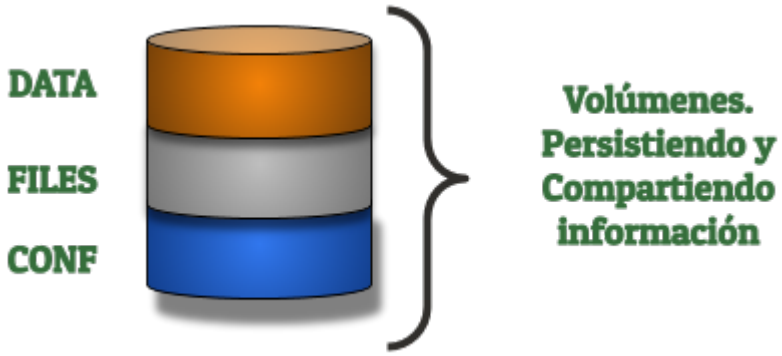
4. Persistencia en docker

En este módulo del curso vamos a adentrarnos en la persistencia de los datos de los contenedores docker. Trataremos lo siguientes aspectos:

- **Necesidad** de persistir los datos de los contenedores.
- **Formas de gestionar** esa persistencia (Volúmenes y Bind Mounts).
- **Operaciones** para la **gestión** de volúmenes y para la **obtención de información** de los mismos.
- Cómo **asociar volúmenes o bind mounts a nuestros contenedores**.
- Uso de la persistencia de los datos como **copia de seguridad.**
- **Compartición de datos** entre distintos contenedores.
- **Depuración de aplicaciones usando bind mounts.**



Juan Diego Pérez Jiménez. *Volúmenes Docker* (Dominio público)

4.1 Los datos en los contenedores

¿Qué pasa con los datos de los contenedores?

Hasta ahora no habíamos hablado de ello pero estoy seguro de que más de uno ya se lo había preguntado y ya se había dado cuenta.

Los **ficheros, datos y configuraciones** que creemos en los contenedores **sobreviven a las paradas** de los mismos pero, sin embargo, **son destruidos si el contenedor es destruido**. Y esto, como todos entendemos es una situación no deseable ya que puede echar por tierra nuestro trabajo.

Por lo tanto tenemos que tener muy presente varios aspectos a la hora de afrontar esta situación y la gestión del almacenamiento de los contenedores:

- Los **datos de un contenedor mueren con él.**
- Los datos de los contenedores **no se mueven fácilmente** ya que están fuertemente acoplados con el host en el que el contenedor está ejecutándose.
- **Escribir** en los contenedores **es más lento que** escribir en el **host** ya que tenemos una capa adicional.

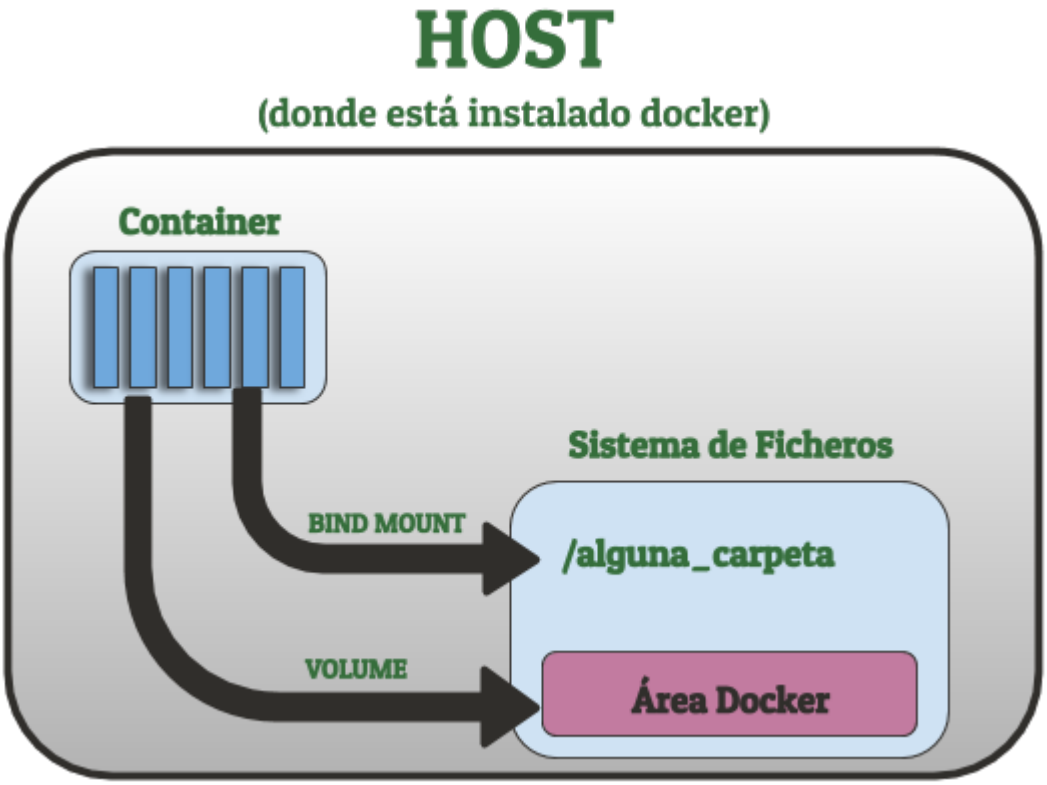
Ante la situación anteriormente descrita Docker nos proporciona **VARIAS SOLUCIONES PARA PERSISTIR DATOS** de contenedores. En este curso **nos vamos a centrar en** las dos que considero que son más importante y útiles para nuestro día a día docente.

