

Almacenamiento

Antonio Espín Herranz

Contenidos

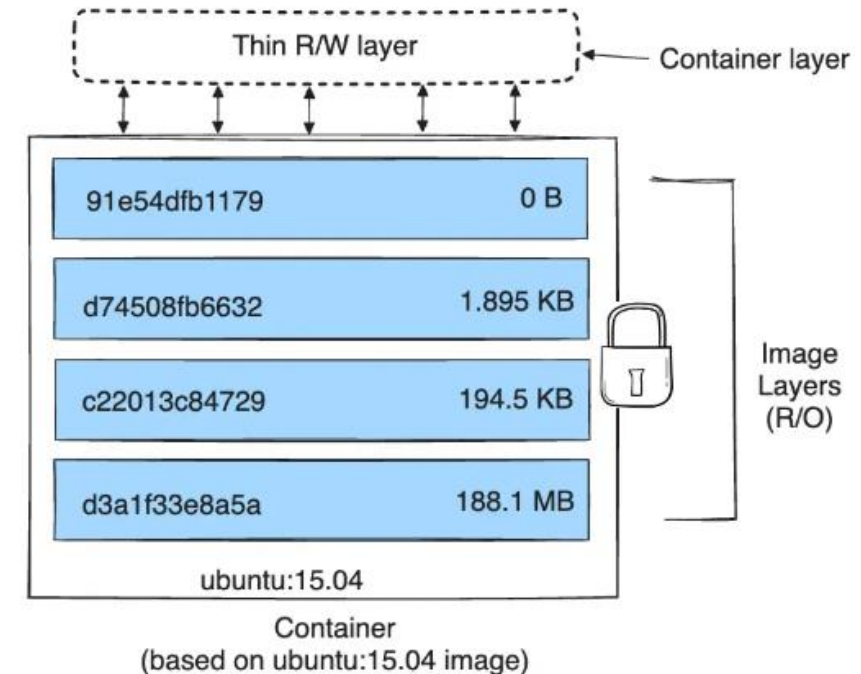
- Introducción
- AUFS
- OverlayFS / Overlay2
- Device Mapper
- BRTFS
- Volúmenes
- Plugin para Volúmenes
- Persistencia de Volúmenes

Introducción

- Por la naturaleza de los contenedores, cualquier dato que se escriba en su sistema de ficheros será eliminado al destruirse los mismos.
- Un contenedor añade una capa de lectura/escritura a la imagen que usa de base, pero que la imagen en sí no puede ser alterada por ninguno de sus contenedores.
- Para aquellos casos en los que necesitemos almacenamiento persistente en sistema de ficheros existen los volúmenes de datos Docker

Copy on Write

- Docker cuenta con un mecanismo llamado **copy-on-write** (CoW), permite crear contenedores instantáneamente.
- No se realiza una copia completa del sistema de archivo en base a la imagen.
- El sistema de almacenamiento lleva el control de lo que ha cambiado respecto al sistema de archivos base.



Copy on write

- Este sistema se puede implementar de varias maneras:
 - **AUFS** (overlay filesystem)
 - El framework mapeador de dispositivos **Device Mapper**, que implementa el método de optimización de eficiencia thin provisioning (capas de almacenamiento a nivel de dispositivo de bloque)
 - El sistema de archivos **BTRFS**
 - El sistema de archivos y administrador de volúmenes **ZFS**

Copy on Write

- Con el mecanismo «**copy-on-write**», podemos tener multitud de contenedores sin ocupar ningún espacio adicional, mientras no **escribamos** cambios dentro del contenedor.
- Permite también hacer cosas como usar el comando **docker commit** para generar una nueva imagen base con los cambios realizados en un determinado contenedor.

Storage Driver

- Ejecutando el comando: **docker info**
- Podemos ver la implementación que está utilizando Docker para la implementación de copy on write.

