Administración de aplicaciones Docker

Alejandro Martínez Fernández

Segundo Cuatrimestre del curso 2017/2018

Índice

Resumen

En este documento expongo las actividades realizadas durante el curso 2017/18 durante mi estancia como alumno interno de DITEC bajo la tutela del profesor Juan Piernas Cánovas.

Las actividades que he realizado han sido investigar el funcionamiento de los contenedores Docker, aquí expongo un manual de uso bastante completo.

Script de instalación

```
#Shell instalador de docker en Fedora

dnf -y update

dnf -y upgrade

dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo

dnf -y install docker-ce

systemctl start docker
```

1. Descripción y definición de conceptos

- **Imagen:** Una Imagen son, básicamente, instrucciones para la creación de un contenedor. Estas son de solo lectura.
- **Contenedor:** Es una instancia ejecutable de una imagen. Se pueden crear, poner en marcha, parar, mover o borrar. Incluso se puede crear una Imagen a partir del estado actual de un contenedor.
- **Volumen:** Es la mejor forma para que los datos en un contenedor se almacenen de una manera persistente y se conserven aún después de eliminar el contenedor. Facilita que varios contenedores compartan el sistema de ficheros. Son administrados directamente por Docker, por lo que son sencillos de mantener. El usuario puede crearlos, borrarlos y modificarlos.
- **Montaje ligado:** Parecidos a los volúmenes, aunque tienen una funcionalidad más limitada. Mientras que los volúmenes se almacenan de manera aislada, los montajes ligados consisten en montar un fichero o directorio en un contenedor. El fichero es referenciado por su ruta.

  Peligroso: El sistema de ficheros de la máquina queda expuesto ante los programas del contenedor por lo que podrían modificarlo.
- **Montaje tmpfs:** Esta opción solo se puede utilizar en linux, y se utiliza cuando no queremos que los datos del contenedor perduren en el tiempo. Esto puede ser así por seguridad, porque el contenedor puede manejar una gran cantidad de datos o por cualquier otro motivo.
- **Registry:** Un registry es una base de datos de imágenes. O dicho de otra forma, un sitio donde guardar las imágenes, un hub. El registry más famoso es el Docker Hub o Docker Store, donde podemos crear una cuenta de usuario y subir nuestras imágenes (también descargarlas). Registry es en sí otra imágen, por lo que sí, podemos descargarla, hacer un contenedor con ella y guardar en el mismo nuestras imágenes.

2. Administración de imágenes y contenedores

2.1. Pull

Empecemos la administración, actualmente no tenemos ninguna imagen con la que crear contenedores, así que lo primero que debemos hacer es descargar alguna. Para descargar una imagen utilizamos el comando pull.

Descargamos la imagen "hello-world":

docker pull hello-world

Además, este comando tiene opciones. Una opción es $-a, --all$ que hace que se descarguen todas las versiones de una imagen.

Ejemplo (**NO** usar, demasiadas versiones):

docker -a docs/docker.github.io

La otra opción, aunque no muy relevante, es *--disable-content-trust* que no verifica el contenido de la imagen.

2.2. Create

Ahora que tenemos una imagen descargada, vamos a crear un contenedor con ella.

docker create hello-world

Como podemos ver, no se ha producido salida alguna, salvo el id del contenedor. Esto es así porque el contenedor se ha creado, pero no se ha puesto en marcha.

2.3. Start

Ahora vamos a activar el contenedor que acabamos de crear, para ello escribimos:

docker start -a **ID**
Siendo **ID** *la salida del comando anterior.*

Este comando activa contenedores que están parados. Estos contenedores pueden estar parados porque que se les haya detenido desde fuera (ahora lo vemos) o porque se hayan creado pero no se hayan puesto en marcha. Opciones:

- **-a, --attach**

  Une la salida estándar y de error a la del terminal.

- **--detach-keys**

  Anula la secuencia de teclas para separarla del contenedor.

- **-i, --interactive**

  Une la entrada estándar del contenedor a la del terminal (muy interesante con otras imágenes que veremos más adelante).

