



**Pontificia Universidad
Católica del Ecuador**
Seréis mis testigos

MANABÍ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE MANABÍ

CARRERA DE SOFTWARE

TRABAJO DE TITULACIÓN

**IMPLEMENTACIÓN DE UN PIPELINE CI/CD AUTOMATIZADO CON
DOCKER**

**Y GITHUB ACTIONS PARA OPTIMIZAR EL DESPLIEGUE Y LA
CALIDAD DE APLICACIONES WEB CONTENERIZADAS**

LÍNEA DE INVESTIGACIÓN

TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

SUBLÍNEA DE INVESTIGACIÓN

INGENIERÍA DE SOFTWARE, INNOVACION Y EMPRENDIMIENTO TIC

PREVIO AL TÍTULO DE

INGENIERO EN SOFTWARE

AUTOR

XAVIER EDUARDO NAVIA BRAVO

TUTOR

ING. VICTOR HUGO ALONZO PALACIOS, M.ENG.

PORTOVIEJO, ABRIL 2026

Certificación del Tutor

En mi calidad de tutor del trabajo de integración curricular, certifico haber revisado el presente manuscrito de investigación, el mismo que se ajusta a las normas vigentes de la Pontificia Universidad Católica del Ecuador Sede Manabí, cumpliendo la Normativa del Trabajo de Integración Curricular; en consecuencia, es apto para su presentación y sustentación.

En la ciudad de Portoviejo a los catorce días del mes de abril de dos mil veinte y seis.

ING. VICTOR HUGO ALONZO PALACIOS, M.ENG

CI: 1307757441

Tutor

2026

Acta de aprobación del Tribunal

El jurado examinador aprueba el presente trabajo de integración curricular en nombre de la Pontificia Universidad Católica del Ecuador, Sede Manabí.

En la ciudad de Portoviejo a los catorce días del mes de abril de dos mil veinte y seis.

ING. JOSE ALEXANDER

MACKENZIE RIVERO, Mgtr.

CI: 0960641694

Lector Revisor 1

ING. JOSE LUIS NARANJO

VILLOTA, M.ENG

CI: 1003528674

Lector Revisor 2

ING. VICTOR HUGO ALONZO PALACIOS, M.ENG

CI: 1307757441

Tutor

Declaración de Originalidad

Este manuscrito no contiene ningún tipo de material que ha sido aceptado para la obtención de un título universitario en otra institución, excepto en forma de información de soporte que ha sido debidamente citada en mi trabajo. Este trabajo es de total responsabilidad del autor, quien declara bajo juramento que ninguna sección de este trabajo de integración curricular infringe los derechos de autor de nadie.

En la ciudad de Portoviejo a los catorce días del mes de abril de dos mil veinte y seis.

Xavier Eduardo Navia Bravo

CI: 1315606010

Autor

Declaración de Derecho del Autor

Autorizo a la Pontificia Universidad Católica del Ecuador a distribuir este manuscrito de investigación en medios físicos y electrónicos con el fin de promover la divulgación de mis resultados a la comunidad científica y a la sociedad en general. Adicionalmente, autorizo el uso de los contenidos de esta investigación como bibliografía para fines académicos, por cualquier medio o procedimiento, citando como fuente de información al autor de este trabajo.

En la ciudad de Portoviejo a los catorce días del mes de abril de dos mil veinte y seis.

Xavier Eduardo Navia Bravo

CI: 1315606010

Autor

Agradecimientos

Agradezco profundamente a mis padres por su apoyo incondicional, comprensión y motivación constante a lo largo de mi formación académica. De igual manera, extendiendo mi gratitud a mis amigos y compañeros, quienes compartieron conocimientos, experiencias y acompañamiento durante este proceso.

Asimismo, expreso mi agradecimiento a mis tutores y profesores por su orientación, paciencia y valiosos aportes académicos, los cuales guiaron el desarrollo de este proyecto, fortaleciendo mi aprendizaje y fomentando mi crecimiento profesional.

También deseo agradecer a mis compañeros de clase que me han ayudado ante alguna duda o pregunta en la construcción del sistema.

Resumen

El presente trabajo de titulación tiene como objetivo desarrollar e implementar un pipeline CI/CD automatizado con Docker y GitHub Actions para optimizar el proceso de construcción, pruebas y despliegue de una plataforma web destinada a psicólogos, mejorando su calidad, seguridad y eficiencia operativa en entornos contenerizados. La investigación adopta un enfoque cuantitativo con un diseño aplicado y experimental que permite comparar el rendimiento del sistema antes y después de la implementación del pipeline.

El estudio se desarrolló utilizando una aplicación web construida con Next.js en el frontend y NestJS en el backend, desplegada en servicios de nube como Vercel y Railway. Se evaluaron métricas del modelo DevOps Research and Assessment (DORA), tales como frecuencia de despliegue, *lead time for changes*, tasa de fallos por cambio y tiempo medio de recuperación, junto con indicadores de seguridad relacionados con vulnerabilidades detectadas mediante herramientas de análisis automatizado.

Los resultados muestran que la implementación del pipeline CI/CD permitió reducir el tiempo promedio de despliegue del sistema de aproximadamente 120 minutos en un proceso manual a menos de 10 minutos mediante automatización, lo que representa una reducción cercana al 90 % en el tiempo de entrega de cambios. Además, la integración de validaciones automáticas de código y pruebas unitarias permitió detectar errores antes del despliegue, mejorando la estabilidad del sistema y reduciendo fallos en producción.

Finalmente, se obtuvo un modelo de pipeline CI/CD reproducible que puede aplicarse en proyectos web con características similares.

Palabras clave: CI/CD, Docker, GitHub, DevOps, seguridad.

Abstract

This undergraduate thesis aims to develop and implement an automated CI/CD pipeline using Docker and GitHub Actions to optimize the build, testing, and deployment processes of a web platform designed for psychologists, thereby improving its quality, security, and operational efficiency in containerized environments. The study adopts a quantitative approach with an applied and experimental design, enabling a comparison of system performance before and after pipeline implementation. The research was conducted using a web application built with Next.js for the frontend and NestJS for the backend, deployed on cloud services such as Vercel and Railway. Performance was evaluated using DevOps Research and Assessment (DORA) metrics, including deployment frequency, lead time for changes, change failure rate, and mean time to recovery, along with security indicators related to vulnerabilities identified through automated analysis tools. The results show that implementing the CI/CD pipeline reduced the average system deployment time from approximately 120 minutes in a manual process to less than 10 minutes through automation, representing a reduction of nearly 90 %. Furthermore, the integration of automated code validation and unit testing enabled early error detection prior to deployment, improving system stability and reducing failures in production environments. Finally, a reproducible CI/CD pipeline model was obtained, which can be applied to web projects with similar characteristics.

Keywords: CI/CD, Docker, GitHub, DevOps, security.

Tabla de Contenido

1. Introducción	1
Contexto y justificación	1
Planteamiento del problema	2
Objetivos de la investigación	3
Preguntas de investigación e hipótesis	4
2. Método	6
Tipo y enfoque de la investigación	6
Alcance de la investigación	7
Unidades de análisis	8
Técnicas de recolección de datos	9
Instrumentos de medición	10
Métricas de evaluación	10
Análisis inicial del proceso de desarrollo	12
Requerimientos del sistema	13
Requerimientos funcionales	13
Requerimientos no funcionales	14
Arquitectura del sistema	15

Arquitectura del frontend	16
Arquitectura del backend	17
Arquitectura de la base de datos	19
Desarrollo e integración del sistema	21
Diseño del pipeline CI/CD	22
Publicación de artefactos del sistema	25
Despliegue automatizado del sistema	27
3. Resultados	29
Resultados de la ejecución del pipeline CI/CD	29
Resultados del despliegue del frontend en Vercel	30
Cálculo de métricas de ejecución del pipeline	32
Síntesis de resultados y relación con métricas DORA	33
4. Discusión	36
Evaluación del logro del objetivo general	36
Análisis del logro de los objetivos específicos	37
Relación con las preguntas de investigación	37
Análisis de métricas DORA y contraste	38
Comparación con estudios previos	39
Fortalezas y limitaciones	39
Implicaciones	40
Recomendaciones	40

Síntesis	40
5. Conclusiones	41
Referencias bibliográficas	44
Lista de Apéndices	45
A. Apéndice A: Guía oficial de métricas DORA	46

Lista de Figuras

1.	Arquitectura general del frontend de la plataforma web.	16
2.	Arquitectura general del backend desarrollado con NestJS.	18
3.	Diagrama entidad-relación de la base de datos de la plataforma web.	20
4.	Diagrama de proceso del sistema de la plataforma web para psicólogos.	22
5.	Flujo general del pipeline CI/CD automatizado.	24
6.	Imágenes docker con GHCR	26
7.	Despliegue automatizado del backend en Railway	28
8.	Panel de métricas operativas del frontend en Vercel	31
9.	Representación gráfica de métricas del frontend	33

Lista de Tablas

1.	Resumen de métricas del frontend en Vercel	31
2.	Resumen de métricas del proceso de integración y despliegue	34
3.	Comparación conceptual de métricas DORA antes y después de la implementación del pipeline CI/CD	39

Introducción

Contexto y justificación

El desarrollo de aplicaciones web modernas exige altos niveles de disponibilidad, seguridad y escalabilidad. En este contexto, las metodologías tradicionales de desarrollo y despliegue de software resultan insuficientes ante la necesidad de entregar actualizaciones frecuentes y confiables en entornos de producción.

La necesidad de acelerar los procesos de desarrollo ha llevado a que, en la actualidad, se prefieran métodos automatizados y de entrega continua en la ingeniería de software (Amaro et al., 2024; Barbosa, 2026; Selinder, 2024). Las prácticas de DevOps se han desarrollado para unir sin interrupciones los procedimientos de creación, de prueba y de funcionamiento del software.

La contenerización es ahora fundamental en las arquitecturas actuales de software, ya que ayuda a empaquetar las dependencias del sistema y asegurar coherencia entre entornos de desarrollo, pruebas y producción (Farihane et al., 2025; Kumar et al., 2024). Diversos estudios indican que utilizar contenedores ayuda a crear sistemas escalables y a automatizar la implementación en la nube y en diferentes entornos (Barbosa, 2026).

Además, investigaciones más recientes indican que integrar medidas de seguridad en los flujos de trabajo de desarrollo es crucial para disminuir riesgos en el software y garantizar la fiabilidad de la entrega constante (Sánchez Ruiz & Domínguez Mayo, 2023). Estas prácticas ayudan a realizar validaciones automáticas, revisar el código y llevar a cabo pruebas de seguridad a lo largo del proceso de desarrollo de software. Investigaciones recientes sobre la seguridad en los flujos de trabajo CI/CD indican que al automatizar la detección de vulnerabilidades y al integrar métodos DevSecOps, se puede mejorar la seguridad del software y disminuir los riesgos derivados de

dependencias débiles (Dhandapani, 2025; Pan et al., 2024).

Planteamiento del problema

El crecimiento de los servicios en línea requiere sistemas tecnológicos que puedan mantener la información segura y accesible para los usuarios. Es fundamental proteger los datos sensibles en plataformas digitales. Aunque la tecnología avanza, todavía hay una brecha entre las ideas teóricas y su aplicación en proyectos web pequeños o en entornos académicos.

