Pontificia Universidad
JAVERIANA
Cali

Propuesta de implementación de Integración Continua (CI) y Despliegue Continuo (CD) en aplicaciones legacy de la Pontificia Universidad Javeriana Cali

Andersson Morales Agredo

Proyecto presentada(o) como requisito parcial para optar al título de:
Magister en Ingeniería de Software

Director(a):
Luisa Fernanda Rincon Perez, Ph.D

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería
Departamento de Electrónica y Ciencias de la Computación

Cali, Colombia
7 de agosto de 2025

Ficha Resumen
Proyecto de Trabajo de Grado

Título: Propuesta de implementación de Integración Continua (CI) y Despliegue Continuo (CD) en aplicaciones legacy de la Pontificia Universidad Javeriana Cali

1. Área de trabajo: Innovación y Desarrollo TI
2. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
3. Estudiante: Andersson Morales Agredo
4. Correo electrónico: amoralesg@javerianacali.edu.co
5. Dirección y teléfono: Calle 14 E N° 37 oeste 47 - 3167051231
6. Director: Luisa Rincon
7. Vinculación del director: Directora Programa
8. Correo electrónico del director: lfrincon@javerianacali.edu.co
9. Co-Director (Si aplica):
10. Grupo o empresa que lo avala (Si aplica): Pontificia Universidad Javeriana Cali
11. Otros grupos o empresas:
12. Palabras clave(al menos 5): DevOps, Desarrollo de software, Entrega Continua, Integración Continua, Pipeline.
13. Fecha de inicio: 01/12/2024
14. Duración estimada (en meses): 6

15. Resumen: Este proyecto de grado propuso una solución de integración y entrega continua (CI/CD) orientada a mejorar los procesos de desarrollo y despliegue de software en aplicaciones legacy de la Pontificia Universidad Javeriana Cali. La propuesta se fundamentó en el aprovechamiento de herramientas ya disponibles en el entorno institucional y tuvo como objetivo diseñar e implementar una solución técnica adaptada al contexto de sistemas heredados, con énfasis en la automatización, la trazabilidad y la eficiencia operativa.

    La solución fue implementada y evaluada en un entorno controlado, mediante pruebas de concepto realizadas sobre aplicaciones funcionales institucionales. Esta implementación permitió verificar la viabilidad técnica de la propuesta, así como evidenciar su impacto positivo en la reducción de errores post-despliegue, en el fortalecimiento de la calidad del software entregado y en la mejora significativa de los tiempos de entrega.

    El proyecto definió y aplicó métricas para comparar el desempeño del pipeline automatizado frente a los procesos manuales previamente utilizados. Los resultados obtenidos reflejaron mejoras tangibles en trazabilidad, seguridad y eficiencia. Además, se identificaron oportunidades de mejora y se documentaron recomendaciones técnicas para una adopción escalable de la solución. Finalmente, esta experiencia puede servir como referente replicable para otras organizaciones académicas que enfrenten desafíos similares en la modernización de sus procesos de entrega de software.

Santiago de Cali, 16 de junio de 2025

Ingeniero:
Jorge Enrique Alvarez
Director Posgrados de Ingeniería
Facultad de Ingeniería y Ciencias
Pontificia Universidad Javeriana - Cali

Con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el Trabajo de Grado y posteriormente optar por el título de Magíster en Ingeniería de Software, nos permitimos presentar a su consideración el proyecto de Trabajo de Grado denominado Propuesta de Implementación de Integración Continua (CI) y Despliegue Continuo (CD) en aplicaciones legacy de la Pontificia Universidad Javeriana Cali, el cual será realizado por el estudiante Andersson Morales Agredo con código 01000000606 perteneciente al énfasis en Software, bajo la dirección de la profesora Luisa Fernanda Rincon Perez.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer la evaluación de este Anteproyecto ante el Tribunal que para el efecto se designe, toda vez que ha revisado cuidadosamente el documento y avala que ya se encuentra listo para ser presentado oficialmente.

Atentamente,

Andersson Morales A.

Digitally signed by Luisa Rincón
DN: OU=Dpto Elect y Ciencias Comput - Facultad Ing y Ciencias, O=Pontificia Universidad Javeriana Cali, CN=Luisa Rincón, E=lfrincon@javerianacali.edu.co
Reason: I am the author of this document
Location:
Date: 2025.05.15 17:30:00-05'00'
Foxit PDF Reader Version: 2024.4.0

Firma	Firma
Andersson Morales Agredo	Luisa Fernanda Rincon Perez
C.C. 1151954898 de Cali	C.C. 1058817701 de Neira (Caldas)

Figura 1: Carta presentación proyecto - Elaboración Propia

Facultad de Ingeniería y Ciencias

Posgrados Facultad de Ingeniería y Ciencias
Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, agosto 4 2025

Autor: Andersson Morales Agrego

Título del Trabajo de Grado: "Propuesta de implementación de Integración Continua (CI) y Despliegue Continuo (CD) en aplicaciones legacy de la Pontificia Universidad Javeriana Cali"

Director: Luisa Fernanda Rincon Perez

Como director del proyecto de grado "Propuesta de implementación de Integración Continua (CI) y Despliegue Continuo (CD) en aplicaciones legacy de la Pontificia Universidad Javeriana Cali" he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Digitally signed by Luisa Rincon
DN: OU=Dpto Elect y Ciencias Comput - Facultad Ing y Ciencias, O=Pontificia Universidad Javeriana Cali, CN=Luisa Rincon, E=lfrincon@javerianacali.edu.co
Reason: I agree to specified parts of this document
Location:
Date: 2025.08.06 10:07:31-05'00'
Foxit PDF Reader Version: 2024.4.0

Firma del Director del Trabajo de Grado

Figura 2: Acta Corrección - Facultad de Ingeniería y Ciencias PUJ

Índice

Índice de figuras

Índice de tablas

Resumen

Este proyecto de grado propuso una solución de integración y entrega continua (CI/CD) orientada a mejorar los procesos de desarrollo y despliegue de software en aplicaciones legacy de la Pontificia Universidad Javeriana Cali. La propuesta se fundamentó en el aprovechamiento de herramientas ya disponibles en el entorno institucional y tuvo como objetivo diseñar e implementar una solución técnica adaptada al contexto de sistemas heredados, con énfasis en la automatización, la trazabilidad y la eficiencia operativa.

La solución fue implementada y evaluada en un entorno controlado, mediante pruebas de concepto realizadas sobre aplicaciones funcionales institucionales. Esta implementación permitió verificar la viabilidad técnica de la propuesta, así como evidenciar su impacto positivo en la reducción de errores post-despliegue, en el fortalecimiento de la calidad del software entregado y en la mejora significativa de los tiempos de entrega.

El proyecto definió y aplicó métricas para comparar el desempeño del pipeline automatizado frente a los procesos manuales previamente utilizados. Los resultados obtenidos reflejaron mejoras tangibles en trazabilidad, seguridad y eficiencia. Además, se identificaron oportunidades de mejora y se documentaron recomendaciones técnicas para una adopción escalable de la solución. Finalmente, esta experiencia puede servir como referente replicable para otras organizaciones académicas que enfrenten desafíos similares en la modernización de sus procesos de entrega de software.

Palabras Clave: DevOps, Desarrollo de software, Despliegue Continuo, Integración Continua, Transformación Digital, Procesos, Automatización, Pipeline, aplicaciones legacy.

Abstract

This master's thesis proposed a continuous integration and delivery (CI/CD) solution aimed at improving the development and deployment processes of legacy

software applications at the Pontificia Universidad Javeriana Cali. The solution was based on leveraging existing tools within the institutional ecosystem and sought to design and implement a technical approach adapted to the legacy systems context, with emphasis on automation, traceability, and operational efficiency.

The solution was implemented and evaluated in a controlled environment through proof-of-concept deployments on functional university applications. This implementation demonstrated the technical feasibility of the proposal and revealed a positive impact on post-deployment error reduction, software quality assurance, and improved delivery times.

The project defined and applied performance metrics to compare the automated pipeline against the previously manual processes. The results showed tangible improvements in traceability, security, and efficiency. Additionally, areas for enhancement were identified, and technical recommendations were documented to support future scalable adoption. Ultimately, this initiative serves as a replicable reference for other academic institutions facing similar challenges in modernizing their software delivery processes.

Keywords: DevOps, Software Development, Continuous Deployment, Continuous Integration, Digital Transformation, Processes, Automation, Pipeline, legacy applications.

Introducción

En el contexto actual de la ingeniería de software, caracterizado por la necesidad de entregas rápidas, confiables y sostenibles, la automatización de procesos de desarrollo y despliegue se ha consolidado como un componente estratégico para garantizar calidad, trazabilidad y eficiencia operativa. En este escenario, prácticas como la integración continua (CI) y el despliegue continuo (CD) han sido ampliamente adoptadas en entornos empresariales y tecnológicos, permitiendo acelerar el ciclo de vida del software y reducir los errores humanos asociados a tareas repetitivas y manuales.

Las instituciones de educación superior, aunque tradicionalmente centradas en su misión académica, también dependen de plataformas tecnológicas críticas para su funcionamiento administrativo, académico y de investigación. En este sentido, la Pontificia Universidad Javeriana Cali no es ajena al desafío de mantener sistemas legacy que, si bien son funcionales, presentan limitaciones en términos de escalabilidad, mantenibilidad y agilidad para responder a las necesidades de cambio y evolución.

Actualmente, gran parte de los procesos de desarrollo y despliegue de software en la universidad se realizan de forma manual, sin una estrategia formal de automatización ni integración entre las herramientas utilizadas. Esto genera cuellos de botella, incrementa el riesgo de errores en ambientes de producción y dificulta la trazabilidad y mejora continua. Estas condiciones impactan negativamente en la eficiencia operativa y en la capacidad de innovación del equipo de desarrollo.

Este proyecto de grado propone una solución de integración y entrega continua (CI/CD) orientada a aplicaciones legacy, con el objetivo de automatizar las etapas clave del ciclo de vida del software en un entorno controlado. A partir del diseño y validación de un pipeline CI/CD, alineado con principios de DevOps y buenas prácticas de ingeniería de software, se busca demostrar la viabilidad técnica y los beneficios operativos de adoptar este tipo de enfoque en contextos institucionales similares. La propuesta se implementará utilizando herramientas disponibles y se validará mediante pruebas de concepto que permitan medir mejoras en eficiencia, calidad y estabilidad en los despliegues.

1. Capítulo 1: Aspectos Generales

1.1. Planteamiento del problema

La Pontificia Universidad Javeriana Cali, reconocida por su excelencia académica, enfrenta importantes desafíos en la gestión y evolución de sus aplicaciones legacy, las cuales constituyen el núcleo de sus procesos académicos, administrativos y de investigación. Estos sistemas, desarrollados hace más de una década, sostienen operaciones críticas, pero actualmente presentan limitaciones que impactan negativamente la eficiencia operativa, la calidad del software y la capacidad de respuesta tecnológica de la institución.

Uno de los principales problemas es el prolongado ciclo de desarrollo y entrega. Según un análisis interno, la implementación de nuevas funcionalidades o mejoras en los sistemas heredados puede tardar entre 3 y 6 meses, debido a la complejidad del código, la falta de documentación actualizada y los flujos manuales de trabajo. Se estima que cerca del 70 % de las aplicaciones legacy en operación presentan esta problemática.

Adicionalmente, el proceso de construcción y despliegue del software presenta una dependencia de procedimientos manuales, que derivan en errores humanos y pérdida de tiempo operativo. Por ejemplo, algunos ingenieros generan los archivos .war de forma indebida, exportándolos directamente desde entornos de desarrollo como Eclipse, sin incluir correctamente las dependencias ni aplicar criterios de versionamiento definidos en el archivo pom.xml. Esta práctica genera artefactos incompletos o incompatibles, que al desplegarse producen fallos o comportamientos inesperados en los entornos destino.

Aunque parte del equipo utiliza el repositorio Nexus como gestor de artefactos para controlar versiones y dependencias, su uso no está estandarizado. Esta inconsistencia provoca una falta de trazabilidad y reproducibilidad en los builds, además de propiciar tiempos muertos por correcciones posteriores, reempaquetamiento y pruebas manuales no planificadas.

A esto se suma que, tras generar el .war, los desarrolladores deben subirlo ma-

nualmente a un repositorio GitLab y luego crear una solicitud de cambio (RFC) en la plataforma de gestión de servicios TI. Dentro de esta solicitud, se incluye la URL del archivo, pero en muchos casos dicha URL no se actualiza correctamente, ya que se reutilizan plantillas anteriores sin verificar la información. Esto ha derivado en múltiples errores durante el despliegue. Entre 2022 y 2023, aproximadamente el 40 % de los despliegues realizados manualmente resultaron en fallos que requirieron intervenciones correctivas posteriores.

Finalmente, la ejecución del despliegue recae en un ingeniero de infraestructura, quien descarga el artefacto y lo instala manualmente en el servidor de aplicaciones. Este procedimiento, además de ser repetitivo, puede estar expuesto a fallos por omisión, descarga incorrecta o errores de manipulación. En conjunto, estos factores representan oportunidades de mejora en la cadena de entrega de software, particularmente en lo que respecta a eficiencia, trazabilidad y control.

En este contexto, se identificó una oportunidad valiosa para modernizar y fortalecer el modelo actual de construcción y entrega de software, mediante la adopción de prácticas de integración y entrega continua (CI/CD). Esta estrategia busca automatizar y estandarizar los procesos clave, minimizar errores humanos, asegurar la calidad del artefacto desde su construcción y mejorar significativamente los tiempos de entrega, sin comprometer la estabilidad de los entornos institucionales.

En el marco de las prácticas DevOps, uno de los pilares fundamentales es la automatización de procesos que tradicionalmente han sido manuales, repetitivos y propensos a errores. Dentro de esta automatización, se destacan las actividades relacionadas con la integración continua (CI) y el despliegue continuo (CD) como mecanismos para mejorar la eficiencia, trazabilidad y calidad del software. En este contexto, el presente proyecto tiene como objetivo proponer una solución técnica que permita automatizar el proceso de construcción, gestión de dependencias y despliegue de aplicaciones legacy, sin pretender convertirse en un marco definitivo de adopción institucional. Para ello, se explorarán tecnologías disponibles actualmente en el entorno universitario, como GitLab, GitLab Runner, Nexus y Apache Tomcat, evaluando su integración y viabilidad dentro de un entorno controlado.

1.1.1. Problema

Considerando el contexto de las aplicaciones legacy en la Pontificia Universidad Javeriana Cali y la necesidad de modernizar los procesos de desarrollo y despliegue, la pregunta principal que este trabajo busca responder es: ¿Cómo podemos incluir prácticas de integración continua (CI) y despliegue continuo (CD) para aplicaciones legacy en la Pontificia Universidad Javeriana Cali, utilizando el ecosistema tecnológico actualmente disponible en la institución? Para abordar este problema de manera integral, se plantean las siguientes preguntas específicas:

1. ¿Cuál es el estado actual de los procesos de construcción, gestión de dependencias y despliegue de las aplicaciones legacy en la universidad?

2. ¿Qué elementos técnicos y operativos deben considerarse en el diseño de una solución CI/CD adaptada a este contexto?

3. ¿Qué herramientas del ecosistema institucional pueden integrarse para automatizar el ciclo de entrega de software heredado?

4. ¿Qué resultados se pueden obtener al comparar el proceso manual tradicional con una solución técnica automatizada en términos de calidad, eficiencia y trazabilidad?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Proponer, implementar y validar en un entorno controlado una solución de integración continua (CI) y despliegue continuo (CD) que automatice procesos clave en el ciclo de entrega de aplicaciones legacy de la Pontificia Universidad Javeriana Cali, utilizando herramientas del ecosistema tecnológico institucional.

1.2.2. Objetivos específicos

1. Realizar un inventario del ecosistema tecnológico actual de la Pontificia Universidad Javeriana Cali, identificando las herramientas y componentes adecuados para la implementación de prácticas CI/CD.

2. Diseñar una solución técnica de integración continua (CI) y despliegue continuo (CD) adaptada a las aplicaciones legacy, que sea compatible con la infraestructura institucional y reutilizable en múltiples proyectos.

3. Implementar la solución propuesta en un entorno controlado, mediante pruebas de concepto y generación de artefactos técnicos que validen su funcionalidad.

4. Evaluar la efectividad de la solución implementada en comparación con el proceso manual actual, considerando métricas como tiempo de compilación, tiempos de despliegue y grado de reutilización en distintos proyectos.

1.3. Alcance

Este proyecto se centra en el diseño, implementación y validación en entorno controlado de una solución de integración continua (CI) y despliegue continuo (CD), orientada a aplicaciones legacy desarrolladas en la Pontificia Universidad Javeriana Cali. La solución será propuesta, probada y documentada utilizando herramientas disponibles en el ecosistema tecnológico institucional. El alcance del trabajo comprende:

- El levantamiento y análisis de la infraestructura y herramientas actuales disponibles para CI/CD.

- El diseño técnico de una solución reutilizable, aplicable a múltiples proyectos.

- La implementación de pruebas de concepto mediante pipelines automatizados en entornos controlados.

- La comparación entre el proceso manual actual y la solución técnica propuesta, en cuanto a eficiencia, reutilización y tiempo de entrega.

- La generación de documentación técnica y artefactos listos para ser utilizados en otros proyectos.

El proyecto no contempla la implementación en ambientes productivos institucionales, ni la adopción formal de un marco DevOps a nivel organizacional. Tampoco aborda la refactorización del código de las aplicaciones legacy, ni modificaciones en su arquitectura funcional. La propuesta se desarrolla únicamente como una prueba de viabilidad técnica en un contexto experimental, que puede servir de base para una futura adopción institucional más amplia.

2. Capítulo 2: Revisión de literatura

En este capítulo se analizan los conceptos clave relacionados con la integración continua (CI), entrega continua (CD), DevOps, automatización de despliegues y buenas prácticas en ingeniería de software, especialmente en entornos con sistemas legacy.

Asimismo, se identifican enfoques metodológicos utilizados en experiencias previas de implementación de CI/CD en organizaciones educativas o con infraestructura tecnológica tradicional, como es el caso de la Pontificia Universidad Javeriana Cali. Esta exploración teórica busca no solo respaldar la solución técnica planteada, sino también identificar oportunidades, riesgos comunes y criterios de éxito reportados en la literatura académica.

2.1. Marco Teórico

2.1.1. DevOps

DevOps busca integrar de manera integral las áreas de desarrollo de software y operaciones de TI, promoviendo una estrecha colaboración y comunicación entre estos equipos, que tradicionalmente han trabajado de forma separada. Este modelo pone especial énfasis en automatizar procesos, en la integración y entrega continua, así como en la supervisión constante durante todo el ciclo de vida del software Srinivas-Ramesh. DevOps también se basa en un cambio de mentalidad cultural donde se menciona un conjunto de principios y prácticas que promueven una mayor comunicación, cooperación e interacción entre las partes interesadas Amaro-Pereira-DaSilva. DevOps integra una filosofía, prácticas culturales y herramientas tecnológicas que potencian la capacidad de una organización para desarrollar y entregar aplicaciones y servicios de manera ágil. Esto permite que los productos evolucionen y se mejoren a un ritmo más acelerado que en organizaciones que siguen procesos tradicionales de desarrollo de software y gestión de infraestructura. La rapidez lograda con DevOps facilita ofrecer un mejor servicio a los clientes y aumenta la competitividad en el mercado Acevedo-Munoz-Galvan.

2.1.2. Principios de DevOps

Los principios fundamentales que sustentan DevOps están diseñados para optimizar el ciclo de vida del desarrollo del software, reducir tiempos de entregas y aumentar la satisfacción de los usuarios finales. A continuación, se detallan los principales principios:

- **Colaboración entre equipos:** Uno de los pilares de DevOps es la colaboración constante entre los equipos de desarrollo y operaciones, eliminando las barreras tradicionales que los separan. Este principio busca fomentar una comunicación fluida, permitiendo que los equipos trabajen en conjunto desde las fases iniciales del desarrollo hasta el despliegue en producción. En lugar de trabajar en silos, el enfoque colaborativo asegura que los problemas se aborden de manera más eficiente y se resuelvan antes de llegar a etapas críticas, lo cual mejora tanto la calidad como la rapidez del proceso (Ramesh, 2024).

- **Entrega continua de valor:** el objetivo de DevOps es la entrega continua de valor al usuario final. Este principio implica que el software se libera en incrementos pequeños y frecuentes, asegurando que nuevas características y correcciones lleguen rápidamente a los usuarios. (Chen, 2015) subraya que aunque la entrega continua trae enormes beneficios, como una mejor respuesta a las necesidades del cliente, también presenta desafíos en términos de coordinación y calidad, especialmente en entornos con aplicaciones legacy.

- **Automatización de procesos:** La automatización de procesos es esencial en DevOps para eliminar tareas manuales que suelen ser repetitivas y propensas a errores. En un entorno DevOps, la automatización permite la integración y despliegue continuo (CI/CD), desde la fase de integración de código hasta las pruebas y el despliegue en producción. Según (Wiedemann et al., 2023), la automatización no solo reduce los errores, sino que también acelera significativamente el ciclo de vida del software, permitiendo una mayor frecuencia en las entregas.

En el desarrollo moderno de software, la automatización de procesos es un componente esencial para lograr eficiencia, reducir errores humanos y acelerar la entrega de productos. Las prácticas de Integración Continua (CI) y Despliegue Continuo (CD) han emergido como pilares dentro de enfoques ágiles y DevOps, permitiendo la detección temprana de errores, validaciones automáticas y despliegues más confiables.

