logs`, recibiremos el siguiente mensaje:

```
# docker logs pruebasyslog
```

"logs" command is supported only for "json-file" and "journald" logging drivers (got: syslog)

journald

Este *driver* tiene dos características importantes:

- Aloja los registros dentro del *journal* del sistema, donde los demás servicios del sistema alojan los suyos. Es una forma centralizada dentro de un sistema individual de alojar toda la información generada.
- Es posible continuar utilizando `docker logs`, esto es útil para evitar búsquedas en ficheros con muchos registros.

La única opción a utilizar es una ya conocida: *tag*, para especificar una etiqueta que se incluirá en cada registro generado por el contenedor. Utilizando este *driver* podemos filtrar los registros a través de dos claves: *CONTAINER_NAME* y *CONTAINER_ID*.

```
# docker run --log-driver=journald -dti --name pruebojournald httpd
a92353192314e77b48ea788a63dc461a759ae83ee83ac2130540979194879f33
# journalctl CONTAINER_NAME=pruebojournald -a
-- Logs begin at Sun 2017-01-22 11:51:26 UTC, end at Sun 2017-01-22 14:30:11 UTC. --
Jan 22 14:30:11 ubuntu-xenial.localdomain dockerd[4181]: AH00558: httpd: Could not reliably de-
termine the server's fully qualified domain name, using 172.17.0.9. Set the 'ServerName' directive
globally to suppress this message
```

```
Jan 22 14:30:11 ubuntu-xenial.localdomain dockerd[4181]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.9. Set the 'ServerName' directive globally to suppress this message
```

```
Jan 22 14:30:11 ubuntu-xenial.localdomain dockerd[4181]: [Sun Jan 22 14:30:11.361046 2017] [mpm_event:notice] [pid 1:tid 139842397853568] AH00489: Apache/2.4.25 (Unix) configured -- resuming normal operations
```

```
Jan 22 14:30:11 ubuntu-xenial.localdomain dockerd[4181]: [Sun Jan 22 14:30:11.361158 2017] [corenotice] [pid 1:tid 139842397853568] AH00094: Command line: 'httpd -D FOREGROUND'
```

gelf

Si los registros son centralmente almacenados utilizando *GrayLog* o *Logstash*, podemos utilizar este *driver* para enviar los registros de los contenedores. Las opciones disponibles son las siguientes:

Opción	Descripción
<code>gelf-address</code>	La dirección del servidor <i>GrayLog</i> o <i>Logstash</i> (o servidores compatibles con <i>gelf</i>) en formato: [udp tcp]://dirección:puerto
<code>gelf-compress-type</code>	Compresión del registro: gzip, zlib o none. Por defecto es gzip.
<code>gelf-compress-level</code>	El nivel de compresión: -1 a 9 (mejor compresión).
<code>tag</code>	La etiqueta a añadir a cada registro generado.

En este ejemplo, ejecutamos un contenedor basado en la imagen *httpd* y redirigimos los registros a un servidor *GrayLog*:

```
# docker run --log-driver=gelf --log-opt gelf-address=udp://servidor:12201
> -di --name pruebagelf httpd
eeafa04831d48904c369247a0e9b2a426ed00ae7b75993628243eedf5371d943
```

Cada registro será registrado dentro de *GrayLog*:

command	<code>httpd -foreground</code>	<input type="button" value="Q ▾"/>
container_id	<code>eeafab48331d48904c369247a0e982a425ed860e7b75993628243eedf5371d943</code>	<input type="button" value="Q ▾"/>
container_name	<code>pruebagelf</code>	<input type="button" value="Q ▾"/>
created	<code>2017-01-22T15:39:12.000000Z</code>	<input type="button" value="Q ▾"/>
image_id	<code>sha256:67a5f36fd84429b09117beac561309209f2d77900a21066209706ff4efca18</code>	<input type="button" value="Q ▾"/>
image_name	<code>httpd</code>	<input type="button" value="Q ▾"/>
level	<code>3</code>	<input type="button" value="Q ▾"/>
message	<code>[Sun Jan 22 15:39:12.000000 2017] {core::netcore} [pid 1:tid 139694085831808] AH00094: Command line: "httpd -D FOREGROUND"</code>	<input type="button" value="Q ▾"/>
source	<code>ubuntu-xenial.localdomain</code>	<input type="button" value="Q ▾"/>
tag	<code>eeafab48331d4</code>	<input type="button" value="Q ▾"/>
timestamp	<code>2017-01-22T15:39:12.000Z</code>	<input type="button" value="Q ▾"/>

fluentd

Si nuestra organización utiliza *Fluentd* como colector de registros, podemos utilizar este *driver* para enviar los registros de los *contenedores*. Las opciones que se pueden especificar son las siguientes:

Opción	Descripción
<code>fluentd-address</code>	Dirección del servidor fluentd, en formato servidor:puerto.
<code>fluentd-buffer-limit</code>	El tamaño máximo, por defecto, son 8MB. Se puede especificar KB, MB o GB.
<code>fluentd-retry-wait</code>	La espera que se realizará si no es posible comunicarse con el servidor para volver a reiniciarlo.
<code>fluentd-max-retries</code>	El máximo de intentos realizados antes de detener el contenedor debido a no poder comunicarse con el servidor.
<code>fluentd-async-connect</code>	Establece que la conexión será realizada en segundo plano.

Opción	Descripción
tag	Una etiqueta que se añadirá a cada uno de los registros.

En el siguiente ejemplo tenemos un servidor *fluentd* que mostrará en pantalla todo lo llegado con la etiqueta con nombre *docker*, ejecutamos un contenedor con el driver y la opción tag:

```
# docker run --log-driver=fluentd --log-opt tag=docker --log-opt fluentd-address=servidor:24224 \
> -dti --name pruebafluentd httpd
446bb9ddb871854b2211c8502c6b541017d7fad5d762017bf1ef5d79597b7757
```

Dentro del servidor *fluentd*, veremos los registros generadores por el contenedor previamente creado.

```
2017-01-22 16:27:29 +0000 [info]: adding match pattern="docker.**" type="stdout"
2017-01-22 16:27:29 +0000 [info]: adding source type="forward"
2017-01-22 16:27:29 +0000 [info]: using configuration file: <ROOT>
<source>
  @type forward
</source>
<match docker.**>
  @type stdout
</match>
</ROOT>
2017-01-22 16:27:29 +0000 [info]: listening fluent socket on 0.0.0.0:24224
2017-01-22 16:26:54 +0000 docker: [/log:"AH00558: httpd: Could not reliably determine the serv-er's fully qualified domain name, using 172.17.0.4. Set the 'ServerName' directive globally to suppress this
message\r", "container_id": "446bb9ddb871854b2211c8502c6b541017d7fad5d762017bf1ef5d79597b7757
", "container_name": "/pruebafluentd", "source": "stdout"}]
2017-01-22 16:26:54 +0000 docker: {"container_id": "446bb9ddb871854b2211c8502c6b541017d7fad5d762017bf1ef5d79597b7757", "container_na-me": "/pruebafluentd", "source": "stdout", "log": "[Sun Jan 22 16:26:54.621877 2017] [mpm_event:notice] [pid 1:tid 140126529394560] AH00489: Apache/2.4.25 (Unix) configured -- resuming normal operations\r"]}
```

```
2017-01-22      16:26:54      +0000      docker:      {"contain-
er_name": "/pruebafluentd", "source": "stdout", "log": "[Sun Jan 22 16:26:54.621981 2017] [core:notice]
[pid: 1:tid: 140126529394560] AH00094: Command line: 'httpd -D FORE-
GROUND'\\r", "container_id": "446bb9ddb871854b2211c8502c6b541017d7fad5d762017bf1ef5d79597b
7757"}
```

awslogs

En el caso de utilizar los servicios web de Amazon (*Amazon Web Services*) podemos utilizar los servicios de *Amazon Cloudwatch Logs* para enviar los registros de nuestros contenedores y analizarlos desde el portal de AWS. Para utilizar este *driver* primero debemos configurar el servicio de *Docker* para indicarle estas credenciales que utilizaremos para conectarnos a *Cloudwatch*. Para ello seguiremos estas instrucciones:

```
# mkdir -p /etc/systemd/system/docker.service.d/
# touch /etc/systemd/system/docker.service.d/aws-credentials.conf
```

Editar el fichero `touch /etc/systemd/system/docker.service.d/aws-credentials.conf` con las credenciales obtenidas en la página web de *Amazon Web Services*:
https://console.aws.amazon.com/iam/home#/security_credential

```
[Service]
Environment="AWS_ACCESS_KEY_ID=<valor de AWS>"
Environment="AWS_SECRET_ACCESS_KEY=<valor de aws>"
```

Una vez reiniciado el servicio (`systemctl daemon-reload && systemctl restart docker`), podemos ejecutar nuestros contenedores utilizando el *driver* de *awslogs*:

```
# docker run --log-driver=awslogs --log-opt awslogs-group=docker \
> --log-opt awslogs-region=eu-central-1 --log-opt tag=pruebaaws \
> -dti --name pruebaaws httpd
8a056a513699c02a8256272c4e5d856dfb699ec6869e283c4793651d036672
```

En la consola de *Cloudwatch* podemos ver los registros generados por el contenedor recién creado:

Time (UTC +/-00:00)	Message
2017-01-11	An older version found at the repository, http://
17:16:08	anonymouse: httpd: Could not safely determine the servers fully qualified domain name, using 172.17.8.1 for the FQDNName directive globally to suppress this message.
17:16:08	anonymouse: httpd: Could not safely determine the servers fully qualified domain name, using 172.17.8.1 for the FQDNName directive globally to suppress this message.
17:16:08	[Sun Jan 22 20:00:00 2017] [error] [client 172.17.8.1] (139) Invalid file descriptor: [client 172.17.8.1 port 44333] (139) Invalid file descriptor: - returning normal shutdown.
17:16:08	[Sun Jan 22 20:00:00 2017] [error] [client 172.17.8.1] (139) Invalid file descriptor: [client 172.17.8.1 port 44333] (139) Invalid file descriptor: - returning normal shutdown.

splunk

El driver *splunk* permite enviar registros tanto a *Splunk Enterprise* como a *Splunk Cloud*. Las opciones que pueden ser especificadas se describen en la siguiente tabla:

Opciones	Descripción
splunk-token	El token necesario para Splunk HTTP Event Collector.
splunk-url	La dirección web para Splunk Enterprise o Splunk Cloud.
splunk-source	Especificar el origen del evento.
splunk-format	Especificar el formato: inline, json o raw. Por defecto es inline.
splunk-sourcetype	Especificar el tipo de evento.
splunk-index	Especificar el índice del evento.
splunk-capath	Ruta al certificado CA (Certificate Authority).
splunk-caname	Nombre a usar para validar el certificado.
splunk-insecureskip-verify	Ignora la verificación del certificado.
tag	Etiqueta a añadir a cada registro generado.

En el siguiente ejemplo enviaremos los registros de nuestro contenedor a una instancia de *Splunk Cloud*. Para ello hemos configurado un *Input* de tipo *inline* que es el formato por defecto para este *driver*. El comando y los argumentos se muestran en este ejemplo:

```
# docker run --log-driver=splunk --log-opt splunk-token=0C0B6028-7466-44CD-8558-2992A0786150 \
> --log-opt splunk-url=https://input-prd-p-b5xfjzmpd5rn.cloud.splunk.com:8088/ \
> --log-opt splunk-verify-connection=true --log-opt tag=pruebasplunk \
> --log-opt splunk-caname=input-prd-p-b5xfjzmpd5rn.cloud.splunk.com \
> --log-opt splunk-capath=/root/splunk.pem --name pruebasplunk -dti httpd
51959c563dd16b3321d588bccfea0e0469ab783fce01f843b02152c22c5dd920
```

Dentro de nuestro portal de *Splunk* podemos observar los registros enviados por nuestro contenedor utilizando este *driver*.

Time	Event
1/25/17 3:02:36.775 PM	{ [-] line: [Wed Jan 25 15:02:36.772594 2017] {core:notice} [pid 1:tid 140404229625728] AH00094: Command line: 'httpd -D FOREGROUND' source: stdout tag: pruebasplunk } Show as raw text host = ubuntuuser@localhost.localdomain source = docker source = stdout sourcetype = httpevent
1/25/17 3:02:36.775 PM	{ [-] line: [Wed Jan 25 15:02:36.772482 2017] {rpn_event:notice} [pid 1:tid 140404229625728] AH00489: Apache/2.4.25 (Ubuntu) configured -- resuming normal operations source: stdout tag: pruebasplunk } Show as raw text host = ubuntuuser@localhost.localdomain source = docker source = stdout sourcetype = httpevent

etwlogs

Este *driver* no acepta ninguna opción y solo es utilizado en entorno Windows con los contenedores que se están ejecutando dentro del sistema.

```
# docker run --log-driver=etwlogs -dti --name pruebaetw httpd
f14bb55aa862d7596b03a33251c1be7dbbec8056bbdead1da8ec5ecebbe29731
```

Un ejemplo de un registro enviado se envía a continuación:

```
container_name: pruebaetw,
image_name: httpd,
container_id: f14bb55aa862d7596b03a33251c1be7dbbec8056bbdead1da8ec5ecebbe29731,
image_id: sha256:c1f9e19bd998d3565b4f345ac9aaaf6e3fc555406239a4fb1b1ba879673713824b,
source: stdout,
log: AH00094: Command line: 'httpd -D FOREGROUND'
```

gcplogs

Google proporciona una plataforma en línea, al igual que Amazon, para ejecutar servidores y aplicaciones. Si ejecutamos Docker dentro de esta plataforma, podemos especificar el *driver* llamado *gcplogs* para que automáticamente envíe los registros a la plataforma. En el caso de ejecutar el contenedor fuera de la plataforma, seguiremos estas instrucciones:

```
# mkdir -p /etc/systemd/system/docker.service.d/
# touch /etc/systemd/system/docker.service.d/gcp-credentials.conf
```

Editar el fichero `touch /etc/systemd/system/docker.service.d/aws-credentials.conf` y apuntar a la ruta del fichero descargado desde la siguiente dirección web (hay que crear clave de cuenta de servicio si no ha sido hecho previamente): <https://console.cloud.google.com/apis/credential>

```
[Service]
Environment="GOOGLE_APPLICATION_CREDENTIALS=/root/gcp.json"
```

Una vez reiniciado el servicio (`systemctl daemon-reload && systemctl restart docker`), podemos ejecutar nuestros contenedores utilizando el driver de `awslogs`:

```
# docker run --log-driver=gcplogs --log-opt gcp-project=charming-scarab-94214
> --log-opt gcp-meta-name=pruebagcp --name pruebagcp -dti httpd
33de2a4da8a90f5b216ad4c830f57872ac4bb8a356c97173506ba7acb806cb7f
```

En la consola de *Google Cloud Platform* podemos ver los registros generadores por el contenedor recién creado en el apartado *Registros*:

The screenshot shows the Google Cloud Platform Metrics interface. On the left, there's a sidebar with navigation links: 'Stackdriver', 'Registros' (which is selected and highlighted in grey), 'Métricas basadas en registros', 'Exportaciones', and 'Uso de recursos'. The main area has tabs for 'CREAR MÉTRICA' and 'CREAR EXPORTACIÓN'. Below these tabs is a search bar with filters: 'Filtrar por etiqueta o búsqueda de texto', 'Global' (selected), 'Todos los registros', and 'Cualquier nivel'. Underneath is a timestamp '2017-01-01 00:00'. A list of log entries is displayed, each with a timestamp and a detailed log message. The first entry is expanded to show its structure:

```

11:39:26.532 {"container": {"name": "/pruebagcp"}, "created": "2017-01-30T10:39:26.532519007Z", "insertId": "7a7eaf67711g", "jsonPayload": {}, "resource": {}, "timestamp": "2017-01-30T10:39:26.532519007Z", "logName": "projects/charming-scarab-94214/logs/gcpLogs-docker-driver"}

```

Below this, three more log entries are listed:

- 11:39:26.534 {"container": {"created": "2017-01-30T10:39:25.963500998Z", "name": "/pruebagcp"}, "insertId": "7a7eaf67711h", "jsonPayload": {}, "resource": {}, "timestamp": "2017-01-30T10:39:25.963500998Z", "logName": "projects/charming-scarab-94214/logs/gcpLogs-docker-driver"}
- 11:39:26.535 {"container": {"created": "2017-01-30T10:39:25.963500998Z", "name": "/pruebagcp"}, "insertId": "7a7eaf67711i", "jsonPayload": {}, "resource": {}, "timestamp": "2017-01-30T10:39:25.963500998Z", "logName": "projects/charming-scarab-94214/logs/gcpLogs-docker-driver"}
- 11:39:26.536 {"data": "[Tue Jan 30 10:39:26.534815 2017] [core instance]"}

12

DOCKER COMPOSE

Introducción

Como hemos observado a través de la acción `build` y el fichero `Dockerfile`, podemos usar **Docker Engine** para crear imágenes y posteriormente desplegar contenedores con ellas. **Docker** nos proporciona otro componente llamado **Compose** para definir y ejecutar contenedores basados en una plantilla.



