Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Automatización de Procesos con Docker, n8n y Modelos de IA: Análisis y Mejora Continua de Código

Autor: Alberto González López

Tutora: Adriana Toni Delgado

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Automatización de Procesos con Docker, n8n y Modelos de IA: Análisis y Mejora Continua de Código
Junio 2025

Autor: Alberto González López

Tutora:
Adriana Toni Delgado
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El desarrollo de software moderno se enfrenta a desafíos crecientes en la gestión de la calidad, seguridad y mantenibilidad del código. Con la proliferación de equipos de desarrollo distribuidos y la complejidad inherente a los entornos de Integración Continua y Despliegue Continuo (CI/CD), la revisión manual del código se ha convertido en un cuello de botella, un proceso costoso, lento y propenso a errores. Esta situación a menudo genera inconsistencias, vulnerabilidades no detectadas y una sobrecarga para los equipos, afectando directamente la eficiencia y la calidad del producto final.

La motivación para este proyecto surge precisamente de la observación de estas ineficiencias en los procesos de revisión y mejora continua del código. La necesidad de una revisión automática de código, complementada con pruebas generadas por agentes de Inteligencia Artificial (IA) y modelos de lenguaje (LLM) específicos, se presenta como una solución crítica. Se busca no solo agilizar estos procesos, sino también elevar los estándares de calidad y seguridad del código, asegurando que el software no solo funcione, sino que sea robusto, seguro y fácil de mantener.

La propuesta central de este trabajo radica en la automatización de este proceso mediante la utilización de Docker para la contenerización, n8n como herramienta de orquestación transversal, y modelos de IA avanzados para el análisis, mejora y generación de pruebas de código. La activación de este flujo de trabajo se vincula directamente a las subidas a GitHub, permitiendo una revisión dinámica y continua tanto del código nuevo como del ya existente, adaptándose a los nuevos estándares de desarrollo y seguridad. Esta iniciativa se alinea con la tendencia actual de DevOps y DevSecOps, donde la automatización y la inteligencia artificial se posicionan como pilares fundamentales para construir pipelines de desarrollo más eficientes y resilientes.

Abstract

Modern software development faces more and more challenges related to code quality, security, and maintainability. With the growth of distributed development teams and the complexity of Continuous Integration and Continuous Deployment (CI/CD) environments, manual code review has become a bottleneck — a process that is expensive, slow, and often leads to mistakes. This situation can create inconsistencies, undetected vulnerabilities, and too much workload for teams, directly affecting the efficiency and quality of the final product.

The idea for this project comes from seeing these problems in code review and continuous improvement processes. We really need an automatic way to review code, and it needs to be helped by tests made by AI agents and specific Large Language Models (LLMs). This is a critical solution. The goal is not just to make these processes faster, but also to make code quality and security better. We want to be sure that software not only works well but is also strong, safe, and easy to keep updated.

The main proposal of this work is to automate this process using Docker for containerization, n8n as the orchestration tool, and advanced AI models for code analysis, improvement, and test generation. This workflow is automatically triggered by code uploads to GitHub, allowing continuous and dynamic review of both new and existing code, while adapting to modern development and security standards. This initiative follows the current trend of DevOps and DevSecOps, where automation and artificial intelligence are becoming essential tools to build more efficient and resilient development pipelines.

Tabla de contenidos

1 Introducción

1.1 Objetivos generales y específicos

El propósito fundamental de este Trabajo Fin de Grado es abordar las ineficiencias en los procesos de desarrollo de software mediante la automatización inteligente.

El objetivo general de este proyecto es construir un sistema automatizado para la mejora continua de la calidad, seguridad y mantenibilidad del código, capaz de identificar automáticamente problemas, sugerir mejoras, generar pruebas unitarias para verificar la corrección de las mejoras y, finalmente, aplicar dichas mejoras al código de forma asistida por IA, con la opción de intervención humana para la validación final.

Para alcanzar este objetivo general, se establecen los siguientes objetivos específicos:

- Investigar y seleccionar herramientas adecuadas para la orquestación (n8n), el control de versiones (Git y GitHub), el análisis estático de código (SonarQube), el análisis de seguridad (OWASP ZAP), y los modelos de lenguaje (LLMs) y APIs de IA para la generación de código y pruebas.
- Diseñar una arquitectura modular basada en contenedores (Docker) que permita la integración fluida y escalable de todas las herramientas seleccionadas, asegurando la robustez y portabilidad del sistema.
- Implementar un flujo de trabajo automatizado y orquestado con n8n que se active de forma sincronizada mediante el uso de webhooks ante la subida de nuevos proyectos o modificaciones en un repositorio GitHub.
- Desarrollar e integrar módulos específicos para el análisis de vulnerabilidades, la generación automática de pruebas unitarias propuestas por los LLMs, la compilación y ejecución de código, y la corrección y mejora asistida por IA.
- Establecer un mecanismo de supervisión humana que permita la revisión y aprobación de las mejoras de código sugeridas por el sistema antes de su aplicación definitiva.
- Realizar una validación funcional exhaustiva del sistema desarrollado para confirmar su eficacia y el cumplimiento de los objetivos planteados.
- Analizar el impacto ambiental derivado del consumo energético asociado al uso intensivo de LLMs, y los riesgos de privacidad inherentes al tratamiento de datos sensibles, proponiendo consideraciones y medidas de mitigación.

1.2 Metodología de trabajo

Para el desarrollo de este Trabajo Fin de Grado, se adoptará un enfoque iterativo e incremental, permitiendo la adaptación y mejora continua a lo largo de las diferentes fases del proyecto. Esta metodología facilita la integración progresiva de componentes y la validación temprana de funcionalidades, crucial en un proyecto que combina diversas tecnologías emergentes.

El proyecto se estructurará en las siguientes fases principales:

Fase de Investigación y Selección de Herramientas: Identificación, evaluación y selección de las herramientas y tecnologías más adecuadas (n8n, Docker, Git/GitHub, webhooks, SonarQube, OWASP ZAP, LLMs y APIs de IA) que conformarán la base del sistema.

Fase de Diseño de la Arquitectura: Definición de la arquitectura global del sistema, incluyendo la modularidad, la orquestación de flujos de trabajo con n8n, y la integración de las herramientas mediante contenedores Docker. Se elaborarán diagramas de arquitectura para visualizar la interacción de los componentes.

Fase de Implementación y Desarrollo: Construcción de los módulos y flujos de trabajo automatizados, integrando las herramientas seleccionadas y desarrollando la lógica necesaria para la revisión, generación de pruebas, compilación, ejecución y mejora del código.

Fase de Pruebas y Validación: Realización de pruebas funcionales para verificar que el sistema opera según lo esperado y que los objetivos específicos se cumplen de manera efectiva. Esto incluirá la validación de la calidad del código mejorado y la eficacia de las pruebas generadas.

Fase de Análisis de Impacto y Documentación: Evaluación de las implicaciones ambientales y de privacidad del sistema, así como la redacción de la memoria final del TFG, detallando todo el proceso, los resultados obtenidos y las conclusiones.

La gestión del proyecto se llevará a cabo mediante un seguimiento continuo de tareas, utilizando herramientas de planificación que permitan la organización de los objetivos específicos en hitos manejables. El control de versiones (GitHub) será fundamental para la gestión del propio código del TFG y los artefactos generados. La calidad y validez del trabajo se asegurarán mediante la revisión constante de manuales técnicos, la aplicación de buenas prácticas de ingeniería de software durante el desarrollo y una fase de pruebas exhaustiva para garantizar la robustez y fiabilidad del sistema propuesto.

1.3 Estructura del documento

La presente memoria final del Trabajo Fin de Grado se organiza en cinco capítulos principales, diseñados para proporcionar una comprensión completa del proyecto:

El Capítulo 1, Introducción, sienta las bases del trabajo, presentando el contexto y la motivación del proyecto, los objetivos generales y específicos que guían su desarrollo, la metodología de trabajo adoptada y, finalmente, la estructura general de este documento.

El Capítulo 2, Análisis y selección de herramientas, se centra en la investigación y justificación de las herramientas tecnológicas clave utilizadas en el proyecto. Se abordará en detalle la herramienta de orquestación n8n, los sistemas de control de versiones Git y GitHub, diversas herramientas de análisis de código (SonarQube y OWASP ZAP), la selección y comparativa de modelos de lenguaje (LLMs) y APIs de IA, así como el uso de Docker para los entornos de ejecución y contenerización, y otras herramientas auxiliares.

El Capítulo 3, Diseño e implementación del sistema, describe la arquitectura propuesta para el sistema automatizado, detallando su diseño modular y basado en contenedores, así como los flujos de trabajo orquestados con n8n. Se explicará la integración y configuración de cada herramienta en la arquitectura, la lógica detrás de la generación automática de pruebas unitarias y los mecanismos de validación y supervisión humana del código.

El Capítulo 4, Resultados y conclusiones generales, evalúa el grado de cumplimiento de los objetivos establecidos al inicio del proyecto. Se discutirán las lecciones aprendidas durante el desarrollo, las limitaciones inherentes al sistema implementado y se esbozarán las futuras líneas de mejora y ampliación para este trabajo.

Finalmente, el Capítulo 5, Análisis de impacto, aborda dos dimensiones críticas del proyecto: el impacto ambiental asociado al consumo energético de los modelos de IA y los riesgos de privacidad derivados del tratamiento de datos. Se presentarán consideraciones de eficiencia, medidas de mitigación y el cumplimiento de normativas aplicables como GDPR.

2 Análisis y selección de herramientas

2.1 Herramienta de orquestación: n8n

2.1.1 Características y ventajas

Para la orquestación central de los flujos de trabajo en este proyecto, se ha seleccionado n8n, una potente herramienta de automatización de código abierto que se distingue por su flexibilidad y capacidad para integrar una vasta gama de servicios y aplicaciones. n8n se posiciona como una solución ideal para este TFG debido a su naturaleza de "fair-code", que permite el acceso completo a su código fuente y su uso gratuito, brindando transparencia y control total sobre el entorno de ejecución.

Entre sus características principales destacan:[1]

- Licencia "Fair-Code" y Código Abierto: Ofrece la libertad de utilizar, modificar y distribuir el software, garantizando un control absoluto sobre el procesamiento de datos y la privacidad, al no depender de servicios en la nube de terceros. Esto es fundamental para la seguridad y la autonomía del proyecto.
- Interfaz Visual de Flujos de Trabajo (Low-Code): n8n proporciona un editor visual intuitivo basado en nodos, que simplifica enormemente la creación de flujos de trabajo complejos. Permite a los usuarios arrastrar, conectar y configurar nodos que representan acciones específicas o integraciones, facilitando la comprensión y depuración visual de los procesos.
- Amplio Soporte de Integraciones (Nodos Predefinidos): Cuenta con más de 200 integraciones listas para usar (nodos predefinidos) para servicios como GitHub, Slack, bases de datos y APIs REST, entre muchos otros. Esto agiliza la conexión con las diversas herramientas que componen el sistema de mejora continua de código.
- Orquestación Condicional y Lógica Compleja: Permite implementar lógica avanzada dentro de los flujos, incluyendo condicionales, bucles, la ejecución de funciones personalizadas con JavaScript/TypeScript, y el almacenamiento temporal de datos. Esto es crucial para manejar las ramificaciones lógicas inherentes a los procesos de análisis y mejora de código.
- Activadores Versátiles (triggers): Los flujos pueden ser disparados por una variedad de eventos, incluyendo webhooks (fundamentales para la integración con GitHub), cron jobs para ejecuciones programadas, eventos internos o activadores manuales desde la interfaz, lo que dota al sistema de gran flexibilidad.
- Persistencia y Manejo de Datos: Facilita el paso y la transformación de datos entre nodos mediante variables y estructuras JSON, además de soportar la interacción con bases de datos y archivos, garantizando la consistencia y el procesamiento adecuado de la información del código.
- Escalabilidad y Despliegue Flexible: n8n está diseñado para escalar horizontalmente, siendo altamente compatible con entornos de contenedores como Docker y Kubernetes. Esta característica es esencial para la robustez y adaptabilidad del sistema en un contexto DevOps y CI/CD[2].
- Comunidad Activa y Extensibilidad: Dispone de una comunidad open-source activa que contribuye a su mejora continua. Además, ofrece la posibilidad de crear nodos personalizados, lo que asegura que la

herramienta puede ser adaptada y extendida para cubrir cualquier necesidad específica del proyecto.

Las ventajas concretas que n8n aporta al objetivo de este TFG son sustanciales. Su capacidad para orquestar la comunicación entre sistemas dispares sin una programación excesiva, la modularidad de sus flujos de trabajo y la flexibilidad en el despliegue lo convierten en el pilar central para la automatización de un pipeline de análisis y mejora continua de código. Permite diseñar procesos complejos de manera visual, lo que reduce el tiempo de desarrollo y facilita el mantenimiento, mientras que su naturaleza autoalojable ofrece un control total sobre los datos y la seguridad.

2.1.2 Integración con APIs y GitHub

La capacidad de n8n para interactuar con APIs externas y, en particular, con GitHub, es fundamental para la operatividad del sistema propuesto[1].[3]

n8n facilita la conexión con APIs externas a través de sus nodos HTTP genéricos, que permiten configurar solicitudes GET, POST, PUT, DELETE, y manejar diferentes tipos de autenticación (API Key, OAuth2, Basic Auth). Esta versatilidad asegura que el sistema pueda interactuar con cualquier servicio web, incluyendo herramientas de análisis de código o modelos de IA que expongan sus funcionalidades a través de una API REST.