- Los **VOLÚMENES** docker.
- Los **BIND MOUNT.**

Existen otras formas como los **tmpfs mounts** para Linux y los **named pipes** para Windows pero para este curso nos quedaremos con los dos ya citados.

4.2 Volúmenes y Bind Mount

Tal y como dijimos en el apartado anterior, de las soluciones de persistencia que nos proporciona docker nos vamos a quedar con dos para este curso, los ***volúmenes*** y los ***bind mounts***. Antes de que hablemos de las características y ventajas de cada uno de ellas las vamos situar dentro de nuestro host con el siguiente esquema general:



Juan Diego Pérez Jiménez. *Persistencia en contenedores* (Dominio público)

VOLÚMENES DOCKER

Si elegimos conseguir la persistencia usando **volúmenes** estamos haciendo que **los datos de los contenedores** que nosotros decidamos se almacenen en una **parte del sistema de ficheros** que es **gestionada** por **docker** y a la que, debido a sus permisos, sólo docker tendrá acceso.

Esa **"ZONA RESERVADA"** de docker cambia de un sistema operativo a otro y también puede cambiar dependiendo de la forma de instalación, pero de manera general podemos decir que es:

- ***/var/lib/docker/volumes*** en las distribuciones de Linux si lo hemos instalado desde paquetes estándar.
- **/var/snap/docker/common/var-lib-docker/volumes** en Linux si hemos instalado docker mediante snap (no lo recomiendo).
- ***C:\ProgramData\docker\volumes*** en las instalaciones de Windows.
- ***/var/lib/docker/volumes*** también en Mac aunque se requiere que haya una conexión previa a la máquina virtual que se crea.

Este tipo de volúmenes se suele usar en los siguiente casos:

- Para **compartir datos entre contenedores**. Simplemente tendrán que usar el mismo volumen.
- Para **copias de seguridad** ya sea para que sean usadas posteriormente por otros contenedores o para mover esos volúmenes a otros hosts.
- Cuando quiero **almacenar los datos** de mi contenedor no localmente si no **en un proveedor cloud.**
- En algunas situaciones donde usando Docker Desktop quiero más rendimiento. Esto se escapa al ámbito de este curso.

Bind Mounts

Si elegimos conseguir la persistencia de los datos de los contenedores usando **bind mount** lo que estamos haciendo es ***"mapear"*** una parte de mi sistema de ficheros, de la que yo normalmente tengo el control, con una parte del sistema de ficheros del contenedor.

Este mapeado de partes de mi sistema de ficheros con el sistema de ficheros del contenedor me va a permitir:

- **Compartir ficheros** entre el host y los containers.
- Que otras aplicaciones que no sean docker tengan acceso a esos ficheros, ya sean código, ficheros etc...

