Universidad CENFOTEC

Maestría en Ingeniería del Software con énfasis en Arquitectura y Diseño de Software

Documento final de Proyecto de Investigación Aplicada 2

Diseño de un modelo arquitectónico de resiliencia para la etapa de resolución de dependencias en pipelines CI/CD basado en detección, clasificación y remediación automática de fallos

Carlos Eduardo Peralta Coto

Agosto, 2026

DECLARACIÓN DE DERECHOS DE AUTOR

DEDICATORIA

Le dedico este trabajo a mi hija Jazmín, por ser mi mayor motivación para nunca rendirme en los estudios, que todos tus sueños se hagan realidad. A mi esposa Estefanny, por su infinita paciencia y apoyo, por aguantarme en cada paso y permitirme seguir adelante. Este logro es nuestro. Y a Dios, por darme la fuerza y la sabiduría necesaria, y poner personas tan magníficas a mi lado.

TRIBUNAL EXAMINADOR

Este proyecto fue aprobado por el Tribunal Examinador de la carrera: Maestría Profesional en Ingeniería del Software con Énfasis en Arquitectura y Diseño Software**,** requisito para optar por el título de grado de Maestría Profesional en Ingeniería del Software con Énfasis en Arquitectura y Diseño Software, la estudiante Carlos Eduardo Peralta Coto.

JIRLEY RAMIREZ CORDERO (FIRMA)
Digitally signed by JIRLEY RAMIREZ CORDERO (FIRMA)
Date: 2026.08.19 13:29:20 -06'00'

M.Sc. Shirley Ramírez Cordero
Tutor

IGNACIO TREJOS ZELAYA (FIRMA)
Firmado digitalmente por IGNACIO TREJOS ZELAYA (FIRMA)
Fecha: 2026.08.20 07:35:52 -06'00'

M.Sc. Ignacio Trejos Zelaya
Lector 1

JUAN PABLO ARAYA GONZALEZ (FIRMA)
PERSONA FISICA, CPF-04-0197-0613.
Fecha declarada: 21/08/2026 09:36:19 AM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

M.Sc. Juan Pablo Araya González
Lector 2

San José, Costa Rica, 18 de Agosto 2026

Diseño de un modelo arquitectónico de resiliencia para la etapa de resolución de dependencias en pipelines CI/CD basado en detección, clasificación y remediación automática de fallos

Peralta Coto, Carlos Eduardo. peralta.coto.carlos@gmail.com

RESUMEN Esta investigación aborda el problema de los fallos recurrentes en la etapa de resolución de dependencias dentro de pipelines CI/CD, los cuales suelen gestionarse de manera reactiva y manual, generando reprocesos, consumo innecesario de recursos y retrasos en la entrega de software. El objetivo del estudio es proponer y validar un modelo arquitectónico que introduzca resiliencia de forma desacoplada en esta etapa crítica del pipeline, permitiendo detectar, clasificar y remediar fallos conocidos de manera gobernada y explícita. Para ello, se diseñó un modelo arquitectónico utilizando el enfoque C4, basado en principios de resiliencia, separación de responsabilidades y toma de decisiones determinista, y se implementó una prueba de concepto que simula escenarios representativos de fallo. Los resultados obtenidos a través de la simulación demostraron que la arquitectura del modelo responde correctamente ante los fallos analizados, ejecutando el flujo completo de detección y remediación, y evidenciando la viabilidad de reducir tanto el tiempo de recuperación como la intervención manual. Como conclusión, la investigación aporta un enfoque innovador al formalizar la resiliencia en una etapa tradicionalmente reactiva del pipeline CI/CD, ofreciendo un modelo arquitectónico reusable y extensible con potencial aplicación en entornos reales.

ABSTRACT This research addresses the problem of recurring failures during the dependency resolution stage of CI/CD pipelines, which are typically handled reactively and manually, leading to re-runs, unnecessary resource consumption, and delays in software delivery. The objective of the study is to propose and validate an architectural model that introduces decoupled resilience into this critical pipeline stage, enabling the detection, classification, and remediation of known failures in a controlled and explicit manner. To achieve this, an architectural model was designed using the C4 approach, based on principles of resilience, separation of responsibilities, and deterministic decision-making. A proof of concept was implemented that simulates representative failure scenarios. The results obtained through the simulation demonstrated that the model´s architecture responds correctly to the analyzed failures, executing the complete detection and remediation flow, and demonstrating the feasibility of reducing both recovery time and manual intervention. In conclusion, this research offers an innovative approach by formalizing resilience in a traditionally reactive stage of the CI/CD pipeline, providing a reusable and extensible architectural model with potential application in real-world environments.

PALABRAS CLAVE Integración continua, resolución de dependencias, manejo de dependencias, detección de fallas, modelo arquitectónico.

I. INTRODUCCIÓN

En el contexto de la ingeniería de software moderna, las prácticas de Integración Continua (Continuous Integration, CI), Entrega Continua (Continuous Delivery) y Despliegue Continuo (Continuous Deployment) se han consolidado como elementos fundamentales para acelerar el desarrollo y la entrega de software manteniendo altos estándares de calidad. Estas prácticas promueven la integración frecuente de cambios de código, la automatización de la construcción del sistema y la ejecución continua de pruebas, permitiendo ciclos de desarrollo más cortos, retroalimentación temprana y liberaciones más frecuentes del software (Shahin, Ali Babar

y Zhu 2017). Como resultado, las organizaciones pueden responder con mayor rapidez a las necesidades del mercado, mejorar la productividad de los equipos de desarrollo y aumentar la confiabilidad de sus procesos de entrega.

La implementación de estas prácticas se apoya en pipelines CI/CD, los cuales automatizan tareas críticas del ciclo de vida del software, tales como: la compilación, ejecución de pruebas, análisis de calidad del código, gestión de artefactos y despliegue en distintos entornos. Sin embargo, estos pipelines dependen de una amplia variedad de recursos y componentes, incluyendo librerías externas, módulos internos, repositorios de dependencias, scripts de construcción y configuraciones específicas de infraestructura. Debido a esta complejidad, la correcta ejecución de los pipelines está estrechamente ligada a la disponibilidad y consistencia de dichas dependencias. Cuando estas son incompatibles, se encuentran desactualizadas o no están disponibles, pueden producirse fallos que interrumpen el proceso de integración continua y afectan la continuidad operativa de la entrega de software.

Uno de los desafíos más relevantes asociados con los pipelines CI/CD es la ocurrencia de fallos durante la ejecución de los diferentes trabajos que los componen. Estos fallos pueden provocar retrasos significativos en los procesos de construcción y validación, reduciendo la productividad de los desarrolladores y aumentando los costos operativos de las organizaciones (Sun, Friberg y Staron 2026). Además, cuando un pipeline falla, los equipos deben esperar a la finalización del proceso para recibir retroalimentación sobre el error, lo que genera interrupciones en el flujo de trabajo y aumenta el tiempo medio de recuperación (Mean Time to Recovery, MTTR). Entre las múltiples causas de fallo reportadas se encuentran pruebas inestables, errores de configuración, problemas de infraestructura y, de manera recurrente, fallos relacionados con la resolución de dependencias externas.

