Taller: Volúmenes y Bind Mounts en Docker (Linux/macOS)

Alec Fabian Corzo Salazar

UNIVERSIDAD PEDAGOGICA Y TECNOLOGICA DE COLOMBIA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BASES DE DATOS II
TUNJA
2025
VOLUMENES Y BIND MOUNTS EN DOCKER

Ejercicio 1 — Bind mount en modo lectura con Nginx

Objetivo: Entender cómo el host controla lo que el contenedor sirve.

1. Cree una carpeta y un archivo: mkdir -p ~/web & echo "Hola desde bind mount /h1>" > ~/web/index.html

```
┌──(corzo㉿Aleec)-[~]
└─$ mkdir -p web

┌──(corzo㉿Aleec)-[~]
└─$ cd web

┌──(corzo㉿Aleec)-[~/web]
└─$ nano index.html

┌──(corzo㉿Aleec)-[~/web]
└─$ ls
index.html

┌──(corzo㉿Aleec)-[~/web]
└─$ cat index.html
<h1>"Hola mundo desde bind mount"</h1>
```

2. Levante Nginx con bind mount de solo lectura: docker run -d -name web-ro -p 8080:80 \ -v ~/web:/usr/share/nginx/html:ro \ nginx:alpine

```
┌──(corzo㉿Aleec)-[~/web]
└─$ docker run -d --name web-ro -p 8080:80 \
  -v ~/web:/usr/share/nginx/html:ro \
  nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
9824c27679d3: Pull complete
6bc572a340ec: Pull complete
403e3f251637: Pull complete
9adfbae99cb7: Pull complete
7a8a46741e18: Pull complete
c9ebe2ff2d2c: Pull complete
a992fbc61ecc: Pull complete
cb1ff4086f82: Pull complete
Digest: sha256:42a516af16b852e33b7682d5ef8acbd5d13fe08fecadc7ed98605ba5e3b26ab8
Status: Downloaded newer image for nginx:alpine
c933f6b778ae8a5f15f1e60cc7ad1a87dbfeec6076524cdcaec232d49b85619c
```

3. Abra http://localhost:8080 y verifique.

4. Edite index.html en el host y recargue el navegador, el cambio debe verse.

5. Intente crear un archivo dentro del contenedor: docker exec -it web-ro sh -lc 'echo test > /usr/share/nginx/html/test.txt' - Debe dar error (“Read-only file system”)

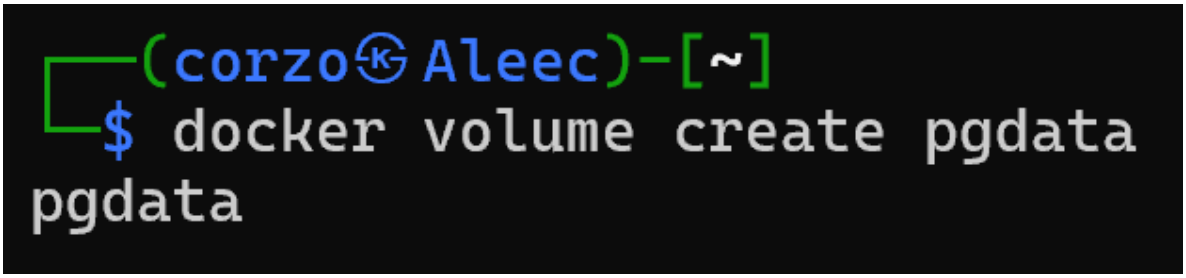
Reflexión:

Aprendí a utilizar un bind mount en modo lectura para que Nginx sirviera archivos directamente desde una carpeta en el host. Esto me permitió comprender cómo los cambios hechos en los archivos locales se reflejan en el contenedor de manera inmediata, y también entendí la importancia de usar la opción de solo lectura para proteger el sistema de archivos, ya que al intentar escribir desde el contenedor se generaba un error.

Ejercicio 2 — Named volume con PostgreSQL

Objetivo: Comprobar persistencia de datos.

