Persistencia de Datos y Arquitectura de Red en Docker

Este documento técnico detalla los mecanismos de persistencia y comunicación en contenedores Docker. El objetivo es comprender cómo el *Docker Daemon* gestiona el almacenamiento fuera del sistema de archivos *Copy-on-Write* (CoW) y cómo funciona la resolución de nombres (DNS) interna.

1. Subsistema de Almacenamiento (Storage Drivers)

Para entender la persistencia, primero se debe comprender la **efimeridad**. Un contenedor se construye sobre una imagen de solo lectura. Cuando un contenedor arranca, Docker añade una capa superior de lectura/escritura (Container Layer).

- **El Problema:** Cualquier cambio realizado en esta capa utiliza un *Storage Driver* (como overlay2 en Linux). Esto implica una penalización de rendimiento (overhead) y, lo más crítico, **los datos están acoplados al ciclo de vida del proceso**. Si el contenedor se destruye (docker rm), esa capa desaparece.
- **La Solución:** "Perforar" esa capa efímera para escribir directamente en el *Host Filesystem*.

1.1. Volúmenes (Docker Volumes) vs. Bind Mounts

Aunque ambos logran persistencia, su gestión a nivel de kernel y daemon es diferente.

A. Volúmenes (Named Volumes)

Son objetos de primera clase gestionados íntegramente por Docker. Se almacenan en una ruta del host gestionada por el daemon (usualmente /var/lib/docker/volumes/ en Linux).

- **Abstracción:** El usuario no necesita conocer la ruta física.
- **Gestión:** Se crean, listan y purgan mediante la CLI de Docker.
- **Rendimiento:** Nativo. Evitan la sobrecarga del sistema de archivos del contenedor.
- **Caso de uso:** Bases de datos (PostgreSQL, MySQL), almacenamiento de logs persistentes. Es la opción semánticamente correcta para producción.

Ejemplo Técnico con PostgreSQL:

1. Creación explícita del objeto Volumen:
   A diferencia de la creación implícita (al vuelo), aquí definimos el almacenamiento primero.
   docker volume create pg_data_prod

2. Inspección del objeto:
   Para verificar dónde reside físicamente (Mountpoint):
   docker volume inspect pg_data_prod

3. Montaje en el contenedor:
   Mapeamos el volumen lógico pg_data_prod a la ruta interna donde Postgres escribe sus datos (/var/lib/postgresql/data).

```
docker run -d \
  --name db-postgres-vol \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -v pg_data_prod:/var/lib/postgresql/data \
  postgres:15-alpine
```

   Nota: Si borramos el contenedor db-postgres-vol, el volumen pg_data_prod y sus datos persisten.

B. Bind Mounts

Vinculan una ruta absoluta específica del sistema operativo anfitrión (Host) a una ruta dentro del contenedor.

- **Dependencia:** Dependen de la estructura de directorios del Host.
- **Riesgos:** Problemas de permisos (UID/GID mismatch) y portabilidad (la ruta /home/usuario/proyecto no existe en todos los servidores).
- **Caso de uso:** Desarrollo activo. Inyectar código fuente o archivos de configuración (postgresql.conf) que se editan en el host y se deben reflejar inmediatamente.

Ejemplo Técnico con PostgreSQL (Inyección de configuración):

Supongamos que tenemos un archivo de configuración customizado en $(pwd)/custom-config/.

```
docker run -d \
  --name db-postgres-bind \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -v $(pwd)/custom-config:/etc/postgresql \
  postgres:15-alpine
```

Tabla Comparativa Técnica

Característica	Named Volume	Bind Mount
Ubicación en Host	/var/lib/docker/volumes/...	Ruta arbitraria definida por usuario
Gestión	CLI Docker (docker volume ...)	Manual (mkdir, chown, etc.)
Portabilidad	Alta (independiente del SO)	Baja (depende de rutas absolutas)
Drivers	Soporta drivers remotos (NFS, S3, Cloud)	Solo sistema de archivos local
Uso Recomendado	**Persistencia de Datos (DBs)**	**Inyección de Código/Config**

2. Subsistema de Redes (Networking)

Docker actúa como un router virtual gestionando interfaces de red (veth pairs) y puentes (bridges).

2.1. Bridge Network (User-defined vs Default)

El concepto crítico aquí es la **Resolución de Nombres (Service Discovery)**.