Esta plantilla contendrá:

- Lista de *imágenes* a utilizar para ejecutar *contenedores*.
- La ruta a los ficheros `Dockerfile` que crearán las *imágenes* previamente especificadas (si fuese necesario).
- Los puertos a *exponer* para acceder a dicho *contenedor*.

- Los volúmenes a utilizar.
- Las variables necesarias para ejecutar aplicaciones.

Las principales características de este componente son:

- Desplegar aplicaciones en diferentes entornos aislados entre ellos.
- Utilizar volúmenes de forma persistente aunque se actualiza la plantilla para utilizar nuevas versiones de contenedores.
- Solo se recrearán los contenedores modificados: reutiliza los contenedores en caso de cambios en la plantilla.
- Variables que serán utilizadas dentro de los contenedores para su comunicación: por ejemplo, el nombre de una base de datos.

Las principales limitaciones son:

- Compose fue diseñado para ejecutar contenedores en un solo servidor y no en alta disponibilidad.
- Más centrado en entornos de desarrollo que de producción.

Los casos de uso más comunes de Docker Compose se describen a continuación:

- Entornos de desarrollo: simplifica todo el proceso de creación de imágenes, de ejecución de contenedores y comunicación entre ellos.
- Automatización de tests a aplicaciones: es posible utilizar lo denominado *continuous integration (CI)*, integración continua, para desplegar aplicaciones que están bajo desarrollo dentro de un contenedor y comprobar su estado y sus dependencias.
- En el caso de solo poseer un único servidor Docker, es posible utilizarlo como herramienta para desplegar aplicaciones.

Para entornos de producción con diferentes servidores, se recomienda utilizar Docker Swarm para crear un cluster que contendrá los contenedores en alta disponibilidad. Detallaremos en profundidad Docker Swarm en el capítulo 14.

INSTALACIÓN

La herramienta para utilizar Docker Compose no está incluida dentro de la instalación de Docker Engine, por ello debemos hacer una instalación manual. El proceso es muy simple: descargar un fichero utilizando el comando `curl` y asignarle permiso de ejecución.

```
# curl -L "https://github.com/docker/compose/releases/download/1.10.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total % Received % Xferd Average Speed Time Time Time Current
          Dload Upload Total Spent Left Speed
100 600  0 600  0  0  935  0--:--:--:--:--:-- 936
100 7857k 100 7857k  0  0 3022k  0 0:00:02 0:00:02 --:--:-- 6635k
# chmod +x /usr/local/bin/docker-compose
```

La versión actual a la hora de escribir este libro es 1.10.0. Una vez descargada la aplicación llamada `docker-compose`, podemos ejecutarla para comprobar el funcionamiento correcto y comprobar la versión instalada.

```
# docker-compose --version
docker-compose version 1.10.0, build 4bd6f1a
```

Como alternativa a esta instalación, es posible utilizar `Pip` (un gestor de paquetes para `Python`) para hacer dicha instalación. El comando a ejecutar es el siguiente (utilizando `pip` con versión 6.0 o superior):

```
# pip install docker-compose
Collecting docker-compose
  Downloading docker_compose-1.10.0-py2.py3-none-any.whl (81kB)
    100% |████████████████████████████████| 81kB 2.3MB/s
  [omitido]

Installing collected packages: six, backports.ssl-match-hostname, websocket-client, ipaddress, docker-pycreds, requests, docker, functools32, jsonschema, texttable, cached-property, dockerpty, enum34, docopt, colorama, PyYAML, docker-compose
Successfully installed PyYAML-3.12 backports.ssl-match-hostname-3.5.0.1 cached-property-1.3.0 colorama-0.3.7 docker-2.0.2 docker-compose-1.10.0 docker-pycreds-0.2.2 dockerpty-0.4.1 docopt-0.6.2 enum34-1.1.6 functools32-3.2.3.post2 ipaddress-1.0.18 jsonschema-2.5.1 requests-2.11.1 six-1.10.0 texttable-0.8.7 websocket-client-0.40.0
# docker-compose --version
docker-compose version 1.10.0, build 4bd6f1a
```

La desinstalación es igual de simple que la instalación que hemos visto:

- `rm /usr/local/bin/docker-compose` -> En el caso de instalación descargando el fichero con `curl`.
- `pip uninstall docker-compose` -> Desinstalar utilizando la utilidad `pip`.

PRIMEROS PASOS

Una vez instalada la utilidad `docker-compose`, podemos crear nuestra primera plantilla para crear un simple *contenedor* basado en una imagen que definiremos con el fichero `Dockerfile`. Para ello dentro de un nuevo directorio dispondremos del siguiente contenido:

- `Dockerfile`: fichero que contendrá la definición de una imagen a utilizar por el *contenedor* que ejecutaremos.
- `docker-compose.yml`: plantilla que definirá el *contenedor* a crear basado en una *imagen* y distintos parámetros para el mismo.
- `www`: directorio contenido en un fichero llamado `index.html`, este directorio será utilizado como contenido para el *volumen* a usar en `/var/www/html`.

El contenido que utilizaremos en el fichero `Dockerfile` es el siguiente y contiene una sintaxis descrita anteriormente:

```
FROM debian:latest
MAINTAINER Alberto Gonzalez <alberto@oforte.net>
# Instalar apache2 y configurar locales
RUN apt-get update && apt-get install -y locales locales-all apache2
RUN locale-gen es_ES.UTF-8
EXPOSE 80
# Variables para Apache 2
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_LOG_DIR /var/log/apache
RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR
# Contenido web
VOLUME /var/www/html/
CMD ["/usr/sbin/apache2", "-D", "BACKGROUND"]
```

El contenido, del que describiremos la sintaxis en este capítulo, para la plantilla de Docker Compose dentro del fichero *docker-compose.yml*, es el siguiente:

```
version: '2'
services:
  web:
    build: .
    ports:
      - "8888:80"
    volumes:
      - ./www/:/var/www/html/
```

Una vez definidos los ficheros, podemos utilizar **docker-compose** con la acción **up** para leer la plantilla y ejecutar los servicios (*contenedores*) definidos.

```
# docker-compose up
Creating network "docker_default" with the default driver
Building web
Step 1 : FROM debian:latest
--> 19134a8202e7
Step 2 : MAINTAINER Alberto Gonzalez <alberto@oforte.net>
--> Running in 41839aea0a24
--> 5577d25d72f2
Removing intermediate container 41839aea0a24
Step 3 : RUN apt-get update && apt-get install -y locales locales-all apache2
--> Running in 595c618e0c44
[...]
Setting up apache2 (2.4.10-10+deb8u7) ...
[...]
--> 87cd58581596
Removing intermediate container 595c618e0c44
Step 4 : RUN locale-gen es_ES.UTF-8
--> Running in 37ae296b1848
Generating locales (this might take a while)...
Generation complete.
--> 660c95842087
Removing intermediate container 37ae296b1848
Step 5 : EXPOSE 80
--> Running in 612911b39369
--> d782a3c3841c
Removing intermediate container 612911b39369
Step 6 : ENV APACHE_RUN_USER www-data
```

```

---> Running in 291ea72cf43
---> 9b0b7d140912
Removing intermediate container 291ea72cf43
[omitido]
Removing intermediate container cf52c7932a4f
Step 12 : RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR
---> Running in dc530b8874bb
---> e13fb39a4e01
Removing intermediate container dc530b8874bb
Step 13 : VOLUME /var/www/html/
---> Running in baab82e79405
---> 98288f664382
Removing intermediate container baab82e79405
Step 14 : CMD /usr/sbin/apache2 -D FOREGROUND
---> Running in 4a44f7be9290
---> 8cc87d31e23a
Removing intermediate container 4a44f7be9290
Successfully built 8cc87d31e23a
Creating docker_web_1
Attaching to docker_web_1
web_1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name,
using 172.18.0.2. Set the 'ServerName' directive globally to suppress this message

```

Observamos que se ha creado y ejecutado un contenedor llamado `docker_web_1`. Desde otra terminal, podemos acceder al puerto 8888 (puerto definido en nuestro `docker-compose.yml` para conectarnos al puerto 80) para comprobar el funcionamiento del contenedor.

```
# curl http://localhost:8888
Prueba Docker Compose
```

Podemos comprobar a través de `ps` y de `inspect`, que la plantilla utilizada para Compose ha sido aplicada correctamente al contenedor recién creado.

```
# docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
24c23e71897e docker_web "/usr/sbin/apache2 -D" 8 min ago Up 8 min 0.0.0.0:8888->80/tcp dock-er_web_1
# docker inspect -f "[.HostConfig.Binds ]" docker_web_1
[/root/docker/www:/var/www/html:rw]
```

Podemos detener la ejecución del contenedor recién creado y del proceso de `docker-compose` pulsando `Control-C`.

ACCIONES BÁSICAS

Hemos visto que con la acción `up`, `docker-compose` leerá la plantilla (por defecto con nombre `docker-compose.yml`) y ejecutará los contenedores especificados.

Las opciones globales que podemos especificar a las acciones son las siguientes:

- `-f fichero`: especifica un fichero diferente al de por defecto (`docker-compose.yml`).
- `-p proyecto`: especifica un nombre de proyecto diferente al de por defecto (nombre del directorio).
- `--verbose`: muestra más información realizando las tareas.

La lista de acciones disponibles:

up [opciones] [servicio]

Inicia los servicios definidos creando los contenedores previamente si fuese necesario. Las opciones que podemos especificar a esta acción son:

- `-d`: modo en *segundo plano*, mostrará el nombre de los contenedores creados.
- `--force-recreate`: por defecto, si algún contenedor definido existe y no ha sido modificado no se recreará. Con esta opción forzará la recreación.
- `--no-recreate`: no volverá a crear ningún contenedor incluso si la imagen ha sido modificada.
- `--no-build`: no creará las imágenes específicas incluso si no existen.
- `--abort-on-container-exit`: si alguno de los contenedores es detenido, detendrá todos los demás. No compatible con la opción `-d`.

En nuestro ejemplo anterior, veíamos que sin utilizar la opción `-d` ejecutaría la aplicación en primer plano y nos mostrará todo el proceso. En el caso de utilizarlo, después de crear y ejecutar los contenedores nos devolverá el acceso a la consola.

ps [servicio]

Esta acción sirve para listar el estado e información de los servicios definidos en la plantilla. La información mostrada es la siguiente:

```
# docker-compose ps
      Name        Command    State    Ports
-----+-----+-----+-----+
  docker_web_1   /usr/sbin/apache2 -D FOREG ... Up      0.0.0.0:8888->80/tcp
```

down [opciones] [servicio]

En el caso de querer detener los servicios y eliminar los contenedores y las redes asociadas, utilizaremos down. Las opciones son:

- **--rmi tipo**: elimina las imágenes asociadas, indicando *all* eliminará todas las imágenes y con *local* solo las imágenes sin nombre especificado.
- **-v (--volumes)**: elimina los volúmenes asociados a los servicios.

Ejemplo de la eliminación del contenedor, imagen y red previamente creados:

```
# docker-compose down --rmi all
Stopping docker_web_1... done
Removing docker_web_1 ... done
Removing network docker_default
Removing image docker_web
```

start [servicio]

Inicia los servicios definidos dentro de la plantilla o únicamente el servicio especificado como argumento, en el caso de que estén detenidos.

```
# docker-compose start
Starting docker_web_1
```

stop [-t timeout] [servicio]

Detiene los servicios definidos dentro de la plantilla o únicamente el servicio especificado como argumento, en el caso de que estén en ejecución.

```
# docker-compose stop
Stopping docker_web_1 ... done
```

SINTAXIS PLANTILLA

Antes de continuar con el resto de acciones, debemos profundizar sobre la sintaxis en la plantilla del fichero *docker-compose.yml*. El formato utilizado es *YAML* (*Yet Another Markup Language*) cuya sintaxis y legibilidad son muy fáciles de entender y aprender.

Como hemos descrito anteriormente, la plantilla definirá servicios (*contenedores*), redes y *volumenes*. La extensión del fichero puede ser *yml* o *yaml*. Las instrucciones más comunes que se definen son las siguientes:

version: versión

Indica la versión del formato a utilizar. Las versiones posibles son:

- 1: La versión histórica. Con ella no se puede definir *volumenes*, *redes* o incluir argumentos para crear las *imágenes*.
- 2: En esta versión todos los servicios deben ser declarados bajo la instrucción *services*. Las *redes* pueden ser especificadas bajo la instrucción *networks*.
- 2.1: Nuevos parámetros para la versión de Docker Engine 1.12.0 y posteriores.
- 3: La versión recomendada en nuevas versiones al ser retrocompatible. Incluye soporte para Docker Swarm.

Ejemplo:

- **version:** '3'

build: directorio

Especifica el directorio donde se aloja el fichero *Dockerfile* (nombre por defecto), para crear la imagen basada en dicho fichero. Ejemplo:

- **build:** ./web/

context: directorio o url

Similar a *build*, pero solo compatible versión 2 y posterior, indica el directorio o una dirección a un repositorio *git* que contenga el fichero *Dockerfile*. Ejemplo:

- **build: git@servidor:/proyecto/**

dockerfile: fichero

Especifica un nombre fichero alternativo al por defecto *Dockerfile*. Ejemplo:

- **dockerfile: Dockerfile-dev**

args: argumentos

Especifica una lista de argumentos a pasar para crear la imagen basada en el fichero *Dockerfile*. El fichero *Dockerfile* deberá incluir las instrucciones **ARG**. Ejemplo:

args:

usuario: root

configuracion: dev.conf

command: comando

Indica un *comando* distinto para la imagen distinto al definido en el fichero *Dockerfile*. Esta expresión se usa para tener una imagen para todos los entornos, pero el fichero plantilla definirá un comando diferente para cada uno de ellos. Ejemplos:

- **command: /usr/sbin/apache2 -D FOREGROUND**
- **command: ["/usr/sbin/apache2", "-D", "BACKGROUND"]**

entrypoint: comando

Indica el punto de entrada para la imagen distinto al definido en el fichero *Dockerfile*. Esta expresión se usa para tener una imagen para todos los entornos, pero el fichero plantilla definirá un punto de entrada para cada uno de ellos. Ejemplos:

- **entrypoint: /usr/bin/comando.sh**

container_name: nombre

Especifica el nombre para el *contenedor* a crear para el servicio. Por defecto, dicho nombre es autogenerado por el nombre del proyecto. No es recomendado especificar un nombre personalizado ya que no podremos escalar los servicios (explicando posteriormente la acción [scale](#)).

depends_on: servicios

Cuando desplegamos varios servicios (*contenedores*), es normal que uno de ellos dependa de los otros. Por ejemplo, un servidor web depende de un servidor de base de datos para su funcionamiento y queremos que primero se ejecute este último. Ejemplo:

```
version: '2'
services:
  web:
    build: .
    depends_on:
      - db
  db:
    image: mariadb
```

environment: valores

Especifica una lista de variables de entorno a pasar en la creación de la imagen. Ejemplo:

```
environment:
  APACHE_RUN_USER: www-data
  APACHE_RUN_GROUP: www-data
  APACHE_PID_FILE: /var/run/apache2.pid
  APACHE_RUN_DIR: /var/run/apache2
  APACHE_LOCK_DIR: /var/lock/apache2
  APACHE_LOG_DIR: /var/log/apache
```

env_file: fichero

Especifica un fichero que contiene las variables de entorno a pasar a la creación de la imagen. Ejemplo:

- **env_file:** dev.env

expose: puertos

Lista de puertos a exponer, solo para la creación de imagen. Ejemplos:

- **expose: 443**
- **expose: 80**

imagen: imagen

Especifica la imagen a utilizar en caso de no utilizar *build* o *context*. Si la imagen no existe, se intentará descargar desde el repositorio oficial o el repositorio definido. Ejemplo:

