Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Desarrollo de Solución Automatizada para Despliegue de Microservicios en Ambientes de Prueba en Empresa de Telecomunicaciones

Proyecto de Titulación

Previo la obtención del Título de:

Magíster en Sistema de Información Gerencial

Presentado por:
Ing. John Jairo Flores Rodríguez
Ing. Kleber Ronald Lino Sánchez

Guayaquil - Ecuador
Año: 2025

Dedicatoria

A mi esposa, por ser mi compañera fiel en este camino. Gracias por tu amor, tu paciencia y tu apoyo incondicional que ha sido clave para alcanzar esta meta.

A mi hijo, mi mayor motivación. Este logro es para ti, con el anhelo de dejarte un ejemplo de vida, esfuerzo y constancia. Que veas que los sueños se pueden alcanzar con disciplina y dedicación, y que cada sacrificio tiene su recompensa.

A mi madre, por enseñarme con su ejemplo los valores que hoy me definen. Gracias por tu amor, tu fortaleza y tu apoyo constante en cada etapa de mi vida.

Ing. John Jairo Flores R.

Dedicatoria

A mis padres, por su amor incondicional, por confiar en mí incluso en los momentos más difíciles y por enseñarme, con su ejemplo, el valor del esfuerzo, la responsabilidad y la humildad. Gracias por ser mi guía y mi fortaleza en cada paso de este camino.

A mi familia en general, por su apoyo constante, por las palabras de aliento en los momentos de cansancio y por celebrar cada pequeño avance como un logro compartido. Este trabajo es también resultado de todo lo que me han dado, comprensión, paciencia y motivación para seguir superándome cada día.

Ing. Kléber Lino Sánchez

Agradecimientos

En primer lugar, agradezco profundamente a Dios por colmar mi vida de bendiciones, por darme salud y fortaleza, y por brindarme la oportunidad de culminar esta etapa importante de mi formación académica.

A mi familia, mi pilar incondicional, gracias por su apoyo constante, su paciencia y su comprensión, los cuales fueron fundamentales para alcanzar este logro.

Ing. John Jairo Flores R.

Agradecimientos

Mi más profundo agradecimiento a Dios, por darme la fuerza, la claridad y la perseverancia para seguir adelante en cada etapa de este proceso. Sin Su guía constante, este logro no habría sido posible.

A mi familia, y en especial a mis padres, por estar siempre ahí, apoyándome sin condiciones, creyendo en mí incluso cuando yo mismo dudaba. Gracias por su paciencia, por motivarme a dar lo mejor y por acompañarme en todo momento.

Ing. Kléber Lino Sánchez

Declaración Expresa

Yo/Nosotros John Jairo Flores Rodríguez y Kleber Ronald Lino Sánchez acuerdo/acordamos y reconozco/reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por mí/nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que me/nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de mi/nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique al/los autor/es que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 07 de Julio del 2025.

John Jairo Flores Rodriguez
Time Stamping Security Data

Ing. John Jairo Flores Rodriguez

Kleber Ronald Lino Sanchez
Time Stamping Security Data

Ing. Kleber Ronald Sanchez Lino

Evaluadores

Firmado electrónicamente por:
JUAN CARLOS GARCIA PLUA
Validar únicamente con FirmaEC

Mgtr. Juan Carlos García

Tutor de proyecto

Firmado electrónicamente por:
LENIN EDUARDO FREIRE COBO
Validar únicamente con FirmaEC

Mgtr. Lenin Freire Cobo

Revisor de proyecto

Resumen

Este trabajo presenta el diseño e implementación de una solución automatizada para optimizar el proceso de despliegue de ambientes de prueba, con el objetivo de reducir el tiempo de ejecución, minimizar la carga operativa del equipo de QA y mejorar la eficiencia técnica frente a la alta demanda de proyectos simultáneos. Se justifica su desarrollo debido a los prolongados tiempos y altos niveles de intervención manual requeridos en el proceso actual.

Para el desarrollo del proyecto, se emplearon herramientas como N8N, Redis, GitLab y Telegram, integradas en una arquitectura distribuida. Se configuró un flujo automatizado que permitió ejecutar despliegues verticales de microservicios mediante comandos estructurados.

Los resultados mostraron una reducción significativa en los tiempos de despliegue de 2 a 3 días a menos de una hora en proyectos de alta complejidad. Además, se evidenció una disminución en la carga operativa y en la frecuencia de errores.

Se concluye que la solución propuesta cumple con los objetivos de eficiencia, escalabilidad y autonomía técnica del equipo de QA y despliegue. El enfoque puede adaptarse a distintos tamaños de proyecto, permitiendo mayor control y rapidez en la preparación de ambientes de prueba.

Palabras clave: Automatización, Despliegue, QA, Microservicios, Optimización.

Abstract

This project addresses the challenge of optimizing the deployment process of testing environments in a telecommunications company. The current process involves deploying around 80 microservices and can take up to three days, resulting in a high operational load for QA and infrastructure teams. To tackle this issue, a distributed architecture was designed using tools such as N8N, GitLab, Redis, and Telegram. This solution allows automated deployment flows to be triggered through structured messages, enabling dynamic configuration generation and execution of GitLab pipelines.

As a result, the average deployment time was reduced from several days to just a few hours, depending on project complexity. The system currently supports vertical deployment only, following a sequential logic based on functional dependencies. Initial tests show stable performance and high potential for scalable implementation.

It is concluded that the proposed automated architecture improves deployment efficiency, reduces human errors, and enhances QA team autonomy in managing test environments.

Keywords: Test environments, automation, deployment optimization, microservices, CI/CD.

Índice general

Abreviaturas

CI	Integración Continua (Continuous Integration)
CD	Despliegue Continuo (Continuous Deployment)
IaC	Infraestructura como Código (Infrastructure as Code)
QA	Aseguramiento de la Calidad (Quality Assurance)
API	Interfaz de Programación de Aplicaciones (Application Programming Interface)
N8N	Herramienta de automatización de flujos sin código (No-code Workflow Automation Tool)
YAML	YAML Ain't Markup Language (Formato de serialización de datos usado en archivos de configuración)
TIC	Tecnologías de la Información y Comunicación
MS	Microservicio
SDLC	Ciclo de Vida del Software (Software Development Life Cycle)
TDD	Desarrollo guiado por pruebas (Test-Driven Development)
Redis	Repositorio de estructuras de datos clave-valor en memoria (Remote Dictionary Server)
GitLab	Plataforma de DevOps para repositorio de código, integración y despliegue

Simbología

min	Minutos
h	Horas
%	Porcentaje
Nº	Número de Microservicios

Índice de figuras

Índice de tablas

CAPÍTULO 1

1. Introducción

En las empresas de telecomunicaciones con alta carga operativa, los procesos manuales para la preparación de ambientes de prueba representan una limitación significativa para la validación oportuna de aplicaciones. Este problema se acentúa en arquitecturas basadas en microservicios, donde cada ambiente requiere el despliegue coordinado de decenas de componentes, generando retrasos operativos y sobrecarga para los equipos técnicos.

Actualmente, la ausencia de automatización y de mecanismos de despliegue selectivo obliga a los ingenieros a levantar todos los microservicios, incluso cuando no han sido modificados. Esta práctica incrementa los tiempos de espera y afecta la productividad del área de QA, dificultando la entrega ágil de proyectos y elevando los riesgos operativos y contractuales.

La presente investigación propone el desarrollo de una solución automatizada que permita gestionar el despliegue de microservicios en ambientes de prueba de forma eficiente y bajo demanda. La propuesta integra herramientas como N8N, GitLab CI/CD, Redis y Telegram, con el objetivo de reducir los tiempos de preparación de ambientes de días a minutos(min), optimizar el uso de recursos técnicos y mejorar la capacidad de respuesta del área de calidad de software.

1.1. Descripción del Problema

El presente trabajo se desarrolla en una empresa grande del sector de telecomunicaciones, específicamente en el departamento de Calidad de Software (QA), donde se requiere validar aplicaciones complejas compuestas por múltiples microservicios. Los involucrados en esta problemática son principalmente los ingenieros de despliegue, un equipo reducido de solo tres personas que deben atender múltiples solicitudes de levantamiento de ambientes de prueba para los proyectos que la empresa desarrolla.

El problema radica en la alta demanda de ambientes de prueba frente a la capacidad limitada del equipo de despliegue, ya que, para cada validación funcional, los QA necesitan que se levanten más de 35 microservicios por ambiente, lo que toma entre 2 a 3 días hábiles.

Las causas principales de este problema están relacionadas a una arquitectura poco flexible que obliga a desplegar todos los microservicios incluso si no han sido modificados, así como la ausencia de procesos automatizados que permitan realizar despliegues selectivos o bajo demanda.

Las consecuencias derivadas de esta situación incluyen retrasos en la entrega de proyectos, sanciones contractuales, insatisfacción de usuarios internos y aumento de costos por la necesidad de ampliar el equipo

técnico. Para resolver el problema, se propone evaluar una arquitectura automatizada que permita gestionar el despliegue de ambientes de prueba de forma más ágil y eficiente, integrando herramientas tecnológicas (N8N,

Redis, GitLab CI/CD y Telegram,) de modo que los QA puedan solicitar ambientes listos y completos mediante la automatización, reduciendo los tiempos de espera de días a minutos y optimizando los recursos disponibles.

1.2 Justificación del Problema

En las empresas de telecomunicaciones con alta carga operativa, el proceso de preparación de ambientes de prueba representa un cuello de botella crítico que afecta directamente la calidad del software y el cumplimiento de los cronogramas, De hecho, estudios recientes evidencian que la formación en tecnologías DevOps y la incorporación de herramientas basadas en inteligencia artificial son claves para superar estos cuellos de botella [1] y mejorar la confiabilidad en entornos de alta demanda operativa [2].

Resolver este problema es importante porque actualmente se requiere de uno a dos días para desplegar todos los microservicios necesarios, lo cual genera retrasos, sobrecarga de trabajo y riesgo de sanciones contractuales. La situación descrita no es particular de una sola organización, sino que representa una problemática común en diversas empresas del sector, especialmente aquellas con estructuras operativas similares. Con la solución propuesta se reducirán drásticamente los tiempos de los despliegues, permitiendo realizarlos en cuestión de minutos. Esto servirá para optimizar los recursos humanos y técnicos, aumentar la productividad del equipo y liberar al personal de tareas repetitivas y manuales [3].

La solución es útil porque mejora la eficiencia operativa, reduce los costos, y permite generar datos valiosos para la toma de decisiones estratégicas [4], ya que la observabilidad y automatización son claves para responder rápidamente a la complejidad de los entornos distribuidos en microservicios. Será especialmente útil para los ingenieros de despliegue, los equipos de QA, los líderes de proyecto y la gerencia de sistemas, al facilitar un proceso más ágil, controlado y sostenible dentro de la gestión de sistemas de información reduciendo los cuellos de botella en los flujos de trabajo e incrementando la capacidad de respuesta operativa [5].

1.3 Objetivos.

1.3.1 Objetivo general

Desarrollar una solución automatizada para el despliegue de microservicios en ambientes de prueba, orientada a reducir los tiempos operativos y optimizar los recursos técnicos en empresas de telecomunicaciones, mediante el uso de herramientas de automatización e integración continua, con el fin de mejorar la eficiencia y sostenibilidad en la gestión de sistemas de información.

1.3.2 Objetivos específicos

1. Analizar la situación actual del proceso de despliegue de ambientes de prueba, identificando tiempos, recursos involucrados, herramientas utilizadas y los principales cuellos de botella en el entorno operativo de QA Desarrollar el control proporcional integral… (Ingeniería en Electrónica y Automatización).
2. Identificar las oportunidades de mejora en la gestión del proceso de preparación de ambientes, utilizando la situación actual del proceso considerando la carga operativa del personal de despliegue, la arquitectura de microservicios y la demanda de ambientes simultáneos.
3. Diseñar una arquitectura automatizada y adaptable, mediante la integración de herramientas utilizadas dentro de la organización y orientadas a reducir el esfuerzo manual y permitir el despliegue eficiente de ambientes de prueba bajo demanda.
4. Evaluar un prototipo funcional de la arquitectura propuesta, en el diseño y que permita a los usuarios del área de QA gestionar de forma autónoma el despliegue parcial o completo de microservicios, con base en las necesidades de cada proyecto.

1.4 Marco teórico

1.4.1 Despliegue de software en entornos empresariales

Ciclo de vida del software

El ciclo de vida de software (Software Development Life Cycle, SDLC) es un modelo estructurado que define las fases necesarias para el desarrollo, implementación y mantenimiento de sistemas informáticos. Este concepto ha sido aplicado en la industria de software para estructurar, planificar y controlar el proceso de desarrollo, con el objetivo de asegurar consistencia y trazabilidad en los proyectos informáticos [6].