<https://docs.docker.com/engine/storage/drivers/select-storage-driver/>

```
Server:
 Containers: 23
  Running: 2
  Paused: 0
  Stopped: 21
 Images: 6
 Server Version: 27.5.1
 Storage Driver: overlay2
```

La selección del storage driver viene normalmente limitada por la disponibilidad de ciertas características, como el soporte de sistemas de archivos, de la distribución y el kernel sobre el que se ejecute Docker

Se puede cambiar el driver actuando sobre los Parámetros de inicio de Docker → **dockerd**

AUFS

- AUFS (Advanced Multi-Layered Unification Filesystem) es un sistema de archivos en Docker que permite gestionar múltiples capas en las imágenes de contenedores.
 - Esta tecnología facilita las operaciones del sistema de archivos al fusionar directorios, mejorando la eficiencia y la flexibilidad.
- AUFS se utilizaba como controlador de almacenamiento por defecto en Docker para Ubuntu y versiones de Debian hoy en día se recomienda utilizar el controlador overlay2, que ofrece ventajas de rendimiento.

OverlayFS / Overlay2

- El controlador overlay2 es el **controlador de almacenamiento predeterminado para Docker** en las versiones más recientes del núcleo Linux.
 - Este controlador utiliza el sistema de archivos OverlayFS, que es más eficiente y ofrece mejor rendimiento en comparación con otros controladores, como AUFS.
- Ventajas de utilizar overlay2:
 - Eficiencia en el almacenamiento: reduce el uso de espacio en disco
 - Mejor rendimiento: que AUFS
 - Mayor soporte y estabilidad: recibe soporte continuo

Device Mapper

- Características:
 - Maneja los dispositivos en bloques
 - Facilita las operaciones de lectura y escritura
 - Facilita la eficiencia en almacenamiento
 - Permite crear SnapShot → crear copias de seguridad y restaurar el estado de los contenedores.
- Desventajas:
 - La configuración es compleja.
 - Tiene un rendimiento inferior al overlay2

BRTFS

- B-Tree File System. Sistema de archivos avanzado
 - Permite snapshot
 - Compresión
 - Duplicación
-
- Pero ocurre lo mismo en cuanto a los anteriores.
 - **El driver Overlay2 tiene mejor rendimiento**

Tipos de almacenamiento

- **Volúmenes:**

- La mejor opción para persistir información. Los controla Docker.
 - En Linux: se almacenan en un determinado directorio que es administrado por Docker (/var/lib/docker/volumes)
 - En Windows: un poco más complicado de localizar.

- **Bind mounts** (almacenamiento compartido entre el contenedor y el host).

- Tienen la desventaja de crear acoplamiento entre el contenedor y el host y a la hora de trasladarlo no está tan aislado.
- Se pueden colocar en cualquier directorio, pero no los gestiona Docker.

- **tmpfs mounts (Linux) / named Pipes (Windows)**

- Almacenamiento en memoria. No escribe en el sistema de archivos.

Volúmenes I

- Un volumen de datos es un directorio especial del **host** que puede ser **compartido** entre varios **contenedores**
- Aporta las siguientes ventajas:
 - **Se inicializan en la creación del contenedor:** Si la imagen base tiene información en el punto de montaje, esos datos se copian en el volumen creado durante la inicialización.
 - **Pueden compartirse y reutilizarse** entre contenedores
 - Los cambios en los datos del volumen **se hacen directamente**
 - **Se gestionan completamente por Docker.**

Volúmenes II

- Si se actualiza una imagen no se actualizan los datos del volumen
- Y más importante, **si una imagen se borra sus volúmenes no se borran.**
 - **Los volúmenes están pensados para persistir** datos independientemente del ciclo de vida de los contenedores que los usen.
- Funcionan tanto en **Windows** como **Linux**.
- Se pueden almacenar en **host remotos** o en **entornos cloud**, cifrar el contenido.
- Los contenidos de un nuevo volumen **pueden ser rellenados de forma previa** por un contenedor.