1. Cree un volumen: docker volume create pgdata



2. Ejecute PostgreSQL: docker run -d -name pg -e POSTGRES_PASSWORD=postgres -p 5432:5432 \ -v pgdata:/var/lib/postgresql/data \ postgres:16-alpine

```
(corzo㉿Aleec)-[~]
$ docker run -d --name pg -e POSTGRES_PASSWORD=postgres   -p 5432:5432 -v pgdata:/var/lib/postgresql/data   postgres:1
6-alpine
Unable to find image 'postgres:16-alpine' locally
16-alpine: Pulling from library/postgres
9824c27679d3: Already exists
01ef787617d5: Pull complete
d444581c5dc1: Pull complete
127625cab66d: Pull complete
7f8bf47818a2: Pull complete
0951477387e1: Pull complete
878e28e3ecd5: Pull complete
d079e32a74cc: Pull complete
cb87d3c01966: Pull complete
40af0ccd9733: Pull complete
0b003ba20c51: Pull complete
Digest: sha256:8ffca822c1933bdc8be7dbbe9c2330974bdb43f5027f47717772fa35925412b0
Status: Downloaded newer image for postgres:16-alpine
0c744106ed39dce1494f18555a51b9ad731d1a733d832f36a58471280b9bd5b8
```

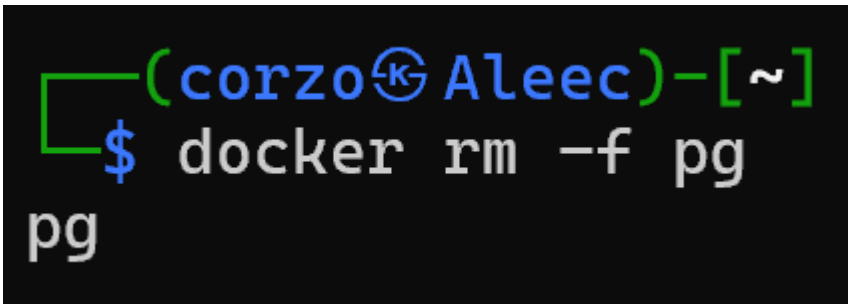
3. Cree una tabla y agregue datos: docker exec -it pg psql -U postgres -c "CREATE TABLE test(id serial, nombre text);" docker exec -it pg psql -U postgres -c "INSERT INTO test(nombre) VALUES ('Ada'),('Linus');" docker exec -it pg psql -U postgres -c "SELECT * FROM test;"

```
(corzo㉿Aleec)-[~]
$ docker exec -it pg psql -U postgres -c "CREATE TABLE test(id serial, nombre text);"
CREATE TABLE

(corzo㉿Aleec)-[~]
$ docker exec -it pg psql -U postgres -c "INSERT INTO test(nombre) VALUES ('Ada'),('Linus');"
INSERT 0 2

(corzo㉿Aleec)-[~]
$ docker exec -it pg psql -U postgres -c "SELECT * FROM test;"
 id | nombre
----+--------
  1 | Ada
  2 | Linus
(2 rows)
```

4. Elimine el contenedor: docker rm -f pg



5. Vuelva a levantarlo usando el mismo volumen y verifique que los datos siguen allí.

```
┌──(corzo㉿Aleec)-[~]
└─$ docker run -d --name pg -e POSTGRES_PASSWORD=postgres \
  -p 5432:5432 -v pgdata:/var/lib/postgresql/data \
  postgres:16-alpine
a1434d3676d5346d4e629f121cf26c10e1a3bebb85fc265e82f4edf7302d68a3

┌──(corzo㉿Aleec)-[~]
└─$ docker exec -it pg psql -U postgres -c "SELECT * FROM test;"
 id | nombre
----+--------
  1 | Ada
  2 | Linus
(2 rows)
```