Según (Cák & Dakić, 2024), a pesar de la popularidad creciente de plataformas como GitLab para automatizar estos procesos, la configuración inicial de pipelines sigue representando un reto técnico importante. En particular, la generación manual de archivos YAML para definir flujos de CI/CD puede resultar compleja, especialmente cuando no se aplican criterios de estandarización. En entornos donde el versionamiento, las dependencias y los entornos de ejecución no están bien definidos, el riesgo de inconsistencias y errores de configuración aumenta considerablemente

- **Medición Continua:** Es un principio clave dentro del marco DevOps, ya que permite la evaluación constante del rendimiento y la fiabilidad del software a lo largo de su ciclo de vida. Como se menciona en el artículo de (Febrianto et al., 2024a), uno de los enfoques más recientes para mejorar la calidad del software es a través de métricas DevOps que miden la disponibilidad, tolerancia a fallos, seguridad y tiempo medio de recuperación (MTTR). Estas métricas permiten a los equipos de desarrollo detectar y corregir errores rápidamente, asegurando un ciclo de retroalimentación constante y mejorando la fiabilidad del sistema.

- **Pipeline CI/CD en DevOps**

  El término *pipeline* en el contexto de DevOps hace referencia a una secuencia automatizada de pasos necesarios para construir, probar y desplegar software de manera continua y confiable. El pipeline es una de las piezas fundamentales en la práctica de Integración Continua y Entrega Continua (CI/CD), ya que permite orquestar todas las fases del ciclo de vida del software desde la integración del código hasta su despliegue en producción.

(Chen, 2015), un pipeline CI/CD bien diseñado proporciona beneficios significativos como la aceleración de entregas, la detección temprana de errores y la mejora en la calidad del software, aunque también introduce desafíos como la configuración inicial, la dependencia de herramientas específicas y la necesidad de pruebas automatizadas robustas.

En el enfoque DevOps, el pipeline actúa como mecanismo integrador de equipos de desarrollo y operaciones, habilitando una entrega más frecuente y fiable de software, lo cual es clave en entornos que buscan agilidad y resiliencia operativa.

- **Compartir responsabilidades:** La cultura DevOps es un entendimiento mutuo entre los desarrolladores y las operaciones. El elemento más crucial es compartir la responsabilidad del software que desarrollan y despliegan (Bijwe & Shankar, 2022). Esto permite a los equipos compartir responsabilidades de manera eficiente, lo que reduce la incidencia de problemas operacionales y aumente la calidad del software.

2.1.3. Prácticas clave de DevOps

- **Integración Continua (CI):** es una práctica clave en DevOps que permite a los desarrolladores fusionar frecuentemente cambios de código en un repositorio central, como Git. Cuando se propone un cambio, se activa un pipeline automático de compilación y pruebas. Si el código supera todas las pruebas (unidad, aceptación, rendimiento, accesibilidad), los cambios se fusionan en el repositorio. Este enfoque, combinado con la entrega continua, permite identificar problemas rápidamente y liberar nuevas versiones de software con mayor frecuencia, mejorando la calidad y agilidad del desarrollo (Virmani, 2015).

- **Entrega Continua (CD)**: es una práctica clave en DevOps que automatiza el despliegue de cambios de código después de pasar por pruebas automáticas, permitiendo que cada cambio validado pueda ser liberado en producción de forma rápida, segura y sin intervención manual. Esta práctica busca reducir los errores humanos, mejorar los tiempos de entrega y mantener el software en un

estado constantemente desplegable (Febrianto et al., 2024b).

En el contexto del presente proyecto, se logró implementar un pipeline funcional que automatiza el proceso de despliegue en un entorno controlado, lo que representa un avance significativo frente al modelo manual anterior. Si bien aún no se incorporan pruebas automatizadas como etapa previa al despliegue—lo cual impide clasificar la solución como una entrega continua completa según los lineamientos más estrictos, sí se consolidan fundamentos sólidos de automatización y estandarización del proceso de liberación de software.

Por tanto, se puede afirmar que esta implementación constituye una fase inicial o parcial de CD, enfocada en el despliegue automatizado como primer paso hacia una solución CI/CD más robusta y alineada con las buenas prácticas del desarrollo moderno.

2.1.4. Automatización y Herramientas

La automatización en DevOps consiste en implementar procesos automáticos para realizar tareas repetitivas como compilación, pruebas y despliegue de software. Esto reduce la intervención manual, minimiza los errores y aumenta la eficiencia del equipo. Al automatizar estos procesos, se facilita la entrega continua de aplicaciones, lo que acelera los ciclos de desarrollo y mejora el tiempo de respuesta ante cambios o nuevas versiones. En resumen, la automatización permite una mayor agilidad en la entrega de software y mejora la calidad general del proceso.

2.1.5. Efectos observados de CI/CD en proyectos reales

Un estudio reciente ha evaluado el impacto de la automatización en entornos reales mediante la adopción de prácticas CI/CD. Estos enfoques, si bien están diseñados para mejorar la eficiencia y confiabilidad del proceso de entrega de software, también introducen desafíos técnicos que deben ser considerados. (Fairbanks et al., 2023), tras analizar más de 12,000 repositorios en GitHub

y GitLab, evidenciaron que los proyectos que integraron pipelines CI/CD incrementaron su velocidad de commits en un 141.19 %. Este dato respalda la idea de que la automatización permite entregas más frecuentes y agilidad en entornos colaborativos.

No obstante, el mismo estudio identificó un aumento del 321.21 % en la cantidad de issues registrados en proyectos con CI/CD, lo que sugiere que la mayor velocidad de integración puede amplificar la aparición de errores o conflictos no previstos. Este hallazgo resalta la necesidad de que la automatización venga acompañada de estrategias de verificación y control de calidad adecuadas.

En cuanto a las plataformas utilizadas, el estudio también comparó el comportamiento entre GitHub y GitLab, identificando que GitHub presentó una mayor eficiencia en términos de velocidad de commits y manejo de errores. Esto se asocia, en parte, a la disponibilidad de herramientas analíticas más robustas y un ecosistema de integración más maduro.

Finalmente, los autores advierten que la implementación de CI/CD conlleva una curva de aprendizaje considerable, particularmente al momento de configurar los pipelines y garantizar la integración adecuada entre herramientas. Este aspecto es especialmente relevante en entornos donde no existen procesos estandarizados o experiencia previa en automatización. Por ello, validar soluciones en entornos controlados, como el propuesto en este proyecto, se convierte en una estrategia metodológica adecuada antes de considerar una adopción más amplia.

2.1.6. Aplicaciones Legacy

Un sistema legado es un sistema sociotécnico que sigue siendo valioso o crucial para una organización, aunque fue creado con tecnologías o enfoques ya desactualizados. Dado que los sistemas legados a menudo cumplen roles fundamentales para las operaciones de la empresa, es necesario conservarlos (Sommerville et al., 2007). Se consideran legados cuando presentan dificultades para adap-

tarse a cambios o actualizaciones. El acelerado progreso tecnológico provoca que estos sistemas de información queden desactualizados e incapaces de satisfacer las exigencias globales actuales. Estos sistemas, denominados sistemas heredados, no solo representan barreras para el desarrollo de las estrategias tecnológicas en las organizaciones, sino que también limitan la competitividad de las empresas (Bakar et al., 2019).

Las organizaciones a nivel mundial enfrentan grandes retos debido a sus conjuntos de sistemas de información heredados. Algunos de los principales factores empresariales que impulsan la modernización de estas aplicaciones incluyen los siguientes (McAllister, 2011):

- Obsolescencia de las tecnologías, incluido el envejecimiento del hardware, así como bases de datos, entornos de desarrollo de software y lenguajes de programación que están perdiendo soporte.
- Altos gastos operativos y de concesión de licencias.
- Software cada vez más frágil causado por años de mantenimiento y mejoras. Prácticamente todas las departamentos de TI parecen tener una serie de módulos que son particularmente difíciles de actualizar sin causar problemas adicionales. El esfuerzo de mantenimiento de alto volumen da lugar a un largo retraso en las solicitudes de mejora, lo que significa que el apoyo de TI está a la zaga de las necesidades de las empresas.
- Dificultades que integren las aplicaciones heredadas independientes en un panorama de TI basado en Internet que más hace hincapié en la conectividad y el intercambio de datos.
- La jubilación de los miembros del personal de desarrollo, y dificultad para localizar al personal nuevo y de reemplazo con conjuntos de habilidades heredadas.

2.1.7. Reusabilidad como Subcaracterística de la Mantenibilidad

La reusabilidad es una subcaracterística de la mantenibilidad, una de las ocho características de calidad definidas en el modelo de calidad del producto de software según la norma ISO/IEC 25010:2011 (ISO/IEC, 2011).Esta subcaracterística se refiere al grado en que un activo puede ser utilizado en más de un sistema o en la construcción de otros activos. En el contexto de la implementación de pipelines CI/CD para aplicaciones legacy, la reusabilidad implica diseñar procesos y componentes que puedan ser aplicados en múltiples proyectos sin necesidad de modificaciones significativas. En estos casos, la reusabilidad puede lograrse mediante estrategias como:

- Modularidad: Dividir el pipeline en componentes independientes que puedan ser reutilizados en diferentes contextos.
- Parametrización: Utilizar variables y configuraciones externas para adaptar el comportamiento del pipeline según las necesidades específicas de cada proyecto.
- Documentación clara: Proporcionar guías y referencias que faciliten la comprensión y adaptación del pipeline en nuevos entornos.

Adoptar prácticas que fomenten la reusabilidad no solo mejora la eficiencia en el desarrollo y mantenimiento de software, sino que también contribuye a la estandarización y calidad de los procesos dentro de la organización.

2.2. Estado del Arte

La adopción de prácticas DevOps ha sido estudiada por su capacidad de mejorar la colaboración entre los equipos de desarrollo y operaciones, acelerar los tiempos de entrega y aumentar la calidad del software. DevOps ha transformado diversas industrias, desde la tecnología hasta el sector financiero, permitiendo la integración continua (CI) y el despliegue continuo (CD) en entornos

complejos. Sin embargo, su adopción en sistemas heredados y en instituciones académicas presenta desafíos, que requieren enfoques específicos adaptados a las características y limitaciones de cada sector.

- Proceso DevOps en Sistemas Legacy: (Cruz & Albuquerque, 2018) proponen un proceso progresivo para la adopción de DevOps en sistemas legacy, destacando la importancia de herramientas como la integración continua, la automatización de pruebas y la infraestructura como código. El artículo aborda los retos técnicos que enfrentan las organizaciones al migrar sus sistemas heredados hacia arquitecturas más modernas, y resalta que la modernización tecnológica debe realizarse de manera gradual, para evitar interrupciones en los procesos operativos. Este enfoque resulta útil para organizaciones empresariales con recursos suficientes para invertir en nuevas tecnologías y herramientas avanzadas como la infraestructura como código. Sin embargo, en el caso de las universidades, la adopción de prácticas DevOps no puede depender exclusivamente de la implementación de herramientas costosas o sofisticadas.Este proyecto de grado se diferencia al proponer soluciones adaptadas a infraestructuras limitadas, enfocándose en la integración de herramientas de automatización que no requieren grandes inversiones iniciales.
- Caso de estudio en una organización financiera: (Su & Storer, 2023) presentan un caso de estudio sobre la adopción de prácticas DevOps en Barclays, una gran organización financiera. La implementación de DevOps en un entorno altamente regulado como el financiero demostró mejoras significativas en la velocidad y la calidad del despliegue de software. Al igual que en otros estudios, se destacó el uso de pipelines automatizados de integración continua (CI) y despliegue continuo (CD), que ayudaron a reducir errores y optimizar los tiempos de entrega. Sin embargo, el estudio también identificó desafíos culturales y de resistencia al cambio, ya que la transformación DevOps requiere una fuerte colaboración entre los equipos de desarrollo y operaciones. Aunque este caso pertenece al sector financie-

ro, ofrece lecciones valiosas sobre cómo gestionar la adopción de DevOps en entornos donde los sistemas son críticos y la resistencia al cambio es un obstáculo significativo. En el contexto académico, aunque las limitaciones presupuestarias y organizacionales difieren, las barreras culturales son comparables, lo que hace que las estrategias implementadas en este estudio sean útiles como referencia para instituciones como la Pontificia Universidad Javeriana Cali.

Aunque este caso pertenece al sector financiero, ofrece lecciones valiosas sobre cómo gestionar la adopción de prácticas DevOps en entornos donde los sistemas son críticos y la resistencia al cambio es un obstáculo significativo. Si bien las condiciones presupuestales son distintas, las barreras culturales y organizacionales pueden ser comparables, lo que permite extrapolar aprendizajes útiles para contextos académicos como el de la Pontificia Universidad Javeriana Cali.

- Automatización y Sistemas Mainframe: El trabajo de et al. (Martínez Villegas et al., 2022) presenta un caso de estudio sobre la implementación de prácticas DevOps en un sistema de mainframe legado, utilizando un enfoque cuasi-experimental. Los autores destacan cómo la automatización de los procesos de despliegue y la integración de un pipeline continuo mejoraron la velocidad, la trazabilidad y la calidad del despliegue en un entorno de sistemas heredados. Según sus hallazgos, el tiempo de despliegue se redujo significativamente gracias al uso de herramientas como control de versiones y pipelines de despliegue. Aunque este estudio muestra mejoras claras en la implementación de DevOps en un sistema legacy, su enfoque está limitado a sistemas de mainframe, que son comunes en grandes organizaciones empresariales, pero menos relevantes en instituciones educativas. Además, el trabajo se enfoca principalmente en la automatización de procesos técnicos, pero no aborda los desafíos culturales y organizacionales que enfrenta la implementación de prácticas DevOps en contextos con equipos de TI pequeños y recursos limitados, como en el caso de universidades. Este proyecto de grado se diferencia al proponer un enfoque

más integral que, además de la automatización, incluye la creación de una cultura DevOps y la adaptación de prácticas ágiles en un entorno académico con restricciones específicas, como la falta de personal dedicado exclusivamente a operaciones y la necesidad de preservar la estabilidad de sistemas críticos.

- Automatización y Cultura DevOps en Organizaciones Grandes: Trigo et al. (Trigo et al., 2022) ofrecen un análisis detallado de la adopción de prácticas de DevOps en una gran empresa de telecomunicaciones europea, identificando tres factores clave: automatización de procesos, cultura organizacional y apoyo institucional a largo plazo. El artículo destaca la importancia de implementar CI/CD para acelerar el ciclo de desarrollo del software y mejorar la estabilidad del mismo. Si bien los recursos y la escala de la organización permiten la adopción de herramientas avanzadas de automatización, también se observan barreras culturales significativas, como la resistencia al cambio por parte de los equipos y la necesidad de un liderazgo sólido para promover la cultura DevOps. En este proyecto, aunque las barreras culturales son similares, el contexto académico implica un enfoque más gradual y con recursos limitados. A diferencia de la organización empresarial estudiada por Trigo et al., en una universidad, la automatización y la adopción de prácticas DevOps deben ajustarse a restricciones presupuestarias, priorizando la implementación de CI/CD con herramientas de bajo costo y asegurando un apoyo institucional enfocado en la sostenibilidad a largo plazo.

En resumen, si bien los estudios existentes sobre DevOps en sistemas heredados han mostrado enfoques útiles para la automatización y la integración continua, estos modelos a menudo requieren recursos y capacidades que no están disponibles en instituciones académicas como la Pontificia Universidad Javeriana Cali. Este proyeco de grado busca llenar este vacío al proponer un enfoque más accesible y realista, adaptado a las necesidades y restricciones de una universidad, que combina la implementación de CI y CD con la creación de una cultura

DevOps, sin comprometer la estabilidad de los sistemas críticos.

Características	Cruz y Albuquerque	Martínez-Villegas	Su y Storer	Propuesta propia
Enfoque en Sistemas Legacy	Sí	Sí	No	Sí
Automatización de procesos	Sí	Sí	Sí	Sí
Limitaciones presupuestarias	No	No	No	Sí
Uso de CI/CD	Sí	Sí	Sí	Sí
Aplicabilidad en contextos educativos	No	No	No	Sí

Tabla 1: Comparación de características de la propuesta con otros trabajos.

2.3. Antecedentes

2.3.1. Entorno tecnológico en la Javeriana Cali

El Centro de Servicios Informático en la Pontificia Universidad Javeriana Cali cuenta con una infraestructura para el desarrollo, mantenimiento y operación de soluciones de software. Las aplicaciones internas están distribuidas entre distintos equipos, con diferentes grados de madurez tecnológica, y una dependencia de sistemas heredados (legacy) que continúan sosteniendo procesos académicos, administrativos y de investigación. El equipo técnico responsable de estos desarrollos se apoya en herramientas como GitLab para la gestión de código fuente, Nexus como repositorio de artefactos y Apache Tomcat como servidor de aplicaciones. Aunque estas herramientas están disponibles en la infraestructura institucional, su uso ha sido parcial y no integrado bajo un enfoque automatizado.

2.3.2. Estructura del área de TI

La Figura 3 presenta la estructura organizacional del área de Tecnologías de la Información (TI) de la Pontificia Universidad Javeriana Cali. Esta estructura está conformada por diversas jefaturas y coordinaciones que se articulan para dar soporte, liderar iniciativas de innovación y mantener la infraestructura tecnológica institucional.

La estructura incluye los siguientes componentes clave:

- **Jefatura de Infraestructura TI:** lidera los procesos relacionados con servidores, redes, entornos de ejecución y soporte de plataformas institucionales. Bajo su dependencia operan los ingenieros senior y junior de infraestructura, responsables de ejecutar despliegues, realizar respaldos y garantizar la disponibilidad técnica de los servicios.
- **Jefatura de Innovación y Desarrollo TI:** orientada a la gestión de la demanda y la planificación de nuevos desarrollos. Su equipo se encarga de canalizar requerimientos funcionales, priorizar iniciativas y liderar el diseño de nuevas soluciones con el apoyo de líderes técnicos y arquitectos de software.
- **Coordinación de Proyectos TI:** tiene como objetivo principal la dirección estratégica de proyectos tecnológicos, asegurando la alineación de las iniciativas con los objetivos institucionales, así como el seguimiento, evaluación y control de cronogramas, riesgos y entregables técnicos.
- **Coordinación de Soporte:** incluye la supervisión de la mesa de servicios y el soporte técnico directo a usuarios. Los agentes de soporte y mesa de servicio atienden incidentes y solicitudes del entorno productivo, especialmente relacionadas con el uso y disponibilidad de aplicaciones institucionales.
- **Coordinación de Seguridad Informática:** encargada de velar por la protección de los activos digitales, la gestión de vulnerabilidades y el cumplimiento de lineamientos de ciberseguridad.
- **Jefatura de CRM:** se encarga de liderar la gestión de relaciones institucionales con usuarios, garantizando la atención a requerimientos funcionales relacionados con plataformas de interacción, seguimiento y gestión de servicios académicos y administrativos.

Estructura organizacional y roles clave en el proceso de automatización

En la Figura 3 se presenta la estructura organizativa del área de Tecnologías de la Información de la Pontificia Universidad Javeriana Cali. Esta estructura agrupa diferentes jefaturas y coordinaciones especializadas que soportan los procesos institucionales desde diversas perspectivas técnicas y operativas.

En el contexto del presente proyecto de automatización de despliegues con prácticas CI/CD, los roles más relevantes son los pertenecientes a la **Gestión de la Demanda** y al equipo de **Infraestructura TI**:

- **Gestores de la Demanda:** son responsables de la recepción, análisis y formalización de requerimientos funcionales provenientes de las distintas áreas académicas y administrativas. Su rol articula la comunicación entre usuarios y desarrolladores, y permite priorizar iniciativas de mejora continua. En el marco de este trabajo, son quienes validan los cambios a desplegar y quienes desencadenan el flujo de liberación una vez completado el proceso de desarrollo.
- **Ingenieros de Infraestructura TI:** bajo la jefatura correspondiente, este equipo se encarga del aprovisionamiento, configuración y mantenimiento de los entornos técnicos en los que se ejecutan las aplicaciones. Dentro del proceso CI/CD propuesto, su papel es crucial, ya que asumen la configuración de los servidores, la administración de artefactos (como los archivos `.war`), y la ejecución de despliegues controlados mediante GitLab Runner. Además, son responsables de activar procedimientos de *rollback* en caso de fallos en producción.

Estos dos grupos colaboran estrechamente para lograr una entrega continua eficiente, segura y alineada con las necesidades institucionales. El éxito de la solución planteada depende en gran parte de la sinergia entre quienes entienden el negocio (gestores de demanda) y quienes dominan la infraestructura técnica.

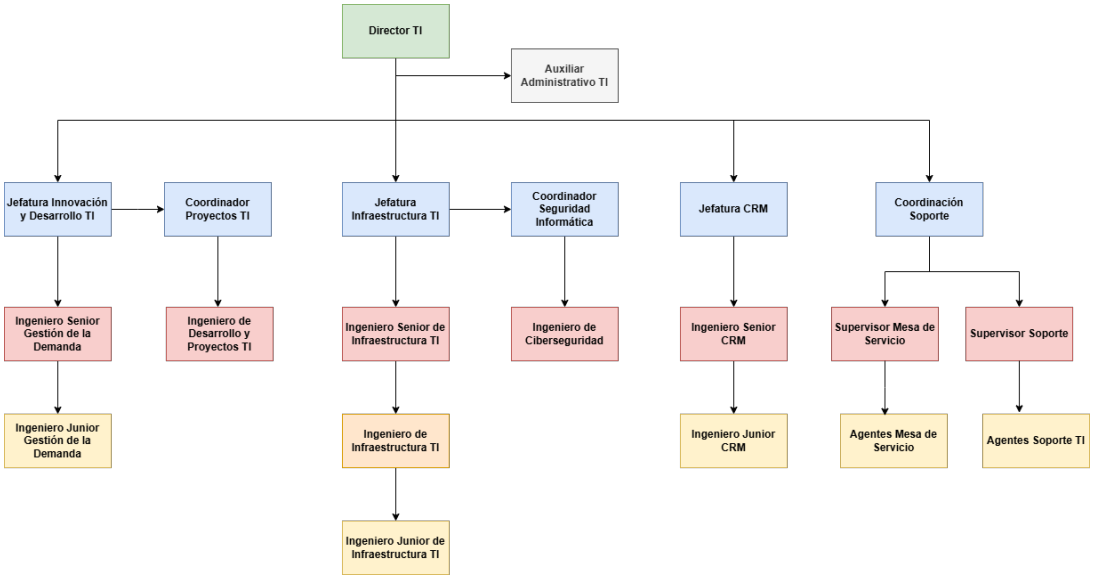


Figura 3: Organigrama funcional del área de TI de la Pontificia Universidad Javeriana Cali - (Elaboración Propia)

Convenciones: azul = liderazgo y coordinación, rojo = roles técnicos senior, amarillo y naranja = roles operativos. Las líneas indican dependencia jerárquica directa.