2.4. Run

Ahora que ya sabemos como descargar imágenes, crear contenedores y ponerlos en marcha se nos viene a la mente una pregunta ¿No hay una forma más rápida?. La respuesta es que sí, y es el comando run. Vamos a ver un ejemplo rápido y sencillo para entenderlo:

docker run hello-world

Ahora, para ver como funcionan algunas de las muchas opciones que tiene este comando vamos a ver más ejemplos:

docker run -it ubuntu

Con este comando creamos y entramos a controlar un contenedor cuya imagen es un SO. En este contenedor podemos escribir los comandos de bash que conocemos, desde un find hasta un apt, pasando por cualquier comando instalado. Además, dentro de este contenedor hay un sistema de ficheros, como el de cualquier sistema linux. Vamos a escribir las siguientes órdenes en este contenedor:

cd /home
mkdir usuario
cd usuario
mkdir Escritorio
cd Escritorio
echo "Creo un fichero y le introduzco un texto" >Mifichero.txt
exit

Las opciones que hemos puesto en el anterior comando son:

- **-i**

  interactive, mantiene abierta la entrada estándar del contenedor en el términal. Necesaria para los sistemas operativos.

- **-t**

  Asgina un pseudo -TTY, lo que hace que sea correcto su uso para la utilización de un shell interactivo.

Otra cosa de la que podemos habernos dado cuenta es que nosotros no teníamos descargada la imagen ubuntu, pero aún así se ha creado el contenedor apropiado. Esto es debido a que, tanto run como create, descargan la imagen ordenada del Docker Hub cuando no la encuentran localmente.

Otro ejemplo muy interesante:

docker run -t -p 4000:4000 docs/docker.github.io:latest

La imagen que acabamos de utilizar es la propia documentación oficial de Docker, que podemos visualizar, como nos dice la propia salida, escribiendo 0.0.0.0:4000 en un navegador. Las opciones que hemos puesto en este comando son:

- **-t**

  El efecto es el mismo que antes, pero en este caso, la diferencia entre ponerlo y no ponerlo, es que si pulsamos las teclas **Ctrl + C** y la opción no está puesta, recuperamos el prompt del sistema, pero el contenedor termina y la documentación ya no es accesible. Si la opción está puesta, como en este caso, podemos recuperar el terminal y seguir visualizando la documentación.

- **-p X:Y**

  El contenedor está escuchando en su puerto Y (podemos imaginar que es un puerto virtual), lo que hacemos es que el puerto X de la máquina se iguala a ese puerto Y (que es el que especifica la imagen). De esta forma es como se consigue que un contenedor haga uso de la red.

El comando run tiene muchas más opciones aparte de las vistas en los ejemplos anteriores. A continuación se muestran algunas opciones más:

- **-a, --attach**

  Liga el terminal con la salida estandar, la entrada estandar o la salida de error estandar del programa ejecutado. Esta opción debe ejecutarse en modo de primer plano y es la que se utiliza por defecto. No es necesario ligar las tres salidas, sino que puede elejirse que salidas se ligan.

- **--add-host=[]**

  Añade un host IP custom con el formato host:ip También añade una línea a /etc/hosts. Esta opción se puede usar varias veces.

- **-c=X, --cpu-shares=X**

  Por defecto se utiliza la opción -c=1024. El parámetro X se utiliza para repartir de manera proporcional el tiempo de CPU entre todos los contenedores cuando estos intentan utilizar más del 100Si por ejemplo tenemos tres contenedores ejecutándose en el sistema con 512, 512 y 1024 respectivamente, el tiempo se repartirá de la siguiente forma: 25Nota: Si se disponen de varios nucleos, el sistema repartirá el tiempo de CPU llevando en cuenta todos ellos.

- **--cpu-count=X**

  Limita el número de CPUs disponibles para la ejecución de contenedores a X. En Windows Server se aproxima al porcentaje de uso de la CPU.

