

Conversación con Gemini

Exported: 15/6/2026 9:11:34

Jordi H.

Has dicho explicame los volúmenes de docker y sobre todo la sintaxis en la docker file. no comprendo lo que he leído en la documentación

Gemini

Entender los volúmenes de Docker puede ser frustrante porque la documentación oficial a veces mezcla conceptos de administración de sistemas con desarrollo. En esencia, un **volumen** es un "túnel" que conecta una carpeta de tu computadora física (host) con una carpeta dentro del contenedor. Como los contenedores son efímeros (si los borras, los datos mueren con ellos), los volúmenes son la forma de hacer que los datos sean **persisten**

1. Tipos de montajes (Volumes vs Bind Mounts)

Aunque solemos decir "volúmenes" para todo, hay dos formas principales de persistir datos:

- **Named Volumes (Volúmenes con nombre):** Docker gestiona dónde se guardan físicamente en el disco. Son ideales para bases de datos
- **Bind Mounts:** Tú le dices exactamente qué carpeta de tu PC quieres mapear. Es lo que usarás constantemente en análisis de datos con Python para ver tus scripts reflejados en tiempo real

2. La sintaxis en el Dockerfile (`VOLUME`)

Aquí es donde suele estar la confusión. En un Dockerfile, la instrucción **VOLUME** tiene una función muy específica y limitada. **Sintaxis:** Dockerfile

CODE

```
VOLUME ["/data"]
```

Lo que SÍ hace:

- Crea un punto de montaje en la imagen.
- Le dice a Docker: "Cualquier cosa que se escriba en /data debe vivir fuera de la capa de unión del contenedor"
- Si no especificas un origen al arrancar el contenedor, Docker creará un volumen anónimo automáticamente.

Lo que NO hace (y por eso te confunde):

- No puedes mapear una carpeta de tu PC desde el Dockerfile. Por seguridad, el Dockerfile no tiene acceso a las rutas de tu computadora (como C:/Usuarios/Proyectos)
- No sirve para desarrollo en vivo. Si usas **VOLUME** en el Dockerfile, no verás tus cambios de código automáticamente

3. La sintaxis "Real": El comando `docker run`

Para conectar tus scripts de Python con el contenedor, la magia ocurre al **ejecutar** el contenedor, no al construirlo. Se usa el flag -v (o --volume)

Estructura:

`docker run -v [RUTA_PC]:[RUTA_CONTENEDOR]:[MODO] [IMAGEN]`

Parte	Descripción	Ejemplo
RUTA_PC	Carpeta absoluta en tu computadora.	/home/usuario/proyecto_python
Separador	Dos puntos.	:
RUTA_CONTENEDOR	Dónde quieres que aparezca dentro	/app/data
MODO (Opcional)	Permisos (ro = solo lectura, rw =	:ro

Ejemplo práctico para análisis de datos:

Si quieres correr un script de Python y que guarde un CSV en tu PC: Bash

CODE

```
docker run -v $(pwd):/app python:3.11 python /app/script.py
```

(Aquí \$(pwd) toma tu carpeta actual y la "pega" dentro de /app en el contenedor).

4. Resumen: ¿Cuándo usar cada cual?

1. En el Dockerfile: Usa VOLUME solo para marcar carpetas que deben ser persistentes por diseño (como /var/lib/mysql), pero no para desarrollo diario
2. En Docker Compose o CLI: Usa montajes (-v) para conectar tu código Python y tus datasets de entrenamiento al contenedor

¿Hay alguna parte específica de la configuración de tu entorno de Python (como librerías o rutas de datasets) que te esté dando error al intentar montarla

Jordi H.

Has dicho ahora explicamelo para el fichero de docker compose

Gemini

Si el Dockerfile es la receta, el **Docker Compose** es el director de orquesta. Aquí es donde realmente vas a configurar tus volúmenes el 99% del tiempo, porque es mucho más limpio que escribir comandos largos en la terminal. En Docker Compose, la sintaxis cambia ligeramente pero el concepto es el mismo: **Origen : Destin**

1. La sintaxis básica en `docker-compose.yml`

Dentro de un servicio, utilizas la palabra clave volumes:. Observa este ejemplo orientado a un entorno de **análisis de datos con Python**:YAM

CODE

```
services:
  analisis:
    image: python:3.11
    volumes:
      # TIPO BIND MOUNT (Desarrollo)
      - ./mi_codigo:/app/scripts

      # TIPO NAMED VOLUME (Persistencia de datos)
      - datos_procesados:/app/output

volumes:
  datos_procesados:
```