- **image: httpd**
- **image: 192.168.56.104:5000/debianapache2**

labels: etiquetas

Lista de etiquetas a establecer al contenedor. Ejemplo:

```
labels:  
  entorno: dev  
  version: 0.1
```

logging: configuración

Especifica el *driver* a utilizar para los registros de los contenedores y las opciones para dicho *driver*. Ejemplo:

```
logging:  
  driver: syslog  
  options:  
    syslog-address: "tcp://192.168.56.101:123"
```

network_mode: red

Especifica el modo de red a utilizar para los servicios. Ejemplo:

- **network_mode: "bridge"**

networks: redes

Especifica la lista de redes a unirse para los servicios.

```
services:
  web:
    networks:
      - database
      - backup
```

ports: puertos

Lista de puertos a exponer, en este caso se expondrán al servidor donde ejecuta los servicios. Ejemplos:

- *ports: 443*
- *ports: 80*
- *ports: 8888:80*

volumes: volúmenes

Lista de volúmenes o puntos de montaje a utilizar. Ejemplo:

```
volumes:
  - ./www/:/var/www/html/
  - ./config/:/etc/apache2/
```

EJEMPLO

Esta plantilla sirve para lanzar un contenedor con un servidor web ejecutando el conocido blog *wordpress* que se conecta a otro contenedor que tiene una base de datos *MySQL*. A través de la expresión **depends_on**, el servicio del servidor web esperará por el servicio de base de datos.

```
version: "2"

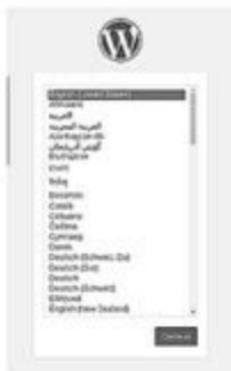
services:
  db:
    image: mysql:5.7
```

```
volumes:  
- db_data:/var/lib/mysql  
environment:  
MYSQL_ROOT_PASSWORD: wordpress  
MYSQL_DATABASE: wordpress  
MYSQL_USER: wordpress  
MYSQL_PASSWORD: wordpress  
  
wordpress:  
depends_on:  
- db  
image: wordpress:latest  
ports:  
- "8000:80"  
environment:  
WORDPRESS_DB_HOST: db:3306  
WORDPRESS_DB_PASSWORD: wordpress  
volumes:  
db_data:1
```

Ejecutando `docker-compose up -d`, ambos servicios se ejecutarán y podremos acceder a través del puerto 8000.

```
# docker-compose up -d  
Creating network "wordpress_default" with the default driver  
Creating wordpress_db_1  
Creating wordpress_wordpress_1
```

Accediendo al puerto 8000 de la dirección IP del servidor Docker que aloja los contenedores, observaremos la página de configuración del blog.



ACCIONES

Una vez vistas las acciones básicas anteriormente y las expresiones que pueden ser incluidas dentro de la plantilla, podemos ver el resto de acciones que nos proporciona `docker-compose`.

`pause [servicio]`

Pausa todos los procesos de todos los servicios o del servicio especificado.

```
# docker-compose pause
Pausing wordpress_db_1 ... done
Pausing wordpress_wordpress_1 ... done
```

Podemos ver el estado de los contenedores utilizando `docker ps`:

docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
826cdbfa8212 wordpress:latest "docker-entrypoint..." 24 hours ago Up 43 seconds (Paused) 0.0.0.0:8000->80/tcp wordpress_wordpress_1
ef92a9ac31da mysql:5.7 "docker-entrypoint..." 24 hours ago Up 43 seconds (Paused) 3306/tcp wordpress_db_1

unpause [servicio]

Reanuda todos los procesos de todos los servicios o del servicio especificado.

```
# docker-compose unpause
Unpausing wordpress_wordpress_1 ... done
Unpausing wordpress_db_1 ... done
```

build [opciones] [servicio]

Crea o recrea las imágenes definidas en la plantilla basándose en los ficheros *Dockerfile*. Las opciones que se pueden especificar son las siguientes:

- **--force-rm**: siempre elimina los contenedores intermedios.
- **--no-cache**: no utiliza la caché al crear las imágenes.
- **--pull**: siempre intenta obtener una nueva versión de la imagen definida en el fichero *Dockerfile*.

Ejemplo:

```
# docker-compose build
Building web
Step 1 : FROM debian:latest
--> 19134a8202e7
Step 2 : MAINTAINER Alberto Gonzalez <alberto@oforte.net>
--> Using cache
--> f807690442fb
Step 3 : RUN apt-get update && apt-get install -y locales locales-all apache2
[...]
Successfully built b53315093fd1
# docker-compose build --no-cache
Building web
Step 1 : FROM debian:latest
--> 19134a8202e7
Step 2 : MAINTAINER Alberto Gonzalez <alberto@oforte.net>
--> Running in a0f3451b56b9
--> 1a889d012789
Removing intermediate container a0f3451b56b9
Step 3 : RUN apt-get update && apt-get install -y locales locales-all apache2
--> Running in 4fcf360f47ed
```

```
Get:1 http://security.debian.org jessie/updates inRelease [63.1 kB]
[omitted]
```

Observamos que **docker-compose** siempre intenta optimizar el proceso utilizando *caché* cuando sea posible, especificando **--no-cache** hará la creación de la imagen sin tener en cuenta la *caché*. En el caso que la plantilla defina utilizar una imagen del repositorio y no crearla, ignorará la acción ***build*** y se mostrará el siguiente mensaje:

```
# docker-compose build
db uses an image, skipping
wordpress uses an image, skipping
```

***config* [opciones]**

Comprueba la sintaxis de la plantilla en búsqueda de errores de sintaxis. Por defecto, si la sintaxis es correcta mostrará el contenido de la misma. Opciones:

- **-q**: solo valida la plantilla, no muestra nada.
- **--services**: muestra el nombre de los servicios, uno por linea.

Ejemplo:

```
# docker-compose config
networks: {}
services:
  db:
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_PASSWORD: wordpress
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_USER: wordpress
    image: mysql:5.7
    restart: always
    volumes:
      - db_data:/var/lib/mysql:rw
  wordpress:
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: db:3306
```

```

WORDPRESS_DB_PASSWORD: wordpress
image: wordpress:latest
ports:
- 8000:80
restart: always
version: '2.0'
volumes:
db_data: {}
# docker-compose config --services
db
wordpress
# docker-compose config -q
# echo $?
0

```

En caso de que la plantilla contenga algún error, se nos mostrará en pantalla qué es incorrecto. En este ejemplo hemos cambiado la instrucción *volumenes* por una instrucción incorrecta llamada *volúmenes*:

```

# docker-compose config
ERROR: The Compose file './docker-compose.yml' is invalid because:
Unsupported config option for services.db: 'volúmenes'

```

create [opciones] [servicio]

Crea los servicios definidos en la plantilla pero no los ejecuta y tampoco crea las redes necesarias. Las opciones posibles a especificar:

- **--force-recreate:** recrea las *ímágenes* incluso si la configuración o los ficheros *Dockerfile* no han sido modificados.
- **--no-recreate:** no recrea las *ímágenes* incluso si la configuración o los ficheros *Dockerfile* no han sido modificados.
- **--no-build:** no crea las *ímágenes* incluso si no existen.
- **--build:** crea las *ímágenes* antes de crear los *contenedores*.

Ejemplo:

```

# docker-compose create
Creating wordpress_db_1

```

Creating wordpress_wordpress_1***events [opciones] [servicios]***

Esta acción pone en escucha al comando `docker-compose` por eventos eventos en tiempo real generados por los contenedores. La única opción que acepta esta acción es: `--json`, que mostrará los eventos en formato JSON. En el siguiente ejemplo, después de poner a la escucha utilizando `events`, en otra sesión se iniciaron los contenedores que definen los servicios (`docker-compose up`):

docker-compose events

```
2017-01-26 20:44:03.219965 container attach efb688[...] (image=mysql:5.7, name=wordpress_db_1)
2017-01-26 20:44:03.513497 container start efb688[...] (image=mysql:5.7, name=wordpress_db_1)
2017-01-26 20:44:03.532600 container at[...] (image=wordpress:latest,
name=wordpress_wordpress_1)
2017-01-26 20:44:05.751128 container st[...] (image=wordpress:latest,
name=wordpress_wordpress_1)
```

exec [opciones] servicio comando [argumentos]

Al igual que la acción `exec` del comando `docker`, con el comando `docker-compose` podemos ejecutar comandos dentro de los contenedores pero haciendo referencia al servicio definido dentro de la plantilla. Opciones:

- `-d`: ejecuta el comando en segundo plano.
- `--user usuario`: ejecuta el comando con el usuario especificado.
- `--index índice`: en el caso de que el servicio contenga varios contenedores, podemos limitar por el índice.

En el siguiente ejemplo obtenemos la versión dentro del gestor de base de datos:

```
# docker-compose exec db mysql -v --version
mysql Ver 5.7.17 for Linux on x86_64 (MySQL Community Server (GPL))
```

kill [opciones] [servicio]

Detiene a la fuerza uno o varios servicios. La opción que se puede especificar es la siguiente:

- **-s SEÑAL:** indica la señal a enviar a los contenedores, por defecto **SIGKILL**.

Ejemplo:

```
# docker-compose start
Starting db ... done
Starting wordpress ... done
```

logs [opciones] [servicio]

Muestra la salida producida por los contenedores. Las opciones que pueden ser especificadas son las siguientes:

- **--no-color:** muestra la salida sin colores.
- **-f/-follow:** mantiene el proceso en primer plano esperando por nuevos registros.
- **-t/-timestamps:** muestra la hora y el tiempo de cada registro.
- **--tail líneas:** muestra el número de líneas específicas.

Ejemplo:

```
# docker-compose logs --no-color --timestamps --tail 5
Attaching to wordpress_wordpress_1, wordpress_db_1
wordpress_1 | 2017-01-26T20:58:09.753d40129Z MySQL Connection Error: [2002] Connection refused
wordpress_1 | 2017-01-26T20:58:13.015066570Z AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.16.0.3. Set the 'ServerName' directive globally to suppress this message
wordpress_1 | 2017-01-26T20:58:13.043227517Z AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.16.0.3. Set the 'ServerName' directive globally to suppress this message
wordpress_1 | 2017-01-26T20:58:13.062249216Z [Thu Jan 26 20:58:13.061936 2017] [mpm_prefork:notice] [pid 1] AH00063: Apache/2.4.10 (Debian) PHP/5.6.30 configured -- resuming normal operations
wordpress_1 | 2017-01-26T20:58:13.062279961Z [Thu Jan 26 20:58:13.061980 2017] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
db_1   | 2017-01-26T20:58:10.697313832Z 2017-01-26T20:58:10.696916Z 0 [Note] End of list of non-natively partitioned tables
db_1   | 2017-01-26T20:58:10.697347372Z 2017-01-26T20:58:10.697116Z 0 [Note] mysqld: ready for connections.
db_1   | 2017-01-26T20:58:10.697370571Z Version: '5.7.17' socket: '/var/run/mysqld/mysqld.sock' port: 3306
MySQL Community Server (GPL)
db_1   | 2017-01-26T20:59:20.246638311Z 2017-01-26T20:59:20.2451732Z 0 [Note] InnoDB: page_cleaner: 1000ms intended loop took 4736ms. The settings might not be optimal. (flushed=0 and evicted=0, during the time.)
```

```
db_1    | 2017-01-26T21:13:03.101130241Z 2017-01-26T21:13:03.009218Z 0 [Note] InnoDB: page_cleaner: 1000ms intended loop took 4079ms. The settings might not be optimal. (flushed=0 and evicted=0, during the time.)\A
```

port [opciones] servicio puerto

Muestra el puerto y la dirección del servidor Docker en que se redirigen las peticiones al contador. Las opciones son:

- **--protocol tcp o udp:** indica el protocolo.
- **--index indice:** en el caso de múltiples contenedores para un servicio, especifica el índice.

Ejemplo de nuestra aplicación *wordpress*, que escuchaba en el puerto 8000:

```
# docker-compose port wordpress 80
0.0.0.0:8000
```

pull [-ignore-pull-failures] [servicio]

Descarga las imágenes de todos los servicios o el servicio especificado.
Ejemplo:

```
# docker-compose pull
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
Digest:
sha256:79690dd87d68fd4d801e65f5479f8865d572a6c7ac073c9273713a9c633022c5
Status: Image is up to date for mysql:5.7
Pulling wordpress (wordpress:latest)...
latest: Pulling from library/wordpress
Digest:
sha256:9e397d900023f0ca04b8ce3ac61b1b212e8ea1487d78da2dea89a9bda87c70c5
Status: Image is up to date for wordpress:latest
```

push [-ignore-push-failures] [servicio]

Publica las imágenes de todos los servicios o el servicio especificado.
Ejemplo:

Contenido fichero docker-compose.yml

```
version: '2'  
services:  
  web:  
    build: .  
    image: localhost:5000/debianapache2  
    ports:  
      - "8888:80"  
    volumes:  
      - ./www:/var/www/html/
```

Crear imagen y publicarla.

```
# docker-compose build  
Building web  
[..]  
Successfully built dd2ab24d15a2  
# docker-compose push  
Pushing web (localhost:5000/debianapache2:latest)...  
The push refers to repository [localhost:5000/debianapache2]  
0f0b5131f2ed: Pushed  
b15b39a99121: Pushed  
a2ae92ffcd29: Mounted from debian  
latest: digest:  
sha256:a9fd866aae5e4ace97c1b855af2792e7bebc6c5d290045cf183a31762a03a186 size:  
948
```

13

DOCKER CLOUD

Introducción

Docker Cloud proporciona un servicio de repositorio en línea de forma gratuita (limitado a un *nodo* y a un *repositorio privado*). Podemos utilizar el mismo usuario y contraseña que hemos creado para <http://hub.docker.com> o es posible de forma gratuita crear una nueva cuenta (denominado *Docker ID*).

Anteriormente detallábamos que Docker Hub servía para alojar las imágenes. Docker Cloud ofrece estas funciones:

- Desplegar servicios en diferentes proveedores *cloud* (en la nube), las opciones son las siguientes:
 - Amazon Web Services (AWS)
 - Microsoft Azure
 - Digital Ocean
 - Packet
 - SoftLayer
- Desplegar servicios en nuestros propios servidores.
- Autocreación de imágenes.
- Autotest de imágenes.

TERMINOLOGÍA

Cuando trabajamos con Docker Cloud es importante tener claros los siguientes términos:

- **Nodo:** un servidor Linux utilizado para desplegar aplicaciones. Docker Cloud no es un servicio de *hosting*. Este *nodo* puede ser un servidor físico, virtual o un proveedor *cloud*.
- **Cluster de nodos:** grupos de *nodos* del mismo tipo y alojados en el mismo proveedor. Permite escalar (añadir nuevos contenedores) para satisfacer nuevas necesidades.
- **Servicio:** los *servicios* son un grupo lógico de *contenedores* basados en la misma *imagen*, permiten escalar las aplicaciones a través de distintos *nodos*.
- **Stack:** colección de *servicios*. Por ejemplo una colección con un servidor web y un servidor de base de datos.

INSTALAR NUESTRO PROPIO NODO

En este libro solo cubriremos la opción de desplegar servicios en nuestro propio servidor. Para ello debemos instalar **Docker Cloud Agent** en nuestro servidor *Linux* y enlazarlo a nuestra cuenta. Las *distribuciones* en las que ha sido confirmado su funcionamiento son las siguientes:

- *Ubuntu 14.04, 15.04, 15.10*
- *Debian 8*
- *Centos 7*
- *Red Hat Enterprise Linux 7*
- *Fedora 21, 22, 23*

Los pasos para instalarlo son los siguientes:

- Asegurarse de que los puertos 6783/tcp, 6783/udp y 2375/tcp están disponibles para acceder remotamente en el servidor.
- Desde la página de **Docker Cloud** acceder desde el menú de la izquierda a *Nodes*.
- Desde la página de *Node*, hacer clic sobre el botón *Bring your own node*. Aparecerá el siguiente recuadro:

Bring your own Node

Docker Cloud lets you use your own host as a media to run containers. In order to do this, you have to first install the Docker Cloud Agent.