- **--cpu-percent=X**

  Limita el porcentaje de uso de CPU disponible para la ejecución de contenedores a X.

- **--cpu-period=X**

  Esta opcíon se utiliza para modificar el tiempo de CPU que el planificador concede a los procesos de los contenedores.

- **--cpu-rt-period=X**

  Con esta opción podemos limitar el periodo de uso de CPU a un determinado tiempo real en microsegundos.

- **--dns=[]**

  Utiliza un servidor de DNS diferente al por defecto.

  Esta opción puede ser usada para sobrescribir la configuración que se le proporciona al contenedor. Normalmente esta opción es necesaria cuando la configuración DNS falla. En este caso, siempre que se ejecute algo habrá que utilizarla.

- **-m, --memory=“”**

  Permite añadir una restricción de memoria al contenedor con el formato <número>[<unidad>], donde unidad puede ser b, k, m o g. Si el sistema soporta memoria swap, esta restricción puede ser mayor que la propia memoria física.

- **--memory-reservation=“”**

  Mismo formato que el anterior. Al añadir este flag, si el sistema detecta que hay poca memoria restante, el sistema obliga a los contenedores a limitar su consumo a lo que se les haya reservado con este comando. Cuando hagamos uso de esta opción deberiamos establecer un límite por debajo del de la opción anterior (pues la anterior opción es como un límite hard y esta como uno soft). Por defecto, la memoria reservada es la misma que la establecida con la opción anterior.

- **--memory-swap=“”**

  Establece el límite de memoria swap que puede utilizar un contenedor. Esta opción debe ser siempre utilizada con la opción -m. El formato utilizado con esta opción es el mismo que el de las dos opciones anteriores, pero si no especificamos unidad se utiliza b. El límite establecido debería ser mayor que el de la opción -m. Por defecto se establece al doble de la opción -m. Si no queremos limitar la memoria swap, utilizaremos -1 como límite.

- **--name**

  Establece un nombre para el contenedor que se va a poner en ejecución. Esta opción es muy útil, pues luego podemos referirnos al contenedor por su nombre (haremos esto) en lugar de por su ID (si no se da un nombre a un contenedor, se le asigna uno aleatorio).

2.5. Ps

Ahora que hemos puesto algunos contenedores en marcha, vamos a entrar ver el estado general del sistema. Para esto vamos a utilizar la orden *ps* que lista los contenedores. Vamos a ver un ejemplo:

docker ps

docker ps -a

docker ps --filter status=exited

El primer comando muestra todos los contenedores que se estén ejecutando en este momento. La salida del comando es la siguiente:

1. **CONTAINER ID**

   ID "corto" del contenedor, sobran las descripciones.

2. **IMAGE**

   Imagen utilizada para crear el contenedor.

3. **COMMAND**

   Programa que se está ejecutando dentro del contenedor.

4. **CREATED**

   Cuanto tiempo ha pasado desde que se creó.

5. **STATUS**

   Estado actual del contenedor (corriendo, terminado, pausado, etc).

6. **PORT**

   Puertos utilizados por el contenedor.

7. **NAME**

   Nombre del contenedor (el que hayamos puesto al crearlo o le haya asignado el sistema).

El resto de comandos utilizan unas opciones las cuales vamos a explicar a continuación junto con el resto de opciones del comando:

- **-a, --all**

  Muestra todos los contenedores.

- **-s, --size**

  Añade una columna con el tamaño de los ficheros del contenedor.

- **-q, --quiet**

  Solo muestra los IDs de los contenedores.

- **--no-trunc**

  No recorta el tamaño de la salida.

- **-n X, --last X**

  Muestra sólo los X últimos contenedores creados.

- **--latest, -l**

  Muestra sólo el último contenedor creado. Equivale a "-n 1".

- **--format**

  Usando una plantilla Go, podemos formatear la salida.

- **-f, --filter**

  Con esta opción, podemos añadir condiciones. En la salida solo se mostrarán los contenedores que las cumplan. Las condiciones que podemos poner pueden abarcar cosas como el id, el nombre o las networks que utilizan.