Varios estudios han puesto de relieve las ventajas de los procesos de CI/CD en entornos industriales. Sin embargo, falta investigación sobre su impacto en proyectos web académicos que utilizan tecnologías modernas como Docker, GitHub Actions y plataformas de implementación en la nube.

No obstante, revisiones sistemáticas recientes sugieren que la adopción gradual de pipelines CI/CD adaptados al contexto de equipos pequeños mejora la calidad del software y la eficiencia del proceso de desarrollo (Ccallo-Luque & Quispe-Quispe, 2024).

En proyectos de desarrollo de software de pequeña escala y en contextos académicos, es frecuente que la integración, validación y despliegue de aplicaciones web se realicen de forma Manual o semiautomatizada.

Esta situación se presenta cuando no existen flujos estandarizados para construir, probar y publicar nuevas versiones, y cuando el proceso depende de actividades repetitivas ejecutadas por el desarrollador sin trazabilidad automática.

El problema concreto es que la falta de automatización en el ciclo de vida del software genera discrepancias entre los entornos de desarrollo, de pruebas y de producción. Esto se traduce en plazos de entrega más largos para los cambios, lo que dificulta garantizar la calidad del código antes del lanzamiento. En consecuencia, existe un mayor riesgo de fallos de configuración, errores durante la implementación y una falta de pruebas claras para auditar los cambios, los

plazos y las validaciones tras el lanzamiento.

Organismos como la *National Security Agency (NSA)* y la *Cybersecurity and Infrastructure Security Agency (CISA)* han destacado que un entorno comprometido de CI/CD podría permitir a los atacantes insertar código malicioso en el software, afectando a todos los sistemas que lo utilizan, incluso sin ser detectados de inmediato por los desarrolladores.

En el ámbito de la plataforma web para servicios psicológicos, se ha observado que realizar despliegues manuales conlleva discrepancias entre los entornos de desarrollo y de producción, retrasos en la publicación y la falta de sistemas para detectar de forma sistemática fallos de seguridad. Estas carencias afectan la calidad del servicio prestado y pueden comprometer la privacidad de los datos de los usuarios.

Por lo tanto, el principal desafío de este estudio es la falta de un sistema automatizado que integre, valide e implemente de manera efectiva, asegurando así la trazabilidad, la reproducibilidad y el control en el desarrollo del software de la plataforma web para la gestión de servicios psicológicos.

Objetivos de la investigación

El objetivo general de esta investigación es implementar un pipeline CI/CD automatizado utilizando Docker y GitHub Actions para evaluar su impacto en el proceso de integración, validación y despliegue de una plataforma web destinada a la gestión de servicios psicológicos.

De manera específica, se plantean los siguientes objetivos:

- Analizar el flujo actual de desarrollo y despliegue de la plataforma web, identificando cuellos de botella y riesgos asociados (DevOps Research and Assessment, 2021).
- Diseñar un pipeline CI/CD que integre procesos automatizados de construcción, pruebas, análisis de código y contenerización.

- Implementar workflows en GitHub Actions para orquestar las distintas etapas del ciclo de vida del software.
- Integrar pruebas automatizadas y herramientas de análisis de vulnerabilidades en el pipeline CI/CD para reforzar la seguridad del sistema (Boles et al., 2024).
- Evaluar el impacto del pipeline mediante métricas operativas basadas en el modelo DevOps Research and Assessment (DORA), tales como frecuencia de despliegue, *lead time for changes*, tasa de fallos por cambio y tiempo medio de recuperación.

Preguntas de investigación e hipótesis

La pregunta principal que guía esta investigación es la siguiente:

- ¿De qué manera la implementación de un pipeline CI/CD automatizado influye en el proceso de integración, validación y despliegue de una plataforma web destinada a la gestión de servicios psicológicos?

A partir de esta pregunta general se derivan las siguientes preguntas específicas:

- ¿Cómo varía el tiempo requerido para integrar y desplegar cambios en el sistema tras la implementación del pipeline CI/CD?
- ¿Cómo cambia la tasa de ejecuciones fallidas del proceso de integración continua después de la automatización del pipeline?
- ¿Qué efecto tiene la integración de herramientas de análisis de vulnerabilidades en la detección de amenazas críticas dentro de las imágenes Docker generadas por el sistema?
- ¿Cómo influye la automatización del pipeline en la trazabilidad y control del proceso de construcción y despliegue del software?

A partir de estas preguntas se plantean las siguientes hipótesis de investigación:

- H1: La implementación de un pipeline CI/CD automatizado reduce el *lead time for changes* entre la integración de un cambio en el repositorio y su despliegue en el entorno de ejecución (DevOps Research and Assessment, 2021).
- H2: La implementación del pipeline CI/CD incrementa la frecuencia de integración del sistema sin aumentar la tasa de ejecuciones fallidas del proceso de integración continua.
- H3: La incorporación de validaciones automatizadas de calidad de código y pruebas unitarias dentro del pipeline CI/CD contribuye a detectar errores en etapas tempranas del proceso de integración continua, reduciendo la probabilidad de fallos en producción.

Método

Este capítulo describe la metodología empleada para el diseño, implementación y evaluación de un pipeline de integración y entrega continua (CI/CD) aplicado a una plataforma web orientada a la gestión de servicios psicológicos. El propósito metodológico del estudio es analizar el impacto de la automatización del ciclo de vida del software en el proceso de desarrollo y despliegue del sistema.

A diferencia de proyectos de desarrollo tradicionales donde el foco principal es la construcción funcional de la aplicación, esta investigación se centra en la implementación de un proceso automatizado de integración, validación y despliegue continuo. La plataforma web desarrollada funciona como un caso de aplicación que permite evaluar de forma empírica el comportamiento del pipeline CI/CD en un entorno real de desarrollo y despliegue.

La metodología se estructura en varias etapas que abarcan desde el análisis inicial del proceso de desarrollo, la definición de requerimientos del sistema, el diseño de la arquitectura tecnológica, la implementación del pipeline automatizado y finalmente la evaluación del impacto mediante métricas operativas.

Tipo y enfoque de la investigación

La presente investigación se clasifica como un estudio de tipo aplicado, ya que busca resolver un problema técnico relacionado con la automatización del proceso de integración y despliegue de software mediante la implementación de un pipeline CI/CD.

El enfoque metodológico es cuantitativo, debido a que el impacto de la solución propuesta se evalúa a partir de métricas operativas del proceso de entrega de software. Estas métricas incluyen

indicadores como el *lead time for changes*, la frecuencia de despliegue, la tasa de fallos por cambio y el tiempo medio de recuperación (MTTR), basados en el modelo DevOps Research and Assessment (DORA).

Asimismo, el diseño de la investigación corresponde a un estudio evaluativo con comparación de escenarios. Se analizan dos situaciones diferenciadas:

- Escenario previo a la automatización del proceso de despliegue.
- Escenario posterior a la implementación del pipeline CI/CD automatizado.

Esta comparación permite evaluar el impacto de la automatización sobre el proceso de construcción, validación y despliegue del sistema.

Desde el punto de vista del diseño metodológico, el estudio corresponde a un diseño no experimental de tipo pretest–posttest sin grupo de control. En este enfoque no se manipulan variables bajo condiciones controladas, sino que se observa el comportamiento del proceso de desarrollo del software antes y después de la implementación del pipeline CI/CD. Este tipo de diseño es común en estudios aplicados de ingeniería de software donde el objetivo es evaluar mejoras introducidas en procesos o herramientas dentro de un entorno real de desarrollo.

Alcance de la investigación

El alcance de esta investigación se centra en la implementación y evaluación técnica de un pipeline CI/CD automatizado aplicado a una plataforma web destinada a la gestión de servicios psicológicos.

El estudio contempla el ciclo completo de integración y despliegue del software, desde la incorporación de cambios en el repositorio de código hasta su despliegue en entornos de nube. Este proceso incluye actividades de construcción del sistema, ejecución de pruebas automatizadas, validación de calidad del código, construcción de imágenes Docker y despliegue

en plataformas de infraestructura en la nube.

En términos tecnológicos, el alcance del estudio incluye el uso de un repositorio de código en GitHub, la automatización del pipeline mediante GitHub Actions, la construcción y almacenamiento de imágenes Docker en GitHub Container Registry (GHCR), y el despliegue automatizado del frontend y backend en las plataformas Vercel y Railway respectivamente.

El análisis del impacto del pipeline se realiza mediante indicadores operativos basados en métricas DORA, calculadas a partir de los registros generados automáticamente por las herramientas de integración continua y despliegue utilizadas durante el estudio.

Como limitación del estudio, la investigación se desarrolla sobre un único sistema de aplicación y dentro de un entorno académico, por lo que los resultados deben interpretarse como evidencia aplicada en un caso específico. Sin embargo, el enfoque metodológico y las prácticas DevOps implementadas permiten extrapolar los principios del estudio a proyectos de desarrollo de software de características similares.

Unidades de análisis

En esta investigación las unidades de análisis no corresponden a usuarios humanos, sino a los artefactos técnicos y procesos involucrados en el ciclo de vida del software. El estudio se centra en el análisis del proceso de integración, validación y despliegue del sistema, así como en los registros generados por las herramientas utilizadas durante la automatización del pipeline.

Las principales unidades de análisis consideradas en el estudio son las siguientes:

- El repositorio de código fuente alojado en GitHub, donde se registran los commits y cambios del sistema.
- Los workflows de integración continua configurados en GitHub Actions.
- Las ejecuciones del pipeline CI/CD generadas automáticamente ante cambios en el

repositorio.

- Las imágenes Docker construidas y publicadas en GitHub Container Registry.
- Los despliegues del frontend y backend realizados en las plataformas Vercel y Railway.
- Los registros de ejecución (logs) generados por el pipeline y las plataformas de despliegue.

El análisis de estas unidades permite observar el comportamiento del proceso de desarrollo y despliegue del sistema, así como evaluar el impacto de la automatización del pipeline CI/CD sobre el ciclo de vida del software.

Técnicas de recolección de datos

Para la recolección de datos se utilizó la técnica de observación del comportamiento del sistema mediante el análisis de registros generados automáticamente por las herramientas de integración continua y despliegue continuo.

Este enfoque permite obtener información objetiva sobre el desempeño del proceso de desarrollo, ya que las herramientas utilizadas registran automáticamente información relevante sobre cada ejecución del pipeline, incluyendo tiempos de ejecución, estado de las validaciones, errores detectados y eventos de despliegue.

Adicionalmente, se empleó un enfoque comparativo para analizar el comportamiento del proceso de desarrollo antes y después de la implementación del pipeline CI/CD. Esta comparación permite identificar cambios en la eficiencia del proceso de entrega del software y evaluar el impacto de la automatización sobre el ciclo de vida del sistema.

Instrumentos de medición

Los datos utilizados en el estudio fueron obtenidos a partir de los registros generados automáticamente por las herramientas utilizadas en el pipeline CI/CD. Estas herramientas funcionan como instrumentos de medición del comportamiento del sistema durante su proceso de integración, validación y despliegue.

En particular, se utilizaron las siguientes fuentes de datos:

- **GitHub Actions:** registros de ejecución de workflows, duración de cada ejecución del pipeline, estado de ejecución (éxito o fallo) y tiempo total de ejecución.
- **GitHub Container Registry:** historial de imágenes Docker construidas por el pipeline y sus respectivas versiones.
- **Vercel:** métricas operativas del frontend desplegado, incluyendo número de solicitudes atendidas, uso de funciones serverless y consumo de recursos.
- **Railway:** métricas operativas del backend desplegado, incluyendo tráfico de red, número de solicitudes, tasa de errores y tiempos de respuesta del servicio.