Puede parecer que el hecho de que otras aplicaciones accedan a esos datos es algo negativo pero precisamente los **bind mounts** es el mecanismo que vamos a **preferir** para la fase de **DESARROLLO** ya que:

- Las **aplicaciones** que podrán acceder a esos ficheros serán los **IDEs o editores de código.**
- Estaremos modificando con aplicaciones locales **código** que a la vez se encuentra **en nuestro equipo y en el contenedor.**
- Y desde mi propio equipo estaré probando ese **código en el entorno elegido, o en varios entornos** a la vez **sin necesidad de tener que instalar absolutamente nada** en mi sistema.

4.3 Gestionando volúmenes y obteniendo información

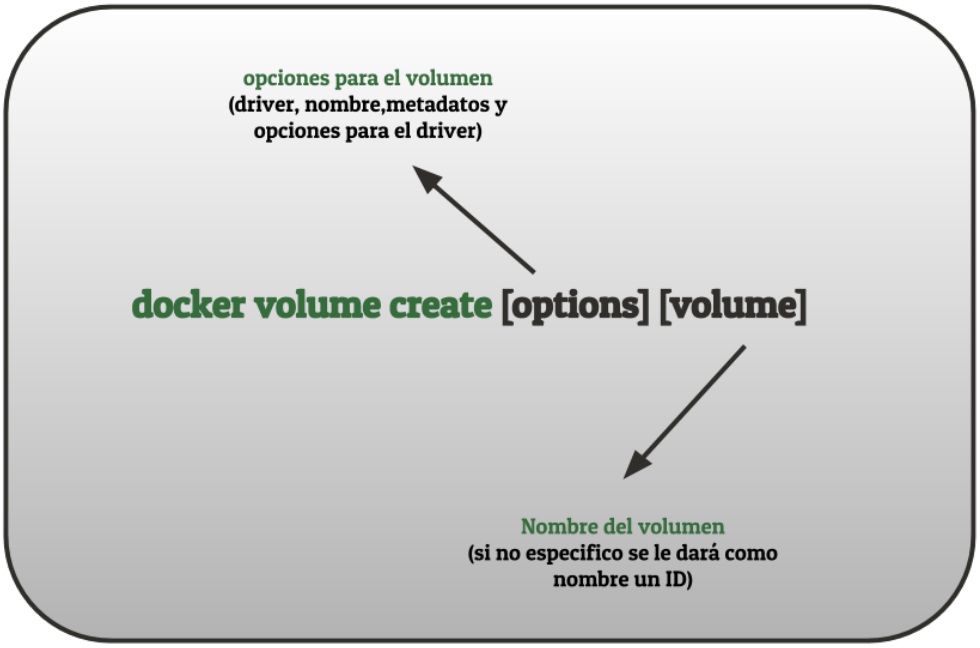
En el apartado anterior presentamos las dos opciones para la persistencia de datos con docker que consideramos que eran de mayor interés para el desarrollo de software: volúmenes y bind mounts. En este apartado nos centraremos en los volúmenes y en las operaciones básicas que podemos hacer con ellos mediante la orden docker volume. Estas operaciones son:

- **Creación** de los volúmenes.
- **Eliminación** de los volúmenes.
- **Obtención de información** de los volúmenes.

En el apartado siguiente veremos como, una vez hayamos definido estos volúmenes, podremos usarlos en nuestros contenedores.

CREACIÓN DE VOLÚMENES

Para la creación de volúmenes vamos a usar la orden **docker volume create** que tiene la siguiente estructura:



Juan Diego Pérez Jiménez. *Creación de volúmenes. Estructura de la orden docker volume create* (Dominio público)

Entre las opciones que podemos incluir a la hora de crear los volúmenes están:

- **--driver o -d** para especificar el driver elegido para el volumen. Si no especificamos nada el driver utilizado es el ***local*** que es el que nos interesa desde el punto de vista de desarrollo porque desarrollamos en nuestra máquina. Al ser Linux en mi caso ese driver local es **overlay2** pero existen otras posibilidades como ***aufs, btrfs, zfs, devicemapper o vfs.*** Si estamos interesados en conocer al detalle cada uno de ellos aquí tenemos más información.
- **--label** para especificar los metadatos del volumen mediante parejas clave-valor.
- **--opt o -o** para especificar opciones relativas al driver elegido. Si son opciones relativas al sistema de ficheros puedo usar una sintaxis similar a las opciones de la orden mount.

- **--name** para especificar un nombre para el volumen. Es una alternativa a especificarlo al final que es la forma que está descrita en la imagen superior.