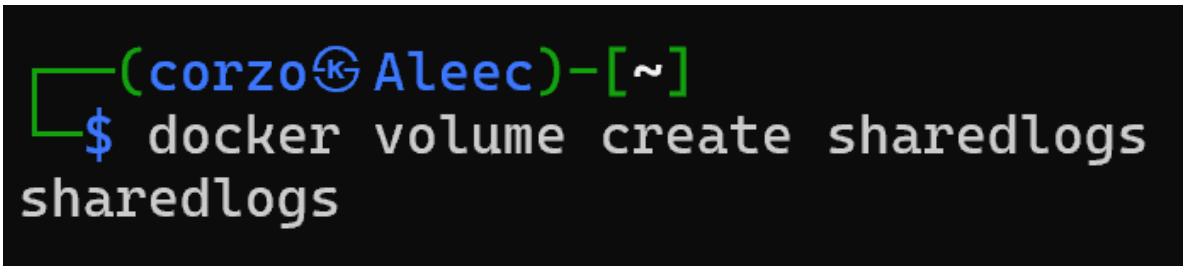
Reflexión:

Comprobé la utilidad de los volúmenes nombrados con PostgreSQL, ya que a diferencia de los bind mounts, estos volúmenes permiten mantener los datos incluso si el contenedor es eliminado. Pude crear una tabla, insertar registros y al volver a levantar el contenedor con el mismo volumen, verifiqué que los datos seguían intactos. Esto me enseñó cómo asegurar persistencia de información en servicios de bases de datos dentro de Docker.

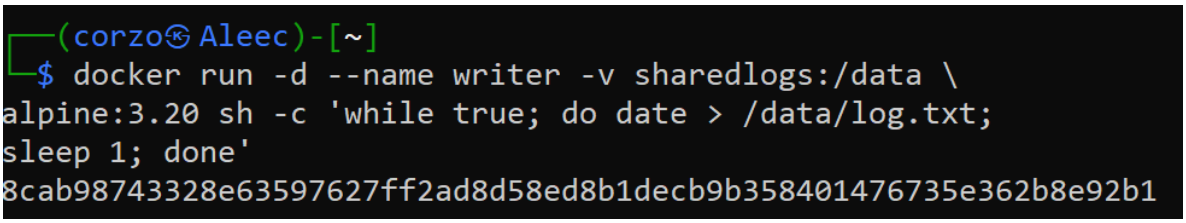
Ejercicio 3 — Volumen compartido entre dos contenedores

Objetivo: Producir y consumir datos simultáneamente.

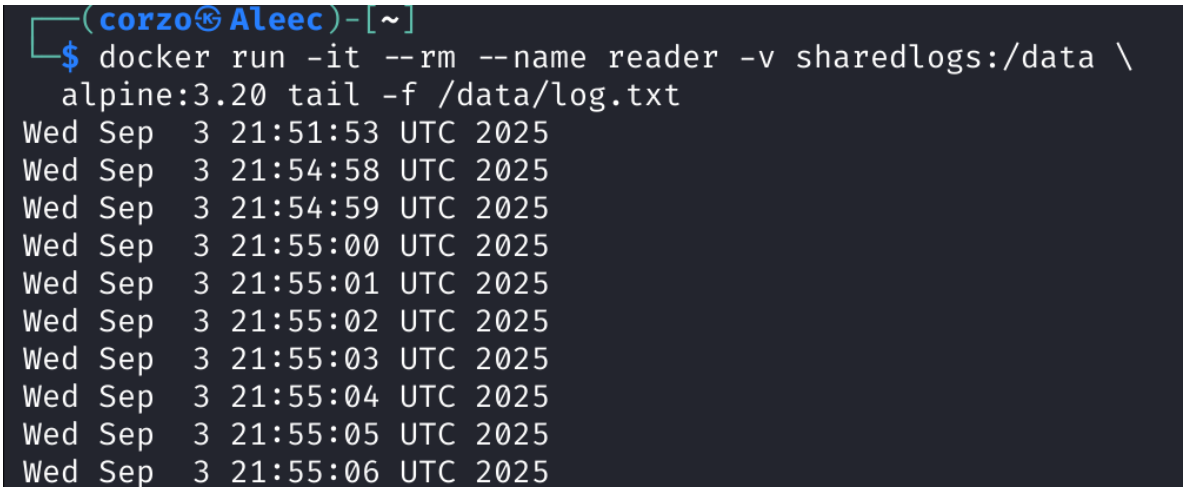
1. Cree un volumen: docker volume create sharedlogs



2. Productor (escribe timestamps cada segundo): docker run -d --name writer -v sharedlogs:/data \ alpine:3.20 sh -c 'while true; do date > /data/log.txt; sleep 1; done'



3. Consumidor (lee en tiempo real): docker run -it --rm --name reader -v sharedlogs:/data \ alpine:3.20 tail -f /data/log.txt - Para detener el proceso ejecute Ctrl+C.