2.3.3. Proceso tradicional de entrega de software

En el modelo actual de la Pontificia Universidad Javeriana Cali, los procesos de construcción y despliegue de software presentan una alta dependencia de tareas manuales. Este flujo tradicional de entrega ha sido utilizado históricamente en el ecosistema institucional para gestionar las actualizaciones de aplicaciones desarrolladas localmente.

La generación de artefactos `.war` se realiza desde entornos locales (como Eclipse), sin controles formales sobre las dependencias ni validaciones automáticas del `pom.xml`. Posteriormente, el artefacto es cargado en un repositorio GitLab y se tramita una solicitud de cambio (RFC) mediante la mesa de ayuda. Esta solicitud contiene una URL que, en muchos casos, no se actualiza correctamente, debido a la reutilización de plantillas pasadas.

El equipo de infraestructura es el encargado de descargar el archivo, verificar su ubicación y proceder con el despliegue en los servidores Tomcat. Todo este flujo presenta riesgos como fallos por errores humanos, falta de trazabilidad, inconsistencias entre entornos y tiempos muertos entre etapas. La Figura 4 ilustra paso a paso este proceso tradicional, destacando los puntos manuales y las posibles fuentes de error.

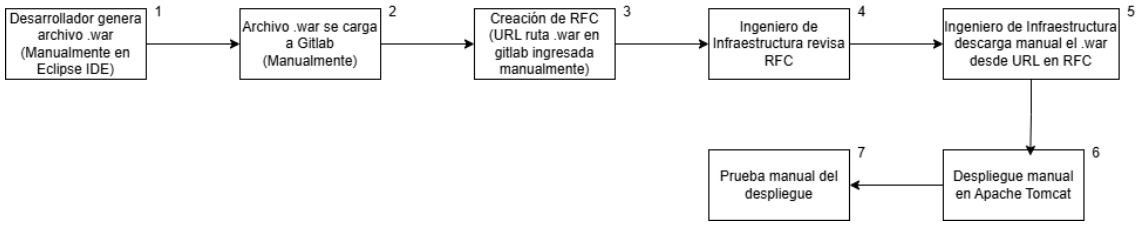


Figura 4: Proceso Actual Despliegue - Elaboración Propia

2.3.4. Iniciativas previas no estructuradas

Algunos equipos técnicos han intentado integrar mejoras parciales como scripts de compilación, uso básico de Nexus para control de versiones, o automatización de pruebas en entornos de desarrollo. Sin embargo, estas iniciativas han sido individuales, no estandarizadas, y en muchos casos no documentadas. La falta de una visión común o de lineamientos técnicos transversales ha impedido que estas mejoras escalen más allá del contexto puntual para el que fueron creadas. No se ha registrado hasta la fecha la implementación de una solución CI/CD institucional, ni un piloto formal que conecte GitLab, Nexus, GitLab Runner y Tomcat en un flujo unificado, trazable y automatizado de integración y despliegue continuo.

2.3.5. Brecha institucional y oportunidad de intervención

Al momento de la definición de esta investigación, la existencia de infraestructura técnica disponible, junto con la ausencia de un modelo de automatización

estructurado, generaba una brecha que representaba tanto un desafío como una oportunidad. Este proyecto busca abordar esa brecha mediante la propuesta, implementación y validación de una solución CI/CD reutilizable, diseñada específicamente para aplicaciones legacy. A diferencia de las iniciativas previas llevadas a cabo hasta ese momento, esta propuesta busca ser documentada, estandarizada, replicable y alineada con buenas prácticas de ingeniería de software. Así, se sientan las bases no solo para resolver un problema técnico puntual, sino también para abrir camino hacia una cultura institucional de automatización, eficiencia y trazabilidad en el ciclo de entrega de software.

Conclusión Capítulo

La revisión de literatura permitió consolidar los fundamentos conceptuales y técnicos sobre los que se sustenta esta propuesta de solución. Se identificaron los beneficios de adoptar prácticas de integración y entrega continua (CI/CD), tanto en términos de eficiencia operativa como de calidad en el proceso de desarrollo y despliegue de software.

Asimismo, se reconoció que la transición hacia enfoques DevOps no es exclusivamente técnica, sino que implica también un cambio cultural y organizacional. Este aspecto es especialmente relevante en contextos como el de la Pontificia Universidad Javeriana Cali, donde existen sistemas legacy, infraestructura tradicional y una cultura operativa aún en proceso de maduración hacia la automatización.

Los hallazgos presentados sirven de insumo para el diseño de una solución técnica alineada con las capacidades actuales de la institución, pero también con visión de crecimiento futuro. En el siguiente capítulo se describe el diagnóstico del ecosistema actual como base para dicha solución.

3. Capítulo 3: Diagnóstico del Ecosistema Tecnológico Institucional

Este capítulo presenta un análisis del entorno tecnológico actual de la Pontificia Universidad Javeriana Cali, con el propósito de identificar los componentes existentes, las limitaciones operativas, las herramientas disponibles y el estado de madurez institucional frente a prácticas de integración y entrega continua (CI/CD). Este diagnóstico sustenta la viabilidad de la solución propuesta en capítulos posteriores y establecer una línea base comparativa que evidencie los aportes del proyecto.

3.1. Introducción al diagnóstico

Como punto de partida para la propuesta de automatización del proceso de entrega de software, se realizó un diagnóstico del ecosistema tecnológico de la Pontificia Universidad Javeriana Cali. Esta etapa sirvió para comprender las condiciones actuales de infraestructura, herramientas, procesos y cultura organizacional, y así garantizar que la propuesta de integración continua y entrega continua (CI/CD) fuera factible, sostenible y adaptada a la realidad institucional.

El diagnóstico se estructuró en cinco dimensiones clave:

1. Infraestructura tecnológica existente.
2. Herramientas y plataformas actualmente disponibles.
3. Flujo operativo vigente en el despliegue de software.
4. Nivel de madurez y adopción de prácticas DevOps.
5. Restricciones y oportunidades de mejora.

3.2. Infraestructura tecnológica institucional

La universidad cuenta con una infraestructura consolidada para el despliegue y operación de aplicaciones internas. Entre los aspectos más relevantes se destacan:

- Servidores Linux dedicados para ambientes de desarrollo, pruebas y producción.
- Acceso mediante redes privadas institucionales con autenticación segura (VPN y SSH).
- Arquitectura clásica basada en servidores Tomcat en entornos monolíticos.
- Infraestructura on-premise y on-cloud.

Sin embargo, esta infraestructura no opera actualmente bajo modelos de contenedorización (Docker) ni orquestación (Kubernetes), lo que impone ciertas limitaciones técnicas a soluciones más modernas de automatización. Por ello, la propuesta se diseñó para integrarse a este entorno sin requerir cambios disruptivos.

3.3. Ambientes y tecnologías por entorno

En la Pontificia Universidad Javeriana Cali se identifican tres entornos fundamentales para el ciclo de vida de desarrollo de software institucional: **Desarrollo**, **Calidad (QA)** y **Producción**. Esta segmentación permite validar progresivamente los artefactos antes de su liberación definitiva, alineándose con las buenas prácticas de DevOps y CI/CD.

- **Desarrollo:** Usado por los desarrolladores para programar, realizar pruebas locales y validar dependencias. Está administrado por los *Ingenieros de Desarrollo e Ingenieros Gestión de la Demanda*, bajo la Jefatura de Innovación y Desarrollo TI.

- **Calidad (QA):** Utilizado para pruebas funcionales y de integración antes de pasar a producción. Su gestión es compartida entre los *Ingenieros de Infraestructura TI* y los responsables de pruebas funcionales en cada proyecto.
- **Producción:** Entorno en el que se alojan las aplicaciones utilizadas por los usuarios finales. Es operado exclusivamente por el *equipo de Infraestructura TI*, asegurando alta disponibilidad y políticas de respaldo.

La mayoría de las aplicaciones institucionales están desarrolladas en tecnologías como **Java EE**, utilizando **Apache Tomcat** como servidor de aplicaciones, **Oracle** como sistema gestor de base de datos y sistemas operativos **Linux**. Esta arquitectura homogénea facilita la automatización y el control de versiones a través de pipelines CI/CD.

A continuación, se presenta una tabla resumen que sintetiza el uso de tecnologías por entorno. Esta información es relevante para identificar posibles puntos de automatización en cada fase y para ajustar los scripts de CI/CD a las restricciones técnicas de cada ambiente.

Tecnología / Entorno	**Desarrollo**	**Calidad (QA)**	**Producción**
Sistema Operativo	Linux	Linux	Linux
Servidor de Aplicaciones	Tomcat	Tomcat	Tomcat
Base de Datos	Oracle	Oracle	Oracle
Acceso SSH	Habilitado	Parcial	Restringido
Copias de Seguridad	Manuales	Manuales	Programadas

Tabla 2: Resumen de ambientes tecnológicos institucionales

Este mapeo de tecnologías por entorno es clave para adaptar los scripts del pipeline CI/CD y asegurar que los despliegues automáticos se comporten de forma coherente con las restricciones de seguridad, acceso y configuración de cada fase del ciclo de vida del software.

3.4. Herramientas técnicas disponibles

Durante el levantamiento se identificaron herramientas claves ya instaladas en la institución, aunque con diferentes niveles de uso y adopción entre los equipos de desarrollo:

- **GitLab**: Plataforma utilizada como repositorio de código fuente. Aunque incluye soporte nativo para CI/CD, sus funcionalidades de automatización son poco utilizadas.
- **Nexus Repository Manager**: Repositorio de artefactos binarios (principalmente archivos .jar y .war), configurado institucionalmente pero con adopción desigual.
- **Apache Tomcat**: Servidor de aplicaciones utilizado ampliamente para ejecutar los sistemas desarrollados en Java.
- **JDK y Maven**: Stack base para construcción de aplicaciones Java legacy.

3.5. Proceso actual de entrega de software

El proceso de entrega de software observado es manual y dependiente del conocimiento tácito de los desarrolladores y del equipo de infraestructura. A continuación se describe el flujo actual:

1. El desarrollador compila la aplicación de forma local utilizando Eclipse o Maven desde consola.
2. El artefacto resultante (.war) se almacena en GitLab como archivo binario.
3. Se crea una solicitud de cambio (RFC) en la plataforma de atención de TI, incluyendo un enlace al artefacto.
4. El equipo de infraestructura descarga el .war desde GitLab, valida manualmente y lo despliega en Tomcat.

5. El respaldo previo del sistema es responsabilidad exclusiva del operador de infraestructura.

Este proceso implica los siguientes riesgos:

- Ausencia de trazabilidad automática de cambios.
- Ausencia de validación automatizada.
- Posibilidad de omitir respaldos.
- Tiempos prolongados entre solicitud de despliegue y ejecución efectiva.

3.6. Evaluación de Madurez DevOps basada en el modelo DORA

Para identificar el nivel de madurez DevOps en la institución, se empleó el modelo propuesto por DORA - DevOps Research and Assessment(Forsgren et al., 2018), ampliamente adoptado a nivel internacional para medir el rendimiento de equipos de desarrollo y operaciones. DORA se enfoca en cuatro métricas clave: frecuencia de despliegue, tiempo de entrega, tiempo medio de recuperación y tasa de fallos en cambios. Su adopción permite clasificar a las organizaciones en niveles de rendimiento bajo, medio o alto, proporcionando una base objetiva para procesos de mejora continua.

Tabla 3: Evaluación del estado actual según el modelo DORA

Métrica DORA	Descripción del estado actual	Nivel de madurez
Frecuencia de despliegue (Deployment Frequency)	Los despliegues en producción se realizan de forma esporádica (1 cada varias semanas), sin una automatización integrada.	Bajo
Tiempo de entrega de cambios (Lead Time for Changes)	Desde que se realiza un cambio hasta que se libera, puede pasar más de una semana, con fuerte dependencia de tareas manuales.	Bajo
Tiempo medio de recuperación (Mean Time to Restore - MTTR))	Ante un fallo en producción, la recuperación depende del personal de infraestructura y puede tomar varios días.	Bajo
Tasa de fallos en cambios (Change Failure Rate)	Se reportan errores con cierta frecuencia luego de despliegues, dada la falta de pruebas automatizadas y controles.	Medio - Bajo

A partir de esta evaluación se concluye que, aunque la institución cuenta con herramientas base que podrían habilitar prácticas DevOps (como GitLab, GitLab Runner y Nexus), su implementación actual refleja un nivel de madurez bajo en términos de prácticas CI/CD. Los procesos presentan una fuerte dependencia de actividades manuales, tiempos extensos entre desarrollo y liberación, y carencia de mecanismos automatizados de validación y recuperación.

El uso de DORA permite evidenciar que el principal cuello de botella se encuentra en la falta de automatización transversal, lo que impacta negativamente la eficiencia, la trazabilidad y la capacidad de respuesta ante incidentes. Esta evaluación sustenta la necesidad de implementar una solución CI/CD robusta, que no solo modernice la infraestructura técnica, sino que eleve el rendimiento

de los equipos y la calidad del software entregado.

3.7. Oportunidades y restricciones

A partir del diagnóstico, se identificaron las siguientes oportunidades para intervención inmediata:

- GitLab no cuenta con runner por lo que es necesario para este proyecto su instalación el cual va habilitar flujos CI/CD sin necesidad de nueva infraestructura.
- Apache Tomcat es ampliamente utilizado, lo que permite estandarizar el despliegue automatizado.
- Los desarrolladores ya usan GitLab, por lo que la curva de aprendizaje es baja.

Restricciones:

- No hay contenedores disponibles: se requiere mantener compatibilidad con el modelo actual.
- Procesos institucionales con desafíos de adaptación al cambio: será necesario acompañamiento y capacitación para facilitar la apropiación de nuevas prácticas.
- Dependencia de infraestructura para despliegues: se debe contemplar validación y comunicación.

3.8. Conclusión del diagnóstico

El diagnóstico presentado detalla el ecosistema tecnológico actual en el que se diseñó la solución CI/CD. Este análisis permite contextualizar el problema, definir lineamientos viables y justificar el diseño de una solución progresiva, reutilizable y alineada con el estado tecnológico de la Pontificia Universidad

Javeriana Cali. De esta manera, se da cumplimiento **Objetivo Específico N.º 1** de este proyecto de grado.

Conclusión del capítulo

El diagnóstico permitió evidenciar una infraestructura funcional pero subutilizada frente a prácticas modernas de automatización. A pesar de contar con herramientas como GitLab, Nexus y Tomcat, su integración es limitada y los procesos de entrega manuales afectan la trazabilidad y confiabilidad. Estos hallazgos fundamentan la necesidad y oportunidad de implementar una solución CI/CD institucionalmente viable.

4. Capítulo 4: Diseño e Implementación de la Solución

Este capítulo describe los componentes y decisiones técnicas que dieron forma a la solución de automatización CI/CD. Se detallan los criterios de selección de herramientas, la estructura modular del pipeline, las estrategias de reutilización, y los mecanismos de despliegue y rollback. La propuesta fue concebida para ser reutilizable, segura y compatible con la infraestructura actual de la universidad.

4.1. Fundamentación y enfoque de diseño

La propuesta de solución parte de una necesidad identificada: automatizar y optimizar el proceso de entrega de software para aplicaciones legacy en la Pontificia Universidad Javeriana Cali. A diferencia de iniciativas centradas en entornos modernos o migración tecnológica, este proyecto se enfoca en sistemas existentes que no pueden ser reestructurados, pero sí mejorados a través de la automatización del ciclo de entrega.

El diseño responde a cuatro principios clave:

- Reutilización: permitir que la solución sea aplicada a múltiples proyectos sin rediseño.
- Compatibilidad: funcionar con la infraestructura y herramientas ya disponibles.
- Escalabilidad técnica: facilitar su adaptación a otros entornos o nuevas etapas (como seguridad o pruebas).
- No intrusividad: no requerir modificaciones en el código fuente legacy.

4.2. Herramientas utilizadas y criterios de selección

El diseño técnico de la solución CI/CD se fundamentó en el uso de herramientas previamente disponibles en el ecosistema tecnológico institucional de la Pontificia Universidad Javeriana Cali. Para su selección, se consideraron criterios técnicos como compatibilidad con la infraestructura existente, facilidad de integración, soporte comunitario y bajo costo de adopción.

Se priorizó el uso de tecnologías y herramientas de código abierto que ya se encontraban en uso parcial, para asegurar continuidad y reducir la necesidad de adquisición de nuevas licencias. Esta elección respondió también a la necesidad de contar con soluciones alineadas con el contexto académico, sostenibles a mediano plazo y que no representaran una carga adicional en términos de soporte o formación.

A continuación se presentan las herramientas seleccionadas:

4.2.1. GitLab

GitLab fue seleccionada como plataforma central para la gestión del ciclo de vida del desarrollo por las siguientes razones:

- **Funcionalidad integrada:** Combina control de versiones, gestión de ramas, Merge Requests y capacidades de CI/CD en un mismo entorno.
- **Disponibilidad institucional:** Ya utilizada por diversos equipos de desarrollo en la universidad.
- **Facilidad de definición de pipelines:** Permite la orquestación de procesos automatizados mediante archivos `.gitlab-ci.yml`.
- **Soporte robusto:** Compatible con flujos Git estándares y repositorios privados protegidos.

Comparativo: A diferencia de Jenkins, que requiere instalación y configuración de múltiples plugins para lograr una funcionalidad similar, GitLab CI/CD

reduce la complejidad operativa al centralizar herramientas clave. Frente a Azure DevOps, GitLab representa una opción libre de licenciamiento y completamente autogestionada, eliminando dependencias externas.

4.2.2. GitLab Runner

GitLab Runner fue configurado como el motor de ejecución para los pipelines definidos en GitLab.

- **Integración nativa:** Diseñado para integrarse sin fricciones con GitLab CI/CD.
- **Instalación controlada:** Fue instalado en un entorno aislado para garantizar seguridad y control de recursos.
- **Compatibilidad con Tomcat:** Su configuración permite ejecutar tareas de construcción y despliegue directamente sobre servidores Apache Tomcat.

Esta elección evitó la necesidad de servidores adicionales para la ejecución de scripts o controladores externos.

4.2.3. Nexus Repository Manager

Nexus fue adoptado como repositorio institucional de artefactos binarios, lo que permitió establecer un ciclo de versionamiento confiable.

- **Gestión de artefactos:** Almacena .JAR, .WAR y dependencias Maven para múltiples proyectos.
- **Integración con Maven:** Compatible con la herramienta de compilación utilizada en proyectos legacy.
- **Trazabilidad y auditoría:** Permite consultar versiones anteriores, generar respaldos y realizar validaciones de integridad.

4.2.4. Apache Tomcat

Como servidor de aplicaciones, Apache Tomcat continúa siendo el estándar de facto dentro de la universidad para desplegar aplicaciones Java.

- **Compatibilidad legacy:** Soporta aplicaciones existentes sin necesidad de migración.
- **Ligero y eficiente:** Fácil de configurar, mantener y monitorear en entornos controlados.
- **Integración simple con CI/CD:** Permite automatizar despliegues mediante SCP y configuración remota.

4.3. Criterios de selección

La decisión final sobre las herramientas utilizadas fue guiada por los siguientes criterios:

1. **Disponibilidad institucional:** Uso previo o instalación factible en entornos internos.
2. **Costo cero de licenciamiento:** Preferencia por herramientas open-source.
3. **Curva de aprendizaje corta:** Facilidad de adopción por parte de los equipos actuales.
4. **Documentación y comunidad activa:** Soporte técnico accesible y buenas prácticas disponibles.
5. **Compatibilidad entre herramientas:** Integración fluida sin necesidad de intermediarios o adaptadores.
6. **Potencial de reutilización:** Posibilidad de aplicar la solución en otros sistemas de la universidad sin rediseñar la arquitectura.

4.4. Solución técnica propuesta

La solución técnica propuesta articula los componentes claves de practicas CI/CD modernas, enfocada en minimizar la intervención humana y garantizar la trazabilidad, reversibilidad y calidad de los despliegues. Este proceso se soporta sobre herramientas ampliamente adoptadas en entornos empresariales como GitLab, GitLab Runner, Apache Tomcat y Nexus, permitiendo una integración fluida entre desarrollo, control de versiones y entornos de prueba o producción.

4.5. Levantamiento de Requisitos

Para garantizar que la solución técnica CI/CD se ajuste adecuadamente a las necesidades reales de los usuarios y al entorno tecnológico institucional, se llevó a cabo un proceso estructurado de levantamiento de requisitos. Este proceso permitió identificar el comportamiento esperado del sistema, así como las condiciones técnicas y organizacionales necesarias para su implementación exitosa. A continuación, se describe la metodología utilizada y los requisitos obtenidos, clasificados según su naturaleza.