Estos instrumentos permiten recolectar información empírica sobre el comportamiento del sistema en entornos reales de ejecución y despliegue, lo que proporciona una base objetiva para evaluar el impacto de la automatización del pipeline.

Métricas de evaluación

Para evaluar el desempeño del pipeline CI/CD se utilizaron métricas basadas en el modelo DevOps Research and Assessment (DORA), ampliamente adoptado en estudios sobre prácticas DevOps y entrega continua de software.

Las métricas consideradas en esta investigación son las siguientes:

- **Lead Time for Changes:** tiempo transcurrido desde la integración de un cambio en el repositorio hasta su despliegue en el entorno de producción.
- **Deployment Frequency:** número de despliegues realizados durante el periodo de observación.
- **Change Failure Rate:** proporción de ejecuciones fallidas del pipeline en relación con el número total de ejecuciones.
- **Mean Time to Recovery (MTTR):** tiempo requerido para restaurar el sistema a un estado funcional tras la detección de un fallo.

Para efectos del análisis, cada métrica fue operacionalizada utilizando los registros generados automáticamente por las herramientas de integración y despliegue continuo utilizadas en el estudio. En particular:

- **Lead Time for Changes:** se calculó como el intervalo de tiempo entre la integración de un cambio en la rama principal del repositorio y la finalización del proceso de despliegue correspondiente.
- **Deployment Frequency:** se determinó contabilizando el número de despliegues realizados automáticamente durante el periodo de observación del sistema.
- **Change Failure Rate:** se estimó como la proporción de ejecuciones fallidas del pipeline respecto al número total de ejecuciones registradas.
- **Mean Time to Recovery (MTTR):** se estimó a partir del tiempo necesario para corregir un fallo detectado y restablecer el funcionamiento del sistema mediante una nueva ejecución del pipeline.

Los datos necesarios para calcular estas métricas fueron obtenidos directamente de los registros de ejecución del pipeline en GitHub Actions y de los eventos de despliegue registrados por las plataformas de infraestructura utilizadas.

Estas métricas permiten evaluar de forma cuantitativa el desempeño del proceso de integración y despliegue continuo implementado en el estudio.

Análisis inicial del proceso de desarrollo

Antes de la implementación del pipeline CI/CD se realizó un análisis del proceso de desarrollo y despliegue del sistema con el objetivo de establecer una línea base que permitiera comparar posteriormente el impacto de la automatización.

En el escenario inicial, el desarrollo de la aplicación se realizaba mediante un flujo tradicional basado en control de versiones con Git, donde los cambios en el código eran integrados manualmente en el repositorio. El proceso de despliegue del sistema dependía de acciones ejecutadas directamente por el desarrollador, tales como la instalación manual de dependencias, la ejecución local de pruebas y la configuración del entorno de despliegue.

Durante este análisis se identificaron varias limitaciones en el proceso inicial:

- Dependencia de configuraciones locales del entorno de desarrollo.
- Ausencia de validaciones automatizadas antes del despliegue.
- Falta de trazabilidad en el proceso de construcción del sistema.
- Ausencia de métricas objetivas para evaluar el desempeño del proceso de despliegue.
- Riesgo de errores humanos durante la ejecución manual de tareas.

Asimismo, el despliegue del sistema requería una serie de pasos manuales ejecutados por el desarrollador, lo cual incrementaba el tiempo necesario para publicar nuevas versiones del software y dificultaba la reproducibilidad del entorno de ejecución.

De manera general, el proceso de despliegue manual implicaba varias etapas ejecutadas directamente por el desarrollador, incluyendo la instalación de dependencias, la ejecución de

pruebas locales y la actualización del entorno de ejecución. Este procedimiento podía requerir varios minutos o incluso horas dependiendo de la configuración del entorno y de las verificaciones realizadas manualmente.

Además, el proceso inicial no generaba registros automáticos que permitieran analizar el comportamiento del sistema ni medir de forma objetiva el tiempo necesario para completar cada etapa del despliegue. Esta ausencia de métricas dificultaba evaluar la eficiencia del proceso de entrega del software antes de la introducción del pipeline automatizado.

Este diagnóstico inicial permitió identificar la necesidad de implementar un pipeline CI/CD automatizado que permitiera estandarizar el proceso de construcción, validación y despliegue del sistema, reduciendo la dependencia de procesos manuales y mejorando la trazabilidad del ciclo de vida del software.

Requerimientos del sistema

El desarrollo de la plataforma web se basó en un conjunto de requerimientos funcionales y no funcionales definidos a partir de las necesidades del sistema de gestión de servicios psicológicos y de los objetivos técnicos del proyecto.

Estos requerimientos permitieron establecer las funcionalidades principales del sistema y las características técnicas necesarias para su implementación, así como definir las condiciones bajo las cuales el sistema debía operar dentro del pipeline CI/CD automatizado.

Requerimientos funcionales

Los requerimientos funcionales definen las capacidades que debe ofrecer el sistema para gestionar la interacción entre los distintos actores involucrados en la plataforma.

Entre los principales requerimientos funcionales se encuentran:

- Registro y autenticación de usuarios con roles diferenciados (administrador, psicólogo y paciente).
- Gestión de citas psicológicas, incluyendo la creación, modificación y cancelación de citas.
- Definición de disponibilidad de los profesionales mediante reglas de horario y excepciones.
- Registro de sesiones y evaluaciones posteriores a cada consulta.
- Almacenamiento de información clínica asociada a las citas realizadas.
- Control de acceso a funcionalidades según el rol del usuario.

Estos requerimientos permiten modelar el comportamiento principal del sistema y establecer las funcionalidades que deben ser soportadas por la arquitectura tecnológica implementada.

Requerimientos no funcionales

Los requerimientos no funcionales definen las características técnicas que debe cumplir el sistema para garantizar su correcto funcionamiento en entornos de despliegue modernos.

Entre los principales requerimientos no funcionales se establecieron los siguientes:

- Seguridad en el acceso al sistema mediante autenticación basada en JSON Web Tokens (JWT).
- Escalabilidad del sistema mediante arquitectura modular.
- Consistencia entre entornos de desarrollo, pruebas y producción.
- Automatización del proceso de despliegue mediante pipeline CI/CD.
- Trazabilidad de cambios mediante control de versiones y registros automáticos de ejecución.

- Disponibilidad del sistema en entornos de infraestructura en la nube.

La definición de estos requerimientos permitió establecer las bases técnicas para el diseño de la arquitectura del sistema y para la integración del proceso de automatización del ciclo de vida del software.

Arquitectura del sistema

A partir de los requerimientos funcionales y no funcionales definidos previamente, se diseñó una arquitectura tecnológica orientada a facilitar la integración con un pipeline CI/CD automatizado y a garantizar la separación de responsabilidades entre los distintos componentes del sistema.

La arquitectura del sistema se estructuró siguiendo un modelo desacoplado que separa claramente el frontend, el backend y la capa de persistencia de datos. Este enfoque permite desarrollar, probar y desplegar cada componente de manera independiente, lo cual resulta especialmente relevante para entornos de integración y entrega continua.

La separación entre frontend, backend y capa de datos también facilita la integración con procesos de integración y entrega continua, ya que cada componente puede ser construido, validado y desplegado de manera independiente dentro del pipeline CI/CD. Esta característica resulta especialmente relevante en entornos DevOps, donde la automatización del ciclo de vida del software requiere artefactos claramente definidos y procesos reproducibles de construcción y despliegue.

Además, la arquitectura fue diseñada considerando la contenerización del backend mediante Docker y su despliegue automatizado en plataformas de infraestructura en la nube, lo que permite garantizar consistencia entre entornos de desarrollo, pruebas y producción.

Arquitectura del frontend

El frontend del sistema fue desarrollado utilizando el framework Next.js, el cual permite construir aplicaciones web modernas basadas en componentes reutilizables y renderizado híbrido. Esta elección tecnológica facilita la integración con plataformas de despliegue automático como Vercel, donde cada cambio en el repositorio puede generar una nueva versión desplegada del sistema.

La aplicación se organiza mediante una estructura de rutas basada en roles de usuario, lo que permite separar claramente las funcionalidades destinadas a administradores, psicólogos y pacientes. Asimismo, se utilizan componentes reutilizables para la construcción de interfaces de usuario, lo que mejora la mantenibilidad del código.

La Figura 1 presenta la arquitectura general del frontend de la plataforma web desarrollada con el framework Next.js. Esta arquitectura define la organización de los componentes principales encargados de gestionar la interfaz de usuario y la interacción con el sistema.

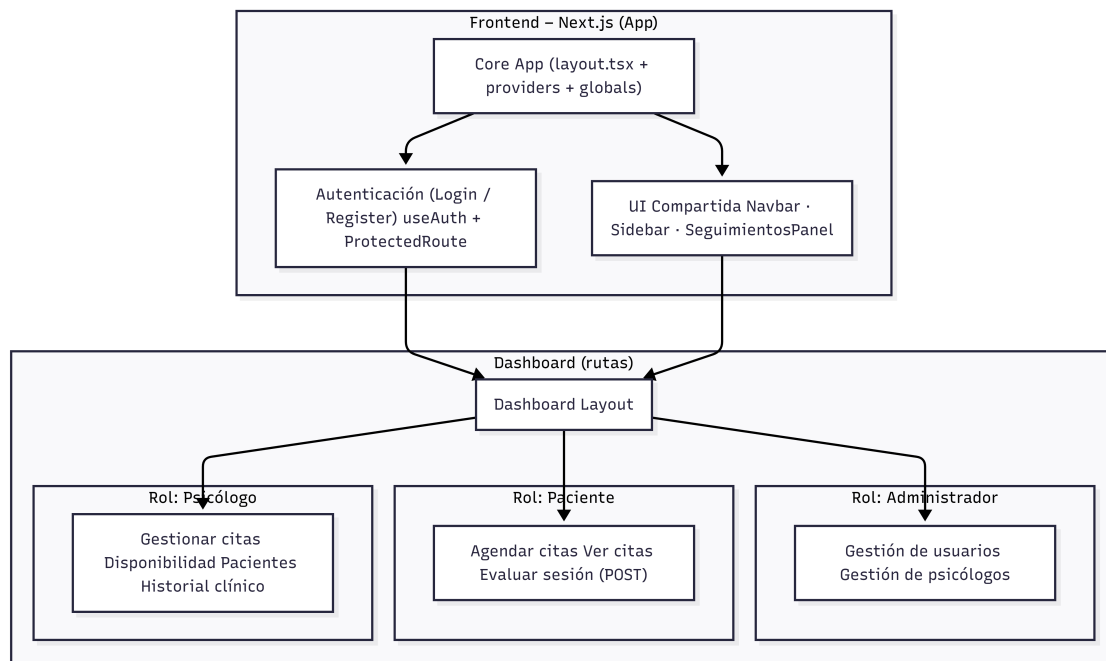


Figura 1. Arquitectura general del frontend de la plataforma web.

Como se observa en la Figura 1, el núcleo de la aplicación está representado por el componente

Core App, que centraliza la configuración global de la interfaz, incluyendo la gestión de proveedores, estilos y estructura principal de la aplicación.