Vamos a ilustrar este funcionamiento con varios ejemplos:

```
# Creación de un volumen llamado datos (driver local sin opciones)

> docker volume create data

# Creación de un volumen data especificando el driver local

> docker volume create -d local data

# Creación de un volumen llamando web añadiendo varios metadatos

> docker volume create --label servicio=http --label server=apache Web
```

Aunque queda fuera del alcance de los objetivos de este curso es importante destacar que mediante otras opciones podríamos indicar que los volúmenes estén en otros dispositivos, en montajes nfs e incluso en servicios de almacenamiento en nube públicas o privadas.

ELIMINACIÓN DE VOLÚMENES

Para la eliminación de los volúmenes creados tenemos dos opciones:

- **docker volume rm** para eliminar un volumen en concreto (por nombre o por id).
- **docker volumen prune** para eliminar los volúmenes que no están siendo usados por ningún contenedor.

A continuación vamos a ver una lista de ejemplos para ilustrar el funcionamiento de ambos:

```
# Borrar un volumen por nombre

> docker volume rm nombre_volumen

# Borrar un volumen por ID

> docker volume rm
a5175dc955cfcf7f118f72dd37291592a69915f82a49f62f83666ddc81f67441

# Borrar dos volúmenes de una sola vez

> docker volume rm nombre_volumen1 nombre_volumen2

# Forzar el borrado de un volumen -f o --force

> docker volume rm -f nombre_volumen
```

```
# Borrar todos los volúmenes que no tengan contenedores  asociados

> docker volume prune

# Borrar todos los volúmenes que no tengan contenedores  asociados sin pedir confirmación (-f o --force)

> docker volume prune -f

# Borrar todos los volúmenes sin usar que contengan cierto valor de etiqueta (--filter)

> docker volume prune --filter label=valor
```

NOTA: NO SE PUEDEN ELIMINAR VOLÚMENES EN USO POR CONTENEDORES, salvo que usemos el flag -f o --force y no es algo recomendado.

OBTENCIÓN DE INFORMACIÓN DE LOS VOLÚMENES

Si queremos obtener información de los volúmenes que hemos creado podemos hacerlo de dos formas:

- Usando **docker volume ls** que nos proporciona una lista de los volúmenes creados y algo de información adicional.
- Usando **docker volume inspect** que nos dará una información mucho más detallada de el volumen que hayamos elegido.

LISTA DE VOLÚMENES DEL SISTEMA

Si ejecutamos la siguiente orden:

```
# Listar los volúmenes creados en el sistema

> docker volume ls
```

Obtendremos una salida similar a la siguiente:

```
pekechis ~ docker volume ls
DRIVER              VOLUME NAME
local               a5175dc955cfcf7f118f72dd37291592a69915f82a49f62f83666ddc81f67441
local               jenkins_home
```

Juan Diego Pérez Jiménez. *Obteniendo la lista de volúemes .docker volume ls* (Dominio público)

Como podemos ver, la información que nos proporciona es limitada, el driver usado y el nombre que tenga cada volumen de la lista. Si no tiene nombre se nos muestra el ID del volumen.

INFORMACIÓN DETALLADA DE UN VOLUMEN

Si queremos información más detallada de un volumen tenemos que ejecutar la siguiente orden:

```
# Información detallada de un volumen por nombre

> docker volume inspect nombre_volumen

# Información detallada de un volumen por ID

> docker volume inspect
a5175dc955cfcf7f118f72dd37291592a69915f82a49f62f83666ddc81f67441
```

Obtendremos una salida similar a la siguiente:

Juan Diego Pérez Jiménez. *Información detallada de un volumen. docker inspect* (Dominio público)

Y la información que nos muestra es:

- La fecha de creación del volúmen.
- El tipo del driver.
- Etiquetas asociadas.
- El punto de montaje.
- El nombre del volumen.
- Las opciones asociadas al driver.
- Y el ámbito del volumen.

4.4 Asociando almacenamiento a los contenedores

Una vez hemos visto en el apartado anterior cómo crear los volúmenes vamos a ver en este apartado como puedo usar los volúmenes y los bind mounts en los contenedores. Para cualquiera de los dos casos lo haremos mediante el uso de **dos flags** de la orden **docker run:**

- El flag **--volume o -v.** Este flag lo utilizaremos para establecer bind mounts.
- El flag **--mount.** Este flag nos servirá para establecer bind mounts y para usar volúmenes previamente definidos (entre otras cosas).

