Práctica: Contenedorización dunha Aplicación Spring Boot

Introdución

Nesta práctica partirás dunha aplicación web de xestión de tarefas desenvolvida con **Spring Boot** que usa **H2** como base de datos en ficheiro. O obxectivo é migrar esta aplicación para que funcione nun entorno **contenedorizado con Docker**, utilizando **PostgreSQL** como base de datos e **Docker Compose** para orquestrar os servizos.

Forma de Entrega

1. Fai un **fork** deste repositorio na túa conta de GitHub
2. Engade o usuario `mrey-profe` como **colaborador** do teu fork:
   - Vai a `Settings → Collaborators → Add people → mrey-profe`
3. Traballa no teu fork realizando as modificacións indicadas en cada fase
4. Escribe o **enlace ao teu repositorio** na tarefa correspondente de **Moodle**

Importante: Non subas o arquivo `.env` con credenciais reais ao repositorio. Asegúrate de que está no `.gitignore`.

Fase 1 — Exploración da Aplicación (sen Docker)

Obxectivo: Coñecer a aplicación antes de modificala.

Tarefas

1. Proba a aplicación localmente para comprobar que funciona con H2.

Fase 2 — Migración a PostgreSQL

Obxectivo: Cambiar a base de datos de H2 a PostgreSQL para preparar a aplicación para un entorno real.

Tarefas

1. **Modifica o `pom.xml`:** Necesitas substituír a dependencia de H2 pola de PostgreSQL. Busca a dependencia adecuada en Maven Repository.

2. **Modifica o `application.properties`:** Adapta a configuración da base de datos para conectar con PostgreSQL:

   - A URL de conexión segue o formato: `jdbc:postgresql://host:porto/nome_bd`
   - PostgreSQL require un driver específico: `org.postgresql.Driver`
   - Hibernate ten un dialecto específico para PostgreSQL
   - Usa variables de entorno con valores por defecto para as credenciais, co formato de Spring Boot: `${NOME_VARIABLE:valor_por_defecto}`

3. **Elimina a configuración da consola H2** (xa non é necesaria)

Pistas

- O driver de PostgreSQL para Maven ten `groupId` = `org.postgresql` e `artifactId` = `postgresql`
- O dialecto de Hibernate para PostgreSQL é `org.hibernate.dialect.PostgreSQLDialect`
- As variables de entorno permítenche configurar credenciais sen hardcodealas no código

Nota: Neste punto a aplicación non funcionará localmente a non ser que teñas PostgreSQL instalado. Iso resolverémolo na seguinte fase con Docker.

Fase 3 — Dockerfile (Compilación Multi-Stage)

Obxectivo: Crear un Dockerfile que compile e execute a aplicación Spring Boot.

Tarefas

1. **Crea un `Dockerfile`** na raíz do proxecto con dúas etapas:

   Etapa 1 — Compilación (Builder):

   - Usa unha imaxe base con Maven e Java 21 (suxestión: `maven:3.9-eclipse-temurin-21`)
   - Copia primeiro o `pom.xml` e descarga as dependencias (aproveita a **caché de capas de Docker**)
   - Despois copia o código fonte e compila o proxecto (sen executar tests)

   Etapa 2 — Execución (Runtime):

   - Usa unha imaxe lixeira só con JRE (suxestión: `eclipse-temurin:21-jre-alpine`)
   - Copia o JAR xerado na primeira etapa
   - Expon o porto da aplicación
   - Define o comando de inicio

Pistas

- A compilación Maven xera o JAR en `target/*.jar`
- `mvn dependency:go-offline` descarga as dependencias sen compilar (útil para caché)
- `mvn clean package -DskipTests` compila sen executar tests

Fase 4 — Docker Compose e Variables de Entorno

Obxectivo: Orquestrar a aplicación e a base de datos con Docker Compose, xestionando credenciais de forma segura.

Tarefas

1. **Crea un arquivo `.env`** na raíz do proxecto coas variables sensibles:

   - Credenciais de PostgreSQL (usuario, contrasinal, nome da base de datos)
   - Portos dos servizos

2. **Crea un arquivo** `.env.example` co mesmo formato pero con valores de exemplo (sen credenciais reais). Este arquivo **si** se sube ao repositorio como referencia.

3. **Crea un arquivo** `docker-compose.yml` con dous servizos:

   Servizo `db` **(PostgreSQL):**

   - Usa a imaxe `postgres:15-alpine`
   - Configura as credenciais usando as variables do `.env`
   - Expon o porto de PostgreSQL ao host
   - Crea un **volume** para persistir os datos
   - Engade un **healthcheck** con `pg_isready`

   Servizo `app` **(Spring Boot):**

   - Constrúe desde o `Dockerfile`
   - Depende do servizo `db` (só arranca cando a BD estea saudable)
   - Pasa as variables de entorno necesarias para que Spring Boot se conecte
   - Expón o porto da aplicación
   - Configura reinicio automático

4. **Configura a rede:** Ambos servizos deben estar nunha rede interna de Docker

5. **Actualiza o** `.gitignore` para excluír:

   - O arquivo `.env` (credenciais sensibles)
   - O directorio de datos de H2 (xa non é necesario)

Pistas

- Docker Compose le automaticamente o arquivo `.env` da mesma carpeta
- As variables no `.env` referenciaranse con `${NOME_VARIABLE}` no `docker-compose.yml`
- O servizo `db` usa como nome de host interno o nome do servizo no Compose
- `pg_isready` é un comando de PostgreSQL para verificar se a BD está lista
- A condición `service_healthy` en `depends_on` asegura que a BD estea lista antes de arrancar a app

---

Fase 4 - Migración a MySQL (opcional)

Repite os pasos anteriores para que a aplicación funcione con mySQL nunha rama diferente do teu repositorio.