

API Spring Boot con GitHub Actions y AWS Elastic Beanstalk

Solución de la Actividad 2 - Explicación Técnica del Flujo CI/CD

1. Funcionamiento del Pipeline

El proyecto utiliza GitHub Actions como motor de automatización, definido mediante archivos YAML en `.github/workflows/`. Se han implementado dos flujos diferenciados:

- **CI (Integración Continua):** Se dispara con cada push o pull_request. Se encarga de clonar el código en un contenedor Ubuntu, configurar el JDK 17 de Amazon Corretto y ejecutar los tests unitarios mediante el Maven Wrapper (`./mvnw test`). Su objetivo es garantizar que ningún código con errores de compilación o lógica sea aceptado.
- **CD (Despliegue Continuo):** Una vez que el código es válido, este pipeline empaqueta la aplicación en un archivo JAR ejecutable, genera un paquete de despliegue comprimido (ZIP) junto con un archivo Procfile y lo envía a AWS Elastic Beanstalk utilizando las credenciales de AWS Academy almacenadas de forma segura en los GitHub Secrets.

2. Descripción del Flujo CI/CD

El flujo sigue una arquitectura de "entrega garantizada":

1. Desarrollo Local:

El código se escribe y valida en IntelliJ. Se utiliza el Maven Wrapper para asegurar la paridad de versiones de Maven entre el local y la nube.

2. (Trigger):

Al realizar un git push origin main, GitHub detecta los cambios y activa los workflows.

3. Construcción y Artefacto:

El servidor de GitHub genera el archivo binario (.jar) y lo renombra a app.jar para que coincida con la instrucción del Procfile.

4. Despliegue:

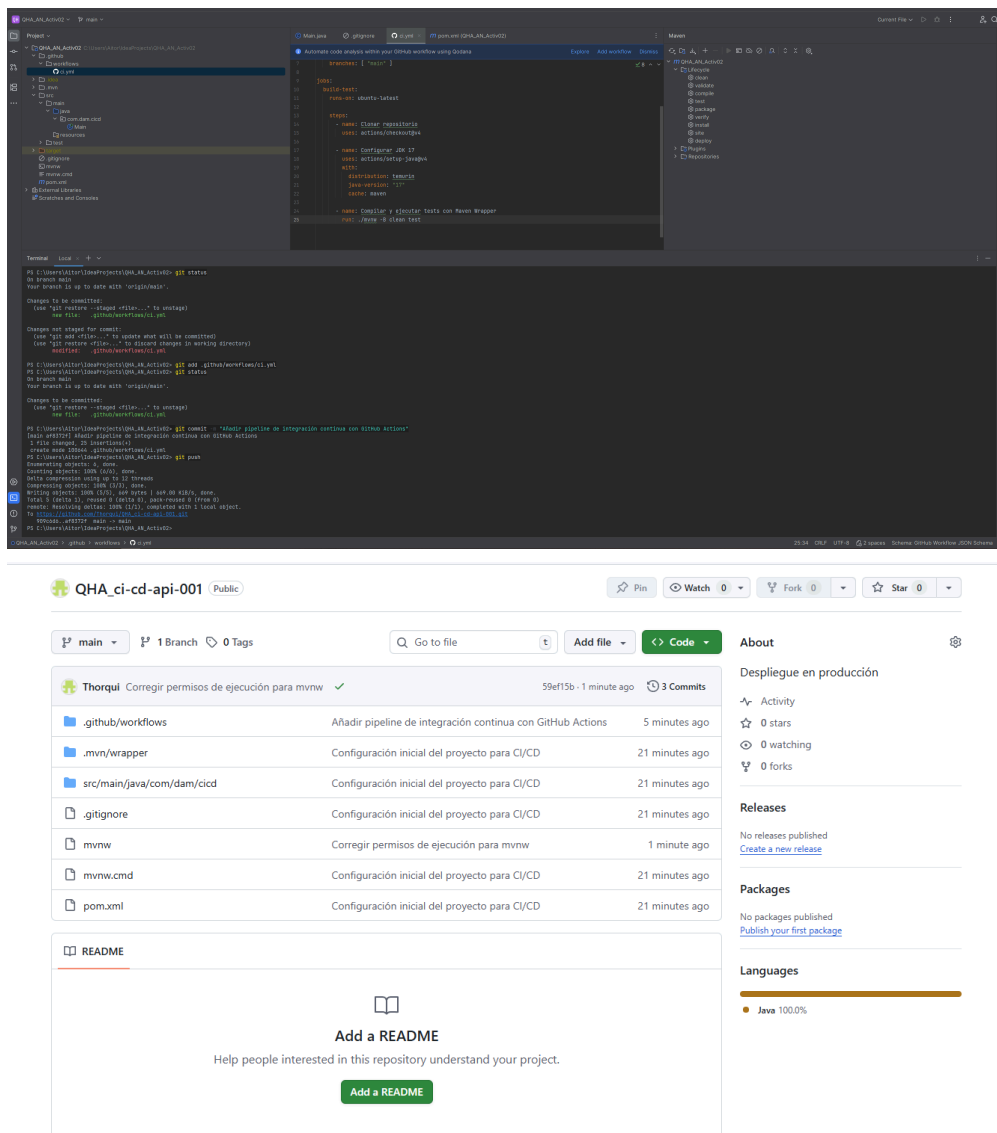
Se utiliza el servicio de AWS Elastic Beanstalk (plataforma Java SE) que recibe el artefacto y actualiza la instancia EC2 sin intervención manual, exponiendo la API en una URL pública. e informe final Excel

