Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Maestría en Tecnologías de la Información

Trabajo Final

“Adaptación de GitOps: Guía para la creación de repositorios de despliegue en proyectos Web aplicado a equipos pequeños de desarrollo de Software”

Autor: Jesús Andrés Zini

Director: Emanuel Irrazábal

Año 2025

Dedicado a:

A mis compañeros de grupo durante la Maestría y Especialización en Tecnologías de la Información. En especial a nuestro querido compañero y amigo Sergio Navarro, quien nos dejó durante el transcurso de este camino compartido.

Resumen

Este trabajo presenta una prueba de concepto de una guía para la creación de repositorios de despliegue de aplicaciones Web en equipos pequeños de desarrollo de Software. Aborda desafíos comunes en el proceso de despliegue, como la falta de automatización y documentación, y propone una solución basada en GitOps. La metodología incluye la implementación de un repositorio dedicado para almacenar todos los archivos, configuraciones y documentación relacionados con el despliegue de cada aplicación.

La investigación se enfoca en equipos pequeños, típicos de la región del NEA (Noreste Argentino), y utiliza encuestas para identificar problemas y prácticas actuales. Se desarrolla de manera iterativa e incremental una guía para la creación de este repositorio colaborativo, aplicable a proyectos Web de diversas tecnologías.

El procedimiento se valida y mejora a través de su implementación en un equipo de trabajo de una empresa regional de la industria del software.

Agradecimientos

Este trabajo se ha llevado a cabo en el marco de la empresa Devlights SRL, y quiero expresar mi agradecimiento a todos sus integrantes y compañeros de trabajo que han contribuido de diversas formas. Agradezco especialmente a aquellos que generosamente dedicaron su tiempo para responder los cuestionarios y utilizaron la solución propuesta en este trabajo.

Índice de Contenidos

Índice de figuras

Índice de tablas

Glosario

Framework: o marco de trabajo, es un conjunto de herramientas predefinidas que facilita el desarrollo de software proporcionando funcionalidades comunes y una estructura para construir aplicaciones.

Stack tecnológico: conjunto de herramientas, tecnologías, lenguajes de programación y frameworks específicos utilizados para desarrollar y ejecutar aplicaciones de software.

Contenerización: proceso de encapsular una aplicación junto con sus dependencias y configuraciones en un contenedor.

KB: una base de conocimiento (o KB, por sus siglas en inglés "Knowledge Base") es una colección organizada de información, datos y recursos que se utiliza para almacenar y compartir conocimiento dentro de una organización o comunidad.

Desarrolladores freelancer: profesional independiente que ofrece sus servicios de desarrollo de software a través de contratos temporales o proyectos específicos.

Aplicación Web monolítica: aplicación Web construida como una sola entidad.

Bash Script: secuencia de comandos para automatizar tareas en sistemas operativos Unix.

CI/CD: acrónimo de Integración Continua e Implementación Continua. Es una práctica de desarrollo de software que se centra en automatizar y agilizar el proceso de entrega de software.

DevOps: conjunto de prácticas, herramientas y filosofía cultural que sirve para automatizar e integrar los procesos que comparten el equipo de desarrollo de software y el de TI.

GitOps: enfoque que usa Git como fuente única de verdad para gestionar infraestructura y despliegues, automatizando la sincronización del entorno con su configuración almacenada.

API: acrónimo de Interfaz de Programación de Aplicaciones. Es un conjunto de reglas y definiciones que permite que diferentes aplicaciones se comuniquen entre sí.

Pipeline: secuencia automatizada de procesos que se ejecutan en orden para realizar tareas específicas.

Backup: copia de seguridad de datos que se realiza con el propósito de restaurar los datos en caso de pérdida.

Proxy: intermediario entre un cliente y un servidor que actúa como un punto de acceso para los clientes que desean acceder a recursos en línea.

Docker: plataforma para empaquetar y ejecutar aplicaciones en diferentes entornos.

Container Registry: servicio o repositorio destinado al almacenamiento, gestión y distribución de imágenes de Docker u otros contenedores.

Frontend: aplicación cliente que interactúa con el usuario.

Backend: aplicación servidor que procesa y almacena datos.

Fullstack: hace referencia a una arquitectura o enfoque de desarrollo que abarca tanto el frontend como el backend de una aplicación o sistema.

Git: herramienta de control de versiones para el desarrollo colaborativo de software.

IaC: Infraestructura como código; enfoque para gestionar la infraestructura de TI como si fuera software.

Microservicios: enfoque arquitectónico y organizativo que utiliza pequeños servicios independientes para la construcción de una aplicación software.

Monolito: aplicación construida como una única unidad, donde todos los componentes están integrados en un solo programa.

Serverless: modelo de computación en la nube que permite a los desarrolladores crear y ejecutar aplicaciones sin administrar la infraestructura subyacente.

SSL: abreviatura de *Secure Sockets Layer* (capa de sockets seguros), un protocolo para navegadores y servidores Web que permite autenticar, cifrar y descifrar la información enviada a través de Internet.

VPS: abreviatura de *Virtual Private Server* (servidor privado virtual), es una máquina que aloja todo el software y los datos necesarios para ejecutar una aplicación o un sitio Web.

Self-Hosted: o auto alojado; es la práctica de alojar y administrar una aplicación o servicio en infraestructura propia en lugar de en servicios externos.

Monorepositorio: un solo repositorio que contiene todos los proyectos y componentes relacionados de un sistema de software en un solo lugar.

Multirepositorio: múltiples repositorios de control de versiones que contienen proyectos o componentes separados de un sistema de software.

Pull Request: solicitud para revisar y fusionar cambios en el código en sistemas de control de versiones.

NLP: rama de la inteligencia artificial que se encarga del análisis y la interpretación del lenguaje humano, permitiendo a las computadoras comprender, procesar y generar texto o habla de manera natural.

CLI: abreviatura de *Command Line Interface* (Interfaz de línea de comandos), es una interfaz de usuario que permite interactuar con un sistema o aplicación mediante comandos de texto.

QA: proceso de asegurar la calidad del software mediante pruebas para verificar que cumple con los estándares establecidos.

Scheduler: componente del sistema operativo que gestiona y programa la ejecución de tareas o procesos.

Commit: registro de cambios en un sistema de control de versiones, como Git, que captura el estado del código en un momento específico.

Multicloud: estrategia de uso de múltiples proveedores de servicios en la nube para distribuir cargas de trabajo, mejorar la redundancia.

Toil: trabajo repetitivo y manual en operaciones de TI que no aporta valor directo y debería automatizarse

Feedback: retroalimentación o respuesta sobre una acción, producto o desempeño, utilizada para mejorar o ajustar futuras decisiones o procesos.

1. Introducción

Este capítulo introduce la investigación que se llevará a cabo, presentando la fundamentación del trabajo final integrador y delineando sus objetivos generales y específicos. Principalmente se abordan los desafíos asociados al despliegue de aplicaciones Web, junto con las soluciones propuestas.

1.1 Fundamentación

En los equipos de desarrollo de software, uno de los procesos clave en el ciclo de vida del proyecto es el despliegue de las aplicaciones Web a Internet; ya sea para entornos de desarrollo, prueba, o producción, así como su transición entre estos entornos [1], como ilustra la Fig. 1.

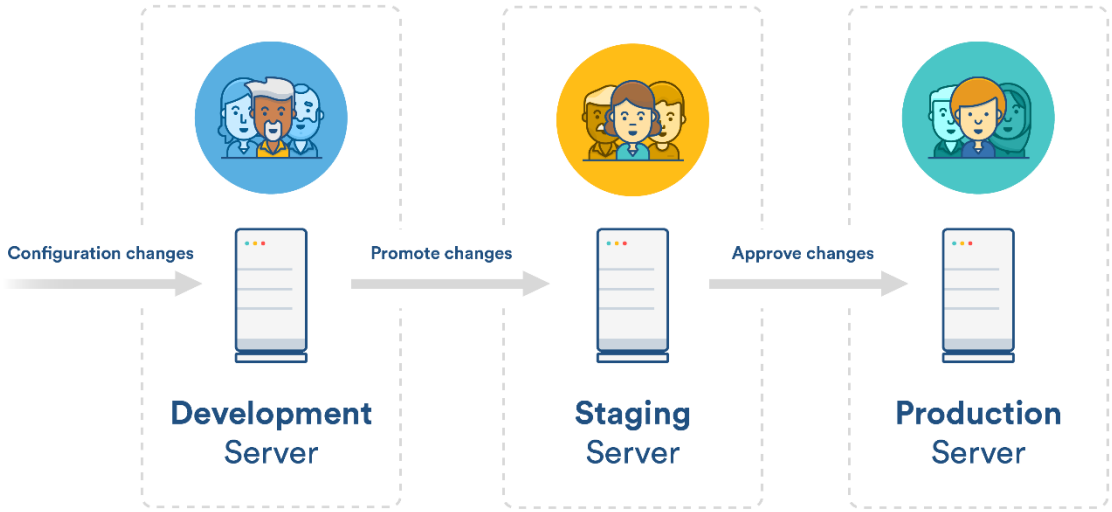

Fig. 1: Entornos de despliegue [2].

En la actualidad, el vertiginoso avance de la tecnología exige entregas de software más rápidas y frecuentes. Para adaptarse a esta demanda, las aplicaciones deben ser capaces de ejecutarse en diversos entornos operativos sin fricciones. En este contexto, las metodologías ágiles han impulsado la adopción de la Integración Continua y la Entrega Continua (CI/CD, por sus siglas en inglés) [3], que automatizan las pruebas, la construcción y el despliegue del software, así como el uso de contenedores [4], una solución clave que

garantiza la portabilidad, el control de versiones y la coherencia entre entornos, facilitando un desarrollo más ágil y eficiente.

A pesar de estos avances, muchas empresas aún enfrentan desafíos significativos en el despliegue del software, lo que puede generar retrasos, inconvenientes e incluso costos adicionales. La elección del entorno de despliegue es un factor clave, ya que puede influir directamente en el costo final que la empresa o el cliente deberán asumir por el alojamiento de las aplicaciones Web.

Uno de los obstáculos más comunes es que, aunque muchos desarrolladores poseen un alto nivel de experiencia en la construcción del código fuente, no siempre cuentan con los conocimientos necesarios para desplegar una aplicación de manera efectiva. Esta falta de experiencia puede dificultar la integración de nuevas funcionalidades y convertirse en una barrera recurrente en el flujo de trabajo. Como resultado, estos desarrolladores suelen depender de otros miembros del equipo, como administradores de sistemas o equipos de operaciones, ya sea para recibir orientación o para delegarles por completo el proceso de despliegue. Esta división de roles entre los integrantes de un proyecto se aborda en el libro “Site Reliability Engineer” [5].

> *El rol de administrador de sistemas requiere un conjunto de habilidades marcadamente diferente al que se requiere de los desarrolladores de un producto, los desarrolladores y los administradores de sistemas se dividen en equipos discretos: ‘desarrollo’ y ‘operaciones’.*

Sin embargo, en empresas pequeñas o en aquellas compuestas por equipos reducidos, como es común en la región del NEA (Noreste de Argentina), no siempre es económicamente viable contratar personal con habilidades especializadas para un único rol, como el equipo de Operaciones mencionado anteriormente. En estos casos, es frecuente e ideal, desde el punto de vista económico, contar con equipos multidisciplinarios que puedan abordar diversas tareas de manera eficiente y colaborativa [6].

Para obtener una comprensión más detallada de estos aspectos, se realizó una breve encuesta conversacional dirigida a equipos de empresas del NEA y desarrolladores freelance, enfocada en los problemas que enfrentan en el despliegue de sus proyectos. Después de analizar las respuestas y mantener charlas con los encuestados, se identificaron principalmente las siguientes inquietudes y desafíos que los equipos deben abordar a diario:

- ¿Dónde y cómo realizar el despliegue del proyecto?
- ¿Cómo configurar el servidor con todas las dependencias necesarias del proyecto?

- ¿Cómo implementar medidas de seguridad en la capa de transporte (SSL por sus siglas en inglés)?
- ¿Cómo desplegar una nueva versión de la aplicación después de realizar actualizaciones?

Según lo indicado por estos equipos, estas cuestiones representan un desafío por diversos motivos, tales como:

- El proceso de despliegue de una aplicación Web, ya sea para su lanzamiento inicial o para desplegar una nueva versión, a menudo se ve retrasado por un aumento significativo en el tiempo necesario (que en promedio varía entre dos y tres horas).
- Es común necesitar la ayuda y el asesoramiento de otros desarrolladores con experiencia o conocimientos especializados, lo que genera una dependencia de su disponibilidad para llevar a cabo el proceso de despliegue.
- Frecuentemente se recurren a alternativas de despliegue que no son las más adecuadas para el contexto, lo que provoca un aumento en los costos y la pérdida de oportunidades para aprovechar recursos existentes en la empresa, como servidores privados virtuales ya en funcionamiento.
- Las fallas en la configuración de seguridad durante el despliegue pueden exponer la aplicación a vulnerabilidades. Errores como dejar puertos abiertos innecesariamente, no restringir el acceso a ciertos entornos o filtración de credenciales pueden comprometer la seguridad del sistema y la integridad de los datos.
- La elección del lugar de despliegue puede tener un impacto significativo en el costo final. Decidir dónde desplegar una aplicación Web no solo afecta la accesibilidad y el rendimiento del sistema, sino también los costos asociados. Optar por una infraestructura inadecuada puede generar gastos innecesarios, ya sea por el uso excesivo de recursos en plataformas costosas o por no aprovechar recursos disponibles internamente.

1.1.1 Realización del Trabajo Final de Especialización en Tecnologías de la Información

Para abordar las problemáticas mencionadas previamente, el Trabajo Final de la Especialización en Tecnologías de la Información [7] (abreviado TFE) se centró en la creación de un Repositorio Base de Despliegue. Este repositorio reúne los archivos y configuraciones necesarias para simplificar y optimizar el proceso de despliegue de

aplicaciones Web monolíticas. Al integrarlo en sus proyectos, los equipos pueden automatizar el proceso de despliegue mediante la contenerización de aplicaciones y la automatización de procesos asociados.

1.1.1.1 Investigación Sobre el Despliegue en Equipos Pequeños

En el proceso, se llevó a cabo una investigación sobre las diversas prácticas de despliegue en equipos de pequeñas y medianas empresas de la región. Entre las metodologías y enfoques identificados, se destacan:

1.1.1.1.1 Despliegue de Aplicaciones Web Realizado Solo con Git

Un enfoque clásico, aún ampliamente utilizado, para el despliegue de aplicaciones Web consiste en desarrollar el proyecto de manera local y, posteriormente, utilizar herramientas de control de versiones, como Git [8], para clonarlo en un servidor remoto, donde se configura y ejecuta. Este proceso se ilustra en la Fig. 2.

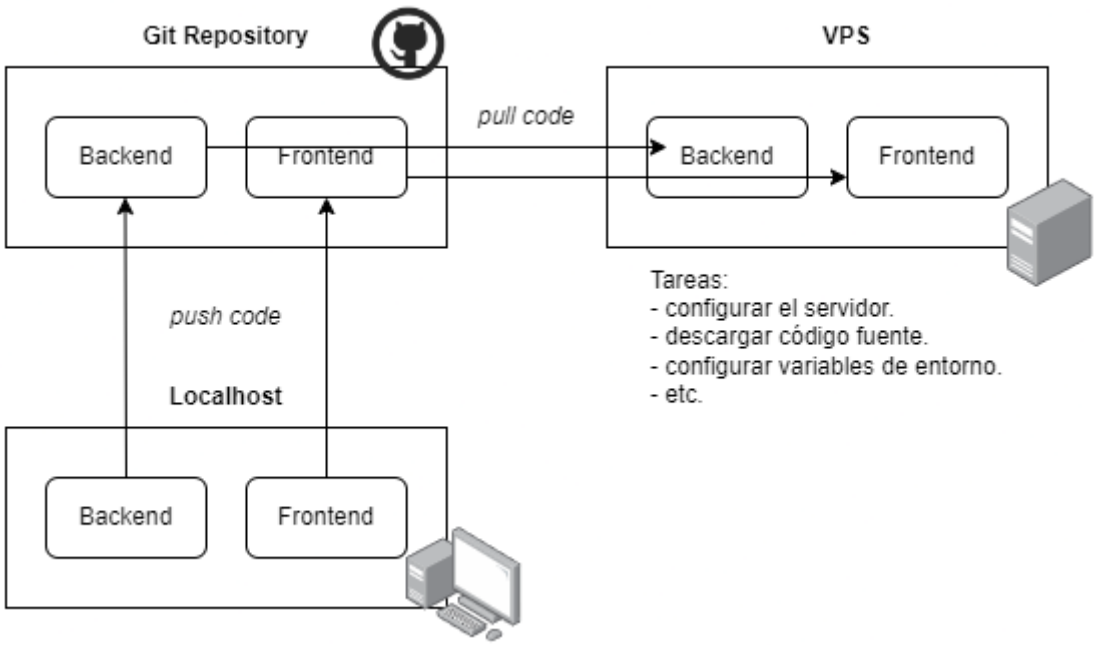

Fig. 2*: Enfoque clásico de despliegue de aplicaciones Web monolíticas a VPS.*

Si bien este método es funcional, desaprovecha herramientas más modernas, como los contenedores y los servicios de integración y despliegue continuo, que permiten una mejor optimización de recursos y la automatización de procesos. Según [9], algunas desventajas que trae consigo el despliegue de aplicaciones Web realizado de forma tradicional son:

- Es necesario instalar manualmente los programas y dependencias requeridos para la ejecución del proyecto en el servidor.
- Se debe transferir todo el código fuente de la aplicación al servidor.
- Una vez que el código está en el servidor, es necesario configurar el proyecto, incluyendo la definición de variables de entorno, entre otros ajustes.
- Migrar el proyecto a otro servidor, como ser de QA o producción, resulta complejo, ya que implica repetir todo el proceso desde cero.
- Se dificulta el seguimiento de los programas y versiones instalados en el servidor, a no ser que estén debidamente documentados.
- Desplegar múltiples aplicaciones en un mismo servidor puede generar incompatibilidades, como conflictos entre versiones del entorno de ejecución o problemas en la asignación de puertos.