Análisis de la estructura:

- ./mi_codigo:/app/scripts:./mi_codigo: Es la ruta en tu PC (relativa al archivo .yaml)./app/scripts: Es la ruta dentro del contenedor. Uso: Si editas un archivo .py en tu PC, el cambio es instantáneo en el contenedor
- datos_procesados:/app/output: Aquí no hay un punto (.) ni una ruta absoluta. Solo un nombre. Docker crea este volumen en un lugar seguro de tu sistema y lo gestiona él. Uso: Ideal para guardar resultados de modelos de ML que no quieres borrar por accident

2. Diferencia clave: Bind Mounts vs. Named Volumes

Es vital entender cuándo usar cada uno en tu flujo de trabajo de datos:

Característica	Bind Mount (Ruta)	Named Volume (Solo nombre)
Sintaxis	- ./data:/container/data	- mi_volumen:/container/data
Control	Tú decides la carpeta exacta en tu	Docker decide la ubicación.
Uso ideal	Código fuente, scripts de Python,	Bases de datos, caché pesado,
Portabilidad	Difícil (las rutas cambian entre	Muy fácil (Docker lo abstrae).

3. ¿Cómo se relacionan el Dockerfile y el Compose?

Esta es la pregunta del millón. Si en tu Dockerfile pusiste VOLUME ["/data"] y en tu Docker Compose pones - ./mi_carpeta:/data, **el Compose gana**

1. El Dockerfile define un "espacio reservado" por seguridad.
2. El Docker Compose mapea ese espacio a una carpeta real que tú puedes ver y tocar en tu explorador de archivos

Ejemplo práctico "Full Stack" de Datos:

Imagina que estás procesando datos de clima:YAML

CODE

```
version: '3.8'
services:
  etl_python:
    build: .
    volumes:
      # Montas tu carpeta de scripts para no tener que reconstruir la imagen cada vez que cambias un
      print()
      - ./usr/src/app
      # Montas una carpeta de datos externa
      - /usuarios/datasets/clima:/data/input:ro # 'ro' significa solo lectura
```

4. Un par de trucos útiles

- Modo Solo Lectura (:ro): Muy recomendado para tus datasets originales. Evita que un error en tu script de Python borre o modifique los datos fuente
 - ./raw_data:/app/data:ro
 - Reflejar cambios: Si usas un Bind Mount (./), asegúrate de que tu contenedor no esté simplemente "copiando" los archivos al inicio. El volumen debe estar activo para que el "túnel" funcione
- ¿Te queda más claro por qué solemos preferir definir casi todo en el Compose en lugar del Dockerfile?

Jordi H.

Has dicho seguimos con el file de docker compose. cada sección services puede contener volumes y luego hay una sección de primer nivel llamada volumes. para qué sirve? qué tipo de contenidos puede tener

Gemini

Esta es la pregunta clave para dominar Docker Compose. Esa "duplicidad" de la palabra volumes suele confundir, pero tiene una lógica estructural muy clara: **el nivel superior define qué existe, y el nivel de servicio**

1. ¿Para qué sirve la sección de primer nivel `volumes`?

La sección de primer nivel sirve para **declarar volúmenes con nombre (Named Volumes)** Cuando pones un nombre ahí, le estás diciendo a Docker: "Crea un espacio de almacenamiento independiente que sobreviva incluso si borro los servicios o contenedores"

Características principales:

- Ciclo de vida independiente: Si haces docker compose down, los contenedores se borran, pero el volumen declarado arriba no se borra. Tus datos están a salvo
- Compartición: Permite que dos o más servicios distintos (por ejemplo, un script de Python y una base de datos) accedan exactamente a la misma carpeta de datos
- Abstracción: No te importa dónde guarda Docker los archivos físicamente (aunque suele ser en /var/lib/docker/volumes/). Solo te importa el nombre que le diste

2. Diferencia de contenidos: ¿Qué va en cada sitio?

Es más fácil entenderlo viendo qué "información" reside en cada lugar:

A. Sección de Primer Nivel (La Definición)

Aquí solo se listan los nombres de los volúmenes y, opcionalmente, configuraciones avanzadas (drivers).YAML

CODE

```
volumes:
datos_analisis: # Definición simple
config_db: # Otra definición
external: true # Indica que este volumen ya existía fuera de este fichero
```

B. Sección dentro de `services` (La Conexión)

Aquí es donde indicas **el mapa**: cómo se conecta ese volumen (u otra carpeta) dentro del contenedor específico.YAM

CODE

```
services:
python_processor:
image: mi_analitica_img
volumes:
- datos_analisis:/app/data # "Usa el volumen de arriba y ponlo en /app/data"
```

3. ¿Qué contenidos puede tener un volumen?