2.6. Stop

Ahora que hemos visto como poner contenedores en marcha, vamos a ver como pararlos. Para esto utilizamos el comando *stop*. Tras un periodo de gracia (que desconozco cuál es por defecto), los mata si no se han parado. Después de esto, si vemos su estado con *docker ps*, será terminado. Vamos a ver un ejemplo:

docker stop NOMBRE

En el campo **NOMBRE** pon el nombre de cualquiera de los contenedores que hemos abierto hasta ahora. Este comando solo tiene una opción, **-t, --time** que especifica el periodo de gracia en segundos a esperar por el comando antes de matarlo. Si a esta opción no le acompañaun tiempo, usa diez segundos.

2.7. Rename

Y ahora, ¿qué pasa si creando un contenedor me olvido de darle un nombre o me equivoco? Bien, Docker ha pensado también en esto. Con el comando **rename** podemos cambiar el nombre de un contenedor. Vamos a ver un ejemplo:

docker rename NOMBRE nombre_ nuevo

Ahora, el antiguo contenedor **NOMBRE** se llama ***nombre_ nuevo***.

2.8. Pause

Otro comando muy interesante es el comando *pause* que para la ejecución de un contenedor (más concretamente, para a los procesos de dentro del contenedor) de la misma forma que lo hace la combinación de teclas **Ctrl** + **Z** en un terminal. Ejemplo:

docker pause NOMBRE

(Como siempre, ***NOMBRE*** *es el nombre de un contenedor. No lo voy a decir más)*

Si a continuación utilizamos el comando *docker ps* veremos que el contenedor sigue en ejecución, pero ha sido detenido.

2.9. Unpause

Acabamos de parar los procesos de un contenedor, ahora vamos a ver como reanudarlos. El comando que vamos a utilizar es **unpause**. Para utilizarlo simplemente seguimos la sintaxis del ejemplo:

docker unpause NOMBRE

2.10. Kill

Ahora vamos a matar contenedores, para ello vamos a usar el comando **kill**.

docker kill NOMRE

Si utilizamos el comando *docker ps -a* podemos ver que la situación final del contenedor es la misma que utilizando la orden stop. La diferencia entre las ordenes es la señal utilizada, pues aquí usamos SIGKILL. Otra característica muy interesante es que la única opción disponible *(-s, --signal)* nos permite enviar otra señal al contenedor. Nota: En realidad la señal se envía al proceso principal del contenedor, no al contenedor en sí.

docker kill -s KILL NOMBRE

2.11. Restart

La siguiente orden que vamos a ver es **restart** que reinicia o relanza un contenedor. Esta opción puede ser interesante si un contenedor da problemas. Además este comando no afecta al espacio de memoria de un contenedor, por lo que no se pierden cambios. Ejemplo:

docker restart NOMBRE

2.12. Update

Antes hemos hablado sobre el problema de equivocarnos poniendo el nombre de un contenedor, pero, nos podemos equivocar en otros detalles (mucho más importante que el nombre) en la creación de este. Por esta razón existe el comando **update** que actualiza la configuración de un contenedor. De esta forma podemos, tanto corregir errores como poner la configuración de un contenedor al día si por ejemplo cambian el contexto de la máquina (durante la realización de esta guía, podemos ir creando más contenedores según veamos y jugando un poco con ellos). Veamos un ejemplo:

docker update --cpus 1 NOMBRE

La explicación de este comando recae en las opciones utilizadas, pues es en ellas donde se especifica la nueva configuración del contenedor. Vamos a ver algunas de las opciones que tiene este comando:

- **--cpu-period**

  Modifica el tiempo de CPU que el planificador concede a los procesos de los contenedores.

- **--cpu-quota**

  Modifica el porcentaje de tiempo de CPU que el planificador concede a los procesos contenedores.

- **--cpus**

  Limita el número de CPUs que puede utilizar el contenedor.

- **-m, --memory**

  Limita la memoria que puede usar el contenedor.

