Práctica: Arquitectura web con Docker usando volumes e bind mounts

Obxectivos

- Construír e despregar unha arquitectura web básica (frontend + backend + base de datos) con Docker e Docker Compose.
- Usar **bind mounts** para cargar código fonte directamente desde o host.
- Empregar un **volume de Docker** para garantir a persistencia dos datos da base de datos.
- Practicar o control de versións con **Git e GitHub**.

Contido inicial

Ao descomprimir o ZIP recibirás:

- Un directorio `frontend/` cunha páxina `index.html`.
- Un directorio `backend/` cun ficheiro Python (`app.py`), un `requirements.txt` e un `Dockerfile`.
- Un directorio `db/` cun script SQL de inicialización (`init.sql`).
- Un `README.md` de referencia.

Parte 1: Git e GitHub

1. Inicializa un repositorio Git local (`git init`).
2. Crea un repositorio remoto en GitHub e vincúlao.
3. Crea ramas separadas (`frontend`, `backend`) para traballar de forma organizada.
4. Fai `commit` dos cambios e realiza `merge` cara á rama principal.
5. Sube o proxecto completo a GitHub.

Parte 2: Estrutura do despregue

Frontend

- Non require un Dockerfile propio.
- Usarás directamente a imaxe `nginx:alpine`.
- Servirá a páxina `index.html` mediante un **bind mount** dende o host.

Backend

- Usa un `Dockerfile` baseado en `debian:bullseye`.
- Instala Python 3 e pip.
- Instala as dependencias de `requirements.txt`.
- Lanza a aplicación `app.py`.
- O código da app será cargado dinámicamente cun **bind mount**, sen recompilar a imaxe tras cada cambio.

Base de datos

- Usa a imaxe `postgres:15`.
- Executa un script SQL de inicialización.
- Usa un **volume** para conservar os datos entre sesións.

Parte 3: docker-compose.yml

Crea un ficheiro `docker-compose.yml` que:

- Defina os tres servizos (`frontend`, `backend`, `db`).
- Use bind mounts para montar os directorios `./frontend` e `./backend` nos contedores.
- Use un volume chamado `db_data` para `/var/lib/postgresql/data`.

Parte 4: Execución e probas

1. Lanza os servizos:

   ```
   docker-compose up
   ```

2. Accede no navegador:

   - `http://localhost:8080` → frontend.
   - `http://localhost:5000` → backend (resposta tipo "API operativa...").

3. Apaga os contedores:

   ```
   docker-compose down
   ```

4. Verifica que os datos da base de datos persisten tras reiniciar os servizos.

Parte 5: Entregable

1. Subir a GitHub:

   - `docker-compose.yml`
   - Dockerfile do backend
   - Código HTML e Python
   - Script SQL
   - README con instrucións

2. Incluír no README:

   - Breve descrición do proxecto
   - Composición dos servizos

    - Comandos principais para executar o proxecto

3. Entregar un arquivo de texto coa URL do repositorio en *GitHub*

Recomendacións

- Fai `commit` frecuentes e con mensaxes claros.
- Usa ramas e merges de maneira organizada.
- Fai push de cada rama a GitHub

Rúbrica de avaliación da práctica: Docker + Git + Compose

Cada criterio puntúase sobre 2 puntos, podendo acadar un máximo de 6 puntos totais.

Criterio	Baixo (0,75 puntos)	Medio (1,5 puntos)	Alto (2 puntos)
Uso de Git e GitHub	O repositorio é incompleto ou con poucos `commits`. Sen ramas nin boas prácticas.	Inclúe `commits` e estrutura razoable. Uso básico de ramas ou mensaxes de `commit` claras.	Uso correcto de ramas, `commits` frecuentes e significativos, estrutura clara, repo publicado en GitHub.
Creación de imaxes Docker	A imaxe funciona con erros ou mal estruturada. Inclúe paquetes innecesarios ou malas prácticas básicas.	A imaxe funciona ben pero con algúns detalles mellorables: limpeza, separación de capas, permisos, etc.	Imaxe funcional, clara, **segura e optimizada**: imaxe base adecuada, non executa como root, instala só o necesario, sen secrets nin lixo.
Composición con Docker Compose	O ficheiro `docker-compose.yml` é incompleto ou contén erros ainda que despregа os servizos requeridos.	Composición funcional pero con limitacións (falta de volumes, dependencias mal definidas...).	Composición completa, funcional, con `bind mounts`, `volumes` e relacións ben establecidas entre servizos.