A partir de este núcleo se organizan los módulos de autenticación y la interfaz de usuario compartida. El módulo de autenticación se encarga de gestionar procesos como inicio de sesión, registro de usuarios y control de acceso mediante rutas protegidas, mientras que los componentes de interfaz compartida incluyen elementos como la barra de navegación, el panel lateral y las vistas principales del sistema.

Una vez autenticado, el usuario accede al *Dashboard* de la plataforma, desde donde se habilitan distintas funcionalidades según el rol asignado dentro del sistema. En el caso del rol de psicólogo, se incluyen herramientas para la gestión de citas, disponibilidad y seguimiento clínico de pacientes. El rol de paciente permite la visualización y programación de citas, así como la evaluación de sesiones. Por su parte, el rol de administrador dispone de funciones orientadas a la gestión de usuarios y psicólogos dentro de la plataforma.

En conjunto, esta arquitectura organiza la interfaz de la aplicación en componentes modulares que facilitan la navegación del usuario y la comunicación con el backend mediante solicitudes HTTP hacia la API del sistema.

Arquitectura del backend

El backend del sistema fue desarrollado utilizando el framework NestJS, el cual proporciona una arquitectura modular basada en principios de programación orientada a servicios. Esta arquitectura facilita la organización del código en módulos independientes que encapsulan funcionalidades específicas del sistema.

La estructura del backend se compone principalmente de los siguientes elementos:

- **Módulos:** agrupan funcionalidades relacionadas dentro del sistema.
- **Controladores:** gestionan las solicitudes HTTP provenientes del frontend.

- **Servicios:** contienen la lógica de negocio del sistema.
- **Entidades:** representan las estructuras de datos persistidas en la base de datos.

Esta organización modular permite mejorar la mantenibilidad del sistema y facilita la ejecución de pruebas automatizadas sobre componentes específicos del backend.

La Figura 2 presenta la arquitectura general del backend implementado en el proyecto utilizando el framework NestJS. Esta arquitectura organiza los componentes del sistema en distintas capas que permiten estructurar la lógica de negocio, la gestión de servicios y el acceso a los datos.

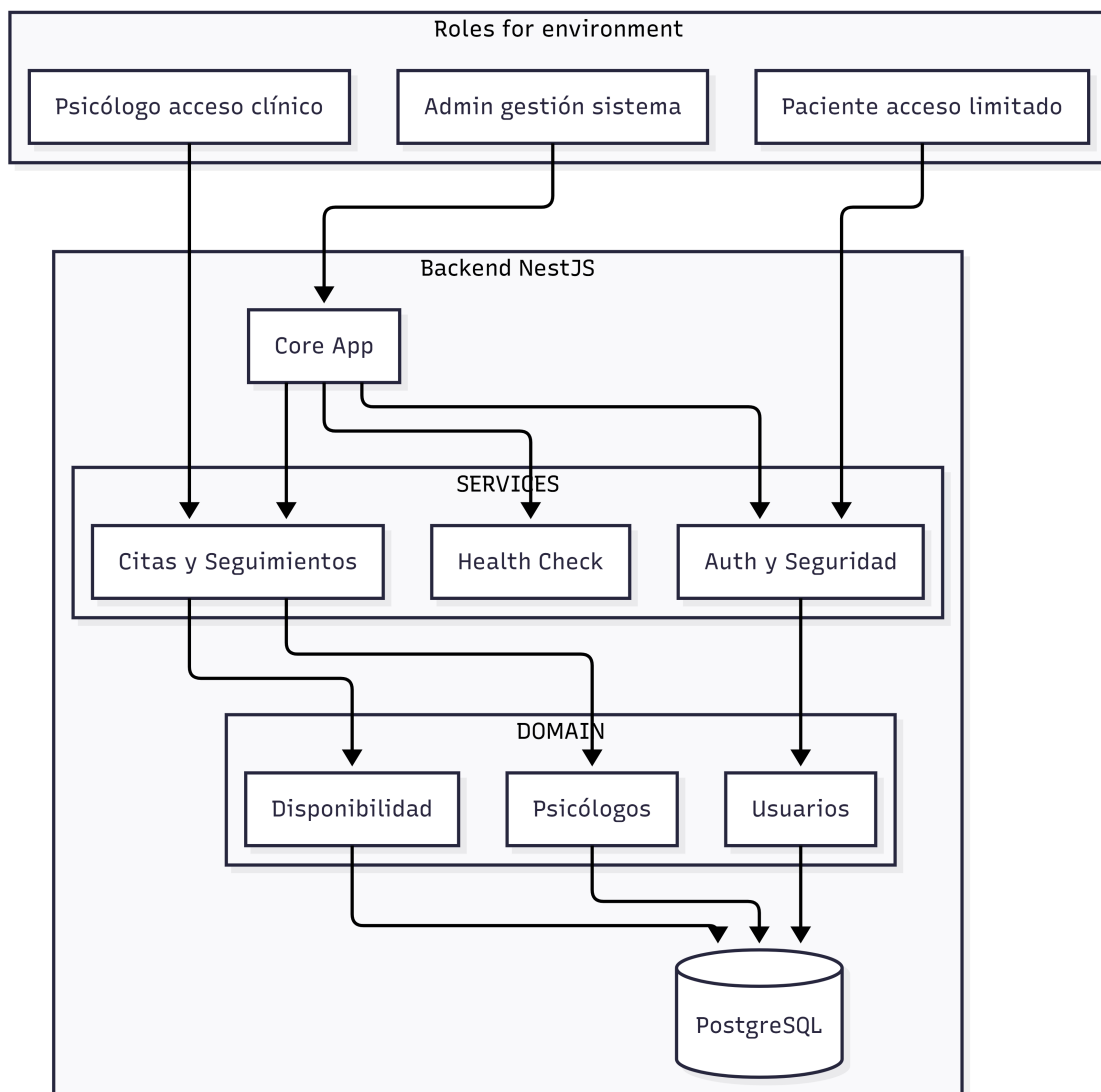


Figura 2. Arquitectura general del backend desarrollado con NestJS.

Como se observa en la Figura 2, el backend está compuesto por un núcleo principal denominado *Core App*, que actúa como punto central de inicialización y coordinación de los diferentes módulos del sistema. A partir de este núcleo se organizan los servicios principales de la aplicación.

En la capa de servicios se incluyen módulos encargados de funcionalidades específicas del sistema, tales como el módulo de autenticación y seguridad, responsable de la gestión de usuarios y control de acceso; el módulo de citas y seguimientos, que administra la programación y registro de sesiones; y el módulo de verificación de estado del sistema (*Health Check*), utilizado para monitorear el funcionamiento de la aplicación.

Estos servicios interactúan con la capa de dominio, donde se gestionan las entidades principales del sistema, incluyendo usuarios, psicólogos y disponibilidad de citas. Finalmente, la capa de persistencia se conecta con una base de datos PostgreSQL, donde se almacenan de manera estructurada los datos del sistema.

Esta arquitectura por capas permite separar responsabilidades dentro del backend, facilitando la organización del código, el mantenimiento del sistema y la escalabilidad de la plataforma.

Arquitectura de la base de datos

Para el almacenamiento de la información del sistema se utilizó una base de datos relacional PostgreSQL. Este tipo de base de datos permite asegurar integridad referencial entre las distintas entidades del sistema y facilita la gestión estructurada de la información.

El diseño del modelo de datos se realizó mediante un esquema entidad-relación que define las principales entidades del sistema, tales como usuarios, citas, sesiones, evaluaciones y disponibilidad de los psicólogos.

La Figura 3 muestra el diagrama entidad-relación de la base de datos utilizada en la plataforma web.

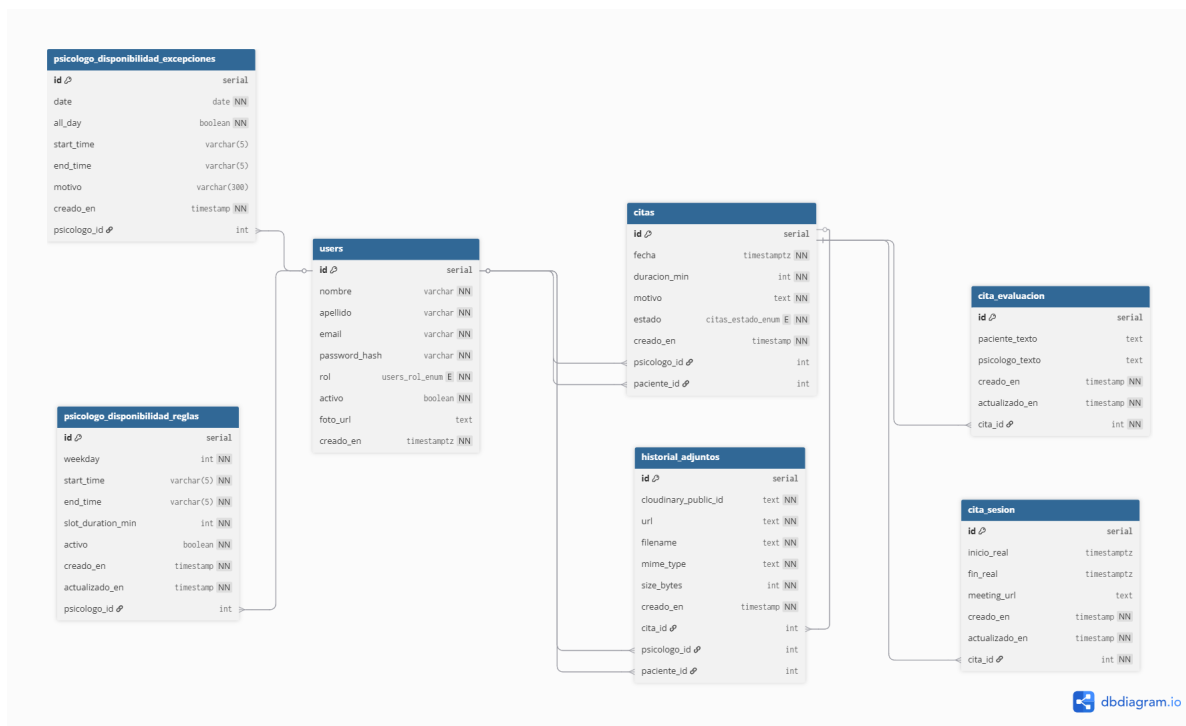


Figura 3. Diagrama entidad-relación de la base de datos de la plataforma web.

El modelo de datos permite representar las relaciones existentes entre los distintos elementos del sistema y garantizar la integridad de la información almacenada en la plataforma.

Como se observa en la Figura 3, el modelo entidad-relación define las principales entidades que participan en el funcionamiento del sistema, incluyendo usuarios, psicólogos, citas, sesiones y evaluaciones. Cada una de estas entidades se encuentra relacionada mediante claves primarias y foráneas que permiten establecer vínculos entre los distintos registros almacenados en la base de datos.

Estas relaciones permiten representar de forma estructurada la interacción entre los distintos actores del sistema y las operaciones realizadas durante la gestión de citas psicológicas. De esta manera, el modelo de datos facilita la implementación de operaciones de consulta, inserción y actualización de información desde el backend de la aplicación.