Las fases clásicas del SDLC son: análisis de requerimientos, diseño del sistema, implementación, pruebas, despliegue y mantenimiento. Estas etapas han sido reconocidas como practicas esenciales en el desarrollo del software, sin importar el modelo utilizado (cascada, iterativo, ágil, espiral entre otros). [7]

Figura 1. Ciclo de Vida de Software

Fuente: https://www.linkedin.com/pulse/modelos-del-ciclo-de-vida-software-daniel-hernandez/

En entornos empresariales con arquitecturas distribuidas, como los microservicios, se ha documentado que la integración de automatización y monitoreo dentro del ciclo de vida del software ayuda a mejorar la confiabilidad del sistema. Waseem et al. argumenta que, para mantener la estabilidad operativa en este tipo de entornos, es fundamental incorporar prácticas de verificación, diseño observable y despliegue automatizado desde las primeras fases del SDLC [8].

Etapas del despliegue de software

El despliegue de software es el proceso mediante el cual una aplicación es trasladada desde un entorno de desarrollo o integración hacia un entorno de producción o prueba. Este proceso está compuesto de fases que están sujetas a políticas de control de versiones, integración continua y gestión de artefactos, factores clave para garantizar la trazabilidad y reproducibilidad de los entornos de prueba [9].

En entornos con alta carga operativa, como el sector de telecomunicaciones, se ha evidenciado que la ausencia de automatización en la etapa de despliegue genera cuellos de botella significativos. Agrawal et al. Identifican que los mayores retrasos operativos ocurren durante las etapas de configuración manual y validación, proponiendo como solución la incorporación de pipelines de entrega continua y herramientas de infraestructura como código (IaC) [10].

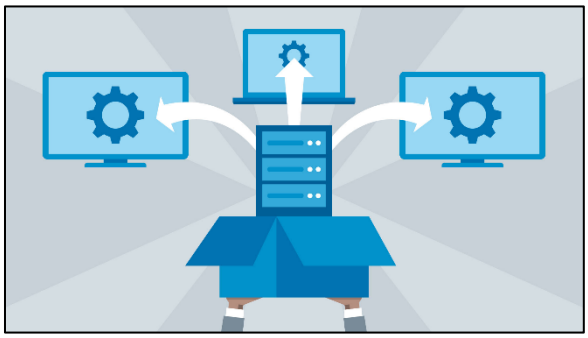

Figura 2. Despliegue de Software

Fuente : https://medium.com/devops-dudes/tagged/deployment-automation

El proceso de despliegue puede dividirse en varias etapas estructuradas que permiten una transición controlada y segura, según Humble y Farley [11], estas etapas incluyen:

- Planificación del despliegue: definición de la estrategia, recursos, ventanas de mantenimiento, mecanismos de reversión y métricas de éxito.
- Preparación del entorno: provisión de infraestructura y configuración del entorno, ya sea de forma manual o automatizada.
- Empaquetado del software: compilación, versionado y almacenamiento del artefacto en un repositorio de gestión como JFrog Artifactory o Nexus.
- Despliegue: traslado e instalación del artefacto en el entorno de destino, mediante procesos automatizados o manuales.
- Verificación post-despliegue: ejecución de pruebas básicas, validación de servicios y monitoreo del sistema.
- Documentación y retroalimentación: registro de resultados, recopilación de observaciones y retroalimentación para procesos futuros.

Ambientes tecnológicos

Los ambientes tecnológicos son entornos controlados y configurables que permiten ejecutar, validar y mantener aplicaciones de software a lo largo de su ciclo de vida. Estos ambientes pueden ser físicos o virtuales y su correcta definición es esencial para garantizar la calidad y estabilidad del sistema desplegado [12].

En muchas definiciones sobre los ambientes tecnológicos se identifica tres tipos principales de ambientes: desarrollo, pruebas (QA) y producción. Cada ambiente cumple con una función específica dentro del proceso de entrega de software. En contextos empresariales donde se trabaja con arquitecturas de microservicios, los

ambientes deben ser capaces de soportar múltiples componentes distribuidos, configuraciones dinámicas, y un alto grado de automatización para facilitar la integración y el despliegue continuo.

La estandarización de ambientes tecnológicos es una práctica recomendada para mitigar inconsistencias entre entornos, especialmente en procesos de validación funcional. Waseem et al. afirman que la falta de consistencia entre los entornos de prueba y producción representa una de las principales fuentes de errores en sistemas distribuidos [13].

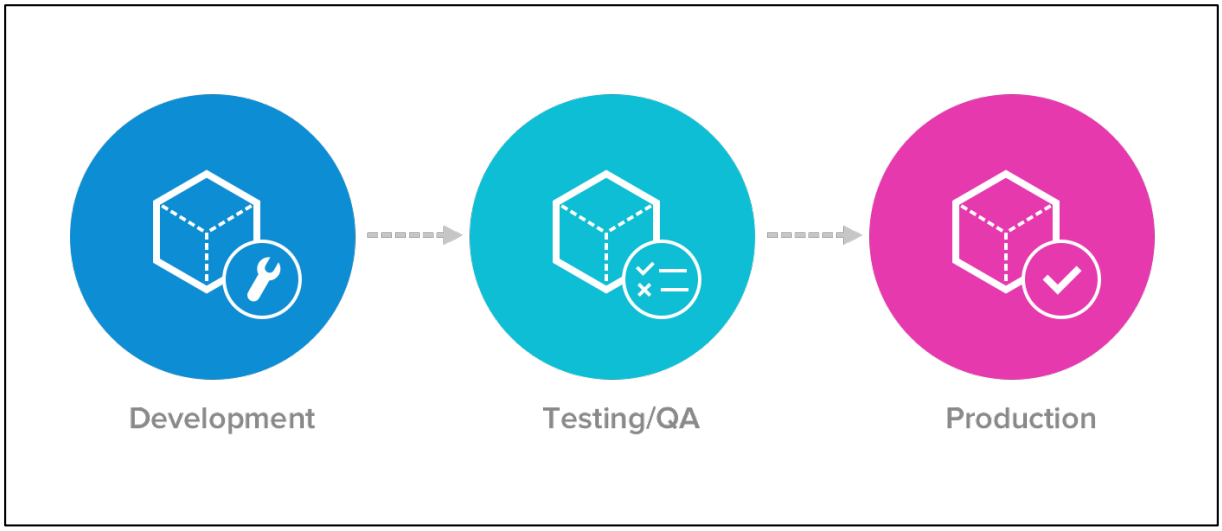

Figura 3. Ambientes Tecnológicos

Fuente: https://www.3csoftware.com/impactecs-in-depth-comparing-models-and-migrating-items-between-models/

Retos comunes en el despliegue

El despliegue de software es una de la fase crítica en el ciclo de vida del desarrollo, especialmente en sistemas empresariales con arquitecturas distribuidas. Esta etapa enfrenta diversos retos técnicos y operativos que pueden afectar la calidad del sistema, el tiempo de entrega y la continuidad de los servicios. Entre los principales desafíos se encuentran: la complejidad en la gestión de dependencias, la falta de automatización, la inconsistencia entre entornos y los errores derivados de configuraciones manuales [14].

Los errores de despliegue suelen originarse por diferencias entre los entornos de prueba y producción, así como por la ausencia de mecanismos de monitoreo y validación continua. Además, existe la necesidad de estandarizar los procesos de despliegue para evitar inconsistencias y reducir el riesgo operativo.

Otro reto frecuente son los cuellos de botella operativos estos se presentan cuando los procesos de despliegue dependen excesivamente de la intervención manual, lo cual genera demoras y sobrecarga en equipos técnicos, especialmente en empresas con alta demanda de ambientes de prueba.

1.4.2 Arquitectura de microservicios

Comparación entre arquitecturas

Las arquitecturas de software definen la forma en que se estructuran, desarrollan y despliegan las aplicaciones. Y se destacan dos enfoques para la construcción de sistemas empresariales: la arquitectura monolítica y la arquitectura de microservicios. Cada una presenta ventajas y limitaciones que han sido ampliamente analizadas en función del tipo de sistema, el volumen de usuarios y la frecuencia de cambios funcionales [15].

La arquitectura monolítica centraliza todos los componentes de una aplicación en un único paquete de despliegue. Esto implica que cualquier modificación, incluso menor, requiere reconstruir y volver a desplegar todo el sistema. Este modelo puede ser adecuado en proyectos pequeños o de baja complejidad, pero en entornos con alta demanda operativa puede derivar en problemas de escalabilidad, acoplamiento excesivo y dificultad de mantenimiento [16].

La arquitectura de microservicios organiza el sistema como un conjunto de servicios independientes que se comunican entre sí mediante interfaces bien definidas. Este enfoque facilita la implementación de despliegues autónomos y escalabilidad horizontal. Waseem et al. señalan que esta separación de responsabilidades permite gestionar mejor los errores, realizar pruebas aisladas y responder de forma más eficiente ante la evolución de los requisitos [17].

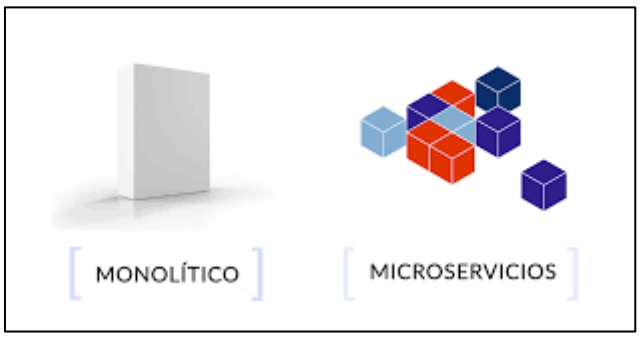

Figura 4. Arquitectura de Microservicio

Fuente: https://ed.team/blog/que-es-y-para-que-sirve-la-arquitectura-de-microservicios

Ventajas de los microservicios

Uno de los beneficios fundamentales de los microservicios es la escalabilidad individualizada. Cada servicio puede ser escalado de forma independiente en función de su demanda, lo que permite optimizar el uso de recursos y adaptarse eficientemente a cargas variables. Taibi et al. sostienen que esta capacidad de escalado granular resulta

especialmente útil en entornos con alta variabilidad operativa, como los sistemas empresariales de telecomunicaciones [18].

Otra ventaja es la facilidad de mantenimiento y actualización de servicios. Al estar desacoplados, los microservicios permiten aplicar cambios en un componente sin afectar el resto del sistema. Waseem et al. resaltan que esta propiedad facilita la implementación de procesos de integración y despliegue continuo (CI/CD), así como la automatización de pruebas en entornos de validación [19].

Además, la arquitectura de microservicios permite una adopción progresiva. A diferencia de modelos que requieren una reestructuración completa, los microservicios pueden ser introducidos de forma gradual, facilitando su incorporación en sistemas legados sin afectar su operación continua [20].

Desafíos de validación e integración

Uno de los principales retos es la complejidad en las pruebas de integración, ya que los microservicios dependen de múltiples componentes desplegados de forma concurrente. Según Smith et al., la verificación funcional en entornos distribuidos requiere configurar ambientes de prueba realistas, reproducibles y sincronizados, lo que representa un desafío logístico y técnico para los equipos de calidad [21].

Además, los microservicios suelen tener interacciones no determinísticas debido al uso de colas, eventos, APIs externas y balanceadores de carga. Esto dificulta la aplicación de pruebas tradicionales y exige estrategias como pruebas de contrato, pruebas de componentes aislados y pruebas end-to-end coordinadas mediante herramientas de orquestación [22].

Los microservicios presentan un reto adicional relacionado con la trazabilidad de fallos durante las pruebas. La naturaleza distribuida del sistema complica la recolección de logs, métricas y eventos relevantes para el diagnóstico de errores. En este contexto, es fundamental contar con soluciones de observabilidad que integren trazas distribuidas, monitoreo de servicios y análisis de dependencias para facilitar la depuración de fallos complejos [23].

1.4.3 Automatización del despliegue de software

Cultura DevOps

En los últimos años, la adopción del enfoque DevOps ha transformado los procesos tradicionales de desarrollo y operaciones, fomentando la automatización, la colaboración y la entrega continua de software. DevOps se ha consolidado como un modelo cultural y técnico que busca la integración fluida entre los equipos de desarrollo (Dev) y operaciones (Ops), con el objetivo de mejorar la calidad, velocidad y confiabilidad del ciclo de vida del software [24].

DevOps promueve prácticas como la integración continua, la entrega continua (CI/CD) y el monitoreo proactivo. Estas prácticas permiten automatizar el ciclo de vida del software, desde la compilación hasta el despliegue en distintos entornos, garantizando calidad y eficiencia en cada etapa del proceso [25].