Como te vas a centrar en **Python y Analítica de Datos**, aquí tienes los casos de uso más comunes para lo que guardarás dentro

1. **Datasets Pesados**: Si manejas CSVs o archivos Parquet de varios GB, no quieres que estén "dentro" de la imagen del contenedor (porque la harían lentísima). Los guardas en un volumen
2. **Modelos Entrenados**: Los archivos .pkl, .h5 o .onnx resultantes de tus entrenamientos.
3. **Bases de Datos**: Si usas PostgreSQL o MongoDB para guardar tus datos procesados, los archivos de la base de datos deben ir en un volumen de primer nivel
4. **Logs y Resultados**: Archivos de texto o imágenes (gráficas de Matplotlib/Seaborn) generadas por tus scripts

4. Comparativa Visual: Bind Mount vs. Named Volume

Es fundamental que veas cuándo usas la sección de primer nivel y cuándo no:

Configuración	¿Usa sección nivel	Ejemplo en services	Uso Típico
Bind Mount	NO	- ./mi_codigo:/app	Tu código Python que
Named Volume	SÍ	- persistencia:/data	Una base de datos o

Ejemplo Completo de flujo de trabajo

Imagina un flujo donde un script de Python limpia datos y los guarda para que otro servicio los analice:YAML

CODE

```
services:
  limpiador:
    build: ./limpiador
  volumes:
    - ./scripts:/src # Bind mount (no necesita nivel superior)
    - compartido:/output # Named volume (Sí necesita nivel superior)

  analizador:
    build: ./analizador
  volumes:
    - compartido:/input:ro # Usa el MISMO volumen en modo solo lectura

# AQUÍ SE DEFINE EL "TUNEL" COMPARTIDO
volumes:
  compartido:
```

En este caso, la sección de primer nivel volumes es la que permite que limpiador y analizador hablen entre ellos usando el mismo disco duro virtual. ¿Te hace sentido esta separación, o hay alguna parte de la sintaxis (como los dos puntos o las rutas relativas) que te siga generando duda

Jordi H.

Has dicho que otros atributos y directivas tiene la sección volumes?

Gemini

La sección de primer nivel volumes en Docker Compose no es solo para dar nombres; es donde configuras el **comportamiento físico** del almacenamiento. Aunque para analítica de datos básica basta con el nombre, conocer estos atributos te permitirá conectar Docker con almacenamiento en la nube o bases de datos externas. Aquí tienes los atributos y directivas más importantes

1. Directivas de la sección de primer nivel (`volumes:`)

Estas se usan para configurar el volumen globalmente antes de que cualquier servicio lo use.

`driver`

Define qué "tecnología" gestiona el volumen. Por defecto es local, pero podrías usar otros para conectar con AWS S3, Azure Storage o carpetas de red.YAM

CODE

```
volumes:
  datos_remotos:
    driver: local # Es el valor por defecto
```

`driver\_opts` (Opciones del Driver)

Es el atributo más potente. Permite pasar configuraciones específicas al driver. Por ejemplo, puedes montar una carpeta de tu PC como si fuera un disco duro externo usando el protocolo NFS o simplemente bind-mounts avanzados.YA

CODE

```
volumes:
  mi_data:
    driver: local
    driver_opts:
      type: 'none'
      o: 'bind'
    device: '/home/usuario/datasets' # Ruta absoluta en tu PC
```

`external`

Si se establece en true, le indicas a Docker Compose que **no intente crear el volumen**, porque ya existe fuera de este archivo (creado manualmente con docker volume create).YAM

CODE

```
volumes:
  db_compartida:
    external: true
```

`labels`

Sirve para añadir metadatos (etiquetas) al volumen. Es útil en entornos de producción para organizar quién es el dueño del dato o qué versión tiene.YAM

CODE

```
volumes:
  modelos_ml:
    labels:
      proyecto: "analisis_ventas"
      version: "1.0"
```

2. Atributos dentro de la sección de servicios (`services:`)

Cuando conectas un volumen a un contenedor, puedes ir mucho más allá del simple origen:destino. Docker introdujo una **sintaxis larga** (más legible) para sustituir a la corta (- v)