Volúmenes III



- Se separa del contenedor.
- Con los volúmenes podemos:
 - Usar un almacenamiento persistente para el contenedor
 - Compartir almacenamiento entre el HOST y un contenedor
 - Compartir almacenamiento entre distintos contenedores

Crear contenedores con Volúmenes

- `docker run -it -v /datos --name ubuntu1 ubuntu bash`
- Crear un directorio /datos y al ponerle bash entramos dentro del contenedor.

```
root@c60d11facda3:/# ls -l
total 52
lrwxrwxrwx    1 root root      7 Apr 22  2024 bin -> usr/bin
drwxr-xr-x    2 root root 4096 Apr 22  2024 boot
drwxr-xr-x    2 root root 4096 Mar 15 17:37 datos
drwxr-xr-x    5 root root  360 Mar 15 17:37 dev
drwxr-xr-x    1 root root 4096 Mar 15 17:37 etc
```


Inspeccionar el volumen

- Con el comando: **docker volumen ls**
 - Podemos listar los volúmenes.
- Y a través del hash que le asigna docker podemos obtener información de ese volumen: **docker volume inspect <número hash>**
 - Dentro del volumen crea una carpeta vacía `_data`

```
[
  {
    "CreatedAt": "2025-03-06T17:34:32Z",
    "Driver": "local",
    "Labels": {
      "com.docker.volume.anonymous": ""
    },
    "Mountpoint": "/var/lib/docker/volumes/d591be4a4b4d77129aafd772c7138416f08b156de3b17baf0cb8cea387f862c0/_data",
    "Name": "d591be4a4b4d77129aafd772c7138416f08b156de3b17baf0cb8cea387f862c0",
    "Options": null,
    "Scope": "local"
  }
]
```

Crear ficheros dentro del directorio del volumen

- Si no estuviera arrancado el contenedor y ya hemos salido de la Shell tenemos que conectarnos de nuevo al contenedor, lo podemos hacer:
 - **`docker start -i nombre_cont`**
- Podemos **entrar** en el **directorio** del **volumen** y crear ficheros.
 - **`touch fichero1`**
- Al crearlo dentro del contenedor automáticamente se crea en el volumen que habíamos creado.
 - **Y si lo creamos en el directorio correspondiente del host donde se almacenan los volúmenes pasaría lo mismo.**
 - En los sistemas **Linux** se almacenan los volúmenes en:
`/var/lib/docker/volumes/<<número_hash>>`
 - En los sistemas **Windows + WSL2** se almacenan en:
 - Lanzar **`wsl`** en una consola

`\\wsl.localhost\docker-desktop\mnt\docker-desktop-disk\data\Docker\volume`
(al final se comenta, se teclea en un explorador de Windows)

Crear ficheros dentro del directorio del volumen II

- Cuando salimos del contenedor o lo paramos con stop el volumen continúa existiendo.
- El volumen es **persistente**.
- Los volúmenes se pueden gestionar con el comando:
 - **docker volume**
 - De la misma forma que los contenedores se les puede asignar un nombre.

Volúmenes - comandos

- Opciones
 - **create** crear un volumen
 - **inspect** visualizar detalles de un volumen
 - **ls** lista volúmenes
 - **prune** Borra volúmenes que no se utilizan
 - **rm** Para borrar un volumen

Crear volúmenes con nombre

- **docker volume create <<nombre>>**
- A partir de este comando, siempre nos referimos al volumen a partir de ese nombre.
 - **docker volume inspect <<nombre>>**
- Si hemos creado un volumen, este, se puede asociar a un contenedor:
 - **docker volume create vol1**
 - **docker volume ls**
 - **docker run -it --name ubuntu7 -v vol1:/dir_contenedor ubuntu bash**