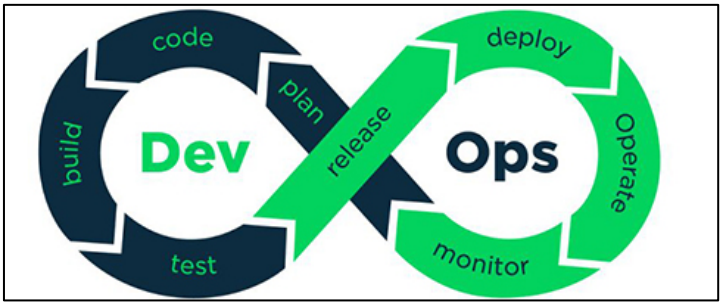

Figura 5. DevOps

Fuente: https://dev.to/emminex/what-is-devops-7f0

En organizaciones con alta demanda operativa, como las empresas de telecomunicaciones, la automatización del despliegue se convierte en un componente esencial para afrontar la complejidad y el dinamismo de los entornos tecnológicos. La ejecución manual de estas tareas resulta costosa, con altos tiempos de ejecución y propensa a errores, lo cual puede impactar negativamente en la estabilidad de los servicios. Por ello, se ha demostrado que la adopción de pipelines automatizados de despliegue contribuye a una mayor agilidad operativa, minimización de fallos humanos y reducción de tiempos de entrega [26].

Integración y entrega continua (CI/CD)

La automatización del despliegue de software ha sido impulsada por la necesidad de acelerar el ciclo de desarrollo de software y aumentar la confiabilidad en la entrega de aplicaciones. Las prácticas de Integración Continua (Continuous Integration, CI) y Entrega Continua (Continuous Delivery, CD) se han consolidado como pilares fundamentales para garantizar la calidad y consistencia del software en todos sus ambientes tecnológicos como el desarrollo prueba y producción.

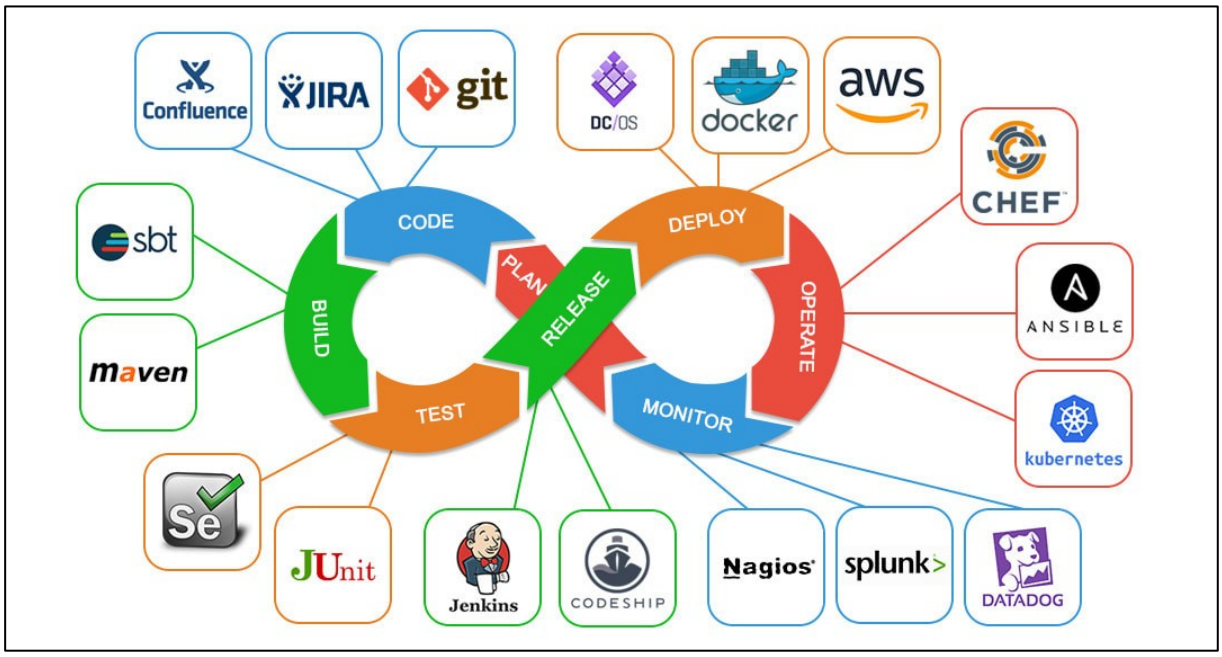

Figura 6. CI/CD

Fuente: https://www.linkedin.com/pulse/devops-qu%C3%A9-son-vs-sysadmin-rodrigo-marti-pascual/

La integración continua permite fusionar de manera frecuente los cambios de código realizados por diferentes desarrolladores en un repositorio compartido, desencadenando automáticamente la ejecución de pruebas unitarias y análisis de calidad. Este proceso busca detectar errores tempranamente y reducir los conflictos de integración. Y la entrega continua amplía este enfoque automatizando el despliegue del software en entornos controlados, permitiendo su liberación de forma rápida y segura siempre que se cumplan los criterios de calidad definidos [27].

En arquitecturas como los microservicios la implementación de pipelines CI/CD permite desplegar y probar cada servicio de manera independiente, manteniendo la coherencia entre versiones y facilitando el aislamiento de errores. H. Shafiq indica que la adopción de CI/CD en entornos distribuidos mejora la eficiencia operativa, reduce los tiempos de ciclo y refuerza la trazabilidad de los cambios en empresas con alta carga tecnológica, como las del sector de telecomunicaciones [28].

Herramientas de soporte

La automatización del despliegue de software se fundamenta en el uso de herramientas que permiten ejecutar procesos complejos de manera repetible, eficiente y controlada. Estas herramientas forman el núcleo de los pipelines de CI/CD y están diseñadas para orquestar tareas de construcción, prueba, integración, empaquetado y despliegue de aplicaciones en múltiples entornos.

Entre las plataformas de automatización más empleadas se encuentran Jenkins, GitLab CI/CD, GitHub Actions y Azure DevOps. Estas herramientas permiten definir flujos de trabajo automatizados que integran las distintas fases

del ciclo de vida del software. Su capacidad para integrarse con sistemas de control de versiones, gestores de artefactos, plataformas en la nube y soluciones de monitoreo las convierte en componentes esenciales en entornos de entrega continua [29].

En los microservicios el despliegue utiliza herramientas de conterenización como Docker, que encapsula cada servicio junto con sus dependencias, garantizando la portabilidad entre entornos. Para orquestar estos contenedores, Kubernetes permite gestionar la distribución de cargas, la resiliencia de los servicios y la escalabilidad automática. Estas tecnologías habilita despliegues segmentados y actualizaciones sin interrupciones, características fundamentales en sistemas distribuidos [30].

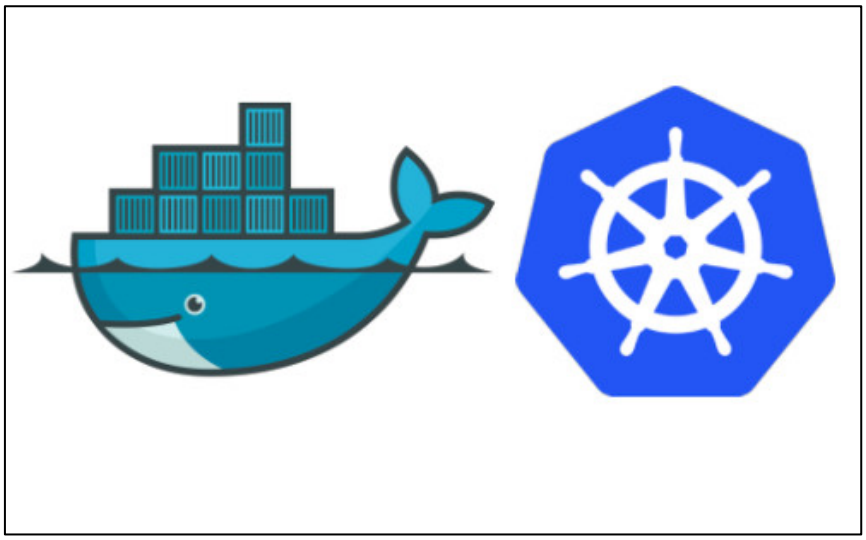

Figura 7. Docker and Kubernets

Fuente: https://sonamthakur7172.medium.com/microservices-design-principle-design-patterns-cd14321c61fd

La integración de herramientas CI/CD, contenedores y orquestadores conforma un ecosistema robusto para automatizar el despliegue de microservicios. Esta arquitectura de soporte es fundamental para garantizar la eficiencia operativa, la reproducibilidad de los entornos de prueba y la calidad del software en sectores con alta demanda tecnológica.

Metricas de desempeño.

La evaluación del desempeño en procesos automatizados de despliegue de software es fundamental para medir la eficiencia, calidad y confiabilidad de los sistemas implementados. Las métricas permiten a los equipos de ingeniería identificar cuellos de botella, validar mejoras continuas y tomar decisiones informadas respecto al diseño y operación de pipelines de despliegue.

A nivel general, las prácticas de DevOps han promovido el uso de indicadores clave como la frecuencia de despliegue (Deployment Frequency), el tiempo medio de entrega (Lead Time for Changes), la tasa de fallos en

cambios (Change Failure Rate) y el tiempo medio de recuperación (Time to Restore Service). Estas métricas han sido reconocidas como esenciales para evaluar el rendimiento de los equipos de desarrollo y operaciones en entornos automatizados y altamente dinámicos [31].

Figura 8. Metricas DORA

Fuente: https://medium.com/gits-apps-insight/dora-metrics-how-to-measure-software-delivery-performance-e890ec2011c0

En las empresas de telecomunicaciones con alta carga operativa, estas métricas permiten gestionar entornos de prueba de forma más efectiva, optimizando la disponibilidad de recursos, reduciendo la intervención manual y garantizando la trazabilidad de los resultados. Además, la recopilación y análisis sistemático de estas métricas, mediante herramientas de observabilidad e integración continua, contribuye a mejorar la trazabilidad de los cambios y a facilitar auditorías técnicas en organizaciones con altos estándares de cumplimiento y disponibilidad.

1.4.4 Herramientas de automatización y orquestación de flujos

N8n

La automatización de flujos de trabajo es una práctica adoptada en entornos de desarrollo y operaciones, ya que permite reducir el esfuerzo manual, disminuir errores operativos y mejorar la eficiencia en tareas repetitivas y críticas para el ciclo de vida del software. Las herramientas de orquestación se emplean para coordinar y ejecutar procesos distribuidos mediante flujos lógicos, integrando distintos servicios, aplicaciones y plataformas.

n8n se ha consolidado como una herramienta de automatización de código abierto que permite crear flujos de trabajo automatizados a través de una interfaz visual basada en nodos. n8n ofrece un modelo de automatización autoalojado y altamente personalizable, lo cual favorece su adopción en entornos corporativos con restricciones de seguridad o necesidades específicas de integración. Esta herramienta admite la conexión con más de 200

servicios a través de conectores predefinidos, incluyendo bases de datos, APIs REST, herramientas DevOps y plataformas de mensajería [32].

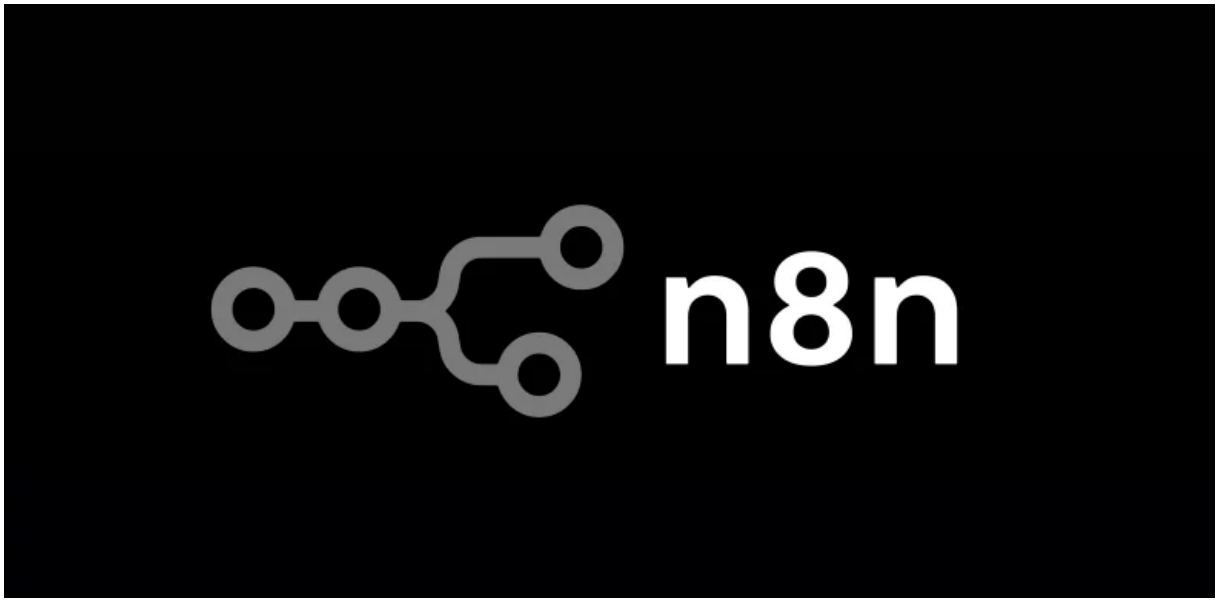

Figura 9. n8n

Fuente: https://www.linkedin.com/posts/ai-insider-intel_automation-ai-devtools-activity-7310724103680720896-93Yg/

Además, n8n soporta tanto ejecuciones programadas como disparadores por eventos, lo cual resulta adecuado para sistemas donde se requiere una reacción automatizada ante cambios en el estado de los servicios, como actualizaciones de versiones o resultados de pruebas automatizadas. Su uso en entornos de telecomunicaciones ha sido reportado en proyectos orientados a la automatización de la gestión de incidencias y del monitoreo de aplicaciones distribuidas [33].