Sintaxis larga (Recomendada para mayor claridad)

En lugar de - ./data:/app/data:ro, usas esto:YAML

CODE

```
services:
  python_task:
    image: python_analisis
    volumes:
      - type: bind # Puede ser 'volume', 'bind', 'tmpfs' o 'npipe'
        source: ./data # Origen en tu PC
        target: /app/data # Destino en el contenedor
        read_only: true # Atributo: protege tus datos originales
        bind:
          propagation: rprivate # Controla cómo se comparten montajes anidados
```

Atributos específicos del montaje:

1. `read_only`: Crucial en analítica. Evita que tu código de Python modifique accidentalmente los datasets fuente
2. `consistency`: (Solo para macOS/Windows). Optimiza la velocidad de sincronización entre tu PC y el contenedor
 - `consistent`: Sincronización perfecta (lento).
 - `cached`: El contenedor tiene prioridad (bueno para leer datasets).
 - `delegated`: El PC tiene prioridad (bueno para escribir logs rápidamente).
3. `tmpfs`: Crea un volumen que vive solo en la memoria RAM. Es ultra rápido pero los datos desaparecen al apagar el contenedor. Ideal para archivos temporales de procesamiento que no quieres que toquen el disco duro

Resumen de utilidad para Análítica de Datos

Para tu camino en **Python y Análisis de Datos**, quédate con este esquema mental:

Si quieres...	Usa este atributo	En qué sección
Usar un volumen creado por otro	<code>external: true</code>	Primer nivel
Evitar borrar tus CSVs originales	<code>read_only: true</code>	Nivel de servicio
Procesar datos en RAM para ir más	<code>type: tmpfs</code>	Nivel de servicio
Montar una carpeta específica de tu	<code>driver_opts (bind)</code>	Primer nivel (o sintaxis corta en

¿Te interesa ver cómo se configuraría un volumen `tmpfs` para acelerar el procesamiento de un `DataFrame` de `pandas` muy grande, o prefieres profundizar en cómo conectar volúmenes entre diferentes archivos de `Compos`

Jordi H.

Has dicho estoy usando `podman` en vez de `docker`. dónde se configura en el producto la zona del disco donde se guardan los `named volumes`

Gemini

Si vienes de `Docker`, la gran diferencia con **Podman** es que este último es "rootless" (sin privilegios de superusuario) por defecto. Esto significa que la ubicación de los volúmenes depende totalmente de **qué usuario** está ejecutando el contenedor. A diferencia de `Docker` (donde todo suele ir a `/var/lib/docker/volumes`), en `Podman` no hay un "único lugar" centralizado para todos los usuarios.

1. ¿Dónde se guardan físicamente?

Si usas Podman como usuario normal (Rootless)

Esta es la configuración estándar y la que más usarás para analítica en tu PC personal. Los volúmenes se guardan en tu "Home"

`~/.local/share/containers/storage/volumes/`

Si usas Podman como Root (Sudo)

Si ejecutas comandos con `sudo podman`, la ruta es idéntica a la de `Docker`:

`/var/lib/containers/storage/volumes/`

2. ¿Dónde se configura esta ruta? (El archivo `storage.conf`)

Podman no guarda esta configuración en un panel de control visual, sino en archivos de texto. El encargado de gestionar dónde vive todo el almacenamiento es el archivo **storage.conf**. Dependiendo de tu sistema, lo encontrarás en:

1. Nivel de usuario: `~/.config/containers/storage.conf` (Si no existe, puedes crearlo).
2. Nivel de sistema: `/etc/containers/storage.conf`.

Cómo cambiar la zona del disco:

Dentro de ese archivo, la directiva que controla la ubicación base es `graphroot`. Si quieres mover tus volúmenes (y tus imágenes) a un disco duro secundario más grande (algo común cuando tus datasets de Python crecen mucho), deberías editarlo así:

CODE

```
[storage]
driver = "overlay"
# Aquí cambias la ruta por defecto
graphroot = "/media/tu_disco_duro/podman_storage"
```

¡Ojo! Si cambias el `graphroot`, Podman dejará de ver las imágenes y volúmenes que tenías antes. Tendrías que mover los datos manualmente a la nueva carpeta.

3. ¿Cómo inspeccionar un volumen específico?

