Runbook Diario — Bajar/Subir y Levantar AI-Service (DEV con JWT + Swagger)

Objetivo: que cada vez que bajes el repo y lo vuelvas a levantar (o cambies de PC/branch), sigas un procedimiento **repetible**, **sin sorpresas**, con **DEV 100% JWT** , Swagger con **Authorize**, y checks claros para diagnosticar fallas.

---

0) Pre-requisitos (una sola vez por máquina)

0.1 Java / Maven Wrapper

- **Java 21** instalado (JDK). Verifica:
  - `java -version`
- El proyecto usa **Maven Wrapper** (`mvnw.cmd`). No dependas de Maven global.

0.2 Puertos estándar del entorno

- AI-Service **DEV**: `8091` (recomendado)
- PostgreSQL local (si usas docker/instancia aparte): por tu config venías usando `5433`
- IAM (issuer): típicamente `9000` u otro (según tu `iam-service`)

Regla enterprise: **cada servicio en DEV tiene su puerto fijo**, y se documenta.

0.3 Docker (si tu Postgres/IAM corren en contenedores)

- Tener Docker Desktop funcionando.
- Verificar:
  - `docker version`
  - `docker compose version`

---

1) Rutina diaria: “Bajar / actualizar proyecto” (sin romper nada)

1.1 Limpieza de estado previo (solo si vienes de otra sesión)

1) Cierra el proceso del servicio si estaba corriendo.
2) Revisa si quedó colgado un Java usando el puerto:
   - **PowerShell**:
     - `netstat -ano | findstr :8091`
     - Si aparece un PID, puedes matarlo:
       - `taskkill /PID <PID> /F`

Esto evita el clásico: “Port already in use”.

1.2 Actualizar repo

En la carpeta del repo: - `git status` - Si tienes cambios locales: - O los commiteas, o los guardas: - `git stash -u` - Trae cambios: - `git pull`

Regla enterprise: antes de levantar, **siempre** confirmar estado limpio o stash.

1.3 Verificar branch y versiones

- `git branch`
- `git log -1 --oneline`

---

2) Levantar dependencias (DB + IAM) — orden correcto

2.1 PostgreSQL

Opción A: Postgres con Docker - `docker compose up -d postgres` *(si tienes un compose)* - Verifica: - `docker ps` - (opcional) `psql` o un cliente para confirmar conexión.

Opción B: Postgres instalado local - Asegúrate que el servicio está iniciado. - Confirma puerto: - `netstat -ano | findstr :5433`

2.2 IAM-Service (issuer JWT)

- Levanta `iam-service` primero (porque AI valida tokens contra el issuer).
- Verifica que el issuer responde:
    - Si tu IAM expone OpenID:
        - `http://localhost:<IAM_PORT>/.well-known/openid-configuration`

  Regla enterprise: AI-Service no debe "adivinar" issuer; **debe existir**.

---

3) Configuración DEV (archivos que siempre deben estar OK)

3.1 Ubicación de configuraciones

En tu módulo **ai-bootstrap**: - `modules/ai-bootstrap/src/main/resources/` - `application.yml` (base) - `application-dev.yml` (DEV)

3.2 `application-dev.yml` — checklist mínimo

Debe incluir (conceptual): - `server.port: 8091` - `spring.datasource.*` apuntando a tu Postgres - **JWT Resource Server**: - `spring.security.oauth2.resourceserver.jwt.issuer-uri: http://localhost:<IAM_PORT>/realms/<realm>` *(ejemplo)* - `springdoc.swagger-ui.*` (si usas propiedades)

Clave: si el `issuer-uri` está mal, Swagger te deja "Authorize" pero el backend rechazará tokens.

3.3 Evitar duplicados YAML

No debes tener dos bloques `server:` duplicados en el mismo archivo. - Solo uno: - `server: port: 8091`

---

4) Build correcto (como CI local)

Desde la raíz del repo:

4.1 Build reactor (rápido y confiable)

```
cd C:\ProyectoAI\ai-service
.\mvnw.cmd -DskipTests clean install
```

> Esto valida compilación y empaquetado de todos los módulos.

