

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

JUAN SEBASTIAN BARAJAS VARGAS

COD:202221704

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 "<h1>Hola desde bind mount /h1>" > ~/web/index.html
```

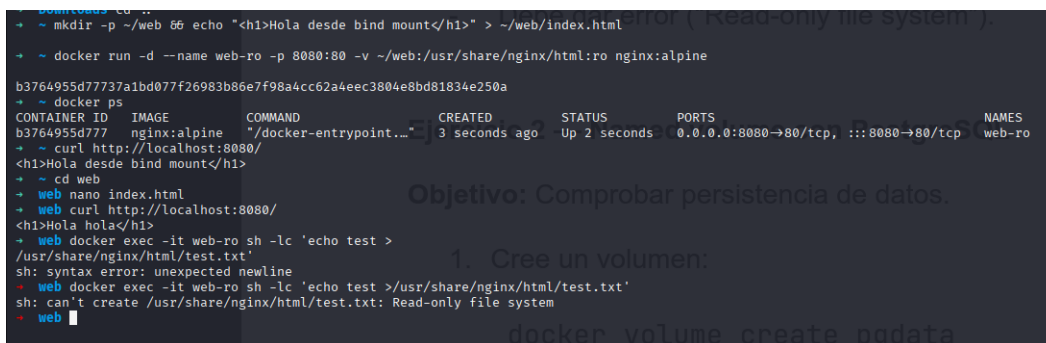
2. Levante Nginx con bind mount de solo lectura:

```
docker run -d -name web-ro -p 8080:80 \ -v ~/web:/usr/share/nginx/html:ro \ nginx:alpine
```

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").



```
+ ~$ mkdir -p ~/web && echo "<h1>Hola desde bind mount</h1>" > ~/web/index.html
+ ~$ docker run -d --name web-ro -p 8080:80 -v ~/web:/usr/share/nginx/html:ro nginx:alpine
b3764955d77737a1bd077f26983b86e7f98a4cc62a4eec3804e8bd81834e250a
+ ~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS      PORTS                               NAMES
b3764955d777  nginx:alpine  "/docker-entrypoint..."  3 seconds ago  Up 2 seconds  0.0.0.0:8080->80/tcp, :::8080->80/tcp  web-ro
+ ~$ curl http://localhost:8080/
<h1>Hola desde bind mount</h1>
+ ~$ cd web
+ web$ nano index.html
+ web$ curl http://localhost:8080/
<h1>Hola hola</h1>
+ web$ docker exec -it web-ro sh -lc 'echo test > /usr/share/nginx/html/test.txt'
sh: syntax error: unexpected newline
+ web$ docker exec -it web-ro sh -lc 'echo test > /usr/share/nginx/html/test.txt'
sh: can't create /usr/share/nginx/html/test.txt: Read-only file system
+ web$
```

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
```

```
+ ~ docker volume create pgdata
pgdata
+ ~ docker run -d --name pg -e POSTGRES_PASSWORD=postgres -p5432:5432 -v pgdata:/var/lib/postgresql/data postgres:16-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
40af0cccd9733: Pull complete
0b003ba20c51: Pull complete
Digest: sha256:8ffca822c1933bdc8be7dbbe9c2330974bdb43f5027f47717772fa35925412b0
Status: Downloaded newer image for postgres:16-alpine
4870f4af087a8a11b6f57ebb1c68c98d7c4c7f78821cf669b32892f4351a3359
```

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;"
```

```
+ ~ docker exec -it pg psql -U postgres -c "CREATE TABLE
test(id serial, nombre text);"
CREATE TABLE
+ ~ docker exec -it pg psql -U postgres -c "INSERT INTO
test(nombre) VALUES ('Ada'),('Linus');"
INSERT 0 2
+ ~ 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.

```
+ ~ docker exec -it pg psql -U postgres -c "CREATE TABLE
test(id serial, nombre text);"
CREATE TABLE
+ ~ docker exec -it pg psql -U postgres -c "INSERT INTO
test(nombre) VALUES ('Ada'),('Linus');"
INSERT 0 2
+ ~ docker exec -it pg psql -U postgres -c "SELECT * FROM
test;"
id | nombre
---+-----
 1 | Ada
 2 | Linus
(2 rows)

+ ~ docker rm -f pg
pg
+ ~ docker run -d --name pg -e POSTGRES_PASSWORD=postgres -p5432:5432 -v pgdata:/var/lib/postgresql/data postgres:16-alpine
052ccda639058fcb32de996e7b3e277a5f323899d029d116c95e6906fc7e733b
+ ~ docker exec -it pg psql -U postgres -c "SELECT * FROM
test;"
id | nombre
---+-----
 1 | Ada
 2 | Linus
(2 rows)

+ ~
```