Específicamente, la integración de n8n con GitHub es un pilar crucial para el disparador y la gestión del flujo de mejora continua del código. n8n se integra con GitHub principalmente a través de:

- Webhooks: Los webhooks de GitHub actúan como los disparadores principales de los flujos de n8n. Cuando un evento específico ocurre en un repositorio (como un push a una rama, la creación de un pull request, o un issue abierto), GitHub envía una notificación (payload JSON) a una URL de webhook configurada en n8n. Esto permite que el flujo de automatización se active de manera reactiva y en tiempo real ante cualquier cambio en el código. Para este proyecto, los eventos de push (al guardar un nuevo proyecto o realizar modificaciones) serán los disparadores clave.
    - ngrok para Exposición de Webhooks: Dado que n8n se despliega en un entorno contenedorizado que podría no estar directamente accesible desde internet, se utiliza ngrok como una herramienta auxiliar indispensable. ngrok crea un túnel seguro desde una URL pública (generada por ngrok) hacia el puerto local donde se ejecuta n8n. Esto permite que los webhooks de GitHub, que requieren una URL accesible desde internet, puedan alcanzar la instancia de n8n autoalojada y disparar los flujos correspondientes.
- Nodos de GitHub: n8n ofrece nodos predefinidos específicos para GitHub que permiten realizar una amplia gama de operaciones directamente desde los flujos de trabajo. Estas operaciones incluyen:
    - Clonar repositorios: Para obtener el código fuente a analizar.
    - Subir y modificar archivos: Para aplicar las mejoras sugeridas por la IA.
    - Crear y gestionar Pull Requests: Para someter las mejoras a revisión humana.
    - Comentar en issues o Pull Requests: Para proporcionar retroalimentación automatizada sobre el análisis y las mejoras.

- Obtener información del repositorio: Para acceder a metadatos del proyecto.

Esta profunda integración con GitHub, impulsada por la eficiencia de los webhooks y la capacidad de exposición de ngrok, permite que n8n no solo orqueste el proceso de análisis y mejora, sino que también actúe como un componente activo dentro del ciclo de vida del desarrollo de software, automatizando tareas que tradicionalmente requerirían intervención manual y asegurando que las revisiones de código y las aplicaciones de mejoras sean fluidas y consistentes.

2.2 Sistemas de control de versiones: Git y GitHub

El desarrollo de software moderno es inconcebible sin la gestión de un sistema de control de versiones robusto, y en este contexto, Git se ha consolidado como el estándar de facto. Como sistema de control de versiones distribuido, Git permite a los equipos de desarrollo rastrear cada cambio en el código fuente, gestionar múltiples versiones de un proyecto, colaborar de manera eficiente y segura, y revertir a estados anteriores si es necesario. Su arquitectura distribuida garantiza que cada desarrollador posea una copia completa del historial del repositorio, lo que aumenta la resiliencia y la flexibilidad en el flujo de trabajo.

Complementando a Git, GitHub[4] emerge como la plataforma de alojamiento de repositorios Git más utilizada, transformando el control de versiones en una experiencia colaborativa. GitHub no es solo un repositorio remoto; es un ecosistema que facilita la gestión de proyectos, la revisión de código y la automatización de flujos de trabajo. Proporciona herramientas esenciales como la gestión de issues, foros de discusión y, crucialmente para este proyecto, la capacidad de configurar webhooks.

Para el sistema de mejora continua de código propuesto en este TFG, GitHub actúa como el punto de origen de los eventos que disparan la automatización. Cualquier cambio, como un push a una rama principal o la apertura de un pull request, genera un evento que, a través de un webhook configurado, notifica instantáneamente a n8n. Esta conexión reactiva es vital, ya que permite que el flujo de análisis y mejora de código se inicie de forma automática e inmediata cada vez que se actualiza el repositorio, asegurando que el código siempre esté bajo escrutinio y alineado con los últimos estándares de calidad y seguridad.

Además de ser el disparador, GitHub sirve como el destino natural para las propuestas de mejora generadas por el sistema. Las correcciones, refactorizaciones o las pruebas unitarias generadas por los modelos de IA pueden ser propuestas de vuelta al repositorio como Pull Requests (PRs). Este mecanismo permite que las mejoras automáticas sean revisadas por desarrolladores humanos, quienes pueden validarlas, solicitar cambios o fusionarlas con la rama principal. Esta integración cierra el ciclo de mejora continua, combinando la eficiencia de la automatización con la supervisión y la inteligencia humana.

2.3 Herramientas de análisis de código

2.3.1 Análisis estático con SonarQube[5]

El análisis estático de código es una práctica fundamental en el ciclo de vida del desarrollo de software, permitiendo la detección temprana de errores, vulnerabilidades, malas prácticas y deuda técnica sin necesidad de ejecutar el

código. Esta práctica es crucial para mantener la calidad, seguridad y mantenibilidad del software a lo largo del tiempo.

Para llevar a cabo este análisis, se ha seleccionado SonarQube, una plataforma de código abierto líder en la evaluación continua de la calidad del código. Sus características principales incluyen:

- Compatibilidad Multilenguaje: Soporta un amplio abanico de lenguajes de programación, como Java, Python, JavaScript, C#, C++, TypeScript y PHP, lo que lo hace versátil para diversos entornos de desarrollo.
- Detección Integral de Problemas: Identifica eficientemente bugs (errores lógicos), vulnerabilidades de seguridad (como inyecciones SQL o exposiciones de datos), y "code smells" (patrones sintomáticos de mal diseño o baja calidad estructural).
- Cálculo de Deuda Técnica: Ofrece una estimación cuantificable del esfuerzo necesario para corregir las deficiencias detectadas, proporcionando una visión clara de la inversión requerida para mejorar la calidad del código.
- Métricas y Reportes Detallados: Genera informes exhaustivos y métricas clave, incluyendo cobertura de pruebas, duplicación de código y complejidad logarítmica, facilitando la monitorización y el seguimiento de la calidad.
- Reglas Personalizables: Permite definir y adaptar conjuntos de reglas específicas, alineándose con los estándares de calidad internos de cualquier organización.
- Interfaz Web Intuitiva: Proporciona un panel de control accesible que permite navegar por los resultados, visualizar el historial de calidad y analizar el estado de los componentes del código.

La integración de SonarQube en un flujo de trabajo automatizado es un proceso sencillo. Dentro de un pipeline de Integración Continua (CI/CD) o un sistema orquestado por n8n, SonarQube se incorpora como una fase de análisis de código. Tras un evento desencadenante en GitHub (como un push o un pull request), el flujo automatizado compila el proyecto y ejecuta el escáner de SonarQube. Los resultados se envían a un nodo de documentación para su visualización y análisis. Es posible configurar umbrales de calidad que, de no cumplirse, podrían incluso bloquear un despliegue. Adicionalmente, SonarQube puede integrarse con n8n para automatizar su ejecución y, mediante webhooks, puede comentar los resultados directamente en los pull requests de GitHub.

Para este Trabajo Fin de Grado, SonarQube aporta ventajas específicas que son cruciales para los objetivos de mejora continua:

- Mejora Sistemática de la Calidad: Permite la detección automática y consistente de errores y malas prácticas, reduciendo la necesidad de revisiones manuales exhaustivas.
- Reducción de la Deuda Técnica: Facilita la identificación y cuantificación de problemas estructurales, lo que ayuda a priorizar y abordar las mejoras.
- Métricas Objetivas: Proporciona datos concretos y trazables sobre la calidad del código, esenciales para la evaluación y demostración de la efectividad del sistema.

- Facilidad de Integración: Su compatibilidad con Docker, GitHub y la capacidad de orquestación de n8n lo hacen ideal para el ecosistema automatizado propuesto.
- Estandarización: Contribuye a la adaptación de prácticas de codificación coherentes y sostenibles.
- Validación Automatizada: Actúa como un criterio objetivo para la aceptación del código, permitiendo la automatización de decisiones dentro del pipeline.

2.3.2 Análisis de seguridad con OWASP ZAP[6]

El análisis de seguridad de aplicaciones es un componente crítico en cualquier ciclo de desarrollo de software, cuyo objetivo es identificar y mitigar vulnerabilidades antes de que puedan ser explotadas. En este ámbito, se distinguen principalmente dos enfoques:

SAST (Static Application Security Testing): Analiza el código fuente, bytecode o binarios sin necesidad de ejecutar la aplicación. Se enfoca en la detección temprana de errores de seguridad comunes directamente en el código.
DAST (Dynamic Application Security Testing): Analiza la aplicación en ejecución, simulando ataques desde el exterior. No requiere acceso al código fuente y es fundamental porque puede detectar vulnerabilidades que solo se manifiestan en tiempo de ejecución, como errores de validación, configuraciones inseguras o fugas de información en las respuestas HTTP, ofreciendo una perspectiva real de un atacante.
El análisis DAST es fundamental ya que complementa al SAST, proporcionando una capa de seguridad adicional al validar la robustez de la aplicación en su entorno de ejecución. Detecta vulnerabilidades que podrían pasar desapercibidas en un análisis estático, asegurando que el sistema sea seguro no solo en su estructura de código, sino también en su comportamiento real.

Para este componente dinámico, se ha seleccionado OWASP ZAP (Zed Attack Proxy), una de las herramientas DAST de código abierto más populares y potentes, desarrollada por la Open Web Application Security Project (OWASP). Sus capacidades principales abarcan:

Escaneo Pasivo y Activo: Permite un escaneo pasivo que inspecciona el tráfico sin enviar cargas maliciosas, y un escaneo activo que simula ataques reales para encontrar vulnerabilidades explotables como SQL Injection, XSS (Cross-Site Scripting), CSRF, y fallos de configuración.
Proxy Interceptor: Actúa como un proxy man-in-the-middle para capturar, analizar y modificar el tráfico HTTP/HTTPS entre el cliente y la aplicación web, facilitando la identificación de puntos débiles.
API REST para Automatización: Expone una API REST completa, lo que permite su control y configuración programática. Esta característica es esencial para integrar ZAP en flujos automatizados de CI/CD.
Modo "Headless": Puede ejecutarse sin interfaz gráfica de usuario, lo que lo hace ideal para su despliegue en entornos de servidores y pipelines de automatización como el propuesto en este TFG, típicamente dentro de contenedores Docker.
Informes Detallados: Genera informes en diversos formatos (JSON, HTML, XML) que resumen las vulnerabilidades detectadas, facilitando el procesamiento y la comunicación de los hallazgos.

La integración de OWASP ZAP en un flujo de trabajo automatizado con n8n y Docker es altamente eficiente. El proceso generalmente implica el despliegue de la aplicación web a analizar en un entorno de pruebas. Posteriormente, ZAP se ejecuta en modo headless dentro de un contenedor Docker, dirigiéndose a la URL de la aplicación y configurando los tipos de escaneo deseados. La orquestación con n8n permite lanzar estos escaneos automáticamente tras un evento relevante (ej. el despliegue de una nueva versión del código). Una vez completado el escaneo, n8n puede recolectar los informes de ZAP, analizarlos y, si es necesario, registrar los hallazgos directamente en GitHub como issues o comentarios en pull requests, cerrando el ciclo de retroalimentación de seguridad.

Las ventajas concretas que OWASP ZAP aporta a este Trabajo Fin de Grado son significativas para fortalecer el sistema de mejora continua del código:

Detección de Vulnerabilidades en Tiempo de Ejecución: Permite identificar fallos de seguridad que solo se manifiestan cuando la aplicación está en funcionamiento, complementando las debilidades que el análisis estático podría pasar por alto.
Complementariedad con SonarQube: Ofrece una visión holística de la seguridad, combinando la detección estática de vulnerabilidades en el código fuente con la evaluación dinámica del comportamiento de la aplicación desplegada.
Evaluación de Seguridad Automatizada: Facilita la automatización de las pruebas de seguridad, permitiendo la detección temprana de riesgos y la posibilidad de bloquear despliegues o alertar al equipo ante vulnerabilidades críticas.
Integración Fluida: Su API REST y el soporte para ejecución en modo headless garantizan una integración sencilla y efectiva con el ecosistema de orquestación (n8n) y contenerización (Docker) del proyecto.
Auditoría de Buenas Prácticas de Seguridad Web: Ayuda a validar el cumplimiento de estándares de seguridad reconocidos, como los detallados en el OWASP Top 10[7].

2.4 Modelos de lenguaje (LLMs) y APIs de IA

Los Modelos de Lenguaje Grandes (LLMs) han irrumpido en el panorama de la inteligencia artificial, provocando una transformación radical en el desarrollo de software. Estos modelos, entrenados con grandes cantidades de datos de texto y código, poseen una capacidad sin precedentes para comprender, generar y manipular lenguaje natural y código fuente. Su habilidad para realizar tareas como la generación de código, la refactorización, la explicación de código, la traducción entre lenguajes de programación y, crucialmente para este proyecto, la generación de pruebas unitarias, los convierte en una pieza central para la automatización inteligente. Son capaces de identificar patrones complejos, inferir intenciones y producir resultados coherentes y contextualmente relevantes, lo que los hace ideales para mejorar la eficiencia y calidad en el ciclo de desarrollo.

2.4.1 Comparativa de modelos disponibles

La selección del LLM adecuado para un proyecto de automatización de código es un factor determinante en el rendimiento y la eficacia del sistema. Al comparar los modelos disponibles, se deben considerar diversos factores críticos:

- Coste: El precio por token de entrada y salida, así como los costes asociados a la inferencia o el fine-tuning, son esenciales para la viabilidad económica del proyecto.
- Rendimiento y Latencia: La velocidad de respuesta del modelo es crucial, especialmente en flujos de trabajo automatizados donde la inmediatez puede impactar la experiencia del usuario y la eficiencia general del pipeline.
- Tamaño del Modelo y Capacidad de Contexto: Los modelos varían en su tamaño y en la cantidad de texto que pueden procesar o generar en una única interacción. Una mayor capacidad de contexto es ventajosa para entender bases de código complejas y generar pruebas más relevantes.
- Capacidad de Fine-tuning: La posibilidad de entrenar el modelo con datos específicos del proyecto (por ejemplo, el estilo de codificación de la organización o patrones de pruebas existentes) puede mejorar significativamente la calidad de los resultados.
- Disponibilidad de API y Facilidad de Integración: La existencia de una API robusta y bien documentada simplifica la integración en el sistema orquestado.
- Licenciamiento: La licencia bajo la cual se ofrece el modelo puede afectar su uso en entornos comerciales o proyectos con requisitos específicos.
- Especialización en Código: Algunos LLMs están específicamente entrenados o optimizados para tareas de código lo que puede resultar en una mayor precisión y relevancia en la generación de código y pruebas.
- Actualmente, el panorama de los LLMs es altamente dinámico, con una constante evolución y aparición de nuevos modelos. Entre los más relevantes para tareas relacionadas con el código se encuentran modelos de la familia GPT (OpenAI)[8] como GPT-4 (con variantes como GPT-4o y GPT-4 Turbo), los modelos de Google (Gemini[9] Pro, Code y APIs), y modelos open-source como Llama[10] (Meta) y sus derivados (por ejemplo, Code Llama, Phi-3). Cada uno presenta fortalezas y debilidades que deben ser evaluadas en función de las necesidades específicas de generación de código y pruebas unitarias de este TFG.