Si quieres saber la ruta exacta de un volumen que ya creaste en tu proyecto de Python, no necesitas buscar a ciegas. Usa el comando de inspección:

CODE

```
podman volume inspect nombre_de_tu_volumen
```

En la salida JSON, busca la clave **Mountpoint**. Te dirá la ruta absoluta exacta en tu disco duro donde están tus archivos.

4. Diferencia clave en el rendimiento (Análisis de Datos)

Como estás enfocado en analítica, ten en cuenta esto: Podman en modo rootless usa un driver llamado `fuse-overlayfs`.

- Ventaja: Más seguro.
- Desventaja: Puede ser un poco más lento al escribir miles de archivos pequeños en un volumen comparado con Docker nativo.

Si notas que tus scripts de Python tardan mucho en guardar archivos en los volúmenes, la solución suele ser ajustar las opciones de `mount_program` en el mismo archivo `storage.conf` que mencionamos arriba. ¿Estás trabajando en Linux o usas Podman Desktop en Windows/Mac? La gestión de las "máquinas virtuales" intermedias cambia un poco la ruta final.

Jordi H.

Has dicho estoy usando un `almalinux`. ya tengo configurado correctamente `runroot` y `graphroot` para usar un disco grande que me permita cualquier tamaño de datos. ahora me gustaría hacer lo mismo para `# Storage` e `path for rootless users` `## rootless_storage_path = "$HOME/.local/share/containers/storage"` pero no sé si debo generar previamente los directorios para cada usuario o hay alguna manera que se generen cuando se necesiten (primera ejecución de un nuevo usuario, por ejem

Excelente elección con **AlmaLinux**; para tareas de analítica y entornos profesionales es una roca. Respecto a tu duda sobre el almacenamiento rootless, no necesitas crear manualmente los directorios para cada usuario. Podman y el motor de almacenamiento subyacente están diseñados para gestionar esto de forma dinámica. Aquí te detallo cómo funciona y cómo deberías configurar

1. Comportamiento automático

No es necesario generar los directorios previamente. Cuando un usuario nuevo ejecuta por primera vez un comando de Podman (por ejemplo, `podman run` o `podman volume create`), el sistema

1. Lee la configuración de `storage.conf`.
2. Resuelve las variables de entorno (como `$HOME` o `$USER`).
3. Crea automáticamente la estructura de carpetas necesaria con los permisos correctos para ese usuario específico

2. Cómo configurar la ruta globalmente

Si quieres que todos los usuarios de tu AlmaLinux guarden sus datos rootless en ese disco grande, pero manteniendo su separación individual, debes editar el archivo `/etc/containers/storage.conf`. En lugar de una ruta estática, utiliza **variables de plantilla**. La más común es usar el nombre del usuario o su UID para evitar colisiones: `ini`, `T`

CODE

```
[storage]
# ... tus otras configuraciones ...

# Ejemplo: Guardar en el disco grande bajo una carpeta por usuario
rootless_storage_path = "/mnt/disco_grande/podman_users/$USER/storage"
```

¿Por qué usar ``$USER`` o ``$UID``?

Si apuntas a todos los usuarios a la misma carpeta exacta (ej. `/mnt/disco_grande/podman_data`), el primero que la use se adueñará de los permisos y el resto recibirá errores de "Permiso denegado". Al usar la variable, cada uno tiene su propio subdirector

3. Consideraciones de permisos (XFS/Ext4)

Dado que estás en AlmaLinux, es muy probable que uses **XFS**. Para que el almacenamiento rootless funcione correctamente en un disco externo o una partición distinta, asegúrate de lo siguiente

- Permisos de ejecución: El disco grande debe estar montado con permisos de ejecución.
- Propiedad de la base: La carpeta raíz (`/mnt/disco_grande/podman_users`) debe tener permisos que permitan a los usuarios escribir en ella, o bien estar configurada con el Sticky Bit (similar a `/tmp`) para que cada uno cree su subcarpeta. Recomendación: Crea la carpeta base y dale permisos 177

```
sudo chmod 1777 /mnt/disco_grande/podman_users
```

4. El problema de las cuotas y sub-UIDs

Como vas a trabajar con analítica de datos pesada, recuerda que en modo rootless, Podman utiliza un rango de UIDs secundarios (definidos en `/etc/subuid`). Si el disco grande está formateado en un sistema que no soporta **User Namespaces** adecuadamente, podrías tener problemas de permisos. En AlmaLinux esto no suele ser problema si usas XFS o Ext4, pero asegúrate de que el disco no esté en NTFS o FAT32, ya que esos sistemas no entienden la propiedad de archivos de Linux que Podman necesita replicar dentro del volu

Resumen de pasos:

1. Edita /etc/containers/storage.conf.
 2. Define rootless_storage_path = "/tu/disco/grande/\$USER/storage".
 3. Asegúrate de que la carpeta raíz en el disco grande tenga permisos de escritura para los usuarios.
 4. No crees nada manualmente; deja que el primer podman volume create de cada usuario haga el trabajo.
- ¿Vas a tener muchos usuarios concurrentes procesando datos o es principalmente para separar tus propios proyectos

Jordi H.