Telegram como canal de integración

La integración de canales de mensajería como Telegram ha adquirido relevancia debido a su arquitectura basada en APIs, su disponibilidad multiplataforma y su capacidad de operar en tiempo real, lo que facilita su adopción como medio de interacción entre sistemas automatizados y operadores humanos.

Telegram ofrece una API robusta para la creación de bots, que permite ejecutar comandos, enviar notificaciones y consultar datos en tiempo real, integrándose eficientemente con flujos de trabajo automatizados. Esta capacidad ha sido aprovechada en la automatización de pipelines de desarrollo y despliegue, así como en la supervisión de entornos de prueba y producción, donde las alertas instantáneas y la interacción directa contribuyen a reducir el tiempo de respuesta ante fallos o eventos críticos [34].

Figura 10. Telegram

Fuente: https://web.telegram.org/a/

Telegram, cuando se integran con herramientas de automatización como n8n, Jenkins o GitLab CI/CD, permiten mejorar la visibilidad de los procesos y habilitar una comunicación bidireccional automatizada, facilitando acciones remotas como la aprobación de despliegues, la ejecución de tareas o la recepción de informes sobre el estado de los sistemas [35]. Este tipo de integración es especialmente valiosa en organizaciones con alta carga operativa, donde el monitoreo constante de múltiples servicios distribuidos requiere canales de notificación eficaces y en tiempo real.

Orquestación de procesos

La orquestación de procesos es un componente esencial dentro de los entornos de automatización moderna, ya que permite coordinar de forma estructurada y lógica la ejecución de tareas distribuidas, particularmente en arquitecturas basadas en microservicios. La orquestación centraliza el control del flujo, facilitando la supervisión y la recuperación ante errores en contextos complejos como los despliegues automatizados [36].

En el ámbito de la ingeniería de software, se han desarrollado diversas plataformas para la orquestación de flujos de trabajo que permiten integrar múltiples herramientas, servicios y procesos en pipelines automatizados. Herramientas como Apache Airflow, Argo Workflows y n8n han sido utilizadas ampliamente para diseñar, ejecutar y monitorear procesos de orquestación, proporcionando funcionalidades como el manejo de dependencias, ejecución paralela y lógica condicional.

Figura 11. Orchestration tools

Fuente: https://medium.com/@sumitmudliar/argo-workflow-vs-apache-airflow-0325158536e5

En entornos de telecomunicaciones con alta carga operativa, la orquestación ha sido empleada para optimizar tareas críticas como el aprovisionamiento de ambientes de prueba, el despliegue continuo de microservicios y la notificación de eventos en tiempo real. La implementación de orquestadores visuales o declarativos contribuye significativamente a reducir la complejidad operativa, mejorar la trazabilidad y garantizar la consistencia de los entornos de prueba [37].

Herramientas open source en entornos DevOps

El uso de herramientas open source ha adquirido una relevancia estratégica, especialmente en organizaciones que buscan agilidad, escalabilidad y control sobre sus flujos de trabajo. Estas herramientas permiten automatizar y orquestar procesos de desarrollo, pruebas y despliegue de manera integral, integrando funcionalidades clave como integración continua, entrega continua y monitoreo.

Proyectos como Jenkins, GitLab CI, Argo CD, y Spinnaker destacan por su adopción generalizada y sus capacidades para gestionar pipelines de automatización en arquitecturas basadas en microservicios. Estas soluciones permiten construir flujos complejos de despliegue, reducir errores humanos, mejorar la trazabilidad y disminuir los tiempos de entrega [38].

Figura 12. Open Source

Fuente: https://directortic.es/estrategia-it/las-ventajas-del-open-source-la-empresa-2017022117457.htm

El uso de herramientas de código abierto también responde a la necesidad de control sobre la infraestructura tecnológica en sectores de telecomunicaciones, donde los sistemas críticos requieren una automatización confiable y transparente. Según estudios, la combinación de estas herramientas con plataformas de contenedores como Kubernetes permite a las organizaciones escalar sus aplicaciones y gestionar microservicios de forma eficiente [39].

CAPÍTULO 2

2. Metodología.

La metodología de la investigación se basó en un enfoque aplicado, con diseño no experimental y corte transversal. Se empleó una metodología descriptiva y tecnológica, orientada a identificar las oportunidades de mejora en el proceso de despliegue de ambientes de prueba, diseñar e implementar una solución automatizada y evaluar su impacto en términos de tiempo de despliegue y carga operativa. El estudio se basó únicamente en métodos cuantitativos, utilizando encuestas estructuradas con escala de Likert y preguntas cerradas. Los resultados fueron analizados mediante estadísticas descriptivas, como el cálculo de frecuencias y porcentajes, mediante la comparación de indicadores antes y después del desarrollo del prototipo, lo que permitió evaluar los cambios percibidos en el proceso de despliegue.

Antes de definir la arquitectura y seleccionar las herramientas tecnológicas, se realizó un diagnóstico mediante encuestas dirigidas al equipo técnico, lo cual permitió identificar los principales desafíos del proceso actual. Entre los hallazgos más relevantes, se observó que el 57.9 % de los encuestados indicó que el despliegue completo de un ambiente de prueba tomaba en promedio 3 días, mientras que el 31.6 % reportó 2 días. Además, más del 70 % manifestó que el nivel de carga operativa era de moderada a alta complejidad. También se evidenció que los errores durante los despliegues eran frecuentes y que la satisfacción con el proceso era limitada, con una mayoría puntuando con apenas 3 sobre 5. Finalmente, se identificó que el 79 % experimentó retrasos en los cronogramas por el tiempo requerido para completar los despliegues.

Estos resultados sirvieron como base para justificar la necesidad de una solución integral que permitiera automatizar tareas críticas del proceso, reducir los tiempos de respuesta y otorgar mayor autonomía al equipo de calidad. A partir de esta evaluación inicial se diseñó la arquitectura propuesta

Nivel de Carga Operativa:

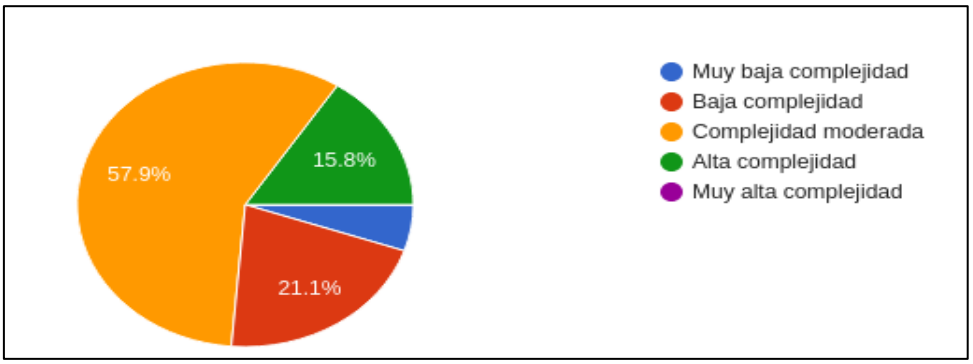

Figura 13. Carga Operativa

Frecuencia de errores en el proceso de despliegue:

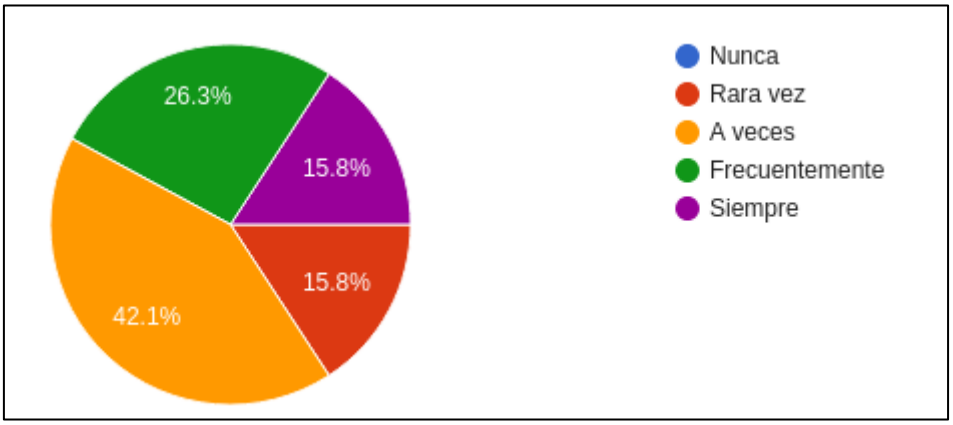

Figura 14. Frecuencia de Errores

Tiempo promedio en levantar ambiente de pruebas completos en días:

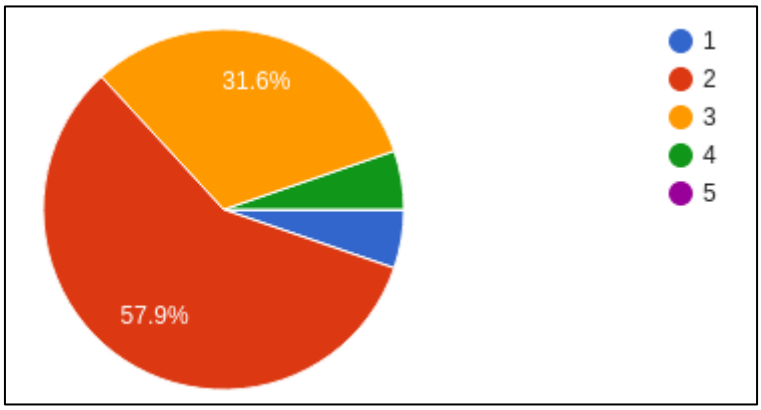

Figura 15. Tiempo promedio

2.1 Enfoque metodológico y arquitectura de la solución propuesta

El enfoque metodológico se centró en la identificación, selección e integración de herramientas tecnológicas que permitieran automatizar el proceso de despliegue de ambientes de prueba. A partir de un análisis del proceso actual y de los principales puntos de mejora, se diseñó una arquitectura integral basada en componentes de código abierto, con el objetivo de reducir los tiempos de despliegue, disminuir la carga operativa y otorgar mayor autonomía al equipo de calidad.

Para seleccionar las herramientas que conformarían la solución, se aplicó una matriz de decisión ponderada enfocada en la herramienta de orquestación de flujos automatizados (workflow engine), comparando N8N, GitHub Actions y Jenkins. Los criterios considerados fueron: integración con APIs, curva de aprendizaje, funcionalidades nativas, escalabilidad, soporte y comunidad. Esta evaluación fue necesaria para validar que su elección seguía siendo la más adecuada frente a alternativas actuales del mercado.

A continuación, se presenta el resultado de la matriz de decisión ponderada:

Tabla I. Ponderación de herramientas de orquestación

Criterio	Peso	N8N	Jenkins	GitHub Actions
Integración con APIs	25%	5	3	5
Curva de aprendizaje	20%	5	2	3
Funcionalidades nativas	20%	5	4	4
Escalabilidad	20%	4	5	5
Soporte y comunidad	15%	4	5	5
Puntaje total		4.65	3.55	4.55

Los resultados confirmaron que N8N destacaba por su facilidad de integración, su bajo nivel de complejidad para los usuarios (curva de aprendizaje baja) y sus potentes funcionalidades nativas para la gestión de flujos, por lo que fue ratificada como la herramienta principal para el diseño de la solución.

Además de N8N, se incorporaron otros componentes tecnológicos, cada uno seleccionado por su pertinencia técnica y por estar ya integrados en el ecosistema de la empresa:

GitLab: Sistema de control de versiones y plataforma de CI/CD ampliamente utilizada en la organización, permitiendo una integración fluida con los repositorios de código y pipelines de despliegue.

Redis: Base de datos en memoria elegida por su alta velocidad de acceso, ideal para almacenar información reutilizable en los flujos de despliegue y facilitar tiempos de respuesta óptimos.

Telegram: Plataforma de mensajería seleccionada como interfaz operativa por su facilidad de uso, su integración con APIs y su uso extendido dentro de la empresa.

OpenShift: Plataforma de orquestación de contenedores, que permite la gestión escalable y automatizada de microservicios y front-end en los ambientes de prueba.