2.4.2 Acceso y Utilización de LLMs mediante APIs en n8n

Para integrar las capacidades de los Modelos de Lenguaje Grandes (LLMs) en el sistema automatizado, el proyecto se apoya en el acceso a estos modelos a través de sus APIs.[1]

Esta estrategia se prefiere sobre el despliegue local o el entrenamiento de modelos propios debido a ventajas significativas en cuanto a la complejidad de infraestructura, costes operativos y acceso a modelos de vanguardia. Utilizar APIs externas permite al sistema beneficiarse de la constante evolución y optimización de los LLMs por parte de sus proveedores, asegurando un rendimiento óptimo y una gestión simplificada.

La herramienta de orquestación n8n juega un papel central en la interacción con estas APIs de IA. En lugar de una "selección independiente de APIs" por parte del desarrollador, el enfoque se traslada a la configuración y aprovechamiento de los nodos de n8n diseñados para interactuar con los principales proveedores de LLMs. n8n simplifica este proceso mediante:

- Nodos Predefinidos de LLMs: n8n ofrece nodos específicos (por ejemplo, para OpenAI, Google AI, o nodos HTTP genéricos altamente configurables) que facilitan la conexión directa a las APIs de los LLMs.

Estos nodos encapsulan la complejidad de las solicitudes HTTP, la gestión de autenticación (claves API, tokens) y el parseo de las respuestas JSON.

- Construcción de Prompts Dinámicos: A través de la lógica de los flujos de n8n, es posible construir prompts complejos y dinámicos para los LLMs, inyectando código fuente, requisitos de pruebas o descripciones de problemas detectados por SonarQube o ZAP. Esta capacidad es crucial para guiar al LLM en la generación de código para mejoras o la generación de pruebas unitarias específicas y relevantes.
- Procesamiento de Respuestas: n8n permite capturar y procesar las respuestas de los LLMs. Por ejemplo, el código generado por la IA puede ser extraído del payload de la respuesta, validado y luego utilizado por otros nodos de n8n para su integración en el repositorio GitHub o para la generación de pruebas unitarias mediante Junit para código Java.

Las funcionalidades específicas que se buscarán en las APIs de los LLMs, accesibles vía n8n, incluyen la capacidad de:

- Generar y refactorizar código: Recibir código existente o una descripción de un problema y devolver una propuesta de mejora o una corrección.
- Crear pruebas unitarias: A partir de un fragmento de código o una descripción de funcionalidad, generar casos de prueba completos con aserciones.

Los criterios para la elección final de la API de LLM se centrarán en la robustez y fiabilidad del nodo o la integración de n8n para dicho proveedor, la calidad y relevancia del código/pruebas generadas por el modelo para los lenguajes de programación del proyecto, la latencia de las respuestas, y la idoneidad de la documentación y el soporte para resolver cualquier desafío de integración.

2.5 Entornos de ejecución y contenedores: Docker

En el contexto de un sistema automatizado que integra múltiples herramientas y garantiza entornos consistentes para el análisis y la mejora de código, la elección de Docker[11] y la tecnología de contenedores es fundamental. Docker es una plataforma que permite empaquetar aplicaciones y todas sus dependencias en unidades estandarizadas llamadas contenedores. A diferencia de las máquinas virtuales, que virtualizan el hardware y requieren un sistema operativo completo para cada instancia, los contenedores comparten el kernel del sistema operativo anfitrión, lo que los hace significativamente más ligeros, eficientes y portátiles. Cada contenedor funciona como un entorno aislado y reproducible, garantizando que una aplicación se ejecute de la misma manera, independientemente del entorno anfitrión.

El uso de contenedores con Docker ofrece beneficios sustanciales en el desarrollo y despliegue de software, especialmente en proyectos complejos o que involucran múltiples servicios:

- Consistencia del Entorno: Elimina el problema del "funciona en mi máquina" al asegurar que la aplicación y sus dependencias se ejecuten de forma idéntica en cualquier entorno (desarrollo, pruebas, producción).
- Aislamiento: Cada contenedor aísla la aplicación del sistema anfitrión y de otras aplicaciones, previniendo conflictos de dependencias y mejorando la seguridad.

- Facilidad de Despliegue: Simplifica el proceso de despliegue, permitiendo que las aplicaciones se muevan entre diferentes entornos de manera rápida y sin fricciones.
- Gestión de Dependencias: Empaqueta todas las librerías, frameworks y configuraciones necesarias junto con la aplicación, lo que simplifica la gestión de dependencias.
- Escalabilidad: Los contenedores pueden ser iniciados, detenidos y escalados horizontalmente de manera eficiente, lo que facilita la adaptación a las demandas cambiantes del sistema.

Para el presente Trabajo Fin de Grado, Docker aporta ventajas específicas y críticas que sustentan la robustez y eficiencia del sistema propuesto:

- Despliegue Simplificado de Herramientas: Facilita enormemente el despliegue y la configuración de herramientas como n8n, SonarQube y OWASP ZAP y ngrok. Cada una de estas herramientas puede ejecutarse en su propio contenedor aislado, lo que simplifica su gestión, actualización y resolución de problemas, sin interferir con otras partes del sistema.
- Entornos Consistentes para Análisis y Pruebas: Asegura que los escaneos de código (con SonarQube), los análisis de seguridad (con OWASP ZAP) y la compilación/ejecución del código analizado se realicen en entornos idénticos y controlados. Esto es crucial para garantizar la reproducibilidad de los resultados y la fiabilidad de las pruebas.
- Aislamiento de Dependencias: Permite que cada componente del sistema funcione con sus versiones específicas de librerías y dependencias sin causar conflictos globales en el sistema anfitrión.
- Portabilidad y Reproducibilidad: El sistema completo, o partes de él, puede ser fácilmente empaquetado y replicado en diferentes entornos, lo que es una gran ventaja para la portabilidad de la solución o partes de ella.

En el flujo de trabajo general orquestado por n8n, Docker se integra de manera transparente. n8n puede interactuar con Docker directamente a través de comandos del sistema operativo (ejecutando comandos docker run, docker exec, docker stop) o mediante la API de Docker. Esto permite a n8n:

- Iniciar contenedores bajo demanda: Lanzar contenedores efímeros para ejecutar escaneos de SonarQube o ZAP sobre el código descargado del repositorio.
- Ejecutar comandos dentro de contenedores existentes: Para realizar compilaciones, ejecuciones de pruebas unitarias o cualquier otra operación de procesamiento de código dentro de un entorno controlado.
- Gestionar el ciclo de vida de los contenedores: Asegurando que los recursos se liberen una vez que las tareas automatizadas han concluido.

Esta sinergia entre Docker y n8n es lo que permite la creación de un pipeline de mejora continua de código altamente automatizado, modular y reproducible.

2.6 Herramientas auxiliares y librerías utilizadas

Además de las herramientas principales ya detalladas, el sistema propuesto se apoya en un conjunto de herramientas auxiliares y librerías que son fundamentales para la orquestación de los flujos de trabajo, la interacción con los servicios y la ejecución de tareas específicas. Estas herramientas aseguran la interoperabilidad y la eficiencia en el procesamiento del código.

- Java y Librerías para JUnit: Dado el enfoque en la generación y validación de pruebas unitarias, el lenguaje de programación Java se utiliza como lenguaje principal para el código que será analizado y para el cual se generarán y ejecutarán las pruebas. En este contexto, las librerías de JUnit son imprescindibles. JUnit es el framework de pruebas unitarias más extendido para Java, y su inclusión es crucial para la compilación, ejecución y validación de las pruebas generadas por los LLMs, permitiendo verificar automáticamente la corrección de las mejoras propuestas.

- Interfaz de Línea de Comando (CLIs): Para la interacción programática y el control sobre las herramientas, se utilizan diversas CLIs:

    - Git CLI: Imprescindible para operaciones básicas de control de versiones como la clonación de repositorios de GitHub (git clone), la gestión de ramas (git checkout), la adición de cambios (git add), la realización de commits (git commit) y el push de código (git push).
    - Sonar-Scanner CLI: Una herramienta de línea de comandos que se utiliza para ejecutar el análisis de código estático de SonarQube. Permite configurar y lanzar el escaneo de un proyecto y enviar los resultados al servidor SonarQube desde cualquier entorno automatizado.
    - Docker CLI: Utilizada para la gestión de contenedores Docker, permitiendo operaciones como iniciar (docker run), detener (docker stop), ejecutar comandos dentro de contenedores (docker exec) y construir imágenes (docker build).
- Formato JSON para el Paso de Datos: El formato JSON (JavaScript Object Notation) es utilizado de manera intensiva para el intercambio de datos entre las distintas herramientas y componentes del sistema. Su naturaleza ligera y legible por desarrolladores y máquinas lo convierte en el estándar para la comunicación con APIs (incluyendo las de los LLMs), la recepción de payloads de webhooks (como los de GitHub) y el procesamiento de resultados de análisis. n8n, en particular, se beneficia enormemente del formato JSON para manipular y transformar la información a lo largo de sus flujos.

- SSH (Secure Shell): Para la ejecución segura de comandos y la automatización de tareas en el servidor anfitrión donde se despliegan los componentes del sistema (por ejemplo, el propio n8n o los contenedores Docker), se utiliza SSH. Esto permite lanzar scripts, gestionar archivos o interactuar con procesos de forma remota y segura, siendo vital para orquestar acciones que requieran acceso directo al sistema operativo del anfitrión.
- ngrok[12]: Esta herramienta es esencial para exponer de forma segura los servicios locales a internet, permitiendo que los webhooks de GitHub alcancen la instancia de n8n que se ejecuta en un entorno privado o detrás de un firewall. ngrok crea un túnel seguro y una URL pública temporal que redirige el tráfico hacia el puerto específico del contenedor de n8n, lo que permite que GitHub envíe sus notificaciones de eventos al orquestador.

- Visual Studio Code (VS Code): Como entorno de desarrollo integrado (IDE) ligero y extensible, VS Code ha sido una herramienta fundamental en la gestión y el desarrollo del proyecto. Su versatilidad permite la edición de código fuente en diversos lenguajes (Java, JavaScript para n8n, configuración Dockerfile, scripts Bash), la gestión de repositorios Git, la depuración y la edición de archivos JSON. Su amplio ecosistema de extensiones lo convierte en una herramienta central para la productividad del desarrollador y la configuración de todos los componentes del sistema.

Estas herramientas y librerías, aunque auxiliares, son pilares que garantizan la fluidez y la seguridad de los flujos de trabajo automatizados, permitiendo una interacción eficaz entre todos los componentes del sistema de mejora continua del código.

3 Diseño e implementación del sistema

3.1 Diseño de la arquitectura del sistema

La filosofía de diseño inherente a la arquitectura de este sistema se basa en los principios de modularidad, separación de responsabilidades, escalabilidad, robustez y mantenibilidad. Este enfoque busca crear un sistema flexible y eficiente, capaz de integrar diversas funcionalidades de análisis y mejora de código de forma cohesiva.

Para traducir esta filosofía a la práctica, se ha optado por una arquitectura basada íntegramente en contenedores Docker. Cada herramienta principal y auxiliar que conforma el sistema —como n8n, SonarQube, OWASP ZAP, y el servicio de ngrok— se despliega como un módulo independiente dentro de su propio contenedor Docker. Esta estrategia no solo refuerza la modularidad al encapsular cada servicio con sus dependencias específicas, sino que también garantiza entornos consistentes y aislados, eliminando los problemas de compatibilidad y facilitando enormemente el despliegue y la gestión de cada componente como un servicio autónomo.

La comunicación entre estos módulos contenerizados se establece principalmente a través de una red Docker compartida. Esta red permite que los contenedores se comuniquen entre sí de forma segura y eficiente utilizando sus nombres de servicio, lo que simplifica la configuración de interconexiones y asegura que los datos fluyan correctamente entre los distintos componentes. Además, se utilizan APIs (especialmente para la interacción con los LLMs y servicios externos), webhooks (para la comunicación reactiva desde GitHub hacia n8n), y volúmenes compartidos cuando es necesario persistir datos o compartir archivos entre contenedores. La incorporación de ngrok ha permitido exponer el webhook de n8n, autoalojado en un entorno local o privado, a internet, permitiendo que GitHub pueda enviarle notificaciones.

3.2 Flujos de trabajo orquestados con n8n

En esta arquitectura, n8n asume el rol central como el orquestador principal del sistema, actuando como coordinador de todas las operaciones del pipeline de mejora continua de código. Su interfaz visual de "low-code" facilita la

construcción y el mantenimiento de flujos de trabajo complejos que, de otra forma, requerirían una programación extensiva.