- **--memory-reservation**

  Cuando se detecta que queda poca memoria restante el sistema obliga a los contenedores a utilizar como mucho esta cantidad de memoria. Evidentemente, esta debe ser menor que la memoria límite.

- **--memory-swap**

  Establece el limite de memoria swap que puede utilizar un contenedor.

2.13. Attach

Un contenedor ya en ejecución se puede unir a un terminal (como si hubiesemos utilizado la opción -a del run) con el comando **attach**. Ejemplo:

docker run --name nombre -it ubuntu
Realizamos acciones dentro del contenedor y al final usamos exit
docker start nombre
docker attach nombre

Si hacemos esto, podemos ver que nos hemos vuelto a conectar al mismo contenedor del que habíamos salido. Esta orden es muy interesante cuando, por ejemplo, apagamos la máquina y queremos volver a conectarnos al contenedor.

Opciones:

1. **--no-stdin**

   No liga la entrada estandar, interesante si solo queremos la salida.

2. **--sig-proxy**

   Atrapa todas las señales del proceso.

2.14. Port

El siguiente comando (**port**) muestra los puertos que está utilizando un contenedor. Estos puertos no están separados de los de la máquina real, por lo que es así como se consigue la comunicación del contenedor con el exterior. Vamos a ver un ejemplo:

docker run -it --name contenedor -p 4000:4000 docs/docker.github.io:latest
docker port contenedor

2.15. Cp

Describamos una situación ficticia, hemos estado trabajando dentro de un contenedor con un fichero, y queremos copiarlo para enviarlo a alguien o tenerlo guardado en otro sitio. Para esto vamos a utilizar el comando **cp** que copia ficheros entre contenedores y el sistema de ficheros local. Ejemplo:

docker cp nombre:/FICHERO /RUTA

/FICHERO es la ruta del fichero del contenedor **nombre** que queremos copiar a **RUTA**. Si el fichero que se va a copiar es un directorio, este se copiará recursivamente (por lo que podemos decir que docker cp equivale al cp -a de Unix).

Opciones:

- **-a, --archive**

  Copia también los UID y GID de los ficheros.

- **-L, --follow-link**

  Si en alguna de las rutas, se utiliza un enlance simbólico a un directorio, este se sigue.

2.16. Rm

Después de haber creado y administrado contenedores, le llega el turno a **rm** que borra contenedores que estén parados. Si se borran, ya no se podrán volver a activar (de hecho, dejarán de aparecer en *docker ps -a*). Ejemplo:

docker rm NOMBRE

Opciones:

- **-f, --force**

  Fuerza la eliminación del contenedor elegido. Para esto envía la señal SIGKILL. Aunque los contenedores deben estar parados, pueden estar corriendo si usamos esta opción.

- **-l, --link**

  Borra los enlaces especificados.

- **-v, --volumes**

  Borra los volúmenes asociados a los contenedores.

2.17. exec

Ahora vamos a ver el comando **exec**, una orden que creo tiene mucha utilidad, pues, por ejemplo, deja de ser necesario entrar dentro de un contenedor para ejecutar un solo comando. Como ya se puede deducir, *exec* ejecuta un comando dentro de un contenedor. El comando utilizado solo estará activo mientras el proceso principal del contenedor lo esté. El comando no se reiniciará si el contenedor es reiniciado. Se utilizará el directorio por defecto del contenedor. Si la imagen subyacente tiene especificado un directorio diferente, se usará este en su lugar. El comando deberá ser ejecutable. Veamos un ejemplo:

docker exec NOMBRE -t top

Opciones:

- **-d, --detach**

  Ejecutar en segundo plano con respecto a este terminal.

- **-i, --interactive**

  Mantiene la entrada estándar abierta aunque no se haya ligado.

- **--privileged**

  Da privilegios extendidos al comando.

- **-t, --tty**

  Establece un pseudo-tty.

- **-u, --user**

  UID.