4. Reinicie el productor y revise que el archivo siga creciendo: docker rm -f writer
docker run -d -name writer -v sharedlogs:/data \ alpine:3.20 sh -c 'while true; do date > /data/log.txt; sleep 1; done'

docker run -rm -v sharedlogs:/data alpine:3.20 sh -lc 'tail -n 3 /data/log.txt'

```
┌──(corzo㉿Aleec)-[~]
└─$ docker rm -f writer
docker run -d --name writer -v sharedlogs:/data \
  alpine:3.20 sh -c 'while true; do date >> /data/log.txt; sleep 1; done'

docker run --rm -v sharedlogs:/data alpine:3.20 sh -lc 'tail -n 3 /data/log.txt'
writer
fedebc71cd2f3e1a9dc4bb9cc61bb4de83138991375e1fafbe7d08609bbb21e1
Wed Sep  3 22:00:07 UTC 2025
Wed Sep  3 22:00:08 UTC 2025
Wed Sep  3 22:00:08 UTC 2025
```

Reflexión:

Este ejercicio me permitió ver cómo un volumen puede ser compartido entre varios contenedores, en este caso uno productor que escribía datos y otro consumidor que los leía en tiempo real. Fue interesante comprobar cómo los cambios generados por un contenedor eran visibles al instante para el otro, lo que demuestra que los volúmenes son muy útiles para comunicar procesos o compartir información sin necesidad de conexiones de red entre contenedores.

Ejercicio 4 — Backup y restauración de un volumen

Objetivo: Aprender a respaldar y restaurar datos.

1. Cree un volumen y añade un archivo: docker volume create appdata docker run -rm -v appdata:/data alpine:3.20 sh -lc 'echo "backup-$(date +%F)" > data/info.txt'

```
┌──(corzo㉿Aleec)-[~]
└─$ docker volume create appdata
docker run --rm -v appdata:/data alpine:3.20 \
  sh -lc 'echo "backup-$(date +%F)" > /data/info.txt'
appdata
```

2. Haga backup a un tar en el host: mkdir -p ~/backups docker run -rm -v appdata:/data:ro -v ~/backups:/backup \ alpine:3.20 sh -lc 'cd /data & tar czf /backup/appdata.tar.gz .'

```
┌──(corzo㉿Aleec)-[~]
└─$ mkdir -p ~/backups
docker run --rm -v appdata:/data:ro -v ~/backups:/backup \
  alpine:3.20 sh -lc 'cd /data && tar czf /backup/appdata.tar.gz .'
```

3. Restaure en un nuevo volumen: docker volume create appdata_restored docker run -rm -v appdata_restored:/data -v ~/backups:/backup \ alpine:3.20 sh -lc 'cd /data & tar xzf /backup/appdata.tar.gz'

```
┌──(corzo㉿Aleec)-[~]
└─$ docker volume create appdata_restored
docker run --rm -v appdata_restored:/data -v ~/backups:/backup \
  alpine:3.20 sh -lc 'cd /data && tar xzf /backup/appdata.tar.gz'
appdata_restored
```

4. Verifique el contenido restaurado: docker run -rm -v appdata_restored:/data alpine:3.20 cat /data/info.txt

```
┌──(corzo㉿Aleec)-[~]
└─$ docker run --rm -v appdata_restored:/data alpine:3.20 cat /data/info.txt
backup-2025-09-03
```

Reflexión:

En el cuarto ejercicio aprendí a realizar respaldos y restauraciones de volúmenes. Pude crear un volumen con información, hacer un backup en el host en formato comprimido y luego restaurarlo en un nuevo volumen, verificando que los datos se

mantenían. Esto me mostró que los volúmenes en Docker, además de dar persistencia, son portables y permiten mantener copias de seguridad para proteger los datos o moverlos entre diferentes entornos.