4.5.1. Metodología del Levantamiento

El proceso de levantamiento de requisitos se desarrolló a través de un enfoque cualitativo y colaborativo, basado en las siguientes fuentes de información:

- **Revisión de flujos de trabajo actuales:** Se analizaron los procedimientos utilizados por los equipos de desarrollo e infraestructura para la entrega de software.
- **Análisis de limitaciones:** Se identificaron los principales problemas y cuellos de botella en el modelo de entrega tradicional.

- **Entrevistas técnicas:** Se realizaron consultas con ingenieros de Gestión de la Demanda y del área de Infraestructura TI, quienes aportaron su experiencia y visión sobre los desafíos existentes.
- **Buenas prácticas de la industria:** Se tuvo en cuenta la documentación oficial de GitLab CI/CD y recomendaciones de expertos para alinear la solución con estándares actuales.

Este enfoque permitió capturar requisitos funcionales, no funcionales y técnicos, fundamentales para el diseño de una solución sostenible y adaptable.

4.5.2. Requisitos Funcionales

Los requisitos funcionales identifican las funcionalidades necesarias para la operación correcta del pipeline CI/CD. En la Tabla 4 se listan los principales requisitos:

Tabla 4: Requisitos funcionales - Elaboración propia

ID	Requisito	Descripción
RF1	Automatización de compilación	El sistema debe compilar automáticamente el código fuente al realizar merge en la rama principal.
RF2	Generación de artefacto WAR	El sistema debe generar automáticamente un archivo `.war` ejecutable a partir del código fuente compilado.
RF3	Despliegue automatizado	El artefacto generado debe desplegarse automáticamente en el servidor Apache Tomcat.
RF4	Restauración manual (Rollback)	Se debe permitir restaurar manualmente la versión anterior en caso de errores durante el despliegue.
RF5	Validación de ramas	El pipeline debe ejecutarse únicamente si el cambio corresponde a las ramas `main` o `master`.
RF6	Respaldo previo al despliegue	El sistema debe respaldar el archivo `.war` anterior antes de cargar una nueva versión.

4.5.3. Requisitos No Funcionales

En la Tabla 5 se presentan los requisitos no funcionales, que garantizan la calidad y la sostenibilidad del sistema:

Tabla 5: Requisitos no funcionales - Elaboración propia

ID	Requisito	Descripción
RNF1	Reutilización	El pipeline debe permitir su reutilización en otros proyectos sin requerir modificaciones significativas.
RNF2	Portabilidad	La solución debe funcionar correctamente en múltiples entornos, sin depender del uso de contenedores.
RNF3	Seguridad	El acceso al servidor remoto y a tokens debe ser controlado mediante el uso de variables protegidas.
RNF4	Trazabilidad	El pipeline debe registrar resultados y mensajes para facilitar la auditoría y el análisis de errores.

4.5.4. Requisitos Técnicos

Finalmente, se establecieron requisitos técnicos que aseguran la viabilidad operativa y la integración de la solución CI/CD con el ecosistema existente. La Tabla 6 los resume:

Tabla 6: Requisitos técnicos - Elaboración propia

ID	Requisito	Descripción
RT1	Integración con GitLab CI	La solución debe incluir un archivo `.gitlab-ci.yml` que defina todas las etapas del pipeline.
RT2	Uso de GitLab Runner	Se requiere un agente GitLab Runner configurado y operativo para la ejecución de los pipelines.
RT3	Apache Tomcat	Debe existir un servidor accesible vía SSH con Tomcat instalado para ejecutar el despliegue del artefacto.
RT4	Repositorio Nexus	El pipeline debe descargar las dependencias desde el repositorio Maven institucional (Nexus).
RT5	Compatibilidad con JDK 1.8	La compilación debe realizarse con Java 1.8, compatible con las aplicaciones actuales.

4.5.5. Flujo de implementación y actores involucrados:

El flujo inicia con la participación de dos actores principales: el Ingeniero Junior o Senior de Gestión de la Demanda y Desarrollo TI, quien realiza un push o crea una merge request en GitLab. El flujo implica una revisión obligatoria por parte de un Ingeniero Senior, quien evalúa y aprueba el merge request. Una vez aprobado, se activa automáticamente el pipeline de GitLab CI. Este pipeline consta de varias etapas:

La Figura 5 representa el flujo de trabajo automatizado implementado mediante GitLab CI/CD, con el propósito de mejorar los procesos de construcción, despliegue y restauración de aplicaciones desarrolladas en la Pontificia Universidad Javeriana Cali.

Este pipeline está diseñado para ejecutarse únicamente cuando se realizan operaciones sobre ramas relevantes (`main` o `master`), y consta de las siguientes etapas diferenciadas por colores:

- **Naranja (GitLab y GitLab Runner):** Representa el entorno donde se dispara y ejecuta el pipeline. GitLab actúa como el orquestador, mientras que GitLab Runner es el agente que ejecuta cada etapa definida en el archivo `.gitlab-ci.yml`.
- **Amarillo (Build):** Etapa inicial que se encarga de compilar el proyecto utilizando Maven, generando el archivo `.war`. Esta etapa valida la integridad del código fuente antes de ser desplegado.
- **Verde (Deploy):** Responsable del despliegue automático del artefacto compilado en el servidor web remoto Apache Tomcat. Previo al despliegue, esta etapa genera un respaldo del artefacto existente.
- **Gris (Rollback):** Etapa manual que se activa únicamente cuando se detectan fallos posteriores al despliegue. Permite restablecer el artefacto anterior disponible desde el repositorio de respaldos, restaurando el funcionamiento del sistema.

Esta estructura modular, acompañada por reglas de activación condicional, permite que el pipeline sea reutilizable, seguro y trazable. Además, los colores utilizados en el diagrama tienen como propósito facilitar la comprensión visual del flujo técnico y las responsabilidades de cada etapa.

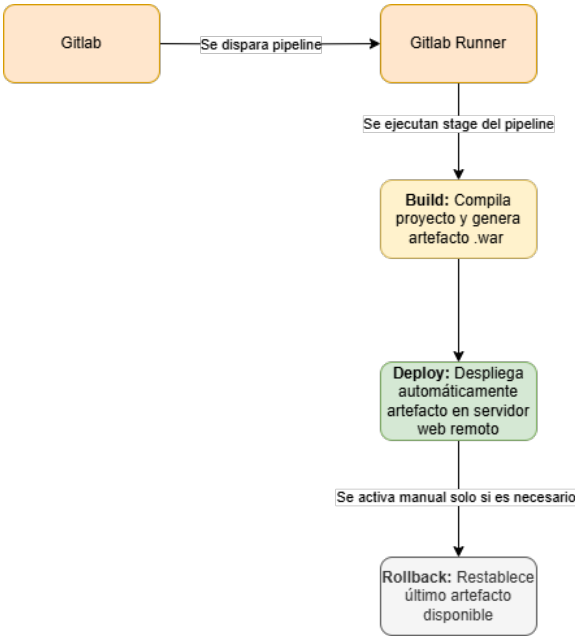


Figura 5: Pipeline solución propuesta - Elaboración Propia

4.5.6. Persistencia y organización de backups

Para garantizar la trazabilidad y recuperabilidad de versiones anteriores, el pipeline organiza los respaldos en carpetas individuales por sistema, utilizando el nombre del artefacto .war como identificador único. Ejemplo: /home/opcgd/backups/nombre-proyecto/nombre-proyecto-20250605194700.war. Esto permite:

1. Conservar múltiples versiones históricas ordenadas cronológicamente.
2. Evitar sobrescritura entre proyectos que reutilizan el mismo pipeline.
3. Facilitar el rollback inmediato desde el pipeline con mínima intervención manual.

4.5.7. Seguridad proceso CI/CD

Todas las operaciones remotas se realizan mediante claves SSH cifradas almacenadas como variables protegidas en GitLab CI/CD, asegurando autenticación

sin exponer credenciales. Los permisos de escritura en los directorios de despliegue son cuidadosamente controlados, permitiendo únicamente al deploy runner autorizado copiar archivos .war.

4.5.8. Diagrama de Contexto

El siguiente diagrama de contexto presenta una visión general de alto nivel sobre la interacción entre los actores responsables del proceso de integración y entrega continua (CI/CD) y el sistema automatizado configurado como parte de la solución propuesta. Este diagrama refleja cómo los roles involucrados colaboran en un entorno controlado, que actualmente no corresponde a un entorno de producción.

En este flujo, el Ingeniero de Innovación (rol desempeñado por ingenieros de gestión de la demanda o de desarrollo TI) realiza acciones como el commit o la solicitud de merge de cambios en el repositorio Git. Estas acciones desencadenan automáticamente la ejecución del pipeline de CI/CD configurado en GitLab. Dicho pipeline ejecuta tareas como compilación, empaquetado del archivo .war, respaldo del artefacto anterior y despliegue automatizado en un servidor Apache Tomcat, todo dentro de un ambiente controlado para pruebas funcionales o validaciones técnicas.

El sistema de CI/CD está compuesto por herramientas como GitLab, GitLab Runner y Tomcat, y permite no solo acelerar el ciclo de despliegue sino también mejorar la trazabilidad de los cambios. Adicionalmente, en caso de una falla funcional posterior al despliegue, se ha habilitado un mecanismo de rollback manual, que puede ser activado por un ingeniero de infraestructura para restaurar la versión previa del artefacto.

A continuación, se presenta el diagrama de contexto correspondiente:

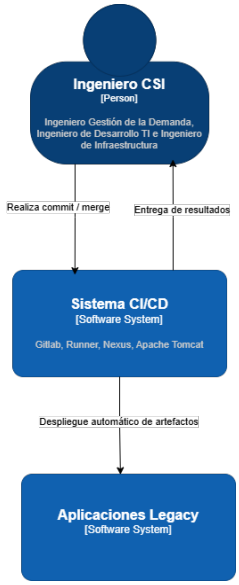


Figura 6: Contexto C4 solución propuesta - Elaboración Propia

Tabla 7: Convenciones del Diagrama de Contexto C4 – Elaboración Propia

Elemento	Significado
Persona (ícono redondo)	Representa a los actores humanos que interactúan con el sistema, como ingenieros TI.
Rectángulo azul	Sistema de software involucrado, como GitLab, GitLab Runner o Tomcat.
Etiqueta de conexión	Describe la interacción entre la persona y el sistema, como Push/Merge o Entrega resultados.
Flechas	Dirección del flujo de información o control entre los elementos.

4.5.9. Diagrama de Contenedores

El siguiente diagrama de contenedores describe cómo se estructura la solución propuesta de integración y entrega continua (CI/CD), especificando los diferentes contenedores de software que participan y cómo interactúan entre sí.

Esta vista proporciona un entendimiento técnico más detallado sobre la solución técnica, ubicando a cada componente en su respectivo entorno lógico de ejecución.

El Ingeniero de Innovación, quien desempeña funciones de desarrollo o gestión de la demanda, realiza un push o merge del código fuente en GitLab, el cual actúa como contenedor de control de versiones y orquestador del pipeline de CI/CD. GitLab está configurado para detectar estos eventos y automáticamente dispara la ejecución del pipeline definido en el archivo .gitlab-ci.yml.

La ejecución del pipeline es gestionada por GitLab Runner, que actúa como agente responsable de interpretar y ejecutar los pasos declarados en el pipeline. Desde este contenedor se realizan dos acciones principales:

1. Compilación y generación del artefacto .war: GitLab Runner descarga las dependencias requeridas desde el servidor Nexus, que funciona como un repositorio de artefactos, y posteriormente genera el archivo .war del proyecto.
2. Despliegue automático del artefacto: El artefacto .war resultante es copiado al servidor Apache Tomcat, el cual aloja la aplicación Java y permite su ejecución.

Todos los contenedores están encapsulados dentro del entorno de infraestructura basado en contenedores o máquinas virtuales, gestionado por la institución. Aunque actualmente esta solución no se encuentra en producción, su diseño es completamente adaptable a entornos productivos.

Este enfoque permite que los ingenieros no solo automaticen el proceso de compilación y despliegue, sino que también reduzcan errores manuales, aceleren las pruebas funcionales y habiliten un mecanismo de recuperación ante fallos mediante rollback, cuando sea necesario.

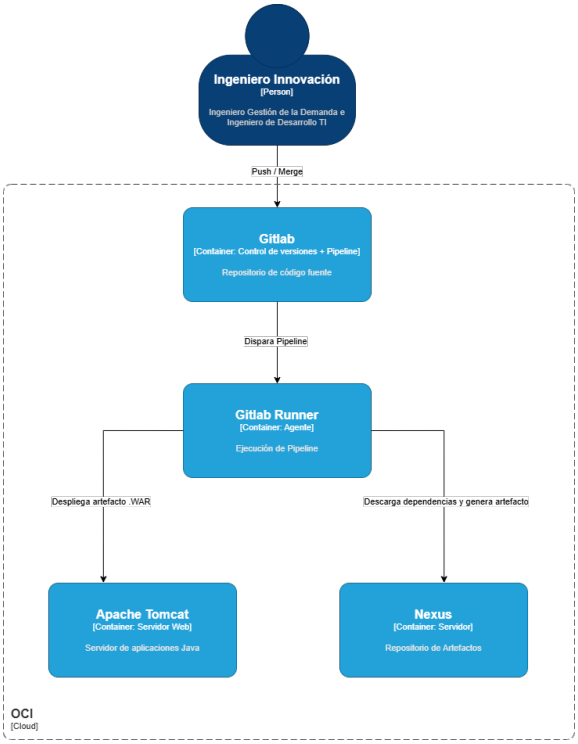


Figura 7: Contenedores C4 solución propuesta - Elaboración Propia

Tabla 8: Convenciones del Diagrama de Contenedores C4 – Elaboración Propia

Elemento	**Significado**
Persona (círculo superior)	Representa a un usuario externo (Ingeniero de Innovación) que inicia las acciones del sistema.
Contenedor (rectángulo con etiquetas)	Componentes que ejecutan funciones clave dentro del sistema, como GitLab, GitLab Runner, Nexus o Tomcat.
Texto en corchetes	Indica el tipo y función del contenedor (por ejemplo, `[Container: Agente]`).
Flechas	Flujo de interacción entre contenedores, por ejemplo, ejecución de pipelines o despliegue de artefactos.
Límite punteado	Representa el entorno en el cual se encuentran los contenedores (como un entorno virtualizado o cloud).

4.6. Decisiones estratégicas y criterios de selección

La solución técnica CI/CD diseñada responde a cinco ejes estratégicos, cada uno sustentado en análisis de riesgo, beneficios y alineación institucional:

1. **Aprovechamiento del ecosistema existente.** Se seleccionaron *GitLab CE* y *Nexus OSS* por su presencia operativa en la universidad, minimizando el tiempo de despliegue inicial, aprovechando los procesos de backup y monitorización central existentes en TI, y evitando la fragmentación de la infraestructura.
2. **Sencillez operacional y rápida adopción.** Se optó por *GitLab Runner* en modo `shell` en lugar de contenedores especializados, reduciendo la complejidad de la orquestación y acelerando la curva de aprendizaje de los equipos al mantener los pipelines definidos exclusivamente en YAML y variables protegidas.
3. **Compatibilidad total con Java EE y Tomcat.** La solución trabaja sobre WARs estándar sin contenedores adicionales ni refactorización de código, respetando las ventanas de mantenimiento académico y evitando riesgos en servicios críticos como matrícula y finanzas.
4. **Modularidad (Principio de Responsabilidad Única).** Cada fase—*build*, *deploy* y *rollback*—se implementa como un *job* autónomo, lo que facilita pruebas unitarias específicas, extensiones independientes y mantiene la trazabilidad de fallos.
5. **Seguridad y gobernanza de secretos.** Todas las credenciales (SSH, tokens Nexus, rutas de backup) residen en variables protegidas de GitLab, con rotación semestral como política recomendada por la unidad de seguridad TI y protocolo de recuperación ante exposición accidental de secretos.

4.7. Estrategias de diseño para la reutilización

Uno de los pilares fundamentales del diseño técnico fue garantizar que la solución pudiera ser adoptada por otros proyectos dentro del ecosistema tecnológico de la universidad. Para ello, se implementaron diversas estrategias arquitectónicas y prácticas de ingeniería de software orientadas a la reutilización, adaptabilidad y sostenibilidad del flujo CI/CD.

Estas estrategias no solo permiten desacoplar componentes, sino que aseguran la independencia de contexto, la coherencia en la automatización y la capacidad de extensión de la solución en nuevos entornos. A continuación, se describen los ejes clave de esta estrategia.

- Modularización del pipeline bajo principios SOLID

  El pipeline fue diseñado bajo un esquema modular y desacoplado, inspirado en principios como el de responsabilidad única (SRP) y abierto/cerrado (OCP). Cada etapa (`build`, `deploy`, `rollback`) se define como una unidad funcional aislada, permitiendo su activación selectiva en función del contexto del proyecto. Este enfoque habilita la extensión sin modificación del código base y facilita la integración de nuevos comportamientos o entornos de despliegue.

  Ejemplo: proyectos que no requieren despliegue automático (por ejemplo, bibliotecas internas) pueden reutilizar la etapa `build` sin heredar lógica innecesaria de despliegue.

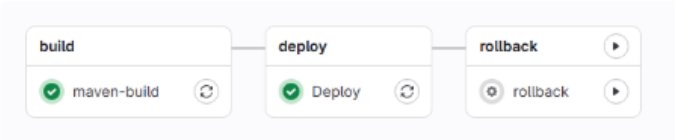


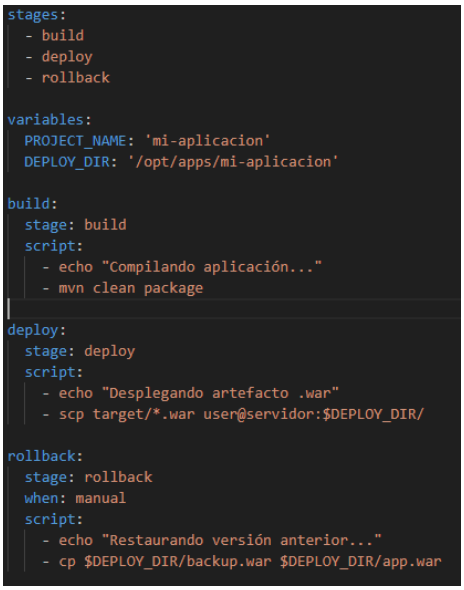
Figura 8: Esquema de pipeline modular basado en principios SOLID. Cada módulo (Build, Deploy, Rollback) es independiente y reutilizable según el contexto del proyecto.

- Definición del pipeline mediante YAML

El pipeline CI/CD fue definido mediante un archivo en formato YAML (Yet Another Markup Language) es un formato utilizado para definir configuraciones. En el contexto de integración y entrega continua (CI/CD), los archivos YAML permiten describir de forma estructurada los pasos del pipeline, las variables, los scripts y las condiciones de ejecución. A través de este formato, herramientas como GitLab CI pueden interpretar y ejecutar automáticamente procesos de construcción, despliegue y rollback. (`.gitlab-ci.yml`), siguiendo la convención establecida por GitLab CI. Este formato permite estructurar de forma clara y legible las etapas (`stages`) y los trabajos (`jobs`), facilitando el mantenimiento y la extensión del flujo de entrega.

Cada etapa del pipeline como `build`, `deploy` y `rollback` se define como un bloque independiente. Esto permite adaptar el comportamiento del pipeline según el contexto del proyecto, el entorno de despliegue o las variables definidas.

Ejemplo de estructura básica en YAML:



```yaml
stages:
  - build
  - deploy
  - rollback

variables:
  PROJECT_NAME: 'mi-aplicacion'
  DEPLOY_DIR: '/opt/apps/mi-aplicacion'

build:
  stage: build
  script:
    - echo "Compilando aplicación..."
    - mvn clean package

deploy:
  stage: deploy
  script:
    - echo "Desplegando artefacto .war"
    - scp target/*.war user@servidor:$DEPLOY_DIR/

rollback:
  stage: rollback
  when: manual
  script:
    - echo "Restaurando versión anterior..."
    - cp $DEPLOY_DIR/backup.war $DEPLOY_DIR/app.war
```

Figura 9: Ejemplo de archivo YAML - (Elaboración Propia)

Este archivo define el flujo secuencial del pipeline e incluye una etapa de `rollback` activable de forma manual para mayor control. Además, se observan buenas prácticas como la parametrización por medio de variables globales y el uso de comandos simples para compilar y desplegar.

El uso de YAML no solo estandariza la definición de los pipelines, sino que también fomenta la reutilización, claridad y trazabilidad de los procesos DevOps definidos.

- Parametrización completa mediante variables externas

  Para garantizar la portabilidad de la solución entre distintos proyectos, se externalizaron todos los parámetros contextuales mediante variables definidas en GitLab CI/CD. Estas variables incluyen:

  - `JAVA_HOME`, `TOMCAT_PATH`
  - `SSH_USER`, `SSH_HOST`, `SSH_PRIVATE_KEY`

  Este mecanismo permite que múltiples equipos utilicen una misma lógica de pipeline, solo es necesario hacer el llamado del pipeline general desde cada pipeline de los proyectos. `.gitlab-ci.yml` original.

- Separación entre lógica reusable y configuración local

  Se implementó un enfoque dual que separa la lógica del pipeline (en el repositorio plantilla) de la configuración específica de cada proyecto (en sus propios repositorios). La lógica se concentra en etapas declaradas y versionadas, mientras que los proyectos consumidores solo deben declarar sus variables y adaptar su estructura de carpetas si es necesario.

- Incorporación de mecanismos de resiliencia: respaldo y rollback

  Uno de los avances más importantes del diseño fue la incorporación de un mecanismo de respaldo automático antes del despliegue. Cada artefacto `.war` existente es respaldado en una carpeta estructurada por nombre del sistema, con timestamp, lo que permite identificar versiones anteriores.