Desarrollo e integración del sistema

El desarrollo del sistema web se realizó considerando desde su fase de diseño la integración con un proceso automatizado de integración y despliegue continuo. La plataforma fue concebida no solo como una aplicación funcional, sino como un entorno de prueba que permitiera evaluar el comportamiento de un pipeline CI/CD aplicado a un proyecto web real.

El backend fue desarrollado utilizando el framework NestJS, implementando una arquitectura modular basada en controladores, servicios y entidades. Esta organización permitió separar la lógica de negocio, el acceso a datos y las interfaces de comunicación con el frontend.

Para la autenticación y autorización del sistema se implementó un mecanismo basado en JSON Web Tokens (JWT), permitiendo controlar el acceso a funcionalidades según el rol de cada usuario dentro del sistema.

Una vez completado el desarrollo funcional del backend, la aplicación fue contenerizada mediante Docker. La creación de una imagen Docker permitió encapsular el código fuente, las dependencias del proyecto y la configuración necesaria para su ejecución en un entorno reproducible.

El frontend del sistema fue desarrollado utilizando Next.js, organizando la estructura de la aplicación en componentes reutilizables y rutas protegidas según los distintos roles de usuario.

Con el fin de representar el flujo funcional del sistema desde la interacción del usuario hasta la persistencia de los datos, se elaboró un diagrama de proceso que describe las principales etapas del funcionamiento de la plataforma.

La Figura 4 muestra el flujo general de interacción del sistema.

Como se observa en la Figura 4, el flujo de interacción del sistema inicia cuando el usuario accede a la plataforma mediante la interfaz del frontend. A través de esta interfaz, el usuario puede realizar distintas acciones, como iniciar sesión, gestionar citas o consultar información del sistema.

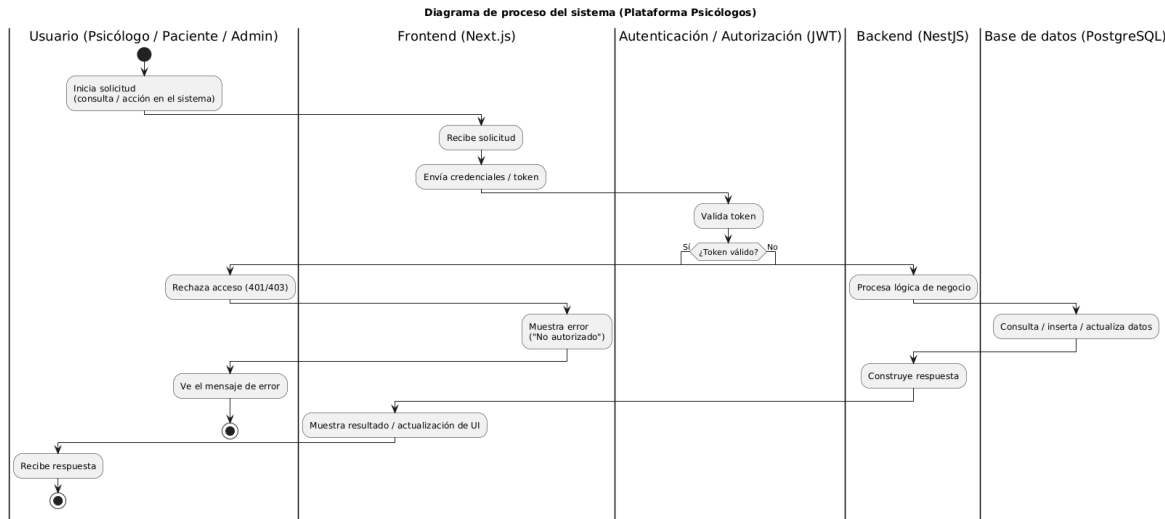


Figura 4. Diagrama de proceso del sistema de la plataforma web para psicólogos.

Las solicitudes generadas desde el frontend son enviadas al backend mediante la API del sistema, donde se aplican las reglas de negocio correspondientes. El backend procesa estas solicitudes utilizando los servicios implementados en la aplicación y realiza las operaciones necesarias sobre la base de datos para almacenar o recuperar información.

Finalmente, los resultados del procesamiento son devueltos al frontend en forma de respuestas HTTP, permitiendo actualizar la interfaz de usuario y mostrar la información solicitada. Este flujo de interacción describe el funcionamiento general del sistema durante la ejecución de las operaciones principales de la plataforma.

Diseño del pipeline CI/CD

Una vez implementada la arquitectura del sistema, se diseñó un pipeline de integración y entrega continua (CI/CD) utilizando GitHub Actions como plataforma de automatización.

La incorporación del pipeline CI/CD en el proyecto permite automatizar tareas repetitivas del proceso de desarrollo, tales como la ejecución de pruebas, la validación del código y la construcción de artefactos de despliegue. Esta automatización reduce la probabilidad de errores

humanos, mejora la reproducibilidad del proceso de construcción del sistema y facilita la trazabilidad de las distintas versiones del software generadas durante el desarrollo.

El pipeline fue configurado para ejecutarse automáticamente cuando se realizan cambios en la rama principal del repositorio. Cada ejecución del pipeline realiza una serie de tareas destinadas a validar la integridad del código y preparar el sistema para su despliegue.

Las principales etapas del pipeline incluyen:

- Instalación de dependencias del proyecto.
- Ejecución de pruebas unitarias mediante el framework Jest.
- Validación del estilo del código utilizando ESLint.
- Construcción de la imagen Docker del backend.
- Publicación de la imagen Docker en GitHub Container Registry.

Este flujo permite garantizar que cada cambio incorporado al repositorio sea validado automáticamente antes de ser utilizado en el proceso de despliegue.

La Figura 5 ilustra el flujo general del pipeline CI/CD implementado en el proyecto.

Como se muestra en la Figura 5, el pipeline se activa cuando el desarrollador realiza un *commit* o una solicitud de integración (*pull request*) en el repositorio del proyecto. A partir de este evento, el sistema de integración continua GitHub Actions ejecuta automáticamente una serie de etapas destinadas a validar el código antes de su despliegue.

En primer lugar, se ejecutan tareas de integración continua (CI), que incluyen la instalación de dependencias, la construcción del proyecto y la verificación del estilo del código mediante herramientas de análisis estático. Posteriormente se ejecutan pruebas automatizadas utilizando el framework Jest, con el objetivo de verificar el correcto funcionamiento de los componentes principales del sistema.

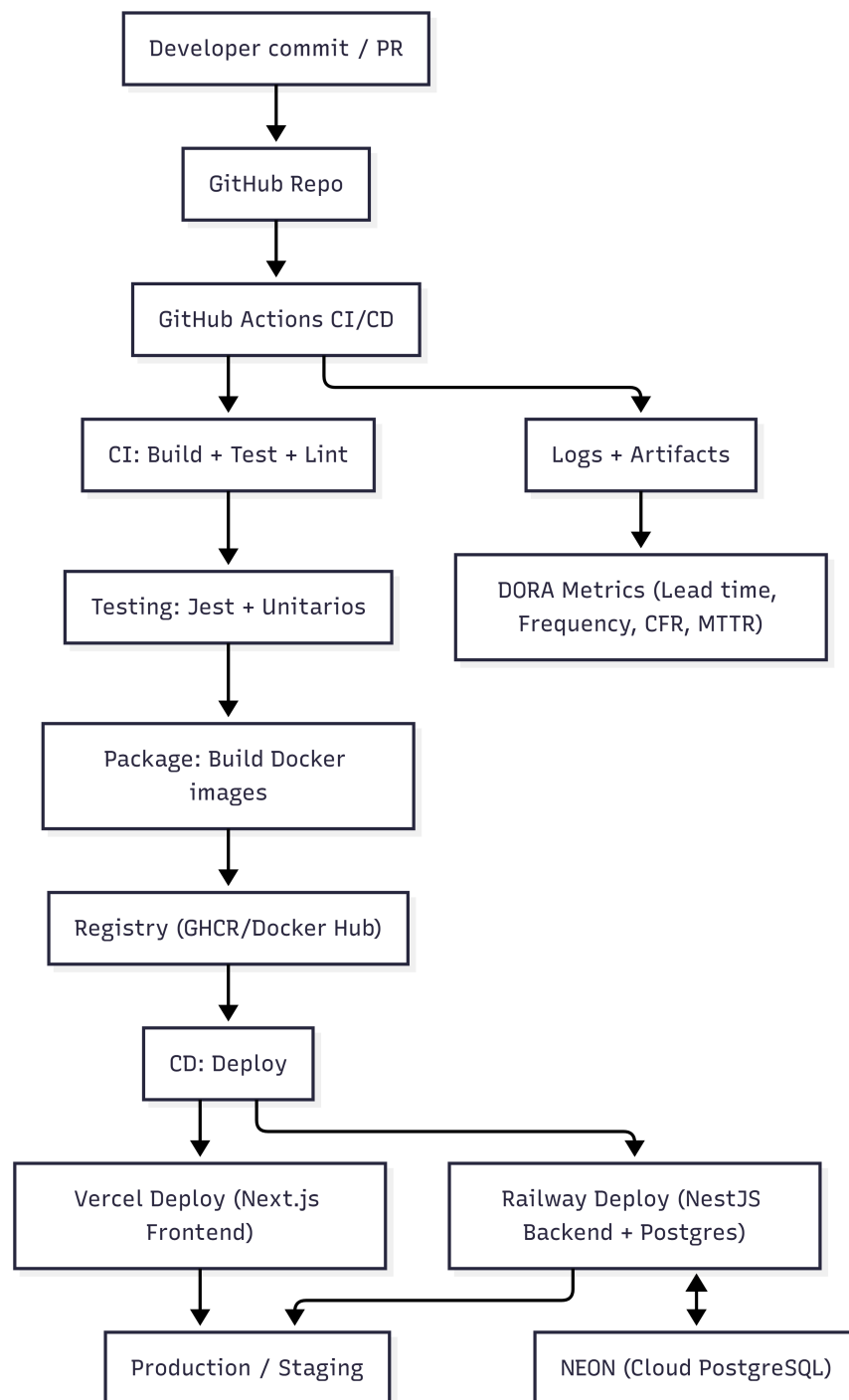


Figura 5. Flujo general del pipeline CI/CD automatizado.

Una vez completadas estas validaciones, el pipeline procede a la fase de empaquetado de la aplicación mediante la construcción de imágenes Docker. Estas imágenes son posteriormente publicadas en un registro de contenedores, como GitHub Container Registry (GHCR) o Docker Hub, permitiendo su almacenamiento y reutilización en entornos de despliegue.

Finalmente, durante la fase de entrega continua (CD), las imágenes generadas son utilizadas para desplegar automáticamente la aplicación en las plataformas de nube utilizadas en el proyecto. En este caso, el frontend desarrollado con Next.js es desplegado en la plataforma Vercel, mientras que el backend implementado con NestJS y la base de datos PostgreSQL se despliegan en la plataforma Railway.

Este proceso automatizado permite garantizar que cada cambio incorporado al repositorio sea validado, construido y desplegado de manera consistente, reduciendo errores humanos y facilitando la trazabilidad del proceso de entrega continua del sistema.

Publicación de artefactos del sistema

La imagen Docker generada durante la ejecución del pipeline se publica automáticamente en GitHub Container Registry (GHCR). Este registro funciona como un repositorio centralizado donde se almacenan las distintas versiones del backend generadas durante el proceso de integración continua.

El uso de un registro de contenedores permite versionar los artefactos generados por el pipeline y facilita su posterior despliegue en plataformas de infraestructura en la nube.