La etapa de resolución de dependencias constituye una de las fases más críticas y vulnerables dentro de un pipeline CI/CD. Durante esta etapa, el sistema debe localizar, descargar, verificar e integrar los componentes requeridos para la construcción del software. Problemas como la indisponibilidad de repositorios, conflictos de versiones, corrupción de artefactos o errores de conectividad pueden impedir el proceso de construcción y desencadenar fallos en cascada en etapas posteriores del pipeline. Sun, Friberg y Staron (2026), Shahin, Ali Babar y Zhu (2017) y Zampetti et al. (2021) identifican los problemas de dependencias como una de las principales causas de fallos y desafíos en pipelines CI/CD, junto con los fallos en pruebas y las configuraciones incorrectas, lo que evidencia la necesidad de fortalecer los mecanismos de resiliencia en esta fase del proceso.

Estudios como Zampetti et al. (2021) y Shahin, Ali Babar y Zhu (2017) hablan de distintos patrones como el retry y el timeout, y de procesos automatizados, para mejorar la resiliencia de sistemas distribuidos. Estos patrones contribuyen a reducir la tasa de errores y a mejorar la capacidad de recuperación frente a fallos transitorios. No obstante, las soluciones existentes suelen abordar problemas específicos de manera aislada y carecen de una estructura arquitectónica que permita integrar de forma sistemática mecanismos de detección, clasificación y remediación automática. De manera similar, la investigación actual sobre pipelines CI/CD se ha concentrado principalmente en aspectos como la reducción de tiempos de ejecución, la optimización de recursos o la priorización de tareas, mientras que la resiliencia integral frente a fallos continúa siendo un área que requiere mayor exploración (Sun, Friberg y Staron 2026; Shahin, Ali Babar y Zhu 2017).

Ante esta situación, surge la necesidad de desarrollar enfoques arquitectónicos que permitan diseñar pipelines más robustos y adaptables, capaces de identificar automáticamente los distintos tipos de fallos, determinar su naturaleza y aplicar estrategias de recuperación apropiadas. Un modelo arquitectónico de este tipo no solo contribuiría a reducir los tiempos de inactividad y la intervención manual, sino que también permitiría analizar fallos de manera sistémica, identificar puntos únicos de fallo, mejorar la coherencia entre las diferentes etapas del pipeline y facilitar la evaluación de escenarios de falla complejos.

La necesidad de incrementar la resiliencia de los pipelines CI/CD frente a fallos relacionados con dependencias requiere un enfoque arquitectónico que permita integrar de forma sistemática mecanismos de detección, clasificación y remediación automática. Un modelo de estas características puede reducir la intervención manual, disminuir los tiempos de recuperación y mejorar la continuidad operativa del proceso de integración continua. En respuesta a esta necesidad, este trabajo propone un modelo arquitectónico orientado a la etapa de resolución de dependencias que incorpora capacidades de autorrecuperación independientes de herramientas o plataformas específicas, con el propósito de fortalecer la confiabilidad y robustez de los pipelines CI/CD.

II. METODOLOGÍA

La investigación se desarrolló bajo un enfoque metodológico de Ciencia de Diseño (Design Science Research, DSR), el cual se orienta a la creación y evaluación de artefactos diseñados para resolver problemas específicos en contextos reales. Este enfoque resulta especialmente apropiado para el estudio, ya que aborda una problemática de carácter práctico relacionada con la resiliencia en la etapa de resolución de dependencias dentro de los pipelines de Integración Continua y Entrega Continua (CI/CD). A partir de este marco metodológico, se propone y valida un artefacto arquitectónico fundamentado en conocimiento científico existente, con el objetivo de mejorar la capacidad del proceso para enfrentar fallos y mantener la continuidad operativa.

El estudio se clasifica como investigación aplicada, ya que tiene como propósito diseñar una solución concreta a una problemática identificada en entornos reales de ingeniería de software, específicamente la falta de mecanismos arquitectónicos estructurados que permitan gestionar de manera resiliente los fallos que pueden ocurrir durante la resolu-

ción de dependencias en pipelines CI/CD. Adicionalmente, el estudio presenta un alcance exploratorio, ya que aborda un tema que ha recibido una atención limitada desde la perspectiva de la arquitectura de software. En otras palabras, se requirió examinar el estado actual del conocimiento, identificar conceptos y enfoques relevantes, y establecer las bases teóricas necesarias para sustentar la propuesta arquitectónica planteada.

La investigación adopta un enfoque cualitativo, basado en el análisis e interpretación de información obtenida de fuentes científicas y técnicas especializadas, así como de prácticas ampliamente utilizadas en los ámbitos de la ingeniería de software y DevOps. Mediante este análisis se logró identificar una atención limitada en la etapa de resolución de dependiencias como punto crítico de fallo, a pesar de su impacto directo en la continuidad de los procesos de construcción y despliegue.

Los estudios revisados y seleccionados mediante un proceso sistemático basado en criterios de calidad académica, evidencian que los enfoques predominantes se centran en el uso de mecanismos de redundancia, cachés locales y repositorios espejo para mitigar fallos asociados a la indisponibilidad de dependencias externas (Shahin, Ali Babar y Zhu 2017; Whitmore 2025). Estos enfoques, si bien efectivos en ciertos escenarios, suelen implementarse de manera aislada y carecen de una integración arquitectónica formal dentro del pipeline, lo que limita su capacidad de adaptación ante fallos complejos o combinados.

En el ámbito de la ingeniería de software, algunos trabajos destacan la importancia de incorporar principios de arquitecturas orientadas a eventos y patrones de resiliencia, como retry, fallback y circuit breakers, en sistemas distribuidos, como se menciona en Hassan, Meng y Wang (2023) y An et al. (2024). No obstante, estos patrones se aplican fundamentalmente en la ejecución de servicios en producción y no en etapas previas del ciclo de vida, como la construcción y resolución de dependencias. Esta brecha revela una falta de extrapolación de prácticas consolidadas de resiliencia hacia el contexto específico de CI/CD.

La literatura también evidencia que las herramientas de gestión de dependencias (por ejemplo, Maven, npm o pip) implementan mecanismos básicos de recuperación de fallos, como reintentos automáticos o validaciones de integridad, pero estos son generalmente limitados, poco configurables y no ofrecen capacidades avanzadas de diagnóstico o remediación automatizada (Whitmore 2025; Sun, Friberg y Staron 2026). En consecuencia, muchos errores requieren intervención manual, lo que introduce retrasos y reduce la eficiencia operativa de los equipos de desarrollo.

Por otra parte, algunos estudios recientes exploran el uso de observabilidad y análisis de logs para identificar patrones de error en pipelines (An et al. 2024; Zampetti et al. 2021), permitiendo detectar fallos recurrentes y mejorar la trazabilidad. Sin embargo, estos enfoques se centran en la monitorización y no en la acción correctiva autónoma, lo que restringe su aporte a la resiliencia activa del sistema.