The following Linux distributions are supported:



Ubuntu 14.04,
LTS



Debian 8



CentOS 7



Red Hat Linux
7



Fedora 21, 22

Run the following command in your Linux host to install the Docker Cloud Agent or click [here](#) to learn more.

```
curl -s https://get.cloud.docker.com/ | sudo -H sh -s 57d40a87bba34788b35ba0d32d0f61c7
```

We recommend you open port 2222 in your firewall for Docker Cloud to communicate with the Docker daemon running in the node. For the overlay network to work, you must open port 4743/tcp and 4743/udp.

Waiting for contact from agent

[Close window](#)

Ejecutar el comando del recuadro gris en nuestro servidor.

```
# curl -Ls https://get.cloud.docker.com/ | sudo -H sh -s 57d40a87bba34788b35ba0d32d0f61c7
-> Adding Docker Cloud's GPG key...
gpg: key EF170D1C: public key "Tutum Inc. (tutum) <info@tutum.co>" imported
gpg: Total number processed: 1
gpg:      imported: 1 (RSA: 1)
OK
-> Installing required dependencies...
-> Installing dockercloud-agent...
[...]
-> Configuring dockercloud-agent...
-> Enabling dockercloud-agent to start on boot on systemd...
Synchronizing state for dockercloud-agent.service with sysvinit using update-rc.d...
Executing /usr/sbin/update-rc.d dockercloud-agent defaults
Executing /usr/sbin/update-rc.d dockercloud-agent enable
-> Starting dockercloud-agent service...
-> Done!
```

Docker Cloud Agent installed successfully

You can now deploy containers to this node using Docker Cloud

Podemos observar el proceso de configuración después de instalar en el siguiente fichero: `/var/log/dockercloud/agent.log`. Después de unos minutos, se completará la configuración desde el portal de Docker Cloud y se visualizará en el recuadro un mensaje similar al siguiente:

We recommend you open incoming port 2375 in your firewall for Docker Cloud to communicate with the Docker daemon running in the node. For the overlay network to work, you must open port 6783/tcp and 6784/tcp.

- ✓ Node 25d4b5d5-2c09-4320-b5e4-3d5c352cccd73.node.dockerapp.io detected

[Close window](#)

En la lista de *nodos* nos aparecerá nuestro servidor y haciendo clic sobre él podemos ver más detalles:



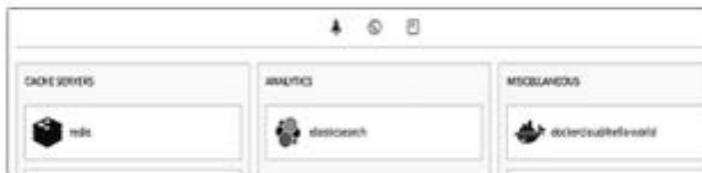
En nuestro servidor podemos ver que durante la instalación y configuración de **Docker Cloud Agent** se han descargado imágenes y se han ejecutado nuevos contenedores:

# docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4803f94040bc	dockercloud/cleanup:latest	"./run.sh"	3 minutes ago	Up 3 minutes	cleanup-13819.227d4ab2	
2c5b4fe0b33e	dockercloud/logrotate:latest	"crond -l"	4 minutes ago	Up 3 minutes	logrotate-	
35671.993dc2c2						
bbdfcfb8603f	dockercloud/events:latest	"/events"	4 minutes ago	Up 4 minutes	events-29223.9bd5ef95	
646cb4fd6dba	dockercloud/ntpd:latest	"./run.sh"	4 minutes ago	Up 4 minutes	ntpd-74379.51acd42c	
29cbc775c961	dockercloud/network:[...]	"./run.sh"	4 minutes ago	Up 4 minutes	weave-58293.163daee	
b375c4a163fb	weaveworks/plugin:1.6.2	"./ho[...]"	8 minutes ago	Up 4 minutes	weaveplugin	
9a549b39e514	weaveworks/weaveexec:1.6.2	"./ho[...]"	8 minutes ago	Up 4 minutes	weaveproxy	
2aa4ef31490b	weaveworks/weave:1.6.2	"./ho[...]"	8 minutes ago	Up 4 minutes	weave	

DESPLEGAR UN SERVICIO

Para desplegar un *servicio* a través del portal de Docker Cloud seguiremos estos pasos:

- En el menú de la izquierda pulsamos **Services**.
- Posteriormente pulsamos **Create** para crear un nuevo servicio. Tendremos tres opciones:
 - *Jumpstarts*: lista de aplicaciones populares.
 - *Imágenes públicas*: búsqueda dentro de **Docker Hub** u otros registros especificados en su dirección.
 - *Repositorios propios*: si poseemos un *repositorio* propio dentro de **Docker Hub** podemos elegir nuestras *imágenes*.



- Una vez elegida alguna opción de la lista, por ejemplo *dockercloud/helloworld*, debemos configurar distintas opciones (las más comunes):
 - Versión de la *imagen* (ejemplo: *latest*, *1.0*, etc.).
 - *Nombre del servicio*.
 - Número de contenedores a desplegar.
 - Estrategia para desplegar los contenedores:
 - Sobre el nodo más vacío.
 - Alta disponibilidad.
 - En todos los nodos.

- Si debe ser reiniciado automáticamente (*autorestart*).
- Si debe ser destruido al ser parado (*autodestroy*).
- Red a utilizar (*bridge, host, container*).
- Configuración del *contenedor*:
- Punto de entrada (*entrypoint*).
- Comando.
- Límite de memoria y CPU.
- Puertos a publicar. En nuestro ejemplo publicaremos el puerto 80:

Ports

Container port	Protocol	Published	Node port
80	tcp	<input checked="" type="checkbox"/>	dynamic

- Variables de entorno.
- Volúmenes.

Una vez configurado con los valores deseados, pulsamos sobre **Create & Deploy** en el menú de la derecha. Podemos pulsar sobre **Timeline** para ver el progreso:

Timeline

```

    Service Start
    -----
    CREATED: Fri, 27 Jan 2017 21:21:25 +0000
    FINISHED: Fri, 27 Jan 2017 21:21:36 +0000
    USER: oforte
    DURATION: a few seconds
    LOCATION: Berlin, Germany
    IP: 95.90.235.182
    USER-AGENT: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36
    REQUEST: POST /api/app/v1/service/start?node=d976d-43e-a57c-1540dcfa1676tar

Preparing to deploy...
Inspecting dockercloud/hello-world:latest image in the registry
Dropping up to date in our database
Deploying...
Choosing best nodes...
Candidate node 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.dockerapp.io. Status: Deployed. Total num containers: 0. No
Choosing node 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.dockerapp.io because of the deployment strategy
Hello-world-d976d43e-a57c-1540dcfa1676tar: Deploying in 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.dockerapp.io
Hello-world-d976d43e-a57c-1540dcfa1676tar: Pulling image dockercloud/hello-world:latest in 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Downloading layer e1f1804a17941
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Downloading layer 5b03ab56e141
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Downloading layer 9418044cf793
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Downloading layer 624ee5173281
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Downloading layer 989bc12c9284
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Creating in 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.dockerapp.io with docker name hello-world-d976d43e-a57c-1540dcfa1676tar
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Attaching container to overlay network dockercloud with ip 16.1.0.2
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Starting task dockercloud/hello-world:latest@14900537082642783a97a133d7fb
Hello-world-d976d43e-a57c-1540dcfa1676tar:1: Waiting on 256d05d7-3c99-4a28-b5e4-3d5c352cc073 node.dockerapp.io
Service Start action on 'Hello-world-d976d43e-a57c-1540dcfa1676tar' has finished successfully
  
```

Una vez ejecutado nuestro **contenedor**, podremos acceder a él a través del puerto que hemos publicado. Para ello hacemos **clic** sobre **General** y observamos el apartado **Endpoints** para obtener la dirección para acceder.

Endpoints

Container Endpoints

hello-world-d976d43e-a57c-1540dcfa1676tar	tcp/32769	http://hello-world-d976d43e-a57c-1540dcfa1676tar:32769
---	-----------	--

Accediendo a la dirección indicada, pulsando sobre ella visualizaremos la siguiente página web:



Hello world!

My hostname is hello-world-d9792f6a-1

Hemos visto que es muy simple desplegar aplicaciones (*servicios*) a través de Docker Cloud.

CREAR UN STACK

Para definir un *stack* es necesario hacerlo manualmente con sintaxis YAML a través de una sintaxis muy similar a la vista para **docker-compose**. En nuestro ejemplo de desplegar *wordpress* con MySQL utilizaremos la siguiente plantilla:

```
db:  
image: mysql:5.7  
environment:  
  MYSQL_ROOT_PASSWORD: wordpress  
  MYSQL_DATABASE: wordpress  
  MYSQL_USER: wordpress  
  MYSQL_PASSWORD: wordpress  
  
wordpress:  
links:  
  - db  
image: wordpress:latest  
ports:  
  - "8000:80"  
environment:  
  WORDPRESS_DB_HOST: db:3306  
  WORDPRESS_DB_PASSWORD: wordpress
```

En el menú de la izquierda hacemos clic sobre **Stacks** y sobre **Create**. En el siguiente formulario debemos llenar el nombre y la plantilla.



Pulsando sobre **Create & Deploy**, se desplegarán ambos contenedores y podremos acceder a *wordpress* a través de una dirección generada por **Docker Cloud**. Una vez finalizada la creación, se mostrará la siguiente página y la dirección *Endpoint* para acceder:

Accediendo a la dirección especificada en *Endpoints*, nos mostrará la página web de configuración de *wordpress*.



SINTAXIS PLANTILLA STACK

Como hemos visto la sintaxis de la plantilla es similar a la utilizada por **docker-compose**, las instrucciones más comunes que podemos utilizar son las siguientes:

- **image: nombre** -> Indica la imagen a utilizar. Es requerido.
- **autodestroy: [no / on-success / always]** -> Indica que si el contenedor es detenido debería ser destruido. Por defecto *no*.
- **autoredeploy: [true / false]** -> Indica si el contenedor debe ser recreado si la imagen indicada es más actual que la utilizada.
- **command: comando** -> Utiliza dicho comando en vez del de por defecto.
- **dns: servidores** -> Especifica servidores DNS a utilizar.
- **dns_search: lista** -> Especifica la búsqueda de dominios DNS a utilizar.
- **environment: variables** -> Lista de variables de entorno.
- **expose: puertos** -> Los puertos a exponer pero sin publicarlos.
- **labels: etiquetas** -> Lista de etiquetas a utilizar.
- **ports: puertos** -> Los puertos a publicar para que sean accesibles.

- **restart:** [no | on-failure | always] -> Indica si un contenedor debe ser reiniciado automáticamente.
 - **volumes:** volúmenes -> Lista de volúmenes a utilizar, con formato:
 - *directorio_contenedor* (/var/www/html)
 - *directorio_servidor:directorio_contenedor* (/web:/var/www/html)
 - *directorio_servidor:directorio_contenedor:ro* (/apps:/apps/:ro) -> Para sólo lectura

ESCALAR UN SERVICIO

Una de las ventajas de los *microservicios* es que podemos desplegar nuevos contenedores para satisfacer las necesidades de carga de nuestra aplicación. Con **Docker Cloud** es muy simple escalar un servicio, solo tenemos que acceder a él (por ejemplo a *hello-word*) e indicar el número de contenedores que queremos a mayores a través del deslizador de la parte superior derecha.



En esta aplicación simple no hay acceso a una base de datos, pero podemos fácilmente observar que utilizando *escalar* en el caso de un servidor web con muchos usuarios, podremos ejecutar nuevos contenedores con nuestra web para soportar la demanda de usuarios y todos esos contenedores accederían a una base de datos común.

TRIGGERS (DISPARADORES)

Un trigger es un disparador para realizar distintas opciones. Dentro de Docker Cloud existen dos triggers:

- *Scale Up*: escala un servicio.
- *Redeploy*: vuelve a desplegar el servicio.

Es posible desde el portal de Docker Cloud crear un trigger específico a un servicio, creará una dirección web especial para que cuando sea llamada (una solicitud *POST*) ejecute el disparador específico. Al acceder a un servicio veremos una sección llamada *Triggers* donde podemos definir nuestro *disparador*:

The screenshot shows the 'Triggers' section of the Docker Cloud interface. A 'SCALE UP' trigger is listed with the name 'newsthook'. The URL for this trigger is `https://cloud.docker.com/api/app/v1/service/79a810ded-8810-4115-ad5b-926516211da9/trigger?serviceId=c1-51ea-42da-a6ff-41453a2e8a48/call`. There is also a '+' button to add more triggers.

Una vez creado nuestro trigger aparecerá la dirección web especial a la que puedes hacer una llamada para *escalar* este servicio desplegando más contenedores. Podemos utilizar el siguiente comando para ello:

```
# curl -XPOST https://cloud.docker.com/api/app/v1/service/79a810ded-8810-4115-ad5b-926516211da9/trigger/6ea5d7c1-51ea-42da-a6ff-41453a2e8a48/call/
```

En el *Timeline* del servicio podemos observar que después de la llamada a la dirección autogenerada, la creación de un nuevo contenedor:

Service Taskflow (Trigger)	
CREATED	Sat, 26 Jan 2011 16:57:24 +0000
FINISHED	Sat, 26 Jan 2011 16:57:31 +0000
USER	efrata
DURATION	a few seconds.
LOCATION	Paris, France
PI	01.121.79.547
USER AGENT	curl/7.26.0
REQUEST	POST /appengine/mailboxes/Postbox/submit?trigger=submit&id=1130-4200-000-414511...

La utilización de triggers es muy útil en combinación con sistemas de monitorización. Por ejemplo si monitorizamos el número de usuarios que están conectados a nuestros servidores web y sobrepasan un número específico, podemos hacer la llamada a la dirección generada por Docker Cloud desde el servicio de monitorización para escalar el servicio y poder complacer la demanda de usuarios.

REPOSITORIOS

Desde Docker Cloud podemos acceder a lista de nuestras imágenes alojadas en Docker Hub y además crear repositorios (donde alojar una imagen) de tipo público o privado. En el menú de la izquierda pulsando sobre *Repositories*, se nos mostrará nuestros repositorios e imágenes. Pulsando sobre **Create** aparecerá el siguiente formulario:

En él especificaremos la siguiente información:

- Nombre.
 - Descripción.
 - Visibilidad.
 - La posibilidad de crear imágenes automáticamente después de cada publicación en GIT (sistema de control de versiones).

Una vez creado nuestro *repositorio*, aparecerá la información de él y cómo subir una imagen al mismo. Ejemplo de publicación de *Imagen*:

Create Repository

oforte / Name _____

Description _____

Visibility

Using 0 of 1 private repositories. Get access

Public 

Public repositories appear in Docker Store search results

Private 

Only you can see private repositories

Build Settings (optional)

Autobuild triggers a new build with every git push to your source code repository `Lauco.muck`



Disconnected



Disconnected

[Cancel](#)

[Create](#)

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: `usuaria`

Password:

Login Succeeded

docker push oforte/libro

The push refers to a repository [`docker.io/oforte/libro`]

`a2ae92ffcd29: Pushed`

`latest: digest: sha256:abbe80c8c87b7e1f652fe5e99ff1799cdf9e0878c7009035afe1bccac129cad8 size: 529`

Y la información aparecerá en el portal de Docker Cloud:

CONECTAR A GITHUB

Github es la plataforma en línea más importante para alojar código fuente de aplicaciones. *Git* es un sistema de versión de controles, que permite controlar los cambios realizados en los ficheros. Nosotros podemos alojar nuestras plantillas y los ficheros necesarios para generar imágenes en esta plataforma y es posible configurar que en cada cambio realizado se genere la *imagen*.

Para ello debemos enlazar nuestra cuenta de **Docker Cloud** con *Github*, para ello en el menú de la izquierda elegimos **Cloud Settings** y buscamos la sección **Source providers**.

Source providers	
Provider	Account
GitHub	No account linked
Bitbucket	No account linked

Pulsando sobre el ícono gris de la derecha referente a Github, nos redireccionará para permitir el enlace de nuestras cuentas.

Authorize application

Docker Cloud Builder by @docker would like permission to access your account



Review permissions

	Repositories
	Public and private
AUTORIZAR APLICACIÓN	

Docker Cloud Builder	
No description	
Visit application's website	
Learn more about OAuth	

Una vez autorizada la aplicación, la sección de *Source Providers* cambiará indicando que la cuenta está enlazada:

Source providers

Provider	Account		
GitHub	softant		
Bitbucket	No account linked		

Ahora podemos crear un nuevo *repositorio* y especificar las opciones de generar la imagen automáticamente.