1.1.1.1.2 Despliegue de Aplicaciones Web Utilizando Contenedores

El despliegue de aplicaciones Web requiere un entorno de tiempo de ejecución que incluya: el servidor Web lógico, los motores de ejecución del lenguaje, las bases de datos y otras dependencias; pero no necesariamente un sistema operativo completo. Una solución a esta cuestión es la creación de un conjunto de estos elementos de software agrupados en contenedores, los cuales incluyen todo lo necesario para ejecutar la aplicación. Existen diversas herramientas que facilitan la construcción de estos contenedores, siendo Docker [10] una de las más conocidas. Su uso puede resultar útil para resolver algunos de los problemas mencionados previamente, como se señala en [11]:

> *La portabilidad y reproducibilidad de un proceso/programa en contenedores brinda la oportunidad de mover y escalar nuestras aplicaciones a través de nubes y centros de datos. Los contenedores garantizan de manera efectiva que esas aplicaciones se ejecuten de la misma manera en cualquier lugar, lo que nos permite aprovechar rápida y fácilmente todos estos entornos.*

No obstante, el uso de contenedores en un servidor privado virtual (VPS, por sus siglas en inglés) no resuelve por sí solo los problemas asociados al despliegue de aplicaciones en estos entornos. Si bien representa una mejora significativa en términos de optimización de recursos y automatización de procesos, su implementación aún presenta desafíos. Sobre todo en equipos pequeños, no existe o no está bien definido un estándar para llevar a cabo el despliegue en servidores utilizando tecnología de contenedores. Esto suele derivar en la adopción de malas prácticas, como ser:

- Clonar el código fuente del proyecto en el VPS, aun cuando esto ya no es necesario gracias al uso de un Container Registry.
- Construir la imagen del contenedor directamente en el servidor en lugar de hacerlo en una máquina local o a través de un pipeline de integración continua.
- No incluir dentro del conjunto de contenedores alguna parte esencial de la aplicación, como el servidor Web, el Proxy inverso o la base de datos.
- Sobrecarga del servidor en memoria debido a la construcción de imágenes o a la codificación de archivos de configuración dentro del propio servidor.
- Sobrecarga del servidor en almacenamiento debido a no eliminar contenedores huérfanos o a clonar el código fuente de múltiples proyectos en el mismo servidor.

1.1.1.1.3 Despliegue de Aplicaciones Web Utilizando Integración y Despliegue Continuo

Para automatizar los procesos de desarrollo y despliegue de software, se emplean estrategias y tecnologías de Integración y Despliegue Continuo. Según [3]:

> *La integración continua (CI) es una práctica que permite incorporar automáticamente los cambios de código en un repositorio compartido de forma periódica. Por su parte, la distribución continua e implementación continua (CD) conforman un proceso en dos etapas: primero, los cambios de código se integran, prueban y distribuyen; luego, dependiendo del enfoque, pueden o no desplegarse automáticamente en el entorno de producción. Mientras que en la distribución continua los cambios no se implementan de forma automatizada en producción, en la implementación continua sí se despliegan automáticamente una vez superadas las pruebas.*

El diagrama de la Fig. 3 muestra el flujo de trabajo a través de CI/CD.

Fig. 3: Flujo de trabajo utilizando CI/CD. Fuente: [3]

Entre las ventajas que estas estrategias ofrecen, se destacan las siguientes:

- Mejora en la calidad del software**:** La integración de pruebas y análisis estáticos del código fuente asegura un producto final de mayor calidad.
- Entrega más rápida**:** La automatización de procesos acelera tanto el ciclo de desarrollo como el de entrega.
- Menor riesgo**:** La detección temprana de errores y la entrega frecuente disminuyen la probabilidad de problemas graves en producción.
- Fomenta la colaboración entre equipos**:** La automatización de procesos y la visibilidad continua del estado del software promueven la colaboración entre los equipos de desarrollo, pruebas y operaciones, contribuyendo a la cultura DevOps.

Hoy en día, es común utilizar herramientas de CI/CD que se integran con sistemas de control de versiones como Git. Entre estas herramientas, se puede optar por soluciones auto alojadas (Self-Hosted) como Jenkins [12], o por servicios en la nube como GitHub Actions [13], Circle CI [14], Travis CI [15], entre otros.
Es importante tener en cuenta que, si bien CI/CD ofrece numerosos beneficios, su implementación puede presentar desafíos, especialmente en entornos donde esta metodología no ha sido adoptada previamente. Algunos de los desafíos más comunes incluyen la configuración inicial de los flujos de trabajo, la capacitación del equipo en nuevas herramientas y prácticas, y la gestión del cambio organizacional.

1.1.1.2 Creación del Repositorio Base de Despliegue y sus Limitaciones

Como se mencionó anteriormente, en el TFE se desarrolló un Repositorio Base de Despliegue que incluye archivos de configuración para la contenerización y automatización del despliegue en una VPS, diseñado para integrarse en diversos proyectos Web.
En comparación con las alternativas de despliegue mencionadas, este enfoque ofrece las siguientes ventajas:

- No es necesario transferir el código fuente del proyecto al servidor.
- Solo se requiere clonar el repositorio de despliegue en el servidor, ahorrando espacio y tiempo.
- No se requiere instalar programas o dependencias adicionales en el servidor, simplificando así su gestión.

- Los scripts de automatización permiten realizar el despliegue en cuestión de segundos.
- La migración entre servidores es más sencilla, ya que el repositorio de despliegue contiene todo lo necesario para el proceso.

La Fig. 4 ilustra el funcionamiento de este método de despliegue.

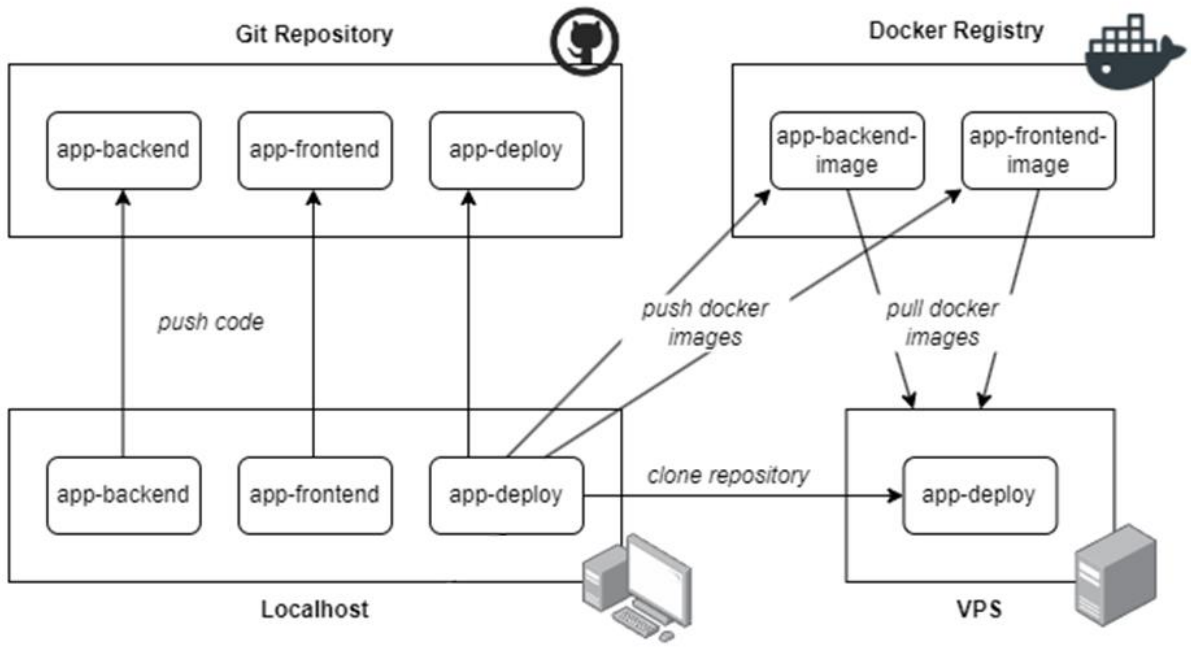

Fig. 4: Enfoque contenerizado de despliegue de aplicaciones Web monolíticas a VPS.

1.1.1.2.1 Limitaciones

A pesar de su practicidad y alineación con las filosofías GitOps [16] y DevOps, ampliamente adoptadas en equipos de desarrollo a gran escala, el concepto del repositorio de despliegue desarrollado en el TFE presenta algunas limitaciones.

Una de las principales desventajas es su enfoque especializado hacia un tipo específico de proyecto, en su mayoría aplicaciones fullstack construidas con Node.js [17], React.js [18] y PostgreSQL [19], debido a que este stack era el más utilizado en la empresa en ese momento. Esta orientación especializada dificulta la adaptación de la solución a otros stacks tecnológicos, limitando su aplicabilidad en entornos que emplean diferentes tecnologías.

Además, se observó que la imposición de la contenerización de la aplicación dentro del repositorio puede generar complicaciones en situaciones donde dicha práctica no sea estrictamente necesaria. Esta rigidez podría aumentar la complejidad en lugar de simplificarla.

Estas limitaciones subrayan la necesidad de evaluar cuidadosamente el alcance y la flexibilidad de la solución propuesta, así como la importancia de explorar alternativas que puedan adaptarse a una gama más amplia de casos de uso y tecnologías.

1.1.2 Conclusiones

El despliegue de aplicaciones Web es un aspecto fundamental en el desarrollo de software y puede tornarse un gran desafío para equipos pequeños, donde los recursos suelen ser limitados. En este capítulo, se han analizado tres enfoques principales de despliegue: el método tradicional mediante Git en una VPS, el uso de contenedores y la adopción de estrategias de integración y despliegue continuo (CI/CD).

El enfoque tradicional sigue siendo ampliamente utilizado debido a su simplicidad y accesibilidad. Sin embargo, presenta desafíos como la configuración manual del servidor y la falta de portabilidad. En contraste, los contenedores ofrecen una solución más flexible y eficiente, al facilitar la portabilidad y la replicabilidad del entorno de ejecución. No obstante, su implementación puede resultar compleja para equipos con poca experiencia en su gestión. Por otro lado, la adopción de CI/CD optimiza los ciclos de desarrollo y mejora la calidad del software, aunque su implementación puede implicar el uso de servicios externos y costos adicionales asociados a herramientas especializadas.

Cada una de estas estrategias tiene ventajas y desafíos, por lo que la elección del enfoque más adecuado dependerá de las necesidades y características de cada proyecto y equipo. En este trabajo, nos enfocaremos principalmente en el despliegue de aplicaciones en servidores VPS, sin limitar nuestra exploración a esta opción, ya que es una solución común entre equipos pequeños de desarrollo de software en nuestra región. No obstante, reconocemos que la solución previamente utilizada presenta limitaciones, como su orientación hacia un tipo específico de proyecto y la rigidez en la contenerización.

Por ello, este trabajo busca definir un método eficiente en términos de tiempo y recursos, que permita a los equipos desplegar aplicaciones de distintos tipos de manera organizada y documentada. A través de esta propuesta, se pretende ofrecer una solución flexible que se adapte a diversas tecnologías y necesidades.

1.2 Objetivo General

Desarrollar una estrategia para el despliegue de aplicaciones Web en servidores Linux y otros entornos, enfocada en equipos de desarrollo de software en pequeñas y medianas empresas, con el fin de fortalecer la gestión del conocimiento dentro de estos equipos

mediante la adopción de un repositorio centralizado de despliegue para cada proyecto, donde se almacenen archivos de configuración, scripts de automatización y documentación relevante. Además, se publicará en línea una guía para la creación y uso de estos repositorios, facilitando su adopción.

1.3 Objetivos Específicos

Para alcanzar el objetivo general definido en el apartado anterior, se han establecido los siguientes objetivos específicos que se indican a continuación.

1.3.1 (OE1)

Analizar las buenas prácticas de la industria del software que abordan los desafíos del despliegue de aplicaciones Web y seleccionar aquellas actividades que puedan aplicarse de manera efectiva en equipos pequeños de desarrollo.

1.3.2 (OE2)

Definir una estrategia de despliegue de aplicaciones Web para equipos pequeños basada en GitOps, estableciendo un repositorio centralizado que almacene todos los archivos relacionados con el proceso de despliegue, incluyendo configuraciones y documentación.

1.3.3 (OE3)

Investigar los métodos de despliegue más utilizados en la industria para distintos stacks tecnológicos, con el fin de elaborar una guía detallada para la creación de un repositorio de despliegue, que incluya los pasos clave, mejores prácticas y configuraciones recomendadas, adaptadas a cada tipo de proyecto.

1.3.4 (OE4)

Desarrollar la guía mencionada en el OE3 de forma iterativa e incremental, asegurando su disponibilidad pública en línea como un recurso de código abierto, de modo que pueda ser actualizado y mejorado continuamente por la comunidad de desarrolladores.

1.3.5 (OE5)

Evaluar y optimizar el procedimiento propuesto mediante su implementación en un equipo de desarrollo de una empresa de la industria del software de la región, identificando oportunidades de mejora.

2. Metodología

En este capítulo se presenta la metodología de desarrollo utilizada para la construcción del proyecto. Se optó por un modelo de ciclo de vida evolutivo debido a su flexibilidad y adaptabilidad, especialmente en proyectos con un alto grado de incertidumbre respecto a los requisitos finales del sistema. Este modelo permitió la entrega temprana de la guía para el armado del repositorio de despliegue y ofreció mayor flexibilidad en la incorporación de nuevas funcionalidades. Además, se describen las etapas del proceso iterativo aplicado a lo largo de la metodología.

2.1 Ciclo de Vida Evolutivo

Para la construcción del proyecto se empleó un modelo de ciclo de vida evolutivo, representado gráficamente en la Fig. 5. Esta estrategia ofrece mayor flexibilidad y adaptación a medida que se incorporan nuevas funcionalidades y características. Además, proporciona un entorno propicio para el desarrollo de software incremental [20] y resulta ideal para proyectos con un alto grado de incertidumbre en los requisitos finales del sistema, ya que permite la entrega temprana de una versión parcialmente funcional, la cual se mejora y evoluciona con base en la retroalimentación de los usuarios finales.

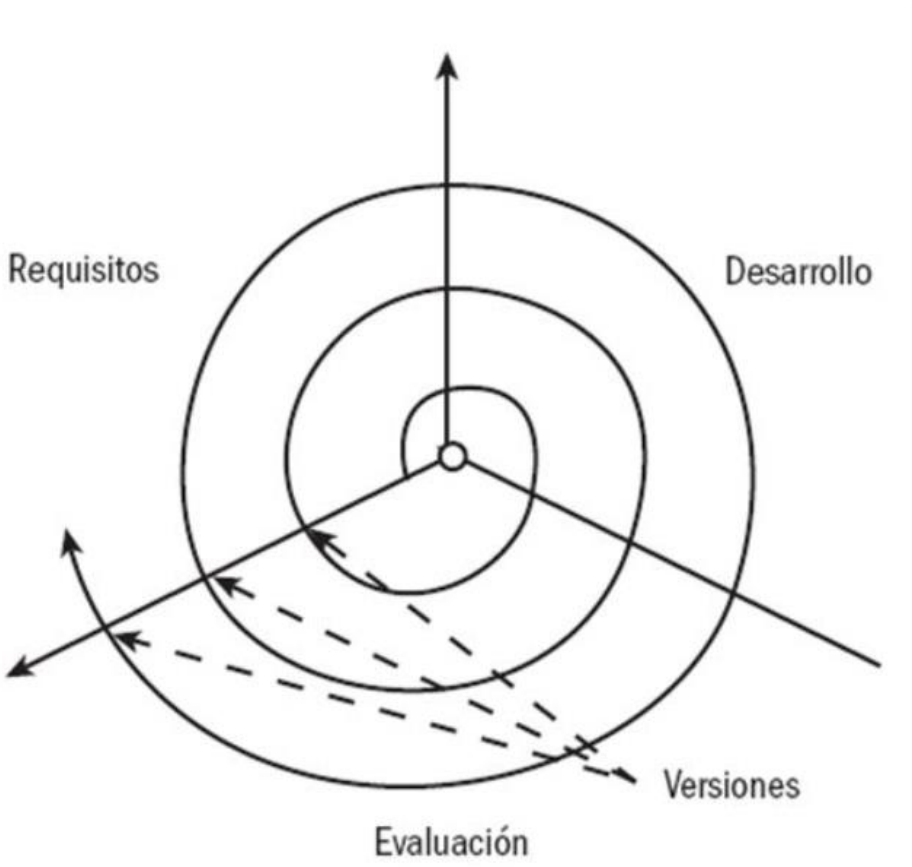

Fig. 5: Modelo de ciclo de vida evolutivo. Fuente: [20]

Por lo mencionado anteriormente se aplicó este modelo a través de un proceso iterativo que se detalla a continuación en la Tabla 1.

