# INTEGRACIÓN CONTINUA EN EL DESARROLLO ÁGIL

TRABAJO FINAL:
Documento de Usuario

MICAELA LEIVA ACHAVAL

# Tabla de Contenidos

En el desarrollo de esta práctica, nos enfocamos en la implementación de un sistema de integración continua utilizando GitHub Actions y Azure para la automatización de pruebas y despliegue de una aplicación web en la nube. A continuación, se detalla el proceso seguido en esta práctica, desde la instalación del entorno hasta la implementación del servicio en Azure.

## CONFIGURACION

Para comenzar, instalamos la infraestructura necesaria en nuestro entorno de desarrollo. Esto incluyó la creación de una cuenta gratuita en GitHub y otra en Azure para el despliegue de la aplicación. Posteriormente, configuramos nuestro entorno local instalando Git para la gestión de versiones, Visual Studio Code como entorno de desarrollo, y Maven para la construcción del proyecto. También se realizó la instalación de Docker y SonarQube para la evaluación de la calidad del código.

A continuación, creamos un repositorio local utilizando Git y lo vinculamos con un repositorio remoto en GitHub. Dentro de este repositorio, establecimos un flujo de trabajo para la integración continua mediante la configuración de un archivo YAML en la carpeta .github/workflows/. Este archivo definió los pasos necesarios para la compilación, prueba, análisis de calidad y despliegue de la aplicación.

## FLUJO DE TRABAJO

El flujo de trabajo se estructuró en cuatro etapas principales: compilación (build), pruebas (test), análisis de calidad (qa) y despliegue (deploy). En la etapa de compilación, se configuró el entorno con la versión adecuada de Java y se ejecutó el comando de Maven para empaquetar la aplicación. Durante la etapa de pruebas, se implementaron pruebas unitarias y funcionales utilizando Selenium y JUnit, verificando que todas las funcionalidades de la aplicación se ejecutaran correctamente.

En la fase de análisis de calidad, se configuró un contenedor de SonarQube para evaluar el código fuente del proyecto y garantizar el cumplimiento de estándares de calidad. Finalmente, en la etapa de despliegue, se configuró una aplicación en Azure Web Apps, y se estableció un flujo automático de implementación desde GitHub utilizando secretos para almacenar las credenciales necesarias.

## MOVIECARDS-SERVICE

El siguiente paso consistió en la creación de un nuevo repositorio llamado moviecards-service, basado en el código fuente proporcionado por el profesor en GitHub. Descargamos el código en formato ZIP, lo descomprimimos y lo convertimos en un nuevo repositorio local mediante Git. Luego, creamos un nuevo repositorio remoto en GitHub y subimos el código utilizando los comandos git add, git commit y git push.

Para desplegar el servicio en Azure, creamos una nueva aplicación en Azure App Service con el nombre moviecards-service-leiva. En el proceso de creación, Azure generó automáticamente un archivo de configuración YAML dentro del repositorio en GitHub, incluyendo un secreto para la autenticación del despliegue. Modificamos la sección deploy en el archivo .github/workflows/main.yaml para reemplazar el secreto asociado al repositorio del profesor por el generado por Azure. Después, eliminamos el archivo YAML del repositorio local, ya que no era necesario conservarlo.

Finalmente, probamos el servicio desplegado utilizando Postman. Realizamos pruebas de las operaciones definidas en ActorController.java y MovieController.java, asegurándonos de que el servicio respondiera correctamente a las solicitudes HTTP. Se verificó la creación de actores y películas, la consulta de registros individuales y la asociación de actores a películas, utilizando las siguientes rutas como se explica en el enunciado:

- POST https://moviecards-service-apellido.azurewebsites.net/actors
- GET https://moviecards-service-apellido.azurewebsites.net/actors
- GET https://moviecards-service-apellido.azurewebsites.net/actors/{idA}
- POST https://moviecards-service-apellido.azurewebsites.net/movies
- GET https://moviecards-service-apellido.azurewebsites.net/movies
- GET https://moviecards-service-apellido.azurewebsites.net/movies/{idM}

- GET https://moviecards-service-apellido.azurewebsites.net/movies/insc/{idA}/{idM}

Tras la verificación de todas las operaciones, confirmamos que el servicio moviecards-service quedó correctamente desplegado y funcional en Azure. Esto completa la implementación de la integración continua y despliegue automatizado de la aplicación, asegurando una entrega eficiente y confiable del software en la nube.

## UNIFICACIÓN DE MOVIECARDS CON EL SERVICIO

Para realizar una nueva versión de la aplicación moviecards, creamos una nueva rama de desarrollo llamada feature/moviecards-version2. En esta rama, realizamos modificaciones en el código para que la aplicación utilizara el servicio previamente desplegado en Azure, asegurando que continuara funcionando en la misma URL de la versión anterior.

En primer lugar, en el código fuente dentro de src/main, realizamos cambios en varios archivos clave. En MovieCardsApplication.java, añadimos un Bean para la creación de una instancia de RestTemplate, lo que permitió la comunicación con el servicio externo. En CardController.java, modificamos la inyección de dependencias para utilizar la anotación @Autowired en ActorService. En ActorServiceImpl.java y MovieServiceImpl.java, reemplazamos el uso de las interfaces JPA por llamadas al servicio externo a través de RestTemplate, asegurando que las operaciones CRUD sobre

actores y películas fueran gestionadas por el servicio moviecards-service en Azure. También eliminamos los constructores innecesarios y modificamos los métodos para utilizar las URLs correctas según nuestro apellido.

Posteriormente, adaptamos las pruebas unitarias en src/test para reflejar los cambios en la lógica de negocio. En ActorServiceImplTest.java y MovieServiceImplTest.java, reemplazamos las referencias a JPA con simulaciones de llamadas a RestTemplate, utilizando Mockito para probar el comportamiento del servicio sin depender de una base de datos local. Se modificaron las estructuras de datos de las pruebas para utilizar arreglos en lugar de listas y se actualizaron las aserciones para garantizar que las respuestas del servicio se gestionaran correctamente. Además, eliminamos pruebas que ya no eran relevantes debido a la nueva arquitectura basada en llamadas REST.

Una vez completados los cambios, realizamos pruebas exhaustivas en la rama feature/moviecards-version2 para validar la correcta integración con el servicio externo. Finalmente, tras verificar el correcto funcionamiento de la aplicación, realizamos un merge de la rama de desarrollo con la rama principal, integrando los cambios en el código principal de la aplicación y asegurando que la nueva versión estuviera operativa en Azure.