3. Decisiones Tomadas y Resolución de Problemas

- Adaptación a Windows: Al desarrollar en Windows y desplegar en Linux (AWS), se tomó la decisión de utilizar comandos de PowerShell para la creación de artefactos locales (Compress-Archive) y de gestionar los permisos de ejecución de archivos mediante Git (chmod +x mvnw) para evitar el error de permisos 126 en el pipeline.
- Seguridad y Credenciales: Se decidió utilizar GitHub Secrets para manejar el AWS_SESSION_TOKEN, una medida obligatoria al trabajar con entornos de AWS Academy/Learner Lab, asegurando que las claves no queden expuestas en el código fuente.
- Compatibilidad de Puerto: Se configuró server.port=\${PORT:8080} en las propiedades de Spring Boot. Esta decisión técnica permite que la aplicación funcione en el puerto 8080 localmente pero se adapte dinámicamente al puerto que AWS asigne en producción.

4. Checklist de Entrega:

URL GitHub: https://github.com/Thorqui/QHA_ci-cd-api-001.git



Captura del pipeline en estado Success en GitHub Actions

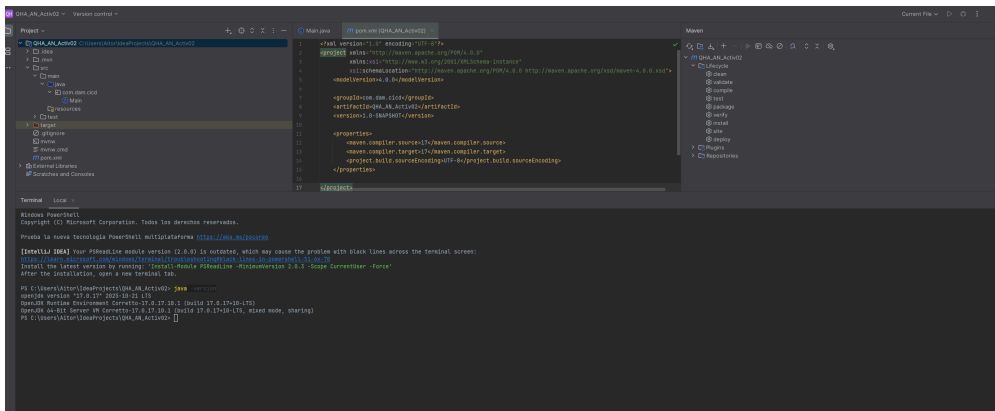
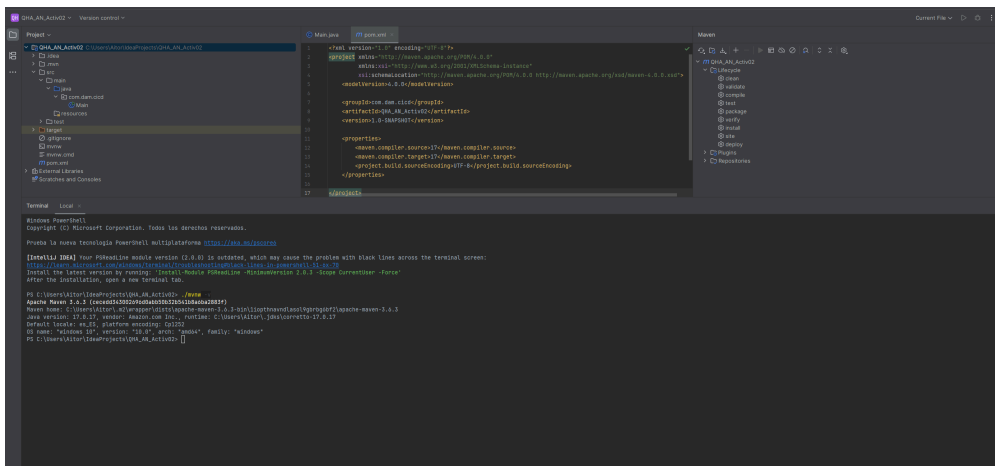
```
mensaje: "API operativa",
estado: "OK",
servicio: "dam-ci-cd-api-001"
```

Capturas totales realizadas del proceso por fases:

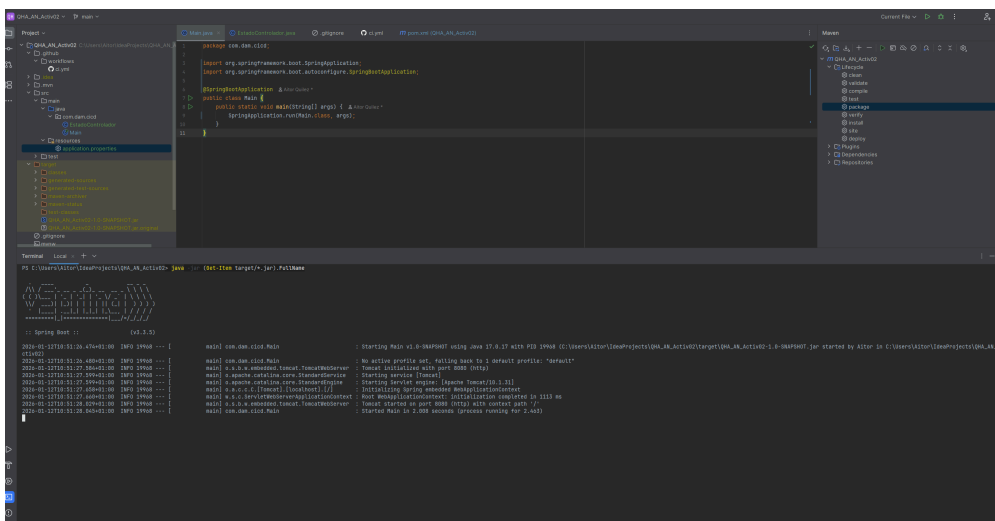
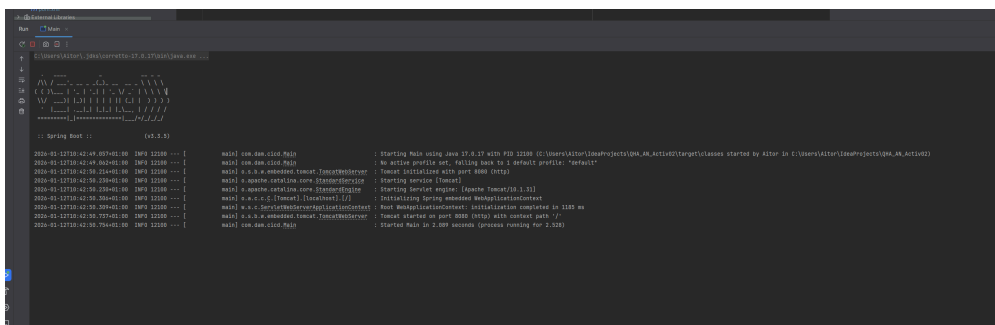
Fase 1: Desarrollo en Local y Entorno

Preparación del entorno de construcción reproducible y validación de la API en local.

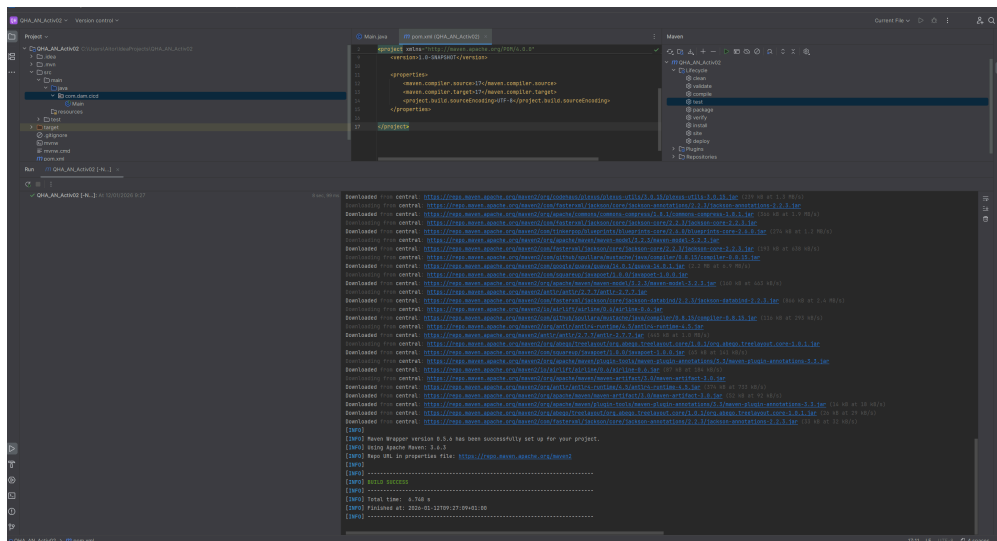




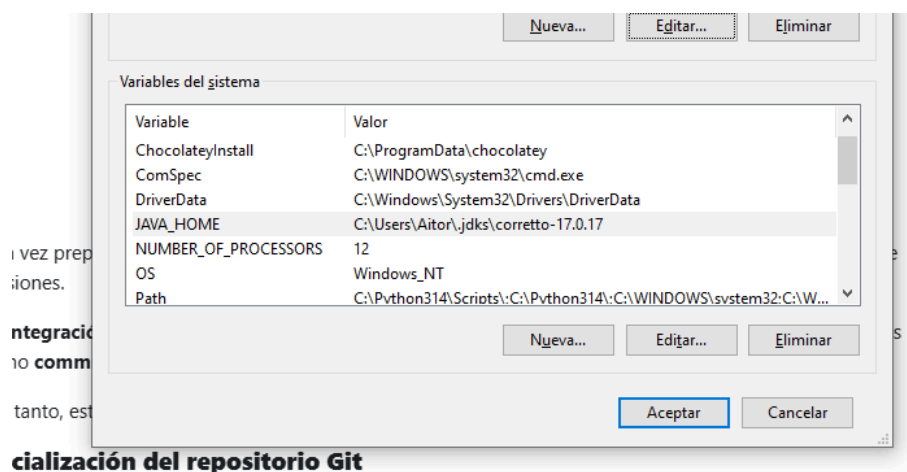
Verificación de versiones: Java 17, Maven y OpenJDK 17