1. **Default Bridge (bridge):** Es la red donde caen los contenedores si no se especifica nada. **Limitación crítica:** No permite resolución DNS automática por nombre de contenedor. Solo se pueden comunicar por direcciones IP (las cuales son volátiles y cambian al reiniciar).
2. **User-Defined Bridge:** Al crear una red propia, Docker habilita un **servidor DNS embebido** (127.0.0.11). Esto permite que un contenedor haga ping basedatos y resuelva correctamente la IP interna, independientemente de cuál sea.

2.2. Aislamiento

Las redes proporcionan seguridad. Un contenedor en la red_frontend no puede acceder a paquetes en la red_backend a menos que esté conectado a ambas.

Ejemplo Práctico con PostgreSQL:

1. **Creación de la red:**
   docker network create red_backend

2. **Despliegue de Postgres en la red:**
   ```
   docker run -d \
     --name servidor-db \
     --network red_backend \
     -e POSTGRES_PASSWORD=secret \
     postgres:15-alpine
   ```

3. Prueba de conectividad (Cliente psql temporal):

Lanzamos otro contenedor en la misma red para probar el DNS.
```
docker run -it --rm \
  --network red_backend \
  postgres:15-alpine \
  psql -h servidor-db -U postgres
```

Observe el flag -h servidor-db. Docker resuelve ese nombre a la IP interna del contenedor creado en el paso 2.

3. Ejercicio de Integración: Despliegue de WordPress (Paso a Paso)

Arquitectura:
Desplegaremos un CMS (WordPress) y su base de datos.

- **Nota de Ingeniería:** WordPress está diseñado nativamente para funcionar con **MySQL/MariaDB**. Aunque existen forks para PostgreSQL, en entornos de producción estándar y para este ejercicio académico, utilizaremos la pila LEMP estándar (Linux, Nginx/Apache, MySQL/MariaDB, PHP).
- **Requisito:** La base de datos no debe exponer puertos al exterior (seguridad), solo el WordPress. Los datos deben ser totalmente persistentes.

Paso 1: Definición de la Infraestructura de Red

Creamos un segmento de red aislado para esta aplicación.
```
docker network create red_wordpress
```

Paso 2: Aprovisionamiento del Almacenamiento (Volúmenes)

Necesitamos dos volúmenes independientes para desacoplar los datos del ciclo de vida de los contenedores.

1. Para la base de datos (persistencia de tablas SQL):
   ```
   docker volume create vol_db_data
   ```

2. Para los ficheros estáticos de WordPress (uploads, plugins, themes):
   ```
   docker volume create vol_wp_html
   ```

Paso 3: Despliegue de la Base de Datos (Backend)

Usaremos MariaDB. Configuraremos las variables de entorno para inicializar la DB.

- --network: La unimos a nuestra red aislada.
- -v: Montamos el volumen creado en el punto de montaje estándar de MySQL (/var/lib/mysql).
- **Importante:** No usamos -p (publish ports). La base de datos será inaccesible desde

internet, solo accesible por contenedores en red_wordpress.

```
docker run -d \
  --name db-mariadb \
  --network red_wordpress \
  -v vol_db_data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=rootpassword \
  -e MYSQL_DATABASE=wordpress_db \
  -e MYSQL_USER=wp_user \
  -e MYSQL_PASSWORD=wp_password \
  mariadb:10.6
```

Paso 4: Despliegue de la Aplicación (Frontend)

Usaremos la imagen oficial de WordPress.

- -p 8080:80: Exponemos el puerto 80 del contenedor al 8080 de su máquina host.
- -v: Montamos el volumen para persistir wp-content.
- Variables de entorno: Le indicamos dónde está la base de datos usando su **nombre de host** (db-mariadb), gracias al DNS de la red Docker.

```
docker run -d \
  --name app-wordpress \
  --network red_wordpress \
  -v vol_wp_html:/var/www/html \
  -e WORDPRESS_DB_HOST=db-mariadb \
  -e WORDPRESS_DB_USER=wp_user \
  -e WORDPRESS_DB_PASSWORD=wp_password \
  -e WORDPRESS_DB_NAME=wordpress_db \
  -p 8080:80 \
  wordpress:latest
```

Paso 5: Verificación y Validación

1. Acceda a http://localhost:8080 en su navegador. Verá el instalador de WordPress. Complete la instalación.
2. Prueba de persistencia:
   Elimine ambos contenedores:
   docker rm -f app-wordpress db-mariadb

   Recree los contenedores ejecutando nuevamente los comandos del Paso 3 y Paso 4.
   Al volver a entrar a http://localhost:8080, no verá el instalador. Verá su sitio web ya configurado. Esto confirma que los datos sobrevivieron en los volúmenes vol_db_data y vol_wp_html.