Persistencia y Conectividad: Volúmenes y Redes en Docker

Hasta ahora, nuestros contenedores han sido efímeros y aislados. Si borramos un contenedor, sus datos desaparecen. Si creamos dos contenedores, no saben de la existencia del otro.
En este documento, abordaremos dos pilares fundamentales para entornos profesionales:

1. **Volúmenes (Storage):** Para que los datos sobrevivan al ciclo de vida del contenedor.
2. **Redes (Networking):** Para que los contenedores se comuniquen entre sí y con el mundo exterior.

Parte 1: Persistencia de Datos (Storage)

Por defecto, todos los archivos creados dentro de un contenedor se almacenan en una **capa de escritura**. Cuando el contenedor se elimina, esa capa **se destruye**.
Para evitar esto, Docker nos ofrece tres mecanismos principales:

1.1. Tipos de Almacenamiento

Tipo	Descripción	Caso de Uso Ideal
Volumes	Gestionados por Docker en una ruta segura del host (/var/lib/docker/volumes/). Son la opción preferida.	Bases de datos, backups, compartir datos entre contenedores.
Bind Mounts	Mapean un archivo o carpeta real de tu máquina (Host) a una ruta del contenedor.	Entornos de desarrollo (editar código en vivo), ficheros de configuración.
tmpfs	Se guardan en la memoria RAM del host. Nunca se escriben en disco.	Datos sensibles (claves) o caché de alto rendimiento que no debe persistir.

1.2. Práctica: Trabajando con Volúmenes (Volumes)

Los volúmenes son la forma "nativa" de Docker. No dependen de la estructura de carpetas de tu SO (Windows/Mac/Linux).

Ejemplo: Persistencia en una Base de Datos

Imagina que ejecutamos un contenedor de MongoDB. Si no usamos volumen, al borrar el contenedor, perdemos los usuarios registrados.

1. **Crear un volumen manualmente:**
   docker volume create mis_datos_mongo

2. Ejecutar el contenedor asociando el volumen:
   La sintaxis es -v nombre_volumen:ruta_dentro_contenedor.
   docker run -d --name mi-mongo -v mis_datos_mongo:/data/db mongo

3. **Prueba de Fuego:**
   - Entra, guarda un dato.
   - Borra el contenedor (docker rm -f mi-mongo).
   - Crea uno nuevo usando el **mismo volumen**.
   - ¡El dato seguirá ahí!

1.3. Práctica: Trabajando con Bind Mounts

Este es el favorito de los desarrolladores. Permite que lo que editas en tu editor de código (VS Code) se refleje instantáneamente dentro del contenedor.

Ejemplo: Servidor Web en Desarrollo (Hot Reload)

Supongamos que tienes una carpeta mi-web en tu escritorio con un index.html.

1. Comando con ruta absoluta (o $(pwd) en Linux/Mac):
   La sintaxis es -v ruta_host:ruta_contenedor.
   docker run -d -p 8080:80 -v $(pwd)/mi-web:/usr/share/nginx/html nginx

2. **El efecto mágico:**
   - Abre tu navegador en localhost:8080.
   - Edita el index.html en tu ordenador. Guarda.
   - Recarga el navegador. ¡El cambio es instantáneo sin reconstruir la imagen!

Parte 2: Redes en Docker (Networking)

La filosofía de Docker es el aislamiento, pero las aplicaciones modernas son distribuidas (Frontend habla con Backend, Backend habla con DB). Para ello usamos **Redes**.

2.1. Drivers de Red Principales

1. **Bridge (Por defecto):** Crea una red privada interna. Los contenedores en la misma red bridge pueden hablarse.
2. **Host:** Elimina el aislamiento de red. El contenedor usa la IP y puertos de la máquina anfitriona directamente.
3. **None:** Aislamiento total. Sin red.
4. **Overlay:** Para conectar contenedores que están en *distintos servidores* (usado en Docker Swarm/Kubernetes).

2.2. La Magia de la Resolución de Nombres (DNS)

La característica más potente de las redes personalizadas de Docker es el DNS automático.

Si dos contenedores están en la misma red personalizada, pueden hacerse ping usando el nombre del contenedor en lugar de la IP (que cambia siempre).

Ejemplo Práctico: Conectando dos contenedores

Vamos a simular una app que necesita hablar con otra. Usaremos la imagen alpine que es muy ligera para hacer pruebas de red.

1. Crear una red personalizada:
docker network create mi-red-app

2. Crear el Contenedor 1 (Servidor):
Lo llamaremos backend y lo uniremos a la red.
docker run -d --name backend --network mi-red-app alpine sleep 3600

3. Crear el Contenedor 2 (Cliente):
Lo llamaremos frontend y lo uniremos a la misma red.
docker run -it --name frontend --network mi-red-app alpine sh

4. Prueba de conexión (Dentro de frontend):
Ahora, dentro de la terminal del frontend, intentamos contactar al backend por su nombre.
Dentro del contenedor frontend
ping backend

Resultado: Verás que resuelve la IP automáticamente (64 bytes from 172.18.0.2...). Docker actúa como un servidor DNS interno.

2.3. Ejemplo: Red tipo "Host"

A veces no quieres gestionar puertos (-p 8080:80). Quieres que el contenedor sea "uno más" en tu red local.
docker run -d --network host nginx

- **Nota:** Esto solo funciona nativamente en Linux. En Windows/Mac el driver host tiene limitaciones por la máquina virtual intermedia.
- El contenedor ya no tiene IP propia interna, usa la IP de tu máquina. Si Nginx usa el puerto 80, ocupará el puerto 80 de tu ordenador real.

3. Resumen de Comandos Útiles

Gestión de Volúmenes

Comando	Acción
docker volume create <nombre>	Crea un volumen nuevo.
docker volume ls	Lista todos los volúmenes.

docker volume inspect <nombre>	Muestra detalles (dónde está en el disco físico).
docker volume prune	**¡Cuidado!** Borra todos los volúmenes no usados.

Gestión de Redes

Comando	**Acción**
docker network create <nombre>	Crea una red tipo Bridge.
docker network ls	Lista las redes disponibles.
docker network connect <red> <cont>	Conecta un contenedor vivo a una red.
docker network inspect <red>	Ver qué contenedores están conectados y sus IPs.

4. Conclusión

Dominar Volúmenes y Redes es el paso que diferencia a un principiante de un usuario intermedio de Docker:

1. Usa **Volúmenes** para bases de datos y backups.
2. Usa **Bind Mounts** para desarrollar y editar código en caliente.
3. Crea **Redes Personalizadas** (User-defined bridges) para que tus contenedores se comuniquen por nombre (backend, db, redis) y no por IP.

En el siguiente nivel, veremos cómo orquestar todo esto de forma automática con **Docker Compose**, evitando tener que escribir estos comandos largos manualmente.