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
```

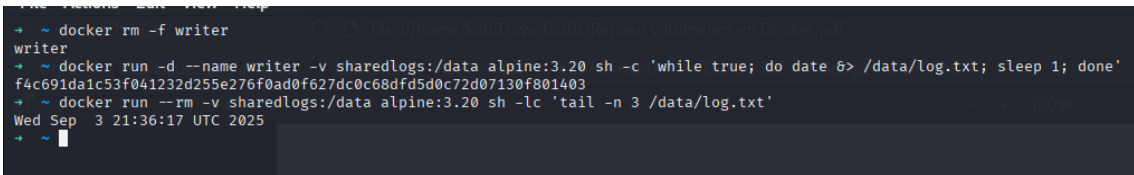
- Para detener el proceso ejecute Ctrl+C.

```
+ ~ docker volume create sharedlogs
sharedlogs
+ ~ docker run -d --name writer -v sharedlogs:/data alpine:3.20 sh -c 'while true; do date > /data/log.txt; sleep 1; done'
Unable to find image 'alpine:3.20' locally
3.20: Pulling from library/alpine
01d036902a3c: Pull complete
Digest: sha256:b3119ef930faabb6b7b976780c0c7a9c1aa24d0c75e9179ac10e6bc9ac080d0d
Status: Downloaded newer image for alpine:3.20
527b86d129b0f67c1860a32a43bfaa5cd0ffed33329613eb1da8613b340f94e
+ ~ docker run -it --rm --name reader -v sharedlogs:/data alpine:3.20 tail -f /data/log.txt
Wed Sep  3 21:28:55 UTC 2025
```

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'
```



```
+ ~ docker rm -f writer  
writer  
+ ~ docker run -d --name writer -v sharedlogs:/data alpine:3.20 sh -c 'while true; do date 6> /data/log.txt; sleep 1; done'  
f4c691da1c53f041232d255e276f0ad0f627dc0c68dfd5d0c72d07130f801403  
+ ~ docker run --rm -v sharedlogs:/data alpine:3.20 sh -lc 'tail -n 3 /data/log.txt'  
Wed Sep 3 21:36:17 UTC 2025  
+ ~
```

Ejercicio 4 — Backup y restauración de un volumen

Objetivo: Aprender a respaldar y restaurar datos.

1. Cree un volumen y añada un archivo:

```
docker volume create appdata docker run -rm -v  
appdata:/data alpine:3.20 sh -lc 'echo "backup-$(date  
+%F)" > data/info.txt'
```

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 .'
```

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'
```

4. Verifique el contenido restaurado:

```
docker run --rm -v appdata_restored:/data alpine:3.20
cat /data/info.txt
```

```
+ ~ docker volume create appdata
appdata
+ ~ docker run --rm -v appdata:/data alpine:3.20 sh -lc 'echo "backup-$(date +%F)" > data/info.txt'
+ ~ 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 .'
+ ~ 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
+ ~ docker run --rm -v appdata_restored:/data alpine:3.20 cat /data/info.txt
backup-2025-09-03
+ ~
```

Entrega:

- Comandos ejecutados.
- Capturas de salida (`curl`, `psql`, `tail -f`, listados).
- Breve reflexión: ¿qué aprendió en cada ejercicio?, ¿qué problemas tuvo?, ¿cómo los resolvió?

En todos los ejercicios tuve un mismo problema: al copiar los comandos desde el enunciado o mi teclado, a veces se corrompían los caracteres (`--rm`, `--name`, `\`, `</h1>`).

1. Entendí cómo un *bind mount* conecta directamente una carpeta del host con el contenedor. Al editar el `index.html` en el host, el cambio se reflejó inmediatamente en el navegador, sin tener que reconstruir nada. Además, confirmé que montar la carpeta en modo `:ro` evita que el contenedor pueda escribir en ella, garantizando seguridad.
2. Comprobé que los *volúmenes* permiten persistencia real de datos. Incluso eliminando el contenedor y recreándolo, la información en la tabla seguía ahí. Esto me ayuda a comprender cómo funcionan los volúmenes en aplicaciones de bases de datos.
3. Aprendí que dos contenedores pueden compartir un mismo volumen y comunicarse de forma indirecta: uno escribe y el otro lee en tiempo real. Esto demuestra cómo se pueden diseñar flujos de entrada y salida con Docker.
4. Comprendí cómo hacer un *backup* de un volumen en un archivo `.tar.gz` y luego restaurarlo en otro volumen. Esto es útil para migrar datos o protegerlos antes de actualizar un contenedor. en el comando `date +%F` (puse `date +%F`). Después de corregir la sintaxis, pude crear el backup, restaurarlo y verificar que el archivo `info.txt` se mantenía con el contenido original.