La Figura 6 muestra el repositorio de imágenes Docker del backend almacenadas en GitHub Container Registry (GHCR), el cual funciona como el registro de contenedores utilizado dentro del pipeline CI/CD del proyecto.

Como se observa en la Figura 6, cada imagen almacenada en el registro corresponde a una versión específica del backend generada automáticamente durante la ejecución del pipeline

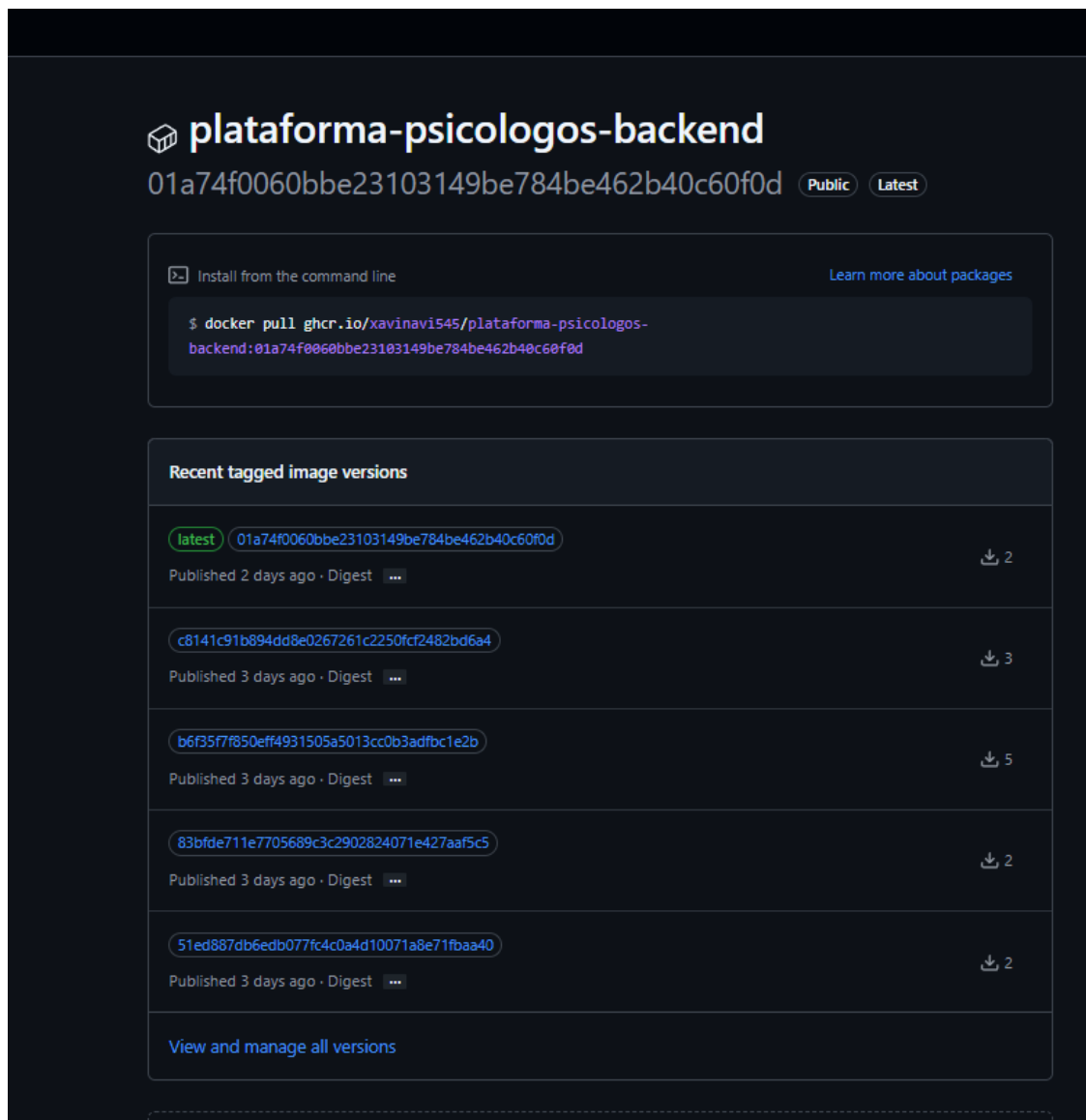


Figura 6. Repositorio de imágenes Docker del backend publicadas automáticamente en GitHub Container Registry (GHCR).

CI/CD. Estas imágenes son construidas a partir del código fuente del proyecto y posteriormente publicadas en el registro de contenedores mediante workflows configurados en GitHub Actions.

El uso de un registro de contenedores permite almacenar y versionar las imágenes Docker generadas durante el proceso de integración continua, facilitando su reutilización en etapas posteriores del despliegue. De esta manera, las imágenes publicadas en GHCR pueden ser utilizadas por las plataformas de infraestructura en la nube para desplegar automáticamente nuevas versiones del sistema.

En el contexto de esta investigación, el registro de contenedores constituye un componente fundamental dentro del pipeline CI/CD, ya que permite mantener un historial versionado de las imágenes del backend y garantizar que el proceso de despliegue utilice artefactos generados y validados previamente por el pipeline automatizado.

Despliegue automatizado del sistema

El despliegue automatizado del sistema se configuró utilizando dos plataformas de infraestructura en la nube: Vercel para el frontend y Railway para el backend.

En el caso del frontend, Vercel se integra directamente con el repositorio de GitHub, lo que permite generar automáticamente un nuevo despliegue cada vez que se detectan cambios en el código fuente.

Para el backend, Railway fue configurado para consumir directamente las imágenes Docker publicadas en GitHub Container Registry. De esta manera, cada nueva versión generada por el pipeline puede ser utilizada para actualizar automáticamente el servicio desplegado.

Como se observa en la Figura 7, el backend del sistema es desplegado en la plataforma Railway utilizando una imagen Docker previamente generada por el pipeline CI/CD. Railway permite ejecutar aplicaciones contenerizadas en un entorno de infraestructura en la nube, facilitando la actualización automática del servicio cuando se publican nuevas versiones de la imagen.

El proceso de despliegue consiste en la obtención de la imagen Docker almacenada en GitHub Container Registry y su ejecución dentro de la infraestructura proporcionada por Railway. De esta manera, cada nueva versión del backend puede ser desplegada automáticamente sin intervención manual del desarrollador.

Esta integración entre el pipeline CI/CD y la plataforma de despliegue permite mantener actualizado el servicio desplegado y garantiza que las versiones ejecutadas en producción correspondan a artefactos generados y validados previamente durante el proceso de integración continua.

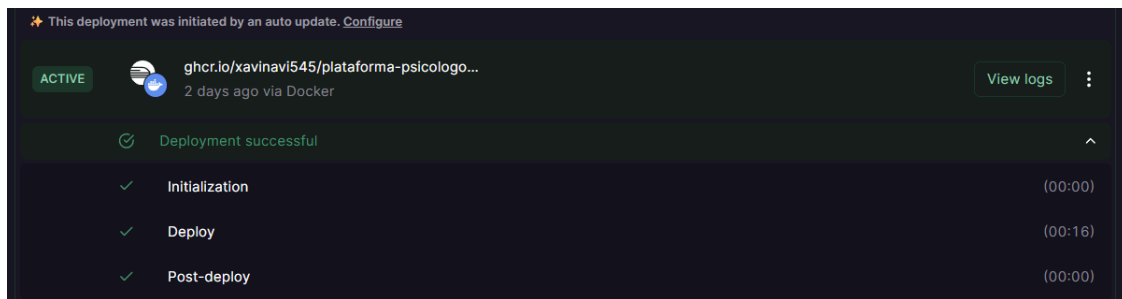


Figura 7. Despliegue automatizado del backend en Railway a partir de una imagen Docker publicada en GHCR.

Este proceso permite automatizar completamente el ciclo de construcción y despliegue del sistema, reduciendo la intervención manual y garantizando que cada versión del software sea desplegada a partir de artefactos generados automáticamente por el pipeline CI/CD.

Resultados

Este capítulo presenta los resultados obtenidos durante la implementación y evaluación del pipeline de integración y entrega continua (CI/CD) desarrollado en el proyecto. Los resultados incluyen el análisis de las ejecuciones del pipeline en GitHub Actions, el comportamiento operativo del frontend desplegado en la plataforma Vercel y la interpretación de estos resultados mediante métricas DevOps basadas en el modelo DORA. La finalidad de este análisis es evidenciar el funcionamiento del sistema automatizado de integración, validación y despliegue implementado durante la investigación.

Resultados de la ejecución del pipeline CI/CD

Durante el desarrollo del proyecto se implementó un pipeline de integración y entrega continua (CI/CD) para el backend de la plataforma web utilizando GitHub Actions. El pipeline fue configurado para ejecutarse automáticamente cuando se integran cambios en la rama principal del repositorio del proyecto.

Cada ejecución del pipeline realiza una serie de etapas destinadas a validar y preparar el sistema para su despliegue. Entre las principales tareas ejecutadas se encuentran la instalación de dependencias del proyecto, la validación del estilo del código mediante ESLint, la ejecución de pruebas unitarias utilizando el framework Jest y la construcción de la imagen Docker del backend.

Con el fin de analizar el comportamiento del pipeline durante su operación, se recopilieron registros de ejecución generados automáticamente por GitHub Actions. Estos registros incluyen información sobre el estado de cada ejecución, la rama del repositorio asociada y la duración

total del proceso.

Resultados del despliegue del frontend en Vercel

Una vez estabilizado el pipeline CI/CD, el frontend de la plataforma web fue desplegado en la plataforma de infraestructura en la nube Vercel. Esta plataforma permite alojar aplicaciones web desarrolladas con frameworks modernos como Next.js y proporciona herramientas de monitoreo para observar el comportamiento operativo del sistema desplegado.

El despliegue del frontend se realizó mediante la integración directa entre el repositorio del proyecto en GitHub y la infraestructura de Vercel. Bajo esta configuración, cada cambio validado en la rama principal del repositorio genera automáticamente una nueva versión desplegada de la aplicación.

Con el objetivo de analizar el comportamiento del sistema en producción, se observaron las métricas operativas proporcionadas por el panel de monitoreo de Vercel durante un intervalo de observación de 12 horas. Este panel reporta información relacionada con solicitudes atendidas por el sistema, transferencia de datos, ejecución de funciones serverless y consumo de recursos computacionales asociados al funcionamiento del frontend desplegado.

La Figura 8 muestra una captura del panel de métricas del frontend desplegado en Vercel durante el intervalo de observación analizado. En esta interfaz se presentan indicadores tales como solicitudes procesadas por el sistema (*Edge Requests*), transferencia de datos (*Fast Data Transfer*), métricas de ejecución de funciones (*Vercel Functions*) y consumo de CPU asociado a la ejecución de las funciones del sistema.