En el repositorio especificado, se alojan los siguientes ficheros:

- *Dockerfile* -> Plantilla para crear la imagen (en el caso de que esté en otro directorio, hay que indicarlo dentro de la sección *Click here to customize the build settings*).
- *index.html* -> Fichero estático a ser utilizado por la plantilla.

Una vez pulsado sobre **Create**, aparecerá la información básica y un menú superior con las distintas secciones (*General*, *Tags*, *Builds*, *Timeline* y *Settings*). Pulsando sobre *Builds* y posteriormente **Configure Automated Builds** podemos configurar los siguientes valores:

- *Repositorio* (especificado anteriormente en la creación).
- Dónde crear la imagen:
 - En nuestro propio nodo.
 - Utilizando la infraestructura de **Docker Cloud**, actualmente de forma gratuita.
- Autotest -> Probar después de cada publicación en el repositorio. Las opciones son:
 - Off -> No se autoprueba.
 - Internal Pull Requests -> Probar en publicaciones internas (mismo repositorio).

- Internal and External Pull Requests -> Probar en publicaciones internas y externas (repositorios externos).
 - *Build Rules* -> Esta sección indica las reglas para autogenerar las imágenes.



Después de pulsar sobre **Save and Build**, se guardarán las opciones y empezará la generación de la imagen basada en el repositorio. Pulsando sobre *Timeline* podemos ver dicho progreso:

Después de cada publicación (*push*) en el repositorio *git*, se autogenerará la *imagen*.

AUTOTEST

Es posible configurar distintas pruebas a realizar después de la generación de la imagen para comprobar su funcionamiento. Para ello en el repositorio debe existir un fichero llamado `docker-compose test.yml` con un contenido similar al siguiente:



```
sut:
build: .
command: realizar_pruebas.sh
```

En el ejemplo anterior se creaba una imagen con un servidor web, la prueba que incluiremos será comprobar si es posible iniciar el servicio de *apache2*.

Contenido del fichero Dockerfile

```
FROM debian:latest
MAINTAINER Alberto Gonzalez <alberto@oforte.net>
RUN apt-get update && apt-get install -y locales locales-all apache2 curl
RUN locale-gen es_ES.UTF-8
EXPOSE 80
VOLUME /var/www
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_LOG_DIR /var/log/apache
RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR
VOLUME /var/www/html/
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Contenido fichero docker-compose.test.yml

```
sut:
build: .
command: /etc/init.d/apache2 start
```

Al añadir el fichero al repositorio, se generará automáticamente la imagen y se harán las pruebas definidas. Podemos ver en los registros dentro de *Timeline*:

```
Starting Test in docker-compose.test.yml...
Building sut
Step 1 : FROM debian:latest
--> e5599115b6a6
[...]
Step 15 : CMD /usr/sbin/apache2 -D FOREGROUND
--> Using cache
--> 49a88e653fd1
Successfully built 49a88e653fd1
Creating bcj9c6vcm2yqncy2puctkxj_sut_1
```

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
172.17.0.3. Set the 'ServerName' directive globally to suppress this message
Starting web server: apache2.
Going to remove bcj9c6vcm2yqncy2puctkxj_sut_1
Removing bcj9c6vcm2yqncy2puctkxj_sut_1...
Removing bcj9c6vcm2yqncy2puctkxj_sut_1 ... done
Tests in docker-compose.test.yml succeeded
Pushing index.docker.io/aforte/debianapache2:latest...
Done!
time="2017-01-28T11:16:33Z" level=debug msg="saving: 4.0 kB"
Build finished
16 minutes ago
```

Build in 'master' (a2517024)

CLIENTE DOCKER CLOUD

Docker Cloud nos proporciona una utilidad para interaccionar con los servicios desde la línea de comandos, lo que nos permitirá realizar acciones sin tener que acceder al portal web.

INSTALACIÓN

Hay dos maneras de instalar este cliente:

- Desplegando un contenedor con la utilidad instalada:

```
# docker run dockercloud/cli -h
Unable to find image 'dockercloud/cli:latest' locally
latest: Pulling from dockercloud/cli

e110a4a17941: Already exists
ff6695dfa07: Pull complete
3e989df1a785: Pull complete
f33c9c7f5598: Pull complete
Digest:
sha256:2eb625805d028d635f5038b30595b13dd62fad72fddoac37409b38cba3514339
Status: Downloaded newer image for dockercloud/cli:latest
usage: docker-cloud [-h] [-v]
[-]
```

- Instalación manual utilizando *pip* o *brew*.

- Para sistemas *Linux* o *Windows* ejecutaremos el comando:

```
# pip install docker-cloud
Collecting docker-cloud
  Downloading docker-cloud-1.0.7.tar.gz
    Collecting future<1,>=0.15.0 (from docker-cloud)
      Downloading future-0.16.0.tar.gz (824kB)
      ...
Successfully built docker-cloud future ago python-dockercloud
Installing collected packages: future, ago, python-dateutil, python-dockercloud, tabulate, docker-cloud
Successfully installed ago-0.0.9 docker-cloud-1.0.7 future-0.16.0 python-dateutil-2.6.0
python-dockercloud-1.0.9 tabulate-0.7.7
```

- Para sistemas *macOS* ejecutaremos:

```
$ brew install docker-cloud
==> Installing dependencies for docker-cloud: libyaml
==> Installing docker-cloud dependency: libyaml
==> Downloading https://homebrew.bintray.com/bottles/libyaml-0.2.7.sierra.bottle.tar.gz
######################################################################## 100.0%
==> Pouring libyaml-0.2.7.sierra.bottle.tar.gz
/usr/local/Cellar/libyaml/0.2.7: 8 files, 312.6K
==> Installing docker-cloud
==> Downloading https://homebrew.bintray.com/bottles/docker-cloud-1.0.7.sierra.bottle.tar.gz
######################################################################## 100.0%
==> Pouring docker-cloud-1.0.7.sierra.bottle.tar.gz
/usr/local/Cellar/docker-cloud/1.0.7: 402 files, 4.2M
```

- Para validar dicha instalación, utilizaremos la opción `-v`.

```
# docker run dockercloud/cli -v
docker-cloud 1.0.7
# docker-cloud -v
docker-cloud 1.0.7
```

Una vez instalado el cliente, debemos hacer `login` a través del cliente de Docker Engine a través de `docker login`.

```
# 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 (aforte): aforte
Password:
Login Succeeded
```

ACCIONES

Las acciones que podemos utilizar con docker-cloud y sus opciones se detallan a continuación.

action [subaccion]

Las subacciones son las siguientes:

- ***ls***: lista las acciones. Opciones:
 - ***-l lineas***: muestra un número determinado de lineas.
 - ***-q***: muestra solo el identificador.
- ***cancel***: cancela una acción en estado pendiente o en progreso.
- ***inspect uuid***: muestra todos los detalles de una acción.
- ***logs uuid***: muestra los registros de una acción. Opciones:
 - ***-f***: espera a por nuevos registros.
 - ***-t lineas***: muestra el número de lineas especificado, por defecto 300.
- ***retry uuid***: reintenta una tarea con estado completado, fallado o cancelado.

Ejemplo:

```
# docker-cloud action ls -l 1
UUID ACTION START END TARGET IP LOCATION
83c1333e Repository Push 2 hours ago 2 hours ago repository/aforte/delbianapache2/tag/latest 52.205.214.174 Ashburn
# docker-cloud action inspect 83c1333e
{
  "user": "aforte",
  "can_be_canceled": false,
  "uuid": "83c1333e-d38f-4ba8-a8a5-6e2eff573a26",
  "end_date": "Sat, 28 Jan 2017 11:16:33 +0000",
```

```

"created": "Sat, 28 Jan 2017 11:16:33 +0000",
"ip": "52.206.214.174",
"start_date": "Sat, 28 Jan 2017 11:16:33 +0000",
"object": "/api/repo/v1/repository/oforte/debianapache2/tag/latest/",
"build_code": null,
"method": "POST",
"can_be_retried": false,
"is_user_action": true,
"state": "Success",
[...]
}
# docker-cloud action logs 83c1333e
Push to oforte/debianapache2:latest

```

container [subacción]

Las *subacciones* son las siguientes:

- **ps:** lista los contenedores. Las opciones son las siguientes:
 - **-q:** solo muestra los identificadores de los contenedores.
 - **-s [Init|Stopped|Starting|Running|Stopping|Terminating|Terminated]:** filtra los contenedores por un estado.
 - **--service servicio:** filtra por un servicio.
 - **--no-trunc:** no limita por tamaño los campos (muestra toda la información).
- **exec contenedor comando:** ejecuta un comando en el contenedor especificado.
- **inspect contenedor:** muestra información detallada de un contenedor.
- **logs contenedor:** muestra los registros de un contenedor. Opciones:
 - **-f:** muestra los registros y espera por nuevos registros.
 - **-t líneas:** muestra el número de líneas especificadas, por defecto 300.
 - **redeploy contenedor:** reconstruye el contenedor.

- **start contenedor:** inicia el contenedor especificado.
- **stop contenedor:** detiene el contenedor especificado.
- **terminate contenedor:** elimina el contenedor especificado.

Ejemplos:

```
# docker-cloud container ps
NAME          UUID        STATUS      IMAGE
hel[...].d3af21d0  Running  dockercloud/hello-world:latest /bin/sh -c 'php -...
hel[...].cb1fca1  Running  dockercloud/hello-world:latest /bin/sh -c 'php -...
hel[...].20f25a0a  Running  dockercloud/hello-world:latest /bin/sh -c 'php -...
# docker-cloud container ps -q
d3af21d0-4519d-468a-8c6e-3d69976db0cb
cb14bca1-d18a-4738-9515-e988397aa59
20f25a0a-06e8-4012-b8fb-0aa7d645d789
# docker-cloud container exec d3af21d0 hostname
hello-world-d9792f6a-1
# docker-cloud container logs d3af21d0
2017-01-28T14:34:26.589854416Z 172.17.0.1 - [28/jan/2017:14:34:25 +0000] "GET /HTTP/1.1"
200 495 "-" "curl/7.38.0"
```

event

Muestra en tiempo real los eventos desde Docker Cloud. Ejemplo:

```
# docker-cloud event
{"type": "action", "action": "create", "parents": ["/api/app/v1/oforte/service/5ee8c545-eeb2-40e4-b41c-07e09e140410"], "/api/app/v1/oforte/stack/e41fb824-5b32-4d73-ac70-a16a2f86b355"], "resource_uri": "/api/audit/v1/action/d1f771ce-196e-4f1f-b724-ee372b6ed9c4", "state": "In progress", "namespace": "oforte", "datetime": "2017-01-28T14:41:26Z", "uuid": "793a7d19-5d72-454b-a02e-9cc3ccce16a0"}
{"type": "service", "action": "update", "parents": ["/api/app/v1/oforte/stack/e41fb824-5b32-4d73-ac70-a16a2f86b355"], "resource_uri": "/api/app/v1/oforte/service/5ee8c545-eeb2-40e4-b41c-07e09e140410", "state": "Starting", "namespace": "oforte", "datetime": "2017-01-28T14:41:26Z", "uuid": "2f04f095-e393-4b1e-a0d1-2f0d0085cd54"}
```

node [subacción]

Las subacciones son las siguientes:

- **ls:** lista los *nodos* disponibles. Con la opción *-q* mostrará solamente los identificadores.
- **inspect nodo:** muestra información detallada de un *nodo*.
- **rm nodo:** elimina un *nodo*.
- **upgrade nodo:** actualiza una nueva versión del servicio de Docker.
- **byo:** muestra las instrucciones para crear un *nodo* propio en un servidor.

Ejemplos:

```
# docker-cloud node ls
UUID          FQDN           LASTSEEN      STATUS    CLUSTER   DOCKER_VER
5d2bd828-f566-4d44-8b7a-7669c0c04375.node.dockerapp.io 6 minutes ago Deployed  1.11.2-cs5
# docker-cloud node inspect 5d2bd828
{
  "availability_zone": null,
  "public_ip": "91.121.79.141",
  "node_type": null,
  "docker_version": "1.11.2-cs5",
  "disk": 60,
  "deployed_datetime": "Sat, 28 Jan 2017 14:17:59 +0000",
  "uuid": "5d2bd828-f566-4d44-8b7a-7669c0c04375",
  "destroyed_datetime": null,
  "external_fqdn": "5d2bd828-f566-4d44-8b7a-7669c0c04375.node.dockerapp.io",
  "node_cluster": null,
  "state": "Deployed",
  "memory": 1024,
  "current_num_containers": 4,
  "tags": [],
  "docker_execdriver": "",
  "docker_graphdriver": "aufs",
  "nickname": "5d2bd828-f566-4d44-8b7a-7669c0c04375.node.dockerapp.io",
  "dockercloud_action_uri": "",
  "private_ips": [
    {}
  ],
  "tunnel": null,
  "region": null,
  "last_seen": "Sat, 28 Jan 2017 14:41:44 +0000",
```

```

    "cpu": 1,
    "resource_uri": "/api/infra/v1/oforte/node/5d2bd828-f566-4d44-8b7a-7669c0c04375/"
}

```

nodecluster [subacción]

Las subacciones son las siguientes:

- **az**: muestra las zonas disponibles.
- **create**: crea un *cluster de nodos*.
- **inspect**: muestra información de un *cluster de nodos*.
- **ls**: lista *cluster de nodos*.
- **nodetype**: muestra todos los tipos disponibles.
- **provider**: muestra todos los proveedores disponibles.
- **region**: muestra todas las regiones disponibles.
- **rm**: elimina un *cluster de nodos*.
- **scale**: escala un *cluster de nodos*.

Ejemplos:

```

# docker-cloud nodecluster provider
NAME      LABEL
aws       Amazon Web Services
digitalocean Digital Ocean
azure     Microsoft Azure
softlayer SoftLayer
packet   Packet
# docker-cloud nodecluster region -p aws
NAME      LABEL      PROVIDER
ap-northeast-1 ap-northeast-1  aws
ap-northeast-2 ap-northeast-2  aws
ap-south-1    ap-south-1    aws
ap-southeast-1 ap-southeast-1 aws
ap-southeast-2 ap-southeast-2 aws

```

eu-central-1	eu-central-1	aws
eu-west-1	eu-west-1	aws
sa-east-1	sa-east-1	aws
us-east-1	us-east-1	aws
us-west-1	us-west-1	aws

repository [subacción]

Las *subacciones* son las siguientes:

- **ls:** lista solo los repositorios externos configurados.
- **inspect:** inspecciona un repositorio externo.
- **register:** registra un repositorio externo en Docker Cloud.
- **rm:** elimina un repositorio externo en Docker Cloud.
- **update:** actualiza un repositorio externo en Docker Cloud.