2.18. Export

Export es el siguiente comando. Exporta el contenido del sistema de ficheros de un contenedor como un fichero tar. Este comando es interesante para la realización de copias de seguridad. Debemos tener en cuenta que no funciona con volúmenes. Ejemplo:

docker export NOMBRE >comprimido.tar

Aquí solo tenemos una opción, *-o, --output* que introduce la salida en el fichero especificado en vez de en la salida estándar. Viendo esto se puede decir que utiliza la misma sintaxis que cpio.

2.19. Wait

Wait bloquea la terminal hasta que el contenedor (o contenedores) seleccionados terminen su ejecución. Cuando se pare la ejecución del contenedor seleccionado, el terminal mostrará el código de salida del contenedor y se desbloqueará. Para este ejemplo necesitaremos dos terminales, las instrucciones en **negrita** irán a uno y las otras al otro:

docker run -it --name unnombre ubuntu
docker wait unnombre
exit

2.20. Events

El siguiente comando sirve para monitorizar los objetos docker. Muestra los eventos correspondientes a TODOS los objetos. Los eventos se muestran en tiempo real y son distintos para cada tipo de objeto. Ejemplo:

docker events
docker events --since '2017-01-05'
docker events --since '10m'
docker events --filter 'type=volume'

Opciones:

- **--format**

  Podemos utilizar una plantilla Go para formatear la salida.

- **--since**

  Solo muestra eventos ocurridos desde la marca de tiempo que se le proporcione.

- **--until**

  Solo muestra eventos ocurridos antes de la marca de tiempo proporcionada.

- **-f, --filter**

  Filtra la salida según las condiciones proporcionadas De todos los comandos con filtro este es el más complejo, pero también el más completo. Si se pone más de un tipo de filtro igual (por ejemplo, dos ID), la condición se evaluará como OR. Si se ponen varios filtros de tipos diferentes, su condición se evalua como AND Posibles filtros: config, container,daemon, event, image, label, network, node, plugin, scope, secret, service, type y volume.

2.21. Stats

El comando que vamos a ver muestra en tiempo real las estadísticas de consumo de un contenedor. Ejemplo:

docker stats NOMBRE

Las opciones de esta orden son las siguientes:

- **-a, --all**

  Muestra la información de todos los contenedores (sin esta opción, solo los que están corriendo).

- **--format**

  Como siempre, podemos formatear la salida utilizando una plantilla Go.

- **--no-stream**

  No monitoriza los contenedores, solo muestra la primera impresión de datos.

- **--no-trunc**

  No recorta la salida.

2.22. Top

Ahora vamos a monitorizar, no los contenedores, sino los procesos que se están ejecutando dentro de uno de ellos. Para esto utilizamos la orden **top**. Ejemplo:

docker top NOMBRE

Este comando tiene las mismas opciones que la orden ps de Unix.

2.23. Inspect

El suguiente comando muestra información de bajo de nivel del objeto (imagenes, contenedores, volumenes, etc) analizado. La información suele mostrarse utilizando un array JSON.
Veamos un ejemplo:

docker inspect -s NOMBRE

Opciones:

- **-f, --format**

  Puedes proporcionarle una plantilla Go para que la salida se adapte a esta.

- **-s, --size**

  Si el objeto es un contenedor, se muestra el tamaño total de sus ficheros.

2.24. Login

Login se utiliza para conectarse a un "registry", que bien puede ser local o puede estar alojada en otro sitio. Para hacer un ejemplo, nos vamos a conectar al Docker Hub, para lo que deberemos tener cuenta.

docker login

Al no haber especificado un registry, se conecta al oficial.
Opciones:

- **-p, --password**

  Contraseña en el siguiente campo (NO RECOMENDABLE).

- **--password-stdin**

  Lee la contraseña de la entrada estándar (por defecto si no se utilizan ficheros de configuración).

- **-u, --username**

  Usuario en el siguiente campo.

2.25. Logout

Este es el comando opuesto, te desconecta de un registry al que estés conectado. Ejemplo:

docker logout localhost:8080

2.26. Images

Images lista las imágenes que tengamos descargadas en la máquina local.
Ejemplo:

docker images

Opciones:

- **-a, --all**

  Muestra todas las imágenes (incluyendo las intermedias).