Tabla 1: Etapas de la metodología aplicada.

Etapa	Detalle
Iteración Nro. 1: Encuesta sobre el despliegue de aplicaciones Web	Se realizó una encuesta dirigida a equipos de desarrollo para conocer las tecnologías con las que trabajan, el rol de los desarrolladores en sus equipos, cómo están constituidos estos equipos y su conocimiento sobre el proceso de despliegue.
Iteración Nro. 2: Interpretación de los datos	Se analizaron los datos recopilados de la encuesta para entender las tecnologías utilizadas, las arquitecturas implementadas y las preferencias en cuanto al despliegue. Se identificó una amplia variedad de lenguajes, frameworks y bases de datos, información que fue considerada para la creación posterior de la guía.
Iteración Nro. 3: Diseño inicial de la guía	Se diseñó la guía utilizando Docusaurus para generar documentación accesible y GitHub como plataforma de alojamiento. La guía se estructuró en tres secciones: creación del repositorio de despliegue, ejemplos en distintos entornos, y utilidades adicionales.
Iteración Nro. 4: Creación de los repositorios plantilla	Se crearon repositorios de despliegue "plantilla" con configuraciones predefinidas y scripts diseñados para automatizar el proceso de despliegue de diversas tecnologías y stacks de desarrollo, los cuales fueron obtenidos a partir de los datos recopilados en la encuesta.
Iteración Nro. 5: Incorporación de estrategias de documentación	Se fortaleció la guía con estrategias de documentación eficientes, incluyendo documentación mínima al inicio, uso de documentación ejecutable y utilización de NLP para reducir la necesidad de documentación manual extensa.
Iteración Nro. 6: Últimos ajustes y mejoras del proyecto	Se realizaron ajustes en el contenido y diseño de la guía, optimizando la experiencia del usuario, corrigiendo errores y aplicando retroalimentación para mejorar la plataforma y su funcionalidad.
Iteración Nro. 7: Documentación y despliegue de la Guía	Se realizaron ajustes finales antes de la publicación de la guía, incluyendo la documentación del repositorio para permitir contribuciones mediante Pull Requests. Se eligió Cloudflare Pages para el alojamiento de archivos estáticos y se configuró la automatización del despliegue con GitHub.

2.1.1 Justificación de la Elección del Modelo Evolutivo

El modelo evolutivo es una opción adecuada cuando el proyecto tiene un alto grado de incertidumbre y cambios constantes. En el proceso de despliegue de aplicaciones Web en servidores Linux y en general, es común encontrarse con cambios en las configuraciones de los servidores, actualizaciones de software y otros ajustes que deben ser implementados. El modelo evolutivo permite hacer frente a estos cambios de manera más efectiva que otros modelos de ciclo de vida. Además, permite un desarrollo iterativo, facilitando una respuesta ágil a los cambios y mejorando la eficiencia del proceso.

Otra ventaja es su enfoque en la entrega continua y gradual del software, lo que facilita una retroalimentación constante y una validación temprana del producto. Esto permite detectar y corregir errores de manera oportuna, mejorando la calidad del resultado final.

Finalmente, el modelo evolutivo también ofrece una mayor flexibilidad en cuanto a los recursos y tiempos necesarios para el desarrollo de software. Al permitir el desarrollo iterativo y modular, los recursos necesarios pueden ser asignados de manera más flexible, lo que se traduce en una mayor eficiencia y eficacia en el proceso de desarrollo [20].

3. Marco Teórico

La automatización de procesos de despliegue es un aspecto crítico en el desarrollo de aplicaciones, especialmente cuando se trata de aplicaciones contenerizadas o proyectos que funcionan en entornos de integración y despliegue continuo. La automatización puede mejorar la velocidad y la fiabilidad del despliegue, y reducir el tiempo y los costos asociados con el desarrollo y el mantenimiento de aplicaciones. En este capítulo se abordarán diferentes metodologías y tecnologías utilizadas en la automatización de procesos de despliegue, incluyendo los contenedores, plataformas para despliegue de aplicaciones Web, infraestructura como código y la integración y despliegue continuo. Se presentarán también trabajos relacionados con el tema. Además, se enseña un resumen de los conocimientos disciplinarios específicos adquiridos a lo largo de las distintas asignaturas cursadas y aplicados en el desarrollo de este trabajo final de maestría.
Este capítulo busca proporcionar una comprensión más profunda del marco teórico detrás de la automatización de procesos de despliegue y sus aplicaciones en el mundo del desarrollo de software.

3.1 Ámbito de Aplicación

Este estudio se aplica a equipos de desarrollo de software en pequeñas y medianas empresas (PYMEs). En particular, la validación se llevará a cabo en una empresa de desarrollo de software de la ciudad de Corrientes, que cuenta con 80 empleados, más de 10 equipos de desarrollo y un promedio de tres despliegues semanales.

3.2 Conocimiento Disciplinar Específico

A continuación, se presentan los conocimientos específicos adquiridos en cada materia de la Maestría en Tecnologías de la Información para su aplicación en este trabajo.

- Procesos de desarrollo de software: Se incorporaron principios de las metodologías ágiles, promoviendo un entorno multidisciplinario dentro del equipo. Se fomentó que todos los desarrolladores pudieran desplegar el proyecto en Internet de manera sencilla, tanto en entornos de desarrollo y pruebas como en producción.

- Prueba de software: La importancia de escribir un código fuente de calidad y documentar los procesos realizados ha sido una lección clave que se ha aplicado en

este trabajo. Además de la automatización de estos procesos para agilizar el trabajo en general.

- Aplicaciones de Cloud Computing: Los conocimientos adquiridos en esta materia, específicamente en el área de redes y servidores, han sido de gran importancia y han permitido abordar de manera efectiva los aspectos técnicos y tecnológicos de la investigación.

- Gestión del conocimiento: Se buscó Generar un flujo de conocimiento dentro de la organización de explícito a explícito; o sea, compartir y transferir conocimientos de manera efectiva entre los miembros del equipo. Esto de acuerdo con el modelo de gestión de conocimiento denominado SECI [21] (por sus siglas en inglés Socialization, Externalization, Combination, and Internalization).

- Desarrollo avanzado de Aplicaciones: Los conocimientos adquiridos en esta materia fueron fundamentales para abordar la arquitectura y estructuración del proyecto. Se utilizaron los conceptos aprendidos sobre la creación de diagramas de proyectos de software, lo que permitió una planificación más eficiente y una comprensión clara de la estructura del proyecto.

3.3 Metodologías y Tecnologías Estudiadas

3.3.1 GitOps

GitOps es una metodología para gestionar la infraestructura de TI y las aplicaciones utilizando Git como fuente única de verdad. La misma está estrechamente relacionado con la filosofía DevOps [22]. En GitOps, todas las configuraciones de infraestructura y código de aplicación se almacenan en un repositorio de Git. Este repositorio actúa como el único lugar donde se define el estado deseado del sistema.
Según la página oficial de OpenGitOps [16], mantenida por la Cloud Native Computing Foundation (CNCF) [23], los principios y características que esta metodología ofrece incluyen:

- **Declaratividad**: En lugar de describir los pasos necesarios para cambiar el estado del sistema, se describe el estado deseado del sistema en un repositorio Git. Esto

permite una gestión más simplificada y desacoplada de la infraestructura y las aplicaciones.

- **Automatización**: Se utiliza la automatización para garantizar que el estado actual del sistema coincida con el estado definido en el repositorio Git. Se utilizan herramientas de CI/CD (Integración Continua/Despliegue Continuo) para aplicar los cambios automáticamente en el entorno en función de las actualizaciones en el repositorio Git.
- **Control de versiones y trazabilidad**: Al utilizar Git como fuente única de verdad, se obtiene el beneficio inherente del control de versiones y la trazabilidad de los cambios en la infraestructura y las aplicaciones. Esto facilita la colaboración entre equipos, la auditoría y la reversión de cambios en caso de problemas.
- **Seguridad y consistencia**: Al definir la infraestructura y las aplicaciones como código, se pueden aplicar las mismas prácticas de seguridad y control de calidad que se utilizan para el desarrollo de software. Esto ayuda a garantizar la coherencia y la seguridad en todos los entornos.

La Fig. 6 representa los componentes clave de la metodología GitOps.

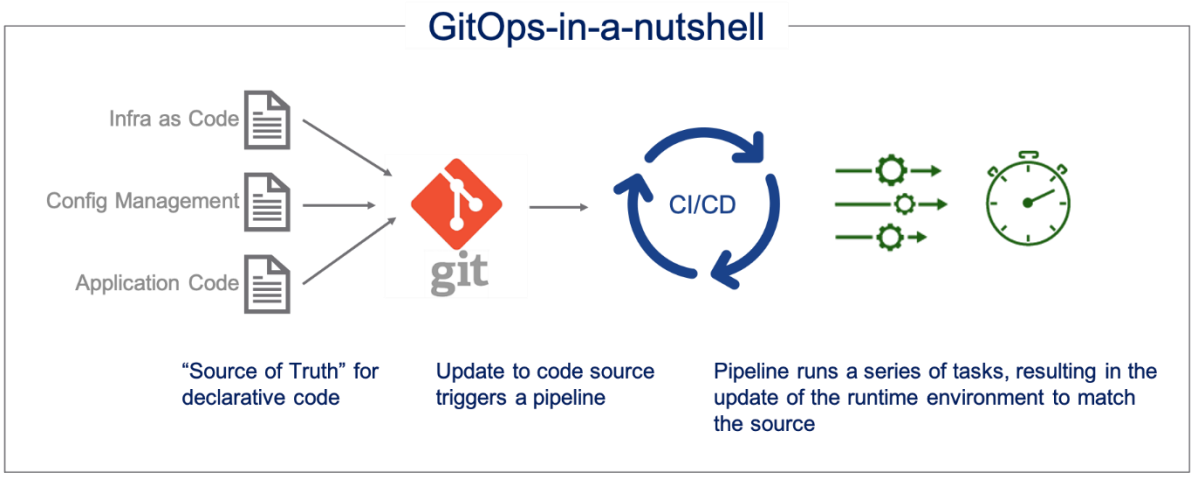

Fig. 6:Componentes GitOps. Fuente: https://blogs.vmware.com/cloud/2021/02/24/gitops-cloud-operating-model/

La adopción de esta metodología puede facilitar la implementación de operaciones de TI en modo *autoservicio*, permitiendo a los usuarios realizar cambios en la infraestructura mediante solicitudes de extracción (PR, por sus siglas en inglés) en repositorios Git, en lugar de depender de procesos manuales y burocráticos. Como se menciona en [24]:

Se puede entender a esta metodología como el resultado de la suma de la Infraestructura como Código y las Pull Requests de Git, o sea: IaC + PR = GitOps.

En resumen, esta metodología puede entenderse como un flujo de trabajo que automatiza la validación y despliegue de cambios en la infraestructura a través de Pull Requests en repositorios Git, optimizando la colaboración entre equipos y asegurando la calidad y seguridad del proceso mediante herramientas de integración y despliegue continuo.
Este enfoque no solo reduce los tiempos de espera para los usuarios, sino que también mejora la transparencia y la seguridad, ya que todos los cambios quedan registrados en el historial del repositorio.

3.3.2 Contenerización

Las tecnologías de contenedores ofrecen una agilidad sin precedentes en el desarrollo y ejecución de aplicaciones. Son especialmente útiles en el desarrollo y despliegue de sistemas, ya que permiten crear entornos de ejecución que incluyen únicamente las dependencias específicas de cada aplicación, como servidores Web, soporte para lenguajes de programación, bases de datos y otros componentes necesarios, sin la necesidad de incluir un sistema operativo completo. Mediante el uso de herramientas de contenedores como ser Docke [10], LXC [25] o Podman [26], podemos empaquetar una aplicación Web y su entorno de ejecución en “imágenes” o paquetes similares a pequeñas máquinas virtuales que puede ser desplegadas rápidamente en cualquier entorno de trabajo [27].
La llegada de los contenedores al mundo del desarrollo transformó la manera en que se despliegan y mantienen las aplicaciones Web modernas. A continuación, se destacan algunas de las implementaciones más importantes en la actualidad.

3.3.2.1 LXC

Pioneros en esta tecnología, los contenedores de Linux (Linux Containers o LXC) permiten empaquetar y aislar aplicaciones junto con todo su entorno de ejecución, es decir, con todos los archivos necesarios para su funcionamiento. A diferencia de otras tecnologías de contenedores, LXC ofrece un enfoque más cercano a la virtualización de sistemas operativos completos, ya que permite ejecutar múltiples contenedores con sus propios entornos completos, utilizando el mismo núcleo del sistema operativo [28].

3.3.2.2 Docker

Docker emplea el concepto de "imágenes" para generar contenedores a partir de un proyecto de software. Se pueden crear múltiples contenedores utilizando una misma imagen. Estas imágenes pueden almacenar e indexar en un repositorio en línea, como DockerHub [29], GitLab Container Registry [30] o incluso en un registro personalizado en un servidor privado. Posteriormente, desde un servidor, es posible descargar estas imágenes desde sus respectivos repositorios y ejecutarlas en dicho servidor. El proceso descrito anteriormente se representa gráficamente en la Fig. 7.

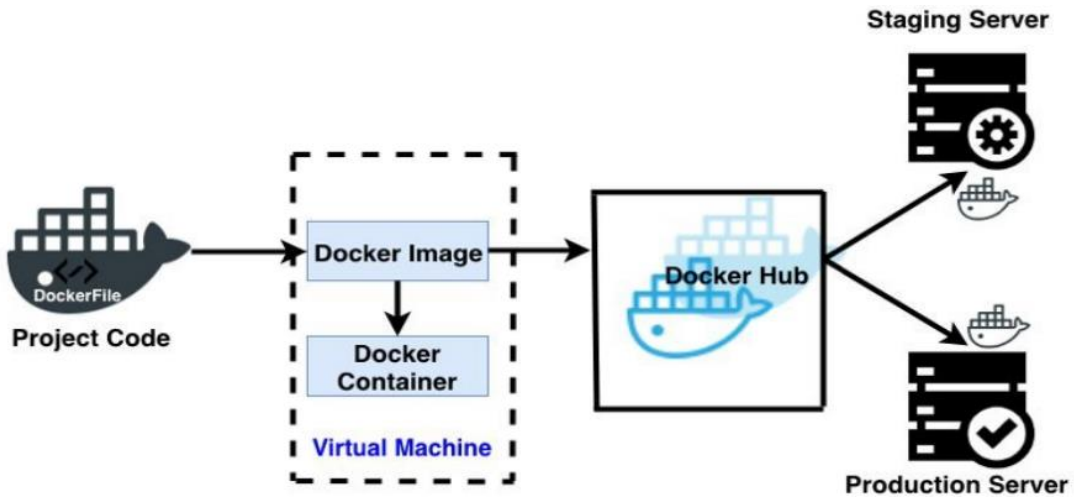

Fig. 7: Ciclo de vida de contenedores en Docker. Fuente [31]

3.3.2.3 Podman

Podman es un motor de contenedores que se ejecuta sin un proceso demonio (es decir, es *daemonless*) para desarrollar, administrar y ejecutar contenedores en sistemas Linux. Es de código abierto y desarrollado por Red Hat [32].

Para la creación de este motor de contenedores, se han descentralizado todos los componentes necesarios para la gestión de los contenedores, individualizándolos en módulos más pequeños que se activan solo cuando es necesario. Esta descentralización ofrece múltiples ventajas, como un servicio más liviano en comparación con otras soluciones de contenedores. Además, al ser desarrollado sin depender de un servicio centralizado, Red Hat ha logrado desvincularlo de la marca Docker (por ejemplo, utilizamos un archivo *Containerfile.yml* en lugar de *Dockerfile.yml*), lo que permite crear contenedores agnósticos [26].

Aunque Podman ofrece ventajas significativas en comparación con Docker, este último sigue siendo el estándar predominante para el manejo de contenedores en la actualidad.