En conjunto, el análisis del estado del arte evidencia tres limitaciones relevantes. En primer lugar, se observa la falta de modelos arquitectónicos integrales que aborden de manera específica la resiliencia durante el proceso de resolución de dependencias. En segundo lugar, las soluciones existentes suelen presentarse como mecanismos independientes, con poca o ninguna coordinación entre ellos, lo que dificulta una gestión unificada de los fallos. Finalmente, persiste un bajo nivel de automatización en las actividades de remediación, por lo que numerosos incidentes requieren intervención manual para su resolución, afectando la continuidad y eficiencia de los pipelines CI/CD.

A partir de estas limitaciones, se identifica una oportunidad de investigación orientada al desarrollo de soluciones que integren de forma sistemática mecanismos de detección, clasificación y remediación automática de fallos dentro de una arquitectura unificada. Una aproximación de este tipo permitiría coordinar las distintas estrategias de resiliencia y responder de manera más efectiva a los desafíos presentes en los pipelines CI/CD. En este sentido, la brecha identificada en la literatura sirve como fundamento para la propuesta presentada en este trabajo, cuyo objetivo es aportar un modelo arquitectónico que contribuya a superar las limitaciones observadas en las soluciones actuales.

El análisis de la literatura evidencia que, aunque existen múltiples mecanismos para la detección y recuperación de fallos en pipelines CI/CD, estos suelen abordarse de forma aislada y carecen de una formalización arquitectónica que permita coordinar procesos de detección, clasificación y remediación automática. También se destaca que la etapa de resolución de dependencias constituye un punto crítico debido a su dependencia de servicios externos, repositorios remotos y configuraciones específicas.

La resiliencia se define como la capacidad de un sistema para mantener su funcionamiento esperado frente a condiciones adversas y recuperarse de fallos sin comprometer la continuidad operativa. Aplicada a pipelines CI/CD, la resiliencia implica:

- Detectar anomalías tempranamente.
- Clasificar adecuadamente los incidentes.
- Ejecutar acciones de recuperación automática.
- Minimizar la intervención humana.
- Preservar la continuidad del proceso de integración continua.

La literatura revisada indica que gran parte de las soluciones existentes poseen un enfoque predominantemente reactivo, orientado a detectar problemas una vez ocurridos, sin incorporar mecanismos coordinados de respuesta arquitectónica.

El análisis permitió identificar una taxonomía de fallos recurrentes durante la resolución de dependencias, la cual se muestra en la tabla 1. Esta clasificación constituye uno de los principales insumos para el diseño del modelo arquitectónico propuesto, ya que permite definir categorías explícitas para la clasificación automática de fallos.

Tabla 1. Taxonomía de fallos en resolución de dependencias

Tipo de fallo	Descripción
Disponibilidad de recursos	Repositorios remotos inaccesibles o indisponibles
Red y conectividad	Timeouts, pérdida de conexión o errores DNS
Conflictos de versiones	Dependencias incompatibles o restricciones inconsistentes
Integridad de artefactos	Paquetes corruptos o firmas inválidas
Configuración del proyecto	Errores en archivos de configuración o gestores de dependencias

Para la detección de fallos, se identifican distintos mecanismos en la literatura. An et al. (2024) proponen mecanismos de abstracción de registros para identificar patrones de fallo mediante el análisis de logs y trazas de ejecución. Este enfoque facilita la detección de errores recurrentes y fallos inestables. Por otro lado, Hassan, Meng y Wang (2023) proponen una técnica llamada UniLoc, basada en la recuperación de información que combina logs del pipeline, historial de cambios, artefactos de compilación y árboles sintácticos abstractos.

En cuanto a la clasificación de fallos, existen diversos enfoques, entre los cuales se pueden resaltar la clasificación en Good Failures y Bad Failures, propuesta por Sun, Friberg y Staron (2026), el cual promueve el principio de fail fast, favoreciendo la detección temprana y reduciendo el impacto organizacional. Otras clasificaciones relevantes son:

- Por etapa del pipeline.
- Según la causa del build.
- Mediante análisis de CI Smells.
- Basada en análisis de logs.

Una vez identificado y clasificado un problema, el sistema debe ser capaz de ejecutar mecanismos de recuperación que permitan restaurar la continuidad operativa sin requerir intervención manual constante. En este contexto, la literatura científica ha propuesto diversas técnicas de remediación que buscan reducir el impacto de los fallos y mejorar la confiabilidad de los procesos de integración continua.

Uno de los mecanismos más utilizados es el retry automático, el cual consiste en reintentar la ejecución de una tarea cuando se detectan errores transitorios, especialmente aquellos relacionados con problemas temporales de conectividad o indisponibilidad momentánea de servicios externos. Este enfoque resulta particularmente útil en la etapa de resolución de dependencias, donde la descarga de paquetes desde repositorios remotos puede verse afectada por condiciones de red inestables. Su principal ventaja radica en que permite recuperar la ejecución del pipeline sin necesidad de intervención humana, reduciendo los tiempos de recuperación y aumentando la disponibilidad del proceso.

Otra práctica ampliamente utilizada corresponde a la configuración de jobs permitidos a fallar (allow failure), mediante la cual determinadas tareas no críticas pueden producir errores sin provocar la interrupción completa del pipeline. Esta técnica resulta adecuada para actividades complementarias, como algunas pruebas experimentales o verificaciones no obligatorias, permitiendo que el flujo principal continúe mientras se registra la incidencia para su posterior análisis. Aunque no constituye una remediación directa del fallo, contribuye a mantener la continuidad operativa al evitar interrupciones innecesarias.

En etapas posteriores del pipeline, particularmente durante el despliegue, la literatura reporta el uso de mecanismos de rollback automático, los cuales permiten restaurar una versión estable previamente validada cuando una nueva liberación presenta problemas. Esta estrategia reduce significativamente el impacto de fallos en ambientes productivos y constituye una práctica fundamental en arquitecturas orientadas a la alta disponibilidad. De manera complementaria, el backtracking incremental permite retroceder gradualmente en la cadena de dependencias hasta identificar una combinación funcional de versiones, facilitando la recuperación ante conflictos de compatibilidad o actualizaciones defectuosas.

La investigación sobre evolución de pipelines también ha identificado la presencia de malas prácticas conocidas como CI Smells, las cuales incrementan la complejidad, reducen la mantenibilidad y favorecen la aparición de errores recurrentes. Como respuesta a este problema, Zampetti et al. (2021) proponen la reestructuración de pipelines y la eliminación sistemática de dichas deficiencias arquitectónicas. Estudios sobre la evolución de configuraciones CI/CD demuestran que la reorganización de etapas, la simplificación de dependencias y la mejora de la modularidad contribuyen a incrementar la estabilidad y facilitar el mantenimiento de los procesos automatizados.