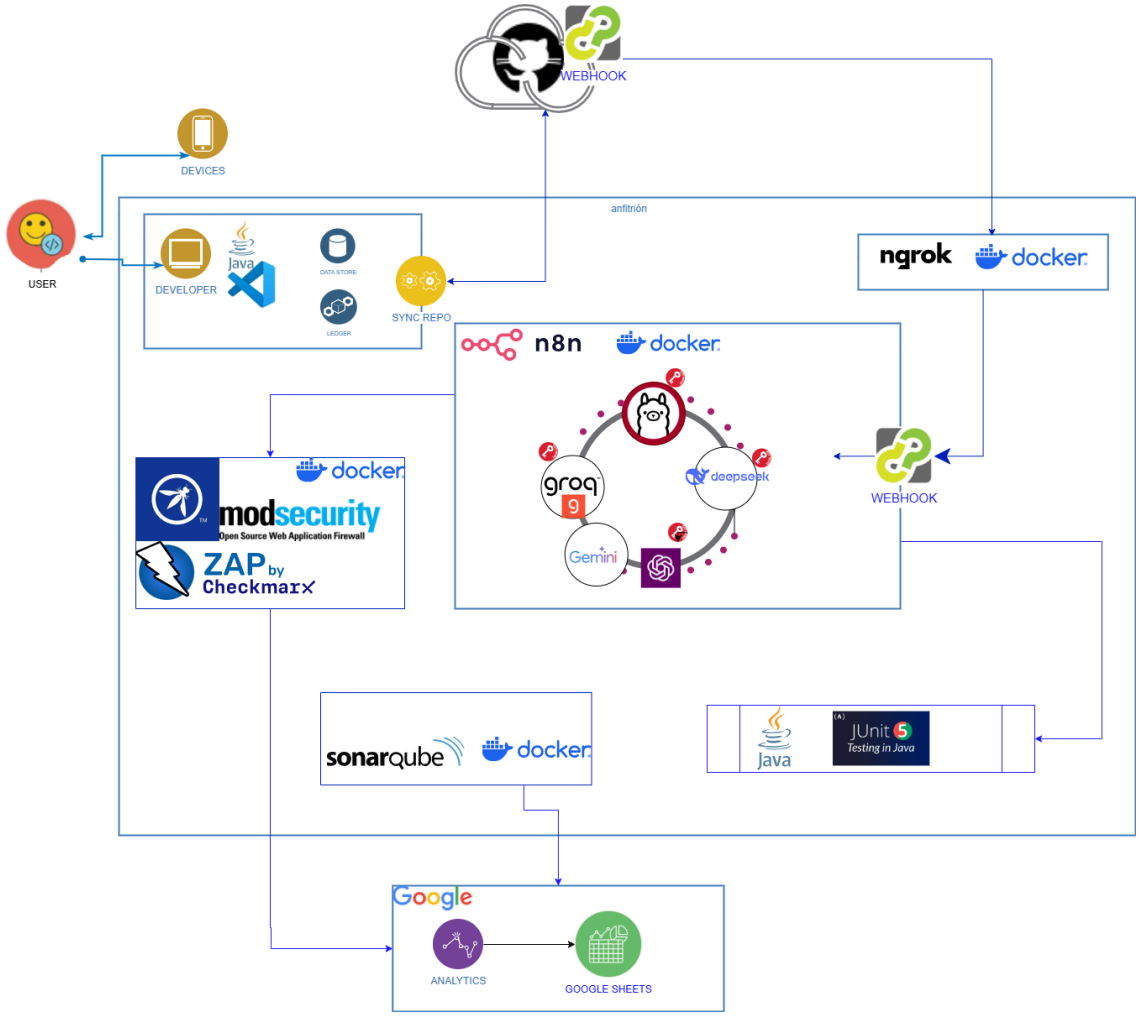
El flujo general de alto nivel que n8n orquesta se inicia con un evento en GitHub. Cuando se realiza un push a un repositorio o se crea un pull request, GitHub envía una notificación (webhook) a la URL expuesta por ngrok, que a su vez la reenvía al contenedor de n8n. Este evento dispara el flujo principal, que coordina las siguientes fases:

- Recepción y Preparación: n8n recibe el payload del webhook de GitHub y extrae la información relevante del repositorio y del evento.
- Descarga y Montaje de Código: Se activa un proceso para descargar el código fuente del repositorio GitHub en un volumen accesible por las herramientas de análisis.
- Análisis de Calidad Estático: Se invoca el contenedor de SonarQube para realizar un análisis estático del código, y n8n espera los resultados.
- Análisis de Seguridad Dinámico: Se despliega la aplicación en un entorno de pruebas y se orquesta el escaneo con OWASP ZAP, obteniendo sus informes de vulnerabilidades.
- Generación Inteligente de Mejoras y Pruebas: Los resultados de los análisis (SonarQube, ZAP) y el código fuente se envían a APIs de LLMs (a través de los nodos de n8n) para que generen sugerencias de mejora de código y pruebas unitarias complementarias.
- Compilación y Validación: El código mejorado y las pruebas generadas se compilan y ejecutan en un entorno controlado y n8n evalúa los resultados de estas pruebas.
- Retroalimentación y Propuesta de Cambios: Basado en la validación, n8n puede tomar decisiones: si las mejoras son exitosas, puede generar un pull request en GitHub con el código refactorizado y las nuevas pruebas; si hay fallos, puede crear un issue en GitHub o enviar notificaciones.

n8n gestiona la comunicación y el trasiego de datos entre los diversos módulos y herramientas mediante la transformación y manipulación de estructuras JSON, la invocación de APIs REST y la ejecución de comandos externos dentro de los contenedores Docker. La capacidad de n8n para crear flujos "modulares" es una ventaja significativa. Esto permite que sub-flujos para tareas específicas (un flujo para SonarQube, otro para ZAP, y otro para la interacción con LLMs) sean invocados por el flujo principal, lo que mejora la legibilidad, mantenibilidad y reusabilidad de la lógica de automatización.

3.3 Diagrama de arquitectura

El diseño arquitectónico del sistema se representa visualmente mediante un diagrama que ilustra la interacción entre sus diversos componentes, tanto internos como externos. Esta sección detalla cada elemento del diagrama y sus conexiones, proporcionando una comprensión clara de cómo el flujo de datos y control se articula a lo largo de todo el proceso.



3.3.1 Usuario/Desarrollador

Representa el/los equipos de desarrollo que interactúan con el sistema al realizar cambios en el código. Es quien inicia la cadena de eventos con cada contribución.

3.3.2 GitHub

Es el repositorio de código utilizado en todo el proceso. Es la pieza fundamental de gestión de código fuente y el principal disparador del sistema. Actúa como el punto de inicio donde se almacena y versiona el código de la aplicación.

GitHub envía un "Webhook (Push/PR Event)" directamente a n8n. Este webhook es una notificación que GitHub genera automáticamente cada vez que se produce un push de código o se abre/actualiza un pull request, conteniendo información sobre el evento.

3.3.3 Contenedor n8n

Actúa como elemento principal de la orquestación. Es responsable de recibir los eventos de GitHub, coordinar la ejecución de todos los análisis y procesos, y gestionar la lógica de evolución en CI/CD (Continuous Integration/Continuous Deployment)[2].

n8n no solo recibe webhooks, sino que también interactúa activamente con la API de GitHub para clonar repositorios, crear pull requests y comentar en issues.

n8n inicia y controla los análisis de seguridad y calidad en los contenedores de SonarQube Server[5], OWASP ZAP, y el proceso de Compilación/Ejecución.

Interacción con LLMs: n8n intercambia información con las APIs de LLMs, enviando prompts con código y datos de análisis, y recibiendo código mejorado o pruebas generadas por la inteligencia artificial.

n8n puede interactuar con un servicio SSH para ejecutar comandos directamente en el servidor anfitrión para realizar tareas de administración o preparación de entornos gestionados directamente por el anfitrión.

3.3.4 Contenedor SonarQube Server

Alberga la instancia del servidor SonarQube. Es el repositorio central donde se envían y almacenan los resultados del análisis de código estático.

n8n orquesta la ejecución del sonar-scanner en el código fuente, y los resultados del análisis se envían al Contenedor SonarQube Server.

3.3.5 Contenedor OWASP ZAP

Contiene la herramienta OWASP ZAP, dedicada a realizar el análisis dinámico de seguridad (DAST) sobre la aplicación web de prueba.

n8n inicia los escaneos de vulnerabilidades en tiempo de ejecución en el Contenedor OWASP ZAP.

3.3.6 Contenedor de Compilación/Ejecución (Java/JUnit)

Un entorno Docker especializado que se utiliza para compilar el código fuente Java (incluido el código mejorado por los LLMs) y ejecutar las pruebas unitarias JUnit generadas.

n8n envía el código fuente y las pruebas para su procesamiento, y recibe los resultados de la compilación y ejecución de las pruebas desde este contenedor.

3.3.7 Contenedor ngrok

Es vital para la implementación en un servidor autoalojado. Expone de forma segura los webhooks internos de n8n a internet.

El Contenedor ngrok establece un túnel de tráfico con el Contenedor n8n permitiendo la comunicación.

Interacción con GitHub: El webhook de GitHub se envía al Contenedor ngrok, el cual a su vez dirige el tráfico a n8n.

3.3.8 Componentes de Almacenamiento/Intercambio

Para facilitar el acceso y el intercambio de código y datos entre los diferentes servicios contenerizados se utilizan volúmenes persistentes:

Volumen Compartido Código Fuente: Este elemento representa un volumen Docker montado o un sistema de archivos compartido en el servidor anfitrión. Es el lugar donde el código fuente del repositorio de GitHub se descarga y donde

los contenedores de análisis (SonarQube, OWASP ZAP) y el entorno de Compilación/Ejecución pueden acceder y manipular el código.

n8n es el responsable de clonar y gestionar el código en este volumen.

3.3.9 Servicios Externos de IA

Los LLMs utilizados para la generación inteligente de código y pruebas son servicios externos a los que se accede a través de sus APIs.

APIs de LLMs Procesan el código y las instrucciones para generar nuevas versiones o pruebas.

El Contenedor n8n intercambia información con las APIs de LLMs, enviando los prompts que contienen el código fuente y los hallazgos de análisis, y recibiendo las respuestas que incluyen el código/pruebas generadas.

3.3.10 Flujos de Control y Datos

Las etiquetas y flechas en el diagrama especifican el tipo de información o acción que se transfiere entre los componentes:

Representa las acciones que n8n inicia o coordina, Iniciar Escaneo SonarQube, "Solicitar Generación LLM".

Indica los datos que se mueven entre componentes, "Webhook Payload", "Código Fuente", "Resultados de Análisis SonarQube", "Informes ZAP", "Prompt LLM", "Código/Pruebas Generadas", "Resultados Junit”.

3.4 Integración de herramientas en la arquitectura

Esta sección detalla la implementación práctica de la arquitectura diseñada, describiendo cómo cada herramienta se despliega, configura y se integra en el flujo de trabajo orquestado por n8n.

3.4.1 Despliegue y configuración de n8n en Docker

El despliegue de n8n, el orquestador central del sistema, se realiza utilizando Docker Compose, lo que garantiza un entorno consistente, portable y fácil de gestionar. La configuración se define en el archivo docker-compose.yml que se detalla en el anexo Docker_n8n.

3.4.2 Configuración de webhooks e integración con GitHub

La integración bidireccional entre el repositorio de código en GitHub y el orquestador n8n es fundamental para la operatividad del sistema de mejora continua. Esta conexión se establece principalmente mediante la configuración de webhooks en GitHub, los cuales actúan como los disparadores de los flujos automatizados en n8n.

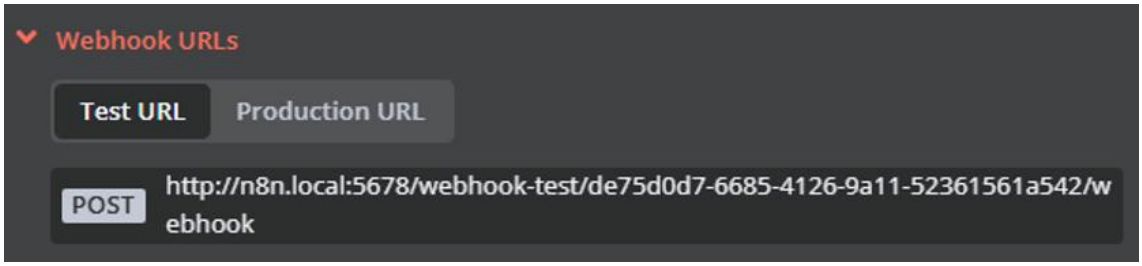
Configuración del Webhook en GitHub

Para que GitHub pueda notificar a n8n sobre los eventos relevantes, se deben seguir los siguientes pasos en la configuración del repositorio objetivo en GitHub:

Acceso a la Configuración del Repositorio: Navegar al repositorio de GitHub que se desea monitorear. Dentro de la configuración del repositorio, acceder a la sección "Webhooks".

Añadir Nuevo Webhook: Seleccionar la opción para añadir un nuevo webhook.

URL del Payload (Payload URL): Este es el punto más crítico. Se configura la URL pública proporcionada por ngrok que redirige el tráfico hacia la instancia de n8n.



El /webhook/your-n8n-webhook-id es la ruta específica del trigger de webhook configurado en n8n.

Tipo de Contenido (Content type): Se establece como application/json, asegurando que GitHub envíe el payload de los eventos en formato JSON, el cual n8n está optimizado para procesar.

Secreto (Secret): Se puede definir un secreto para firmar las peticiones del webhook. n8n tiene la capacidad de verificar este secreto, añadiendo una capa de seguridad para asegurar que las notificaciones provienen realmente de GitHub.

Eventos a Monitorear :

Para este proyecto, los eventos más relevantes son:

Pushes (eventos de push): Se selecciona para que el flujo se dispare cada vez que se sube nuevo código o se realizan modificaciones en las ramas del repositorio.

Pull requests (eventos de pull request): Opcionalmente, se pueden monitorear para disparar análisis o validaciones adicionales cuando se abre, actualiza o cierra un pull request.

Estado Activo (Active): Asegurar que el webhook esté marcado como activo para que las notificaciones comiencen a enviarse.

Integración y ejecución del Flujo en n8n:

Una vez configurado el webhook en GitHub, el flujo en n8n se configura con un nodo de tipo "Webhook" como su punto de inicio (start node). Este nodo se pone a la escucha en la URL especificada (la misma que ngrok expone). Cuando GitHub envía el payload JSON de un evento (push), el nodo Webhook de n8n lo recibe y dispara el flujo de trabajo.

A partir de este punto, el flujo de n8n utiliza el payload recibido (que contiene información detallada sobre el evento, como el repositorio afectado, la rama, los commits, los archivos modificados) para orquestar las siguientes etapas del sistema, incluyendo:

Clonación del repositorio: Utilizando nodos de ejecución de comandos (Execute Command) o nodos específicos de GitHub en n8n, se clona el repositorio afectado en el volumen compartido (/home/node/shared) accesible por los contenedores de análisis.

Envío de retroalimentación a GitHub: Una vez que los análisis y mejoras se han completado, n8n puede interactuar de nuevo con la API de GitHub para:

Crear un nuevo pull request con el código mejorado y las pruebas generadas por la IA.

Añadir comentarios a issues o pull requests existentes, informando sobre los resultados de los análisis de calidad y seguridad (SonarQube, OWASP ZAP).

Abrir nuevos issues si se detectan vulnerabilidades críticas o problemas de calidad que requieran atención manual.

3.4.3 Integración y ejecución de SonarQube

La integración de SonarQube en el flujo de trabajo automatizado es esencial para la evaluación continua de la calidad, seguridad y mantenibilidad del código fuente. SonarQube se despliega como un servicio contenerizado independiente, interactuando con n8n para la orquestación del análisis y el procesamiento de sus resultados.