Esto se debe, en gran parte, a que Docker es ampliamente utilizado por la comunidad de desarrolladores y por los principales proveedores de computación en la nube, como Amazon Web Services (AWS) [33], Microsoft Azure (Azure) [34] y Google Cloud Platform (GCP) [35].

3.3.3 Plataformas para el Despliegue de Aplicaciones Web

Al desplegar aplicaciones Web en Internet, existen diversas opciones disponibles. Las más comunes incluyen servidores dedicados, servidores privados virtuales (VPS) y plataformas de computación en la nube (Cloud Computing) que ofrecen distintos servicios y niveles de abstracción.

3.3.3.1 Servidores Dedicados

Un servidor dedicado es una máquina física exclusiva para alojar una aplicación o sitio Web, sin compartir recursos con otros. Ofrece un mayor control y personalización, ya que el propietario tiene acceso completo a los recursos y configuraciones del servidor. Además, proporciona un nivel superior de seguridad y privacidad. Sin embargo, suele ser más costoso debido al mantenimiento de la infraestructura física.

La elección de servidores dedicados para el despliegue de aplicaciones no es común, siendo más frecuente en el caso de aplicaciones críticas que gestionan información sensible o confidencial.

3.3.3.2 Servidores Privados Virtuales (VPS)

Un servidor privado virtual es una partición virtual dentro de un servidor físico, que actúa como si fuera un servidor dedicado con su propio sistema operativo, espacio en disco, memoria y ancho de banda. Cada VPS puede ser configurado y personalizado de manera individual, lo que permite a los usuarios tener más control sobre su alojamiento Web.

Ofrecen un equilibrio entre la funcionalidad de un servidor dedicado y el costo de un alojamiento compartido, ya que, aunque los recursos físicos del servidor se comparten, el usuario tiene libertad para instalar su propio software y ajustar la configuración.

Entre los principales usos de los VPS en el mundo del desarrollo Web se incluye tanto desarrollo y pruebas de aplicaciones, como también despliegue de producción.

3.3.3.3 Computación en la Nube

La computación en la nube (conocida como "Cloud Computing" en inglés) es un modelo de computación que proporciona servicios de tecnología de la información a través de Internet. Los servicios de computación en la nube se ofrecen a través de servidores remotos gestionados por proveedores de servicios en la nube, como AWS, Azure, GCP, entre otros. Estos proveedores ofrecen una variedad de servicios, como almacenamiento de datos, procesamiento de datos, alojamiento de sitios Web y aplicaciones, análisis de datos, inteligencia artificial, aprendizaje automático y mucho más.
Los principales modelos de computación en la nube se ven representados en la Tabla 2.

Tabla 2: Modelos de computación en la nube.

Modelo de Computación en la Nube	Descripción	Ejemplo
IaaS (Infraestructura como servicio)	Este modelo proporciona acceso a recursos informáticos, como servidores, almacenamiento y redes, a través de Internet. El usuario es responsable de la configuración y administración de su infraestructura en la nube.	AWS EC2, DigitalOcean Droplets, Google Cloud Compute Engine, IBM Cloud Infrastructure
PaaS (Plataforma como servicio)	Este modelo proporciona una plataforma de desarrollo en la nube que permite a los desarrolladores crear y ejecutar aplicaciones sin tener que preocuparse por la infraestructura subyacente. El usuario es responsable de la gestión de la aplicación.	Heroku, Google App Engine, Microsoft Azure App Service, Red Hat OpenShift
SaaS (Software como servicio)	Este modelo proporciona aplicaciones de software completas que se ejecutan en la nube y están disponibles para su uso a través de Internet. El usuario no tiene que preocuparse por la gestión de la infraestructura o la aplicación.	Google Workspace (anteriormente G Suite), Microsoft 365, Salesforce

Entre las principales ventajas de la computación en la nube se incluyen la escalabilidad, la flexibilidad y la eficiencia. Los usuarios pueden escalar sus recursos (tanto horizontalmente

como verticalmente) de acuerdo a sus necesidades, pagar solo por los recursos que utilizan y acceder a servicios de alta calidad, como bien se explica en [36]:

> *La computación en nube ofrece nuevas oportunidades para desplegar aplicaciones escalables de forma eficiente, permitiendo que las aplicaciones empresariales puedan ajustar dinámicamente sus recursos de cómputo bajo demanda.*

A pesar de las ventajas que ofrece la computación en la nube, es importante considerar algunas de sus desventajas en comparación con servicios más tradicionales, como los VPS:

- A menudo, es difícil estimar el costo final de los servicios utilizados, y dependiendo del servicio elegido, pueden surgir costos adicionales imprevistos.
- La dependencia de un proveedor de servicios en la nube puede complicar la transición a otro proveedor si es necesario. Además, algunos proveedores ofrecen servicios propietarios que no son interoperables con otras plataformas, lo que limita el uso de herramientas y aplicaciones de terceros.
- La curva de aprendizaje asociada con la configuración y administración de la infraestructura en la nube requiere habilidades técnicas específicas, lo que puede implicar una inversión significativa de tiempo y recursos.

3.3.4 Infraestructura como Código (IaC)

La infraestructura como código es una práctica que permite gestionar y aprovisionar automáticamente la infraestructura de TI mediante código y herramientas de desarrollo de software, como explica Red Hat en [37]:

> *La infraestructura como código permite a los equipos de TI gestionar la infraestructura de manera más eficiente y escalable, ya que el código se puede versionar, probar y desplegar automáticamente. Además, esta práctica también ayuda a reducir errores y a aumentar la velocidad y la agilidad en la entrega de aplicaciones y servicios de TI.*

Las herramientas de infraestructura como código desempeñan un papel crucial en el desarrollo y la orquestación de software nativo y a escala. Permiten automatizar la creación y gestión de la infraestructura mediante código, en lugar de configurar manualmente cada

recurso, como servidores, redes y almacenamiento [38]. Esto permite que la infraestructura se trate como cualquier otro componente de software, lo que significa que puede ser probada, versionada, dividida en diferentes entornos y actualizada de manera más fácil y rápida.

3.3.5 Integración y Despliegue Continuo (CI/CD)

Como parte de la transformación *Agile* en los últimos años, hemos visto a las organizaciones de TI adoptar principios de integración continua en su ciclo de vida de entrega de software, lo que ha mejorado la eficiencia de los equipos de desarrollo [39].
Las prácticas continuas, es decir, la integración, entrega e implementación continuas, son las prácticas de la industria de desarrollo de software que permiten a las organizaciones lanzar nuevas funciones y productos de manera frecuente y confiable [40].

Actualmente, existen diversas herramientas para la implementación de CI/CD, tales como Jenkins, GitHub Actions y GitLab CI/CD [41]. No obstante, es factible además llevar a cabo una implementación de despliegue continuo a través de scripts programados manualmente. Este enfoque permite automatizar tareas como detener contenedores o procesos, descargar la versión más reciente del código o las imágenes y reiniciar los servicios, facilitando un despliegue eficiente de aplicaciones Web en servidores VPS.

3.4 Trabajos Académicos Relacionados

3.4.1 Approaches for Documentation in Continuous Software Development

El artículo "Approaches for Documentation in Continuous Software Development" de Theo Theunissen, Stijn Hoppenbrouwers y Sietse Overbeek (2022) [42], aborda el desafío de la documentación en el desarrollo continuo de software (CSD, por sus siglas en inglés), donde predominan enfoques ágiles, Lean y DevOps. Los autores resaltan que, aunque la documentación es esencial para preservar el conocimiento sobre decisiones de diseño, construcción y mantenimiento, los desarrolladores suelen minimizar su creación y consulta, lo que puede llevar a la vaporización del conocimiento, es decir, la pérdida de información clave a lo largo del ciclo de vida del software.
Para mitigar este problema, proponen tres enfoques principales:

- **Documentación inicial mínima suficiente (Just Enough Upfront Documentation)**: Sugiere generar una documentación inicial mínima pero suficiente para arrancar un proyecto o iteración, enfocándose en la comunicación de ideas, interfaces entre subsistemas y un plan general. La documentación más detallada se desarrolla al final, cuando las decisiones de diseño están más definidas.
- **Documentación ejecutable (Executable Documentation)**: Propone que la documentación sea validable y ejecutable, como especificaciones automatizadas, pruebas (TDD, BDD, ATDD) y scripts de infraestructura como código. Este enfoque facilita su actualización y reduce la fricción con los desarrolladores.
- **Análisis automatizado de texto (Automated Text Analytics)**: Utiliza técnicas de procesamiento de lenguaje natural (NLP, por sus siglas en inglés) y aprendizaje profundo para extraer decisiones de diseño desde comentarios en Git y otras fuentes de texto no estructurado, reduciendo la dependencia de la documentación manual.

Los autores también analizan los artefactos documentales asociados a cada enfoque, como diagramas, descripciones de interfaces, pruebas ejecutables y análisis automatizados de texto; y destacan que estos enfoques pueden aplicarse en distintas etapas del ciclo de vida del software y en diversos niveles de madurez tecnológica (TRL, por sus siglas en inglés).

3.4.2 Simplifying the DevOps Adoption Process

El artículo *"Simplifying the DevOps Adoption Process"* de Ineta Bucena y Marite Kirikova [43], publicado en el contexto de la Universidad Técnica de Riga, analiza la adopción de DevOps en pequeñas empresas y propone un método estructurado para facilitar este proceso. Los autores destacan que, si bien DevOps busca cerrar la brecha entre desarrollo y operaciones, su implementación no es trivial y requiere cambios significativos en la organización y los flujos de trabajo.
El estudio identifica cuatro categorías principales de desafíos en la adopción de DevOps:

- **Falta de conciencia**: Incluye la ausencia de una definición clara de DevOps, la percepción de que es una moda pasajera y la falta de comunicación y capacitación.
- **Falta de apoyo**: Se refiere a la resistencia al cambio tanto a nivel gerencial como de los equipos, así como la falta de confianza en el proceso.
- **Problemas tecnológicos**: Abarca desafíos relacionados con entornos heterogéneos, restricciones de la industria y la integración de herramientas.

- **Adaptación de procesos organizacionales**: Considera la complejidad estructural, la distribución geográfica de los equipos y la integración con procesos existentes.

Para abordar estos desafíos, los autores proponen un método de adopción de DevOps basado en los siguientes pasos:

1. **Identificación de impedimentos**: Se analizan obstáculos utilizando un marco de "categorías de impedimentos" para priorizar áreas de mejora.
2. **Evaluación del nivel de madurez**: Se establece el nivel actual de DevOps en la organización y se define el nivel objetivo.
3. **Priorización de prácticas de DevOps**: Se seleccionan prácticas clave en función de los impedimentos detectados y los objetivos de madurez.
4. **Selección de herramientas**: Se eligen herramientas que respalden las prácticas identificadas.
5. **Elección de un objeto de adopción**: Se inicia el proceso con un proyecto o sistema pequeño.
6. **Definición de métricas**: Se establecen indicadores para medir el éxito de la adopción.
7. **Implementación y evaluación**: Se ejecuta la estrategia, se analizan los resultados y se ajusta el enfoque en fases posteriores.

El método fue validado en una empresa de tamaño mediano, donde demostró su utilidad para priorizar prácticas y herramientas, así como para mitigar algunos de los desafíos clave en la adopción de DevOps. Los autores concluyen que esta metodología puede simplificar la adopción de DevOps en pequeñas empresas, aunque su aplicabilidad en grandes organizaciones aún no ha sido evaluada.

3.4.3 Exploring GitOps for Smaller-Scale Applications

En la tesis "Exploring GitOps for Smaller-Scale Applications" de André Nordström (2024) [44], se investiga la aplicación de los principios de GitOps en entornos que no dependen de Kubernetes [47], enfocándose en aplicaciones de menor escala. El estudio busca demostrar que esta metodología puede implementarse sin necesidad de Kubernetes, lo que resulta especialmente relevante para equipos pequeños o proyectos que no requieren la complejidad ni los recursos asociados a arquitecturas basadas en microservicios.

Como prueba de concepto, Nordström desarrolla un operador de GitOps basado en Docker, demostrando que es posible aplicar los principios GitOps (declaratividad, versionado, automatización y reconciliación continua) sin depender de Kubernetes. Para evaluar su desempeño, lo compara con una solución basada en Argo CD [48].

El operador basado en Docker es más liviano y consume menos recursos que Argo CD, lo que lo hace adecuado para aplicaciones de menor escala. Sin embargo, Argo CD ofrece características avanzadas, como la gestión de múltiples clústeres y *rollbacks* automáticos, lo que lo hace más apropiado para entornos complejos.

La implementación del operador en Docker se lleva a cabo con Docker Compose y se automatiza mediante pipelines de CI/CD con GitHub Actions. La infraestructura se gestiona con Terraform, permitiendo una configuración declarativa y reproducible. La evaluación muestra que ambos operadores cumplen con los principios de GitOps, aunque la solución basada en Docker es más adecuada para aplicaciones simples y de bajo tráfico.

Si bien el operador de Docker es una alternativa viable para proyectos pequeños, presenta limitaciones en escalabilidad y seguridad, principalmente por el acceso directo a la interfaz de programación de aplicaciones (API, por sus siglas en inglés) de Docker. Se recomienda continuar investigando y desarrollando herramientas que mejoren estos aspectos en entornos no basados en Kubernetes.

Esta tesis contribuye a la literatura al demostrar que GitOps no está exclusivamente ligado a Kubernetes y puede aplicarse en entornos más simples. Además, sienta las bases para futuras investigaciones sobre su implementación en aplicaciones de menor escala, beneficiando a equipos con recursos limitados o que no requieren la complejidad de Kubernetes.

3.4.4 GitOps: The Evolution of DevOps

Beetz y Harrer (2022) presentan en su artículo "GitOps: The Evolution of DevOps" [22], publicado en IEEE Software, una exposición detallada acerca de la evolución de las prácticas DevOps hacia el paradigma GitOps. Los autores proponen que GitOps surge como una respuesta natural a las limitaciones encontradas en los métodos tradicionales de gestión de infraestructuras, enfatizando la necesidad de una mayor trazabilidad, control y consistencia en los despliegues.

En el trabajo se establece que GitOps se basa en la utilización de repositorios Git como única fuente de la verdad, lo que permite definir el estado deseado de sistemas y aplicaciones de forma declarativa mediante archivos de configuración (por ejemplo, en formatos YAML o JSON). Este enfoque posibilita la automatización de la integración y

despliegue continuo (CI/CD), garantizando que cualquier modificación en el repositorio se traduzca en cambios automáticos y verificables en el entorno de producción.
Entre las principales ventajas destacadas se encuentran:

- **Trazabilidad y auditoría:** La centralización de los cambios en Git proporciona un registro histórico detallado, facilitando la identificación y corrección de errores.
- **Seguridad y confiabilidad:** La inmutabilidad de los commits y la capacidad de revertir cambios refuerzan la integridad operativa y la resiliencia del sistema.
- **Consistencia en entornos distribuidos:** GitOps permite replicar configuraciones de manera uniforme en entornos multicloud o distribuidos, mejorando la gestión de infraestructuras heterogéneas.
- **Feedback continuo:** La integración con sistemas de monitoreo asegura que el estado actual del sistema se alinee con el estado deseado, habilitando una respuesta automática ante desviaciones.

El artículo también aborda desafíos relevantes, como la complejidad inherente a la migración de infraestructuras tradicionales a un modelo GitOps, así como la necesidad de un cambio cultural que fomente la colaboración y la disciplina en el uso de Git. Asimismo, se discuten aspectos críticos relacionados con la gestión de emergencias, en las que la automatización debe complementarse con procedimientos adecuados para mitigar incidentes.
Finalmente, Beetz y Harrer ilustran la aplicabilidad de GitOps mediante estudios de caso y ejemplos prácticos que demuestran mejoras significativas en eficiencia operativa, velocidad de despliegue y resiliencia del sistema. Concluyen que GitOps representa una evolución natural y prometedora dentro de las prácticas DevOps, anticipando que su consolidación en la industria continuará impulsando avances en la gestión automatizada y segura de infraestructuras.

3.4.5 GitOps: A Path to More Self-Service IT

Limoncelli (2018) explora en su articulo "GitOps: A Path to More Self-Service IT" [24], cómo el enfoque GitOps transforma la operación de sistemas IT al facilitar procesos de autoservicio, integrando la gestión de infraestructuras con herramientas de control de versiones y automatización. El autor propone que, al utilizar Git como la fuente única de la verdad para almacenar configuraciones declarativas, se puede implementar un flujo de

trabajo basado en Pull Requests (PR) que permite a los usuarios solicitar cambios de manera directa, los cuales son evaluados, probados y desplegados de forma automática.
Entre los principales aportes y beneficios destacados se encuentran:

- **Automatización y Validación Continua:** La integración de sistemas de Continuous Integration (CI) y Continuous Deployment (CD) permite que los cambios propuestos sean sometidos a pruebas automatizadas antes de su despliegue, garantizando la consistencia y reduciendo errores.
- **Empoderamiento del Usuario y Reducción de Tiempos de Respuesta:** GitOps posibilita que los usuarios realicen solicitudes de modificación a través de PR, lo que acelera la ejecución de cambios y mejora la transparencia en el proceso, reduciendo la dependencia de la intervención directa del equipo de IT.
- **Beneficios Operacionales para IT:** La adopción de GitOps permite que las operaciones de IT sean más escalables y seguras, al mismo tiempo que se disminuye el trabajo manual (toil) y se mejora la trazabilidad de los cambios. Esto contribuye a un mejor retorno de la inversión (ROI) en automatización, ya que se aprovechan herramientas existentes de gestión de código fuente.
- **Evolución Progresiva:** El artículo describe una trayectoria evolutiva en la implementación de GitOps, partiendo de un uso básico del repositorio Git como respaldo, avanzando hacia la integración de procesos automatizados de pruebas y, finalmente, alcanzando un nivel en el que se pueden eliminar progresivamente las revisiones manuales sin comprometer la seguridad ni la calidad de los despliegues.