  Adicionalmente, se incluyó una etapa manual de `rollback` que permite a los ingenieros de infraestructura restaurar la última versión válida en caso

de errores en producción o comportamiento inesperado post-despliegue. Esta capacidad evita el uso de snapshots pesados o restauraciones completas, optimizando el tiempo de recuperación ante fallos (MTTR).

- Documentación operativa para adopción institucional

  Como complemento al diseño técnico de la solución CI/CD, se elaboró una guía operativa institucional con el propósito de facilitar su adopción, reutilización y sostenibilidad en el tiempo. Esta guía documenta de forma detallada el uso del pipeline, desde la configuración inicial hasta los procedimientos manuales como el rollback.

5.2 Contenido de la guía

- **Guía paso a paso para la adopción del pipeline**: instrucciones detalladas para clonar, adaptar y ejecutar el pipeline en diferentes contextos, incluyendo ejemplos de uso con ramas `main` y `master`.
- **Políticas de versionamiento de artefactos**: lineamientos sobre el uso del repositorio Nexus institucional y el etiquetado semántico de versiones para garantizar trazabilidad y orden.
- **Protocolo de recuperación ante errores**: acciones a ejecutar en caso de fallas en el pipeline, incluyendo activación manual de rollback y restauración de versiones anteriores.
- **Declaración y uso de variables**: listado de variables de entorno utilizadas en el archivo `.gitlab-ci.yml`, con su propósito y contexto de uso.
- **Validación y socialización**: la guía fue validada por miembros clave de los equipos de TI de la universidad, específicamente:
  - Ingeniero Senior de Infraestructura TI.
  - Ingeniero Senior de Gestión de la Demanda.
  - Ingeniero de Proyectos y Desarrollo TI.

Durante sesiones técnicas realizadas entre abril y mayo de 2025, se revisaron las instrucciones y procedimientos propuestos, asegurando su coherencia con las políticas internas de infraestructura y seguridad.

  - **Disponibilidad**: la guía con su explicación, capturas y recomendaciones técnicas, ha sido incluida como **Anexo A** de este documento, permitiendo su consulta independiente y reutilización futura por parte de otros equipos de desarrollo o infraestructura.

- **Disponibilidad y mantenimiento**

  El documento completo se encuentra almacenado en el repositorio institucional de GitLab, bajo la sección de `Documentación Operativa`. Se acordó que su mantenimiento estará a cargo del área de Gestión de la Demanda, en coordinación con el equipo de Infraestructura, quienes podrán actualizarla conforme evolucionen los entornos y herramientas institucionales.

4.8. Compatibilidad con infraestructura institucional existente

A diferencia de otras soluciones basadas en contenedores u orquestadores (Docker/Kubernetes), la implementación actual fue diseñada para ejecutarse directamente sobre entornos tradicionales (Tomcat sobre Linux), lo que la hace compatible con la infraestructura productiva de múltiples unidades de la universidad. Esto redujo barreras técnicas y permitió validar su viabilidad sin necesidad de reestructurar servidores existentes.

Estas estrategias no solo garantizan la calidad técnica del pipeline, sino que aseguran su sostenibilidad, extensibilidad y alineación con una visión institucional de automatización evolutiva.

La Tabla 9 presenta un resumen de estas estrategias de diseño orientadas a la reutilización institucional, detallando su propósito técnico y el beneficio

específico que aportan al contexto académico en el cual fue implementada la solución.

Tabla 9: Estrategias de diseño orientadas a la reutilización institucional

Estrategia	**Propósito**	**Beneficio Institucional**
Modularización del pipeline	Separar lógica por etapas reutilizables (build, deploy, rollback)	Permite la adopción parcial o total en distintos proyectos según sus necesidades
Uso de variables externas	Parametrizar rutas, entornos y credenciales	Facilita la configuración sin modificar el código fuente del pipeline
Separación lógica-configuración	Mantener la lógica en un repositorio plantilla y la configuración en cada proyecto	Garantiza una solución genérica reutilizable con personalización local mínima
Respaldo automático y rollback	Preservar versiones anteriores del artefacto `.war` antes de despliegue	Mejora la resiliencia, permite recuperación rápida ante fallos
Documentación operativa	Guiar a otros equipos en la adopción del pipeline	Reduce curva de aprendizaje y estandariza el proceso en toda la universidad
Compatibilidad con entornos tradicionales	Evitar dependencias con Docker u orquestadores externos	Asegura que la solución funcione en la infraestructura actual de producción

4.9. Funcionamiento de la Solución

La solución propuesta se implementó en un entorno controlado, configurado para simular las condiciones técnicas de los entornos utilizados por la Pontificia Universidad Javeriana Cali. El entorno incluyó una instancia de GitLab CE, un servidor GitLab Runner, un servidor Nexus y un servidor Apache Tomcat,

todos conectados de forma privada para emular el flujo real de integración y despliegue de aplicaciones legacy.

1. Configuración del GitLab Runner El GitLab Runner fue instalado específicamente como parte de este proyecto en una máquina virtual Linux dedicada. Se utilizó el modo de ejecución shell, ya que no se implementaron contenedores Docker en esta versión. La configuración del runner se realizó con permisos controlados y fue registrado directamente con el proyecto central de CI/CD, que contenía la lógica principal del pipeline reutilizable.

2. Configuración de Nexus y Tomcat Nexus fue configurado como repositorio privado para almacenar los artefactos generados (.war). Las credenciales de acceso fueron definidas como variables protegidas en GitLab para evitar exposición y permitir el uso controlado desde los pipelines. Apache Tomcat fue preparado con un entorno de prueba idéntico al productivo, asegurando que los despliegues fueran realistas y confiables para la validación.

3. Estructura técnica del pipeline CI/CD El pipeline fue implementado mediante un archivo `.gitlab-ci.yml` centralizado en un repositorio base. Este archivo fue diseñado para ser modular, reutilizable y parametrizable.

   Los demás proyectos hacen uso del pipeline mediante la directiva include, lo que permite mantener una única fuente de configuración y facilitar el mantenimiento a largo plazo.

   Fases del pipeline:

   a) build:

   - Ejecuta `mvn clean install` para compilar el código y generar el artefacto `.war`.
   - Valida que no existan errores de compilación ni conflictos de dependencias.
   - Utiliza variables globales para definir versiones, nombres de artefactos y entorno.

b) deploy:

- Se conecta al servidor `Tomcat` mediante `SHH` seguro.
- Copia el archivo `.war` al directorio correspondiente `(webapps/)`.
- Reinicia el contexto de la aplicación o el servicio, según configuración.

4. Características avanzadas del pipeline

- Trazabilidad total: cada ejecución del pipeline queda registrada con logs detallados, estado de cada etapa y artefacto generado.
- Reusabilidad estructural: el pipeline fue construido en bloques reutilizables, permitiendo su adopción por otros proyectos sin reescritura.
- Aislamiento de configuraciones: se utilizaron variables globales y archivos de configuración separados para entornos, credenciales y parámetros.
- Notificación automática: al finalizar cada ejecución, se envía un resumen por correo electrónico o notificación en GitLab al responsable del proyecto.

4.10. Descripción del proceso actual sin CI/CD

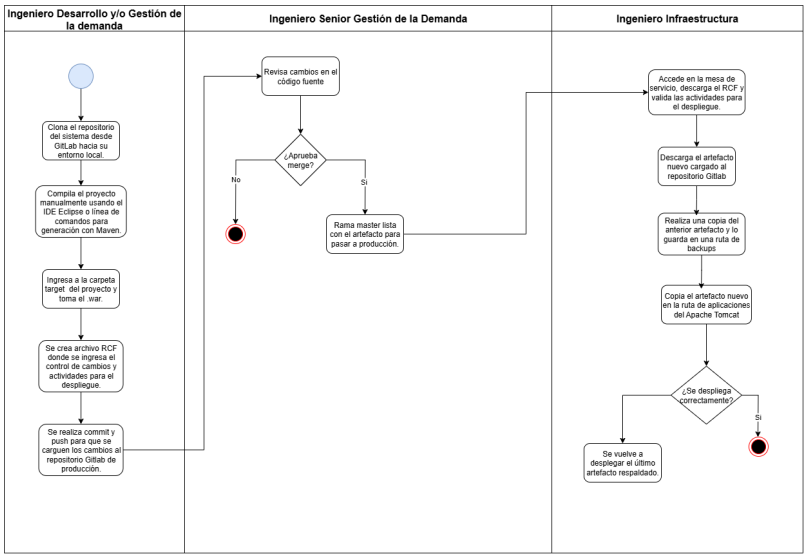


Figura 10: Diagrama actividades proceso actual - (Elaboración Propia)

Tabla 10: Convenciones del Diagrama de Actividades – Elaboración Propia

Elemento	**Significado**
Círculo inicial	Punto de inicio del flujo de actividades. Representa el inicio del proceso tras solicitud de merge.
Rectángulo	Actividad o acción ejecutada por un rol, como revisión, despliegue o activación manual.
Rombo	Punto de decisión que bifurca el flujo de acuerdo con el cumplimiento de una condición.
Flecha	Flujo de secuencia entre actividades o decisiones.
Círculo con borde doble	Punto de finalización del proceso. Marca el fin del flujo del pipeline o del rollback.
Swimlanes	División de responsabilidades por rol: desarrollo, gestión de demanda y operación/infraestructura.

- **Paso 1. Responsable: Ingeniero Gestión de la demanda**
  Clona el repositorio del sistema desde GitLab hacia su entorno local.
- **Paso 2. Responsable: Ingeniero de demanda**
  Compila el proyecto manualmente usando Eclipse o línea de comandos para generalo con Maven.

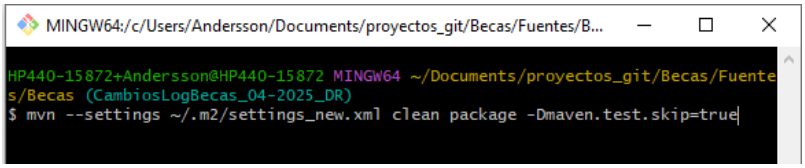


Figura 11: Paso 1 - Proceso actual

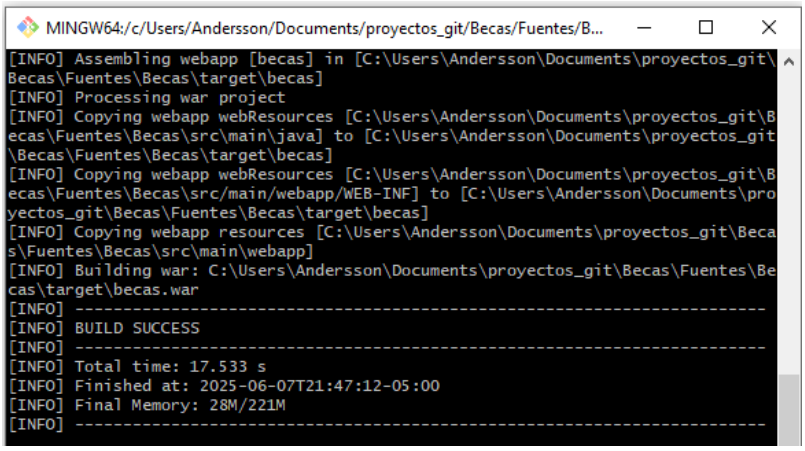


Figura 12: Paso 1.1 - Proceso Actual

- **Paso 3. Responsable: Ingeniero de demanda**
  Ingresa a la carpeta `target` del proyecto que es la ruta donde se genera el artefacto y copia el `.war` en una subcarpeta del proyecto (Pasos a producción - carpeta-cambios)

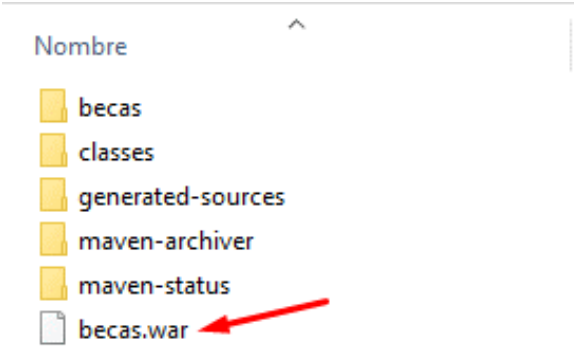


Figura 13: Paso 3 - Proceso Actual

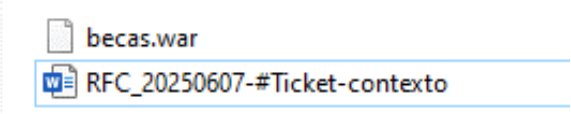


Figura 14: Paso 4 - Proceso Actual

- **Paso 5. Responsable: Ingeniero de demanda**
  En ese archivo RFC de control de cambios se ingresa el contexto del cambio realizado, las actividades que debe realizar el ingeniero de infraestructura, por ejemplo: desplegar archivo `.war`, plan de rollback y URL en gitlab donde se encuentra ese artefacto a desplegar.

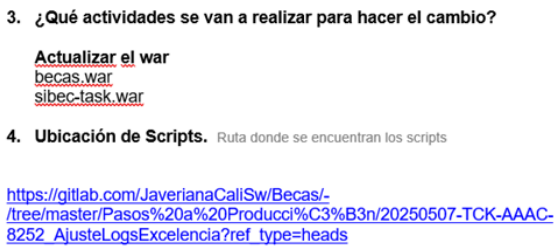


Figura 15: Paso 5 - Proceso Actual

En el punto donde se agrega el la URL donde se encuentra el artefacto es un paso propenso a errores porque muchas veces, se utiliza

una anterior plantilla y esa URL se olvida modificar, por lo tanto, el ingeniero de infraestructura toma ese archivo `.war` anterior y lo despliega, causando un error en el funcionamiento del sistema.

- **Paso 6. Responsable: Ingeniero de demanda**
  Luego de crear la carpeta con el artefacto y archivo RFC, procede a realizar commit y push para que carguen los cambios realizadados en el código fuente y archivos para el paso a producción al repositorio de gitlab.

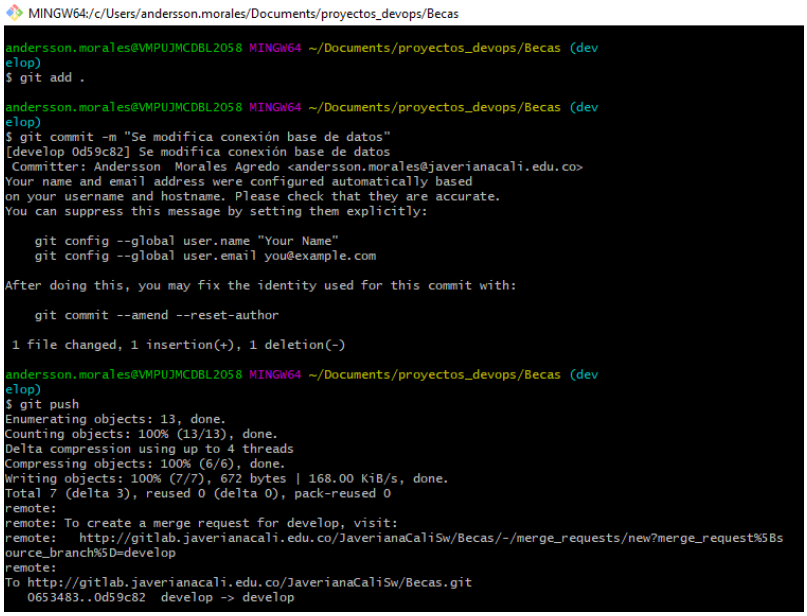


Figura 16: Paso 6 - Proceso Actual

- **Paso 7. Responsable: Ingeniero de Innovación**
  Solicita la revisión y merge request a un ingeniero senior gestión de la demanda en gitlab.

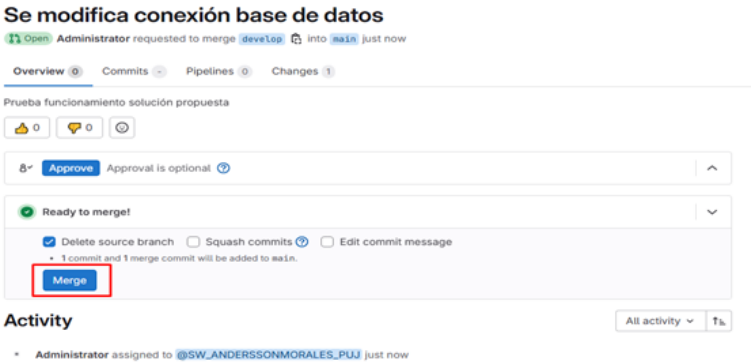


Figura 17: Paso 7 - Proceso Actual

- **Paso 8. Responsable: Ingeniero de Innovación**
  Crea un cambio y reléase en la aplicación de mesa de servicio, donde ingresa lo mismo que en el archivo RFC y envía el cambio a aprobación al ingeniero senior, ingeniero de seguridad y jefes de las unidades.
- **Paso 9. Responsable: Ingeniero de Infraestructura**
  Una vez el cambio es aprobado, el ingeniero de infraestructura accede al release registrado en la mesa de servicios TI, descarga el RFC asociado y accede a la URL del repositorio GitLab indicada en el requerimiento. Desde allí, descarga el archivo `.war` correspondiente al nuevo despliegue. A continuación, realiza una copia de seguridad del archivo `.war` actualmente en producción, almacenándolo en una carpeta de respaldos. Finalmente, reemplaza el artefacto existente por el nuevo archivo .war en la ruta webapps del servidor Apache Tomcat, donde se alojan las aplicaciones web de la universidad.
- **Paso 10. Responsable: Ingeniero de infraestructura**
  Realiza el cierre del release en la mesa de servicio TI y notifica al ingeniero de innovación para que proceda con las validaciones del funcionamiento del sistema.
- **Paso 10. Responsable: Ingeniero de infraestructura e Ingeniero de Innovación**

Si se presentan fallos en el funcionamiento del sistema después del paso a producción o en el proceso de despliegue, el ingeniero de innovación notifica al ingeniero de infraestructura para que proceda con el *rollback* al artefacto previamente funcional. El ingeniero de infraestructura accede al servidor correspondiente, navega hasta el directorio de respaldos y localiza el archivo `.war` anterior. Posteriormente, realiza su despliegue manual en la ruta `webapps` del servidor de aplicaciones `Apache Tomcat`, sobrescribiendo el artefacto defectuoso con la versión estable.

4.11. Descripción de la solución propuesta CI/CD propuesto

A continuación, se describe de manera secuencial el funcionamiento de la solución CI/CD diseñada para optimizar los procesos de integración y despliegue de aplicaciones legacy en la Pontificia Universidad Javeriana Cali. Esta propuesta automatiza tareas previamente manuales, asegurando mayor trazabilidad, control y eficiencia en la gestión de los despliegues. Para facilitar la comprensión, en los pasos que lo requieren se incluyen imágenes ilustrativas del funcionamiento real de la solución.

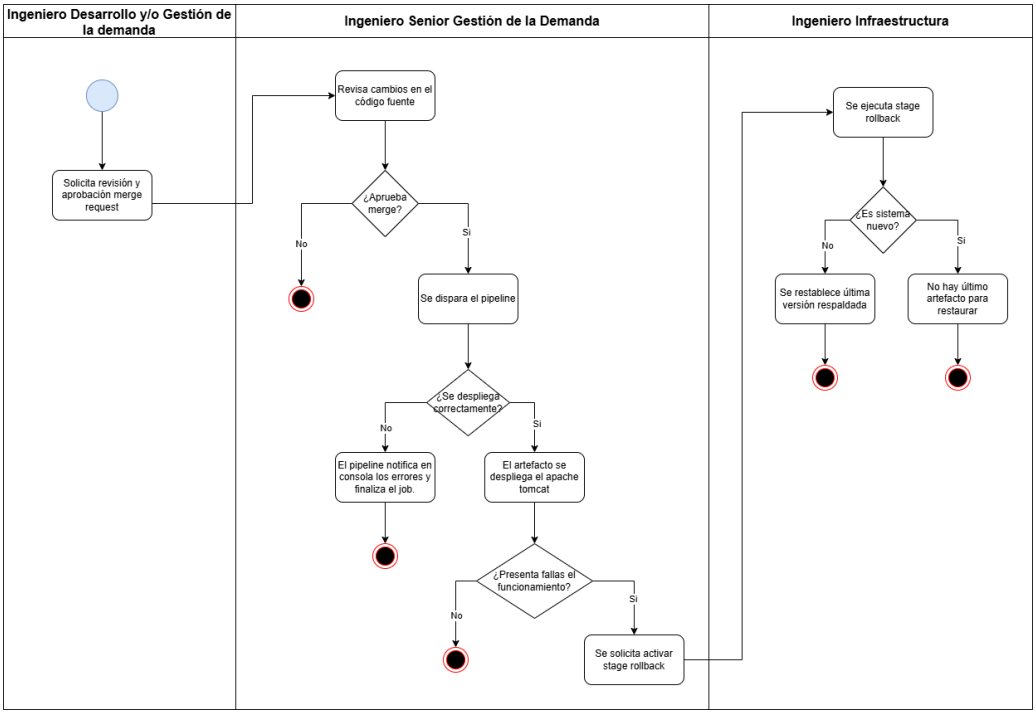


Figura 18: Diagrama Actividades solución propuesta - Elaboración Propia

Tabla 11: Convenciones del Diagrama de Actividades – Elaboración Propia

Elemento	**Significado**
Círculo inicial	Punto de inicio del flujo de actividades. Representa el inicio del proceso tras solicitud de merge.
Rectángulo	Actividad o acción ejecutada por un rol, como revisión, despliegue o activación manual.
Rombo	Punto de decisión que bifurca el flujo de acuerdo con el cumplimiento de una condición.
Flecha	Flujo de secuencia entre actividades o decisiones.
Círculo con borde doble	Punto de finalización del proceso. Marca el fin del flujo del pipeline o del rollback.
Swimlanes	División de responsabilidades por rol: desarrollo, gestión de demanda y operación/infraestructura.