Despliegue del Servidor SonarQube y Base de Datos

El servidor de SonarQube y su base de datos asociada se despliegan en contenedores Docker separados, pero interconectados en la misma red (TFG-network) que el contenedor de n8n. Esto garantiza una comunicación eficiente y una gestión de datos robusta. La configuración para estos servicios se define en el archivo docker-compose.yml en el anexo Docker_Sonarqube

3.4.4 Integración y ejecución de OWASP ZAP

La detección de vulnerabilidades de seguridad en tiempo de ejecución es un paso crítico para asegurar la robustez de las aplicaciones. Para ello, se integra OWASP ZAP en el flujo de trabajo automatizado, el cual se encarga de realizar un análisis dinámico de seguridad (DAST) sobre el código. Al igual que SonarQube, ZAP se despliega como un servicio contenerizado para mantener la modularidad y la consistencia del entorno.

Despliegue del Servidor OWASP ZAP

OWASP ZAP se despliega en un contenedor Docker que forma parte de la misma red (TFG-network) que el resto de los servicios principales. Esto asegura la comunicación directa con n8n y permite que ZAP acceda al código de pruebas que se esté analizando.

La configuración del servicio de ZAP se define en el archivo docker-compose.yml de la forma que presenta el anexo Docker_owaszap

Ejecución del Análisis de Seguridad con ZAP a través de n8n

La orquestación del análisis DAST con OWASP ZAP se realiza mediante un flujo de trabajo en n8n, complementando el análisis estático de SonarQube. El proceso general dentro de n8n es el siguiente:

Despliegue de la Aplicación a Testear: Antes de ejecutar ZAP, n8n debe asegurar que la aplicación web esté desplegada y en ejecución en un entorno accesible por ZAP.

Preparación y Lanzamiento del Escaneo con ZAP:

Nodos HTTP Request en n8n: n8n utiliza nodos "HTTP Request" para interactuar con la API REST de ZAP. Esto permite enviar comandos programáticamente para iniciar el escaneo.

Autenticación API: Cada solicitud a la API de ZAP incluirá la api.key configurada, garantizando una comunicación segura.

Definición del Alcance (Target URL): Se le proporciona a ZAP la URL de la aplicación a escanear.

Configuración de Escaneo: n8n puede orquestar diferentes tipos de escaneo:

spider (para descubrimiento de URLs): La API de ZAP permite iniciar un "spider" para mapear la aplicación y descubrir todas sus rutas y puntos de entrada.

active scan (para búsqueda de vulnerabilidades): Una vez que el spider ha completado su trabajo, n8n puede iniciar un escaneo activo, que simula ataques reales para encontrar vulnerabilidades conocidas.

Gestión de Contextos: Se pueden cargar archivos de contexto de ZAP para definir rutas específicas, reglas de autenticación y exclusiones, lo que mejora la precisión y eficiencia del escaneo.

Autenticación en la Aplicación: Si la aplicación a testear requiere autenticación, n8n puede orquestar los pasos para iniciar sesión en ella utilizando la API de ZAP para gestionar sesiones y tokens, asegurando que el escaneo cubra áreas protegidas de la aplicación.

Monitorización y Recuperación de Resultados:

Consulta de la API de ZAP: n8n monitorea el progreso del escaneo de ZAP consultando periódicamente su API REST. Esto permite a n8n saber cuándo el escaneo ha finalizado y obtener los resultados de acuerdo al formato de diseño elegido.

Recuperación de Informes: n8n utiliza la API de ZAP para generar y descargar los informes de vulnerabilidades. Estos informes, generalmente en formato JSON o XML, son fácilmente procesables por n8n.

http://zaproxy:8080/JSON/spider/action/scan/

http://zaproxy:8080/JSON/spider/view/status/

http://zaproxy:8080/JSON/spider/view/results/

http://zaproxy:8080/JSON/alert/view/alerts/

http://zaproxy:8080/JSON/alert/view/alertsSummary/

Procesamiento de Resultados y Retroalimentación:

Análisis del Informe: n8n parsea el informe de ZAP para identificar vulnerabilidades detectadas (SQL Injection, XSS, falta de encabezados de seguridad), su nivel de riesgo y las URLs afectadas.

Toma de Decisiones: Basado en la severidad y tipo de las vulnerabilidades, n8n puede:

Crear automáticamente un nuevo issue en GitHub para cada vulnerabilidad crítica o de alto riesgo, asignando el problema al equipo de desarrollo y proporcionando detalles relevantes.

Comentar en el pull request original si el análisis DAST se lanzó como parte de una validación de PR, alertando sobre los riesgos de seguridad detectados.

Enviar notificaciones a canales de comunicación (Slack, correo electrónico) al equipo de seguridad o desarrollo.

Opcionalmente, pasar información estructurada sobre las vulnerabilidades a los LLMs para que generen soluciones o parches de seguridad, integrando la remediación automática.

La integración de OWASP ZAP en este flujo automatizado proporciona una capa fundamental de seguridad dinámica, detectando vulnerabilidades en el comportamiento real de la aplicación que no siempre son evidentes en el código estático. Esto fortalece la capacidad del sistema para entregar código no solo de alta calidad, sino también robusto y seguro frente a ataques.

3.4.5 Interacción con APIs de LLMs para generación de código y pruebas

La capacidad de generar, refactorizar y probar código de forma inteligente es el núcleo de la mejora continua automatizada que propone este sistema. Esta funcionalidad se logra mediante la interacción programática con diversas APIs de Modelos de Lenguaje Grandes (LLMs), orquestada de manera centralizada a través de n8n. La elección de APIs externas simplifica la infraestructura y permite acceder a los modelos más avanzados sin la complejidad de su despliegue y mantenimiento local.

LLMs Utilizados para la Integración

Para las pruebas y la integración en este Trabajo Fin de Grado, se han explorado y utilizado los siguientes LLMs, interactuando con ellos a través de los nodos HTTP o nodos específicos de IA de n8n:

deepseek-r1distill-llama-70b: Un modelo de gran tamaño, conocido por su capacidad de razonamiento y generación de código, que puede ofrecer resultados detallados y coherentes en tareas de refactorización y prueba.

models/gemini-2.0-flash-thinking-exp-1219: Una versión experimental de los modelos Gemini de Google, optimizada para velocidad y eficiencia, lo que la hace adecuada para procesos automatizados donde la latencia es un factor. Su capacidad de "pensamiento" experimental apunta a una mejor comprensión de contextos complejos.

models/gemma-3.2-27-it: Un modelo de la familia Gemma de Google, que es más ligero y eficiente, pero aún capaz de realizar tareas de codificación. Su naturaleza "instruction-tuned" (IT) lo hace particularmente apto para seguir instrucciones precisas en la generación de código y pruebas.

Flujos de Interacción con LLMs en n8n

Dentro de los flujos de trabajo de n8n, la interacción con estos LLMs se estructura en sub-flujos dedicados que encapsulan las tareas específicas de análisis y generación de código. A cada uno de estos sub-flujos se le solicita al LLM una serie de valoraciones y propuestas, alimentadas por el código fuente obtenido del repositorio de GitHub.

Revisión Sintáctica del Código:

Contexto: El LLM recibe un fragmento de código y se le pide que realice una revisión sintáctica exhaustiva.

Propósito: Identificar errores de sintaxis, inconsistencias con los estándares del lenguaje (Java en este caso) o malas prácticas de formato. La respuesta del LLM incluye observaciones y correcciones sugeridas a nivel de estructura del lenguaje.

Valoración Funcional del Código:

Contexto: Se proporciona al LLM el código fuente y una prompt detallada con las características de comportamiento del LLM y la estructura del resultado en formato JSON con las diferentes versiones del código resultante, corregido, revisado y pruebas Junit.

Propósito: El LLM realiza una valoración desde el punto de vista funcional. Esto implica no solo identificar posibles bugs lógicos o ineficiencias, sino también evaluar si el código cumple con su propósito declarado, si es legible, si se ajusta a los patrones de diseño comunes, y si es susceptible de mejoras en su lógica o rendimiento.

Propuesta de Nueva Versión de Código:

Propósito: Esta nueva versión busca corregir los problemas identificados, refactorizar el código para mejorar su calidad y mantenibilidad, o incluso añadir nuevas funcionalidades básicas si se ha detectado alguna omisión importante. Los prompts se diseñan para guiar al LLM a generar código que sea funcionalmente equivalente o mejorado, y que cumpla con las buenas prácticas de codificación.

Propuesta de Código para Pruebas Unitarias (JUnit):

Contexto: El LLM recibe la versión original del código, junto con la información sobre la funcionalidad esperada y, si es posible, casos de uso específicos.

Propósito: Se le encarga al LLM que proponga código para pruebas unitarias utilizando el framework JUnit. Estas pruebas están diseñadas para verificar la correcta funcionalidad del código, asegurar que los cambios no introducen regresiones y validar las correcciones o mejoras realizadas. El LLM genera los

casos de prueba, incluyendo las aserciones necesarias para validar el comportamiento esperado de las funciones o métodos.

n8n gestiona esta interacción a través de nodos HTTP Request que se conectan a los endpoints de las APIs de cada LLM. Los prompts se construyen dinámicamente utilizando datos de los nodos previos del flujo. Las respuestas de los LLMs, que contienen el código generado o las valoraciones, son entonces parseadas por n8n y utilizadas en los siguientes pasos del flujo, como la compilación y ejecución de las pruebas, o la preparación de un pull request en GitHub. Esta modularidad y la capacidad de integrar múltiples LLMs permiten al sistema aprovechar las fortalezas de cada modelo para diferentes aspectos de la mejora continua del código.

3.4.6 Compilación y ejecución de pruebas JUnit

El último eslabón en el flujo de mejora continua automatizada es la validación de las mejoras propuestas mediante la compilación y ejecución de pruebas unitarias. Esta fase es vital para asegurar que el código generado o refactorizado por los LLMs no solo cumple con las expectativas funcionales y de calidad, sino que también es sintácticamente correcto y no introduce regresiones.

Entorno de Ejecución Contenerizado

La compilación del código Java y la ejecución de las pruebas JUnit se llevan a cabo dentro de un entorno Dockerizado y aislado. Esto garantiza un ambiente consistente y reproducible, independientemente del sistema operativo anfitrión. Para ello, se utiliza un contenedor basado en una imagen de Java (openjdk:17-jdk) que incluye el Java Development Kit (JDK) y las herramientas necesarias para construir proyectos Java, como Maven o Gradle (no utilizadas en este TFG).

El código fuente modificado por el LLM y las pruebas JUnit generadas por los mismos LLMs se montan en este contenedor de ejecución a través de un volumen compartido, /home/userAGL/TFG/data/code/java. Esto permite que el contenedor acceda directamente a los archivos necesarios para la compilación y ejecución.

Orquestación y Ejecución con n8n

El proceso de compilación y ejecución de pruebas se realiza mediante una acción ssh de n8n utilizando su nodo "Execute Command" para interactuar con el contenedor de ejecución. El flujo de trabajo en n8n sigue estos pasos clave:

Recepción del Código Generado: Tras la interacción con los LLMs n8n recibe el código fuente propuesto y el código de las pruebas JUnit específicas para ese código.

Compilación del Código:

n8n lanza un comando ssh exec en el contenedor de ejecución de Java.

El comando invoca la herramienta de construcción, javac, para compilar el código fuente Java y las pruebas JUnit. Se monitorea la salida para detectar errores de compilación.

Ejecución de las Pruebas JUnit:

Una vez que la compilación es exitosa, n8n lanza otro comando (ej., mvn test o un comando directo de JUnit) para ejecutar las pruebas.

Estas pruebas se ejecutan con la "nueva versión de código" generada por el LLM y con las propias pruebas JUnit diseñadas por cada agente IA-LLM (deepseek-r1distill-llama-70b, models/gemini-2.0-flash-thinking-exp-1219, models/gemma-3.2-27-it). Esto permite una validación directa de la calidad y corrección de las propuestas de la IA.

```
/home/userAGL/Oracle_JDK-23/bin/java -jar /home/userAGL/Oracle_JDK-23/lib/junit-platform-console-standalone-1.10.0.jar --class-path ".:/home/userAGL/Oracle_JDK-23/lib/*:./" --scan-class-path \ --include-classname [Execute node ‘Read/Write Files from Disk1’ for preview]
```

```
cd /home/userAGL/TFG/data/code/java/corregido && pwd && /home/userAGL/Oracle_JDK-23/bin/javac -cp .:/home/userAGL/Oracle_JDK-23/lib/*{{$json.fileName.replace('/home/node/shared','/home/userAGL/TFG/data') }}
```

n8n captura la salida de la ejecución de las pruebas JUnit para analizar los resultados. Utiliza un nodo de n8n para consolidar la información en informes en Google Sheets.

Análisis de Resultados y Retroalimentación

Tras la ejecución de las pruebas, n8n procesa los resultados:

Verificación de Aserciones: Se comprueba si todas las pruebas pasaron exitosamente. La presencia de fallos indica que el código mejorado o las propias pruebas generadas por el LLM no son correctas o completas.

Decisiones del Flujo:

Si todas las pruebas pasan, el sistema considera que la mejora propuesta por el LLM es válida y puede proceder a la fase de integración final, generar un pull request en GitHub con el código y las pruebas aceptadas.

Si las pruebas fallan, n8n registra un issue en GitHub, añade un comentario al pull request original con los detalles de los fallos.

Esta fase de compilación y ejecución automatizada de pruebas es el mecanismo de control de calidad final del ciclo de mejora. Asegura que las propuestas de los LLMs no solo son teóricamente correctas, sino que también funcionan en la práctica, proporcionando una base sólida para la integración de código asistida por inteligencia artificial.

3.5 Flujo completo de mejora continua del código

El sistema propuesto materializa un ciclo de mejora continua del código altamente automatizado, diseñado para operar de forma proactiva desde la detección de cambios hasta la propuesta de soluciones validadas. Este flujo integra diversas herramientas especializadas, orquestadas por n8n, para garantizar la calidad, seguridad y eficiencia del código fuente.

El ciclo se inicia con un disparador reactivo y se desarrolla en varias fases interconectadas:

Inicio del Flujo (Disparador de GitHub):

El punto de entrada del flujo es un webhook configurado en GitHub. Cada vez que un desarrollador realiza un push a una rama o abre un pull request en el repositorio de código, GitHub envía un payload JSON a una URL específica. Para que n8n, desplegado en un entorno privado o detrás de un firewall, pueda recibir estas notificaciones, se utiliza ngrok. ngrok establece un túnel seguro desde una URL pública de internet hacia el puerto local donde n8n está escuchando. Este evento es capturado por el nodo "Webhook" de n8n, que inicia la ejecución de todo el flujo de trabajo automatizado.

