Práctica Guiada: El Pipeline de Datos (Redes y Volúmenes)

Introducción al Escenario

Como futuros especialistas en Big Data e IA, os encontraréis constantemente con la necesidad de mover datos de un sitio a otro. En esta práctica vamos a simular una arquitectura de microservicios muy común:

1. **Un Generador de Datos (Sensor):** Simulará un dispositivo IoT enviando lecturas de temperatura.
2. **Un Servidor Web (Dashboard):** Mostrará esos datos en tiempo real al usuario.
3. **Un Auditor (Monitor):** Comprobará internamente que el sistema funciona.

Objetivos Técnicos:

- Compartir información entre contenedores usando **Volúmenes**.
- Comunicar contenedores entre sí usando **Redes Bridge**.
- Entender la diferencia entre persistencia y conectividad.

Paso 1: Preparando el Terreno (La Red)

Primero, necesitamos una "carretera" privada para que nuestros contenedores se comuniquen de forma segura y por nombre (DNS), sin exponerse innecesariamente.

1. Crea una red personalizada tipo bridge:
   docker network create red_datos

2. Verifica que se ha creado:
   docker network ls

¿Por qué hacemos esto? Si usamos la red por defecto de Docker, no podemos llamar a los contenedores por su nombre (tendríamos que saber su IP, que cambia siempre). Al crear nuestra propia red, Docker activa el **DNS automático**.

Paso 2: El Almacenamiento Compartido (El Volumen)

Necesitamos un lugar neutral donde el **Sensor** deje los datos y el **Dashboard** los recoja. Si guardamos los datos *dentro* del sensor, el dashboard no podrá verlos.

1. Crea un volumen gestionado por Docker:
   docker volume create volumen_registros

2. Inspecciónalo para ver que es real (mira el campo Mountpoint):
   docker volume inspect volumen_registros

Paso 3: Desplegando el "Sensor" (Productor de Datos)

Vamos a usar una imagen ligera (alpine) para simular un sensor. Este contenedor escribirá la fecha y un dato aleatorio en un archivo de texto cada 5 segundos.
Ejecuta el siguiente comando (fíjate bien en el montaje del volumen -v):

```
docker run -d \
  --name sensor_iot \
  --network red_datos \
  -v volumen_registros:/datos \
  alpine \
  sh -c "while true; do echo 'Temperatura: '$(shuf -i 20-30 -n 1)'ºC - Hora: '$(date) >> /datos/lecturas.txt; sleep 5; done"
```

Análisis del comando:

- --network red_datos: Lo conectamos a nuestra red privada.
- -v volumen_registros:/datos: Enganchamos el volumen creado en el Paso 2 dentro de la carpeta /datos del contenedor.
- sh -c "...": Es un pequeño script en bucle infinito que escribe en /datos/lecturas.txt.

Prueba: Aunque el contenedor corre en segundo plano (-d), está generando datos ahora mismo.

Paso 4: Desplegando el "Dashboard" (Consumidor de Datos)

Ahora levantaremos un servidor web **Nginx**. Su función será leer ese mismo archivo de texto y mostrártelo en el navegador.
Para que esto funcione, debemos "engañar" a Nginx. Él espera servir archivos desde /usr/share/nginx/html. Nosotros le montaremos nuestro volumen justo ahí.

```
docker run -d \
  --name dashboard_web \
  --network red_datos \
  -p 8080:80 \
  -v volumen_registros:/usr/share/nginx/html \
  nginx
```

Análisis del comando:

- -p 8080:80: Abrimos el puerto 8080 de tu máquina para ver la web.
- -v volumen_registros:/usr...: **¡Truco!** Montamos el *mismo* volumen que usa el sensor. Lo que el sensor escribe, Nginx lo sirve.

¡Comprobación Visual!

Abre tu navegador web y ve a:

👉 http://localhost:8080/lecturas.txt
(Deberías ver cómo aparecen nuevas líneas cada 5 segundos si refrescas la página. ¡Acabas de comunicar dos contenedores vía storage!)

Paso 5: El Auditor (Comunicación por Red)

Ya hemos probado que comparten datos (Volumen). Ahora probemos que se ven por red. Vamos a lanzar un tercer contenedor interactivo para hacer de "técnico de mantenimiento".

1. Lanza un contenedor Alpine interactivo en la misma red:
   docker run -it --rm --name auditor --network red_datos alpine sh

2. Prueba de Ping (DNS):
   Dentro de la terminal del contenedor auditor, intenta contactar al sensor_iot y al dashboard_web por su nombre:
   ping -c 2 sensor_iot
   ping -c 2 dashboard_web

   Resultado: Verás que responde. El auditor "ve" a los otros máquinas.
3. Prueba de Servicio (Curl simulado):
   Vamos a intentar descargar los datos del dashboard desde dentro de la red interna, sin salir a internet ni usar localhost.
   (Instalamos curl primero porque alpine viene "pelado")
   apk add --no-cache curl
   curl http://dashboard_web/lecturas.txt

   Resultado: ¡Verás los datos! El auditor ha pedido la web al dashboard_web usando la red interna de Docker.
4. Escribe exit para salir del auditor.

Paso 6: Persistencia (La prueba final)

¿Qué pasa si el contenedor del sensor se rompe? ¿Perdemos los datos históricos?

1. Borra el sensor (sin piedad):
   docker rm -f sensor_iot

2. Vuelve a mirar el navegador (localhost:8080/lecturas.txt).
   - **¿Se han borrado los datos?** No. Siguen ahí. El contenedor ha muerto, pero el **Volumen** sobrevive.
   - Obviamente, ya no se generan líneas nuevas, pero el histórico está a salvo.

Paso 7: Limpieza del Entorno

Como buenos profesionales, debemos dejar el entorno limpio al terminar.

1. Borrar los contenedores:
   docker rm -f dashboard_web

(El sensor ya lo borramos antes, y el auditor se borró solo al salir gracias al flag --rm)

2. Borrar la red:
   docker network rm red_datos

3. Borrar el volumen (¡Ahora sí borramos los datos definitivamente!):
   docker volume rm volumen_registros

Resumen de Conceptos Aprendidos

Concepto	**En esta práctica...**
Volumen (volumen_registros)	Actuó como una "carpeta compartida en red" o un disco duro externo USB que pinchamos en dos ordenadores a la vez.
Red Bridge (red_datos)	Actuó como el router de una oficina (LAN). Permitió que el auditor llamara al dashboard_web por su nombre.
Persistencia	Demostramos que al matar al sensor_iot, los datos (lecturas.txt) seguían vivos en el volumen.