El trabajo de Limoncelli se posiciona como un referente en la adopción de GitOps, resaltando sus ventajas tanto para los usuarios como para los equipos de IT, y abogando por una transformación cultural en la forma de gestionar infraestructuras. Esta propuesta no solo optimiza la eficiencia operativa, sino que también promueve un entorno de colaboración y autoservicio que resulta cada vez más relevante en el contexto de la administración de sistemas modernos.

3.4.6 Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery

Mvirmani et al. (2015) abordan en “Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery” [45], la evolución y expansión de las prácticas de DevOps, enfatizando la necesidad de trascender la integración continua para

alcanzar una entrega continua eficaz. El artículo se estructura en torno a la identificación de las limitaciones que enfrentan los equipos de operaciones al absorber el ritmo acelerado de los equipos de desarrollo y propone una transformación integral del ciclo de vida del desarrollo de software (SDLC).
Entre los principales aportes destacan:

- **Integración de fases en el SDLC:** Se analizan las distintas etapas (planificación continua, integración continua, despliegue continuo, pruebas continuas y monitoreo continuo) destacando cómo cada una contribuye a crear un flujo de trabajo automatizado y eficiente.
- **Transición de CI a CD:** El estudio enfatiza que, aunque la integración continua ha mejorado significativamente la eficiencia en el desarrollo, la verdadera optimización de la entrega de software se logra mediante la automatización completa del proceso de despliegue. Para ello, se propone la adopción de prácticas de entrega continua respaldadas por herramientas de automatización.
- **Infraestructura como Código (IaC):** Se ilustra, a través de un caso de estudio, cómo mantener la infraestructura en forma de código permite automatizar la provisión de recursos, reducir tiempos de despliegue y facilitar la escalabilidad, contribuyendo a una mayor eficiencia operativa y reducción de costos.
- **Implicaciones organizacionales y culturales:** Los autores señalan que la implementación exitosa de DevOps no solo depende de la adopción tecnológica, sino también de una transformación cultural que fomente la colaboración entre los equipos de desarrollo y operaciones. Esta integración es crucial para responder de manera ágil a las cambiantes necesidades del mercado.

Este trabajo destaca que la adopción de DevOps implica una reestructuración profunda del proceso de entrega de software, orientada a maximizar la eficiencia y calidad mediante la automatización y la integración de todas las fases del SDLC. Los beneficios descritos, tales como la reducción del tiempo de comercialización, el ahorro en costos y la mejora de la calidad, se presentan como elementos clave para que las organizaciones puedan competir en entornos de alta dinámica

3.4.7 Auto-Scaling Web Applications in Hybrid Cloud Based on Docker

Li y Xia (2016) proponen en “Auto-Scaling Web Applications in Hybrid Cloud Based on Docker” [46], una solución innovadora para la gestión dinámica de recursos en aplicaciones

Web mediante la implementación de técnicas de autoescalado en un entorno híbrido de nube, utilizando la tecnología de contenedores Docker. En este trabajo se aborda la problemática derivada de las variaciones en la demanda de las aplicaciones Web, las cuales pueden experimentar picos de acceso repentinos que requieren una rápida ampliación de recursos y, posteriormente, una reducción para evitar el desperdicio cuando la demanda disminuye.

El artículo presenta una arquitectura compuesta por un balanceador de carga, un servidor de gestión y un registro privado de Docker, que en conjunto permiten desplegar aplicaciones Web en contenedores distribuidos tanto en nubes privadas como públicas. La propuesta se fundamenta en un controlador elástico que combina algoritmos predictivos (basados en análisis de series temporales para estimar la demanda futura) con modelos reactivos que monitorean en tiempo real la utilización de recursos. De esta manera, el sistema es capaz de iniciar o detener contenedores en cuestión de segundos, ajustando la capacidad de procesamiento de forma automática para garantizar la calidad del servicio.

La evaluación experimental del prototipo demuestra la efectividad del enfoque, evidenciando que la solución permite responder ágilmente a cambios en la carga de trabajo y optimizar el uso de los recursos en un entorno de nube híbrido.

Este trabajo resulta relevante al ofrecer una alternativa flexible y escalable para la gestión de aplicaciones Web, contribuyendo al estado del arte en técnicas de autoescalado y despliegue en la nube.

3.5 Conclusiones

Este capítulo ha presentado el marco teórico que sustenta la investigación, abordando metodologías y tecnologías clave en la automatización de despliegues, con un enfoque en la adaptación de GitOps para equipos pequeños de desarrollo de software. Se han explorado conceptos fundamentales como la contenerización, las plataformas de despliegue de aplicaciones Web, la infraestructura como código (IaC) y la integración y despliegue continuo (CI/CD), esenciales para comprender el contexto de esta investigación.

GitOps se ha destacado como un enfoque innovador para gestionar infraestructura y aplicaciones de manera declarativa, automatizada y segura, utilizando Git como fuente única de verdad. Su adopción mejora la colaboración, la trazabilidad y la consistencia en los despliegues, aspectos críticos para equipos pequeños que buscan optimizar recursos y aumentar su eficiencia operativa.

También se ha analizado el papel de las tecnologías de contenedores, como Docker, LXC y Podman, en la modernización del desarrollo y despliegue de aplicaciones Web. Estas

herramientas garantizan la portabilidad y consistencia de las aplicaciones en distintos entornos, desde desarrollo hasta producción.

En cuanto a las plataformas de despliegue, se ha examinado la evolución desde servidores dedicados y VPS hacia la computación en la nube, resaltando sus ventajas en escalabilidad, flexibilidad y eficiencia. No obstante, se han identificado desafíos como la estimación de costos y la dependencia de proveedores, factores clave a considerar en su adopción.

La infraestructura como código (IaC) y las prácticas de CI/CD han sido presentadas como pilares fundamentales en la automatización del despliegue. Estas prácticas no solo optimizan la gestión de la infraestructura, sino que también reducen errores y aceleran la entrega de software, lo que resulta crucial para equipos pequeños que buscan mantenerse competitivos en un entorno dinámico.

Además, se han revisado estudios académicos sobre la adopción de DevOps y GitOps, así como la documentación de procesos en entornos de desarrollo continuo. Estos trabajos proporcionan un marco de referencia valioso para comprender los desafíos y oportunidades de implementar estas metodologías en equipos pequeños, aportando ideas relevantes para esta investigación.

En síntesis, este capítulo ha establecido las bases teóricas necesarias para comprender los principios y tecnologías que sustentan este trabajo final de Maestría. La combinación de GitOps, contenerización, IaC y CI/CD ofrece un enfoque eficiente, seguro y escalable para la automatización de despliegues en equipos de todos los tamaños.

Los siguientes capítulos profundizarán en su aplicación práctica, enfocándonos en equipos pequeños, con el objetivo de desarrollar una guía que facilite la adopción de GitOps en estos entornos.

4. Desarrollo y Resultados

En este capítulo se presenta la implementación de la propuesta planteada, incluyendo aspectos técnicos relevantes, iteraciones del proceso de desarrollo, enlace a la guía para la creación de un repositorio de despliegue y ejemplos de utilización. Además, se muestran los resultados obtenidos en la ejecución de la propuesta.

4.1 Propuesta de Despliegue para Equipos Pequeños

El despliegue de aplicaciones Web puede ser un proceso confuso y complejo debido a la diversidad de enfoques disponibles y a la cantidad de tareas adicionales involucradas. Aunque el uso de servicios de terceros, contenedores y entornos de integración y despliegue continuo puede optimizarlo, la mera adopción de estas tecnologías no garantiza una solución clara y sistemática. En muchos casos, el proceso sigue siendo desafiante para algunos miembros del equipo, especialmente cuando la documentación generada durante el desarrollo es insuficiente.

Una forma de abordar estos problemas es aplicar los principios de DevOps, un enfoque que busca mejorar la colaboración entre equipos y optimizar la entrega de software. En el libro *The DevOps Handbook* [49] se describe tres principios fundamentales, conocidos como *Las Tres Vías*, que establecen las bases para una implementación eficaz de DevOps:

- **Primera Vía:** Optimiza el flujo de trabajo desde el desarrollo hasta la entrega de valor al cliente.
- **Segunda Vía:** Destaca la importancia de una retroalimentación rápida entre equipos para prevenir y corregir errores de manera eficiente.
- **Tercera Vía:** Fomenta la experimentación continua, la innovación y el aprendizaje a partir de los fracasos.

Basándose en estos principios, este trabajo final de maestría propone una estrategia para el despliegue de aplicaciones Web en equipos pequeños de desarrollo de software. Para ello, se adopta un enfoque basado en GitOps, una metodología utilizada en entornos de desarrollo a gran escala, donde el despliegue suele involucrar una variedad de recursos como microservicios, múltiples servicios en la nube, herramientas de orquestación de contenedores, pipelines de integración y despliegue continuo, múltiples instancias de la aplicación y bases de datos distribuidas, entre otros [36].

Si bien en equipos pequeños la complejidad del despliegue es menor, sigue representando un desafío y una parte fundamental del ciclo de vida del proyecto. Por ello, se propone una estrategia inspirada en GitOps que permita ejecutar el proceso de despliegue de manera sistemática y eficiente, incluso en entornos de menor escala.

4.1.1 Gestión de Configuraciones para el Despliegue en Equipos de Gran Escala

En proyectos y equipos de gran envergadura, además de contar con repositorios para alojar el código fuente de las distintas partes de la aplicación, como el frontend, los servicios de backend y las APIs, es común disponer de un repositorio específico que almacena todas las configuraciones necesarias para el despliegue. Este enfoque permite gestionar de manera más eficiente los procesos de desarrollo y despliegue, facilitando una evolución del software más ordenada y efectiva.
En general, estas configuraciones y archivos incluyen, entre otros:

- Manifiestos para orquestadores de contenedores, como Kubernetes [47].
- Manifiestos para administradores de paquetes, como Helm [50].
- Configuraciones de Proxies y balanceadores de carga, como AWS ELB [51].
- Configuraciones de malla de servicios (*Service Mesh*), como Linkerd [52] o Istio [53].
- Archivos de infraestructura como código, como CloudFormation [54] o Terraform [55].
- Configuraciones de plataformas de mensajería para microservicios (*message brokers*), como Apache Kafka [56] o RabbitMQ [57].

4.1.1.1 Integración de Documentación e Infraestructura en el Repositorio de Código Fuente

Basándonos en los principios de Infraestructura como Código (IaC) [37] y GitOps [16], la documentación y los archivos relacionados al despliegue deben formar parte del repositorio de código fuente del proyecto. Aunque existen herramientas dedicadas a la documentación, como Confluence de Atlassian [58], a menudo se presentan problemas como la pérdida de enlaces o la falta de adopción por parte de los desarrolladores. Por esta razón, resulta efectivo que la documentación y los archivos de configuración relacionados con el despliegue residan dentro del repositorio o conjunto de repositorios del proyecto. Esta práctica no solo facilita el acceso a la información, sino que también permite la

implementación de futuras automatizaciones basadas en estos archivos y en la documentación almacenada

4.1.2 Aplicación de GitOps en Equipos Pequeños: Diseño de un Repositorio de Despliegue

El enfoque principal propuesto en este trabajo consiste en aplicar la estrategia del "repositorio de despliegue", inspirada en la metodología GitOps, pero adaptada a una escala más pequeña. Esta propuesta, además, incluye la creación de una guía o procedimiento para generar dicho repositorio, que pueda ser ajustado a proyectos específicos con diferentes stacks tecnológicos y arquitecturas.

El repositorio de despliegue, creado a partir de la guía mencionada, actúa como el punto de partida del proceso, ya que incluye un conjunto básico de archivos, directorios, configuraciones y documentación esenciales para el despliegue del proyecto. Dependiendo de las características particulares de cada proyecto, este repositorio podría contener archivos como Dockerfile y docker-compose.yml [11], así como configuraciones para el servidor Web y el Proxy inverso, utilizando Nginx [59] para gestionar la redirección de solicitudes.

Además, el repositorio podría ofrecer documentación sobre la instalación y actualización de la seguridad en la capa de transporte (SSL/HTTPS) mediante Certbot [60], de la autoridad certificadora Let’s Encrypt [61]. También se incluirían ejemplos de scripts Bash [62] que permiten a los desarrolladores automatizar tareas repetitivas, como despliegues, actualizaciones de versiones y backups, en lugar de ejecutar comandos manualmente.

Una ilustración de esta configuración de la aplicación desplegada, que incluye el servidor Web, el Proxy inverso y la gestión de SSL, se muestra en la Fig. 8.

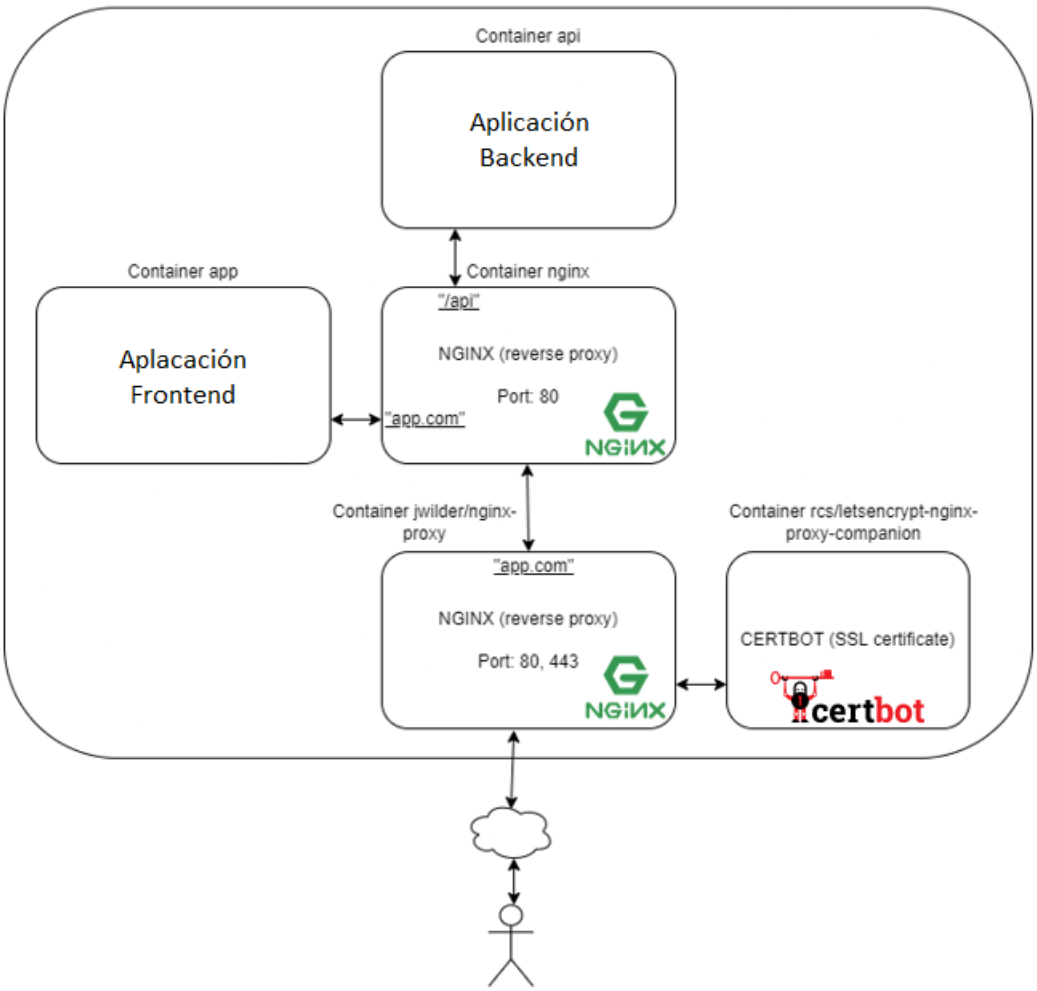

Fig. 8: Estructura de proyectos Web y redireccionamiento.