Fase de Análisis Inicial (Calidad y Seguridad):

Una vez activado, n8n toma el control y orquesta las primeras fases de análisis:

Clonación del Repositorio: n8n primero clona el repositorio de GitHub con el código fuente más reciente en un volumen compartido dentro del entorno Docker (/home/userAGL/TFG/data/code/java), haciéndolo accesible para las herramientas de análisis.

Análisis Estático con SonarQube: Se invoca el sonar-scanner CLI dentro de un contenedor Docker persistente integrado en el mismo Docker que el servidor SonarQube. Este escáner analiza el código fuente en busca de bugs, vulnerabilidades, code smells y deudas técnicas. n8n espera la finalización del análisis y, consulta la Quality Gate de SonarQube. Si la Quality Gate no pasa, el sistema puede generar una notificación o un issue en GitHub, aunque el flujo principal puede continuar para ofrecer mejoras incluso en estos casos.

Análisis Dinámico con OWASP ZAP: Paralelamente, n8n orquesta el despliegue de la aplicación en un entorno de pruebas y lanza un escaneo DAST utilizando la API de OWASP ZAP. ZAP simula ataques reales para identificar vulnerabilidades que solo se manifiestan en tiempo de ejecución (fallos de autenticación, errores de configuración del servidor web, inyecciones XSS o SQL). n8n recupera los informes de ZAP y los integra por medio de un nodo n8n en Google Sheets.

Fase de Generación Inteligente (LLMs):

Los resultados de los análisis de calidad y seguridad y el código fuente, n8n interactúa con las APIs de los LLMs. Aquí, se construyen prompts dinámicos que contienen el código original, los hallazgos relevantes de SonarQube y ZAP, y las instrucciones específicas para los LLMs. Se utilizan los siguientes modelos para estas tareas: deepseek-r1distill-llama-70b, models/gemini-2.0-flash-thinking-exp-1219 y models/gemma-3.2-27-it. A cada uno de estos modelos se les solicita realizar las siguientes tareas críticas:

Revisión Sintáctica: Evaluar la corrección y estilo del código desde una perspectiva sintáctica.

Valoración Funcional: Proporcionar una evaluación del código desde el punto de vista de su lógica y funcionalidad.

Propuesta de Nueva Versión de Código: Generar una versión mejorada o refactorizada del código fuente, abordando los problemas detectados por el análisis.

Generación de Pruebas Unitarias (JUnit): Crear código de pruebas unitarias específico para la funcionalidad del código modificado, utilizando el framework JUnit.

Fase de Validación Automatizada:

Una vez obtenidas las propuestas de código mejorado y las pruebas unitarias de los LLMs, el sistema procede a la validación automatizada:

Compilación y Ejecución de Pruebas: n8n escribe el código mejorado y las pruebas JUnit generadas por el LLM en el volumen compartido. Luego, lanza comandos dentro de un contenedor Docker de ejecución Java (con JUnit) para compilar el código y ejecutar las pruebas. Es fundamental destacar que estas pruebas se realizan con la versión de código revisado y mejorado, y utilizando las pruebas JUnit diseñadas por cada uno de los agentes IA-LLM que han participado en el proceso.

Análisis de Resultados: n8n monitorea la salida de la ejecución de pruebas para determinar si todas las aserciones pasaron con éxito.

Fase de Retroalimentación y Supervisión Humana:

El ciclo culmina con la retroalimentación a los desarrolladores y la integración controlada:

Creación de Pull Requests Automatizados: Si las pruebas JUnit generadas por la IA pasan con éxito en el código mejorado, n8n crea automáticamente un Pull Request (PR) en GitHub. Este PR contiene el código original, la propuesta de mejora del LLM, y las pruebas JUnit que validan estos cambios. Opcionalmente, el PR se complementa con comentarios automatizados que resumen los hallazgos de SonarQube y ZAP, así como los resultados de la ejecución de las pruebas.

Mecanismo de Supervisión Humana: Este PR actúa como el mecanismo de supervisión humana. Los desarrolladores pueden revisar los cambios propuestos por la IA, evaluando su calidad, su coherencia con los estándares internos no codificados, y su impacto funcional. Un revisor humano debe aprobar explícitamente el PR para que los cambios se fusionen en la rama principal, manteniendo el control final sobre la base de código.

Este flujo completo, desde la detección de cambios hasta la propuesta de soluciones validadas y la supervisión humana, establece un sistema robusto y eficiente para la mejora continua y automatizada de la calidad y seguridad del código, liberando a los desarrolladores de tareas repetitivas y permitiéndoles centrarse en la innovación y el diseño.

3.6 Validación funcional del sistema

La validación funcional del sistema propuesto es una etapa crítica para demostrar la operatividad y la eficacia de la arquitectura y los flujos de trabajo diseñados. El objetivo principal de esta validación es confirmar que el sistema funciona de extremo a extremo, que cada una de sus fases cumple con su propósito específico, que la interacción entre los componentes es fluida, y que el ciclo de mejora continua del código se completa exitosamente desde la detección de un cambio hasta la propuesta validada de una mejora.

Enfoque de la Validación

La validación se llevó a cabo mediante pruebas de integración de extremo a extremo que simularon el comportamiento real del sistema en un entorno controlado. Esto implicó:

Configuración completa: Despliegue de todos los servicios (n8n, SonarQube, OWASP ZAP, ngrok, base de datos) en sus respectivos contenedores Docker, interconectados a través de la red TFG-network.

Webhooks activos: Configuración de los webhooks de GitHub apuntando a la URL de ngrok, asegurando que los eventos de push o pull request fueran correctamente recibidos por n8n.

Ejecución de flujos completos: Se ejecutaron los flujos de trabajo de n8n de manera integral, abarcando todas las fases: desde la recepción del evento de GitHub, pasando por los análisis de calidad y seguridad, la interacción con los LLMs, la compilación y ejecución de pruebas, hasta la generación de pull requests.

Casos de Prueba y Resultados Operacionales

Se diseñaron y ejecutaron varios escenarios de prueba para confirmar la operatividad de las funcionalidades clave del sistema. A continuación, se detallan ejemplos representativos de los casos probados, enfocándose en la confirmación de la ejecución exitosa de cada fase:

Caso de Prueba 1: Confirmación del Flujo de Detección y Propuesta de Mejora Estática

Escenario: Se introdujo intencionadamente code smell, un método excesivamente largo o con excesiva complejidad logarítmica en una clase Java simple dentro de un repositorio de prueba.

Acción de Disparo: Se realizó un git push a la rama main del repositorio en GitHub.

Resultados Operacionales Confirmados:

- n8n recibió el webhook y clonó el repositorio correctamente en el volumen compartido.
- El análisis de sonar-scanner se inició y completó, detectando el code smell según lo esperado y enviando los resultados al servidor SonarQube.
- n8n procesó los hallazgos de SonarQube y, basándose en estos, realizó un informe detallado que se integró en una cuenta de Google Sheets.
- El LLM respondió con una propuesta de código refactorizado y un conjunto de pruebas JUnit.
- El sistema orquestó la compilación del código refactorizado y la ejecución de las pruebas JUnit generadas por el LLM en el contenedor de ejecución. Las pruebas se ejecutaron exitosamente.
- Finalmente, n8n generó un pull request en GitHub, presentando la propuesta de mejora y las pruebas asociadas.

Caso de Prueba 2: Confirmación del Flujo de Detección de Seguridad y Propuesta mejorada

Escenario: Se utilizó un fragmento de código Java que exponía una vulnerabilidad simple -un encabezado de seguridad inexistente en una aplicación web básica- y se desplegó una instancia de la aplicación.

Acción de Disparo: Se realizó un git push del código a GitHub, y se aseguró el despliegue de la aplicación de prueba accesible por ZAP.

Resultados Operacionales Confirmados:

- n8n inició el flujo, y orquestó el despliegue y la accesibilidad de la aplicación de prueba.
- OWASP ZAP se conectó a la aplicación y realizó el escaneo dinámico, detectando la vulnerabilidad de seguridad.
- n8n recuperó el informe de vulnerabilidades de ZAP y lo utilizó para generar un informe detallado que se integró en una cuenta de Google Sheets.
- El LLM propuso una modificación de código para corregir la vulnerabilidad.
- El código corregido fue compilado y las pruebas unitarias JUnit se ejecutaron, confirmando la corrección a nivel funcional.
- Se creó un pull request en GitHub con la solución propuesta y los detalles de la vulnerabilidad.

Caso de Prueba 3: Confirmación de la Generación de Pruebas Unitarias para Nuevo Código

Escenario: Se añadió un nuevo método en un archivo Java sin ninguna prueba unitaria asociada.

Acción de Disparo: Se realizó un git push de este nuevo código a GitHub.

Resultados Operacionales Confirmados:

- n8n activó el flujo y procesó el código.
- Se envió el nuevo método a un LLM.
- El LLM generó un conjunto de pruebas JUnit específicas para la funcionalidad del nuevo método.
- Las nuevas pruebas JUnit fueron compiladas y ejecutadas con éxito junto con el código, demostrando su validez.
- Finalmente, se generó un pull request en GitHub que incluía la adición de las pruebas JUnit al repositorio.

Conclusión de la Validación Operacional

La validación funcional confirmó la plena operatividad del sistema de mejora continua de código automatizada. Se demostró que todos los componentes (GitHub, ngrok, n8n, SonarQube, OWASP ZAP, LLMs, entornos de ejecución Docker) interactúan de forma efectiva y que el flujo de trabajo completo, desde la detección de eventos hasta la generación de pull requests con código y pruebas validadas, se ejecuta de manera consistente. Esta fase de validación establece la base empírica de que el sistema cumple con sus objetivos de automatización y orquestación.

3.7 Conclusiones técnicas de la implementación

3.7.1 Análisis de los resultados de calidad y seguridad

La implementación del sistema automatizado permitió realizar un análisis sistemático de la calidad del código y la seguridad de la aplicación, utilizando SonarQube y OWASP ZAP, respectivamente. Los resultados obtenidos en las pruebas de validación operacional proporcionaron una base para evaluar la

efectividad de estas herramientas como puntos de partida para la intervención de los LLMs.

3.7.1.1 Resultados del Análisis de Calidad con SonarQube

Para evaluar la calidad estática del código y la deuda técnica, se realizó un análisis exhaustivo del proyecto de prueba “hola-mundo” utilizando SonarQube a través de SonarScanner. El análisis se completó exitosamente, permitiendo a SonarQube construir el AST (Árbol de Sintaxis Abstracto) del código e identificar una variedad de problemas de calidad y seguridad. Este informe se detalla en el anexo informe sonarqube:

3.7.1.2 Resultados del Análisis de Seguridad con OWASP ZAP

El análisis dinámico de seguridad (DAST) de la aplicación, realizado con OWASP ZAP, proporcionó un panorama detallado de las vulnerabilidades presentes. El alertsSummary arrojó la siguiente distribución de riesgos:

Riesgos Altos: 0

Riesgos Medios: 90

Riesgos Bajos: 121

Informacionales: 23

Esta distribución indica que, si bien no se detectaron vulnerabilidades de riesgo "Alto", existe una cantidad considerable de incidencias de riesgo "Medio" y "Bajo" que requieren atención. A continuación, se detallan algunos ejemplos clave de las advertencias y soluciones propuestas por ZAP, que serían cruciales para la fase de generación de código por los LLMs: y descritos en el anexo Informe OWAS ZAP.

4 Resultados y conclusiones

4.1 Evaluación de los objetivos alcanzados

A lo largo de este Trabajo Fin de Grado se ha logrado cumplir de manera efectiva el objetivo principal: diseñar e implementar un sistema automatizado de mejora continua del código, integrado en un flujo de trabajo reproducible y adaptable a entornos reales de desarrollo.

Los objetivos específicos como la automatización del análisis estático, la detección de vulnerabilidades en tiempo de ejecución, la generación de pruebas automatizadas mediante LLMs y la orquestación de estos procesos mediante flujos definidos en n8n, se han alcanzado satisfactoriamente. El sistema desarrollado permite ejecutar tareas clave del ciclo de vida del software sin intervención manual, y con resultados claros y trazables.

4.2 Lecciones aprendidas

El trabajo con cada una de las herramientas ha aportado aprendizajes concretos, tanto técnicos como metodológicos:

n8n ha demostrado ser una herramienta versátil para construir flujos automatizados sin necesidad de desarrollar infraestructura personalizada. Su facilidad para conectar APIs y manejar flujos condicionales ha sido clave en la orquestación de los procesos.

SonarQube, aunque requiere una configuración inicial algo exigente (reglas de calidad, perfiles, autenticación con tokens), ha resultado ser una solución robusta para análisis estático. La integración vía CLI y API REST es directa, y los informes generados son muy útiles para identificar deuda técnica y fomentar buenas prácticas.

OWASP ZAP ha supuesto un reto interesante por su orientación a análisis dinámico. Integrarlo en el pipeline ha requerido definir condiciones específicas de escaneo, pero su capacidad para detectar vulnerabilidades explotables en tiempo de ejecución aporta un valor fundamental que complementa al análisis estático.

Los LLMs han resultado una herramienta sorprendentemente potente en la generación de pruebas unitarias y sugerencias de refactorización. La experiencia demuestra que, si se emplean con control, son un gran aliado en fases de validación y mejora del código.

GitHub y JUnit han servido como base para la integración del sistema en flujos típicos de desarrollo, asegurando que la solución es compatible con prácticas estándar de control de versiones, pruebas y colaboración entre equipos.

Docker ha permitido aislar los entornos de ejecución de cada componente del sistema, facilitando el despliegue local y remoto, y asegurando consistencia entre entornos de desarrollo y producción.

4.3 Limitaciones del sistema desarrollado

Aunque el sistema cumple su función de forma satisfactoria, se han identificado varias limitaciones técnicas y operativas que deben ser consideradas para futuras iteraciones o despliegues en entornos más exigentes:

- Uso de LLMs con modelos comerciales: La integración de modelos de lenguaje requiere el uso de APIs que operan bajo licencias comerciales, lo que implica la necesidad de contar con tokens de acceso y modelos de facturación adecuados. Esto limita el uso libre y sostenido del sistema, y plantea desafíos en proyectos de código abierto o con recursos limitados.