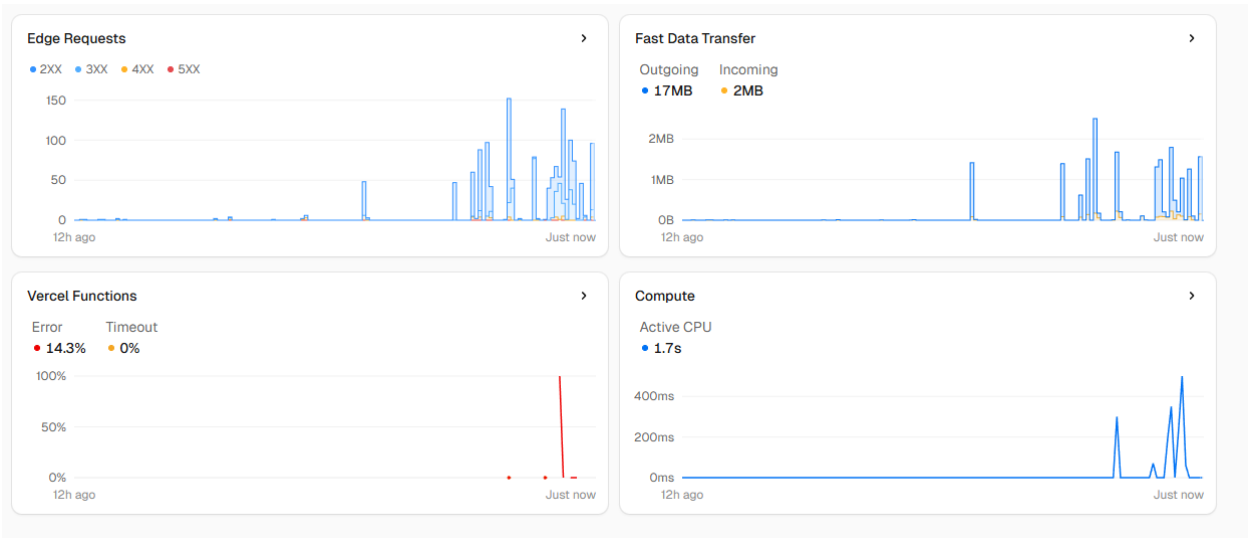


Figura 8. Panel de métricas operativas del frontend desplegado en Vercel durante un intervalo de observación de 12 horas.

A partir de la información mostrada en el panel de monitoreo se extrajeron los valores numéricos más representativos del comportamiento del sistema durante el intervalo analizado. Estos valores corresponden a métricas reportadas directamente por la plataforma Vercel y se resumen en la Tabla 1.

Tabla 1. Resumen de métricas operativas observadas en el frontend desplegado en Vercel durante el intervalo de observación de 12 horas

Métrica	Valor observado
Transferencia de datos saliente (Outgoing)	17 MB
Transferencia de datos entrante (Incoming)	2 MB
Errores en funciones	14.3 %
Timeouts de funciones	0 %
Tiempo activo de CPU	1.7 s

La Tabla 1 resume los principales indicadores operativos observados en el sistema durante el intervalo de análisis. La transferencia de datos corresponde al volumen de información intercambiada entre los usuarios y la aplicación desplegada. Por su parte, el indicador de errores en funciones reporta el porcentaje de ejecuciones de funciones serverless que finalizaron con

error durante el periodo de observación. El indicador de *timeout* representa el porcentaje de ejecuciones de funciones que excedieron el tiempo máximo de ejecución permitido por la plataforma.

Cálculo de métricas de ejecución del pipeline

Para analizar el comportamiento del pipeline se calcularon métricas descriptivas a partir de los registros de ejecución observados en GitHub Actions. En particular, se calculó la duración promedio de las ejecuciones utilizando la media aritmética.

La media de duración de las ejecuciones se calcula mediante la expresión:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.1)$$

donde x_i representa la duración de cada ejecución registrada y n corresponde al número total de ejecuciones analizadas.

A partir de los registros observados ($n = 10$ ejecuciones), se obtuvo una duración promedio de aproximadamente 72,5 segundos para las ejecuciones exitosas del pipeline.

Asimismo, la tasa de fallos del pipeline se estimó utilizando la proporción de ejecuciones fallidas respecto al total de ejecuciones observadas:

$$FailureRate = \frac{N_{fail}}{N_{total}} \times 100 \quad (3.2)$$

donde N_{fail} representa el número de ejecuciones fallidas y N_{total} el número total de ejecuciones registradas.

En el conjunto de ejecuciones analizadas se observaron $N_{fail} = 2$ ejecuciones fallidas de un total de $N_{total} = 10$, lo que corresponde a una tasa de fallo del 20%.

Con el fin de facilitar la comparación visual entre estos indicadores, se construyó una

representación gráfica utilizando los valores numéricos obtenidos del panel de monitoreo. La Figura 9 muestra una gráfica de barras que resume los valores reportados por la plataforma Vercel para cada uno de los indicadores observados.

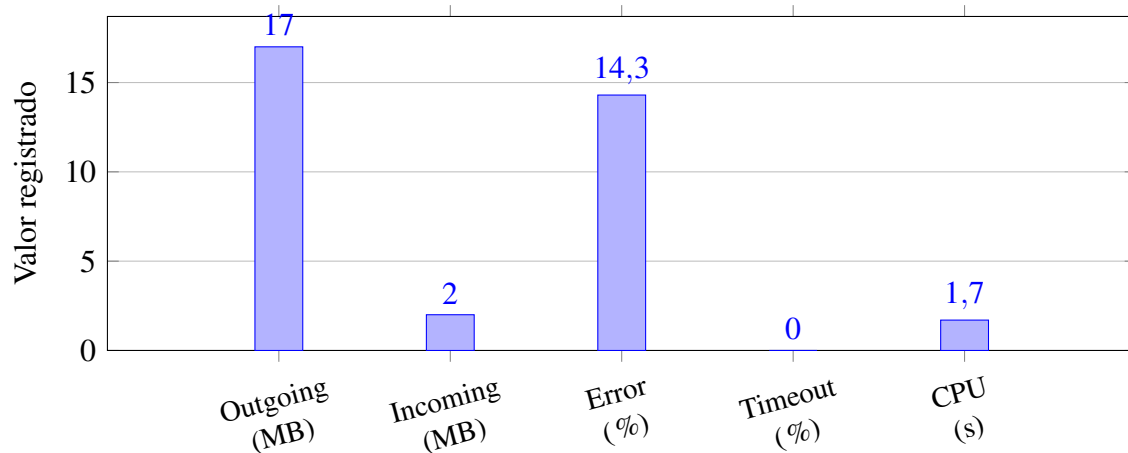


Figura 9. Representación gráfica de los indicadores operativos del frontend desplegado en Vercel durante el intervalo de observación analizado.

La gráfica presentada en la Figura 9 permite visualizar de manera comparativa los indicadores observados durante el periodo de análisis. Durante la ventana de observación no se registraron eventos de *timeout* en la ejecución de funciones del sistema. Asimismo, los valores reportados de transferencia de datos y consumo de CPU corresponden al comportamiento operativo del frontend durante el intervalo analizado.

Síntesis de resultados y relación con métricas DORA

Para interpretar los resultados obtenidos se utilizó como referencia el modelo de métricas DORA (DevOps Research and Assessment), ampliamente empleado para evaluar el desempeño de procesos de integración y entrega continua en entornos de desarrollo de software (DevOps Research and Assessment, 2021, 2025). En el contexto de esta investigación, las métricas fueron aproximadas a partir de los registros observados durante la ejecución del pipeline CI/CD.

La frecuencia de despliegue se interpretó a partir del número de ejecuciones del pipeline

registradas en GitHub Actions. El *lead time for changes* se aproximó mediante la duración total de las ejecuciones exitosas del pipeline, mientras que la tasa de fallos por cambio (*change failure rate*) se estimó considerando la proporción de ejecuciones fallidas respecto al total de ejecuciones observadas. Debido al alcance del estudio, el tiempo medio de recuperación (*mean time to recovery*) no fue medido de forma directa, aunque el pipeline permite generar nuevas ejecuciones automáticamente tras la corrección del código.

El *Lead Time for Changes* se aproximó a partir de la duración total de las ejecuciones exitosas del pipeline, utilizando el promedio de las duraciones registradas en GitHub Actions.

$$LeadTime = \frac{\sum_{i=1}^n t_i}{n} \quad (3.3)$$

donde t_i representa la duración de cada ejecución exitosa del pipeline y n corresponde al número total de ejecuciones exitosas observadas.

A partir de estos registros fue posible identificar valores representativos de las métricas DORA consideradas en esta investigación. La Tabla 2 presenta un resumen de los indicadores observados durante el periodo de análisis.

Tabla 2. Resumen de métricas del proceso de integración y despliegue continuo observadas en el sistema

Métrica	Valor observado
Lead Time for Changes	Entre 60 y 80 segundos por ejecución del pipeline
Deployment Frequency	Despliegues automáticos generados por cada cambio integrado en la rama principal
Change Failure Rate	20 % de ejecuciones fallidas en el conjunto de ejecuciones analizadas
Mean Time to Recovery (MTTR)	Recuperación mediante nueva ejecución del pipeline tras corrección del error

El *Lead Time for Changes* corresponde al tiempo transcurrido entre la integración de un cambio en el repositorio y la finalización del proceso de validación y construcción del sistema dentro del

pipeline CI/CD. Este valor se estimó a partir de las duraciones observadas en las ejecuciones exitosas del pipeline registradas en GitHub Actions.

La *Deployment Frequency* se relaciona con la frecuencia con la que el sistema puede generar nuevas versiones desplegables. En el contexto del pipeline implementado, cada cambio integrado en la rama principal del repositorio puede generar automáticamente una nueva ejecución del proceso de integración continua.

Por su parte, el *Change Failure Rate* fue estimado a partir de la proporción de ejecuciones fallidas del pipeline respecto al total de ejecuciones observadas durante el periodo de análisis. Este indicador permite identificar la estabilidad del proceso de validación automática del sistema.

Finalmente, el indicador *Mean Time to Recovery (MTTR)* describe el tiempo necesario para restaurar el funcionamiento del sistema tras la detección de un error en el pipeline. En este caso, la recuperación se realiza mediante la corrección del código y la ejecución de una nueva integración continua.

En conjunto, los resultados presentados en este capítulo evidencian el funcionamiento del pipeline CI/CD implementado en el proyecto y permiten observar el comportamiento del proceso automatizado de integración, validación y despliegue del sistema dentro de un entorno real de desarrollo. La automatización de estas tareas permitió estandarizar el flujo de integración continua, reducir la intervención manual durante el proceso de despliegue y generar evidencia observable del funcionamiento del pipeline mediante registros de ejecución y métricas operativas del sistema.

Discusión

En este capítulo, se exploran y se dan significado a los resultados conseguidos al poner en marcha el proceso automatizado de CI/CD y al lanzar la plataforma web en entornos reales de nube. A diferencia del capítulo anterior sobre los resultados, aquí se adopta una perspectiva más interpretativa y crítica, con el objetivo de entender el alcance de los hallazgos en relación con la problemática planteada, los objetivos definidos, las preguntas de investigación y las hipótesis propuestas.

Se combinan los resultados técnicos del sistema con los aspectos metodológicos descritos en el capítulo de Método. En este sentido, los resultados no se consideran únicamente como datos operativos, sino como evidencia concreta que permite evaluar cómo la automatización del ciclo de vida del software mediante prácticas DevOps influye en el proceso de desarrollo y despliegue. Esta visión general ayuda a comprender mejor el impacto de las mejoras observadas en el sistema.