Dado que la solución propuesta combinaba múltiples componentes, se procedió a realizar una evaluación integral de la arquitectura, con el fin de describir sus principales características técnicas y operativas, proporcionando una visión completa del sistema en su conjunto. Para ello, se elaboró un cuadro de síntesis que resume los atributos más relevantes de la arquitectura diseñada.

Tabla II. Resumen de atributos de arquitectura diseñada

Criterio	Solución propuesta (N8N, GitLab, Redis, Telegram, OpenShift)
Arquitectura integrada	Sí
Flujo automatizado	Completo y controlado por eventos
Persistencia de datos	Redis (rápida, en memoria)
Comunicación	APIs + Telegram
Curva de aprendizaje	Media-Baja
Escalabilidad	Alta (contenedores + OpenShift)
Costos	Gratuito/Open Source
Comunidad y soporte	Alta (GitLab, Telegram, OpenShift)

La combinación de estos componentes permitió diseñar una arquitectura modular, escalable y replicable, que optimiza los tiempos de despliegue y reduce la carga operativa. La solución integral propuesta demostró ser adecuada para abordar los retos identificados en el proceso actual de preparación de ambientes de prueba.

2.2 Diseño conceptual de la solución

Basándose en el enfoque metodológico y en la arquitectura integral definida en la sección anterior, se procedió a la elaboración del diseño conceptual de la solución automatizada de despliegue de ambientes de prueba. Este diseño tuvo como propósito establecer los componentes fundamentales, sus interacciones y los flujos lógicos necesarios para automatizar tareas que anteriormente eran ejecutadas de forma manual por el personal de despliegue.

El diseño conceptual se centró en la construcción de un sistema modular, orientado a eventos y adaptable a diferentes entornos de prueba. Para ello, se definieron los siguientes elementos clave:

Flujos de automatización (workflows): Fueron diseñados en N8N como nodos conectados entre sí, siguiendo una lógica condicional basada en comandos recibidos desde Telegram. Cada flujo representó una tarea automatizada específica, como consulta de datos parametrizados en Redis, ejecución de despliegues vía GitLab CI/CD y notificación de resultados.

Base de validación dinámica: Redis se utilizó como base de datos en memoria para almacenar temporalmente información clave relacionada con la ejecución de los flujos. Su alta velocidad de lectura y escritura permitió obtener respuestas inmediatas y eficientes. los cuales eran consultados por todos los flujos relacionados, permitiendo así reutilizar información y asegurar coherencia cuando diferentes usuarios solicitaban el despliegue de un mismo microservicio.

Interfaz de usuario vía Telegram: Se implementó un bot de Telegram que permitió la comunicación directa entre los usuarios y el sistema. A través de comandos abreviados y mensajes en lenguaje natural, los ingenieros de calidad pudieron interactuar con el sistema sin necesidad de acceder a plataformas técnicas o portales internos.

Orquestación del despliegue: GitLab se empleó como repositorio de código y motor de ejecución para los pipelines de despliegue. Cada solicitud activada desde N8N generó una llamada al pipeline correspondiente, pasando variables para el correcto despliegue en OpenShift.

Gestión de contenedores en OpenShift: Finalmente, el despliegue de microservicios y front-end se llevó a cabo en la plataforma OpenShift mediante llamadas a su API REST, lo que permitió una integración directa y segura para el despliegue, seguimiento y cierre de pods en el entorno de prueba.

Como resultado, el diseño conceptual permitió construir un sistema centrado en el usuario final (QA, despliegue), con mínima intervención manual, alta reutilización y orientado a mejorar la eficiencia operativa. Además, se consideraron desde el inicio aspectos como la trazabilidad, el manejo de errores, los tiempos de espera, todo ello documentado para facilitar el mantenimiento y la escalabilidad del modelo.

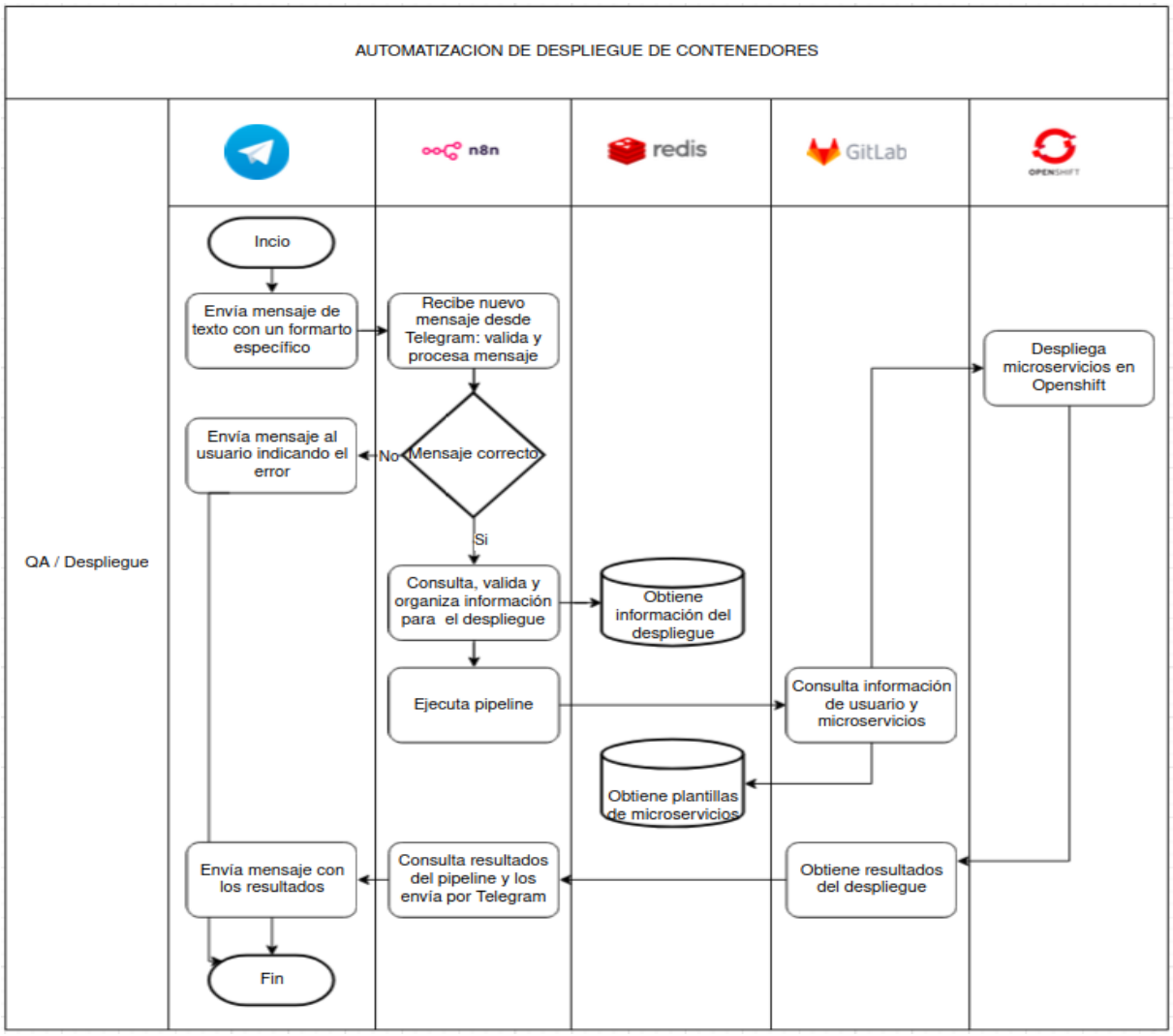

Figura 16. Diagrama de automatización del despliegue en ambientes de prueba.

2.3 Diseño detallado del sistema automatizado

Una vez definido el diseño conceptual, se procedió a estructurar el diseño detallado del sistema automatizado de despliegue. Esta fase comprendió la definición técnica minuciosa de cada componente, los parámetros de ejecución, la integración entre herramientas y las validaciones necesarias para garantizar un despliegue total o parcial eficiente y seguro.

El sistema se diseñó bajo una arquitectura distribuida, basada en contenedores, e incluyó los siguientes módulos técnicos:

Flujos de trabajo en N8N: Se implementaron flujos de trabajo orientados a eventos, donde cada ejecución representó una operación automatizada iniciada por comandos estructurados enviados desde Telegram. Los

primeros nodos de cada flujo fueron responsables de recuperar información de configuración desde Redis, necesaria para continuar con la ejecución de tareas como validaciones de datos para el despliegue.

A lo largo del flujo, se definieron nodos condicionales que permitieron bifurcar la ejecución de acuerdo con al tipo de componente (microservicio o front-end). Estas bifurcaciones activaban subflujos que incluían la generación dinámica de archivos de propiedades, bloqueo temporal de ejecuciones paralelas, y llamadas a pipelines definidos en GitLab CI/CD para realizar el despliegue correspondiente.

Integración con Redis: Se diseñaron estructuras clave-valor que almacenaron datos relacionados con cada flujo. Redis actuó como memoria intermedia para validar las condiciones requeridas antes de ejecutar un proceso. Por ejemplo, se verificó si los campos necesarios (CPU, Memoria y Réplicas) estaban presentes antes de proceder al despliegue.

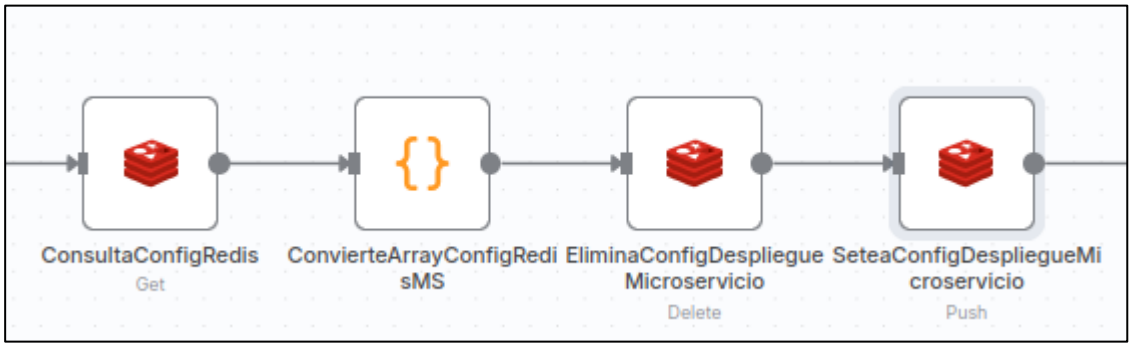

Figura 17. Integración con redis

Interacción con Telegram: A través de un bot configurado, se habilitó la ejecución de comandos estructurados enviados directamente desde Telegram. El usuario QA/Despliegue puede enviar un bloque con parámetros como usuario, nombre_ms, qa, bd_url y kafka_ip, los cuales son interpretados automáticamente por los flujos definidos en N8N.

Por ejemplo, al enviar la información, el bot procesa los valores recibidos y ejecuta un flujo de despliegue específico en OCP (OpenShift), sin que el usuario deba acceder directamente a la infraestructura ni realizar acciones técnicas manuales.

Esta integración permite que el equipo QA realice despliegues, sean totales o parciales y sin necesidad de conocimientos técnicos avanzados, optimizando los tiempos operativos y reduciendo errores humanos.

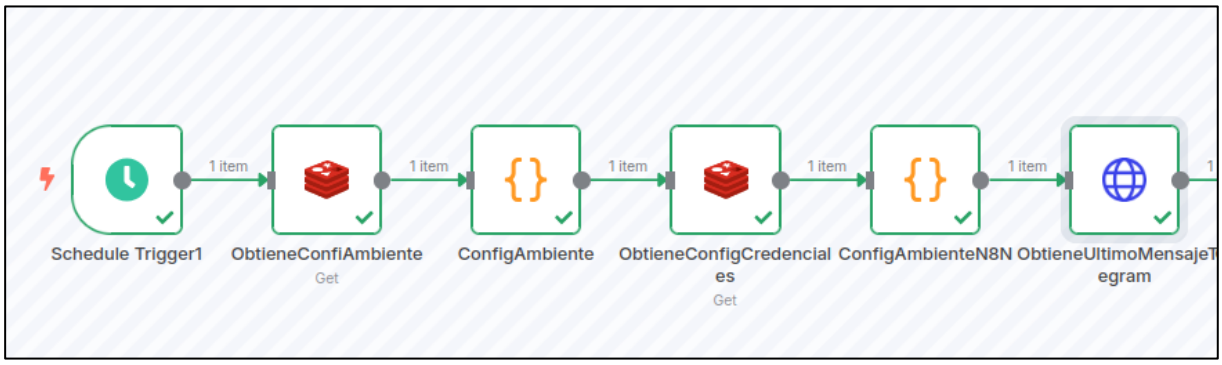

Figura 18. Integracion con telegram

Automatización de despliegue con GitLab CI/CD: Se construyó una llamada al pipeline correspondiente en GitLab. El sistema utilizó la API de GitLab para disparar la ejecución del pipeline, pasando como variables dinámicas los datos obtenidos desde Redis y el flujo de N8N. Se aseguraron validaciones previas para evitar ejecuciones duplicadas y controlar condiciones de error.