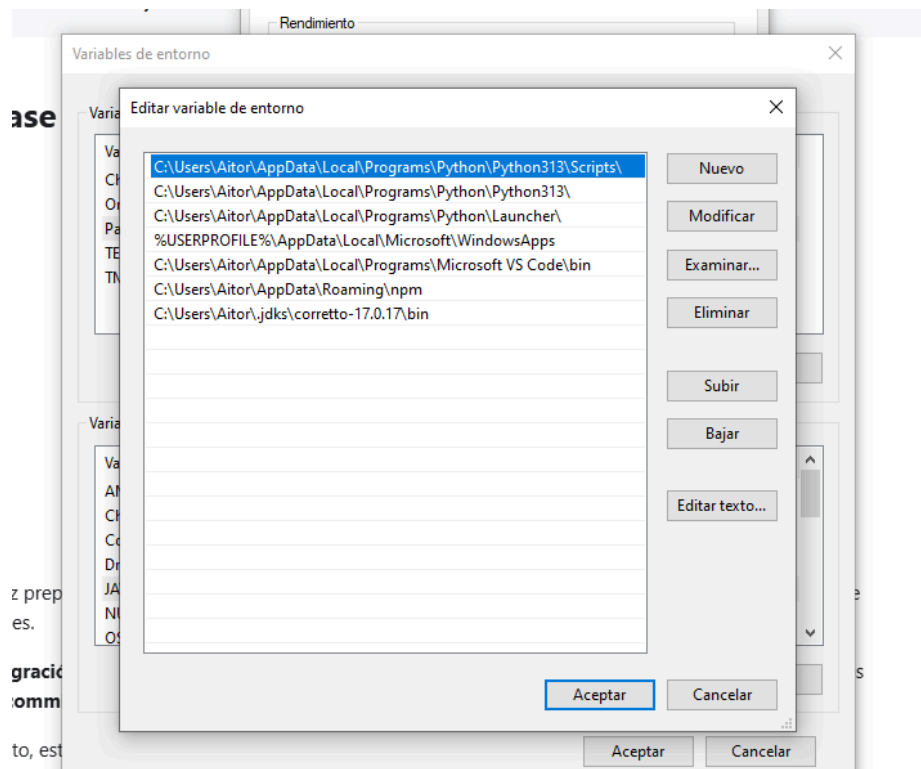
Ejecución y validación de la aplicación Spring Boot en el entorno local



Resultado satisfactorio del empaquetado (Build Success)