- Entorno anfitrión Windows y WSL2: Aunque WSL2 permite ejecutar herramientas Unix en entornos Windows con una integración razonablemente buena, se han detectado ciertas limitaciones importantes que han impactado en el desarrollo del sistema. Entre ellas, destacan las solicitudes de reinicios frecuentes del sistema operativo anfitrión, que provocan la interrupción de servicios críticos como contenedores Docker, procesos de análisis o flujos automatizados en n8n.

- Curva de configuración inicial elevada: La integración completa de herramientas como n8n, SonarQube, OWASP ZAP, LLMs y sistemas de testing requiere un conocimiento técnico intermedio-avanzado, especialmente en lo relativo a redes, contenedores, automatización de flujos y seguridad. Esto puede suponer una barrera de entrada para equipos pequeños o con menor experiencia en DevOps.

- Escalabilidad y rendimiento: El sistema ha sido probado en entornos locales y controlados. En contextos con múltiples desarrolladores, repositorios grandes o despliegues concurrentes, podrían surgir problemas de escalabilidad y rendimiento que requieren optimización de recursos, paralelización de procesos o distribución del pipeline en nodos separados.

4.4 Líneas futuras de mejora y ampliación

Como continuación de este trabajo, se proponen las siguientes líneas de evolución:

- Infraestructura dedicada (on-premise o en la nube): Una de las mejoras prioritarias consiste en desplegar el sistema sobre una infraestructura más estable y dimensionada adecuadamente, ya sea en servidores locales o en entornos cloud. Esto permitiría superar las limitaciones del entorno de desarrollo actual, garantizar la disponibilidad continua de los servicios, aumentar la capacidad de procesamiento y facilitar el manejo de múltiples proyectos en paralelo.
- Gestión avanzada de tokens y licencias de LLMs: En una futura iteración, se podría automatizar el control del consumo de tokens de los modelos de lenguaje y evaluar alternativas open-source o autoalojadas que reduzcan la dependencia de servicios externos.
- Mejorar la segmentación del código enviado a los LLMs, utilizando análisis sintáctico y semántico para reducir el ruido y mejorar la precisión de las respuestas.
- Extender el sistema a otros lenguajes de programación, actualmente está diseñado para entornos Java.

- Evaluar herramientas alternativas de análisis dinámico y modelos LLM autoalojados, para reducir dependencia de servicios externos y mejorar el cumplimiento normativo.

4.5 Aplicabilidad en entornos CI/CD reales

La principal conclusión técnica del proyecto es que la construcción de un pipeline automatizado de análisis y mejora del código es viable, escalable y aplicable en entornos reales de integración continua.

Este tipo de solución permite:

- Detectar errores y vulnerabilidades antes de que lleguen a producción.
- Fomentar buenas prácticas de desarrollo.
- Reducir la deuda técnica de forma sostenible.
- Integrar inteligencia artificial de forma responsable y productiva.

Además, se alinea con prácticas modernas de ingeniería de software, reduciendo la carga manual de los equipos de Quality Assurance QA y facilitando la colaboración multidisciplinar.

5 Análisis de Impacto

Esta sección evalúa el impacto del uso de la inteligencia artificial en el desarrollo del proyecto, centrándose en las implicaciones medioambientales y los riesgos de privacidad asociados al tratamiento de datos.

5.1 Evaluación del impacto ambiental del uso de IA

El uso de modelos de inteligencia artificial conlleva un consumo energético significativo, generando una huella ambiental que debe ser considerada.

5.1.1 Consumo energético de los modelos de lenguaje

Los Grandes Modelos de Lenguaje (LLMs) utilizados en este proyecto demandan una considerable cantidad de energía. El entrenamiento de modelos avanzados requiere miles de horas de procesamiento en hardware especializado (GPUs), consumiendo megavatios-hora. La fase de inferencia también implica un consumo energético, aunque menor que el entrenamiento, que se acumula con cada solicitud. La escala de este consumo depende directamente del tamaño y complejidad del modelo, así como del volumen de solicitudes.

Según un reportaje titulado *"La sed de ChatGPT" de National Geographic España. La cantidad de agua que consume la IA es alarmante"*, se estima que generar un texto de 100 palabras con ChatGPT consume, en promedio, 519 mililitros de agua, equivalente a una botella estándar. Este consumo se debe principalmente al uso de agua en los sistemas de refrigeración de los centros de datos donde operan estos modelos de inteligencia artificial.[13]

5.1.2 Consideraciones de eficiencia en la arquitectura

Docker: Los contenedores son más ligeros que las máquinas virtuales, lo que implica un menor overhead energético.

n8n: Como orquestador, n8n funciona por eventos (webhooks de GitHub). Esto es más eficiente que el polling constante.

Modularidad: La modularidad de n8n es una ventaja. Permite ejecutar solo los módulos necesarios según el cambio o el tipo de análisis requerido, evitando el consumo de ejecutar "todo" siempre.

5.1.3 Medidas de mitigación

Para mitigar el consumo energético, se están adoptando soluciones de hardware más eficientes. El uso de unidades de procesamiento gráfico (GPUs) y unidades de procesamiento tensorial (TPUs) especializadas puede reducir significativamente el consumo de energía en comparación con las unidades de procesamiento central (CPUs) tradicionales.[14]

La refrigeración de los centros de datos representa una parte significativa de su consumo energético. Se están desarrollando tecnologías de refrigeración más eficientes, como la refrigeración líquida directa al chip y sistemas de aire avanzados, que pueden reducir el consumo de energía en un 20-50%.[14]

Algunos países están implementando regulaciones para controlar el crecimiento de los centros de datos y su impacto ambiental. Por ejemplo, Irlanda y los Países Bajos han pausado temporalmente la construcción de nuevos centros de datos para evaluar su sostenibilidad y consumo de recursos.[15]

5.2 Análisis de riesgos de privacidad en el tratamiento de datos

Es fundamental evaluar los riesgos asociados al tratamiento de datos, especialmente al involucrar servicios en la nube y modelos de lenguaje.

5.2.1 Tipos de datos tratados y posibles riesgos

Código Fuente y propiedad Intelectual: El riesgo principal es la fuga de algoritmos secretos, lógica de negocio o código propietario si se envía a una IA externa.

Credenciales/Secrets: Riesgo alto de que los desarrolladores comprometan accidentalmente claves API, contraseñas o tokens en el código y posteriormente se envíen a una IA. Estas credenciales podrían quedar expuestas.

Datos Personales: El código podría contener datos personales, con el riesgo que esto implica de incumplimiento normativo.

Metadatos de GitHub: Nombres de usuario, correos, mensajes de commit, discusiones en Pull Requests. Aunque menos sensible que el código, puede revelar información sobre los desarrolladores y el proyecto.

Datos de la IA: Las respuestas de la IA son datos a proteger.

El proveedor de IA podría usar el código para entrenar sus modelos.

5.2.2 Normativas aplicables y cumplimiento. GDPR.

Reglamento General de Protección de Datos.

Si se opera en la UE o se procesa código que pueda contener datos de carácter personal de ciudadanos de la UE, se debe dar cumplimiento al GDPR.

- Minimización de datos (Art. 5.1.c): solo se trata la información estrictamente necesaria para la operación de cada análisis.
- Limitación de propósito (Art. 5.1.b): los datos se usan únicamente con fines de mejora de calidad y seguridad del código.
- Derecho al olvido: el sistema permite borrar o no almacenar de forma permanente ningún dato tras la finalización del análisis.
- Transferencia internacional: en caso de uso de LLMs de terceros (como OpenAI), se evalúa si la transferencia de datos cumple con cláusulas contractuales estándar aprobadas por la Comisión Europea.
- Además, el sistema promueve el cumplimiento de principios de privacy-by-design, asegurando desde la fase de diseño arquitectónico que la privacidad es prioritaria.

Este sistema contribuye al cumplimiento de los Objetivos de Desarrollo Sostenible de la ONU, en particular al **ODS 9** (Industria, innovación e infraestructura) y al **ODS 12** (Producción y consumo responsables), al promover prácticas seguras, responsables y sostenibles en el desarrollo de software.

6 Bibliografía

[1] «n8n Community - Connect, Learn, and Share Automation Insights - n8n», n8n Community. Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://community.n8n.io/?utm_source=n8n_app&utm_medium=app_sidebar

[2] «CI/CD», *Wikipedia, la enciclopedia libre*. 25 de julio de 2023. Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=CI/CD&oldid=152668077

[3] «GitHub credentials | n8n Docs». Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://docs.n8n.io/integrations/builtin/credentials/github/?utm_source=n8n_app&utm_medium=credential_settings&utm_campaign=create_new_credentials_modal

[4] «GitHub.com Documentación de ayuda», GitHub Docs. Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://docs-internal.github.com/es

[5] «SonarQube for IntelliJ Documentation». Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://docs.sonarsource.com/sonarqube-for-ide/intellij/

[6] «ZAP – Automate ZAP». Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://www.zaproxy.org/docs/automate/

[7] «OWASP Top 10», *Wikipedia, la enciclopedia libre*. 19 de enero de 2025. Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=OWASP_Top_10&oldid=164864642

[8] «OpenAI Cookbook». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://cookbook.openai.com/

[9] «Modelos de Gemini | Gemini API», Google AI for Developers. Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://ai.google.dev/gemini-api/docs/models?hl=es-419

[10] «Ollama». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://ollama.com

[11] «Docker Hub Container Image Library | App Containerization». Accedido: 2 de junio de 2025. [En línea]. Disponible en: https://hub.docker.com

[12] «Configuration File | ngrok documentation». Accedido: 14 de abril de 2025. [En línea]. Disponible en: https://ngrok.com/docs/agent/config/

[13] «La sed de ChatGPT: la IA consume una cantidad de agua alarmante», National Geographic España. Accedido: 1 de junio de 2025. [En línea]. Disponible en: https://www.nationalgeographic.com.es/ciencia/agua-que-gasta-chatgpt-y-otros-modelos-ia_23812

[14] A. Shebila, «Powering Generative AI: The Data Center Energy Challenge and Sustainable Solutions», EDGE DC. Accedido: 1 de junio de 2025. [En línea]. Disponible en: https://edge.id/articles/powering-generative-ai-the-data-center-energy-challenge-and-sustainable-solutions

[15] «As generative AI asks for more power, data centers seek more reliable, cleaner energy solutions», Deloitte Insights. Accedido: 1 de junio de 2025. [En línea]. Disponible en: https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2025/genai-power-consumption-creates-need-for-more-sustainable-data-centers.html

7 Anexos

Docker_n8n

```
services:
    n8n:
        stdin_open: true
        tty: true
        container_name: n8n
        ports:
            - 5678:5678
        volumes:
            - /home/userAGL/TFG/n8n_files:/home/node/.n8n
            - /home/userAGL/TFG/data:/home/node/shared
            - /usr/bin:/home/node/host
            - /mnt/wslg:/mnt/wslg  # Montar PulseAudio desde WSLg
        image: docker.n8n.io/n8nio/n8n
#        extra_hosts:
#            - "n8n.local:host-gateway"
        environment:
             - N8N_HOST=n8n.local           # Escucha en todas las interfaces
             - N8N_SECURE_COOKIE=false #Disable SSL
             - GENERIC_TIMEZONE=Europe/Madrid
             - TZ=Europe/Madrid
             -   PULSE_SERVER=unix:/mnt/wslg/PulseServer      #   Configurar
PulseAudio en el contenedor
             -  N8N_COMMUNITY_PACKAGES_ALLOW_TOOL_USAGE=true  #  Permite  usar
n8n-nodes-mcp
networks:
  default:
     name: TFG-network
     external: true
```

La sección services define los contenedores que componen el entorno. En este caso, se configura un único servicio llamado n8n:

stdin_open: true y tty: true:

Estas directivas son comunes para contenedores que requieren una interfaz interactiva o pseudo-TTY, lo que puede ser útil para la depuración o el funcionamiento interno de n8n, aunque no siempre son estrictamente necesarias para la ejecución en modo servicio.

container_name: n8n:

Asigna un nombre específico al contenedor de n8n, facilitando su identificación y gestión dentro del entorno Docker.

ports: - 5678:5678:

Mapea el puerto 5678 del contenedor (donde n8n escucha por defecto) al puerto 5678 del host. Esto permite acceder a la interfaz de usuario web de n8n desde el navegador a través de http://localhost:5678 o la IP del servidor.

volumes:

Esta sección es crucial para la persistencia de datos y el acceso a recursos del host o compartidos:

- /home/userAGL/TFG/n8n_files:/home/node/.n8n: Monta el directorio /home/userAGL/TFG/n8n_files del host en /home/node/.n8n dentro del contenedor. Este volumen es esencial para la persistencia de la configuración de n8n, las credenciales, los flujos de trabajo (workflows) y los datos internos de la aplicación, asegurando que los cambios y el estado de n8n se mantengan entre reinicios del contenedor.

- /home/userAGL/TFG/data:/home/node/shared: Monta el directorio /home/userAGL/TFG/data del host en /home/node/shared dentro del contenedor. Este volumen está diseñado para ser un espacio de trabajo compartido donde se descargará el código fuente del repositorio de GitHub y donde las diferentes herramientas (SonarQube, OWASP ZAP, entorno de compilación) podrán acceder al mismo código para su análisis y procesamiento.

- /usr/bin:/home/node/host: Monta el directorio /usr/bin del host en /home/node/host dentro del contenedor. Esto permite al contenedor de n8n acceder a binarios del sistema operativo del host, lo cual es necesario para ejecutar comandos externos (como git, sonar-scanner, docker CLI si el n8n interactúa con el demonio de Docker) directamente desde los nodos "Execute Command" de n8n.

image: docker.n8n.io/n8nio/n8n: Especifica la imagen Docker oficial de n8n que se utilizará para crear el contenedor, asegurando el uso de una versión estable y mantenida por los desarrolladores de n8n.

environment: Define variables de entorno para configurar el comportamiento de n8n:

- N8N_HOST=n8n.local: Configura la dirección IP o el nombre de host en el que n8n escuchará. n8n.local será un nombre de host interno o un alias que facilite la comunicación dentro de la red Docker.

- N8N_SECURE_COOKIE=false: Deshabilita el uso de cookies seguras (requiere HTTPS). Para entornos de desarrollo o pruebas donde HTTPS no está configurado (especialmente si ngrok maneja la seguridad SSL externa), esta configuración es necesaria.