Asimismo, las herramientas modernas de despliegue incorporan mecanismos de error handling, los cuales permiten capturar excepciones específicas y ejecutar acciones correctivas previamente definidas. Este enfoque introduce capacidades de recuperación controlada y evita que errores particulares provoquen la interrupción total del proceso. De forma complementaria, el enfoque denominado Process Oriented Dependability (POD) modela las operaciones del pipeline como procesos observables, utilizando métricas y registros para mejorar las capacidades de diagnóstico, recuperación y análisis posterior de incidentes. Este modelo resulta especialmente útil en entornos distribuidos y arquitecturas cloud donde la complejidad operacional dificulta la identificación rápida de las causas raíz de los fallos.

Otra estrategia destacada en la literatura consiste en el uso de arquitecturas modulares y desacopladas, las cuales buscan minimizar la propagación de errores entre componentes del pipeline. Al reducir el nivel de acoplamiento entre las distintas etapas, un fallo puede aislarse más fácilmente y limitar su impacto sobre el resto del sistema. Este principio arquitectónico resulta especialmente relevante en ecosistemas modernos de integración continua, donde múltiples herramientas, repositorios y servicios externos interactúan de forma permanente.

En la tabla 2 se resumen las técnicas principales identificadas en la literatura.

Tabla 2. Técnicas de resiliencia y recuperación en CI/CD

Técnica	Propósito
Retry automático	Recuperación de fallos transitorios
Allow Failure	Evitar interrupciones por errores no críticos
Rollback automático	Restaurar estados estables
Backtracking incremental	Revertir dependencias problemáticas
Eliminación de CI Smells	Mejorar mantenibilidad del pipeline
Error Handling automatizado	Gestionar excepciones durante despliegues
Process Oriented Dependability (POD)	Mejorar recuperación basada en procesos
Arquitecturas desacopladas	Reducir propagación de fallos

La revisión de la literatura evidencia, sin embargo, que la mayoría de estas técnicas han sido desarrolladas principalmente para las etapas de construcción, pruebas y despliegue, existiendo una menor cantidad de investigaciones orientadas específicamente a la resolución de dependencias. Además, muchas soluciones se implementan de manera aislada y carecen de una estructura arquitectónica que coordine los procesos de detección, clasificación y remediación de forma integrada. Esta situación coincide con una de las principales limitaciones identificadas en el estado de la cuestión: la ausencia de modelos arquitectónicos explícitos que formalicen la resiliencia como una capacidad transversal del pipeline.

A partir de estas observaciones, la literatura permite establecer un conjunto de requerimientos fundamentales para el diseño de pipelines resilientes. En primer lugar, se requiere la capacidad de detectar fallos de manera temprana mediante monitoreo continuo, análisis de anomalías y recopilación de información diagnóstica como registros, métricas y trazas de ejecución. La detección temprana adquiere especial relevancia debido a que los errores identificados en fases iniciales suelen presentar menores costos de corrección y un impacto reducido sobre el proceso de desarrollo.

En segundo lugar, se requiere que los sistemas sean capaces de clasificar adecuadamente los incidentes detectados. Esto implica diferenciar entre problemas de red, indisponibilidad de repositorios, conflictos de versiones, errores de configuración o fallos de integridad de artefactos, además de determinar su nivel de impacto y posible causa raíz. La clasificación constituye un elemento esencial para habilitar mecanismos de respuesta automatizada, ya que permite seleccionar la estrategia de recuperación más apropiada para cada situación.

La capacidad de remediación automática representa el tercer requisito fundamental. La literatura destaca la necesidad de incorporar reintentos controlados, mecanismos de fallback, repositorios alternativos y estrategias de almacenamiento en caché que permitan reducir la dependencia de recursos externos y mejorar la capacidad de recuperación del pipeline. Estas capacidades deben operar de manera coordinada para minimizar la intervención manual y disminuir los tiempos de recuperación ante incidentes.

Finalmente, los estudios consultados enfatizan la importancia de incorporar mecanismos de gobernanza y control dentro de la arquitectura del pipeline. Duvall, Matyas y Glover (2007) señalan que los procesos de integración continua deben diseñarse bajo el principio de fail fast, y Sun, Friberg y Staron (2026) distinguen entre fallos tempranos y tardíos, destacando la necesidad de priorizar la detección y contención en etapas previas a la integración. Estas capacidades pueden implementarse mediante gates de validación, políticas de promoción entre etapas y criterios automáticos de bloqueo que regulen el flujo de ejecución del pipeline.

En la misma línea, Whitmore (2025) plantea que los pipelines modernos deben incorporar mecanismos explícitos de gobernanza capaces de regular actualizaciones automáticas de dependencias, restricciones de versiones y procesos de validación. Dichos mecanismos permiten complementar la resiliencia reactiva con capacidades preventivas orientadas a evitar la propagación de cambios defectuosos a través de múltiples sistemas. La integración de estas políticas dentro de la arquitectura contribuye a establecer un modelo de resiliencia gobernada, donde las acciones de recuperación se ejecutan dentro de límites previamente definidos y alineados con los objetivos organizacionales.

La síntesis de estos hallazgos proporciona la base conceptual para el diseño de un modelo arquitectónico de resiliencia orientado a la etapa de resolución de dependencias. En particular, los requerimientos de detección, clasificación, remediación, observabilidad y gobernanza constituyen los pilares fundamentales que orientan tanto la construcción del artefacto propuesto como la posterior evaluación conceptual basada en escenarios de fallo representativos.

A partir del análisis de la literatura sobre resiliencia en pipelines CI/CD, gestión de dependencias y evolución de pipelines, se identificaron un conjunto de requerimientos funcionales y no funcionales que deben ser satisfechos por una solución orientada a la detección y remediación automática de fallos durante la resolución de dependencias. Estos requerimientos constituyen la base para el diseño del modelo arquitectónico propuesto. Estos requerimientos se visualizan en la tabla 3.

III. PROPUESTA DE SOLUCIÓN

Al finalizar el proceso iterativo, se propone una capa de resiliencia externa y desacoplada para gestionar fallos recurrentes en la etapa de resolución de dependencias de pipelines CI/CD, sin reemplazar las herramientas existentes, mediante el uso de distintas técnicas de detección, clasificación y remediación de errores. Este modelo arquitectónico se centra en la etapa de resolución de dependencias, permitiendo que el pipeline reaccione ante fallos conocidos y recurrentes.