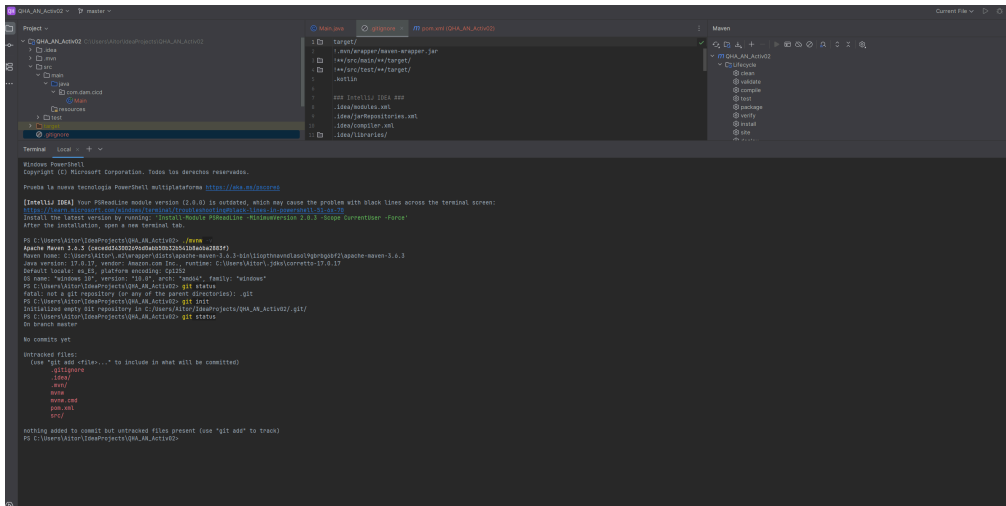
Configuración de variables de entorno



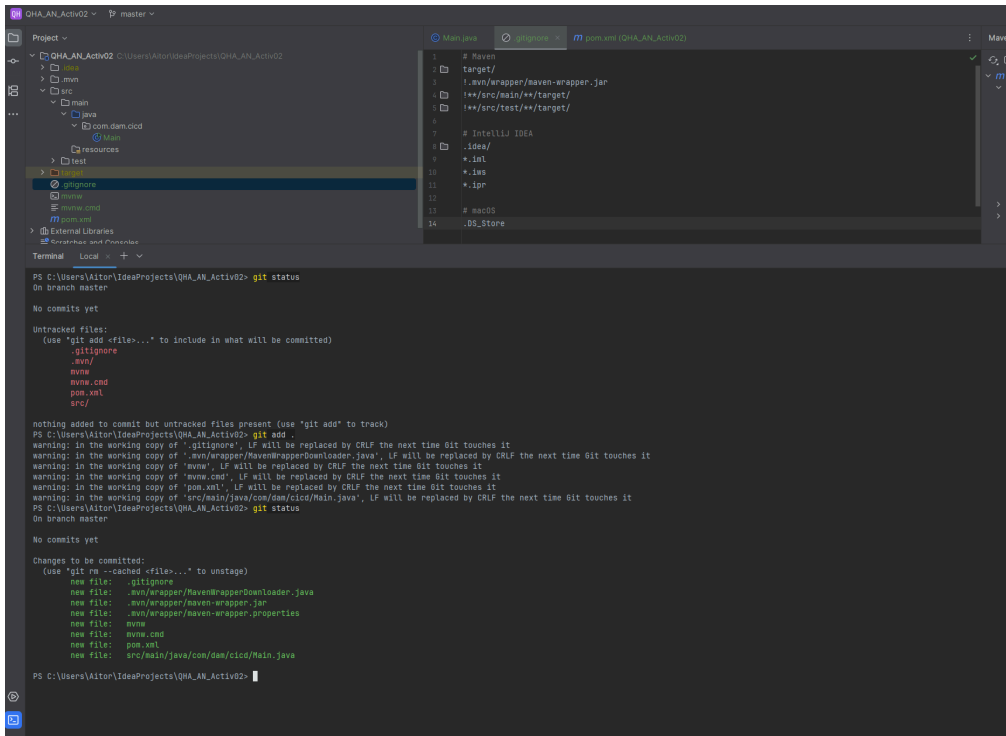
Resolución de problemas en Windows: Configuración de variables de entorno y JAVA_HOME

Fase 2: Control de Versiones (Git)

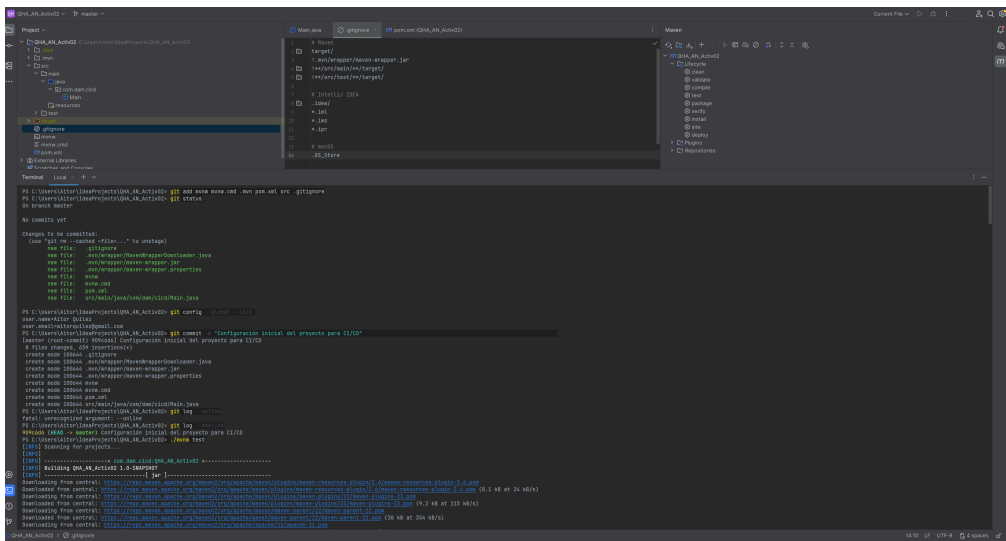
Inicialización del repositorio y gestión de estados de los archivos.



Inicialización del repositorio Git local



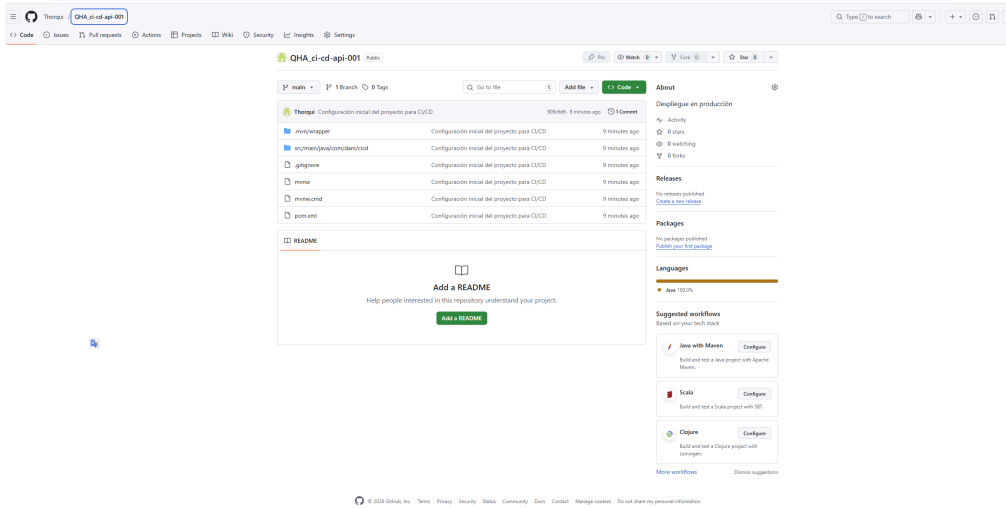
Preparación de archivos (Staging) para el primer commit