- GENERIC_TIMEZONE=Europe/Madrid y - TZ=Europe/Madrid: Establecen la zona horaria del contenedor a Europe/Madrid, lo que asegura la consistencia en el registro de tiempo de los eventos y las operaciones de n8n.

- N8N_COMMUNITY_PACKAGES_ALLOW_TOOL_USAGE=true: Permite el uso de nodos de paquetes de la comunidad de n8n que pueden invocar herramientas externas, lo cual es vital para la flexibilidad y extensibilidad de los flujos de trabajo.

networks: Define la red a la que se conectará el contenedor.

default: name: TFG-network external: true: Especifica que el contenedor se conectará a una red Docker ya existente y llamada TFG-network. Esta red externa es fundamental para permitir que n8n se comunique con otros contenedores del sistema (como SonarQube Server, OWASP ZAP) que también estarán conectados a esta misma red, facilitando la interconexión entre todos los servicios contenerizados del proyecto.

Configuración de ngrok para Acceso Externo:

Aunque no está directamente en este docker-compose.yml, la configuración de n8n a través del puerto 5678 se complementa con la implementación de ngrok. Una vez que n8n está en ejecución, ngrok se inicia en otro contenedor creando

un túnel seguro que mapea una URL pública de ngrok

al puerto local 5678 del host (donde n8n es accesible). Esta URL pública es la que se configura en GitHub como la URL del webhook, permitiendo que los eventos de push o pull request en el repositorio disparen los flujos de n8n de forma remota.

Docker_Sonarqube

```
services:
  sonarqube:
    image: sonarqube:latest # O la versión específica que se desee, ej. sonarqube:lts
    container_name: sonarqube
    ports:
      - "9000:9000" # Exposición de la interfaz web
    environment:
      SONARQUBE_JDBC_URL: jdbc:postgresql://db/sonarqube
      SONARQUBE_JDBC_USERNAME: sonar
      SONARQUBE_JDBC_PASSWORD: sonar
    depends_on:
      - db # Asegura que la base de datos se inicie antes que SonarQube
    volumes:
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
      - sonarqube_logs:/opt/sonarqube/logs
      # Importante: se monta un volumen para que SonarQube acceda al código si es necesario
      # - /home/userAGL/TFG/data:/opt/sonarqube/data # Esto es incorrecto, debería ser para el código
      # CORRECCIÓN: Si el escáner se ejecuta desde otro lado (ej. sonar-scanner),
      # SonarQube Server NO necesita acceso al código fuente aquí.
      # Este volumen aquí sobrescribiría los datos de SonarQube con el código.
      # El volumen `/home/userAGL/TFG/data` es para el ESCÁNER, no para el servidor.

  db:
    image: postgres:13 # Versión específica de PostgreSQL
    container_name: sonarqube_db
    restart: unless-stopped # Reinicia el contenedor a menos que se detenga manualmente
    environment:
      POSTGRES_USER: sonar
      POSTGRES_PASSWORD: sonar
      POSTGRES_DB: sonarqube # Nombre de la base de datos para SonarQube
    volumes:
      - sonarqube_db_data:/var/lib/postgresql/data # Persistencia de los datos de la BD

  sonar-scanner:
    image: sonarsource/sonar-scanner-cli:latest
    container_name: sonar-scanner
    environment:
      - SONAR_TOKEN=${SONAR_TOKEN} # Token de autenticación
      # - SONAR_HOST_URL=http://127.0.0.1:9000
      - SONAR_PROJECT_KEY=hello:world
      - SONAR_PROJECT_NAME=HelloWorld
      - SONAR_PROJECT_VERSION=0.1
```

```
    depends_on:
      - sonarqube # Asegura que SonarQube esté en ejecución antes de lanzar el
escáner
    volumes:
      - /home/userAGL/TFG/data/code/java:/usr/src # Monta el código a analizar
en el contenedor
    entrypoint: ["/bin/sh", "-c"]
    command:
e        - sonar-scanner -Dsonar.projectKey=mi-proyecto -Dsonar.sources=. -
Dsonar.host.url=http://sonarqube:9000 && tail -f /dev/null
```

Explicación de la Configuración:

Servicio sonarqube:

image: Utiliza la imagen oficial de SonarQube para asegurar un despliegue estándar.

ports: Expone el puerto 9000 para acceder a la interfaz web de SonarQube desde el host.

environment: Configura las credenciales JDBC para que SonarQube se conecte a la base de datos PostgreSQL (db es el nombre del servicio en la red Docker).

depends_on: - db: Garantiza que el contenedor db se inicie y esté disponible antes que el contenedor sonarqube.

volumes: Define volúmenes persistentes para los datos de SonarQube, extensiones y logs, asegurando que la información de análisis y la configuración no se pierdan al reiniciar el contenedor.

Servicio db (Base de Datos PostgreSQL):

image: postgres:13: Utiliza una imagen específica de PostgreSQL como base de datos para SonarQube.

environment: Establece las variables de entorno para la configuración de la base de datos (usuario, contraseña, nombre de la BD).

volumes: Monta un volumen persistente para los datos de PostgreSQL, esencial para no perder los datos del servidor SonarQube.

Servicio sonar-scanner:

image: sonarsource/sonar-scanner-cli:latest: Emplea la imagen oficial del cliente de línea de comandos de SonarQube.

environment: Se configura con SONAR_TOKEN (inyectado desde el host para seguridad) y parámetros por defecto del proyecto. En un flujo real de n8n, estos parámetros serían dinámicos.

depends_on: - sonarqube: Asegura que el servidor SonarQube esté en ejecución antes de intentar escanear.

volumes: - /home/userAGL/TFG/data/code/java:/usr/src: Este es el volumen crucial que monta la carpeta local que contiene el código fuente del proyecto Java dentro del contenedor del sonar-scanner, para que este pueda acceder y analizar los archivos.

command: Muestra un ejemplo de cómo se ejecutaría el sonar-scanner. Hay que aclarar que en el flujo de n8n, este comando sería invocado por el nodo "Execute Command" de n8n, y el contenedor sonar-scanner es efímero (creado y destruido por n8n para cada análisis) o un servicio persistente que n8n

"despierte". La opción tail -f /dev/null es una práctica común en Docker Compose para mantener el contenedor en ejecución en segundo plano para depuración.

Ejecución del Análisis de SonarQube a través de n8n

La orquestación del análisis estático se lleva a cabo directamente desde un nodo de un flujo de trabajo de n8n. Tras el evento desencadenante de GitHub, el flujo de n8n realiza las siguientes acciones principales relacionadas con SonarQube:

Preparación del Entorno y Código:

El código fuente del proyecto a analizar se clona desde GitHub al volumen compartido (/home/userAGL/TFG/data/code/java en este ejemplo) que es accesible por el contenedor del sonar-scanner.

Utilizando un nodo "Execute Command" en n8n, se invoca al sonar-scanner CLI. Esto se hace ejecutando un comando similar a docker run --rm --network TFG-network -v /home/userAGL/TFG/data/code/java:/usr/src -e SONAR_TOKEN=${SONAR_TOKEN} sonarsource/sonar-scanner-cli:latest sonar-scanner -Dsonar.projectKey=mi-proyecto -Dsonar.sources=. -Dsonar.host.url=http://sonarqube:9000. Este comando lanzaría un contenedor efímero del sonar-scanner, que realiza el análisis y envía los resultados al servidor SonarQube.

Se utilizan variables de entorno dinámicas (como el SONAR_TOKEN y la URL del host SonarQube) para asegurar que el escáner se comunique correctamente con la instancia del servidor SonarQube en la red Docker.

Docker_owaszap

```
services:
  zaproxy:
    image: zaproxy/zap-stable # Imagen oficial estable de OWASP ZAP
    container_name: zaproxy
    user: zap # Ejecuta el contenedor con el usuario 'zap' por seguridad
    ports:
      - "8080:8080" # Mapea el puerto 8080 del contenedor (proxy) al puerto
8080 del host
    command: > # Comando para iniciar ZAP en modo daemon y configurar la API
      zap.sh -daemon -host 0.0.0.0 -port 8080
      -config api.addrs.addr.name=.*
      -config api.addrs.addr.regex=true
      -config api.key=mi-api-key # Clave de la API para autenticación
    networks:
      default:
        name: TFG-network
        external: true
```

Explicación de la Configuración:
image: zaproxy/zap-stable: Utiliza la imagen oficial de Docker de OWASP ZAP, que proporciona una versión estable y lista para operar de la herramienta.
container_name: zaproxy: Asigna un nombre descriptivo al contenedor, lo que facilita su identificación y gestión dentro del entorno Docker.
user: zap: Ejecuta el contenedor con el usuario zap.
ports: - "8080:8080": Mapea el puerto 8080 del contenedor (el puerto predeterminado del proxy y la API de ZAP) al puerto 8080 del host. Esto permite que n8n y otras herramientas se comuniquen con ZAP.
command: Especifica el comando que ZAP ejecuta al iniciarse:

zap.sh -daemon: Inicia ZAP en modo demonio, sin interfaz gráfica, para interactuar desde los flujos de n8n.
-host 0.0.0.0 -port 8080: Configura ZAP para que escuche en todas las interfaces disponibles en el puerto 8080.
-config api.addrs.addr.name=.* -config api.addrs.addr.regex=true: Permite que la API de ZAP sea accesible desde cualquier dirección IP.
-config api.key=mi-api-key: Establece una clave para la API que se utilizará para autenticar las solicitudes enviadas a ZAP.
networks: Conecta el contenedor a la red Docker TFG-network, asegurando que ZAP pueda comunicarse con n8n.

informe sonarqube
Resumen de Seguridad del Código Java - Proyecto "Hola Mundo"
Fecha del Informe: 30 de mayo de 2024
Herramienta: SonarQube

Resumen Ejecutivo

El análisis de seguridad realizado en el proyecto "Hola Mundo" reveló un total de 274 problemas ⚠️, lo que resulta en una deuda técnica estimada de 2682 minutos. Se identificaron vulnerabilidades y code smells de distintas severidades que requieren atención para mejorar la calidad y seguridad del código.

Hallazgos Clave

🔴 Problemas Críticos: 5
🟠 Problemas Mayores: 30
🟡 Problemas Menores: 3
Total Problemas: 274
Deuda Técnica Estimada: 2682 minutos

Detalle por Severidad

🔴 Críticos (5): Estos problemas requieren atención inmediata ya que pueden tener un impacto significativo en la seguridad o funcionalidad de la aplicación. Incluyen:
Ubicaciones incorrectas de archivos.
Alta complejidad cognitiva en métodos.
Sobrescritura del método `finalize()`.

🟠 Mayores (30): Estos problemas pueden llevar a errores, vulnerabilidades o dificultades en el mantenimiento del código a mediano plazo. Los problemas más frecuentes son:
Uso de `System.out` y `System.err` para logging.
Parámetros de método no utilizados.
Uso incorrecto de formato para argumentos.
Asignaciones inútiles a variables locales.

🟡 Menores (3): Estos problemas representan principalmente mejoras en la legibilidad y mantenibilidad del código. Incluyen:
Uso de métodos deprecated.
Variables locales no utilizadas.

Recomendaciones Clave

1. 🔴 Priorizar Problemas Críticos: Resolver inmediatamente los problemas marcados como críticos.
2. 🟠 Implementar Logging Adecuado: Reemplazar el uso de `System.out` y `System.err` con un framework de logging robusto.
3. 🟠 Refactorizar Métodos Complejos: Reducir la complejidad cognitiva de los métodos más extensos.
4. 🟡 Eliminar Código Innecesario: Remover variables y parámetros sin usar.
5. ⚠️ Revisar SqlInjectionVulnerability.java: Inspeccionar el código en busca de posibles vulnerabilidades de inyección SQL.

Informe OWAS ZAP

Riesgo Medio: Missing Anti-clickjacking Header

Descripción: Los navegadores modernos soportan los encabezados HTTP Content-Security-Policy y X-Frame-Options. La ausencia de estos encabezados permite ataques de clickjacking, donde un atacante puede engañar al usuario para que haga clic en elementos ocultos de una página web maliciosa.
Solución: Se recomienda configurar el servidor web, de aplicación o balanceador de carga para establecer uno de estos encabezados en todas las páginas, preferiblemente con la directiva DENY (si la página nunca debe ser enmarcada) o SAMEORIGIN (si solo es por páginas del mismo servidor). Alternativamente, implementar la directiva frame-ancestors de Content Security Policy.
Riesgo Medio: Cross-Site Request Forgery (CSRF)

Descripción: Posibles vulnerabilidades que permiten a un atacante obligar a los usuarios finales a ejecutar acciones no deseadas en una aplicación web en la que están autenticados.
Solución: Se proponen varias fases de remediación:
Arquitectura y Diseño: Utilizar librerías o frameworks que prevengan esta debilidad (OWASP CSRFGuard). Generar un nonce único por cada formulario y verificarlo. Identificar operaciones peligrosas y solicitar confirmación. Usar el control de gestión de sesiones de ESAPI.
Implementación: Asegurar que la aplicación esté libre de problemas de Cross-Site Scripting (XSS), ya que la mayoría de las defensas CSRF pueden ser eludidas con scripts controlados por el atacante. No usar el método GET para solicitudes que cambien el estado. Validar el encabezado HTTP Referer (aunque puede tener limitaciones).
Riesgo Bajo: X-Powered-By Header

Descripción: La presencia del encabezado X-Powered-By revela información sobre la tecnología subyacente del servidor o aplicación, lo que podría ser útil para un atacante.
Solución: Configurar el servidor web, de aplicación o balanceador de carga para suprimir este encabezado o proporcionar detalles genéricos.
Riesgo Bajo: X-Content-Type-Options Header Missing

Descripción: La ausencia del encabezado X-Content-Type-Options: nosniff puede permitir el "MIME-sniffing" por parte del navegador, lo que podría llevar a ataques de cross-site scripting si se interpreta contenido de forma incorrecta.

Solución: Asegurar que el servidor establezca el encabezado Content-Type apropiadamente y el X-Content-Type-Options a nosniff para todas las páginas web.
Los resultados de ZAP son particularmente valiosos como input para los LLMs, ya que no solo identifican vulnerabilidades sino que también ofrecen soluciones concretas y detalladas. Esto permite que los LLMs reciban un contexto rico para generar código de parche o refactorización que aborde directamente la vulnerabilidad, y también para crear pruebas unitarias o de integración que verifiquen la mitigación de dichos riesgos.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 10:12:37 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)