Evaluación del logro del objetivo general

El propósito principal del estudio fue diseñar e implementar un pipeline CI/CD automatizado utilizando Docker y GitHub Actions, con el fin de optimizar los procesos de construcción, pruebas y despliegue de una plataforma web orientada a la gestión de servicios psicológicos. Con base en los resultados obtenidos, se puede decir que este objetivo se cumplió de manera satisfactoria.

La integración de herramientas como GitHub Actions, GitHub Container Registry, Vercel y Railway demuestra que el enfoque planteado es viable desde el punto de vista técnico. También

evidencia que la automatización reduce errores derivados de configuraciones manuales y mejora la consistencia del sistema en diferentes entornos.

Análisis del logro de los objetivos específicos

Respecto al primer objetivo específico, relacionado con el análisis del flujo inicial de desarrollo, se observó que la falta de automatización generaba inconsistencias, errores de configuración y una fuerte dependencia de acciones manuales. Durante las primeras ejecuciones del pipeline se identificaron problemas asociados a dependencias incorrectas, errores de estilo y configuraciones inadecuadas.

En cuanto al segundo objetivo, enfocado en el diseño del pipeline CI/CD, los resultados muestran que fue posible integrar procesos automatizados de validación de código, pruebas unitarias y construcción de imágenes Docker. Esto permitió mejorar la calidad del software de manera progresiva.

Finalmente, el análisis de métricas operativas permitió obtener datos concretos sobre el comportamiento del sistema, lo cual sirvió como base para evaluar el impacto de la automatización implementada.

Relación con las preguntas de investigación

Los resultados obtenidos permiten responder a las preguntas de investigación planteadas. En relación con la reducción del tiempo de entrega de cambios, se evidenció que el tiempo entre la modificación del código y su despliegue en producción disminuyó en comparación con el proceso manual.

En cuanto a la estabilidad del sistema, no se detectaron fallos críticos persistentes tras los despliegues automatizados. La integración de herramientas como ESLint y Jest permitió detectar

errores en etapas tempranas.

Análisis de métricas DORA y contraste

Las métricas DORA se utilizaron como referencia para evaluar el desempeño del pipeline CI/CD implementado. En relación con la hipótesis H1, los resultados muestran una reducción clara del tiempo entre la integración del código y su despliegue.

Sin embargo, en el backend se identificaron retrasos asociados a la infraestructura de Railway, lo que demuestra que no todo depende del pipeline.

En cuanto a la hipótesis H2, se evidenció un aumento en la frecuencia de despliegue sin afectar negativamente la estabilidad del sistema. Cada cambio activaba automáticamente un nuevo proceso de validación y despliegue.

Respecto a la hipótesis H3, la incorporación de validaciones automáticas permitió detectar errores de manera temprana, reduciendo la probabilidad de fallos en producción.

A continuación, se presenta una comparación conceptual entre el proceso manual y el automatizado:

Tabla 3. Comparación conceptual de métricas DORA antes y después de la implementación del pipeline CI/CD

Métrica	Proceso sin automatización	Proceso con pipeline CI/CD
Frecuencia de despliegue	Despliegues manuales y esporádicos dependientes del desarrollador	Despliegues automáticos generados tras cada integración en la rama principal
Lead time for changes	Dependiente de procesos manuales de construcción y despliegue	Aproximadamente 60–80 segundos para la ejecución del pipeline de integración
Change failure rate	Mayor probabilidad de errores detectados tardíamente	Errores detectados durante las etapas iniciales de validación del pipeline
Mean time to recovery (MTTR)	Corrección manual tras detectar errores en el sistema	Recuperación mediante corrección del código y nueva ejecución automática del pipeline

Esta tabla permite visualizar de forma general las mejoras observadas. Aunque no son valores exactos, ayudan a entender el cambio en el proceso de desarrollo.

Comparación con estudios previos

Los resultados coinciden con investigaciones recientes que indican que la automatización mejora la estabilidad del software y reduce errores. También se relacionan con informes que destacan mejoras en la frecuencia de despliegue y tiempos de entrega al adoptar prácticas DevOps.

Fortalezas y limitaciones

Una fortaleza importante es el uso de un sistema real desplegado en la nube, lo que permitió evaluar el pipeline en condiciones cercanas a la práctica profesional.

También destaca el uso de herramientas ampliamente utilizadas en la industria, lo que facilita la replicación del estudio.

No obstante, el estudio presenta limitaciones. Se basa en un solo caso. Además, las métricas se

analizaron en periodos de tiempo relativamente cortos.

Implicaciones

Los resultados muestran que implementar CI/CD es viable incluso en proyectos pequeños o académicos. La automatización mejora la eficiencia y reduce errores humanos.

Desde el punto de vista académico, este trabajo sirve como referencia para futuras investigaciones en ingeniería de software.

Recomendaciones

Se recomienda ampliar el estudio a más proyectos para obtener resultados comparativos. También sería útil realizar análisis en periodos más largos. Además, se sugiere integrar herramientas de monitoreo más avanzadas para mejorar la observabilidad del sistema.

Síntesis

En este capítulo se interpretaron los resultados obtenidos tras la implementación del pipeline de CI/CD. Se evidenció que la automatización mejora la eficiencia, la estabilidad y la calidad del proceso de desarrollo.

El análisis permitió establecer una base para las conclusiones finales del estudio.

Conclusiones

El objetivo principal de este proyecto era para crear y puesto a operación un automatizado CI/CD que utiliza Docker y GitHub Acciones para realizar el edificio, testaje, y procesos de despliegue de una plataforma de web diseñada para servicios psicológicos gestores. A través del desarrollo del sistema, implementación de pipeline, y análisis de los resultados, las conclusiones siguientes han sido dibujadas.

Al principio, notamos que al usar un sistema automatizado de integración y entrega continua (CI/CD), se logra acelerar el tiempo que se tarda en implementar cambios en el software en comparación con hacerlo manualmente. Automatizar tareas como validar, ejecutar pruebas, revisar estilos, construir componentes y preparar despliegues agilizó el proceso completo, desde unir el código hasta tener una versión lista para ser desplegada. Específicamente, las pruebas exitosas realizadas en GitHub Actions mostraron tiempos promedio de ejecución de alrededor de 60 a 80 segundos, respaldando la hipótesis H1 propuesta en el estudio.

Además, se observó que aumentar la frecuencia de implementación no tuvo un impacto negativo en la estabilidad del sistema durante los períodos analizados. Gracias al uso de integración continua y validaciones automáticas, pudimos hacer despliegues más seguido sin errores importantes en la operación del sistema.

Como resultado, nuestros descubrimientos respaldan la idea de que es factible aumentar la frecuencia de actualizaciones del software sin comprometer la estabilidad del sistema, lo cual confirma nuestra hipótesis H2.

La introducción del pipeline de CI/CD ha ayudado a que el proceso de desarrollo sea más eficiente al disminuir la necesidad de realizar tareas manuales y al estandarizar la forma de

trabajar. Además, al utilizar herramientas como ESLint para revisar el código estático y Jest para realizar pruebas unitarias, se pueden detectar errores rápidamente durante la integración continua. Esto mejora la confiabilidad del software antes de ser lanzado en la nube.

La incorporación de contenedores Docker ha ayudado a mantener la coherencia del sistema al estandarizar el entorno de ejecución del backend. Esto ha evitado problemas causados por configuraciones diferentes entre los entornos de desarrollo y producción.

Aunque en este estudio no se pudo confirmar por completo la hipótesis H3 con pruebas cuantitativas directas, la organización del proceso y el uso de contenedores sientan una buena base para futuras integraciones de herramientas de análisis de vulnerabilidades en las imágenes de Docker.

Durante la evaluación, también descubrimos que algunos factores externos, como la infraestructura de despliegue, pueden afectar los tiempos de actualización del sistema en producción. En concreto, notamos que el backend en Railway a veces tardaba hasta dos horas en actualizar a la última imagen Docker disponible en el registro de contenedores.

Por último, el trabajo presenta un ejemplo que demuestra que es factible utilizar métodos contemporáneos de creación de software en proyectos académicos o pequeños.

Referencias Bibliográficas

- Amaro, R., et al. (2024). DevOps Metrics and KPIs: A Multivocal Literature Review. *ACM Computing Surveys*. <https://doi.org/10.1145/3652508>
- Barbosa, D. (2026). Highly Efficient Software Development Using DevOps and Cloud Technologies. *Future Internet*. <https://doi.org/10.3390/fi18010050>
- Boles, C., Hyatt, L., & Thomas, J. (2024). *Grype vs. Trivy: A Comparative Analysis of Docker Image Vulnerability Scanners* (inf. téc.). Montana State University. https://www.montana.edu/cyber/products/Grype_Vs_Trivy_Boles_et_al.pdf
- Ccallo-Luque, M., & Quispe-Quispe, A. (2024). Adoption and Adaptation of CI/CD Practices in Very Small Software Development Entities. *arXiv*. <https://arxiv.org/abs/2410.00623>
- DevOps Research and Assessment. (2021). *2021 Accelerate State of DevOps Report*. <https://dora.dev/research/2021/dora-report/2021-dora-accelerate-state-of-devops-report.pdf>
- DevOps Research and Assessment. (2025). *DORA Metrics - Four Keys*. <https://dora.dev/guides/dora-metrics-four-keys/>
- Dhandapani, S. (2025). Enhancing Software Supply Chain Security Through STRIDE-Based Threat Modelling of CI/CD Pipelines. *arXiv*. <https://arxiv.org/abs/2506.06478>

- Farihane, R., et al. (2025). CI/CD Pipeline Optimization Using Artificial Intelligence: A Systematic Mapping Study. *Engineering Proceedings*.
<https://www.mdpi.com/2673-4591/112/1/32>
- Kumar, P., et al. (2024). Containerization in Multi-Cloud Environments: Challenges and Opportunities. *arXiv*. <https://arxiv.org/abs/2403.12980>
- Pan, Z., Shen, W., Wang, X., et al. (2024). Understanding Security Threats in Open-Source Software CI/CD Pipelines. *arXiv*. <https://arxiv.org/abs/2401.17606>
- Sánchez Ruiz, J. M., & Domínguez Mayo, F. J. (2023). A Benchmarking Proposal for DevOps Practices on Open Source Software Projects. *arXiv*. <https://arxiv.org/abs/2304.14790>
- Selinder, E. (2024). DevOps KPIs and Software Delivery Performance. *KTH Royal Institute of Technology*. <https://www.diva-portal.org/smash/get/diva2:1936481/FULLTEXT01.pdf>

Lista de Apéndices

- Apéndice A: Guía oficial de métricas DORA

Apéndice A: Guía oficial de métricas DORA

En este apéndice se presenta un material documental de referencia utilizado como soporte conceptual para la investigación. El contenido corresponde a una guía oficial sobre métricas DORA (*DevOps Research and Assessment*), publicada en formato PDF por la empresa Appfire.

El documento describe de manera estructurada las métricas fundamentales empleadas para evaluar el desempeño de los procesos de entrega continua de software en equipos DevOps. Entre las métricas abordadas se incluyen la frecuencia de despliegue, el *lead time for changes*, la tasa de fallos de cambios y el tiempo medio de recuperación (*mean time to recovery*), las cuales constituyen el marco de referencia utilizado en el análisis del presente estudio.

Nota. Documento de referencia consultado para la definición conceptual y metodológica de las métricas DORA utilizadas en la investigación. Fuente: [Appfire](#) (2023).



Guía oficial de métricas DORA

DORA metrics

The proven framework for high performing