Has dicho no seran muchos usuarios pero se irán creando poco a poco. cómo migro un usuario que ya está usando su .local

Gemini

Migrar los datos de un usuario que ya tiene contenedores, imágenes y volúmenes en su \$HOME/.local es un proceso delicado porque Podman utiliza **User Namespaces**. Esto significa que los archivos dentro de los volúmenes no pertenecen realmente a tu usuario, sino a una serie de "UIDs secundarios" que Podman mapea. Para hacerlo correctamente en AlmaLinux sin romper los permisos de tus datos de analítica, si que estos pa

1. Preparar el terreno (Como Root)

Primero, asegúrate de que la nueva ubicación en el disco grande permita que el usuario gestione sus propios archivos y los de sus contenedores. Supongamos que el usuario se llama analista1 y tu disco grande está en /mnt/datos. Ba

CODE

```
# Crear la base si no existe
sudo mkdir -p /mnt/datos/podman_users

# Dar permisos para que los usuarios puedan crear sus carpetas
sudo chmod 1777 /mnt/datos/podman_users
```

2. La migración (Como el Usuario)

Importante: El usuario debe cerrar todas sus sesiones y detener cualquier contenedor o pod que esté corriendo. Bas

CODE

```
# 1. Detener todo lo relacionado con podman
podman stop -a
podman system migrate # Prepara el sistema para cambios
```

El comando de copia crítico

No puedes usar un cp normal. Debes usar rsync con el flag -a (archive) y -X (atributos extendidos/SELinux) para mantener la integridad de los metadatos de los contenedores: Bas

CODE

```
# Crear el destino
mkdir -p /mnt/datos/podman_users/$USER/storage

# Sincronizar los datos actuales al disco nuevo
rsync -aXHAW --progress ~/.local/share/containers/storage/ /mnt/datos/podman_users/$USER/storage/
```

3. Cambiar la configuración

Ahora debes decirle a Podman que, de ahora en adelante, busque los datos en la nueva ruta. Si quieres hacerlo solo para ese usuario, crea o edita su archivo personal
~/.config/containers/storage.confIni, TOML

CODE

```
[storage]
driver = "overlay"
graphroot = "/mnt/datos/podman_users/TU_USUARIO/storage"
```

Nota: En el archivo de configuración del usuario **no uses variables como \$USER**, escribe la ruta absoluta real. Las variables funcionan mejor en el archivo global /etc/containers/storage.conf

4. Verificación y Limpieza

Una vez cambiada la configuración, verifica que Podman reconoce la nueva ruta: Bash

CODE

```
podman info | grep -i "storage"
```

Si todo aparece correctamente (imágenes y volúmenes listados con `podman volume ls`), puedes proceder a borrar la carpeta antigua para liberar espacio en el disco principal: Bas

CODE

```
# SOLO después de confirmar que todo funciona
rm -rf ~/.local/share/containers/storage/
```

5. El "Truco" del Enlace Simbólico (Alternativa rápida)

Si no quieres lidiar con archivos de configuración para cada usuario que se cree, hay una forma más "limpia" a nivel de sistema de archivos

1. Mueves los datos como explicamos arriba.
2. Borrás la carpeta original ~/.local/share/containers/storage.
3. Creas un enlace simbólico:

```
ln -s /mnt/datos/podman_users/$USER/storage ~/.local/share/containers/storage
```

De esta forma, Podman cree que sigue escribiendo en el \$HOME, pero físicamente los datos de analítica están en el disco grande

¿Cuál prefieres?

La opción del **archivo de configuración** es la más "profesional" y robusta en AlmaLinux, mientras que el **enlace co**** es más sencillo si vas a tener muchos usuarios y no quieres explicarles cómo editar archivos .conf. ¿Crees que los usuarios actuales necesitarán ayuda para mover sus propios scripts de Python también, o solo te preocupa el almacenamiento de los volúmenes de datos?