Es importante que tengamos en cuenta dos cosas importantes a la hora de realizar estas operaciones:

- Al usar tanto volúmenes como bind mount el contenido de lo que tenemos **sobreescribirá la carpeta destino en el sistema de ficheros del contenedor** en caso de que exista. Y si

nuestra carpeta origen no existe y hacemos un bind mount esa carpeta se creará pero lo que tendremos en el contenedor es una carpeta vacía. Con esto hay que tener especial cuidado, sobre todo cuando estamos trabajando con carpetas que pueden contener datos y configuraciones varias.

- Si usamos imágenes de DockerHub, debemos **leer la información que cada imagen nos proporciona en su página** ya que esa información suele indicar cómo persistir los datos de esa imagen, ya sea con volúmenes o bind mounts, y cuáles son las **carpetas importantes** en caso de ser imágenes que contengan ciertos servicios (web, base de datos etc...)

Como acostumbramos en este curso vamos a ilustrar todo esto mediante una serie de ejemplos a los que añadiremos varios opciones

```
# BIND MOUNT (flag -v): La carpeta web del usuario será el directorio raíz del servidor apache. Se crea si no existe

> docker run --name apache -v /home/usuario/web:/usr/local/apache2/htdocs -p 80:80 httpd

# BIND MOUNT (flag --mount): La carpeta web del usuario será el directorio raíz del servidor apache. Se crea si no existe

> docker run --name apache -p 80:80 --mount type=bind,src=/home/usuario/web,dst=/usr/local/apache2/htdocs httpd

# VOLUME (flag --mount). Mapear el volumen previamente creado y que se llama Data en la carpeta raíz del servidor apache

> docker run --name apache -p 80:80 --mount type=volume,src=Data,dst=/usr/local/apache2/htdocs httpd

# VOLUME (flag --mount). Igual que el anterior pero al no poner nombre de volumen se crea uno automáticamente (con un ID como nombre)

> docker run --name apache -p 80:80 --mount type=volume,dst=/usr/local/apache2/htdocs httpd
```

En cualquiera de los dos casos, cuando creamos un bind mount o asociamos un volumen a un contenedor, esto queda reflejado en la salida de la orden **docker inspect sobre dicho contenedor**, de una manera similar a la imagen inferior.

Juan Ddiego Pérez Jiménez. *Volumen y Bind Mount en Uso* (Dominio público)

4.5 Herramientas Docker en Visual Studio Code

Hasta ahora durante todo el curso he usado **el terminal para interactuar con docker.** Es la herramienta a la que estoy acostumbrado y en el sistema operativo en el que suelo trabajar. Los **usuarios de Docker Desktop** para Windows y Mac ya habréis podido comprobar por vosotros mismos que desde el panel de control **se pueden hacer de manera visual operaciones** como la gestión de imágenes y la gestión de los contenedores.

Tras casi 4 capítulos completos poco a poco nos vamos aproximando a la parte del curso que más tiene que ver con la coletilla "para el desarrollo" que acompaña al título del mismo. Como desarrolladores usamos varios IDEs o editores de código y muchos de ellos tienen extensiones que me van a permitir trabajar e interactuar con docker de manera visual desde el propio editor, de igual manera, y en algunos casos con muchas más funcionalidades, a cómo lo haríamos desde la aplicación de escritorio Docker Desktop.

Existen muchos editores pero yo llevo ya varios años usando Visual Studio Code para casi todo salvo para el desarrollo Java, así que en el siguiente vídeo os voy a mostrar qué extensiones son las recomendables y cómo usarlas. Estas extensiones son desarrolladas por Microsoft que es la misma empresa que desarrolla el Visual Studio Code y son las siguientes:

- Remote - Containers
- Docker

Vamos a ver cómo funciona de manera rápida en el siguiente vídeo:

https://youtu.be/1dTShnsjsFE?list=PL-8CyWabyNa85xowmOeBMCspbrn6qNWgl

OTROS IDEs

Para otros IDEs como Eclipse o IntelliJ os pongo los enlaces de los plugins para las extensiones docker:

- Plugin Docker para IntelliJ
- Plugin Docker para Eclipse