Ejemplo:

```
# docker-cloud repository register quay.io/aforte/apache2
Please input username and password of the registry:
Username: aforte
Password:
quay.io/aforte/apache2
# docker-cloud repository ls
NAME          IN_USE
quay.io/aforte/apache2 no
```

run [opciones]

Crea y ejecuta un nuevo servicio. Las opciones que se pueden especificar se detallan en la siguiente tabla:

Opción	Descripción	Ejemplo
-n nombre	Nombre para el servicio.	-n web01
--cpushares	La prioridad para el CPU.	--cpushares 600
--memory	La memoria restringida.	--memory 128M

Opción	Descripción	Ejemplo
-t cantidad	El número de contenedores a usar. Defecto 1.	-t 3
-r comando	El comando a ejecutar al iniciar el contenedor. Por defecto el definido en la imagen.	-r /bin/aplicacion
--entrypoint cmd	El comando de punto de entrada.	--entrypoint zsh
-p puerto	El puerto a publicar para ser accesible.	-p 8888:80
--expose PUERTO	Expone sin publicar un puerto.	--expose 80
-e variable	Especifica variables de entorno.	-e USER=root
--env-file fichero	Especifica un fichero con las variables de entorno.	--env-file dev.env
--autodestroy valor	Indica si el contenedor debe ser destruido al finalizarse. Los valores posibles son:	--autodestroy ON_FAILURE
	<ul style="list-style-type: none"> • OFF • ON_FAILURE • ALWAYS 	
--autoredeploy	Indica si debe el contenedor debe ser recreado si la <i>imagen</i> es más actual.	
--autorestart valor	Indica si el contenedor debe ser reiniciado o iniciado en caso de ser detenido. Valores:	--autorestart ALWAYS
	<ul style="list-style-type: none"> • OFF • ON_FAILURE • ALWAYS 	
-v volumen	Indica el volumen a utilizar, con el formato: • /directorio_contenedor • /dir_servidor:/dir_contenedor	-v /var/www/
--deployment-strategy	La estrategia para distribuir los contenedores, los valores posibles son:	--deployment-strategy EVERY_NODE
	<ul style="list-style-type: none"> • EMPIEST_NODE, • HIGH_AVAILABILITY • EVERY_NODE 	
--sync	Espera a que la tarea finalice. (No asíncrono)	

Ejemplo:

```
# docker-cloud run dockercloud/hello-world --memory 128 -p 80 --sync
In progress.
```

```
Success
79a810ed-8810-4115-ad5b-926516211daf
```

service [subacción]

Las *subacciones* son las siguientes:

- **ps**: lista todos los *servicios*. Las opciones que podemos especificar:
 - **-q**: solo muestra los identificadores de los *contenedores*.
 - **-s [Init|Stopped|Starting|Running|Stopping|Terminating|Terminated]**: filtra los *contenedores* por un estado.
- **inspect servicio**: muestra información detallada de un servicio.
- **create [opciones] image**: mismas opciones que la acción run, crea el servicio pero no lo ejecuta.
- **logs image**: muestra los registros de un servicio.
- **run**: igual que **docker-cloud run**.
- **redeploy servicio**: vuelve a crear un *servicio*.
- **scale num**: escala un *servicio* al número especificado de *contenedores*.
- **set [opciones] servicio**: establece opciones (Ver opciones de run).
- **start servicio**: inicia un *servicio* que está detenido.
- **stop servicio**: detiene un *servicio* que está detenido.
- **terminate servicio**: elimina un *servicio* que está detenido.
- **env subacción contenedor**: opera con las variables de entorno, las subacciones son:
 - **ls**: lista las variables de entorno.
 - **add**: añade una variable de entorno a un servicio.
 - **set**: reemplaza variables de entorno.

- **update:** actualiza el valor de una variable de entorno existente.
- **rm:** elimina una variable de entorno.

Ejemplos:

```
# docker-cloud service ps -s Running
NAME      UUID      STATUS      #CONTAINERS IMAGE          DEPLOYED      PUBLIC DNS      STACK
hello [...] 79a810ed      Running      1 dockercloud/hello-world:latest 13 minutes ago  hello-world-887e90d1.79a810ed[...]
# docker-cloud service inspect hello-world-887e90d1
{
  "tty": false,
  "synchronized": true,
  "domainname": null,
  "labels": {},
  "pid": "none",
  "linked_to_external_service": [],
  "sequential_deployment": false,
  "public_dns": "hello-world-887e90d1.79a810ed.svc.dockerapp.io",
  "bindings": [],
  "target_num_containers": 1,
  "deployed_datetime": "Sat, 28 Jan 2017 15:58:22 +0000",
  [...]
}
# docker-cloud service logs hello-world-887e90d1
hello-world-887e90d1-1 | 2017-01-28T15:59:40.728813765Z 95.90.235.182 - -
[28/jan/2017:15:59:39 +0000] "GET /HTTP/1.1" 200 495 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36"
hello-world-887e90d1-1 | 2017-01-28T15:59:40.728862590Z 95.90.235.182 - -
[28/jan/2017:15:59:39 +0000] "GET /logo.png HTTP/1.1" 200 13133 "http://hello-world-887e90d1-1.6c9d5578.cont.dockerapp.io:32772/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36"
hello-world-887e90d1-1 | 2017-01-28T15:59:40.728874667Z 95.90.235.182 - -
[28/jan/2017:15:59:40 +0000] "GET /favicon.ico HTTP/1.1" 200 495 "http://hello-world-887e90d1-1.6c9d5578.cont.dockerapp.io:32772/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36"
# docker-cloud service scale hello-world-887e90d1 2
79a810ed-8810-4115-ad5b-926516211d0f
# docker-cloud container ps --service hello-world-887e90d1 -q
6c9d5578-107d-4226-a800-a126cd0a6ea
605bbe23-5151-49bd-b559-82351e79bc48
```

stack [subacción]

Las subacciones son las siguientes:

- **ls**: lista los *stack* y su estado. Con la opción *-q* muestra solamente los identificadores.
- **inspect stack**: muestra información detallada de un *stack*.
- **create -n name -f fichero [--sync]**: crea un *stack* a partir de un fichero.
- **export [-f fichero] stack**: exporta un *stack* a la pantalla o a un fichero.
- **redeploy stack**: vuelve a desplegar los *contenedores* de un *stack*.
- **start stack**: inicia los *contenedores* definidos.
- **stop stack**: detiene los *contenedores* definidos.
- **terminate stack**: destruye los *contenedores* definidos.
- **up stack -f fichero -n name**: crea y despliega los *contenedores* definidos en un fichero.
- **update -f fichero stack**: actualiza la definición de un *stack* con el contenido de un fichero especificado.

Ejemplos:

```
# docker-cloud stack ls
NAME      UUID      STATUS      DEPLOYED      DESTROYED
wordpress-mysql e41fb824  Running  18 hours ago
# docker-cloud stack inspect wordpress-mysql
{
  "current_num_containers": 2,
  "name": "wordpress-mysql",
  "synchronized": true,
  "destroyed_datetime": null,
  "uuid": "e41fb824-5b32-4d73-ac70-a16a2f86b355",
  "state": "Running",
  "services": [
    "/api/app/v1/oforte/service/5ee8c545-eeb2-40e4-b41c-07e09e140410/",
    "/api/app/v1/oforte/service/c9b4776f-20a8-4da6-b7d6-af9408a073ed/"
  ]
}
```

```

"deployed_datetime": "Fri, 27 Jan 2017 21:50:03 +0000",
"nickname": "wordpress-mysql",
"dockercloud_action_url": "",
"resource_uri": "/api/app/v1/oforte/stack/e41fb824-5b32-4d73-ac70-a16a2f86b355/"
}
# docker-cloud stack export wordpress-mysql
db:
environment:
- MYSQL_DATABASE=wordpress
- MYSQL_PASSWORD=wordpress
- MYSQL_ROOT_PASSWORD=wordpress
- MYSQL_USER=wordpress
image: mysql:5.7
wordpress:
environment:
- WORDPRESS_DB_HOST=db:3306
- WORDPRESS_DB_PASSWORD=wordpress
image: wordpress:latest
links:
- db
ports:
- 8000:80

```

trigger [subacción]

Las subacciones son las siguientes:

- ***ls servicio***: lista los triggers con información de un servicio.
- ***create [-n name] -u operación servicio***: crea un nuevo trigger para un servicio.

La operación puede ser *redeploy* o *scaleup*.

- ***rm servicio trigger***: elimina un trigger de un servicio.

Ejemplos:

```

# docker-cloud trigger ls hello-world-887e90d1
UUID      NAME      OPERATION      URL
6ea5d7c1  new-webhook  SCALEUP  https://cloud.docker.com/api/[...]
# docker-cloud trigger create -o redeploy hello-world-887e90d1
79a810ed-8810-4115-ad5b-926516211ddf

```

up [opciones]

Igual que ***docker-cloud stack up***.

14

DOCKER HUB

Introducción

Docker Hub es un portal que permite realizar más acciones que almacenar imágenes, también nos permite:

- Automáticamente crear *ímágenes* a partir de un repositorio de control de versiones.
- Permite enviar una notificación a una página web cuando la creación de la imagen ha finalizado (*webhook*).
- Crear organizaciones (grupos de trabajo) para organizar el acceso a las imágenes.

ORGANIZACIONES

Una organización dentro de **Docker Hub** permite crear equipos de trabajo para dar permisos a usuarios a repositorios de imágenes compatibles. Una organización puede contener repositorios privados o públicos igual que con una cuenta personal. En el menú superior pulsaremos sobre **Organizations** y posteriormente sobre **Create Organization** y aparecerá el siguiente formulario:

Create Organization

Organizations can have multiple Teams. Team can have different permissions. NameSpace is unique and this is where repositories for this organization will be created.

NameSpace:

Organization Full Name:

Country:

Location:

Greater Area:

Website URL:

Create

rcilibros's teams

Choose Team:

mario	X
dani	X

Edit team

Team Name:

Description:

Save

En la siguiente pantalla elegiremos los usuarios (con cuenta en Docker ID) para el primer grupo, que especifica a los propietarios: *owners*. Los usuarios añadidos recibirán un correo al realizar tarea.

Es posible crear nuevos *Teams* (*equipos*) utilizando el botón **Create Team**, especificando el nombre en el formulario de la derecha (*Team Name*) y pulsando **Save**.

REPOSITORIOS

En el caso de tener creada una organización podemos especificar en el desplegable en la parte superior izquierda si queremos trabajar con nuestra cuenta personal o con la organización.



Al crear un repositorio bajo una *organización*, podemos dar permisos individuales a los equipos que hayamos creado. Seleccionando el repositorio y pulsando sobre *Collaborators* nos permitirá elegir el equipo y el permiso: *Admin*, *Write* (Escritura) o *Read* (Lectura).

Team	Access	Action
programacion	Read	<button>Remove</button>
rcbook	Admin	<button>Remove</button>

AUTOMATIZAR CREACIÓN

Con **Docker Hub** es posible automatizar la creación de imágenes a través de un repositorio en un control de versiones alojado en *Github* o *Bitbucket*. Primero debemos enlazar nuestra cuenta con uno de ellos, para ello iremos a la configuración de nuestra cuenta y pulsamos sobre *Linked Accounts & Services*.

The screenshot shows the 'Linked Accounts & Services' section of the Docker Hub settings. It displays two cards: one for 'Link GitHub' with the GitHub logo and another for 'Link Bitbucket' with the Bitbucket logo.

Pulsando sobre la plataforma a la que queremos conectarnos (solo es posible utilizar una), nos pedirá aceptar la autorización para enlazar nuestras cuentas. Elegiremos a continuación la opción recomendada: *Public and Private (Recommended)* y seguiremos las instrucciones indicadas. A continuación se actualizará la información:



Después de enlazar la cuenta, todo está listo para crear nuestra automatización. Para ello en el menú superior, pulsamos sobre **Create** y posteriormente **Create Automated Build**. Elegimos el proveedor que configuramos anteriormente, *Github* o *Bitbucket*, y posteriormente el *repositorio* que contiene el fichero *Dockerfile*.

The configuration page includes fields for Repository Namespace & Name (set to 'rcibros' and 'Hello-docker'), Visibility ('public'), Short Description ('Ejemplo de automatización.'), and a note about branch matching. A 'Create' button is at the bottom.

Después de pulsar sobre **Create**, podemos acceder a lista de nuestros repositorios donde se especificará qué repositorios tienen activada la automatización.



Accediendo a dicho repositorio, aparecen dos opciones en el menú superior:

- *Build Details*: muestra el historial del proceso de creación de la imagen.
- *Build Settings*: la configuración para la creación y la posibilidad de realizar la creación manualmente.

En esta última opción, podemos ver que por defecto se generará la imagen a cada publicación (*push*) al repositorio siempre que sea en la rama (*branch*) llamada *master* y la versión de la imagen generada será *latest*. Es posible utilizar distintas ramas para generar distintas versiones de las imágenes.

Pulsando sobre el botón *Trigger*, se iniciará la creación de la imagen a partir del contenido del repositorio. Podemos ver el estado a través del menú *Build Details*.

Status	Actions	Tag	Created	Last Updated
Building	Cancel	latest	a minute ago	a minute ago

Es posible hacer clic sobre ese registro para ver el contenido del fichero *Dockerfile* y ver el progreso de la creación.

Dockerfile

```
FROM debian:latest
MAINTAINER Alberto Gonzalez <alberto@oforte.net>
# Instalar apache2 y configurar locales
RUN apt-get update && apt-get install -y locales locales-all apache2 curl
RUN locale-gen es_ES.UTF-8
EXPOSE 80
VOLUME /var/www
# Variables para Apache 2
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_LOG_DIR /var/log/apache2
RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR
# Contenido web
VOLUME /var/www/html/
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Logs

```
Building in Docker Cloud's infrastructure...
Cloning into '.'...
KernelVersion: 4.4.0-59-generic
Os: Linux
BuildTime: 2017-01-10T21:09:49.600531904+00:00
AptVersion: 1.2.6-0.1
Version: 1.12.6-0.1
GlibcVersion: 2.24
Arch: amd64
Generator: go1.6.4
Starting build of index.docker.io/rclibres/libro-docker:latest...
4: 5: 6: 7: 8: 9:
```

WEBHOOKS

Es posible, después de cada creación de una *imagen* (a través de `docker push` o con automatización) hacer una llamada a una página web: ya sea para realizar una notificación, ejecutar una aplicación o hacer diferentes pruebas con la nueva *imagen*.

Para ello dentro del *repositorio*, en el menú superior accedemos a *Webhooks* y añadimos a la página web a la que queremos hacer la llamada. En este ejemplo hemos utilizado para ver la solicitud la siguiente página: <http://requestb.in>



Después de la finalización de creación de una imagen o de su publicación, una solicitud será realizada.

15

SWARM

Introducción

Una de las limitaciones que se describía en la introducción era la falta de desplegar contenedores en alta disponibilidad de una forma sencilla y nativa por parte de **Docker Engine**. Anteriormente a la versión 1.12 de Docker, el uso de **Docker Swarm** era un componente separado, pero a partir de dicha versión forma parte de **Docker Engine** con el nombre de **Swarm mode**. En este libro solo detallaremos este último.

Las características de **Swarm mode** son las siguientes:

- Administración de *Cluster* utilizando **Docker Engine**.
- Diseño descentralizado, es posible crear un *cluster* utilizando solo una imagen.
- Facilidad de *escalar* servicios para desplegar contenedores para satisfacer necesidades.
- Facilidad de definir servidores *frontend* y *backend*.
- Monitorización del estado del *cluster* y posibilidad de solucionar problemas automáticamente.
- Descubrimiento de servicio a través de *DNS* integrado en **Swarm**.
- Balance de carga (*Load Balancing*) a distribuir entre los contenedores.
- Seguridad en comunicaciones entre los *nodos Swarm*.

- Facilidad de actualizar los sistemas sin necesidad de paradas.

TERMINOLOGÍA

Cuando trabajamos con **Swarm** es importante tener claro los siguientes términos:

- **Orquestación:** describe el flujo para desplegar aplicaciones relacionadas entre sí.
- **Swarm:** es un *cluster* de Docker engines, también llamados *nodos*, donde se desplegarán los *servicios*. El cliente de Docker incluye comandos para administrar *nodos* (añadir o eliminar), y desplegar y orquestar *servicios* a través de dichos *nodos*.
- **Nodo:** un *nodo* es una instancia de Docker Engine que participará en Swarm. En entornos de producción habrá diferentes *nodos* separados en distintos servidores y diferentes localizaciones.
- **Nodo administrador (Manager node):** es el encargado de repartir las tareas a los *nodos de trabajo*.
- **Nodo de trabajo (Worker node):** recibe y ejecuta tareas repartidas por el *nodo administrador*.
- **Servicios:** un *servicio* es la definición de tareas a ejecutar en un *nodo de trabajo*. Un servicio contiene la *Imagen* a utilizar y los comandos a ejecutar dentro de los *contenedores*.
- **Tareas:** una *tarea* consiste en el comando a ejecutar dentro de un *contenedor* específico.
- **Balance de carga (Load Balancing):** el administrador de *swarm* utiliza un balance de carga de conexiones entrantes para exponer los servicios deseados de forma pública. A través de un *DNS* interno automáticamente balanceará la carga a los *contenedores* de un *servicio*.

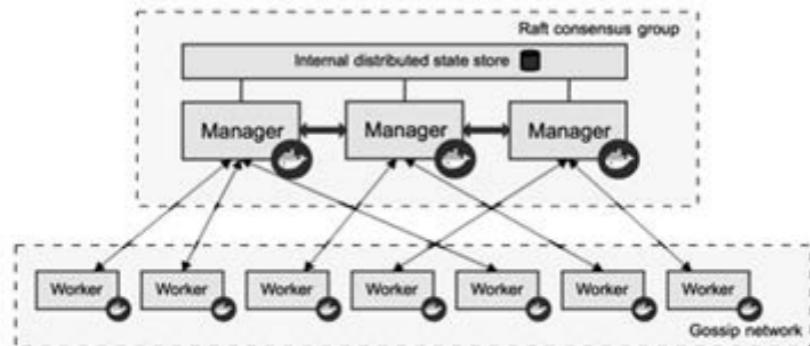
ARQUITECTURA

Los requisitos para desplegar nodos **Swarm** son los siguientes:

- Al menos una red para distribuir las comunicaciones, pero se recomienda tener 3 servidores: un *nodo administrador* y dos *nodos de trabajo*.

- Docker Engine 1.12 o posterior.
- Los siguientes puertos abiertos entre los servidores:
 - Puerto 2377/tcp para comunicaciones administrativas del cluster.
 - Puerto 7946/tcp y udp, para la comunicación entre los *nodos*.
 - Puerto 4789/tcp y udp, para tráfico de red.

En la siguiente imagen se ve la arquitectura en un entorno de producción con 10 servidores, distribuidos como 3 *nodos administradores* y 7 *nodos de trabajo*.



En nuestra instalación utilizaremos los siguientes servidores:

Nombre	Descripción	Dirección IP
manager1	Nodo administrador	192.168.56.105
worker1	Nodo de trabajo	192.168.56.106
worker2	Nodo de trabajo	192.168.56.107

CREAR UN SWARM

El cliente de Docker Engine nos ofrece la acción swarm para crear y manipular los componentes de Swarm. Para inicializar la creación utilizaremos el siguiente comando:

- docker `swarm init` --advertise-addr <IP-Nodo-administrador>

Este comando lo podemos ejecutar desde nuestro *nodo administrador (manager1)* y producirá la siguiente salida:

