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.

Obxectivos de Aprendizaxe

- Comprender as diferenzas entre unha base de datos embebida (H2) e unha base de datos cliente-servidor (PostgreSQL)
- Crear un **Dockerfile** con compilación multi-stage
- Orquestrar múltiples contenedores con **Docker Compose**
- Xestionar credenciais sensibles con arquivos `.env`
- Aplicar boas prácticas de seguridade e configuración en contornos Docker

📦 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. Clona o teu fork e executa a aplicación localmente:

   ```
   mvn spring-boot:run
   ```

2. Accede á aplicación web en http://localhost:8080 e crea algunhas tarefas

3. Accede á **consola H2** en http://localhost:8080/h2-console:

   - JDBC URL: `jdbc:h2:file:./data/tareasdb`

    - Usuario: `sa`
    - Contrasinal: *(baleiro)*
    - Executa: `SELECT * FROM TAREAS;`

4. Observa o directorio `data/` que se creou no proxecto. Estes son os ficheiros da base de datos H2
5. Detén a aplicación, vólvea executar e verifica que os datos persisten
6. Explora a **documentación Swagger** en http://localhost:8080/swagger-ui.html

Preguntas para reflexionar

- Onde se almacenan fisicamente os datos?
- Que pasaría se borrases o directorio `data/`?
- Cales son as limitacións dunha base de datos embebida como H2 para un entorno de produción?

🔄 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

2. **Aplica boas prácticas de seguridade:**

   - Crea un usuario non-root para executar a aplicación
   - Engade un **healthcheck** que verifique o endpoint `/actuator/health`

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
- En Alpine Linux, os comandos para crear usuarios son `addgroup -S` e `adduser -S`
- O healthcheck pode usar `wget --spider` para verificar un URL

Recursos

- Documentación oficial de Dockerfile
- Spring Boot Docker Guide
- Dockerfile Best Practices

🎼 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
- Expon 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

Recursos

- Documentación de Docker Compose
- Imaxe oficial de PostgreSQL en Docker Hub
- Variables de entorno en Docker Compose

📝 Verificación Final

Cando remates todas as fases, verifica que todo funciona correctamente:

```
# 1. Copiar o arquivo de exemplo de variables de entorno
cp .env.example .env

# 2. Editar o .env cos valores desexados
nano .env

# 3. Construír e executar os contenedores
docker-compose up -d --build
```

```
# 4. Verificar que os contenedores están en execución
docker-compose ps

# 5. Acceder á aplicación
# Abrir http://localhost:8080 no navegador

# 6. Verificar o healthcheck
curl http://localhost:8080/actuator/health

# 7. Crear algunhas tarefas e verificar que persisten após reiniciar
docker-compose down
docker-compose up -d
# As tarefas anteriores deben seguir aí
```

📊 Criterios de Avaliación

Criterio	Puntuación
Fase 2 — Migración a PostgreSQL	
Dependencia PostgreSQL correcta no `pom.xml`	1 punto
Configuración de `application.properties` con variables de entorno	1 punto
Fase 3 — Dockerfile	
Dockerfile multi-stage funcional (builder + runtime)	1,5 puntos
Caché de dependencias Maven (copiar `pom.xml` antes que `src/`)	0,5 puntos
Usuario non-root	0,5 puntos
Healthcheck configurado	0,5 puntos
Fase 4 — Docker Compose e `.env`	
`docker-compose.yml` con dous servizos funcionais	1,5 puntos
Volume para persistencia de datos de PostgreSQL	0,5 puntos
Healthcheck de PostgreSQL e `depends_on` con condición	0,5 puntos
Rede interna configurada	0,5 puntos
Arquivo `.env.example` incluído e `.env` excluído en `.gitignore`	0,5 puntos
Reinicio automático configurado	0,5 puntos
Total	**9 puntos**

> O punto restante ata o 10 valorarase pola calidade xeral do traballo: orde, limpeza do código, comentarios explicativos e correcto funcionamento de todo o conxunto.

📚 Recursos Xerais

- Documentación Oficial de Spring Boot
- Guía de Docker
- Referencia de Docker Compose
- PostgreSQL Docker Hub
- Spring Boot con Docker (guía oficial)
- Variables de Entorno en Spring Boot