- **--format**

  Se puede formatear la salida con una plantilla Go, como siempre.

- **-q, --quiet**

  Solo muestra los IDs númericos de las imágenes.

- **--no-trunc**

  No recorta la salida.

- **-f, --filter**

  Como es habitual en este tipo de comandos, podemos filtrar la salida. Las condiciones de filtrado en esta ocasión abarcan campos como la etiqueta o cuando fueron creadas.

2.27. Search

Pongamos de ejemplo que queremos una imagen con una funcionalidad concreta. Para encontrar una imagen así, tendríamos que ir a un navegador y buscarla. En su lugar, tenemos la alternativa **search** que busca lo que se le proporcione en el Docker Hub. Veamos un ejemplo:

docker search database

Esta orden tiene las siguientes opciones (algunas están obsoletas, por lo que no las voy a comentar):

- **-f, --filter**

  Igual que con el comando ps, aquí también podemos añadir filtros a la salida. Estos filtros pueden ser, el número mínimo de estrellas, si la imagen es oficial y si está automatizada.

- **--format**

  Podemos formatear la salida utilizando una plantilla Go.

- **--limit X**

  Limita la salida a las X imágenes con más estrellas (starred).

- **--no-trunc**

  No recorta la salida.

2.28. History

Esta orden muestra el historial de una imagen (que no un contenedor). La salida tiene los siguientes campos:

1. **ID**

   ID de la imagen.

2. **CreatedSince**

   Intervalo de tiempo que ha pasado desde que se creó la imagen.

3. **CreatedAt**

   Cuando se creó la imagen.

4. **CreatedBy**

   Comando utilizado para crear la imagen.

5. **Size**

   Tamaño que ocupa la imagen en disco.

6. **Comment**

   Comentario de la imagen.

Veamos un ejemplo:

docker history debian

Opciones:

- **--format**

  Utiliza una platilla Go para dibujar la salida.

- **-H, --human**

  Los datos numéricos se muestran en un formato legible.

- **--no-trunc**

  No trunca la salida.

- **-q, --quit**

  Solo muestra IDs numericos.

2.29. Save

El siguiente comando guarda una imagen en un archivo .tar. La salida la produce por la salida estándar, po lo que habrá que redirigirla. Pues ser útil para llevarnos una imagen de un sitio a otro sin tener que estar pensando en los registry. Ejemplo:

docker save hello-world >imagen.tar

Al igual que en export, tenemos la opción *-o, --output* para redirigir la salida a un fichero directamente.

2.30. Load

Ahora vamos a ver el comando opuesto a *save*, **load**. Este carga una imagen de un archivo .tar o de la entrada estándar.

Ejemplo:

docker load <archivo.tar

Opciones:

- **-i, --input**

  Lee de un fichero en lugar de la entrada estandar.

- **-q, --quiet**

  No muestra salida, se calla.

2.31. Tag

A veces trabajamos con varias versiones de la misma imagen. Para diferenciar estas diferentes versiones, recurrimos a las etiquetas. Así, **tag** comando añade una etiqueta a una versión de una imagen. La versión se puede especificar con Image:Versión pero si no se especifica, se utiliza la versión "latest" por defecto. Ejemplo:

docker tag hello-world version1

2.32. Rmi

Ahora vamos a ver un comando que a mí me gusta bastante, **rmi**, que elimina imágenes. Ejemplo:

docker rmi hello-world

Opciones:

- **-f, --force**

  Si una imagen tiene varias etiquetas, si no usamos esta opción, borramos solo la etiqueta seleccionada. Si por el contrario utilizamos esta opción Y el ID de la imagen, borramos todas las etiquetas también.

- **--no-prune** No elimina los padres que estén sin etiquetar.