- **Paso 1. Responsable: Ingeniero de demanda**
  Clona el repositorio del sistema desde GitLab hacia su entorno local.
- **Paso 2. Edición y commit del código:** El desarrollador realiza cambios en el código fuente desde su entorno local y hace commit al repositorio GitLab.

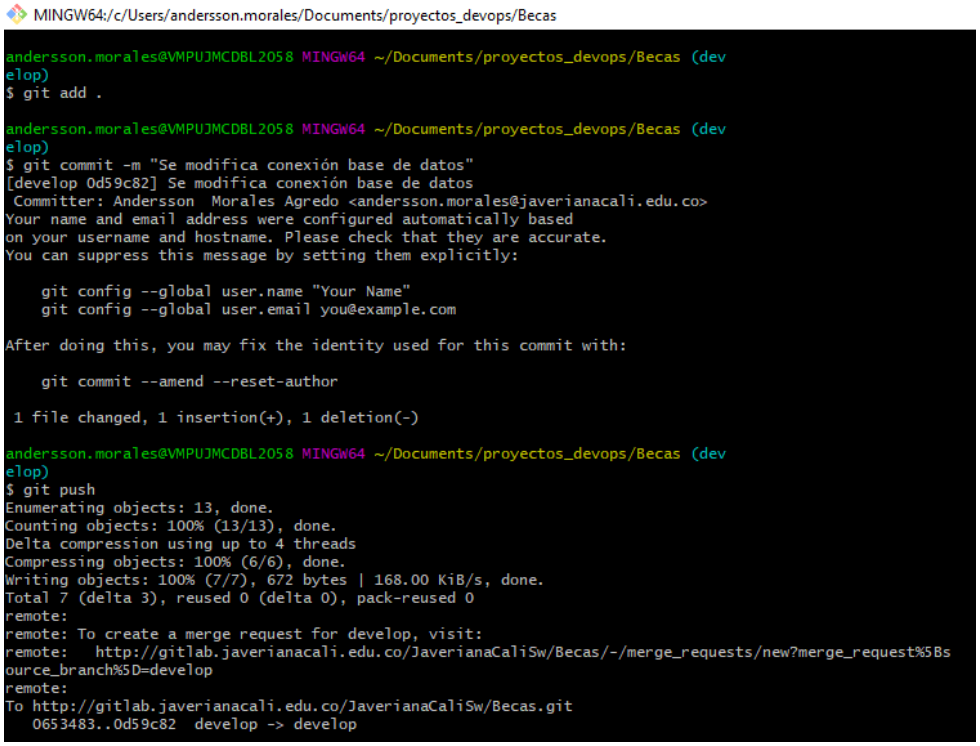


Figura 19: Paso 2 - solución técnica

- **Paso 3. Solicitud de merge:** Se crea una Merge Request (MR) para fusionar la rama de desarrollo con la rama principal. Esta solicitud es revisada por un ingeniero senior.

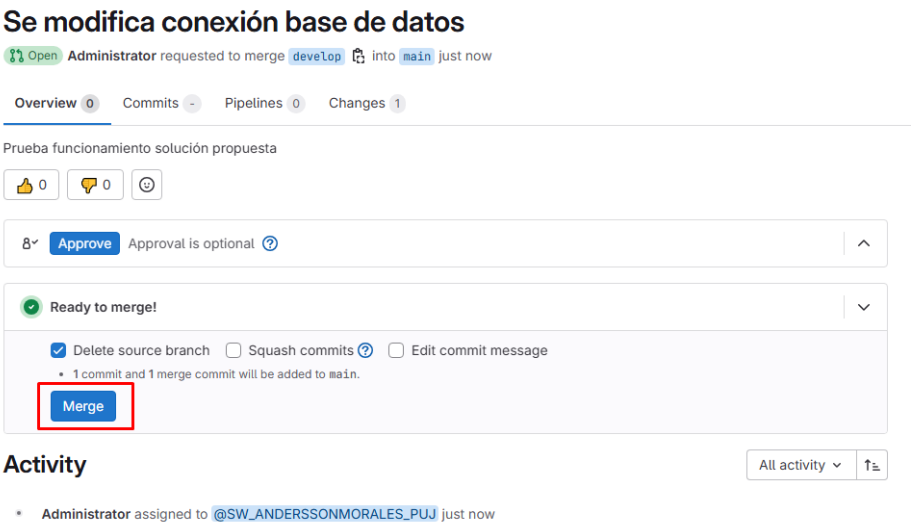


Figura 20: Paso 3 - solución técnica

- **Paso 4. Aprobación y activación del pipeline:** Al aprobar el MR, GitLab dispara automáticamente el `pipeline` definido en el archivo `.gitlab-ci.yml`

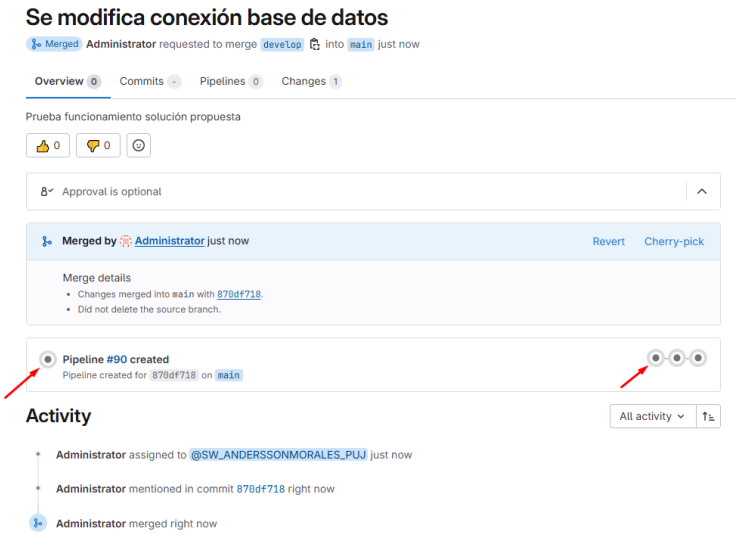


Figura 21: Paso 4 - solución técnica

- **Paso 5. Ejecución del stage 'build':** El GitLab Runner compila el proyecto con `Maven`, genera el archivo `.war` y lo conserva como artefacto.

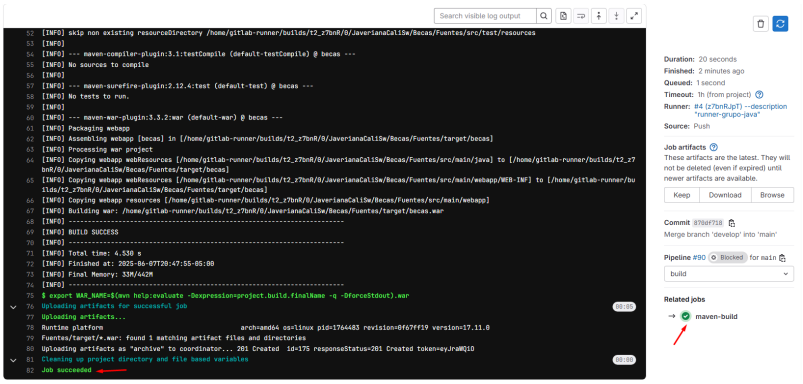


Figura 22: Paso 5 - solución técnica

- **Paso 6. Ejecución del stage 'deploy':** Se verifica si existe una versión previa en Tomcat. Si existe, se respalda. Luego, se despliega el nuevo `.war` en `Tomcat` mediante `scp`(Secure Copy Protocol) permite transferir archivos de manera segura entre servidores a través de una conexión `SSH`, asegurando que el despliegue del artefacto se realice con integridad y confidencialidad.

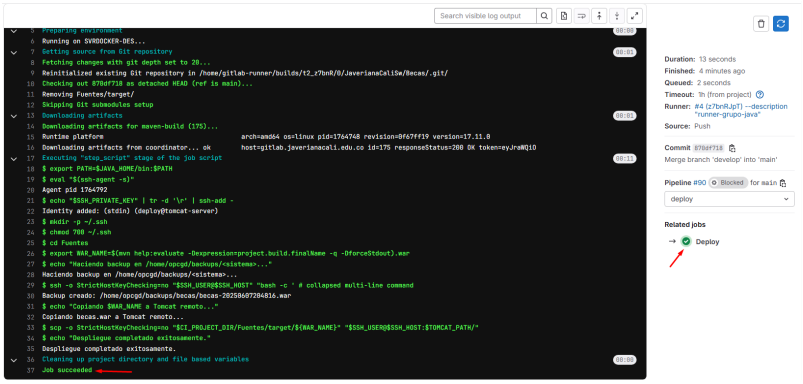


Figura 23: Paso 6 - solución técnica

- **Paso 7. Finalización y reporte:** El `pipeline` notifica su estado

(éxito o fallo). Si fue exitoso, el nuevo sistema queda desplegado.

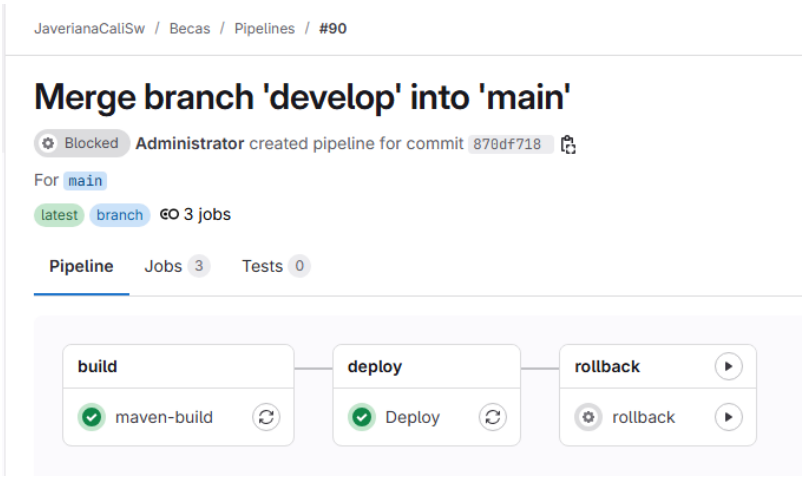


Figura 24: Paso 7 - solución técnica

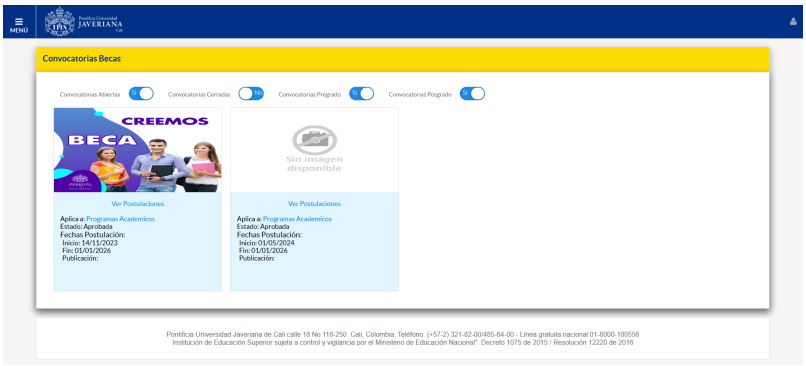


Figura 25: Paso 7 - Aplicación desplegada exitosamente

- **Paso 8. Rollback (manual):** En caso de fallas, un ingeniero de infraestructura puede ejecutar manualmente el stage 'rollback', restaurando el último respaldo automático del `.war`previo.

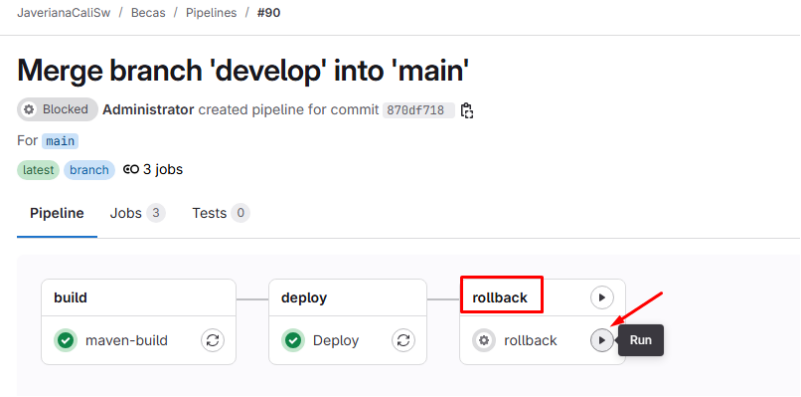


Figura 26: Paso 8 - solución técnica

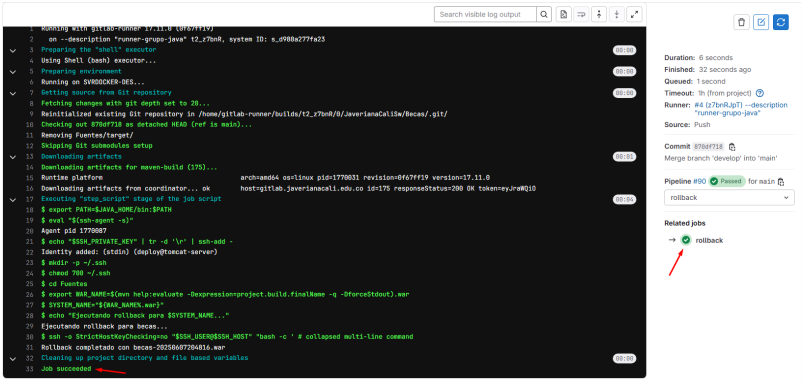


Figura 27: Paso 8.1 - solución técnica

6.5.1. Comparación operativa entre procesos: antes y después

Una vez documentados tanto el proceso tradicional como el modelo propuesto con CI/CD, es posible establecer una comparación directa entre ambos enfoques de entrega de software. A continuación, se analiza el cambio en cinco etapas clave del proceso: compilación, carga del artefacto, documentación, despliegue y rollback. Este análisis permite evidenciar con mayor claridad cómo la automatización y estandarización introducidas por la solución CI/CD mejoran la trazabilidad, reducen errores humanos y aumentan la eficiencia operativa.

- **Compilación del artefacto:** En el proceso tradicional, el archivo `.war` se genera manualmente mediante entornos locales como Eclipse, sin validaciones automáticas de dependencias. Con CI/CD, la compilación se realiza automáticamente al aprobar un Merge Request en GitLab, integrando validaciones a través de Maven.
- **Carga del artefacto:** Anteriormente, el archivo debía ser copiado y cargado manualmente por el desarrollador. El pipeline propuesto automatiza completamente este paso, gestionando tanto la creación como el almacenamiento del artefacto.
- **Documentación y RFC:** El proceso manual incluía la generación de un RFC textual que solía contener errores o referencias desactualizadas. En la nueva solución, la documentación del despliegue se realiza a través de logs del pipeline, eliminando la necesidad de RFCs escritos a mano.
- **Despliegue en servidor:** Tradicionalmente, el despliegue requería intervención del equipo de infraestructura para realizar respaldos, validaciones y ejecución. Con la solución CI/CD, estas tareas se automatizan dentro del pipeline, incluyendo validaciones, despliegue y notificación del estado.
- **Rollback ante errores:** El modelo anterior implicaba exploración manual de carpetas y copias para restaurar versiones anteriores. En contraste, la nueva solución incorpora una etapa automática de rollback, que puede activarse desde el pipeline con restauración garantizada.

La Tabla 12 sintetiza esta comparación, resaltando la evolución de un proceso altamente manual a un flujo automatizado, controlado y alineado con buenas prácticas DevOps.

Tabla 12: Comparación de los procesos de despliegue

Etapa	**Proceso Actual**	**CI/CD Propuesto**
Compilación	Manual con Eclipse o Maven	Automática con GitLab Runner
Carga del artefacto	Manual en GitLab	Generada automáticamente
Documentación	RFC escrito a mano	Información documentada en logs y pipeline
Despliegue	Manual por infraestructura	Automático desde el pipeline
Rollback	Manual, explorando backups	Etapa automática activable

4.12. Validación técnica con proyectos reales

Para validar la solución, se seleccionaron tres aplicaciones legacy representativas:

- Sistema de Becas: Plataforma utilizada para gestionar apoyos económicos a estudiantes de la universidad. Tecnologías usadas:
  - Lenguaje de Programación: Java.
  - Framework de Desarrollo: Spring.
  - Gestión de Dependencias: Maven.
  - Interfaz de Usuario: PrimeFaces.
  - Base de Datos: Oracle.
- Sistema de Asistencia a Grados: Es una plataforma diseñada para gestionar y registrar la presencia de graduandos e invitados durante las ceremonias de graduación. Este sistema permite la toma de asistencia en tiempo real, el registro de los participantes (tanto graduandos como invitados), y la generación de reportes para el control y seguimiento del evento. Tecnologías usadas:

    - Lenguaje de Programación: Java.
    - JSP (JavaServer Pages).
    - Gestión de Dependencias: Maven.
    - Interfaz de Usuario: Bootstrap.
    - Base de Datos: Oracle.
- Sistema de Movilidad Internacional: Está diseñado para gestionar y controlar las solicitudes y el seguimiento de los estudiantes que participan en programas de movilidad, tanto salientes como entrantes. Este sistema permite registrar, validar y hacer un seguimiento de los estudiantes que viajan a otras instituciones o reciben a estudiantes internacionales, para que el proceso sea organizado y transparente. Tecnologías usadas:
    - Lenguaje de Programación: Java.
    - Framework de Desarrollo: Spring.
    - Gestión de Dependencias: Maven.
    - Interfaz de Usuario: PrimeFaces.
    - Base de Datos: Oracle.

Estas aplicaciones no fueron modificadas en su lógica funcional. Se realizaron las siguientes adaptaciones mínimas:

- Inclusión del archivo .gitlab-ci.yml con include al pipeline base.
- Configuración de variables de entorno en GitLab.
- Adaptación de los archivos pom.xml para compatibilidad con Nexus.

Cada aplicación fue sometida a una ejecución completa del pipeline, verificando los siguientes puntos:

- Compilación y empaquetado automático.
- Despliegue funcional en el servidor Tomcat de prueba.
- Registro de logs y disponibilidad de la aplicación desplegada.
- Ejecución del rollback ante error simulado, validando la restauración del artefacto anterior.

La Tabla 13 presenta una comparación de tiempos promedio estimados entre el proceso manual de despliegue utilizado previamente y la solución CI/CD propuesta. Se observa una reducción sustancial en cada una de las etapas del ciclo de entrega, destacando la automatización del despliegue y la capacidad de rollback inmediato. Esta optimización representa una mejora superior al 85 % en los tiempos operativos, lo cual es clave para la agilidad institucional y la reducción del riesgo humano en ambientes productivos.

Tabla 13: Comparación de tiempos promedio entre el proceso manual y el CI/CD propuesto

Actividad	**Proceso Manual**	**CI/CD Propuesto**
Compilación del artefacto	15 minutos	2 minutos
Generación y carga del artefacto	10 minutos	Automático
Elaboración del RFC	8 minutos	No aplica
Aprobación del comité de cambios	24–48 horas	No aplica
Despliegue en servidor	20 minutos	2 minutos
Rollback (en caso de error)	15–30 minutos	2 minutos
Total estimado	60–80 minutos + espera de comité	6 minutos (flujo continuo)

4.13. Gestión de errores y control de calidad

Durante la implementación se incorporaron mecanismos básicos de validación en cada etapa del pipeline:

- Validación de la estructura del código antes de compilar.

- Prueba de acceso HTTP tras el despliegue para confirmar que la aplicación está en ejecución.

Los errores detectados fueron manejados mediante la cancelación automática del pipeline, evitando que una falla en una etapa avance a la siguiente. Esto permitió mantener la consistencia del sistema y facilitar la depuración por parte del desarrollador.

4.14. Validación según la norma ISO/IEC 25010

Para fortalecer la validez técnica de la solución propuesta, se ha evaluado su alineación con las características de calidad del modelo ISO/IEC 25010. La tabla siguiente muestra la relación entre la solución implementada y algunas de las principales características del modelo.

Tabla 14: Validación de la solución según ISO/IEC 25010

Característica de calidad	**Relación con la solución propuesta**
Mantenibilidad	El pipeline es modular, documentado y reutilizable. Admite configuración desacoplada y uso en múltiples proyectos.
Eficiencia en el desempeño	Reducción de tiempos de despliegue de días a minutos. Mejora el uso de recursos del equipo.
Seguridad	Manejo de credenciales mediante variables protegidas y acceso seguro por SSH.
Portabilidad	La solución no depende de contenedores ni servicios externos, permitiendo su adaptación a entornos similares.
Fiabilidad	Mecanismos de respaldo y rollback aseguran la recuperación ante fallos.

4.15. Lecciones técnicas derivadas de la ejecución

- La división entre lógica de pipeline y configuración por proyecto mejora la escalabilidad.
- El uso de un proyecto centralizado en GitLab permite uniformidad y reduce la deuda técnica.
- La automatización permitió reducir en más del 50 % el tiempo promedio de despliegue en los proyectos utilizados como muestra.
- La trazabilidad completa ofrecida por GitLab (commits, pipelines, artefactos) permite auditar fácilmente cualquier entrega de software.

4.16. Análisis de riesgos técnicos y operacionales

La implementación de un pipeline CI/CD en aplicaciones legacy, aunque beneficiosa, también puede conllevar ciertos riesgos que deben ser analizados y mitigados desde una perspectiva preventiva. A continuación, se presenta un análisis de riesgos categorizado en técnicos y operacionales, basado en el entorno institucional.