4.1.2.1 Estructuración de Repositorios y la Incorporación del Repositorio de Despliegue

A la hora de estructurar los repositorios en un equipo de desarrollo, es importante considerar el contexto del proyecto, ya que tanto la composición del equipo como las prácticas adoptadas pueden influir en la organización del código y los archivos de despliegue. En este sentido, existen tres enfoques principales para la gestión de repositorios: Monolito, Monorepo y Multirepo, cada uno con características y aplicaciones específicas, como se detalla a continuación:

- **Monolito:** Un único repositorio para toda la aplicación. Este enfoque es ideal para proyectos más pequeños, pero puede volverse difícil de mantener a medida que el sistema crece y se vuelve más complejo.

- **Monorepo:** Un único repositorio que agrupa múltiples proyectos o módulos relacionados. Facilita la coordinación y el control de versiones, aunque su gestión puede volverse más demandante a medida que el repositorio aumenta en tamaño y complejidad.
- **Multirepo:** Cada componente o módulo del sistema se almacena en repositorios separados. Este enfoque brinda mayor flexibilidad e independencia a los equipos de desarrollo, pero requiere una estrategia de integración bien definida para evitar problemas de compatibilidad y sincronización.

4.1.2.1.1 Enfoque Monolítico

En los proyectos monolíticos, todo el código del sistema, incluyendo tanto la parte del cliente como la del servidor, se almacena dentro de una única carpeta que actúa como el repositorio central de Git, tal como se muestra en la Fig. 9. Para aplicar un enfoque GitOps en este tipo de proyectos, se añade un **directorio deployment** en el directorio raíz, permitiendo centralizar los archivos y configuraciones necesarias para el despliegue.

Fig. 9: Esquema de carpetas en proyectos Monolíticos.

4.1.2.1.2 Enfoque de Monorepo

En los proyectos que adoptan el enfoque Monorepo, un único repositorio contiene múltiples servicios o módulos, como el backend y el frontend. En estos casos, la estructura del repositorio suele organizarse de manera similar a la ilustrada en la Fig. 10. Siguiendo el mismo criterio que en el enfoque Monolito, se incorpora un **directorio deployment** al mismo nivel que las carpetas de los servicios, facilitando la gestión centralizada del despliegue.

Fig. 10: Esquema de carpetas en proyectos Monorepo.

4.1.2.1.2 Enfoque de Multirepo

En los proyectos que adoptan un enfoque Multirepo, cada servicio o componente del sistema se gestiona en un repositorio independiente, lo que permite una separación clara del código y una mayor autonomía en el desarrollo. Como menciona Martin Fowler en [40].

> *Esta división en líneas de trabajo separadas es fundamental para optimizar el flujo de trabajo en equipos de desarrollo de software.*

Dado que los desarrolladores suelen especializarse en áreas específicas del proyecto, como el backend o el frontend, este enfoque permite gestionar el código de forma independiente y aplicar buenas prácticas adaptadas a cada tecnología. Además, contribuye a una organización más estructurada del proyecto.

Siguiendo esta misma lógica, la implementación de GitOps en un entorno Multirepo no implica añadir una carpeta dentro de un repositorio existente, sino la creación de un **repositorio independiente** destinado exclusivamente a los archivos de despliegue. Este repositorio se gestiona como un proyecto separado con su propio control de código fuente, asegurando una administración más estructurada del proceso de despliegue. Un esquema representativo de esta estructura se muestra en la Fig. 11.

Fig. 11: Esquema de carpetas en proyectos Multirepo.

4.1.2.3 Ventajas y Aplicaciones del Repositorio de Despliegue

A lo largo de los distintos enfoques presentados, se ha mostrado cómo incorporar la carpeta o repositorio de **deployment** en cada escenario. Como se mencionó anteriormente, este repositorio tiene un papel fundamental en la documentación y gestión del despliegue de la aplicación, tanto en su implementación inicial como en futuras actualizaciones. Mantener una documentación clara y detallada permite la creación de scripts de automatización, optimizando tareas repetitivas y minimizando errores. Además, facilita la definición de procesos programados, como copias de seguridad y otras tareas críticas para la operación del sistema.

Entre las principales ventajas de esta estrategia se destacan las siguientes:

- **Estandarización del proceso de despliegue**, garantizando coherencia y consistencia en todos los entornos.
- **Reducción de errores y ahorro de tiempo** gracias a la automatización y documentación del proceso.
- **Integración del despliegue como una fase clave del desarrollo**, en línea con la cultura DevOps [39].
- **Uso de Git como herramienta central de documentación**, lo que facilita el acceso y adopción por parte del equipo de desarrollo.
- **Mayor flexibilidad y portabilidad del proyecto**, al documentar y automatizar el proceso de despliegue, lo que simplifica la migración entre servidores o entornos.
- **Gestión y personalización de la configuración para cada entorno**, permitiendo ajustes sin afectar el código fuente principal.

- **Seguimiento claro de cambios relacionados con el despliegue**, asegurando transparencia y trazabilidad en cada actualización.

Es importante destacar que el repositorio de despliegue sigue siendo una herramienta esencial incluso cuando se trabaja con servicios externos, como plataformas en la nube, PaaS o soluciones de CI/CD. Al igual que en entornos de despliegue en servidores privados virtuales (VPS), este repositorio actúa como un almacén centralizado de configuraciones clave para gestionar la infraestructura de despliegue. En estos casos, puede incluir documentación detallada sobre credenciales requeridas, enlaces a recursos relevantes y configuraciones específicas de cada servicio externo utilizado.
De este modo, se garantiza una gestión coherente del despliegue, independientemente de la plataforma empleada, lo que facilita la colaboración del equipo y el mantenimiento del proyecto a lo largo de su ciclo de vida.

4.1.2.3 Seguridad en el Manejo de Claves y Credenciales

Es fundamental garantizar la protección de las claves y credenciales utilizadas para acceder a servidores y otros servicios externos sensibles. Por esta razón, es crucial evitar almacenarlas directamente en el repositorio de despliegue. No obstante, se puede documentar su ubicación en dicho repositorio, asegurando que su acceso y gestión se realicen de manera segura.
Guardar claves y credenciales en repositorios de código representa un riesgo significativo, ya que expone esta información a accesos no autorizados. En su lugar, se recomienda utilizar soluciones de almacenamiento seguro, como plataformas en la nube, por ejemplo, Google Drive [63], o recurrir a gestores de contraseñas especializados como Bitwarden [64].

4.2 Desarrollo en Iteraciones

A continuación, se describe de manera detallada el proceso de desarrollo del proyecto, aplicando un enfoque iterativo basado en el modelo de ciclo de vida evolutivo que se presentó en capítulos anteriores.

4.2.1 Iteración Nro. 1: Encuesta sobre Despliegue de Aplicaciones Web en Diversos Entornos

El primer paso para definir la creación de un repositorio colaborativo con documentación sobre el despliegue de aplicaciones Web de diferentes stacks tecnológicos y en diversos tipos de servidores y servicios consistió en la elaboración de una encuesta utilizando Google Forms. Esta encuesta se distribuyó entre equipos de empresas ubicadas en la región del NEA y desarrolladores independientes, con el objetivo de comprender mejor su entorno de trabajo.

Cabe destacar que la encuesta se realizó en inglés, ya que se contó con la participación de desarrolladores de habla inglesa, quienes colaboran en equipos de desarrollo en los que también participan los desarrolladores de la región del NEA.

Las preguntas y respuestas de la encuesta se pueden consultar en las Figuras Fig. 12, Fig. 13, Fig. 14, Fig. 15 y Fig. 16:

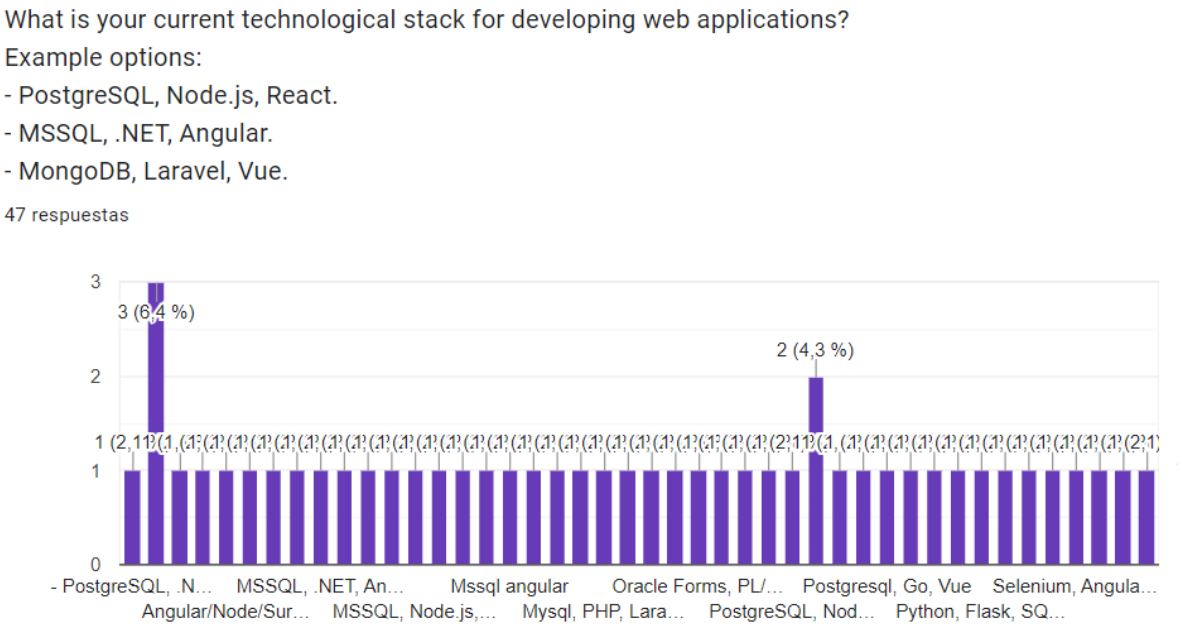

Fig. 12: 1ra pregunta de la encuesta y sus respuestas.

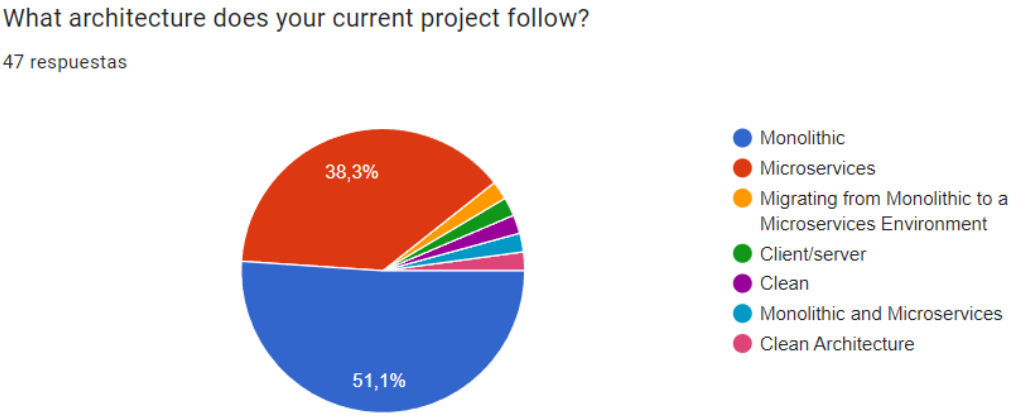

Fig. 13: 2da pregunta de la encuesta y sus respuestas.

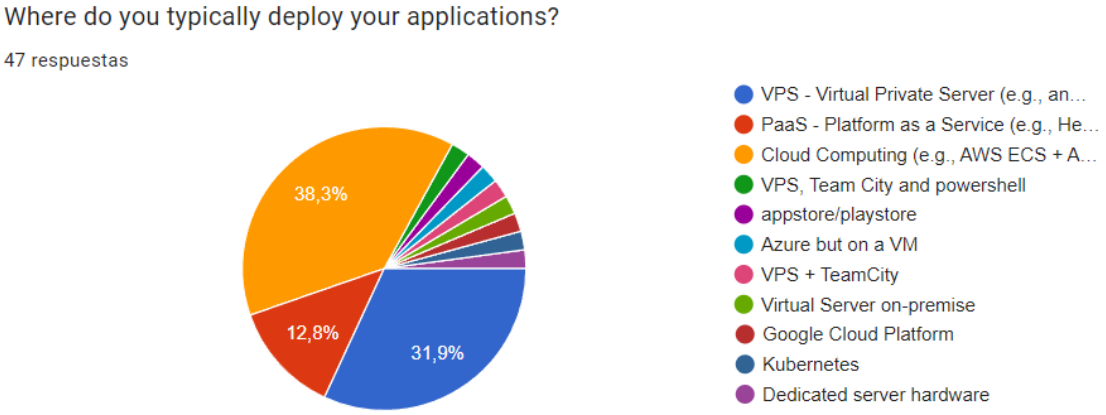

Fig. 14: 3ra pregunta de la encuesta y sus respuestas.

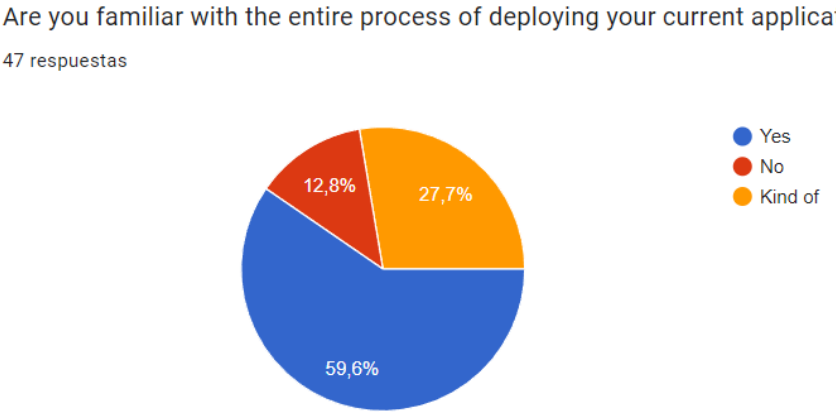

Fig. 15: 4ta pregunta de la encuesta y sus respuestas.

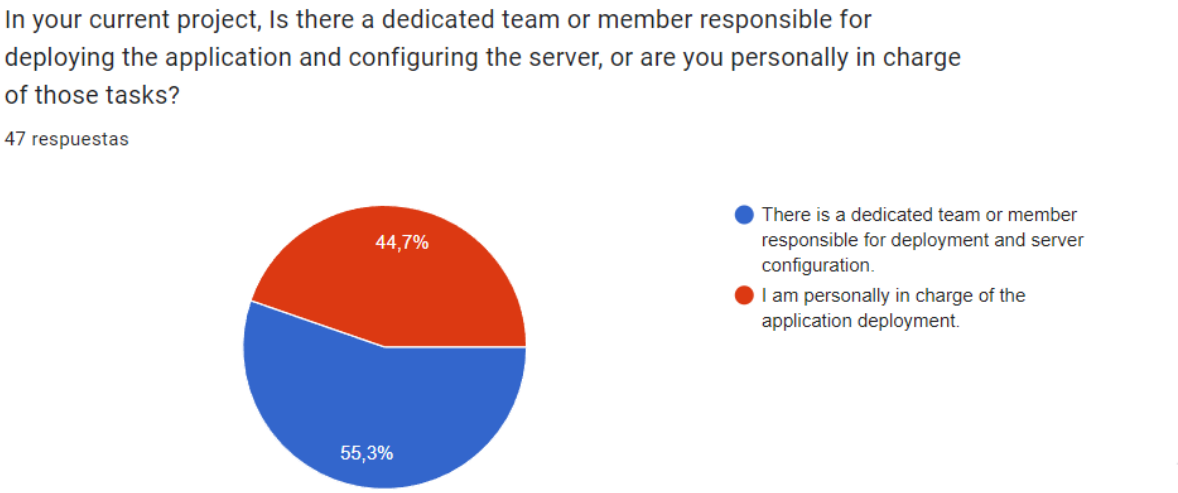

Fig. 16: 5ta pregunta de la encuesta y sus respuestas.

4.2.2 Iteración Nro. 2: Análisis e Interpretación de los Datos

Tras realizar un análisis de los datos recopilados, se han podido identificar las siguientes conclusiones:

4.2.2.1 Análisis de los Resultados de la Encuesta

Tecnologías de Desarrollo Web:

Lenguajes más comunes: Se observa una amplia variedad de lenguajes de programación utilizados, con Node.js, .NET, Python, y Java siendo algunos de los más mencionados.

Frameworks y bibliotecas: React, Angular, y Vue son populares para el desarrollo de interfaces de usuario, mientras que Flask, Django, Spring Boot, y Express.js son comunes en el desarrollo backend.

Bases de datos: PostgreSQL, MongoDB, MySQL, y MSSQL son mencionados con frecuencia como opciones de bases de datos.

Arquitectura de Aplicaciones:

Monolíticas vs. Microservicios: Se observa una mezcla de arquitecturas monolíticas y basadas en microservicios. Algunos están migrando de una arquitectura monolítica a una basada en microservicios.

Arquitecturas "limpias": Algunos mencionan seguir una arquitectura limpia o arquitectura hexagonal en sus proyectos.