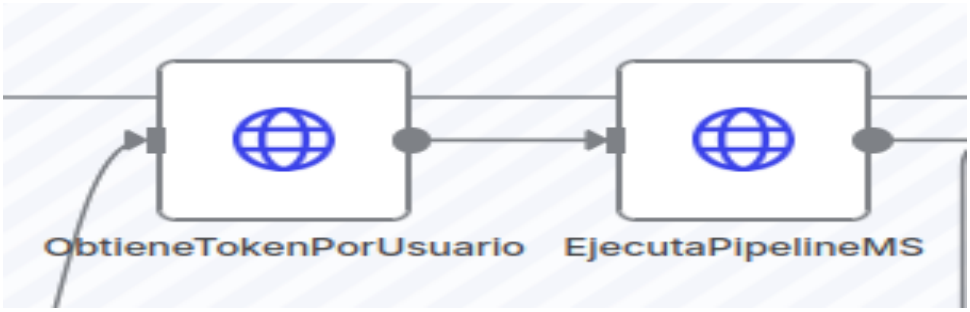

Figura 19. Automatización de despliegue con GitLab CI/CD

Despliegue de contenedores en OpenShift: Los pipelines de GitLab incluyeron scripts que interactuaron directamente con OpenShift para desplegar contenedores de microservicios y front-end. La estructura del script permitió escalar pods, revisar logs en caso de error y confirmar el estado de los despliegues.

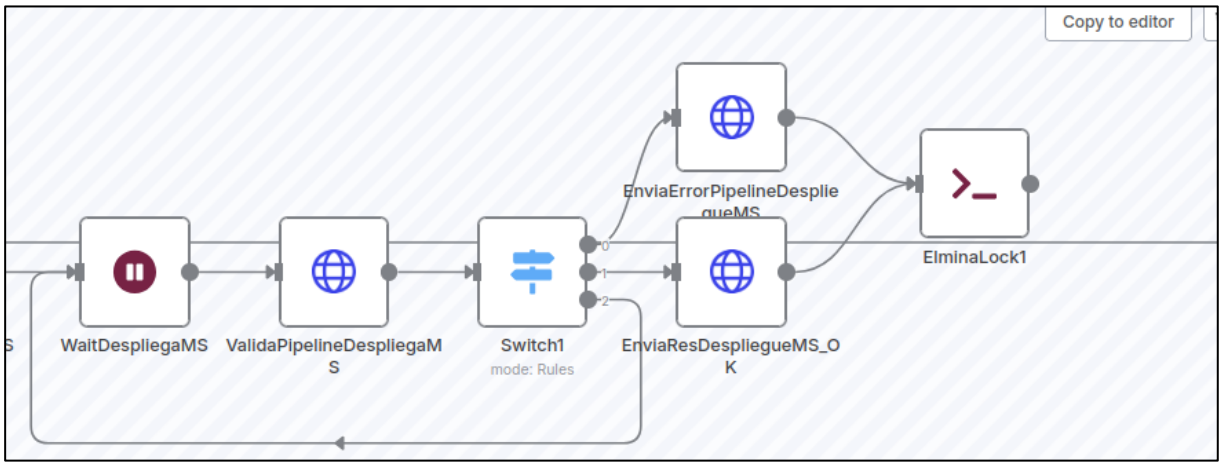

Figura 20. Despliegue de contenedores en OpenShift

Lógica condicional y control de errores: Todos los flujos de N8N incluyeron nodos para verificar el estado de la respuesta de cada API utilizada. En caso de error, se enviaron mensajes personalizados al canal de Telegram, explicando el motivo de la falla y sugiriendo acciones correctivas.

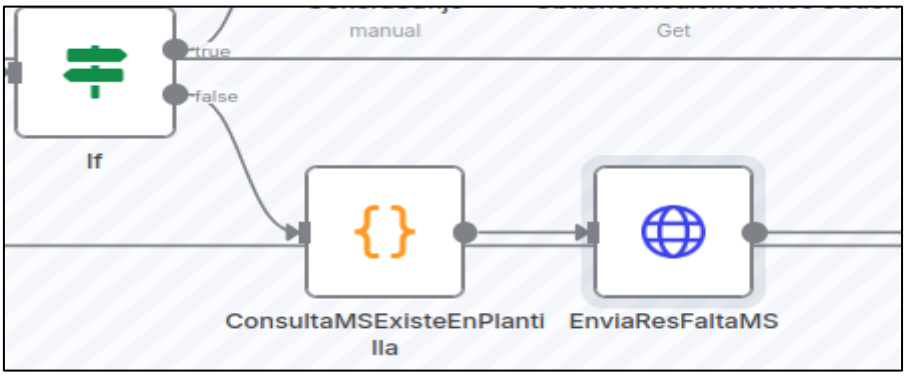

Figura 21. Lógica condicional y control de errores

Auditoría y trazabilidad: Cada ejecución quedó registrada tanto en N8N como en GitLab, permitiendo reconstruir el historial completo de acciones. Se utilizó un identificador único por flujo para facilitar el seguimiento y análisis posterior. Este enfoque aseguró transparencia y capacidad de diagnóstico frente a incidencias.

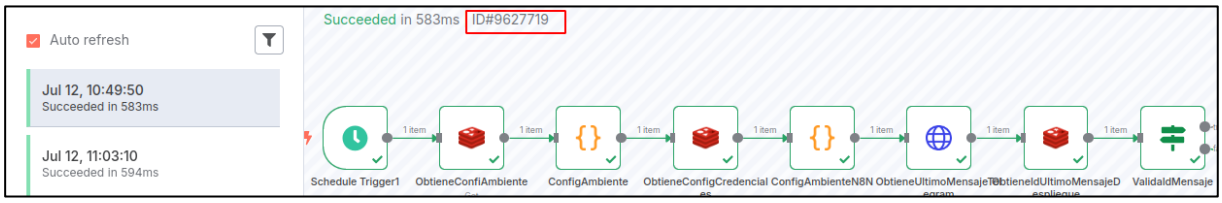

Figura 22. Auditoría y trazabilidad 1

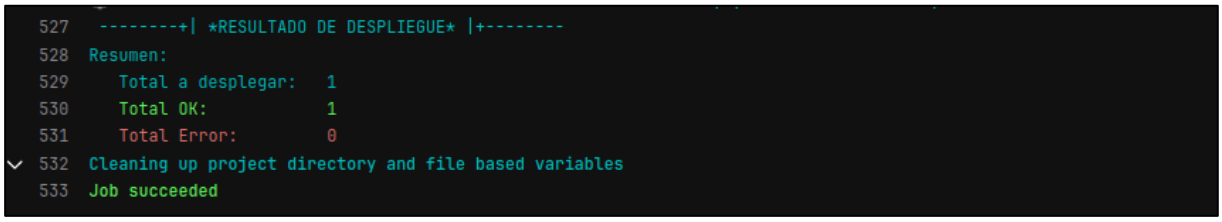

Figura 23. Auditoría y trazabilidad 2

El diseño detallado permitió construir una solución robusta, reutilizable y extensible, adaptable a otros entornos o proyectos similares. La combinación de herramientas de código abierto, flujos condicionales y validaciones dinámicas proporcionó una base sólida para escalar el sistema y mantenerlo de manera sostenible.

2.4 Participantes del estudio

El estudio contó con la participación directa de profesionales del área de calidad y despliegue de software de una organización dedicada a servicios tecnológicos. En total, se involucraron 29 personas, distribuidas en 26 ingenieros de calidad (QA) y 3 ingenieros de despliegue, quienes formaron parte activa del ciclo de vida de los ambientes de prueba.

Los participantes fueron seleccionados por su experiencia previa en procesos manuales de despliegue y pruebas funcionales, así como por su involucramiento frecuente en proyectos que requerían levantar ambientes compuestos por múltiples microservicios y front-end. Esta experiencia fue clave para obtener opiniones fundadas sobre los tiempos, carga operativa y dificultades técnicas antes y después del desarrollo del sistema automatizado.

La participación fue voluntaria y se garantizó el anonimato de las respuestas. Todos los ingenieros recibieron una explicación clara sobre los objetivos del estudio y la utilidad de la información recopilada, la cual se enfocó en validar la efectividad de la solución desarrollada.

Los roles dentro del equipo permitieron obtener una visión integral: los QA aportaron información sobre el proceso de solicitud, pruebas y validaciones de ambientes, mientras que los ingenieros de despliegue brindaron datos sobre la configuración, ejecución y seguimiento de los procesos manuales y automatizados. Esta diversidad de puntos de vista permitió evaluar de manera más objetiva el impacto del sistema automatizado en todo el flujo de trabajo.

2.5 Recolección de datos

La recolección de datos se llevó a cabo durante el proceso de desarrollo del prototipo de automatización de ambientes de prueba. Para ello, se elaboró un instrumento en forma de encuesta estructurada. La encuesta fue diseñada para capturar información antes y después de la puesta en marcha de la solución automatizada, permitiendo así una comparación directa sobre la experiencia y percepción de los usuarios.

Las preguntas cuantitativas utilizaron una escala tipo Likert de 5 puntos, facilitando la medición de percepciones sobre tiempo de despliegue, carga operativa y facilidad de uso. Adicionalmente, se incluyeron preguntas abiertas para obtener opiniones y observaciones cualitativas. El cuestionario aplicado se detalla en el Anexo 1.

2.6 Análisis de datos

El análisis de los datos se realizó una vez concluido el proceso de recolección de la información. Para el tratamiento de los datos cuantitativos, se utilizaron métodos estadísticos descriptivos como el cálculo de promedios, porcentajes y distribución de frecuencias. Estos resultados fueron representados visualmente mediante gráficos de barras y gráficos circulares, lo cual facilitó la interpretación comparativa de las condiciones antes y después de la automatización.

Se definieron dos variables principales que serán analizadas: el tiempo total de despliegue de ambientes de prueba y la carga operativa percibida por los participantes. Las respuestas fueron clasificadas en función del tipo de pregunta, y aquellas de naturaleza subjetiva fueron evaluadas utilizando una escala de Likert de 5 puntos, permitiendo medir el grado de acuerdo de los encuestados con afirmaciones relacionadas con la eficiencia, facilidad de uso y autonomía en los procesos.

En el caso de los datos cualitativos, se aplicó un análisis temático, el cual consistió en agrupar las respuestas abiertas en categorías según patrones recurrentes identificados en los comentarios. Por ejemplo, varias respuestas coincidieron en el tema *"reducción de tiempos de espera para realizar pruebas"*, mientras que otras se agruparon bajo el tema *"menor dependencia del equipo de despliegue"*. Este agrupamiento permitió extraer conclusiones significativas sobre la percepción del personal técnico respecto al impacto de la solución automatizada, tanto en términos de productividad como de colaboración entre equipos.

2.7 Evaluación del prototipo funcional

La evaluación del prototipo funcional se llevó a cabo una vez desarrollada la solución automatizada basada en N8N, Redis, GitLab CI/CD, y comandos de control a través de Telegram. Este prototipo permitió realizar despliegues selectivos de microservicios y contenedores frontend en ambientes de prueba, simulando el flujo de trabajo real que habitualmente es gestionado manualmente por los ingenieros de despliegue.

Para medir el impacto de la solución, se utilizaron dos enfoques complementarios: la comparación de indicadores clave antes y después del desarrollo, y la percepción del personal técnico involucrado. Los indicadores evaluados fueron el tiempo total de despliegue y la carga operativa relacionada con las tareas necesarias para poner en funcionamiento un entorno de pruebas funcional.

El diseño del prototipo tuvo como objetivo reducir significativamente el tiempo de despliegue que actualmente requiere entre 2 a 3 días, llevándolo a tiempos mínimos mediante automatización; así como disminuir la carga operativa del equipo de despliegue. Estos impactos serán evaluados mediante el análisis de los datos recolectados, utilizando encuestas estructuradas aplicadas a los participantes.

Además, se consideraron aspectos adicionales como la estabilidad del sistema, la facilidad de adopción, la autonomía de los ingenieros de calidad (QA) para ejecutar ambientes bajo demanda, y la claridad en la visualización de procesos. Los comentarios obtenidos durante la evaluación servirán como insumo para identificar oportunidades de mejora y establecer lineamientos para una posible implementación a escala en otros proyectos o departamentos de la organización.

2.8 Consideraciones éticas

Durante el desarrollo de esta investigación, se observaron principios éticos fundamentales que garantizaron el respeto y la protección de los participantes y de la información recolectada. Se aseguró en todo momento la voluntariedad de los participantes, quienes fueron informados previamente sobre los objetivos del estudio, el tipo de información que se recabaría, y el uso exclusivo con fines académicos y de mejora interna de procesos.