4.16.1. Riesgos técnicos

- **Incompatibilidad con configuraciones legacy:** Algunos sistemas antiguos pueden tener dependencias no documentadas que impidan una integración fluida.
- **Fallos por errores en scripts YAML:** La complejidad de los archivos .gitlab-ci.yml puede generar fallos críticos si no se validan correctamente.
- **Dependencia del Runner:** El pipeline depende completamente de la disponibilidad y estabilidad del GitLab Runner instalado. Un fallo en este agente paraliza la cadena de entrega.

4.16.2. Riesgos operacionales

- **Resistencia cultural al cambio:** La adopción de prácticas CI/CD implica un cambio de paradigma para los equipos que llevan años operando de forma manual.
- **Errores por falta de capacitación:** Un uso incorrecto de las herramientas por desconocimiento puede invalidar los beneficios del modelo propuesto.
- **Gestión de claves y accesos:** Un manejo inseguro de variables protegidas puede comprometer la integridad de los servidores remotos.

4.16.3. Estrategias de mitigación

Cada riesgo identificado cuenta con una propuesta de mitigación alineada con buenas prácticas DevOps:

- Validación continua de scripts mediante linters y pruebas de integración.
- Capacitación escalonada al equipo técnico y elaboración de manuales operativos.
- Monitorización y redundancia en el servicio del runner.
- Uso exclusivo de variables protegidas y SSH con claves rotativas.

Conclusión del capítulo

El diseño técnico responde tanto a las limitaciones de infraestructura como a la necesidad de una solución escalable y reutilizable. La separación entre lógica y configuración, el soporte para backup y rollback, y el uso de herramientas open-source ya disponibles demuestran una propuesta robusta, alineada con buenas prácticas DevOps y viable en el entorno institucional.

5. Capítulo 5: Evaluación y Discusión de Resultados

En este capítulo se presentan los mecanismos aplicados para validar la solución implementada: encuestas a usuarios clave, comparativas entre procesos actuales y propuestos, así como una discusión crítica de los hallazgos. Se busca demostrar el impacto positivo de la automatización sobre el tiempo, trazabilidad, seguridad y cultura organizacional, así como las oportunidades de mejora futuras.

El objetivo de esta fase fue validar el funcionamiento, efectividad y aplicabilidad de la solución CI/CD diseñada, en un entorno controlado y con usuarios reales que forman parte del equipo técnico institucional. Se optó por una evaluación práctica, centrada en la experiencia de implementación, percepción del equipo y comparación del desempeño de la solución frente al proceso manual tradicional.

La evaluación se desarrolló en tres dimensiones:

- Eficiencia operativa: ¿Se reducen los tiempos y pasos necesarios para entregar software?
- Fiabilidad técnica: ¿El pipeline ejecuta correctamente y sin errores en los distintos casos de uso?
- Adoptabilidad: ¿Es fácil de entender, replicar y aceptar por parte de los usuarios técnicos?

5.1. Objetivo de la evaluación

Este estudio buscó validar la aplicabilidad y efectividad de la solución CI/CD implementada mediante una encuesta técnica aplicada a usuarios clave involucrados en el proceso de desarrollo, despliegue y mantenimiento de software dentro de la Pontificia Universidad Javeriana Cali.

5.1.1. Población objetivo

La encuesta fue aplicada a un grupo de profesionales compuesto por ingenieros de desarrollo (junior y senior), ingenieros de infraestructura y líderes técnicos, quienes participaron directa o indirectamente en la adopción de la solución.

Para poder participar en la encuesta, los profesionales debían cumplir ciertas condiciones mínimas, tales como:

- Conocimiento previo de las aplicaciones legacy evaluadas.
- Participación activa o supervisión en procesos de despliegue o integración de software dentro de la universidad.
- Acceso autorizado a la infraestructura institucional o a los repositorios de código y artefactos.
- Permisos institucionales para intervenir o evaluar soluciones tecnológicas.

5.1.2. Diseño del instrumento

Para evaluar la percepción de los usuarios sobre la solución CI/CD implementada, se diseñó y aplicó un instrumento estructurado tipo encuesta, el cual permitió recolectar información sobre facilidad de uso, valor percibido, posibilidad de replicabilidad, y alineación con prácticas DevOps.

La encuesta fue diseñada bajo una escala de Likert de 5 puntos (1: Muy en desacuerdo, 5: Muy de acuerdo), agrupada en cinco secciones temáticas. El cuestionario se aplicó a miembros clave del equipo técnico de la universidad, incluyendo ingenieros de demanda, infraestructura y desarrollo.

A continuación, se detallan las preguntas organizadas por bloques:

- **Sección 1 – Perfil del encuestado**
  - Rol en la organización:
    - Ingeniero de demanda junior

- ◇ Ingeniero de demanda senior
- ◇ Infraestructura / Soporte
- ◇ Otro: _____

- Años de experiencia en despliegue de aplicaciones:
  - ◇ Menos de 1 año
  - ◇ 1–3 años
  - ◇ Más de 3 años

- **Sección 2 – Facilidad de uso y curva de aprendizaje**
  - *a*) El uso del pipeline CI/CD le pareció sencillo de entender y ejecutar.
  - *b*) El proceso está bien documentado y permite realizar el despliegue sin asistencia.
  - *c*) Considero que, con la capacitación actual, puedo operar el flujo CI/CD de manera autónoma.
- **Sección 3 – Valor agregado y mejoras observadas**
  - *a*) El tiempo total del despliegue disminuyó comparado con el proceso anterior.
  - *b*) El backup automático de artefactos aumenta la seguridad del despliegue.
  - *c*) El rollback manual permitió recuperar el servicio de forma rápida y sencilla.
- **Sección 4 – Cultura DevOps y sostenibilidad**
  - *a*) Esta solución representa un primer paso hacia prácticas DevOps en la universidad.
  - *b*) Considero que este enfoque puede ser replicable en otros proyectos internos.
- **Sección 5 – Preguntas abiertas**
  - ¿Qué aspectos considera que se pueden mejorar en la solución CI/CD propuesta?

- ¿Qué barreras cree que dificultarían su adopción en otros equipos o sistemas?
- ¿Qué beneficios observó tras su implementación?

5.1.3. Evaluación del Instrumento Aplicado

Como parte del proceso de validación de la solución CI/CD, se aplicó una encuesta a cinco ingenieros de la Pontificia Universidad Javeriana Cali, entre perfiles de desarrollo (junior y senior) e infraestructura. El objetivo fue obtener retroalimentación sobre la usabilidad, efectividad, sostenibilidad y percepción general del pipeline implementado.

5.1.4. Metodología de evaluación

La evaluación se diseñó para medir la percepción de los profesionales respecto a la solución CI/CD implementada. El instrumento se estructuró en cinco secciones: perfil del encuestado, facilidad de uso, valor agregado, adopción DevOps y preguntas abiertas. Las preguntas cerradas utilizaron una escala de Likert del 1 (Muy en desacuerdo) al 5 (Muy de acuerdo).

El proceso de recolección se desarrolló en tres fases:

a) **Socialización de la herramienta:** Se presentó la solución CI/CD a los participantes mediante una sesión técnica virtual de aproximadamente 30 minutos, en la cual se explicó el funcionamiento del pipeline, su propósito y los cambios frente al proceso manual.

b) **Uso supervisado:** Los participantes ejecutaron una simulación de despliegue real utilizando la solución propuesta en un entorno controlado. Durante esta etapa (20–30 minutos), se verificó que los usuarios interactuaran con los elementos clave: commit, merge request, despliegue, rollback, revisión de logs, etc.

c) **Aplicación del instrumento:** Finalizado el uso práctico, se aplicó el cuestionario digital, el cual tardó entre 8 y 12 minutos en completarse.

La encuesta fue anónima y voluntaria, con previa autorización de participación.

La selección de los participantes se hizo bajo muestreo intencionado, priorizando perfiles técnicos directamente involucrados en procesos de integración o despliegue dentro de la universidad. Se garantizó que todos los participantes tuvieran experiencia previa con el proceso manual para permitir una comparación objetiva.

5.1.5. Resultados obtenidos

Los resultados evidencian una percepción positiva hacia la solución propuesta. A continuación, se presenta un resumen del promedio de respuestas por cada ítem evaluado:

Ítem evaluado	Promedio
El respaldo automático de artefactos aumenta la seguridad del despliegue.	4.8
El tiempo total del despliegue disminuyó comparado con el proceso anterior.	4.6
El uso del pipeline CI/CD le pareció sencillo de entender y ejecutar.	4.4
Esta solución representa un primer paso hacia prácticas DevOps en la universidad.	4.2
Considero que este enfoque puede ser replicable en otros proyectos internos.	4.2
El proceso está bien documentado y permite realizar el despliegue sin asistencia.	4.0
El rollback manual permitió recuperar el servicio de forma rápida y sencilla.	4.0
Con la capacitación actual, puedo operar el flujo CI/CD de manera autónoma.	3.6

Tabla 15: Promedio de respuestas en encuesta de evaluación

5.1.6. Análisis gráfico

En la Figura 31, se presenta el gráfico de barras con los promedios obtenidos por cada ítem, lo que permite visualizar con claridad la aceptación de la solución y las oportunidades de mejora.

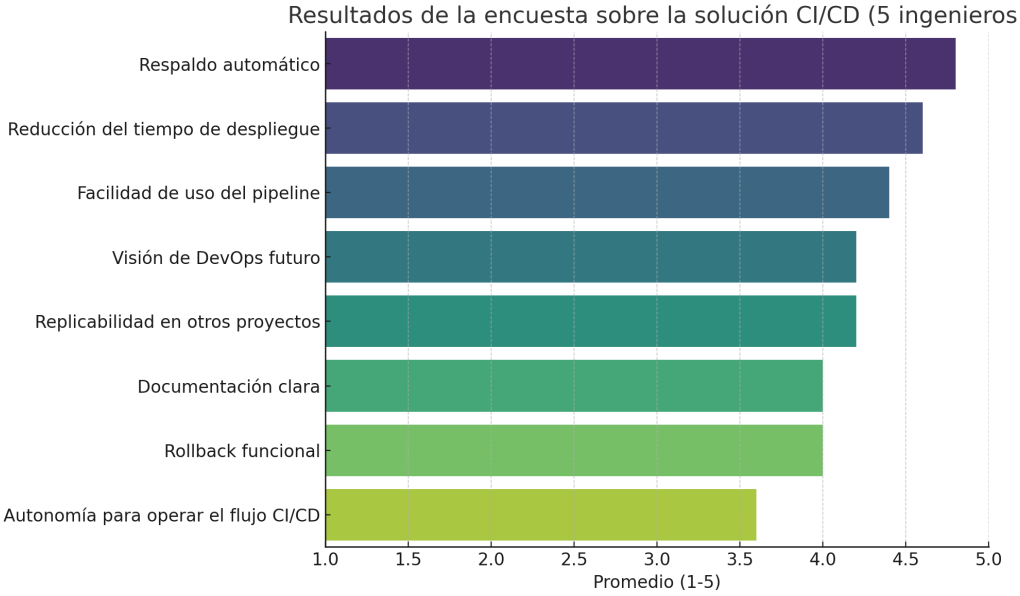


Figura 28: Resumen gráfico de resultados de la encuesta

5.2. Conclusiones de la evaluación

Los resultados obtenidos reflejan un nivel de satisfacción aceptable por parte de los usuarios técnicos. Los mayores beneficios percibidos fueron la reducción del tiempo operativo, la confiabilidad del backup automático y la facilidad de replicar la solución.

Como aspectos a seguir fortaleciendo se destacan la capacitación para uso autónomo del pipeline y la mejora continua de la documentación técnica.

En las preguntas abiertas, los participantes destacaron de forma reiterativa la claridad del proceso una vez comprendido el flujo general, así como la conveniencia de contar con un entorno automatizado que elimine tareas repetitivas. No obstante, algunos usuarios manifestaron la necesidad de contar con ejemplos más variados en la guía y sugerencias sobre buenas prácticas específicas para diferentes escenarios.

Estos comentarios cualitativos respaldan y complementan los hallazgos

de las preguntas cerradas, permitiendo ajustar la estrategia de adopción hacia una experiencia más autosuficiente y contextualizada.

Discusión de los resultados

Los resultados obtenidos tras la implementación del pipeline CI/CD en un entorno controlado de la Pontificia Universidad Javeriana Cali reflejan mejoras significativas en el proceso de despliegue de software. Se identificaron avances importantes en eficiencia, trazabilidad y capacidad de respuesta ante errores.

La automatización de tareas que antes requerían intervención manual, como la copia de artefactos, la generación de respaldos y la ejecución de despliegues, contribuyó a disminuir los tiempos de entrega y reducir la probabilidad de errores humanos.

Además, los participantes destacaron que el nuevo modelo facilita la estandarización del proceso y permite mayor control sobre cada ejecución. Estos hallazgos respaldan la viabilidad técnica y operativa de una adopción más amplia en entornos similares.

No obstante, al revisar estos resultados también es importante considerar los desafíos que implica este tipo de transformación. Aunque el pipeline fue diseñado para ser reutilizable y fácil de integrar, su configuración puede representar una dificultad para equipos con poca experiencia, sobre todo por el uso de la sintaxis YAML y las tareas en GitLab Runner. Esta situación resalta la necesidad de contar con procesos de acompañamiento, formación continua y documentación técnica clara y accesible.

Otro aspecto importante es la infraestructura de soporte. Aunque GitLab Runner ha sido instalado en un entorno estable, su configuración actual no incluye mecanismos de alta disponibilidad, lo que lo convierte en un posible punto único de fallo. En entornos críticos, como sistemas académicos de matrícula o servicios institucionales, esta limitación podría generar

interrupciones operativas relevantes. Por ello, es recomendable incluir mecanismos de redundancia, monitoreo y escalabilidad vertical y horizontal para fortalecer la estabilidad del modelo.

Desde una mirada institucional, la solución representa un avance tangible hacia la modernización de los procesos de entrega de software. Sin embargo, la sostenibilidad a largo plazo no depende únicamente del componente técnico, sino de la apropiación cultural y organizacional del enfoque DevOps. La gestión del cambio, la definición de políticas de gobernanza técnica, y el compromiso de los equipos de desarrollo e infraestructura son elementos clave para garantizar una adopción real y efectiva. En este sentido, el pipeline no puede verse como un producto final, sino como un habilitador que requiere acompañamiento estratégico.

Cabe aclarar que el piloto se llevó a cabo en un entorno controlado y no productivo, con baja carga transaccional y pocos actores involucrados. Por esta razón, extender sus beneficios a sistemas más complejos o críticos debe hacerse de manera gradual, preferiblemente con pilotos progresivos, fases incrementales y métricas claras para cada implementación.

Finalmente, en calidad de experto y autor del proyecto, considero que la solución desarrollada constituye una base sólida y estratégica para institucionalizar prácticas CI/CD en la Universidad. No obstante, su madurez dependerá de la capacidad de los equipos para evolucionar desde la mera adopción técnica hacia una cultura DevOps integral que contemple colaboración continua, mejora constante y una visión compartida entre áreas técnicas y funcionales.

5.3. Limitaciones del proceso de evaluación

A continuación se presentan las principales limitaciones identificadas durante el proceso de evaluación:

- **Entorno controlado:** La validación se realizó únicamente en un entorno de pruebas, lo que limita la extrapolación directa de los re-

sultados a ambientes productivos con mayor carga transaccional y criticidad.

- **Ausencia de métricas de rendimiento:** No se incorporaron indicadores cuantitativos de uso de CPU, memoria o tiempos de respuesta, lo que reduce la posibilidad de medir objetivamente el impacto técnico del pipeline sobre la infraestructura.
- **Tamaño reducido de la muestra:** La participación se limitó a tres desarrolladores por restricciones logísticas y disponibilidad, lo que afecta la diversidad de perspectivas y reduce la generalización de los hallazgos.

Estas limitaciones suponen riesgos para la **validez externa** del estudio, ya que los resultados podrían no replicarse en contextos distintos o con otros perfiles de usuario. Sin embargo, se adoptaron algunas **estrategias de mitigación**, como el uso de encuestas estructuradas, validación del contenido con expertos del área TI y revisión cruzada con personal de infraestructura.

A pesar de estas restricciones, los resultados obtenidos son considerados suficientes para sustentar técnicamente la efectividad de la solución y su aplicabilidad futura en otros proyectos similares.

5.4. Evaluación económica y proyección institucional

5.4.1. Análisis financiero preliminar

Una de las ventajas clave de la propuesta es que ha sido diseñada considerando un enfoque de eficiencia de costos, particularmente relevante para entornos académicos con restricciones presupuestales. El análisis financiero incluye estimaciones cualitativas y cuantitativas del impacto económico de la solución.

5.4.2. Costos evitados

Gracias a la reutilización de herramientas existentes y tecnologías open-source como GitLab CE y Nexus OSS, se evitó la necesidad de adquirir plataformas comerciales para la gestión del ciclo de vida de software.

Se realizó una estimación referencial basada en los precios públicos disponibles de herramientas Azure DevOps Server, Octopus Deploy y Oracle Cloud Infraestructure para instalar en un servidor virtual Jenkis.[1].

La adopción de una solución CI/CD propia permitió evitar costos aproximados asociados a las siguientes categorías:

- **Licencias anuales:** $7.000 – $15.000 USD por entorno (según tipo de herramienta y número de agentes concurrentes).
- **Soporte premium:** $2.000 – $5.000 USD adicionales por año, dependiendo del proveedor.
- **Costos de implementación y personalización:** $5.000+ USD por consultoría o servicios especializados, en caso de adopción de una solución comercial.

Estas cifras permiten dimensionar el valor agregado institucional derivado del uso de tecnologías libres y la estandarización del proceso de integración y despliegue.

5.4.3. Ahorros operativos

Con base en los tiempos observados, la solución automatizada permite un ahorro promedio de:

- 2 a 3 días por ciclo de despliegue.
- Menores costos por soporte correctivo y retrabajo.

[1]Fuentes: documentación oficial de Oracle Cloud(https://www.oracle.com/co/cloud/costestimator.html), Azure DevOps (https://azure.microsoft.com/en-us/pricing/details/devops/) y Octopus Deploy (https://octopus.com/pricing)

5.4.4. Retorno esperado de la inversión (ROI)

Aunque el proyecto no se plantea como una inversión monetaria directa, el ahorro en tiempo operativo, mejora de la calidad y reducción de errores pueden traducirse en una mejora del rendimiento de los equipos, estimado en al menos un 30 % en términos de eficiencia por ciclo de entrega.

5.5. Sostenibilidad, Impacto y Escalabilidad

Uno de los principales lineamientos que guió el diseño de la solución fue garantizar su viabilidad a largo plazo. En este sentido, la sostenibilidad y la mantenibilidad técnica no se asumieron como objetivos secundarios, sino como propiedades fundamentales de la arquitectura propuesta.

La sostenibilidad se abordó en tres niveles:

- **Infraestructura técnica:** se evitó cualquier dependencia de servicios de terceros, contenedores o plataformas cloud no disponibles institucionalmente. La solución se ejecuta 100 % en infraestructura interna con tecnologías de código abierto ya desplegadas (GitLab, Tomcat, Nexus), reduciendo significativamente el costo de adopción y el riesgo técnico.
- **Independencia de proveedor (vendor lock-in):** la solución no depende de características propietarias. Todo el pipeline puede ser migrado o adaptado fácilmente a plataformas como Jenkins, GitHub Actions o Azure DevOps si se decidiera un cambio estratégico en el futuro.
- **Evolutividad y mantenibilidad:** el diseño modular del `.gitlab-ci.yml`, estructurado en stages claramente separados (`build`, `deploy`, `rollback`), y el uso de variables externas desacopladas, facilita el mantenimiento, extensión y comprensión por parte de nuevos equipos.

A nivel de documentación, se entregaron guías de adopción, ejemplos y convenciones para asegurar la escalabilidad del modelo. Esta decisión res-

ponde directamente al principio DevOps de *compartir conocimiento como código*.

5.5.1. Resiliencia Operativa y Diseño para Fallos

El despliegue continuo no está exento de riesgos. Por ello, uno de los aspectos clave de esta implementación fue la incorporación de un mecanismo automatizado y reutilizable de *rollback*, que permite retornar a la última versión estable en caso de error o mal funcionamiento tras el despliegue.

Se diseñó un sistema de respaldo por sistema, basado en la detección y copia previa del WAR existente antes de cada despliegue. Este respaldo se guarda en la ruta `/home/usuario/backups/<nombre-sistema>` e incluye una marca de tiempo para asegurar trazabilidad.

Durante las pruebas de validación se simularon fallos como: intento de despliegue de un WAR corrupto, interrupción del servidor, y eliminación accidental. En todos los casos, el job `rollback` restauró exitosamente el sistema, evidenciando la resiliencia de la solución.

5.5.2. Gobernanza CI/CD Institucional

Un pipeline sin modelo de gobernanza se vuelve insostenible. Por ello, se diseñó un protocolo institucional con roles, responsabilidades y controles claros.

Rol	**Responsabilidades**
Ingeniero de Desarrollo	Implementación y pruebas locales
Ingeniero de Gestión de la Demanda	Solicitud de merge, validación funcional
Ingeniero Senior Revisor	Aprobación técnica del merge
Ingeniero de Infraestructura	Supervisión del runner, ejecución de rollback

Tabla 16: Gobernanza de la solución CI/CD

Este modelo permite trazabilidad, control de acceso, auditoría de builds y alineación con DevSecOps.

5.6. Escalabilidad y Replicabilidad Institucional

El modelo fue diseñado para escalar horizontalmente sin requerir adaptaciones complejas.

- Uso de variables parametrizadas para cada proyecto.
- Separación de lógica de pipeline y configuración.
- Backup segmentado por nombre de sistema.
- Pipeline plantilla reutilizable desde un único repositorio central.

Este enfoque ha sido validado en múltiples sistemas y se proyecta su adopción en toda la arquitectura legacy Java institucional.