Despliegue y Configuración del Servidor:

Despliegue en la Nube vs. Servidores Privados Virtuales (VPS): La mayoría prefiere desplegar en la nube, como AWS, Azure, o Google Cloud Platform, aunque aún hay quienes optan por VPS.
Responsabilidad en el Despliegue: La responsabilidad del despliegue varía, con algunos desarrolladores encargándose personalmente y otros equipos dedicados para esta tarea.

4.2.2.2 Conclusiones

- Se observa una gran diversidad en las tecnologías empleadas para el desarrollo Web.
- Aunque persiste el uso de arquitecturas monolíticas, se nota una creciente tendencia hacia la adopción de microservicios.
- La mayoría de los encuestados prefiere realizar el despliegue en la nube, lo que sugiere una adopción creciente de servicios en la nube para el alojamiento de aplicaciones Web.
- Existen una combinación de responsabilidades en el proceso de despliegue, desde desarrolladores individuales hasta equipos dedicados.

4.2.2.3 Reflexión sobre la Necesidad un Repositorio de Despliegue

El análisis respalda la necesidad de un repositorio centralizado de despliegue y documentación, que simplifique el proceso de implementación de aplicaciones Web en distintos entornos, y se concluye lo siguiente:

Responsabilidad del Despliegue: Si bien algunos desarrolladores asumen la responsabilidad del despliegue, muchos dependen de equipos dedicados. Esto subraya la importancia de un repositorio centralizado, que facilite una colaboración más eficiente y asegure que todos los miembros del equipo estén alineados con los procedimientos de despliegue.

Adopción de la Nube: Si bien la mayoría de los encuestados prefiere desplegar en la nube, esta preferencia puede aumentar la complejidad debido a la variedad de servicios y configuraciones disponibles. Un repositorio de despliegue, en estos casos, podría proporcionar una guía estandarizada para desplegar en entornos de nube, facilitando la migración y el mantenimiento de las aplicaciones.

Transición hacia Microservicios: La tendencia hacia arquitecturas basadas en microservicios plantea desafíos adicionales en términos de despliegue y coordinación entre servicios. Un repositorio de despliegue y documentación podría ayudar a gestionar esta complejidad al proporcionar una vista centralizada de todos los componentes del sistema y sus configuraciones asociadas.

4.2.3 Iteración Nro. 3: Diseño Inicial de la Guía

Para el desarrollo de la guía, se optó por utilizar Docusaurus [65]. Esta elección se fundamenta en su capacidad para generar sitios Web estáticos de manera sencilla y su flexibilidad para organizar y presentar contenido de forma clara y accesible. Además, se seleccionó GitHub [66] como plataforma de alojamiento del código fuente, dado su amplio uso en la comunidad de proyectos de código abierto. Esta decisión garantiza que la guía será accesible públicamente, fomentando la colaboración y la contribución de otros desarrolladores.

La guía se organizará en tres apartados principales, cada uno de los cuales abordará un aspecto esencial del proceso de despliegue mediante la utilización de un repositorio centralizado:

1. **Creación del Repositorio de Despliegue**: Esta sección ofrecerá una guía detallada y paso a paso sobre cómo crear y configurar un repositorio de despliegue, siguiendo las mejores prácticas recomendadas. El objetivo es ayudar a los usuarios a establecer el repositorio de manera eficiente y bien estructurada.
2. **Ejemplos en Distintos Entornos**: En esta sección, se presentarán ejemplos prácticos de archivos de configuración para el despliegue de aplicaciones Web en diferentes entornos, como servidores VPS, plataformas en la nube (AWS, Azure, Google Cloud Platform), servicios externos de CI/CD, así como configuraciones específicas para entornos de desarrollo, pruebas y producción.
3. **Utilidades Adicionales**: Aquí se recopilarán y describirán varias utilidades y documentación complementaria que pueden resultar útiles durante el proceso de despliegue y mantenimiento de aplicaciones web. Esto incluirá configuraciones para servidores VPS, scripts de automatización de copias de seguridad, herramientas de monitoreo y gestión de registros (logs), entre otros.

Cada sección fue diseñada con la flexibilidad necesaria para facilitar su expansión y actualización continua, permitiendo que la comunidad de desarrolladores contribuya con nuevos conocimientos y experiencias.
Además, el diseño inicial de la guía se alinea con el método propuesto en "Simplifying the DevOps Adoption Process" de Bucena y Kirikova [43], ya que al incorporar herramientas como Bash, Docker, Nginx, entre otras, se responde a la necesidad de contar con soluciones accesibles y estandarizadas, lo que ayuda a mitigar los desafíos tecnológicos en la adopción de la metodología DevOps.

4.2.4 Iteración Nro. 4: Creación de los Repositorios Plantilla

Como resultado de las iteraciones previas, se identificó la necesidad de contar con repositorios de despliegue estandarizados que pudieran servir como base para diferentes stacks tecnológicos. Esto permitiría a los equipos trabajar con configuraciones predefinidas y documentadas, lo que agilizaría la creación de repositorios de despliegue para cada proyecto. Además, se busca fomentar la colaboración activa de los desarrolladores, invitándolos a expandir esta base de conocimiento abierto mediante la incorporación de nuevas plantillas y aportes.
En esta iteración, se crearon repositorios plantilla (templates) que abarcan las tecnologías más utilizadas en la empresa Devlights SRL, asegurando su aplicabilidad en proyectos reales. La selección de tecnologías se basó en los datos obtenidos en la iteración Nro. 1 de análisis y en la experiencia práctica dentro de la empresa, priorizando los stacks más frecuentes.
Los repositorios de ejemplo creados para esta primera instancia fueron los siguientes:

- **Stack MERN con pm2:** aplicaciones desarrolladas con MongoDB, Express.js, React y Node.js (MERN) en servidores VPS, utilizando PM2 para gestionar procesos en producción.
- **Stack PERN con Docker en VPS:** aplicaciones desarrolladas con PostgreSQL, Express.js, React y Node.js (PERN) utilizando Docker para la contenerización de los servicios y su ejecución en servidores VPS.
- **Stack PERN con Prisma ORM:** aplicaciones desarrolladas con PostgreSQL, Express.js, React y Node.js (PERN) utilizando Prisma ORM.
- **Spring Boot en Tomcat:** aplicaciones desarrolladas en Spring Boot sobre un servidor Tomcat.

- **React + PocketBase:** aplicaciones desarrolladas en React como frontend y PocketBase como backend y base de datos.

Cada repositorio plantilla incluye archivos de configuración y scripts base que facilitan la implementación y automatización del despliegue. En los casos donde es necesario desplegar múltiples servicios, se han creado archivos Docker Compose, prescindiendo de tecnologías como Kubernetes, más adecuadas para equipos grandes y arquitecturas de microservicios. Esta aproximación sigue la línea del trabajo de Nordström en "Exploring GitOps for Smaller-Scale Applications" [44], donde se demuestra que GitOps puede aplicarse sin depender de orquestadores como Kubernetes; utilizando soluciones más simples como Docker Compose para la gestión de la infraestructura en entornos más pequeños y con recursos limitados.
Los beneficios de esta iniciativa incluyen:

- Fomentar la adopción de GitOps, promoviendo el versionado y la automatización en los procesos de despliegue.
- Facilitar la incorporación de nuevos desarrolladores al equipo, al contar con configuraciones predefinidas y documentadas.
- Reducir la incertidumbre al elegir dónde y cómo desplegar ciertos tipos de aplicaciones, proporcionando scripts y archivos de configuración claros.

4.2.5 Iteración Nro. 5: Incorporación de Estrategias de Documentación

En equipos pequeños, la documentación suele quedar relegada en favor de la velocidad de desarrollo. Para mitigar este problema, se optó por integrar enfoques que equilibren la necesidad de documentar con la eficiencia operativa.
En esta iteración, se decidió fortalecer la guía con estrategias de documentación que faciliten la preservación del conocimiento en el proceso de despliegue de aplicaciones Web. Para ello, se tomaron como referencia las estrategias presentadas en el artículo *"Approaches for Documentation in Continuous Software Development"* [42].
Las estrategias incorporadas en la guía incluyen:

- **Documentación inicial mínima suficiente (*Just Enough Upfront Documentation*):** Se recomienda generar solo la documentación esencial al inicio del despliegue, enfocándose en la estructura del repositorio y el plan general. Los

detalles más específicos se documentarán progresivamente a medida que se tomen decisiones concretas.

- **Documentación ejecutable (*Executable Documentation*):** Se promueve la utilización de archivos y configuraciones versionadas, como README.md, Dockerfiles, YAML y scripts de Bash, de manera que la documentación pueda ser integrada en procesos automatizados y posteriormente utilizada en secuencias de comandos para su ejecución.
- **Análisis automatizado de texto (*Automated Text Analytics*):** Se sugiere explorar herramientas de procesamiento de lenguaje natural (NLP) para extraer información relevante desde commits en Git, código fuente, archivos de infraestructura y otras fuentes, reduciendo la necesidad de documentación manual extensa. Por ejemplo, se sugiere el uso de herramientas de IA como ChatGPT [67].

4.2.6 Iteración Nro. 6: Últimos Ajustes y Mejoras del Proyecto

En esta iteración se llevaron a cabo una serie de actividades destinadas optimizar la guía de creación del repositorio de despliegue, así como su plataforma de presentación. Estas mejoras abarcaron desde la revisión de contenido hasta la optimización de la experiencia del usuario. La Tabla 3 detalla las principales actividades llevadas a cabo en esta fase:

Tabla 3: Actividades de refinamiento y mejoras del proyecto

Actividad	Descripción
Revisión y corrección de contenido	Revisión exhaustiva para corregir errores gramaticales, ortográficos y de formato, asegurando coherencia y precisión en el material.
Incorporación de retroalimentación	Integración de comentarios y sugerencias de los usuarios para mejorar la guía en función de sus necesidades.
Optimización de la experiencia del usuario	Mejora de la navegación y usabilidad de la plataforma para facilitar el acceso a la información y mejorar la experiencia de uso.
Diseño y estilización visual	Aplicación de principios de diseño Web para mejorar la presentación visual de la guía, utilizando esquemas, colores, tipografías y elementos gráficos adecuados.
Pruebas de funcionalidad	Ejecución de pruebas exhaustivas para garantizar el correcto funcionamiento de la plataforma en distintos dispositivos y navegadores, verificando la operatividad de todas sus

	características.
Documentación y mantenimiento	Elaboración de documentación detallada sobre el mantenimiento y actualización de la guía, con instrucciones para agregar nuevo contenido y gestionar solicitudes de extracción (*pull requests*).

4.2.7 Iteración Nro. 7: Documentación y Despliegue de la Guía

En la fase final del proyecto, se realizaron los últimos ajustes previos a su publicación en Internet. Como parte de este proceso, se documentó el repositorio para facilitar la contribución de otros desarrolladores mediante la creación de *Pull Requests* (PR) en GitHub, permitiendo así la actualización colaborativa de la guía.

Para el despliegue, se evaluaron diversas plataformas de alojamiento de archivos estáticos, considerando que la guía consiste principalmente en documentación. Entre las opciones analizadas se encontraban GitHub Pages [68], Netlify [69] y Cloudflare Pages [70]. Finalmente, se optó por Cloudflare Pages debido a su facilidad de uso y su capacidad para replicar el sitio en servidores distribuidos globalmente a través de su Red de Distribución de Contenido (CDN, por sus siglas en inglés).

Además, se configuró una integración automatizada que despliega la guía cada vez que se fusiona una rama de característica (feature branch) con la rama principal (master) del proyecto. La Fig. 17 presenta una representación visual de esta configuración:

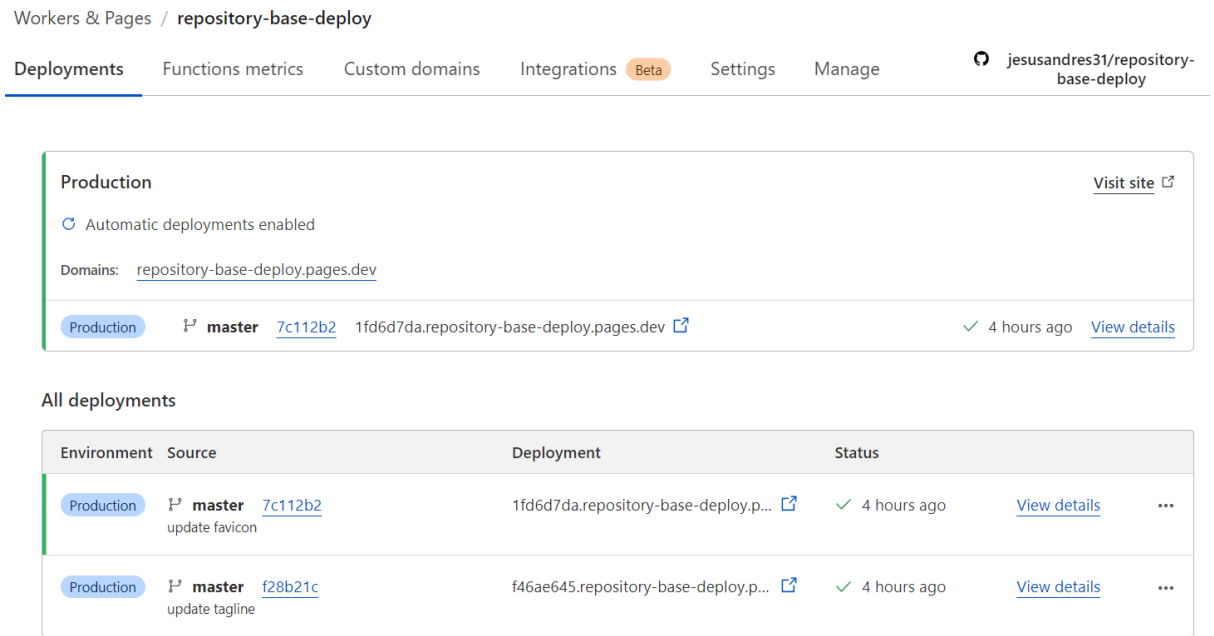

Fig. 17: Configuración del proyecto en Cloudflare Pages

4.2.7.1 Enlaces del Proyecto

La guía está disponible en línea y puede consultarse en el siguiente enlace:

https://gitopsguide.pages.dev

En la Fig. 18 se presentan capturas de la página principal de la guía desplegada, junto con sus secciones principales.

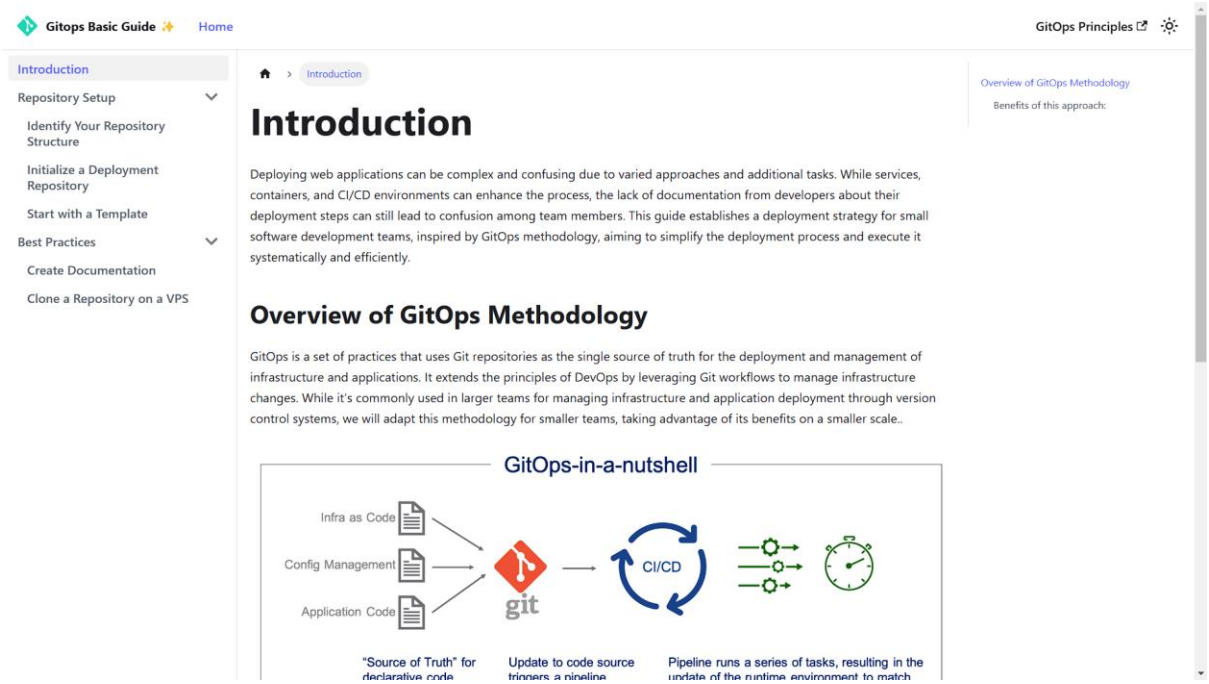

Fig. 18: Guía desplegada.