Compartir volúmenes

- Los volúmenes se pueden compartir con otros contenedores.
- `docker run -it --name ubuntu8 -v vol1:/datos ubuntu bash`
- También se puede compartir en modo solo lectura (añadiendo `:ro`)
- `docker run -it --name ubuntu8 -v vol1:/datos:ro ubuntu bash`
- Creando ficheros desde uno de los contenedores, podemos verlos desde el otro contenedor.
 - Si está parado, arrancarlo → `docker start -i <<nombre_cont>>`

Compartir múltiples volúmenes

- Al crear un nuevo contenedor le podemos indicar que comparta los volúmenes de otro contenedor.
 - `docker run -it --name ubuntu10 --volumes-from ubuntu9 ubuntu bash`
- Automáticamente nos tiene que añadir un directorio que se llamará igual que en el contenedor origen.
- **Varios contenedores pueden apuntar al mismo volumen.**

Volúmenes huérfanos

- Cuando tenemos varios contenedores que apuntan al mismo volumen y se borra uno de ellos no ocurre nada (el volumen no se borra).
- Si eliminamos todos los contenedores que apuntan a un volumen,
 - El contenido del volumen permanece en el directorio y **el volumen se puede asignar a otro contenedor**.
 - Docker cuando ponemos la opción `-v` comprueba si ya existe o no el volumen, si no existe lo crea.
- Podemos borrar **rm** el volumen o hacer un **prune (borra todos los volúmenes que están sin utilizar, sólo los anónimos)**.

Borrar volúmenes

- **docker volume rm nombre_hash**
- Si el volumen está ligado a un contenedor no nos dejará borrarlo.
- Nos dará un error, e indicará los hash de los contenedores que tienen asociado ese volumen.
- **docker volume prune**
 - Emite un mensaje con los megas que se han liberado.

Bind mounts

- Se utiliza para compartir información entre el contenedor y el host.
- Al crear el contenedor podemos indicar un directorio del host y un directorio del contenedor.
- Ejemplo:
 - **`docker run -it -v directorio_host:directorio_cont --name ubuntu10 ubuntu bash`**
 - Utilizar rutas absolutas
 - Si alguno de los directorios no existe lo crea siempre y cuando tengamos permisos.
 - Si vamos creando ficheros en el directorio del contenedor, automáticamente los podemos ver en el directorio del host y al revés también → funciona en ambos sentidos.

Bind mounts

- Si utilizamos esta posibilidad de compartir el directorio del host con el directorio del contenedor, no la muestra en los volúmenes.
 - El comando: **docker volume ls**
 - No muestra nada de ese directorio
- Lo veremos al hacer un inspect del contenedor.
- Con este comando si lo podríamos ver en la información que muestra del contenedor:
 - **docker inspect ubuntu10 > ubuntu10.txt**
 - No aparece como un volumen, si no, como un montaje

```
"HostConfig": {  
  "Binds": [  
    "/home/antonio/directorio1:/dir1"  
  ],  
}
```

Almacenamiento temporal: TMPFS

- A diferencia de los volúmenes y los bind mounts, un montaje tmpfs es temporal y solo persiste en la memoria del host.
- Cuando el contenedor se detiene, el montaje tmpfs se elimina y los archivos escritos allí no se persisten.
- Los montajes tmpfs son ideales cuando no se desea que los datos persistan ni en el host ni dentro del contenedor.
 - Esto puede deberse a **razones de seguridad** o para proteger el **rendimiento** del contenedor cuando la aplicación necesita escribir un gran volumen de datos de estado no persistentes.

Almacenamiento temporal: TMPFS

- **docker run -d -i --name ubuntu15 --tmpfs /datos Ubuntu**
- **docker exec -it ubuntu15 bash**
- Crear fichero y salir del contenedor, comprobar si se mantiene o no, ese fichero.
 - **docker stop ubuntu15**
 - **docker start ubuntu16**
 - **docker exec -it ubuntu16 bash**
 - El contenido que se hubiera creado en /datos ya no existe porque estaba configurado como almacenamiento temporal.