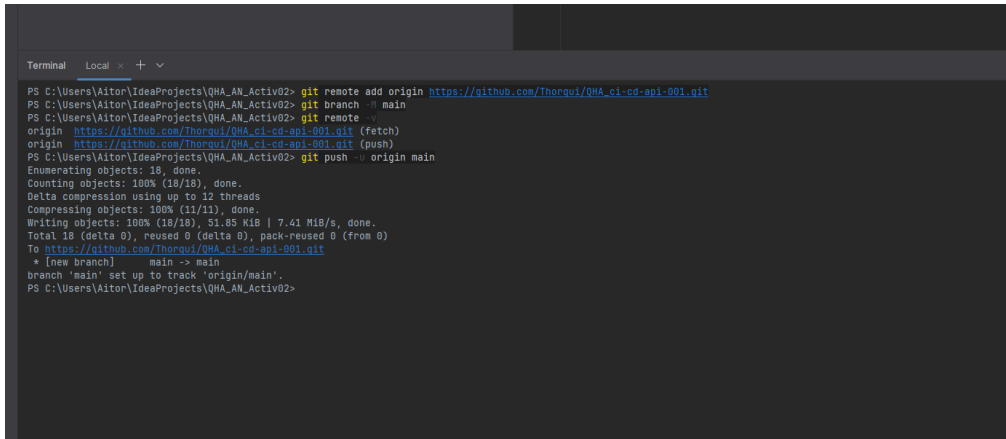
Seguimiento de la estructura de archivos en el control de versiones

Fase 3: Conexión con GitHub

Configuración del repositorio remoto y sincronización de código.



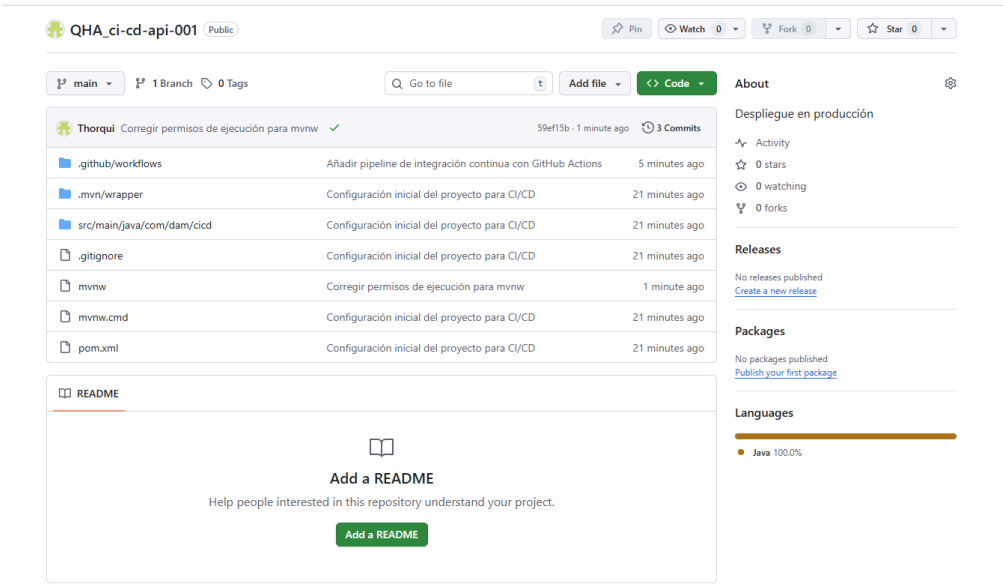
Creación del repositorio vacío en la plataforma GitHub



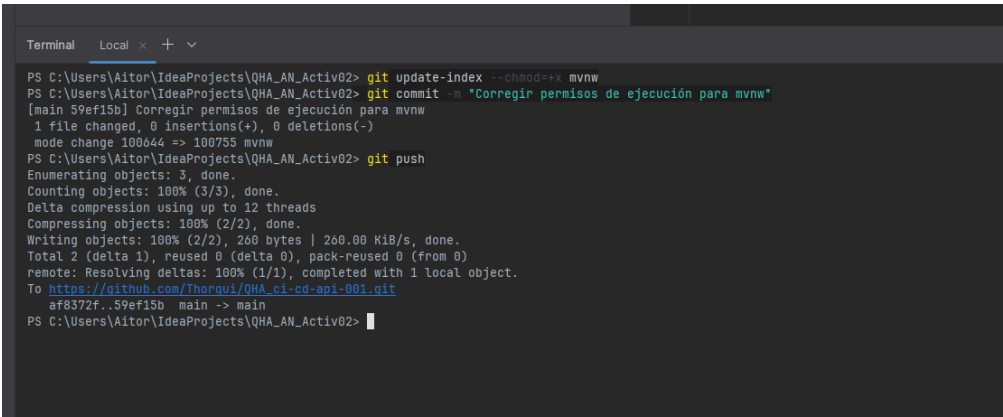
Vinculación del repositorio local con el origen remoto