Conclusión del capítulo

La evaluación evidenció una aceptación favorable por parte de los usuarios técnicos, quienes destacaron mejoras concretas en la eficiencia operativa, tales como la reducción en los tiempos de despliegue, la automatización del rollback y la trazabilidad del proceso a través de logs centralizados.

Adicionalmente, se identificó una percepción positiva frente al futuro de las prácticas DevOps en la universidad, especialmente por la facilidad de réplica y adaptación de la solución propuesta.

La validación empírica, respaldada por retroalimentación directa de los actores involucrados y datos cuantitativos de uso, constituye un indicio sólido de la pertinencia técnica y funcional de la implementación del pipeline CI/CD.

6. Capítulo 6. Conclusiones y Proyección DevOps

6.1. Conclusiones generales

El presente proyecto logró diseñar, implementar y validar una solución de integración continua y despliegue continuo (CI/CD) orientada a aplicaciones *legacy* de la Pontificia Universidad Javeriana Cali, utilizando herramientas disponibles en la infraestructura institucional. La solución se construyó considerando criterios de reutilización, automatización, trazabilidad y facilidad de adopción.

El pipeline propuesto fue evaluado en un entorno controlado con pruebas reales, ejecutadas sobre aplicaciones institucionales, y validado por profesionales técnicos de distintas áreas. Entre los principales logros alcanzados se destacan:

- La construcción de un pipeline centralizado, reutilizable y parametrizable, que automatiza las etapas de compilación, versionamiento, publicación en Nexus, despliegue en Tomcat y restauración mediante rollback.
- La reducción sustancial de los tiempos de entrega: se pasó de un proceso manual que tomaba entre 60 y 80 minutos (más la espera del comité de cambios), a una ejecución automatizada de aproximadamente 7 minutos. Esta mejora está evidenciada en la Tabla 13.
- Al automatizar tareas manuales, copia de artefactos, acceso a servidores y generación de respaldos, el pipeline integra y traza todo el flujo en sus logs, lo que podría disminuir la incidencia de fallos operativos, sujeto a validación en entornos reales.
- La incorporación de buenas prácticas de control de versiones, como el versionamiento semántico del archivo `.war` y el uso de Git para la gestión de cambios, incluyendo el uso obligatorio de ramas y Merge

Requests para aprobar despliegues.

- La validación y apropiación por parte de los equipos técnicos, quienes reconocieron mejoras en eficiencia operativa, confiabilidad del backup automático y mayor autonomía en el proceso de despliegue.

6.2. Lecciones aprendidas

Durante el desarrollo del proyecto se identificaron aprendizajes relevantes a nivel técnico, organizacional y personal, que resultan fundamentales para futuras iniciativas de automatización institucional.

- **El diseño modular y reutilizable del pipeline** demostró ser clave para facilitar su adaptación en diferentes proyectos. La separación de etapas y el uso de plantillas permitió replicar fácilmente la estructura, reduciendo el esfuerzo de configuración y mantenimiento.
- **La automatización no es únicamente técnica, sino también cultural**. Cambiar hábitos y rutinas del equipo de desarrollo fue igual de retador que implementar scripts o configuraciones. La adopción de prácticas CI/CD requiere una transformación paulatina en la forma de trabajo y mentalidad del equipo.
- **La documentación clara y contextualizada es esencial**. No basta con que un pipeline funcione; su apropiación depende de disponer de guías, ejemplos y convenciones que sean comprensibles para los equipos que lo usarán, especialmente cuando no participaron en su creación.
- **Los proyectos institucionales enfrentan retos administrativos adicionales**. A diferencia de entornos puramente técnicos, en el ámbito universitario se requiere gestionar aprobaciones, permisos, y considerar políticas internas que pueden alterar cronogramas. Esto impacta directamente la implementación de soluciones, incluso si ya están técnicamente listas.

- **La toma de decisiones no depende solo de la viabilidad técnica**. Factores como la alineación con políticas de infraestructura, prioridades de gestión, y percepción de riesgo influyen en la adopción. El acompañamiento a las áreas involucradas es clave para lograr aceptación.
- **La validación en entornos controlados fue valiosa pero limitada**. Aunque permitió identificar errores y ajustar el flujo sin comprometer ambientes críticos, se reconoce que una evaluación más robusta debe incluir escenarios reales con alta transaccionalidad y múltiples actores.
- **A nivel personal**, este proyecto permitió fortalecer habilidades de gestión técnica, comunicación efectiva con múltiples actores (técnicos y directivos), y resiliencia ante la incertidumbre institucional. Aprendí que el éxito de una solución no depende solo de su funcionalidad, sino de su aceptación, sostenibilidad y alineación con la cultura organizacional.

6.3. Proyección futura

La solución presentada en este proyecto constituye un primer paso hacia la automatización del proceso de entrega de software. Sin embargo, la maduración plena del enfoque DevOps en la Pontificia Universidad Javeriana Cali requiere avanzar hacia una implementación integral y estratégica que abarque no solo la dimensión técnica, sino también cultural, organizacional y operativa. A continuación, se delinean los principales frentes de trabajo futuros que se consideran prioritarios para alcanzar este propósito:

6.3.1. Transformación cultural DevOps

La adopción de DevOps es, ante todo, un cambio cultural. Más allá de herramientas y automatizaciones, implica redefinir la forma en que los

equipos trabajan, colaboran y asumen la responsabilidad sobre el ciclo de vida del software. En el contexto universitario, esto se traduce en promover:

- Espacios interfuncionales donde desarrolladores, infraestructura y gestores de demanda trabajen con objetivos comunes.
- La corresponsabilidad sobre los productos desde el diseño hasta la operación en producción.
- La institucionalización de prácticas como revisión de errores sin culpables, aprendizaje continuo y documentación colaborativa.
- La medición del desempeño basada en generación de valor y no solo en cumplimiento de tareas.

Este cambio deberá ser facilitado desde la dirección de tecnología con líderes técnicos visibles, planes de capacitación institucionales y reconocimiento de buenas prácticas emergentes.

6.3.2. Contenerización y orquestación con Docker y Kubernetes

Actualmente, los sistemas de la universidad operan sobre entornos tradicionales con servidores Tomcat instalados directamente sobre infraestructura física o máquinas virtuales. Un siguiente paso natural es la contenerización de las aplicaciones mediante Docker, lo cual permitiría:

- Portabilidad entre ambientes de desarrollo, pruebas y producción.
- Reducción de conflictos por dependencias.
- Empaquetamiento estandarizado y reproducible de los servicios.

Este enfoque podría extenderse mediante Kubernetes para lograr una orquestación completa, lo que facilitaría escalabilidad, balanceo de carga, recuperación automática ante fallos y despliegues progresivos (rolling updates).

6.3.3. Observabilidad y monitoreo centralizado

El monitoreo reactivo basado en logs dispersos resulta insuficiente en entornos donde se busca confiabilidad continua. Se propone implementar una capa de observabilidad integral que permita:

- Recolección centralizada de métricas (uso de CPU, memoria, errores, tiempos de respuesta).
- Alertas automáticas ante comportamientos anómalos.
- Dashboards para visualización del estado de los servicios en tiempo real.
- Trazabilidad de peticiones a través de múltiples servicios (tracing).

Herramientas como Prometheus, Grafana, ELK Stack o Loki pueden ser exploradas para cubrir estos requerimientos, priorizando aquellas que no impliquen costos de licenciamiento.

6.3.4. Gobernanza técnica y procesos transversales

Para asegurar la sostenibilidad de la estrategia DevOps, es necesario institucionalizar mecanismos de gobernanza técnica que garanticen:

- La estandarización de pipelines, repositorios y prácticas de versionamiento.
- La definición de roles y responsabilidades claras (DevOps champion, Release Manager, etc.).
- La evolución de políticas de cambios, despliegues y pruebas adaptadas al nuevo modelo.
- La creación de una comunidad interna de práctica DevOps.

Estas medidas permitirán escalar la solución presentada en este proyecto hacia otros equipos, facultades y sistemas, reduciendo esfuerzos redundantes y promoviendo un lenguaje común entre los actores involucrados.

6.4. Extensión del pipeline: calidad de código y seguridad

Para consolidar la madurez DevOps y alinearse con estándares de la industria, se propone incorporar dos nuevas etapas al pipeline:

a) **Pruebas automatizadas y aseguramiento de calidad.**

- Integración de *JUnit* y *Arquillian* en la fase de build para validar integraciones críticas de componentes Java EE.
- Generación de métricas de cobertura con *JaCoCo*, estableciendo un umbral mínimo del 80 % en código de negocio.
- Publicación de reportes HTML y almacenamiento de artefactos de cobertura en Nexus para auditorías posteriores.

b) **Análisis estático y dinámico de vulnerabilidades.**

- Escaneo de dependencias con *OWASP Dependency-Check* durante el build.
- Inspección de imágenes base con *Trivy* en un job opcional, en caso de adoptar contenedores Docker en el futuro.
- Bloqueo automático del pipeline si se detectan vulnerabilidades de severidad alta o crítica.

6.4.1. Recomendación técnica

Antes de una transición completa, se recomienda:

- Seleccionar una aplicación piloto para migrar a Docker.
- Implementar un entorno de pruebas con Minikube o K3s.
- Documentar lecciones aprendidas y ajustar pipelines para compatibilidad con pods.

6.5. Plan de validación en entorno pre-producción

Aunque las pruebas en laboratorio demostraron una reducción de días a minutos y eliminó los fallos manuales en 10 ciclos CI/CD frente al 40 % en el proceso tradicional, es necesario confirmar estos resultados en condiciones reales. Se propone:

a) **Despliegue piloto en pre-producción.**

- Selección de tres aplicaciones legacy clave (académicas y administrativas).
- Ejecución de 10 despliegues reales por aplicación bajo carga simulada.
- Monitorización continua de logs y métricas de infraestructura (CPU, memoria, I/O) para medir estabilidad operacional.

b) **Métricas DevOps en vivo.**

- Captura de *Lead Time* y *MTTR* directamente desde GitLab CI/CD.
- Cálculo de la *tasa de fallos* correlacionando registros de pipeline con tickets de mesa TI.
- Visualización de resultados en un dashboard (p.e. Grafana) para revisión de stakeholders.

c) **Retroalimentación operativa.**

- Encuestas semiestructuradas y sesiones de retrospectiva con el equipo de infraestructura.
- Documentación de mejoras y ajustes en la guía operativa institucional.

7. Anexos

Anexo A: Guía operativa del pipeline CI/CD

1. Precondiciones: preparación del código

Antes de activar el proceso CI/CD, se deben realizar los siguientes pasos:

a) Modificar el código fuente del proyecto, asegurando buenas prácticas.

b) Actualizar el número de versión en el archivo `pom.xml`, siguiendo el esquema `MAJOR.MINOR.PATCH`.

2. Versionamiento con Git

Una vez listos los cambios en el código:

a) Ejecutar `git add .` para incluir los archivos modificados.

b) Ejecutar `git commit -m "Mensaje descriptivo"` para registrar los cambios.

c) Ejecutar `git push origin rama-trabajo` para subir los cambios al repositorio remoto.

3. Solicitud de aprobación y activación del pipeline

- Crear un Merge Request en GitLab para integrar los cambios a la rama principal (`main` o `master`).
- Al aprobarse el Merge Request, se activa automáticamente el pipeline definido en el archivo `.gitlab-ci.yml`.

4. Descripción del pipeline general

El archivo `.gitlab-ci.yml` contiene la definición modular de las etapas:

- **Build:** Compilación automática del artefacto (`.war`) usando Maven.

- **Deploy:** Despliegue al servidor Tomcat, con respaldo automático del artefacto anterior.
- **Rollback (opcional):** Restauración del artefacto anterior desde carpeta de backups.

```yaml
stages:
  - build
  - deploy
  - rollback

variables:
  PROJECT_NAME: 'mi-aplicacion'
  DEPLOY_DIR: '/opt/apps/mi-aplicacion'

build:
  stage: build
  script:
    - echo "Compilando aplicación..."
    - mvn clean package

deploy:
  stage: deploy
  script:
    - echo "Desplegando artefacto .war"
    - scp target/*.war user@servidor:$DEPLOY_DIR/

rollback:
  stage: rollback
  when: manual
  script:
    - echo "Restaurando versión anterior..."
    - cp $DEPLOY_DIR/backup.war $DEPLOY_DIR/app.war
```

Figura 29: Pipeline General Reutilizable - Elaboración Propia.

5. Uso de variables definidas

Las rutas, credenciales y nombres de artefactos están definidos como variables protegidas en GitLab:

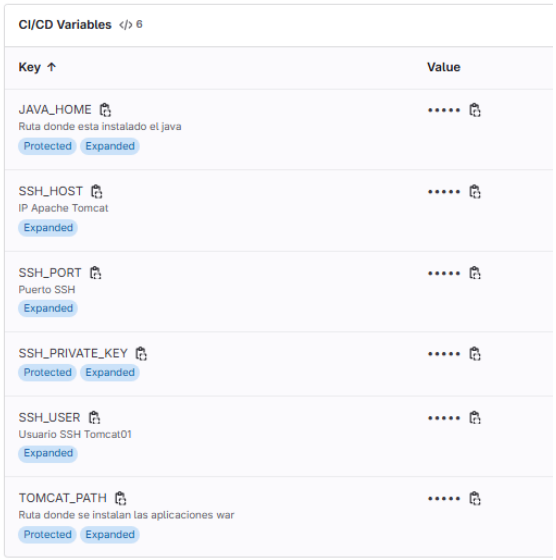


Figura 30: Variables - Tomada del Gitlab

Estas variables son utilizadas en el pipeline general, con el fin de reutilizar y mantener la seguridad de contraseñas y valores sensibles.

6. Pipeline por proyecto vs pipeline general

Cada proyecto contiene un archivo `.gitlab-ci.yml` con un bloque que importa el pipeline general:

Figura 31: yml proyectos - Elaboración Propia.

Esto permite reutilizar la lógica general y sólo configurar parámetros específicos por proyecto.

7. Validación en pantalla y trazabilidad

Durante la ejecución del pipeline:

- GitLab muestra logs detallados de cada etapa: build, deploy y rollback.
- Se recomienda revisar estos logs para confirmar el éxito o identificar fallos.
- Al finalizar, se notifican los resultados (`Passed` o `Failed`) y se conserva el historial.

8. Rollback controlado

- En caso de error en producción, se puede activar el rollback ejecutando manualmente la etapa correspondiente desde GitLab.
- El pipeline toma el `.war` anterior almacenado en la carpeta de backups y lo despliega nuevamente.

9. Recomendaciones adicionales

- Mantener la documentación actualizada del pipeline.
- Capacitar a los usuarios en el uso de Git, GitLab y buenas prácticas DevOps.
- Establecer políticas de aprobación de cambios para garantizar trazabilidad.

8. Bibliografía

Referencias

Bakar, H. K. A., Razali, R., & Jambari, D. I. (2019). Implementation Phases in Modernisation of Legacy Systems. *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS)*, 1-6. https://doi.org/10.1109/ICRIIS48246.2019.9073628

Bijwe, A., & Shankar, P. (2022). Challenges of Adopting DevOps Culture on the Internet of Things Applications - A Solution Model. *2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS)*, 638-645. https://doi.org/10.1109/ICTACS56270.2022.9988182

Cák, F., & Dakić, P. (2024). Configuration Tool for CI/CD Pipelines and React Web Apps. *2024 14th International Conference on Advanced Computer Information Technologies (ACIT)*, 586-591. https://doi.org/10.1109/ACIT62333.2024.10712482

Chen, L. (2015). Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, *32*(2), 50-54. https://doi.org/10.1109/MS.2015.27

Cruz, V. L., & Albuquerque, A. B. (2018). A DevOps Introduction Process for Legacy Systems. *2018 XLIV Latin American Computer Conference (CLEI)*, 139-148. https://doi.org/10.1109/CLEI.2018.00025

Fairbanks, J., Tharigonda, A., & Eisty, N. U. (2023). Analyzing the Effects of CI/CD on Open Source Repositories in GitHub and GitLab. *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, 176-181. https://doi.org/10.1109/SERA57763.2023.10197778

Febrianto, R. F., Nugraheni, D. M. K., Suharto, E., Widodo, A. P., & Ariyanti, Y. D. P. (2024a). Deployment Strategy with Integration Testing Implementation Using DevOps Method in Development and Production Environment. *2024 7th International Conferen-*

ce on Informatics and Computational Sciences (ICICoS), 256-261. https://doi.org/10.1109/ICICoS62600.2024.10636830

Febrianto, R. F., Nugraheni, D. M. K., Suharto, E., Widodo, A. P., & Ariyanti, Y. D. P. (2024b). Deployment Strategy with Integration Testing Implementation Using DevOps Method in Development and Production Environment. *2024 7th International Conference on Informatics and Computational Sciences (ICICoS)*, 256-261. https://doi.org/10.1109/ICICoS62600.2024.10636830

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press.

ISO/IEC. (2011). Maintainability - ISO/IEC 25010 Standard [Accessed: 2024-05-12].

Martínez Villegas, A., Noreña Cardona, P., Suescún Monsalve, E., González Palacio, L., & Pardo Calvache, C. (2022). Implementación de prácticas DevOps en un Sistema de Mainframe Legado. *Investigación e Innovación en Ingenierías*, *10*(2), 129-146. https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=fap&AN=161907576&lang=es&site=eds-live&scope=site&custid=s9496075

McAllister, A. J. (2011). The Case for Teaching Legacy Systems Modernization. *2011 Eighth International Conference on Information Technology: New Generations*, 251-256. https://doi.org/10.1109/ITNG.2011.51

Ramesh, R. S. (2024). DevOps and Agile Computing. *2024 1st International Conference on Communications and Computer Science (InCCCS)*, 1-5. https://doi.org/10.1109/InCCCS60947.2024.10593332

Sommerville, I., Melnikoff, S., Arakaki, R., & de Andrade Barbosa, E. (2007). *Engenharia de software*. Pearson Prentice Hall. https://books.google.com.co/books?id=ifIYOgAACAAJ

Su, L., & Storer, T. (2023). A Case Study of DevOps Adoption within a Large Financial Organisation. *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 403-413. https://doi.org/10.1109/ICSME58846.2023.00053

Trigo, A., Varajão, J., & Sousa, L. (2022). DevOps adoption: Insights from a large European Telco. *Cogent Engineering*, *9*(1), 1-31. https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=asn&AN=161674990&lang=es&site=eds-live&scope=site&custid=s9496075

Virmani, M. (2015). Understanding DevOps bridging the gap from continuous integration to continuous delivery. *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 78-82. https://doi.org/10.1109/INTECH.2015.7173368

Wiedemann, A., Wiesche, M., Gewald, H., & Krcmar, H. (2023). Integrating development and operations teams: A control approach for DevOps. *Information and Organization*, *33*(3), 100474. https://doi.org/https://doi.org/10.1016/j.infoandorg.2023.100474

9. Glosario de Términos

DevOps Un conjunto de prácticas que combina el desarrollo de software (Dev) y las operaciones de TI (Ops) con el objetivo de reducir el ciclo de vida de los sistemas y aumentar la entrega continua mediante la automatización, mejora de procesos y colaboración entre equipos.

Sistemas Legacy Sistemas de software antiguos que aún son esenciales para las operaciones de una organización pero que pueden presentar dificultades para integrarse con nuevas tecnologías o mantener actualizaciones frecuentes.

Integración Continua (CI) Una práctica de desarrollo de software donde los desarrolladores integran cambios de código en un repositorio central de manera continua, seguido de la ejecución automática de pruebas.

Despliegue Continuo (CD) Una práctica que extiende la integración continua mediante la automatización del despliegue del código en entornos de producción, asegurando que las aplicaciones estén listas para su lanzamiento en cualquier momento.

Automatización de Procesos Uso de tecnología para realizar tareas repetitivas sin intervención humana, lo que mejora la eficiencia y reduce errores.

Ciclo de Vida del Software Todas las fases por las que pasa una aplicación, desde su planificación y desarrollo hasta su mantenimiento y eventual retiro.

Pipeline Secuencia automatizada de tareas configuradas en un sistema de integración y entrega continua (CI/CD), que permiten compilar, probar, empaquetar y desplegar software de forma controlada y reproducible. Un pipeline define los pasos que deben ejecutarse desde que se produce un cambio en el código fuente hasta que se entrega un artefacto listo para producción o pruebas.

Mantenibilidad Atributo de calidad del software definido por la norma

ISO/IEC 25010:2011, que hace referencia a la facilidad con la que un producto de software puede ser modificado para corregir errores, mejorar su rendimiento o adaptarse a cambios en el entorno. Incluye subcaracterísticas como modularidad, reusabilidad, analizabilidad, modificabilidad y capacidad de prueba.

Gitlab Plataforma de gestión del ciclo de vida del software que integra control de versiones con Git, seguimiento de incidencias, revisión de código, y funcionalidades de integración y entrega continua (CI/CD). GitLab permite automatizar flujos de trabajo a través de archivos de configuración (.gitlab-ci.yml) que definen los pasos del pipeline.

GitLab Runner Componente de ejecución de GitLab CI/CD. Es un agente que se instala en una máquina física o virtual y que se encarga de ejecutar los jobs definidos en los pipelines de GitLab. Puede funcionar en diferentes entornos (Docker, Shell, VirtualBox, entre otros) y permite distribuir y automatizar tareas como compilación, pruebas o despliegue del software.

Artefacto Archivo generado como resultado del proceso de construcción (build) de una aplicación. Puede incluir ejecutables, archivos .jar, .war, bibliotecas u otros paquetes que serán desplegados o reutilizados.

Nexus (Repositorio de artefactos) Herramienta que actúa como repositorio centralizado para almacenar artefactos de software, como bibliotecas y archivos .jar o .war. Facilita el versionamiento y la gestión de dependencias entre proyectos.