La solución propuesta adopta un enfoque arquitectónico orientado a la resiliencia, concebido como una capa externa y desacoplada que se integra con los pipelines CI/CD existentes, específicamente en la etapa de resolución de dependencias. Este enfoque parte del principio de que la resiliencia no debe implementarse únicamente a nivel de herramientas individuales, sino como una capacidad trans-

Tabla 3. Requerimientos para un framework resiliente en pipelines CI/CD

Categoría	Requerimientos principales	Objetivo
Detección de fallos	Monitoreo continuo, detección de anomalías, alertas y captura de logs/métricas	Identificar fallos tempranamente
Clasificación de fallos	Identificación, priorización, detección de recurrencia y análisis de causa raíz	Seleccionar la remediación adecuada
Remediación automática	Reintentos, fallback, caché de dependencias, aislamiento y recuperación	Mantener la continuidad del pipeline
Gestión de dependencias	Control de versiones, validación de artefactos, gestión transitiva y redundancia	Reducir riesgos de dependencias externas
Adaptabilidad	Reconfiguración de etapas, integración de herramientas y monitoreo evolutivo	Permitir la evolución del pipeline
Visibilidad y trazabilidad	Dashboard, historial de eventos, registro de remediaciones e información contextual	Facilitar diagnóstico y auditoría
Gobernanza	Políticas de ejecución, control de acceso, validación de cambios y alineación organizacional	Garantizar control y cumplimiento

versal del sistema, capaz de observar, analizar y reaccionar ante fallos recurrentes de forma sistemática y gobernada. En lugar de modificar o reemplazar los mecanismos nativos de resolución de dependencias provistos por herramientas como Maven, npm, Gradle o pip, la arquitectura propuesta rodea dichos mecanismos y actúa sobre sus resultados. De esta manera, el sistema se posiciona como un observador reactivo que interpreta los fallos producidos durante la resolución de dependencias y ejecuta acciones correctivas predefinidas, sin interferir directamente en los algoritmos internos de resolución. Esta decisión permite mantener la compatibilidad con múltiples ecosistemas tecnológicos y evita introducir complejidad innecesaria en componentes altamente especializados.

Se utiliza una separación clara de responsabilidades, dividiendo el proceso de resiliencia en etapas bien definidas: detección del fallo, clasificación del mismo, toma de decisiones basada en reglas explícitas y ejecución de acciones de remediación controladas. Esta descomposición facilita la comprensión del sistema, mejora su mantenibilidad y habilita la evolución independiente de cada componente conforme cambian las necesidades del entorno o del pipeline.

Otro aspecto central es la priorización de la explicabilidad y la gobernanza de las decisiones. En el contexto de un pipeline CI/CD, donde la reproducibilidad y la trazabilidad son críticos, la arquitectura favorece el uso de reglas explícitas y determinísticas para la clasificación de fallos y la selección de acciones de remediación. Esto permite comprender y auditar fácilmente el comportamiento del sistema, reduciendo el riesgo asociado a decisiones opacas o difíciles de justificar, especialmente en entornos empresariales o regulados.

La arquitectura se basa en principios como el bajo acoplamiento, la independencia tecnológica y la extensibilidad. Al concebir el sistema de resiliencia como un servicio independiente del pipeline, la solución puede desplegarse, escalarse y evolucionar sin afectar directamente la operación normal del CI/CD.

A partir de la síntesis de la literatura relacionada con resiliencia en pipelines CI/CD, gestión de dependencias y remediación automática de fallos, se definieron los requerimientos que debe satisfacer la solución propuesta. Como se muestra en la tabla 4, estos requerimientos se agrupan en siete categorías que abarcan desde la detección y clasificación de fallos hasta aspectos de gobernanza, trazabilidad y evolución del pipeline. La identificación de estas capacidades permitió establecer los criterios arquitectónicos que orientan el diseño del modelo propuesto.

Tabla 4. Asignación de requerimientos a componentes arquitectónicos

Requerimiento	Componente Arquitectónico Responsable
Monitoreo continuo	Motor de Observabilidad
Detección de anomalías	Motor de Detección
Clasificación de fallos	Motor de Clasificación
Identificación de causa raíz	Motor de Clasificación
Reintentos automáticos	Motor de Remediación
Fallback de repositorios	Motor de Remediación
Gestión de caché	Gestor de Dependencias
Trazabilidad de eventos	Repositorio Histórico
Gestión de políticas	Motor de Gobernanza
Reconfiguración dinámica	Orquestador de Pipeline

La arquitectura propuesta introduce una capa de resiliencia desacoplada que se integra de manera no intrusiva con los pipelines CI/CD existentes, específicamente en la etapa de resolución de dependencias. Desde una perspectiva de alto nivel, el sistema se concibe como un sistema externo y complementario al pipeline, cuyo objetivo es observar los fallos que ocurren durante dicha etapa, interpretarlos de forma estructurada y ejecutar acciones de remediación gobernadas, sin reemplazar ni modificar los mecanismos nativos de resolución de dependencias provistos por las herramientas subyacentes.

En el nivel de contexto del modelo C4, la arquitectura sitúa al Sistema de Resiliencia como un actor central que interactúa con el pipeline CI/CD y con los repositorios de dependencias externos. El pipeline continúa siendo responsable de la ejecución de las etapas de construcción, incluyendo la resolución de dependencias, mientras que el sistema de resiliencia actúa únicamente cuando se produce un fallo relevante. Esta relación permite mantener una separación clara de responsabilidades: el pipeline ejecuta, y el sistema propuesto gestiona la resiliencia ante el fallo, mientras que los gestores de dependencias se encargan de manejar las referencias.

La interacción entre el pipeline y el sistema de resiliencia se basa en el intercambio de eventos o señales de fallo, tales como códigos de error, logs o resultados de ejecución. Estos eventos son consumidos por el sistema de resiliencia, que los procesa mediante un flujo interno compuesto por mecanismos de detección, clasificación y toma de decisiones. A partir de este análisis, el sistema puede ejecutar acciones de remediación previamente definidas o, en su defecto, pro-

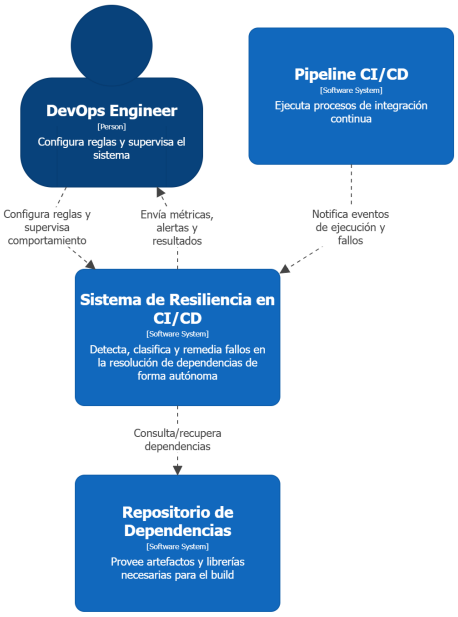


Figura 1. Diagrama general de contexto de la arquitectura.