Fase 4: Integración Continua (CI) y Correcciones

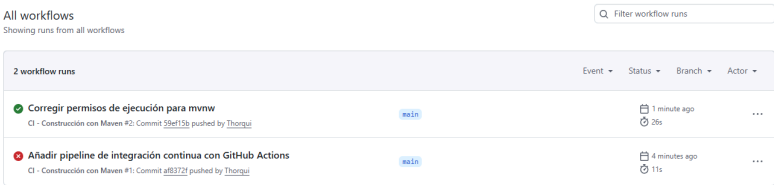
Automatización de pruebas y resolución de incompatibilidades Windows-Linux.



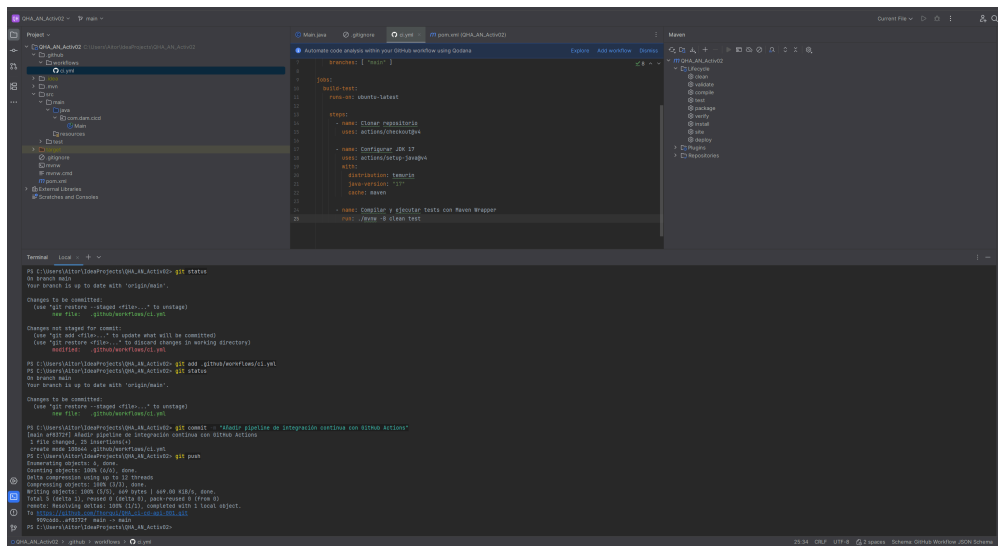
Subida de código mediante Personal Access Token



Corrección de permisos de ejecución para mvnw mediante Git update-index



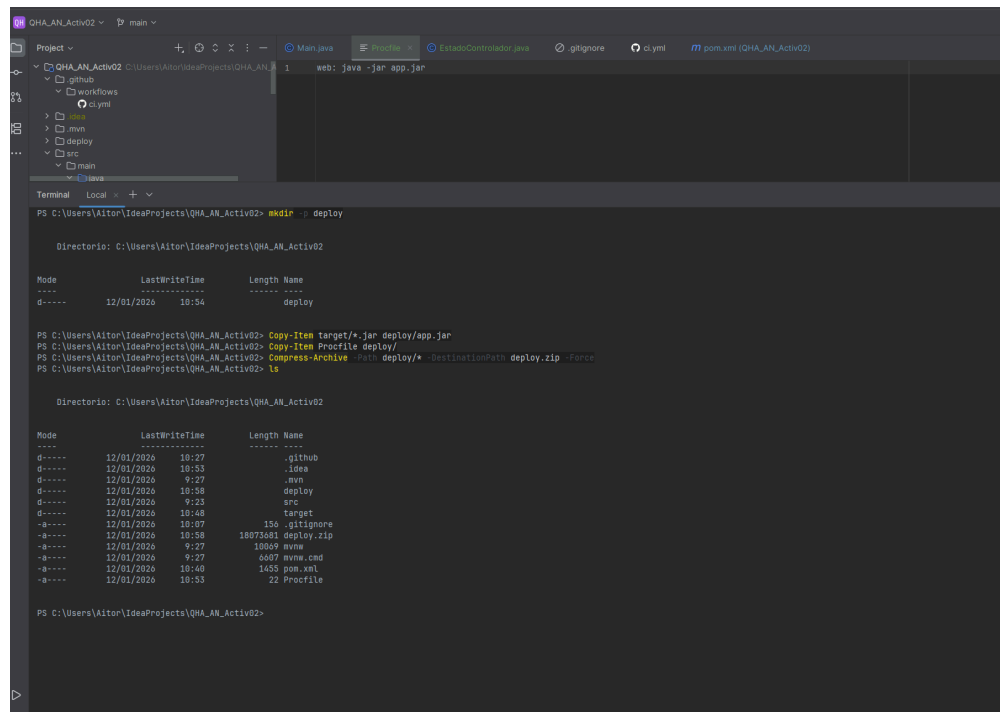
Ajuste de flujos de trabajo tras detectar fallos en el pipeline



Pipeline de GitHub Actions en estado Success

Fase 5: Despliegue Continuo (CD) y AWS

Puesta en producción de la API en AWS Elastic Beanstalk.