4.2 Si quieres forzar dependencias del módulo al correr

Usa `-am` (also-make) cuando corras boot:

```
.\mvnw.cmd -pl modules/ai-bootstrap -am spring-boot:run "-Dspring-boot.run.arguments=--spring.profiles.active=dev"
```

---

5) Run correcto (DEV) — el comando estándar

Comando estándar recomendado:

```
cd C:\ProyectoAI\ai-service
.\mvnw.cmd -pl modules/ai-bootstrap -am spring-boot:run "-Dspring-boot.run.arguments=--spring.profiles.active=dev"
```

5.1 ¿Por qué “no vuelve al prompt”?

Porque `spring-boot:run` mantiene el proceso vivo (es el servidor). En enterprise: - Se deja corriendo en esa terminal. - Si quieres recuperar el prompt, abres otra terminal o lo levantas como servicio.

Para detenerlo: - `CTRL + C`

---

6) Verificación rápida (smoke tests)

6.1 Health

- `http://localhost:8091/actuator/health` Debe dar **UP**.

6.2 Swagger

- `http://localhost:8091/swagger-ui/index.html`

Si te sale pop-up (básico) **NO está bien configurado JWT**.

6.3 Endpoint de negocio

Tu endpoint: - `POST http://localhost:8091/api/v1/ai/complete`

> En Swagger: busca el controller que lo expone y prueba desde ahí.

---

7) Flujo enterprise: obtener token y usar Swagger Authorize

7.1 Obtener token desde IAM

Depende de tu IAM (Keycloak u otro). Conceptualmente: 1) Login / client credentials 2) Recibir `access_token` (JWT)

7.2 Authorize en Swagger

1) Abre Swagger UI.
2) Click **Authorize**.
3) Pega:
    - `Bearer <TU_TOKEN>`
4) Authorize.

7.3 Probar request

En el endpoint `POST /api/v1/ai/complete` envía Body JSON:

```
{
  "prompt": "Escribe un resumen de 3 líneas sobre SolverIA."
}
```

---

8) Diagnóstico rápido de errores típicos

8.1 “Port already in use”

- Ver quién ocupa el puerto:
    - `netstat -ano | findstr :8091`
- Matar proceso:
    - `taskkill /PID <PID> /F`

8.2 Swagger pide “iniciar sesión” (pop-up)

Esto indica que tu seguridad sigue en modo **httpBasic** o que Swagger no está usando Bearer. Checklist: - En `SecurityConfig`: - No debe estar `.httpBasic()` en DEV. - Debe estar `oauth2ResourceServer().jwt()`. - Debe existir configuración `issuer-uri`. - Debes tener OpenAPI security scheme Bearer.

8.3 401 aunque pusiste Bearer

- Token vencido
- `issuer-uri` incorrecto (issuer mismatch)
- Audience/roles no coincide con tu configuración

8.4 DB DOWN

- Revisa `spring.datasource.url`
- Revisa puerto 5433
- Revisa contenedor/servicio

---

9) Rutina “Subir cambios” (push) — sin romper main

1) `git status`
2) Corre build local:
   - `.\mvnw.cmd -DskipTests clean install`
3) (opcional) corre tests:
   - `.\mvnw.cmd test`
4) Commit con mensaje claro:
   - `git add .`
   - `git commit -m "fix(ai-dev): jwt swagger authorize runbook"`
5) Push:
   - `git push`

Enterprise: “si no compila local, no se sube”.

---

10) Estándar recomendado de operación diaria (resumen ejecutivo)

Cada día: 1) `git pull` 2) Asegurar DB + IAM arriba 3) `clean install` 4) `spring-boot:run` con profile dev 5) `actuator/health` 6) Swagger + Authorize con Bearer 7) Probar `/api/v1/ai/complete`

---

11) Nota importante (para que nunca vuelva el caos)

- Documenta **puertos** y **URLs** en un `RUNBOOK.md` en el repo.
- Deja un `Makefile` (Linux/mac) y `scripts/*.ps1` (Windows) para automatizar:
  - levantar dependencias
  - build
  - run
  - health checks