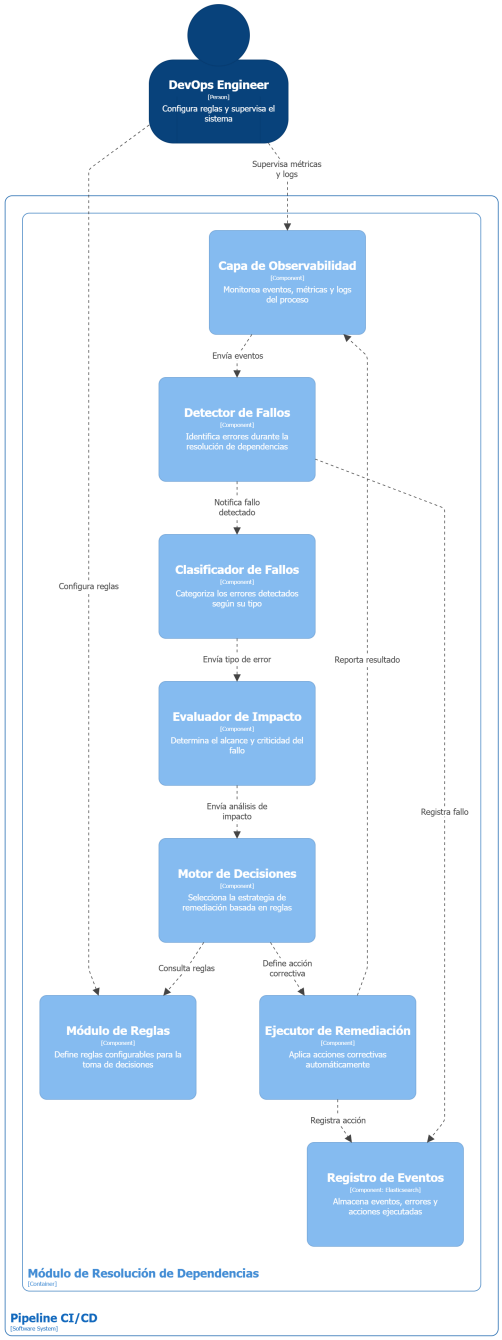
porcionar información diagnóstica enriquecida al pipeline o a los responsables de operación.

Un aspecto clave de la arquitectura es su independencia tecnológica. El modelo no asume un ecosistema de herramientas específico ni un proveedor concreto de CI/CD, lo que permite su aplicación en contextos heterogéneos. De igual forma, la arquitectura no impone un único repositorio de dependencias, sino que contempla la interacción con múltiples fuentes externas, como repositorios públicos o privados, manteniendo la flexibilidad necesaria para adaptarse a distintos entornos organizacionales.

El diagrama de contexto mostrado en la figura 1 (C4 Nivel 1) representa visualmente estas relaciones, destacando los límites del sistema, los principales actores y los flujos de interacción a alto nivel. Dicho diagrama sirve como punto de partida para comprender la propuesta arquitectónica en su conjunto y establece las bases para la exploración posterior de vistas más detalladas, como la arquitectura de contenedores, componentes y despliegue.

El diagrama de componentes de la figura 2 presenta la estructura interna del Módulo de Resolución de Dependencias, elemento central de la propuesta arquitectónica para mejorar la resiliencia en pipelines CI/CD. Este módulo está conformado por componentes especializados que colaboran para detectar, analizar y remediar fallos durante la resolución de dependencias. Su diseño desacoplado favorece la mantenibilidad, extensibilidad y adaptación del sistema.

La capa de observabilidad recopila eventos, métricas y



Component View: Pipeline CI/CD - Módulo de Resolución de Dependencias
martes, 23 de junio de 2026, 8:38 p. m. hora estándar central

Figura 2. Diagrama de componentes de la arquitectura.

registros generados durante la ejecución del pipeline, proporcionando la información necesaria para el monitoreo del sistema. A partir de estos datos, el detector de fallos identifica anomalías y errores relevantes en tiempo de ejecución.

Una vez detectado el fallo, el clasificador de fallos determina su categoría, mientras que el evaluador de impacto analiza

su alcance y criticidad dentro del pipeline. Esta información es utilizada por el motor de decisiones, encargado de seleccionar la estrategia de remediación más adecuada según un conjunto de reglas predefinidas.

Las reglas son gestionadas por el módulo de reglas, que permite configurar el comportamiento del sistema sin modificar su implementación. Posteriormente, el ejecutor de remediación aplica de forma automática las acciones correctivas definidas, contribuyendo a reducir la intervención manual y mejorar la continuidad operativa.

Finalmente, el registro de eventos almacena la información relacionada con los fallos detectados y las acciones ejecutadas, facilitando la trazabilidad, auditoría y análisis posterior del proceso de remediación.

La prueba de concepto combina múltiples patrones arquitectónicos que actúan de forma complementaria para fortalecer la resiliencia y la disponibilidad del sistema. Como patrón estructural principal, se adopta una arquitectura orientada a eventos, donde los servicios se comunican mediante mensajería asíncrona, eliminando el acoplamiento temporal y permitiendo que los eventos continúen procesándose aun cuando alguno de los componentes se encuentre temporalmente fuera de servicio. Sobre esta base, la solución implementa una arquitectura de microservicios con responsabilidad única, en la que cada servicio ejecuta una función específica dentro del flujo de detección, análisis y remediación de fallos. Este desacoplamiento facilita el despliegue, la actualización y el escalado independiente de los componentes, limitando la propagación de errores y mejorando la disponibilidad general del sistema. La toma de decisiones se realiza mediante un motor de reglas basado en primera coincidencia, el cual externaliza la lógica de remediación en un conjunto de reglas configurables e interpretables. Esta aproximación incrementa la flexibilidad de la solución al permitir modificar las políticas de respuesta sin alterar el código de los servicios, además de favorecer la trazabilidad y el mantenimiento de la plataforma.

IV. RESULTADOS

Con el propósito de validar la viabilidad técnica del modelo arquitectónico propuesto, se implementó una prueba de concepto (PoC) compuesta por once servicios contenedorizados, orquestados mediante Docker Compose y comunicados a través de un bus de eventos sobre RabbitMQ, con persistencia de resultados en PostgreSQL, capa de observabilidad respaldada en Elasticsearch e integración con un pipeline de CI/CD real (Jenkins) que ejecuta de forma automatizada las pruebas y la verificación extremo a extremo del sistema. Esta implementación ejecuta lógica concreta para cada acción correctiva (reintentos con backoff configurable, limpieza de caché, sustitución de dependencias por equivalentes alternativas y finalización gobernada del pipeline), y persiste cada decisión y su resultado en una tabla relacional (remediation_actions) que permite trazabilidad y auditoría posterior.

La validación del comportamiento del sistema se realizó en dos niveles complementarios, lo que permitió verificar tanto la corrección de la lógica de negocio de forma aislada, como su correcto funcionamiento en un entorno de ejecución distribuido equivalente al de producción.

En el primer nivel, se desarrolló una suite de pruebas unitarias y de integración ligera utilizando el framework pytest, ejecutada sin dependencias de infraestructura externa. Esta suite verifica, en primer lugar, el correcto funcionamiento de la interfaz de administración de reglas, comprobando que las operaciones de consulta, reemplazo completo e incorporación incremental de reglas se comporten según lo esperado. En segundo lugar, se valida el comportamiento del servicio simulador de pipelines, confirmando que su interfaz HTTP de disparo manual publique eventos con la estructura correcta y que permita sobrescribir de manera selectiva los atributos del evento generado (estado, dependencia, versión y mensaje de error), lo cual resulta indispensable para poder reproducir de manera determinista un escenario de fallo específico durante las pruebas. En tercer lugar, se incluye una prueba que ejecuta de manera íntegra, en memoria y sin necesidad de contenedores, la cadena completa de recuperación ante un fallo de tipo timeout: se parte de un mensaje de error representativo, se obtiene su clasificación, se consulta el conjunto de reglas vigente, se obtiene la decisión de remediación correspondiente y se ejecuta el manejador de acción asociado, verificando en cada paso que el resultado coincida con el comportamiento esperado según el mapeo determinista descrito anteriormente. Esta prueba constituye la validación más representativa de la lógica de negocio del sistema, dado que recorre la totalidad de la cadena de decisión sin intermediarios de infraestructura. Por último, se evalúan de forma aislada los manejadores concretos de cada acción de remediación, incluyendo el caso de una acción no reconocida por el sistema, para el cual se verifica que se aplique un comportamiento de respaldo seguro en lugar de producir un error no controlado. Gracias a instrucciones en el archivo README.md, se pueden reproducir fácilmente estas pruebas, y obtener los resultados mostrados en la figura 3.