```
manager1:~# docker swarm init --advertise-addr 192.168.56.105
Swarm initialized: current node (kf6hvvedlrs018emhx3thm6rx7) is now a manager.
```

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

```
docker swarm join \
--token SWMTKN-1-684aa2y74d1xg87rg3odukp260ypov7a0hdynecd4o2mheb3bi-8z6mi6ajqmvzlwp9ajycpvoj1
192.168.56.105:2377
```

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

Utilizando la acción `info` podemos obtener información del estado actual de Swarm:

```
manager1:~# docker info
[...]
Swarm: active
  NodeID: kf6hvvedlrs018emhx3thm6rx7
  Is Manager: true
  ClusterID: pom2c9cqck25se1z5z4w5u1ng
  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: 192.168.56.105
  Manager Addresses:
```

```
192.168.56.105:2377
```

```
[...]
```

Podemos listar los nodos de Swarm utilizando `docker node ls`.

```
manager1:~# docker node ls
ID          HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
kf6h6vdris018ermhx3thm6rx7 *  manager1  Ready  Active    Leader
```

AÑADIR LOS NODOS DE TRABAJO

En la salida de la acción `init` nos mostraba el comando que debemos ejecutar en los *nodos de trabajo* para añadirlos a Swarm. En el caso de no tener la salida disponible, es posible volver a obtenerlo con el siguiente comando desde el *nodo administrador*:

```
manager1:~# docker swarm join-token worker
To add a worker to this swarm, run the following command:
```

```
docker swarm join \
--token SWMTKN-1-684aa2y74d1xg87rg3odukp260ypov7a0hdynecd4o2mheb3bi-
8z6mi6ojqxvzlwp9ajycpvojl \
192.168.56.105:2377
```

Debemos ejecutar dicho comando en todos los *nodos de trabajo* que queremos añadir.

```
worker1:~# docker swarm join \
> --token SWMTKN-1-684aa2y74d1xg87rg3odukp260ypov7a0hdynecd4o2mheb3bi-
\
```

```
> 192.168.56.105:2377
```

This node joined a swarm as a worker.

```
worker2:~# docker swarm join \

```

```
> --token SWMTKN-1-684aa2y74d1xg87rg3odukp260ypov7a0hdynecd4o2mheb3bi-
\
```

```
> 192.168.56.105:2377
```

This node joined a swarm as a worker.

Una vez configurados los *nodos de trabajo*, podemos listar su estado desde el *nodo administrador* con el comando anteriormente indicado.

```
manager1:~# docker node ls
ID          HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
by64v13d13ddqo4vb5xsrmgal  worker1  Ready  Active
kf6hvedlrs018ermhx3thm6rx7 * manager1  Ready  Active      Leader
pjrx4t4qv3ufdfiodzdqwfkh1  worker2  Ready  Active
```

En la columna *MANAGER STATUS* observamos qué nodos son *administrador*, podemos tener varios y uno de ellos será el líder (*Leader*).

DESPLEGAR UN SERVICIO

Una vez configurados nuestro *nodo administrador* y nuestros *nodos de trabajo*, todo está listo para desplegar servicios. Para ello utilizaremos la acción `service create` que tiene la siguiente sintaxis:

- `docker service create [opciones] imagen comando`

Donde las opciones básicas son las siguientes:

- `--name nombre`: el nombre para el servicio.
- `--replicas número`: el número de nodos a desplegar en el servicio.
- `-p/-publish puerto`: el puerto a publicar para ser accesible.

Ejemplo:

```
manager1:~# docker service create --name servicio1 --replicas 1 httpd
todog68qixhn49n9kjqqpeavj
```

Para listar los *servicios* y el estado de ellos, ejecutamos el siguiente comando.

```
manager1:~# docker service ls
ID      NAME      MODE      REPLICAS  IMAGE
todog68qixhn  servicio1  replicated  1/1    httpd:latest
```

Para obtener información detallada de un *servicio*, al igual que con otros objetos de Docker, utilizaremos la subacción `inspect` (la opción `--pretty` muestra la salida sin ser en formato JSON).

```
manager1:~# docker service inspect --pretty servicio1
ID:todog68qixhn49n9kjqqpeavj
Name:servicio1
Service Mode:Replicated
Replicas:2
Placement:
UpdateConfig:
Parallelism:1
On failure:pause
Max failure ratio: 0
ContainerSpec:
  Image: httpd:latest@sha256:1407eb0941b010035f86dfe8b7339c4dd4153e2f7b855ccc67dc0494e9f2756c
Resources:
  Endpoint Mode:vip
```

Al ejecutar este servicio, le especificamos que se ejecutará solo en un nodo (`--replicas 1`); para ver en cuál se está ejecutando actualmente utilizamos la subacción `ps`.

```
manager1:~# docker service ps servicio1
ID          NAME      IMAGE      NODE      DESIRED STATE CURRENT STATE      ERROR PORTS
o27ah3ludj7g  servicio1.1  httpd:latest  worker1  Running   Running 10 seconds ago
```

Si accedemos al *nodo de trabajo* donde está ejecutando dicho servicio, en este caso `worker1`, podemos listar los contenedores ejecutándose con el comando conocido: `docker ps`.

```
worker1:~# docker ps
CONTAINER ID        IMAGE           COMMAND       CREATED          STATUS          PORTS     NAMES
9ec582cc964b        httpd:latest    "About a minute ago"   Up About a minute   80/tcp    servicio1.1.serv1_1
```

ESCALAR UN SERVICIO

En la creación del *servicio* anterior hemos especificado que solo se utilice una *replica* de la aplicación a desplegar a partir de una *imagen*. Es posible después de la creación, *escalar* para que se desplieguen más contenedores distribuidos a través de Swarm. Para ello utilizaremos el siguiente comando:

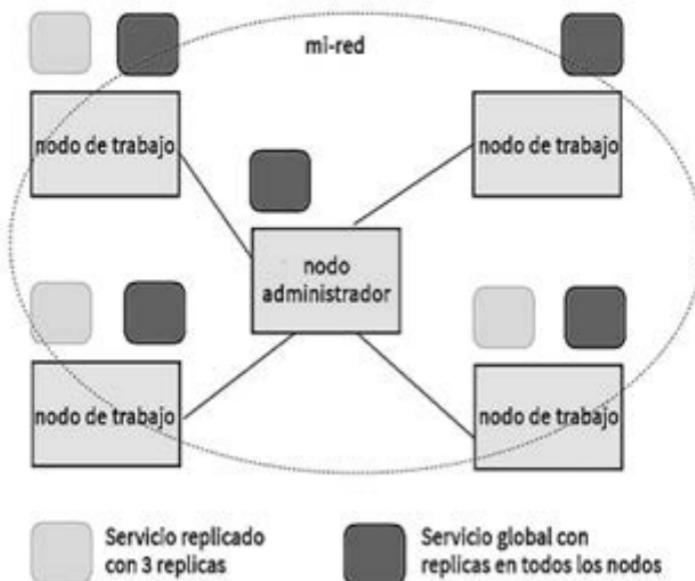
- `docker service scale servicio=replicas`

En el siguiente ejemplo desplegamos 4 *replicas* al *servicio* creado previamente.

```
manager1:~# docker service scale servicio1=4
servicio1 scaled to 4
```

```
manager1:~# docker service ps servicio1
ID          NAME      IMAGE      NODE      DESIRED STATE CURRENT STATE      ERROR      PORTS
o27ah3hdj7g  servicio1.1  httpd:latest  worker1  Running   Running  14 minutes ago
i3o4glvty0na  servicio1.2  httpd:latest  worker2  Running   Running  1 second ago
zsbs97x5zjk  servicio1.3  httpd:latest  manager1  Running   Running  17 seconds ago
kmmpj8leg1a9i  servicio1.4  httpd:latest  worker2  Running   Running  1 second ago
```

Observamos que los contenedores se han desplegado a través de todos los *nodos* que forman el *cluster Swarm*. La arquitectura al utilizar más de 1 *replica* se refleja en la siguiente imagen:



ELIMINAR UN SERVICIO

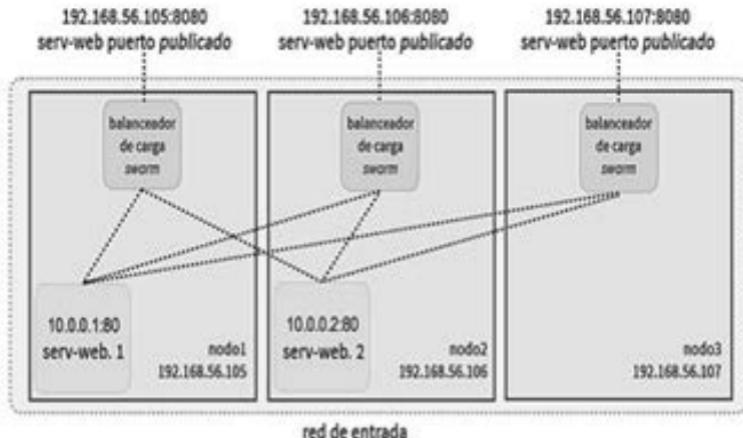
Una vez que un servicio ya no sea necesario, podemos eliminarlo como se muestra a continuación:

```
# docker service rm servicio1
servicio1
```

Pasados unos minutos después de la eliminación del servicio, los *contenedores* que se ejecutaban en los *nodos* serán eliminados.

PUBLICAR PUERTOS

Al crear un servicio podemos especificar el puerto que queremos publicar y será accesible desde todos los nodos. En la siguiente imagen se describe la arquitectura al ejecutar un servidor web con 2 *replicas* y publicando el puerto 80 a través del puerto 8080 de los *nodos*.



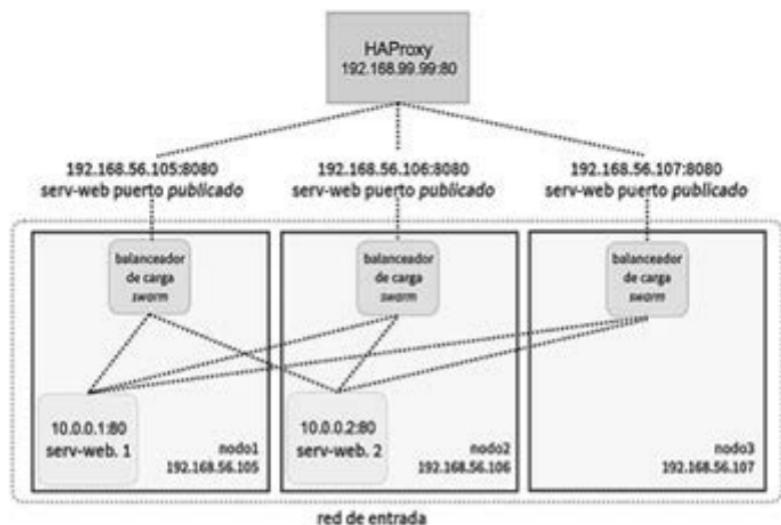
Podemos observar en el siguiente ejemplo que el puerto 8080 es accesible en todos los *nodos* y ejercerá como *balanceador de carga*.

```
manager1:~# docker service create --name servicio1 --replicas 3 --publish 8080:80 dockercloud/hello-world
n791zqvnchyh6vf9wtxvc6qu4y
manager1:~# curl -s 192.168.56.105:8080|grep hostname
<h3>My hostname is 60678b862771</h3></body>
manager1:~# curl -s 192.168.56.106:8080|grep hostname
<h3>My hostname is 1c2a5e88e264</h3></body>
manager1:~# curl -s 192.168.56.107:8080|grep hostname
<h3>My hostname is 60678b862771</h3></body>
manager1:~# curl -s 192.168.56.107:8080|grep hostname
<h3>My hostname is 9147e5bf8bcd</h3></body>
```

Es posible también publicar un puerto a un servicio previamente creado, utilizando el siguiente comando:

- **docker service update --publish-add puertosrv:puertocon servicio**

Teniendo los contenedores distribuidos en un nuevo *cluster* y publicando un puerto, podemos utilizar un *balanceador de carga* externo que apunte a los *nodos*. Uno de los más conocidos es *HAProxy*. En la siguiente imagen se muestra su arquitectura:



ELIMINAR NODO

Para eliminar un *nodo* utilizaremos la subacción `leave`, ejecutando el comando en el servidor donde queremos realizar dicha acción. Se puede ejecutar dentro de un *nodo de trabajo* o en un *nodo administrador*.