2.33. Info

Info muestra información sobre el sistema y la instalación de Docker. Muestra información tal como la versión del kernel o el número de contenedores e imágenes (únicas). Ejemplo:

docker info

La única opción que proporciona este comando es *-f, --format* que permite proporcionarle una plantilla Go para que la salida se adapte a esta.

2.34. Version

Una opción muy parecida a la anterior es **version** que muestra la información sobre la versión del sistema Docker. Ejemplo:

docker version

Al igual que en la orden *info*, la única opción que tenemos aquí es *-f, --format* que también no permite formatear la salida con una plantilla Go.

3. Administración de memoria

Durante esta sección veremos como administrar los volúmenes y montajes de memoria de los contenedores.

3.1. Volumen

Ya hemos visto qué son los volúmenes, ahora vamos a ver cuáles son sus ventajas:

- Fáciles de almacenar y transportar.
- Fáciles de administrar.
- Muy seguros a la hora de compartirlos.
- Se pueden almacenar en remoto, en una nube, encriptar y añadir funcionalidades.
- Los nuevos pueden partir del contenido de un contenedor.

3.2. Montaje ligado

3.3. Montaje tmpfs

Referencias

[1] https://docs.docker.com/engine/docker-overview/#docker-engine

[2] https://docs.docker.com/get-started/

[3] https://docs.docker.com/get-started/part2/

[4] https://docs.docker.com/get-started/part3/

[5] https://docs.docker.com/get-started/part4/

[6] https://docs.docker.com/get-started/part5/

[7] https://docs.docker.com/storage/

[8] https://docs.docker.com/storage/volumes/

[9] https://docs.docker.com/storage/bind-mounts/

[10] https://docs.docker.com/storage/tmpfs/

[11] https://docs.docker.com/engine/reference/commandline/create/

[12] https://docs.docker.com/engine/reference/commandline/pull/

[13] https://docs.docker.com/engine/reference/commandline/start/

[14] https://docs.docker.com/engine/reference/commandline/run/

[15] https://docs.docker.com/engine/reference/commandline/ps/

[16] https://docs.docker.com/engine/reference/commandline/stop/

[17] https://docs.docker.com/engine/reference/commandline/rename/

[18] https://docs.docker.com/engine/reference/commandline/pause/

[19] https://docs.docker.com/engine/reference/commandline/unpause/

[20] https://docs.docker.com/engine/reference/commandline/kill/

[21] https://docs.docker.com/engine/reference/commandline/restart/

[22] https://docs.docker.com/engine/reference/commandline/update/

[23] https://docs.docker.com/engine/reference/commandline/attach/

[24] https://docs.docker.com/engine/reference/commandline/port/

[25] https://docs.docker.com/edge/engine/reference/commandline/cp/

[26] https://docs.docker.com/edge/engine/reference/commandline/rm/

[27] https://docs.docker.com/edge/engine/reference/commandline/exec/

[28] https://docs.docker.com/engine/reference/commandline/export/

[29] https://docs.docker.com/engine/reference/commandline/wait/

[30] https://docs.docker.com/engine/reference/commandline/events/

[31] https://docs.docker.com/engine/reference/commandline/stats/

[32] https://docs.docker.com/engine/reference/commandline/top/

[33] https://docs.docker.com/engine/reference/commandline/inspect/

[34] https://docs.docker.com/edge/engine/reference/commandline/login/

[35] https://docs.docker.com/engine/reference/commandline/logout/

[36] https://docs.docker.com/engine/reference/commandline/images/

[37] https://docs.docker.com/engine/reference/commandline/search/

[38] https://docs.docker.com/engine/reference/commandline/history/

[39] https://docs.docker.com/edge/engine/reference/commandline/save/

[40] https://docs.docker.com/engine/reference/commandline/load/

[41] https://docs.docker.com/engine/reference/commandline/tag/

[42] https://medium.freecodecamp.org/an-introduction-to-docker-tags-9b5395636c2a

[43] https://docs.docker.com/engine/reference/commandline/rmi/

[44] https://docs.docker.com/engine/reference/commandline/info/

[45] https://docs.docker.com/engine/reference/commandline/version/