En el segundo nivel de validación, se implementó una verificación de extremo a extremo orquestada mediante integración continua, configurada de manera equivalente tanto en GitHub Actions como en Jenkins, con el propósito de demostrar la portabilidad de la estrategia de validación entre distintas plataformas de automatización. Este proceso despliega la totalidad de la arquitectura mediante orquestación de contenedores, deshabilitando temporalmente la emisión periódica y automática de eventos del simulador para garantizar la reproducibilidad del experimento. Una vez confirmada la disponibilidad del servicio simulador, se ejecuta la suite completa de pruebas unitarias contra los servicios ya desplegados, y a continuación se invoca de manera explícita el disparo de un evento de fallo de tipo timeout a través de la interfaz HTTP correspondiente. Finalmente, el proceso consulta de manera reiterada la base de datos relacional hasta confirmar la existencia de un registro de remediación correspondiente a la acción esperada con estado de ejecución exitosa, lo cual constituye la evidencia empírica de que la

Figura 3. Resultados de pruebas de integración.

cadena completa de eventos se ejecuta correctamente en un entorno distribuido real.

La combinación de ambos niveles de validación permite sostener que el comportamiento determinista observado a nivel de unidad se preserva cuando el sistema opera bajo las condiciones de concurrencia, latencia de red y persistencia propias de un entorno distribuido, lo que respalda la viabilidad del enfoque propuesto como base para un sistema de resiliencia de mayor escala.

El diseño del sistema incorpora mecanismos explícitos de tolerancia a fallos de infraestructura, particularmente en lo relativo al establecimiento de conexiones con los servicios de mensajería y persistencia: cada servicio implementa una política de reintento de hasta ciento veinte intentos con un intervalo de un segundo entre cada uno, equivalente a dos minutos de tolerancia ante demoras en la disponibilidad de dichos servicios durante el arranque del sistema. Adicionalmente, cada evento generado conserva una referencia explícita al evento que lo originó, lo que permite reconstruir de manera completa la cadena causal de cualquier fallo procesado por el sistema a partir de los registros almacenados en la base de datos de auditoría.

La ejecución de la PoC confirma que el flujo completo definido por el modelo arquitectónico opera de manera consistente y sin intervención manual. Sobre una corrida representativa registrada en los logs estructurados de los servicios, se generaron 32 eventos de ejecución de pipeline, de los cuales 23 correspondieron a fallos simulados en la etapa de resolución de dependencias y 9 a ejecuciones exitosas, proporción consistente con la tasa de fallo configurada en el simulador para representar un entorno con incidencia significativa de errores. De los 23 fallos generados, el 100 % fue detectado correctamente por el servicio de detección, clasificado por el servicio de clasificación y resuelto mediante una decisión explícita del motor de reglas, evidenciando que no se presentaron pérdidas de eventos ni fallos de comunicación entre los componentes desacoplados del sistema.

La tabla 5 resume la distribución de los 23 fallos detectados según la categoría asignada por el clasificador determinista, así como la severidad base asociada a cada una.

Tabla 5. Casos de fallos detectados durante la evaluación.

Categoría de fallo	Casos detectados
Connection lost	7
Timeout	5
Checksum mismatch	5
404 / Not found	4
Version conflict	2

Cada categoría de fallo fue resuelta de manera determinista mediante una única acción correctiva, conforme a las reglas vigentes en rules_manager/rules.json. La tabla 6 presenta el resultado de la ejecución de dichas acciones, registrado por el servicio failure_solver y persistido en la base de datos relacional.

Tabla 6. Resultados de las acciones de remediación ejecutadas.

Acción de remediación	Ejecuciones	Resultado
change_mirror_and_retry	7	success
increase_timeout_and_retry	5	success
clean_cache_and_retry	5	success
validate_dependency_and_fallback	4	success
finalize_pipeline	2	failed

Tanto el código de la aplicación como los resultados se pueden observar desde la página de zenodo, accediendo a este link: https://doi.org/10.5281/zenodo.21054391.

Otro resultado es la implementación parcial de esta arquitectura en una empresa líder en el diseño y fabricación de equipos de prueba automatizados (ATE) y robótica, en donde se utiliza hasta la clasificación automatizada para reducir los tiempos de detección y clasificación de errores, de manera exitosa. La implementación no se limitó a errores en la etapa de resolución de dependencias, sino que se utiliza para clasificar errores en todas las etapas del pipeline, diferenciando entre errores de infraestructura, errores de usuario, y errores de producto.

V. DISCUSIÓN

De los 23 fallos remediados, 21 (91,3 %) culminaron en un estado exitoso, es decir, el pipeline simulado quedó en condición de continuar su ejecución sin intervención manual. Los 2 casos restantes (8,7 %), ambos clasificados como Version conflict, derivaron en la acción finalize_pipeline, cuyo resultado es intencionalmente fallido: esto no representa una falla del sistema de resiliencia, sino la aplicación correcta de una política de gobernanza que evita que el modelo intente resolver de forma automática un conflicto de versiones semánticamente complejo, lo cual es coherente con las limitaciones arquitectónicas declaradas en la propuesta (el modelo no

busca sustituir el algoritmo de resolución de dependencias ni decidir versiones óptimas, sino contener el fallo y reportarlo de manera explícita). En este sentido, la métrica relevante para la eficacia del modelo es la proporción de fallos para los que existe una acción de remediación automática definida y ejecutable, la cual alcanzó el 100 % de los casos (23 de 23), mientras que la proporción de fallos resueltos sin requerir intervención manual adicional fue del 91,3 %.

Además de estos resultados, se midió la latencia entre etapas del flujo de resiliencia utilizando las marcas de tiempo (timestamps) registradas de forma independiente por cada servicio en su respectivo log estructurado. La tabla 7 resume los tiempos observados para los 23 ciclos completos de detección, clasificación, decisión y remediación capturados durante la ejecución. Los datos están almacenados en el archivo compose_text.txt y se puede acceder también desde la base de datos.

Tabla 7. Tiempos de ejecución entre las etapas del proceso de remediación.