Antes de aplicar las encuestas, se solicitó el consentimiento informado de cada participante, dejando en claro que su participación era anónima, confidencial y que podían retirarse en cualquier momento sin consecuencias. Las respuestas fueron almacenadas sin asociar datos personales ni laborales específicos, lo que permitió preservar la privacidad de los colaboradores involucrados.

Asimismo, se evitó cualquier forma de presión jerárquica que pudiera influir en las respuestas del personal técnico. Se promovió un ambiente neutral y seguro para expresar opiniones y percepciones sobre el sistema actual y el prototipo automatizado propuesto.

Finalmente, se respetaron los principios de integridad académica, evitando el plagio, reportando fielmente los resultados obtenidos, y citando adecuadamente todas las fuentes de información consultadas. Esta investigación fue desarrollada bajo el compromiso de contribuir positivamente al entorno organizacional, sin afectar negativamente a ninguna persona o área involucrada.

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

En este capítulo presentaremos los principales resultados esperados y evidencia de los avances alcanzados en el desarrollo de la solución automatizada orientada a optimizar el proceso de despliegue de ambientes de prueba. La propuesta se enfoca en reducir los tiempos de despliegue, disminuir la carga operativa del equipo técnico y mejorar la autonomía del área de calidad.

3.1 Estado actual del desarrollo

Actualmente se ha desarrollado un prototipo funcional basado en una arquitectura distribuida compuesta por herramientas como N8N, Redis, GitLab y Telegram. Este sistema permite ejecutar flujos automatizados mediante comandos enviados desde Telegram, los cuales son interpretados por flujos configurados en N8N. Estos flujos realizan validaciones previas, obtienen configuraciones y datos desde Redis y ejecutan pipelines de despliegue definidos en GitLab CI/CD, lo que permite desplegar selectivamente microservicios y Front-end sobre entornos gestionados por OpenShift.

3.2 Flujo representativo implementado

A continuación, se muestra un flujo representativo que integra los componentes claves de la solución:

1. El usuario QA envía un mensaje estructurado a través de Telegram.
2. N8N recibe el mensaje, valida los datos y consulta a la base Redis.
3. Según los valores recuperados, N8N activa condiciones y ejecuta acciones como:
    a. Generar propiedades de despliegue.
    b. Llamar al pipeline correspondiente en GitLab.
    c. Despliegues en Openshift
    d. Informar del resultado por Telegram.

Esta arquitectura ha demostrado un comportamiento estable en pruebas internas, permitiendo consolidar las herramientas técnicas para su implementación.

3.3 Cumplimiento de Objetivos

A continuación, se describe el nivel de cumplimiento de los objetivos establecidos:

- Análisis del proceso actual: Se realizó mediante encuestas y revisión del flujo de trabajo actual, donde se evidenció una carga operativa alta y tiempos prolongados de despliegue.

- Identificación de oportunidades de mejora: Se propuso una arquitectura modular que permite despliegues selectivos y más rápidos, considerando la carga del equipo de despliegue.
- Diseño de una arquitectura automatizada: Se implementó una arquitectura automatizada, integrando herramientas open source y logrando adaptabilidad a distintos tipos de proyectos (alto, medio, bajo).
- Evaluación funcional del prototipo: Basado a las encuestas obtenidas se ha logrado el objetivo.

3.3.1 Cuadro de complejidad y tiempo de despliegue

Esta clasificación permite visualizar cómo la automatización se adapta a distintos escenarios según el tamaño y la exigencia del proyecto. Se utiliza como evidencia comparativa para validar que el nuevo enfoque mejora significativamente los tiempos, que originalmente eran de 2 a 3 días. Además, sirve como sustento para el cumplimiento de los objetivos específicos 2, 3 y 4. Especialmente en cuanto a eficiencia y reducción de carga.

Tabla III. Tiempos estimados de despliegue

Tipo de Proyecto	Nº de Microservicios	Tiempo estimado de despliegue
Alto	80	1 hora
Medio	50	35 minutos
Bajo	30	20 minutos

Con la nueva solución que incluye herramientas como n8n para la orquestación y el uso de OpenShift, el proceso se reduce considerablemente en el caso de los proyectos más complejos, el tiempo estimado bajó a una hora. Esta diferencia no solo aligera la carga del equipo, sino que también permite responder más rápido cuando hay varias solicitudes en paralelo, algo muy común cuando varios QA están trabajando al mismo tiempo. Gracias a esta mejora, se están cumpliendo los objetivos planteados, en especial los relacionados con reducir tiempos y hacer más eficiente la gestión de ambientes. La tabla presentada sirve como respaldo de esta evolución y demuestra cómo el sistema se adapta bien a proyectos de distintos tamaños.

La forma actual del despliegue se realiza en la modalidad Vertical**,** despliegue por capas o funcionalidades específicas en orden jerárquico o de prioridad por los nodos creados en n8n.

3.3.2 Cuadro de carga operativa

Uno de los aspectos más relevantes que se evidenció durante este trabajo fue la carga operativa requerida para desplegar ambientes de prueba antes y después de la automatización. Este análisis, además de ser técnico, nos ayuda a entender cómo los cambios aplicados impactan en la eficiencia del proceso desde una perspectiva humana y operativa. Antes de la automatización, desplegar un ambiente para proyectos complejos representaba una tarea ardua, que demandaba mucho tiempo y la participación constante del personal de despliegue y QA. Por ejemplo, los proyectos catalogados como “Altos” alcanzaban el nivel máximo de carga (5), reflejando el esfuerzo crítico que se necesitaba para completar el flujo. Esta situación no solo consumía tiempo, sino que también elevaba el

riesgo de errores y cuellos de botella. Con la implementación de una arquitectura automatizada y flexible, esta carga operativa disminuyó considerablemente. En proyectos de alta complejidad, el esfuerzo se redujo de 5 a 3, y en los proyectos de tipo medio y bajo, los valores bajaron aún más, llegando incluso a 1 en el mejor de los casos. Esto quiere decir que, hoy en día, gracias a la automatización y a herramientas como el orquestador n8n, el personal puede enfocarse más en otras tareas como el análisis de calidad y pruebas. Esta mejora se alinea directamente con el objetivo específico 2, que plantea la necesidad de identificar oportunidades de mejora en la preparación de ambientes, especialmente en lo que se refiere a la carga operativa. También guarda estrecha relación con el objetivo 3, ya que demuestra que la arquitectura automatizada no solo funciona, sino que también permite una gestión más eficiente con menor intervención humana.

Tabla IV. Comparativa de cargas operativa

Tipo de Proyecto	Carga operativa antes (escala 1 a 5)	Carga operativa después (escala 1 a 5)
Alto	5	3
Medio	4	2
Bajo	3	1

3.3.3 Cuadro de errores operativos

Antes de implementar la arquitectura automatizada, el proceso de despliegue presentaba una alta probabilidad de fallos operativos. Era común enfrentar errores como seleccionar versiones incorrectas, ejecutar pipelines de forma incompleta o tener fallas de configuración. Estos errores eran especialmente frecuentes en los proyectos más complejos, donde la cantidad de microservicios y pasos a seguir elevaban el riesgo de equivocaciones humanas. Con la automatización, este panorama cambió significativamente. Al incorporar validaciones automáticas y establecer un flujo estandarizado en las tareas de despliegue, se redujo considerablemente la necesidad de intervención manual. Este cambio trajo consigo una importante disminución en la frecuencia de errores, haciéndolo evidente incluso en proyectos de alta exigencia. La tabla presentada refleja claramente esta evolución. Por ejemplo, en los proyectos clasificados como “Altos”, los errores pasaron de ser de alta frecuencia a baja frecuencia. En los de tipo “Medio”, se mantuvieron en niveles bajos, y en los proyectos “Bajos”, donde ya eran mínimos, la automatización prácticamente eliminó los errores por completo. Este resultado no solo demuestra una mejora en la confiabilidad del proceso, sino que también respalda el cumplimiento de los objetivos específicos 2 y 3, al reducir la carga operativa y elevar la calidad del servicio. Además, fortalece el diseño de una arquitectura más robusta, adaptable y menos propensa a errores humanos.

Tabla V. Errores operativos

Tipo de Proyecto	Errores antes de automatización	Errores con automatización
Alto	Alta frecuencia	Baja frecuencia
Medio	Media frecuencia	Baja frecuencia

Bajo	Media-baja	Mínima

3.4 Evidencias gráficas del flujo implementado

A continuación, se presentan capturas del flujo implementado en N8N, donde se evidencia la integración con Redis, GitLab y Telegram.

Inicio del flujo automatizado desde Telegram: Donde se envía el mensaje estándar para el despliegue del microservicio o los microservicios.

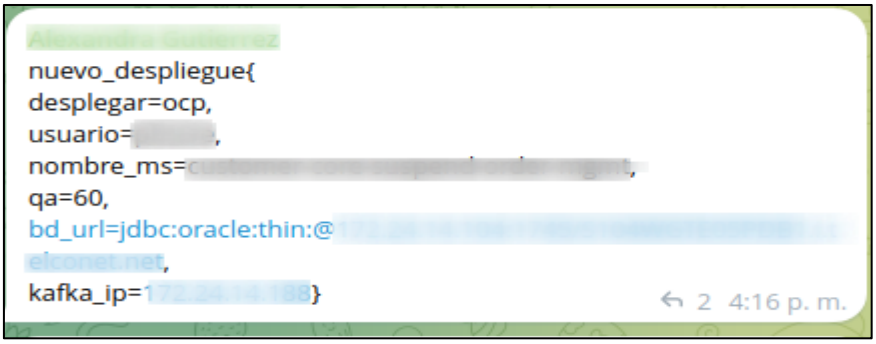

Figura 24. Evidencias gráficas del flujo implementado 1

Así mismo el Bot de telegram envía respuestas para validar información recibida y confirmación del despliegue.

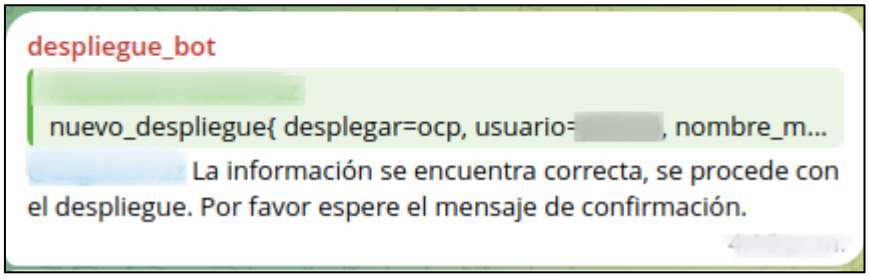

Figura 25. Evidencias gráficas del flujo implementado 2

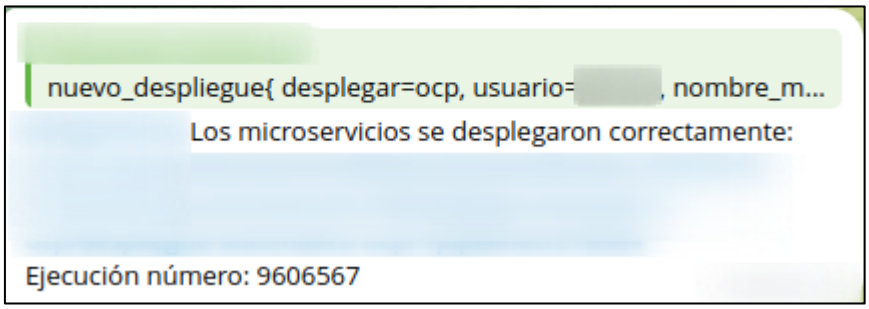

Figura 26. Evidencias gráficas del flujo implementado 3

Tiempo de despliegue del microservicio:

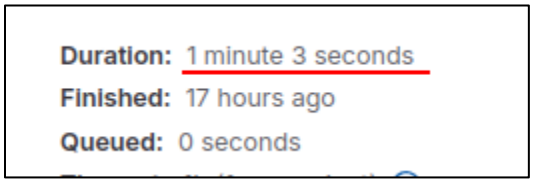

Figura 27. Evidencias gráficas del flujo implementado 4

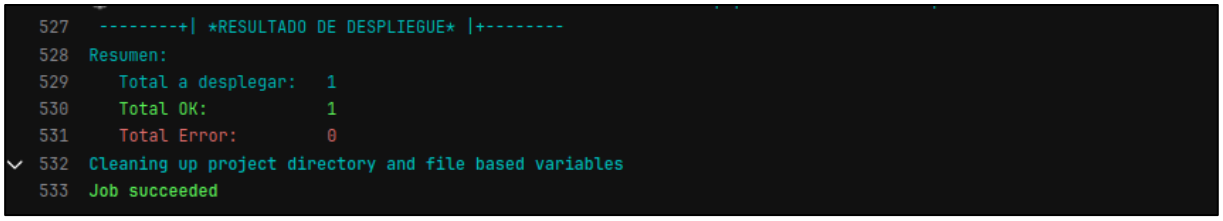

Figura 28. Evidencias gráficas del flujo implementado 5

3.5 Limitaciones

Durante la implementación se han identificado algunas limitaciones que deben considerarse:

Nuevas formas de mantenimiento: Al incluir múltiples herramientas, se debe considerar una estrategia clara de soporte y actualización.