Bind mounts en Windows

- Cuando hacemos un **bind mounts** desde Windows tenemos que tener en cuenta las **rutas** en **Windows**, **por lo demás, los comandos son iguales**.
- Probar fuera de **WSL**:
 - **docker run --name ubuntu21 -d -it -v c:\tmp\vol1:/datos Ubuntu**
 - **docker exec -it ubuntu21 bash**
 - Si creamos ficheros dentro de la carpeta /datos del contenedor, automáticamente aparecen en la carpeta de Windows.
 - Funciona en ambos sentidos, igual que con WSL.

Volúmenes en Windows

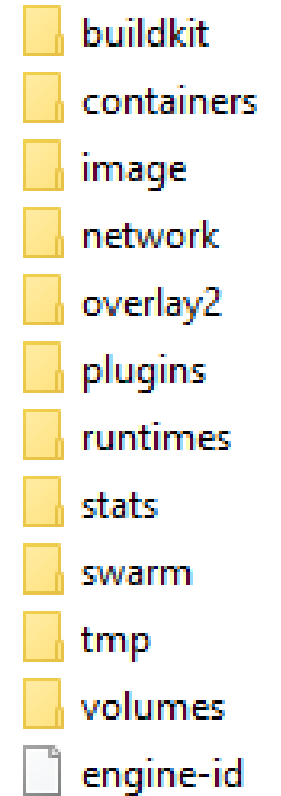
- `docker run --name ubuntu22 -it -d -v vol1:/datos ubuntu`
- Si hacemos: `docker inspect ubuntu22`
- Le vemos también con el comando: `docker volume ls`

```
"Mounts": [  
  {  
    "Type": "volume",  
    "Name": "vol1",  
    "Source": "/var/lib/docker/volumes/vol1/_data",  
    "Destination": "/datos",  
    "Driver": "local",  
    "Mode": "z",  
    "RW": true,  
    "Propagation": ""  
  }  
]
```

Si no fuera de tipo volumen
En el **inspect** veríamos
Type: bind
Igual que en Linux

Volúmenes en Windows

- ¿Dónde los guarda en Windows?
- Lo hace a través de la carpeta (teclear en un explorador)
 - **\\wsl.localhost\docker-desktop\mnt\docker-desktop-disk\data\docker**



Plugin para Volúmenes

- Características generales:
 - Persistencia de datos
 - Integración con almacenamiento externo
 - Fuera del host de Docker
- Fácil configuración
- Compatibilidad

Plugin para Volúmenes

- **Plugin gratuitos:**
 - **Rclone:** permite montar sistemas de archivos remotos en contenedores
 - Compatible con almacenamiento en la nube: Google Drive, Amazon S3 ...
 - **NFS** (Network File System)
 - Montar archivos NFS en contenedores Docker
 - **CIFS** (Common Internet File System)
 - Para montar sistemas de archivos CIFS en contenedores Docker
 - **GlusterFS:**
 - Sistema de archivos distribuido que permite montar contenedores de Docker
 - Se necesita alta disponibilidad y escalabilidad

Plugin para Volúmenes

- Instalar un plugin de volúmenes:
 - Docker tiene soporte para varios plugins, como **Flocker**, **Rex-Ray** o plugins específicos de proveedores de almacenamiento.
 - Instala el plugin **rexray/s3fs** para usar almacenamiento en **Amazon S3**
 - `docker plugin install rexray/s3fs`
- Cuando ya se ha instalado se puede crear un volumen utilizando el plugin como driver
 - `docker volume create --driver rexray/s3fs my_volume`

Plugin para Volúmenes

- Los plugins de volúmenes permiten integrar los volúmenes de los contenedores con sistemas de almacenamiento externos, como **Amazon EBS (Elastic Block Store)**
 - Es una buena alternativa a que se almacenen en el host de Docker
- https://docs.docker.com/engine/extend/plugins_volume/