```
worker2:~# docker swarm leave
Node left the swarm.

manager1:~# docker node ls
ID           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
by64v13d13ddqo4vb5xsrmgal  worker1  Ready  Active
kf6hvedlrs018emhx3thm6rx7 * manager1  Ready  Active      Leader
pjx4t4qpv3ujfdrloudzdpqwfkhi  worker2  Down   Active
```

Si ejecutamos la acción en un *nodo administrador* siendo el único, toda la configuración del *cluster* se borrará.

Para eliminar un nodo de la configuración, utilizaremos el siguiente comando:

```
manager1:~# docker node rm worker2
worker2

manager1:~# docker node ls
ID           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
by64v13d13ddqo4vb5xsrmgal  worker1  Ready  Active
kf6hvedlrs018emhx3thm6rx7 * manager1  Ready  Active      Leader
```

DOCKER SERVICE

Hemos visto las *subacciones* y las opciones básicas para `docker service`. En esta sección lo detallaremos con mayor profundidad.

Las subacciones disponibles son las siguientes:

`ls [opciones]`

Lista los servicios de Swarm, las opciones son las siguientes:

- `-f/-filter filtro`: filtra la salida a través del filtro especificado.
- `-q`: muestra solo los identificadores.

Ejemplo:

```
manager1:~# docker service ls
ID      NAME      MODE      REPLICAS  IMAGE
9arlkpxodnfu servicio2 replicated 3/3    httpd:latest
n791zqvnycyh6 servicio1 replicated 3/3    dockercloud/hello-world:latest
manager1:~# docker service ls -f name=servicio1
ID      NAME      MODE      REPLICAS  IMAGE
n791zqvnycyh6 servicio1 replicated 3/3    dockercloud/hello-world:latest
```

inspect [opciones] servicio

Muestra información detallada de un servicio. Las opciones son las siguientes:

- **-f/-format formato:** formatea la salida mostrada.
- **--pretty:** muestra la salida en formato amigable y no en JSON.

Ejemplo:

```
manager1:~# docker service inspect --pretty servicio1
ID:n791zqvnycyh6v9wttxvc6qu4y
Name:servicio1
Service Mode:Replicated
Replicas:3
Placement:
UpdateConfig:
Parallelism:1
On failure:pause
Max failure ratio: 0
ContainerSpec:
  Image:dockercloud/hello-world:latest@sha256:7[...]
Resources:
Endpoint Mode:vip
Ports:
  PublishedPort 8080
    Protocol = tcp
    TargetPort = 80
# docker service inspect --format='{{.Spec.Mode.Replicated.Replicas}}' servicio1
3
```

ps [opciones] servicio

Muestra las tareas (contenedores) y su estado del servicio especificado. Opciones:

- **-f/-filter filtro:** filtra la salida por el filtro especificado.
- **-q:** muestra solo los identificadores.
- **--no-trunc:** no limita el número de caracteres a mostrar.

Ejemplos:

```
manager1:~# docker service ps servicio1
ID          NAME      IMAGE           NODE  DESIRED STATE CURRENT STATE    ERROR PORTS
ry88s1f7ver servicio1.2 dockercloud/hello-world:latest manager1  Running   Running 32 minutes ago
4fmyr2asjy47s servicio1.2 dockercloud/hello-world:latest worker2  Running   Running 32 minutes ago
83gwmrmr7as2 servicio1.2 dockercloud/hello-world:latest worker2  Running   Running 32 minutes ago
manager1:~# docker service ps servicio1 -f name=servicio1.2
ID          NAME      IMAGE           NODE  DESIRED STATE CURRENT STATE    ERROR PORTS
ry88s1f7ver servicio1.2 dockercloud/hello-world:latest manager1  Running   Running 36 minutes ago
```

scale servicio=replicas

Escala un servicio al número de *replicas* especificado.

create [opciones] imagen

Crea un *servicio* utilizando la *imagen* especificada. La lista de opciones se muestra en la siguiente tabla:

Opción	Descripción	Ejemplo
<code>--dns servidores</code>	Lista de servidores DNS a usar.	<code>--dns 8.8.8.8</code>
<code>--dns-search servs</code>	Lista de nombres DNS a buscar.	<code>--dns-search docker.com</code>
<code>--env variables</code>	Lista de variables a utilizar	<code>--env USER=root</code>
<code>--env-file fich</code>	Fichero que incluye variables de entorno.	<code>--env-file dev.env</code>
<code>--health-cmd comanda</code>	Comando para comprobar el estado.	<code>--health-cmd comprobar.sh</code>

Opción	Descripción	Ejemplo
<code>--health-interval</code> <i>núm</i>	Intervalo de comprobación.	<code>--health-interval 10s</code>
<code>--health-retries</code> <i>núm</i>	Número de intentos.	<code>--health-retries 6</code>
<code>--health-timeout</code> <i>núm</i>	Máximo tiempo a esperar por la comprobación.	<code>--health-timeout 2s</code>
<code>--hostname</code> <i>nombre</i>	El nombre de host para el contenedor.	<code>--hostname servweb</code>
<code>--label</code> <i>etiquetas</i>	Etiquetas a establecer.	<code>--label env=dev</code>
<code>--limit-cpu</code> <i>núm</i>	Limitar el uso de CPU (defecto 0.000)	<code>--limit-cpu 0.60</code>
<code>--limit-memory</code> <i>bytes</i>	Limitar el uso de Memoria (defecto 0B)	<code>--limit-memory 128MB</code>
<code>--log-driver</code> <i>driver</i>	El driver a utilizar para los registros.	<code>--log-driver journald</code>
<code>--log-opt</code> <i>opciones</i>	Las opciones para el driver de registros.	<code>--log-opt tag=servweb</code>
<code>--mode</code> <i>modo</i>	El modo de servicio, dos opciones: <ul style="list-style-type: none"> • <i>replicated</i> (por defecto) • <i>global</i> <i>En modo global</i> creará un contenedor en cada nodo. No compatible con <code>--replicas</code> .	<code>--mode global</code>
<code>--name</code> <i>nombre</i>	Nombre a utilizar para el servicio.	<code>--name servweb</code>
<code>--network</code> <i>red</i>	La red a utilizar para los contenedores.	<code>--network ingress</code>
<code>--no-healthcheck</code>	No hacer comprobación del estado.	
<code>--publish</code> <i>puerto</i>	Publicar un puerto para ser accesible.	<code>--publish 8080:80</code>
<code>--replicas</code> <i>numero</i>	El número de contenedores a ejecutar cuando se utiliza el modo por defecto <i>replicated</i> .	<code>--replicas 3</code>

Opción	Descripción	Ejemplo
<code>--reserve-cpu num</code>	Reserva de CPU.	<code>--reserve-cpu 0.60</code>
<code>--reserve-memory num</code>	Reserva de memoria.	<code>--reserve-memory 128MB</code>
<code>--restart-condition val</code>	La condición para reiniciar el contenedor cuando es detenido. Opciones: • none • on-failure • any	<code>--restart-condition on-failure</code>
<code>--restart-delay num</code>	Demora para reiniciar el contenedor.	<code>--restart-delay 5s</code>
<code>--restart-max-attempts num</code>	El número máximo de intentos para reiniciar un contenedor.	<code>--restart-max-attempts 5</code>
<code>--user usuario</code>	El usuario a usar dentro de los contenidos.	<code>--user www-data</code>
<code>--workdir directorio</code>	El directorio a trabajar dentro del contenedor.	<code>--workdir /var/www/html/</code>

En el siguiente ejemplo se utiliza el modo global y se limita la memoria:

```
manager1:~# docker service create --mode global --name servicio4 --limit-memory 128MB \
> --publish 8083:80 httpd
k2228d5feldwvvtwczz149i7
manager1:~# docker service ps servicio4
ID          NAME      IMAGE      NODE      DESIRED STATE CURRENT STATE      ERROR PORTS
534u7utsoky servicio4 [...] httpd:latest worker1  Running      Running 5 seconds ago
eg8msaq5878h servicio4 [...] httpd:latest worker2  Running      Running 5 seconds ago
2mfrkqgg1jon servicio4 [...] httpd:latest manager1  Running      Running 5 seconds ago
```

update [opciones] servicio

Actualiza un servicio con las opciones específicas; las opciones son similares a las de `create`:

Opción	Descripción	Ejemplo
<code>--dns-add servidores</code> <code>--dns-rm servidores</code>	Añade o elimina servidores DNS a usar.	<code>--dns-add 8.8.8.8</code> <code>--dns-rm 8.8.8.8</code>
<code>--dns-search-add srv</code> <code>--dns-search-rm srv</code>	Añade o elimina nombres DNS a buscar.	<code>--dns-search-add .local</code> <code>--dns-search-rm .local</code>
<code>--env-add variables</code> <code>--env-rm variables</code>	Añade o elimina variables de entorno a utilizar.	<code>--env-add USER=root</code> <code>--env-del USER=root</code>
<code>--health-cmd comando</code>	Comando para comprobar el estado.	<code>--health-cmd comprobar.sh</code>
<code>--health-interval num</code>	Intervalo de comprobación.	<code>--health-interval 10s</code>
<code>--health-retries num</code>	Número de intentos.	<code>--health-retries 6</code>
<code>--health-timeout num</code>	Máximo tiempo a esperar por la comprobación.	<code>--health-timeout 1s</code>
<code>--hostname nombre</code>	El nombre de host para el contenedor.	<code>--hostname servweb</code>
<code>--label-add etiquetas</code> <code>--label-rm etiquetas</code>	Añade o elimina etiquetas a los contenedores.	<code>--label-add version=0.2</code> <code>--label-rm version=0.1</code>
<code>--limit-cpu num</code>	Limitar el uso de CPU (defecto 0.000)	<code>--limit-cpu 0.60</code>
<code>--limit-memory bytes</code>	Limitar el uso de memoria (defecto 0B)	<code>--limit-memory 128MB</code>
<code>--log-driver driver</code>	El driver a utilizar para los registros.	<code>--log-driver journald</code>
<code>--log-opt opciones</code>	Las opciones para el driver de registros.	<code>--log-opt tag=servweb</code>
<code>--no-healthcheck</code>	No hacer comprobación del estado.	
<code>--publish-add puerto</code> <code>--publish-rm puerto</code>	Publica o despública un puerto para ser accesible.	<code>--publish-add 8088:80</code> <code>--publish-rm 8080:80</code>
<code>--replicas numero</code>	El número de contenedores a ejecutar cuando se utiliza el modo por defecto replicated.	<code>--replicas 3</code>

Opción	Descripción	Ejemplo
<code>--reserve-cpu num</code>	Reserva de CPU.	<code>--reserve-cpu 0.60</code>
<code>--reserve-memory num</code>	Reserva de memoria.	<code>--reserve-memory 128MB</code>
<code>--restart-condition val</code>	La condición para reiniciar el contenedor cuando es detenido. Opciones: <ul style="list-style-type: none">• <code>none</code>• <code>on-failure</code>• <code>any</code>	<code>--restart-condition on-failure</code>
<code>--restart-delay num</code>	Demora para reiniciar el contenedor.	<code>--restart-delay 5s</code>
<code>--restart-max-attempts num</code>	El número máximo de intentos para reiniciar un contenedor.	<code>--restart-max-attempts 5</code>
<code>--user usuario</code>	El usuario a usar dentro de los contenedores.	<code>--user www-data</code>
<code>--workdir directorio</code>	El directorio a trabajar dentro del contenedor.	<code>--workdir /var/www/html/</code>

DOCKER NODE

Las subacciones y las opciones para la acción `node` se detallan a continuación:

`ls [opciones]`

Lista los *nodos* de Swarm, las opciones son las siguientes:

- `-f/-filter filtro`: filtra la salida a través del filtro especificado.
- `-q`: muestra solo los identificadores.

Ejemplo:

```
manager1:~# docker node ls -f name=manager1
ID          HOSTNAME STATUS AVAILABILITY MANAGER STATUS
kf6hvedlrs018emhx3thm6rx7 * manager1  Ready Active      Leader
```

ps [opciones] [nodo]

Muestra los contenedores de Swarm las opciones son las siguientes:

- **-f/–filter** filtro: filtra la salida por el filtro especificado.
- **--no-trunc**: no limita el número de caracteres a mostrar.

Ejemplo:

```
manager1:~# docker node ps -f name=servicio1.3 worker1
ID          NAME      IMAGE      NODE      DESIRED STATE     CURRENT STATE
61gwanmr7n6J servicio1.3  dockercloud/hello-world:latest worker1  Running  2 hours ago
```

inspect [opciones] [nodo]

Muestra los contenedores de Swarm, las opciones son las siguientes:

- **-f/–format** formato: formatea la salida mostrada.
- **--pretty**: muestra la salida en formato amigable y no en JSON.

Ejemplos:

```
manager1:~# docker node inspect --pretty worker1
ID:by64v13d13ddqo4vb5xsrmgal
Hostname:worker1
Joined at:2017-01-29 10:28:02.701512479 +0000 utc
Status:
  State:Ready
  Availability:Active
  Address:192.168.56.106
Platform:
  Operating System:linux
  Architecture:x86_64
Resources:
  CPUs:2
  Memory:992.2 MiB
Plugins:
  Network:bridge, host, macvlan, null, overlay
  Volume:local
Engine Version:1.13.0
```

```
manager1:~# docker node inspect -f "{{.Description.Engine.EngineVersion}}" worker1
1.13.0
```

`rm [-f] [nodo]`

Elimina un *nodo* del *cluster Swarm*.

`promote [nodo]`

Promociona un *nodo de trabajo* a *nodo administrador*.

Ejemplo:

```
manager1:~# docker node promote worker2
Node worker2 promoted to a manager in the swarm.
manager1:~# docker node ls
ID          HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0x15f3f0ihb2In2eqkng0v1i  worker2  Ready  Active        Reachable
by64v13d13ddqo4vb5xsrngeal  worker1  Ready  Active
kj6khvedlrs018ermhx3thm6rx7 * manager1  Ready  Active        Leader
```

`demote [nodo]`

Degrada un *nodo administrador* a *nodo de trabajo*.

Ejemplo:

```
manager1:~# docker node demote worker2
Manager worker2 demoted in the swarm.
```

`update [opciones] [nodo]`

Actualiza ciertos parámetros de un *nodo*. Las opciones son las siguientes:

- **--availability modo**: la disponibilidad del nodo, tres opciones posibles:
- **active**: el nodo está activo y es posible desplegar contenedores en él.
- **pause**: en dicho nodo no es posible desplegar nuevos contenedores.

- **drain:** en dicho nodo no es posible desplegar nuevos contenedores y se detendrán todos los contenedores que se están ejecutando en él, ejecutándose en otros nodos.
- **--label-add etiquetas:** añade etiquetas a un nodo.
- **--label-rm etiquetas:** elimina etiquetas a un nodo.
- **--role rol:** especifica el rol de un nodo; worker (nodo de trabajo) o manager (nodo administrador)

Ejemplo:

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
i01ppwp4sup	servicio4[...]	httpd:latest	worker2	Running	Running 25 minutes ago
imhryngycq0	servicio3.2	httpd:latest	worker2	Running	Running 5 minutes ago
veawvycholu9	servicio2.2	httpd:latest	worker2	Running	Running 5 minutes ago
oghp4wifjk	servicio2.3	httpd:latest	worker2	Running	Running 5 minutes ago
manager1:# docker node update --availability drain worker2					
worker2					
ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
i01ppwp4sup	servicio4[...]	httpd:latest	worker2	Shutdown	Shutdown about a minute ago
imhryngycq0	servicio3.2	httpd:latest	worker2	Shutdown	Shutdown about a minute ago
veawvycholu9	servicio2.2	httpd:latest	worker2	Shutdown	Shutdown about a minute ago
oghp4wifjk	servicio2.3	httpd:latest	worker2	Shutdown	Shutdown about a minute ago

ÍNDICE ANALÍTICO

A

Almacenamiento
 AUFS, 99
 Device mapper, 105
 OverlayFS, 103
 OverlayFS2, 103
 ZFS, 115

C

chroot, 1
Componentes
 Docker Cloud, 10
 Docker compose, 10
 Docker Hub, 10
 Docker para Linux, 10
 Docker para Mac, 10
 Docker para Windows, 10
Contenedores, 39
 contenido, 7
 exponer puertos, 35
 modo interactivo, 31
 segundo plano, 32

D

Docker, 4
`docker attach`, 41
`docker build`, 74
Docker Cloud, 181

Cluster de nodos, 182
Docker Cloud Agent, 184
Endpoints, 187
Escalar, 191
Nodo, Véase
 Repositorios, 193
Servicio, 182
Stack, 182
`docker commit`, 64
Docker Compose, 159
`docker create`, 39
`docker events`, 54
`docker exec`, 45
`docker export`, 48
`docker history`, 62
Docker Hub, 213
 Repositorios, 215
 Webhooks, 218
Docker ID, 67
`docker images`, 57
`docker import`, 63
`docker inspect`, 30
`docker kill`, 44
`docker login`, 67
`docker logout`, 68
`docker logs`, 29
`docker network`, 90
`docker node`
 `docker node ls`, 239
 `docker node inspect`, 238
 `docker node ls`, 237

docker <u>node</u> promote, 239	
docker <u>node</u> ps, 238	
docker <u>node</u> rm, 239	
docker <u>node</u> update, 239	Etiquetas, 127
docker <u>pause</u> , 44	
docker <u>ps</u> , 28	I
docker <u>pull</u> , 60	Imágenes, 57
docker <u>push</u> , 68	Instalación, 11
docker <u>rename</u> , 42	Iptables, 38
docker <u>restart</u> , 42	L
docker <u>rm</u> , 45	Limitar recursos, 133
docker <u>rmi</u> , 63	Logging drivers, 145
docker <u>run</u> , 27	O
docker <u>save</u> , 62	OpenVZ, 2
docker <u>search</u> , 59	R
docker service	Redes, 91
docker <u>service create</u> , 233, 235	Registros, 145
docker <u>service inspect</u> , 232	Repositorio local, 68
docker <u>service ls</u> , 231, 232	S
docker <u>service ps</u> , 233	Servicio, 20
docker <u>service scale</u> , 233	Swarm, 221
docker <u>start</u> , 34	V
docker <u>stop</u> , 42	Virtualización, 2
docker <u>swarm init</u> , 224	W
docker <u>swarm join</u> , 225	Windows, 22
docker <u>swarm join-token</u> , 225	
docker <u>unpause</u> , 44	
docker-cloud, 202	
docker-compose	
docker-compose <u>down</u> , 166	
docker-compose <u>ps</u> , 166	
docker-compose <u>start</u> , 166	
docker-compose <u>stop</u> , 166	
docker-compose <u>up</u> , 165	
Dockerfile, 73	