The screenshot shows an IDE with a terminal window. The terminal displays the following commands and output:

```
PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02> mkdir deploy
Directorio: C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02

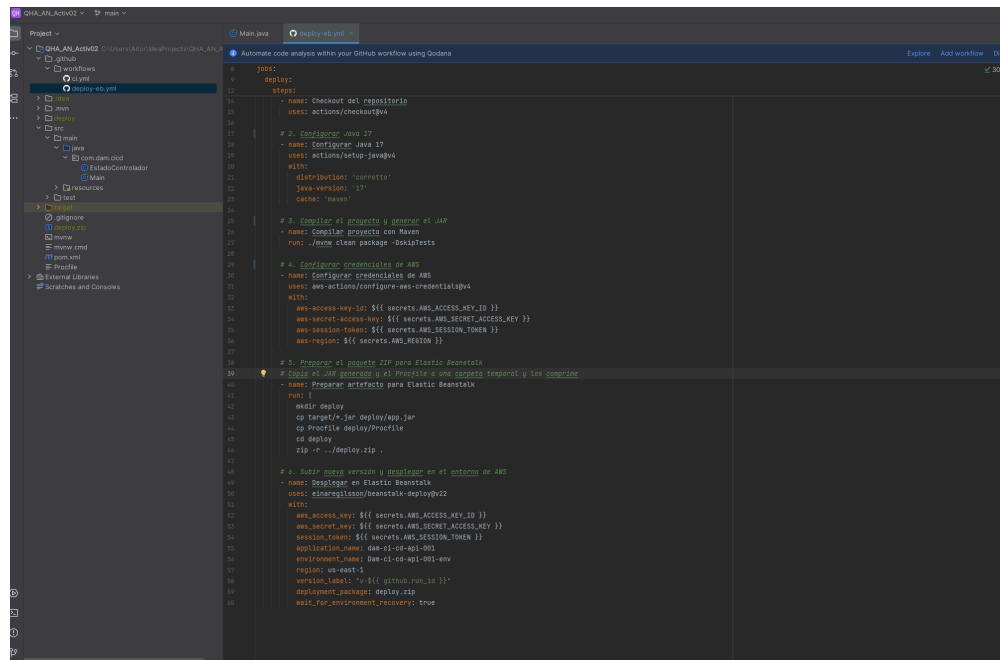
Mode                LastWriteTime         Length Name
----                -
d-----            12/01/2026   10:54         deploy

PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02> Copy-Item target/*.* deploy/app.jar
PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02> Copy-Item Procfile deploy/
PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02> Compress-Archive -Path deploy/* -DestinationPath deploy.zip -Force
PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02> ls

Directorio: C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02

Mode                LastWriteTime         Length Name
----                -
d-----            12/01/2026   10:27         .github
d-----            12/01/2026   10:53         .idea
d-----            12/01/2026    9:27         .mvn
d-----            12/01/2026   10:50         deploy
d-----            12/01/2026    9:23         src
d-----            12/01/2026   10:48         target
-a-----            12/01/2026   10:07         150  .gitignore
-a-----            12/01/2026   10:58         18073681 deploy.zip
-a-----            12/01/2026    9:27         10069 mvnw
-a-----            12/01/2026    9:27         6087 mvnw.cmd
-a-----            12/01/2026   10:40         1455 pom.xml
-a-----            12/01/2026   10:53         22  Procfile

PS C:\Users\Aitor\IdeaProjects\QHA_AN_Activ02>
```



The screenshot shows an IDE with a GitHub Actions workflow file named `deploy-eb.yml`. The workflow is configured to automate the deployment of a Java application to AWS Elastic Beanstalk. The workflow includes the following steps:

- Checkout del repositorio**: Checkout the repository using `actions/checkout@v4`.
- Configurar Java 17**: Configure Java 17 using `actions/setup-java@v4` with distribution 'corretto' and java-version '17'.
- Compilar el proyecto y generar el JAR**: Compile the project and generate the JAR file using `mvn clean package -DskipTests`.
- Configurar credenciales de AWS**: Configure AWS credentials using `aws-actions/configure-aws-credentials@v4` with the `aws-access-key-id`, `aws-secret-access-key`, `aws-session-token`, and `aws-region` secrets.
- Preparar el paquete ZIP para Elastic Beanstalk**: Prepare the deployment package by copying the JAR file and the Procfile to a temporary directory and creating a ZIP file.
- Subir nueva versión y desplegar en el entorno de AWS**: Deploy the new version to AWS Elastic Beanstalk using `aws-actions/elasticbeanstalk-deploy@v2` with the `aws-access-key-id`, `aws-secret-access-key`, `aws-session-token`, `aws-region`, `application-name`, `environment-name`, `version-label`, `deployment-package`, and `wait-for-environment-recovery` parameters.

Preparación de la carpeta de despliegue y configuración del Procfile

Verificación de la API accesible públicamente mediante URL de AWS