Conectividad y permisos: Algunos procesos pueden depender de APIs externas, tokens o acceso a recursos limitados por políticas internas de cada empresa.

Sobrecarga operativa: En situaciones donde múltiples usuarios QA ejecutan despliegues simultáneos, existe el riesgo de sobrecargar los recursos del clúster de OpenShift (OCP). Esto puede provocar caídas generales que afectan a todos los proyectos levantados, impactando directamente en la disponibilidad del entorno y el rendimiento general del sistema.

Tiempos de despliegue variables: Dependiendo del tamaño o la complejidad del proyecto, el tiempo de despliegue puede incrementarse considerablemente. Esto puede generar cuellos de botella cuando otros proyectos deben esperar a que finalice un despliegue previo, afectando la eficiencia del flujo de trabajo en entornos con alta demanda de despliegues paralelos.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Se logró caracterizar detalladamente la situación actual del proceso de despliegue de ambientes de prueba en entornos QA, identificando los tiempos que demanda en el despliegue de los microservicios , los recursos involucrados y los principales cuellos de botella que afectan la eficiencia operativa.
- Uno de los principales obstáculos detectados fue la alta intervención manual requerida para la preparación de ambientes, especialmente ante la demanda de ambientes simultáneos. Esto generaba sobrecarga en el personal técnico y afectaba los tiempos de entrega y la calidad de las pruebas.
- A partir del análisis, se evidenció que la automatización del proceso a través de herramientas como orquestadores, pipelines y servicios de mensajería reduce significativamente la intervención humana, agiliza la ejecución y mejora la confiabilidad del sistema.
- Gracias a la automatización, el tiempo total de despliegue se redujo de 2 a 3 días a solo horas(h), incluso en proyectos de alta complejidad. Este resultado evidencia una mejora concreta en el cumplimiento del objetivo 2 (eficiencia operativa) y objetivo 3 (diseño de una arquitectura automatizada y adaptable).
- La disminución de la carga operativa permite que el personal de despliegue y QA se enfoque en tareas de mayor valor, como la supervisión y soporte de los servidores, liberando tiempo y recursos.
- Se comprobó que el modelo propuesto es escalable y se adapta a diferentes niveles de complejidad, permitiendo una gestión más autónoma del despliegue bajo demanda, lo cual responde directamente al objetivo 4.
- Finalmente, el enfoque planteado sienta una base sólida para continuar optimizando los procesos QA, orientándose hacia una solución más sostenible, eficiente y alineada con las necesidades reales del entorno operativo.

4.2 Recomendaciones

Tras culminar lo planificado en la propuesta, se proponen las siguientes recomendaciones clave para continuar fortaleciendo el modelo automatizado de despliegue:

- Implementar un sistema de métricas y monitoreo desde el inicio, para registrar los tiempos de despliegue, los errores más comunes y las cargas operativas. Esto permitirá evaluar de manera objetiva el impacto real de la automatización y detectar oportunidades de mejora.
- Realizar pruebas piloto controladas aplicando los flujos automatizados en proyectos de baja complejidad. Esto permitirá validar el diseño de forma segura antes de escalar su aplicación a proyectos más complejos.
- Escalar progresivamente la automatización, comenzando por los microservicios más estables y extendiéndola paulatinamente a componentes más críticos a medida que las integraciones se consoliden.
- Documentar claramente cada etapa del proceso automatizado, con versiones controladas y adaptaciones registradas. Esto facilitará la mantenibilidad del sistema y apoyará en la formación de nuevos integrantes del equipo.
- Capacitar al personal involucrado, especialmente a QA y despliegue, mediante talleres prácticos sobre el uso de las herramientas integradas y los nuevos flujos, para asegurar una adopción efectiva del modelo.
- Fomentar la mejora continua explorando la integración de herramientas de inteligencia artificial (IA) que sugieran ambientes óptimos según la complejidad del proyecto o los cambios en el código.
- Permitir el escalamiento horizontal habilitando el envío de múltiples peticiones simultáneas desde QA, sin necesidad de esperar que finalice un flujo. Esto es clave para optimizar el tiempo de despliegue cuando varios QA trabajan en paralelo, aprovechando la capacidad de OpenShift y la orquestación del flujo con n8n.

Referencias

[1] E. Sarmiento-Calisaya, A. Mamani-Aliaga, y J. C. S. D. P. Leite, «Introducing Computer Science Undergraduate Students to DevOps Technologies from Software Engineering Fundamentals», en *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, en ICSE-SEET '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 348–358. doi: 10.1145/3639474.3640071.

[2] S. M. Saleh, I. M. Sayem, N. Madhavji, y J. Steinbacher, "Advancing Software Security and Reliability in Cloud Platforms through AI-based Anomaly Detection," en *Proceedings of the 2024 on Cloud Computing Security Workshop*, CCSW '24. New York, NY, USA: Association for Computing Machinery, 2024.

[3] S. Smith, E. Robinson, T. Frederiksen, T. Stevens, T. Cerny, M. Bures, y D. Taibi, "Benchmarks for End-to-End Microservices Testing," arXiv, 2023. [En línea]. Disponible: https://arxiv.org/abs/2306.05895

[4] M. Waseem, P. Liang, M. Shahin, A. Di Salle, y G. Márquez, "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective," arXiv, 2021. [En línea]. Disponible: https://arxiv.org/abs/2108.03384

[5] R. Agrawal, P. Banerjee, M. Harman, y A. Groce, "An Industrial Case Study on DevOps Pipeline Bottlenecks," en *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Madrid, España, 2021, pp. 1–10. doi: 10.1109/ICSE-SEIP52600.2021.00006.

[6] R. S. Pressman and B. Maxim, Software Engineering: A Practitioner's Approach, 8th ed., New York, NY, USA: McGraw-Hill, 2014.

[7] I. Sommerville, Software Engineering, 10th ed., Boston, MA, USA: Pearson, 2015.

[8] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective," arXiv preprint, arXiv:2108.03384, 2021.

[9] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective," arXiv preprint, arXiv:2108.03384, 2021.

[10] R. Agrawal, P. Banerjee, M. Harman, and A. Groce, "An Industrial Case Study on DevOps Pipeline Bottlenecks," in 2021 IEEE/ACM 43rd Int. Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP), Madrid, Spain, 2021, pp. 1–10.

[11] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston, MA, USA: Addison-Wesley, 2010.

[12] J. García-Galán, J. García-Alonso, J. Berrocal, and J. M. Murillo, "A Review of Architectures and Platforms for Edge Computing," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–36, Oct. 2021.

[13] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective," *arXiv preprint*, arXiv:2108.03384, 2021.

[14] D. Taibi, L. Lavazza, and S. Janes, "Microservices in Practice: A Survey Study," *Journal of Systems and Software*, vol. 180, p. 111018, Jul. 2021.

[15] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," *IEEE Cloud Computing*, vol. 8, no. 2, pp. 22–32, Mar./Apr. 2021.

[16] D. Taibi and V. Lenarduzzi, "On the Definition of Microservice Bad Smells," *IEEE Software*, vol. 38, no. 1, pp. 56–62, Jan. 2021.

[17] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, “Design, Monitoring, and Testing of Microservices Systems: The Practitioners’ Perspective,” *arXiv preprint*, arXiv:2108.03384, 2021.

[18] D. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” *IEEE Cloud Computing*, vol. 8, no. 2, pp. 22–32, Mar./Apr. 2021.

[19] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, “Design, Monitoring, and Testing of Microservices Systems: The Practitioners’ Perspective,” *arXiv preprint*, arXiv:2108.03384, 2021.

[20] A. Sampaio, T. Batista, and S. Reinehr, “Microservices in Agile Software Development: A Systematic Mapping Study,” in *Proc. 20th International Conference on Software Technologies (ICSOFT)*, 2023, pp. 203–210.

[21] S. Smith et al., “Benchmarks for End-to-End Microservices Testing,” *arXiv preprint*, arXiv:2306.05895, 2023.

[22] T. Cerny, M. Bures, and D. Taibi, “Testing Microservices: A Survey of Challenges and Practices,” in *Proc. 37th ACM/SIGAPP Symposium on Applied Computing (SAC)*, 2022, pp. 1278–1285.

[23] R. Sigurbjörnsson and D. Garlan, “Enabling Observability for Microservice-Based Systems Through Automated Model Extraction,” in *Proc. 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2020, pp. 123–129.

[24] M. Wiedemann, M. Wiesche, and H. Krcmar, “Understanding the Influence of DevOps Practices on Software Development Performance,” Journal of Systems and Software, vol. 178, pp. 110980, 2021.

[25] T. Dingsøyr, F. Fægri, and H. Kulkarni, “Continuous Delivery and DevOps: Experiences from the Trenches,” IEEE Software, vol. 38, no. 1, pp. 50–57, Jan.–Feb. 2021.

[26] J. Ordóñez and M. Vera, “Automatización de Procesos DevOps para el Despliegue de Servicios en Entornos de Red 5G,” Revista Politécnica, vol. 49, no. 1, pp. 55–64, 2023.

[27] D. Gruen and N. Forsgren, “The State of DevOps Report,” *Google Cloud and DORA*, 2021.

[28] H. Shafiq, A. M. Khan, and S. Mahmood, “DevOps-Driven Continuous Integration and Continuous Delivery Pipeline for Microservices: Industrial Case Study,” *IEEE Access*, vol. 9, pp. 129256–129269, 2021.

[29] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2021.

[30] M. García-Valls, T. Cucinotta, and C. Lu, “Challenges in Real-Time Cloud Infrastructures: A Survey,” Journal of Systems Architecture, vol. 122, p. 102291, 2021.

[31] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps, IT Revolution, 2021.

[32] S. Ahmad and A. Bhatti, “Open Source Workflow Automation Tools for Modern DevOps: A Comparative Study,” IEEE International Conference on Automation, Control and Smart Systems (ICACSS), pp. 101–106, 2021.

[33] L. M. Vargas et al., “Automated Incident Management in Telco Systems through Open-Source Orchestrators,” IEEE Latin America Transactions, vol. 20, no. 4, pp. 732–739, 2022.

[34] A. M. Salazar, J. A. Peña, and F. García, “Messaging APIs as Integration Interfaces in DevOps Environments: An Empirical Analysis,” IEEE Latin America Transactions, vol. 19, no. 5, pp. 821–829, 2021.

[35] R. Rodrigues and M. Oliveira, “Orchestrating Automated Workflows with Messaging Applications: A Case Study on Telegram Bots,” Journal of Systems Integration, vol. 14, no. 2, pp. 45–53, 2023.

[36] T. Cerny, M. Bures, and D. Taibi, “Microservices: A Systematic Mapping Study on Orchestration,” ACM Computing Surveys, vol. 54, no. 12, pp. 1–38, Dec. 2021.

[37] J. R. López and M. Andrade, “Automated Test Environment Provisioning Using Workflow Orchestration: A Telecom Case Study,” IEEE Access, vol. 11, pp. 51234–51245, 2023.

[38] L. N. Mohammed et al., “Evaluation of Open Source Tools for CI/CD Automation in DevOps,” Journal of Systems and Software, vol. 190, p. 111351, Jan. 2022.

[39] M. Álvarez, R. Rosero, and D. Salinas, “Evaluación de herramientas DevOps de código abierto para la gestión automatizada de microservicios en telecomunicaciones,” Revista I+D Tecnológico, vol. 18, no. 2, pp. 23–31, 2022.

Anexos

Anexo 1. Cuestionario de Evaluación del Prototipo de Automatización

Indicaciones:

A continuación, se presenta el cuestionario aplicado a los participantes del estudio. La encuesta fue estructurada en dos bloques: antes y después de la implementación de la solución automatizada. Las preguntas utilizaron una escala de Likert.

Parte A: Situación antes del desarrollo

1. ¿Cuánto tiempo en promedio toma levantar un ambiente de prueba completo (en días)?
2. ¿Qué nivel de carga operativa percibes al realizar los despliegues?
3. ¿Con qué frecuencia se producen errores durante el proceso de despliegue?
4. ¿Qué nivel de satisfacción tienes con el proceso actual de despliegue?
5. ¿Con qué frecuencia experimentas retrasos en los cronogramas debido al tiempo requerido para el despliegue?

Parte B: Situación después del desarrollo

6. ¿Cuánto tiempo en promedio toma levantar un ambiente de prueba completo (en días)?
7. ¿Qué nivel de carga operativa percibes al realizar los despliegues?
8. ¿Con qué frecuencia se producen errores durante el proceso de despliegue?
9. ¿Qué nivel de satisfacción tienes con el proceso actual de despliegue?
10. ¿Con qué frecuencia experimentas retrasos en los cronogramas debido al tiempo requerido para el despliegue?