Etapas	Prom. (s)	Med. (s)	Min. (s)	Max (s)
Detección a Clasificación	0,11	0,02	0,02	2,03
Clasificación a Decisión	0,16	0,03	0,03	2,81
Decisión a Remediación	2,34	3,02	1,02	4,17
Ciclo completo	2,61	3,06	1,07	9,01

Los resultados muestran que las etapas de detección y clasificación operan con una latencia mínima (mediana inferior a 0,1 s), dado que ambas son evaluaciones deterministas basadas en coincidencia de patrones sobre el texto del error, sin dependencias externas bloqueantes. La etapa de decisión añade una latencia ligeramente mayor por la consulta HTTP al servicio de reglas (rules_manager), aunque se mantiene por debajo de los 3 segundos en el peor caso observado. La mayor parte del tiempo total del ciclo se concentra en la etapa de remediación (mediana de 3,02 s), explicada por los retardos de espera (backoff) configurados explícitamente en las reglas de negocio y no por ineficiencias del modelo arquitectónico. En conjunto, el ciclo completo de autorrecuperación, desde la detección del fallo hasta la ejecución de la acción correctiva, se completó en un promedio de 2,61 segundos (mediana de 3,06 s), con un máximo observado de 9,01 segundos para el escenario más exigente, el cual es remediación con múltiples reintentos.

Adicionalmente a la simulación operativa, el comportamiento determinista del modelo fue verificado mediante un conjunto de pruebas automatizadas que cubren los puntos críticos del flujo de decisión: clasificación de errores por patrón textual, selección de reglas de remediación según categoría, ejecución concreta de los manejadores de acción (retry, limpieza de caché, sustitución de dependencias) y comportamiento de respaldo (fallback) ante acciones no reconocidas por el motor de decisiones.

Este comportamiento determinista se valida también de forma continua: el propio repositorio de la PoC incorpora un pipeline de integración continua (configurado en Jenkins, con un flujo equivalente en GitHub Actions) que, en cada ejecución, levanta la totalidad de los servicios mediante Docker Compose, espera la disponibilidad del simulador, ejecuta la batería de pruebas unitarias y de integración, dispara mediante el webhook /simulate un fallo determinista de tipo timeout, y verifica la existencia de un registro de remediación en estado success para ese pipeline específico antes de declarar la ejecución como satisfactoria.

VI. CONCLUSIONES

En primer lugar, se concluye que los fallos en la resolución de dependencias representan una causa de interrupciones en pipelines CI/CD, afectando tanto la productividad de los equipos como la confiabilidad del proceso. Estos fallos, asociados a problemas de conectividad, conflictos de versiones o integridad de artefactos, requieren mecanismos automatizados que permitan su detección y tratamiento oportuno.

El modelo propuesto demuestra que la incorporación de una arquitectura basada en detección, clasificación y remediación automática de fallos contribuye a mejorar la resiliencia del pipeline. La integración de componentes como el detector de fallos, clasificador, evaluador de impacto y motor de decisiones permite una respuesta estructurada y adaptable ante distintos tipos de errores. Otro hallazgo relevante es que la aplicación de patrones arquitectónicos como Event-Driven Architecture, microservicios y Rule Engine favorecen la desacoplación, escalabilidad y evolución del sistema, alineándose con las necesidades dinámicas de los entornos CI/CD modernos.

Además, se evidencia que la clasificación de fallos permite priorizar acciones correctivas y optimizar el uso de recursos, enfocando los esfuerzos en aquellos errores con mayor impacto en el pipeline. Finalmente, se concluye que la resiliencia en pipelines CI/CD no depende únicamente de herramientas, sino de la correcta integración entre arquitectura, procesos y mecanismos de automatización, consolidándose como un enfoque clave para mejorar la confiabilidad del desarrollo de software.

Si bien los resultados obtenidos permiten validar la viabilidad conceptual y arquitectónica del modelo propuesto, existen diversas líneas de trabajo futuro que permitirían ampliar su alcance, profundizar su evaluación y aumentar su impacto práctico en entornos de integración continua reales. Una primera línea de evolución consiste en extender el sistema de resiliencia a otras etapas del pipeline CI/CD, más allá de la resolución de dependencias. Etapas como la compilación, la ejecución de pruebas automatizadas, el análisis estático de código o incluso el despliegue presentan fallos recurrentes con patrones identificables que podrían beneficiarse de un enfoque similar de detección, clasificación y remediación gobernada. Esta expansión permitirá evaluar la aplicabilidad general del modelo como una capa transversal de resiliencia para múltiples fases del ciclo de vida del pipeline.

Otra línea relevante de trabajo futuro es la comparación del enfoque determinista basado en reglas explícitas con

enfoques basados en inteligencia artificial o aprendizaje automático. Dicho análisis permitiría contrastar aspectos como la explicabilidad, la gobernanza, la precisión, el costo operativo y la adaptabilidad de ambos enfoques en el contexto de la gestión de fallos en pipelines CI/CD. Esta comparación podría aportar evidencia empírica sobre las ventajas y desventajas de cada aproximación, así como explorar escenarios híbridos donde ambos enfoques coexistan de manera complementaria.

Finalmente, una línea de investigación adicional podría centrarse en la formalización del modelo como un conjunto de patrones arquitectónicos reutilizables, facilitando su adopción en distintos contextos organizacionales. La documentación de buenas prácticas, antipatrones y decisiones arquitectónicas asociadas permitiría que el modelo trascienda el caso de estudio específico y se consolide como una referencia para el diseño de sistemas resilientes en pipelines CI/CD.

REFERENCIAS

An, G. et al. (2024). "Just-in-Time Flaky Test Detection via Abstracted Failure Symptom Matching". En: *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 741-752. DOI: 10.1109/ICSME58944.2024.00078.

Duvall, Paul M., Steve Matyas y Andrew Glover (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley.

Hassan, F., N. Meng y X. Wang (2023). "UniLoc: Unified Fault Localization of Continuous Integration Failures". En: *ACM Transactions on Software Engineering and Methodology* 32.6, pp. 1-31. DOI: 10.1145/3593799.

Shahin, M., M. Ali Babar y L. Zhu (2017). "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". En: *IEEE Access* 5, pp. 3909-3943. DOI: 10.1109/ACCESS.2017.2685629.

Sun, S., D. Friberg y M. Staron (2026). ""Good" and "Bad" Failures in Industrial CI/CD–Balancing Cost and Quality Assurance". En: *Software Engineering and Advanced Applications*. Ed. por D. Taibi y D. Smite. Vol. 16083. Lecture Notes in Computer Science. Cham: Springer. DOI: 10.1007/978-3-032-04207-1_6.

Whitmore, D. J. R. (2025). "Architecting Resilient Continuous Integration and Delivery Ecosystems for Large-Scale Java Enterprises: An Integrated Perspective on Information Needs, Modular Evolution, and Pipeline Governance". En: *International Journal of Next-Generation Engineering and Technology* 2.10, pp. 17-22. URL: https://aimjournals.com/index.php/ijnget/article/view/412.

Zampetti, F. et al. (2021). "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study". En: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 471-482. DOI: 10.1109/ICSME52107.2021.00048.