Además, el código fuente del proyecto se encuentra alojado en un repositorio público, accesible a través del siguiente enlace:

https://github.com/jesusandres31/gitops-guide

Este repositorio fue creado con el objetivo de brindar acceso a los detalles técnicos del proyecto y permitir que cualquier persona interesada pueda explorarlo. Asimismo, se fomenta la colaboración y el intercambio de conocimientos, invitando a los desarrolladores a contribuir con mejoras o sugerencias mediante *Pull Requests* en GitHub

En la Fig. 19 se muestra una imagen del repositorio junto con los pasos para colaborar en el proyecto.

README

Deployment Repository Guide 📦 🛠️ ⚙️

This project is a step-by-step guide to creating a Deployment Repository for Web App projects.

Local Development

1. Clone the repository:

```
git clone https://github.com/jesusandres31/repository-base-deploy.git
```

2. Instgall dependencies:

```
cd repository-base-deploy
npm install
```

3. Start the project:

```
npm start
```

Contributing

1. Create a new feature branch:

```
git checkout -b feature/my-feature
```

2. Make your changes and commit them:

```
git commit -am "Add new feature"
```

3. Push your branch to the repository:

```
git push origin feature/my-feature
```

Fig. 19: Repositorio de la guía.

4.3 Utilización y Validación de la Solución en un Entorno Real

En esta sección se presenta una demostración práctica del uso de la guía para la creación de un repositorio de despliegue, siguiendo la metodología GitOps y adaptándola a las necesidades de proyectos y equipos pequeños. La validación se llevará a cabo en un entorno real dentro de la empresa de software Devlights SRL.

Para esta prueba de concepto, se seleccionaron tres proyectos con diferentes stacks tecnológicos. Aunque cada uno presenta particularidades en su implementación, el objetivo principal es demostrar la aplicabilidad del enfoque GitOps en proyectos de menor escala. La

metodología propuesta es lo suficientemente flexible como para adaptarse a distintos stacks con las modificaciones necesarias en cada caso.
La Tabla 4 presenta una descripción de los proyectos utilizados en la validación.

Tabla 4: Proyectos seleccionados para la implementación del repositorio de despliegue.

Nombre del Proyecto	**Descripción**	**Tecnologías Utilizadas**	**Infraestructura y Ambiente de Despliegue**
Liber App	Aplicación para la gestión de un estudio contable, que permite administrar clientes, contribuyentes, pagos de impuestos y la ingesta de datos desde archivos Excel a una base de datos SQL. También gestiona otros aspectos contables.	**Frontend:** React.js **Backend:** Node.js y Express **Base de datos:** PostgreSQL.	- VPS con Linux Ubuntu y Node.js instalado para la construcción del proyecto. - Nginx se utiliza para servir el frontend y como Proxy inverso para las solicitudes al backend. - PostgreSQL como base de datos. - Rclone para la gestión de copias de seguridad.
CtesF5 Management	Aplicación para la gestión de alquileres de canchas de fútbol 5 en distintas localidades de Corrientes. También administra la venta de bebidas.	**Frontend:** React.js **Backend y Base de datos:** Pocketbase (Headless CMS)	- VPS con Linux Ubuntu y Node.js instalado para la construcción del proyecto. - Nginx se utiliza para servir el frontend y como Proxy inverso para las solicitudes al backend. - PocketBase como base de datos. - Rclone para la gestión de copias de seguridad.

Ddash - Docker Web Dashboard	Aplicación web interna de Devlights SRL para gestionar los contenedores desplegados en VPS o máquinas virtuales utilizadas en el entorno de desarrollo de diversas aplicaciones Web.	**Frontend:** React.js **Backend:** Golang **Base de datos:** SQLite.	- VPS con Linux Ubuntu y Docker. - Cada contenedor incluye las dependencias de cada servicio, con instancias separadas para el frontend, backend y el Proxy/servidor Web.

Es importante destacar que el proceso de despliegue de cada uno de estos proyectos se llevó a cabo tanto con, como sin la metodología propuesta, con el objetivo de evaluar y comparar su desempeño posteriormente.

4.3.1 Procedimiento para Generar un Repositorio de Despliegue Utilizando la Guía

La guía desarrollada, proporciona un flujo estructurado de configuración y toma de decisiones para establecer un repositorio de despliegue acorde a las necesidades del proyecto. La Fig. 20 muestra la sección *Configuración de Repositorio* de esta, la cual sirvió como referencia para agregar el repositorio o directorio de despliegue en los proyectos evaluados.

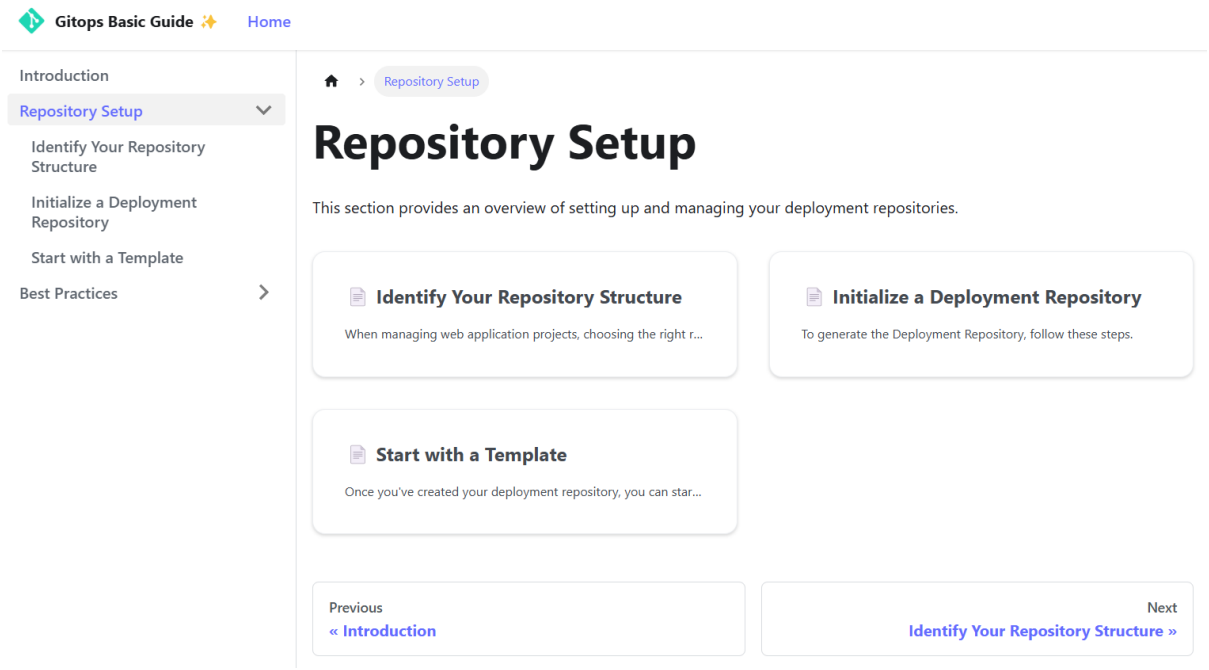

Fig. 20: Sección 'Configuración de Repositorio' de la guía

El proceso seguido consistió en los siguientes pasos:

1. **Definir la estructura del repositorio**: Se accedió a la sección *Identify Your Repository Structure*, donde se describen los distintos enfoques para organizar repositorios: monolítico, monorepo y multirepo. Según la estructura del proyecto, la guía ofrece recomendaciones sobre cómo organizar el código y gestionar la infraestructura de despliegue.
2. **Inicializar el repositorio de despliegue**: En la sección *Initialize a Deployment Repository*, se siguieron los pasos indicados para crear un repositorio de despliegue. La guía proporciona instrucciones detalladas para configurar un nuevo repositorio dentro de un monorepo existente o establecer un repositorio independiente en un entorno multirepo.
3. **Utilizar una plantilla de configuración**: Finalmente, la sección *Start with a Template* permitió incorporar configuraciones predefinidas adaptadas a diferentes escenarios de despliegue. Gracias a esta referencia, los desarrolladores pudieron seleccionar la plantilla más adecuada para su stack tecnológico y entorno de ejecución.

4.3.2 Aplicación del Repositorio de Despliegue en Proyectos Reales

Como se mencionó previamente, se seleccionaron tres proyectos reales para validar la aplicación de la guía en la creación de un repositorio de despliegue. Para cada uno de estos proyectos, se utilizó la guía como referencia para estructurar el repositorio, incorporando los archivos de configuración y los scripts necesarios para automatizar el proceso de despliegue. A continuación, se presentan los resultados obtenidos en cada caso:

4.3.2.1 Liber App

Este proyecto emplea una estructura de multirepositorio, en la que cada componente tiene su propio control de versiones independiente. Inicialmente, se crearon dos repositorios principales:

- Repositorio del frontend: Contenía todos los archivos relevantes para la aplicación cliente.
- Repositorio del backend: Almacenaba los archivos del servidor y la base de datos del proyecto.

Para optimizar la gestión del despliegue, la documentación, la automatización de procesos como la implementación de nuevas versiones y la realización de copias de seguridad automáticas de la base de datos, se incorporó un tercer repositorio dedicado a esas tareas. Este repositorio se describe y se muestra en la Fig. 21.

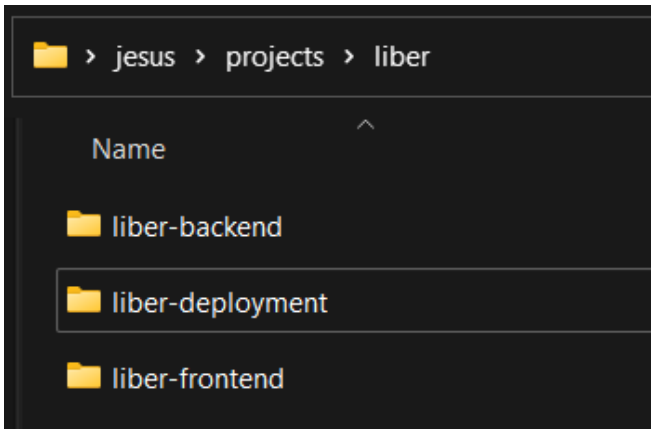

Fig. 21: Imagen de ejemplo de la solución 1.

Como plataforma de alojamiento de repositorios, se utilizó Bitbucket, como se muestra en la Fig. 22.

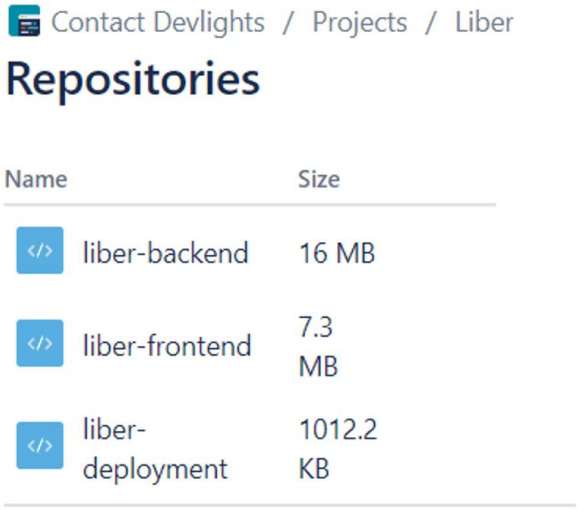

Fig. 22: Imagen de ejemplo de la solución 2.

Como punto de partida, se empleó una de las plantillas disponibles en la guía, la cual proporcionaba archivos genéricos adecuados para este tipo de proyectos. Posteriormente, dichos archivos fueron personalizados y ajustados a las configuraciones específicas del proyecto. El repositorio de despliegue quedó estructurado en los siguientes directorios y archivos:

- **/cronjob**: Contiene archivos para la configuración de tareas programadas en el scheduler del sistema operativo.
    - **dump-and-save-db.sh**: Script para realizar copias de seguridad de la base de datos.
- **/envs**: Archivos de configuración de entornos (desarrollo, prueba, producción, etc.).
    - **/back**: Configuración del entorno del backend.
        - **/private**: Archivos privados correspondientes a servicios utilizados por la API.
        - **.env**: Archivo de variables de entorno del backend.
    - **/front**: Configuración del entorno del frontend.
        - **.env**: Archivo de variables de entorno del frontend.
- **/nginx**: Archivos de configuración del servidor Web y Proxy inverso.
    - **app.liberarg.com**: Archivo de configuración de Nginx.
- **/scripts**: Carpeta que almacena los scripts utilizados en el despliegue y tareas relacionadas.
    - **deploy.sh**: Script para la instalación inicial y la implementación de nuevas versiones de la aplicación.
- **README.md**: Documentación con la información necesaria para desplegar la aplicación y configurarla en distintos entornos.

La Fig. 23 muestra la estructura en detalle.

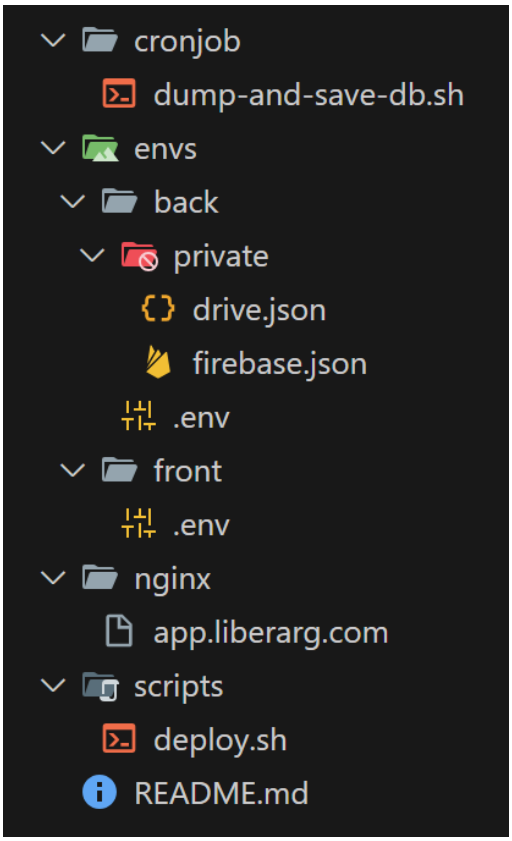

Fig. 23: Imagen de ejemplo de la solución 3.

Para la primera iteración del despliegue, se utilizó una VPS de desarrollo, lo que permitió que el cliente pudiera probar la aplicación mientras se desarrollaban nuevas funcionalidades. Este despliegue en un entorno de prueba y desarrollo facilitó el refinamiento de los archivos de configuración y los scripts de automatización específicos para el proyecto. La Fig. 24 presenta un diagrama de la arquitectura desplegada.

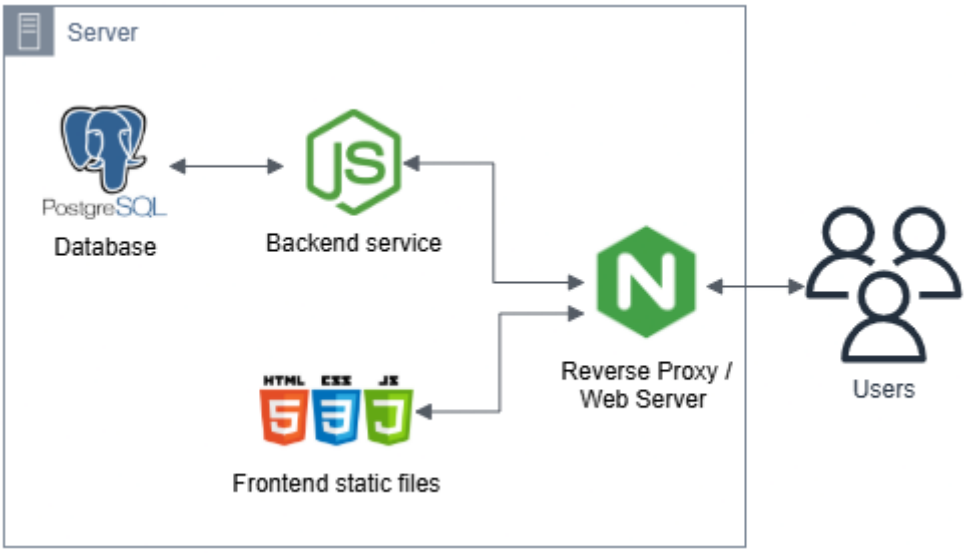

Fig. 24: Imagen de ejemplo de la solución 4.

Una vez contratada la VPS para el entorno de producción, se procedió a clonar los tres repositorios (backend, frontend y deployment). El despliegue se realizó utilizando un script de Bash para llevar a cabo el proceso de forma automatizada.

Gracias a la experiencia previa en el entorno de desarrollo y a la documentación generada durante el proceso, la instalación de las dependencias se llevó a cabo de manera rápida y eficiente. Además, se configuró una tarea programada (crontab) para ejecutar el script de backup de la base de datos cada domingo, almacenando la copia en una cuenta de Google Drive. La Fig. 25 presenta un diagrama de la arquitectura desplegada.

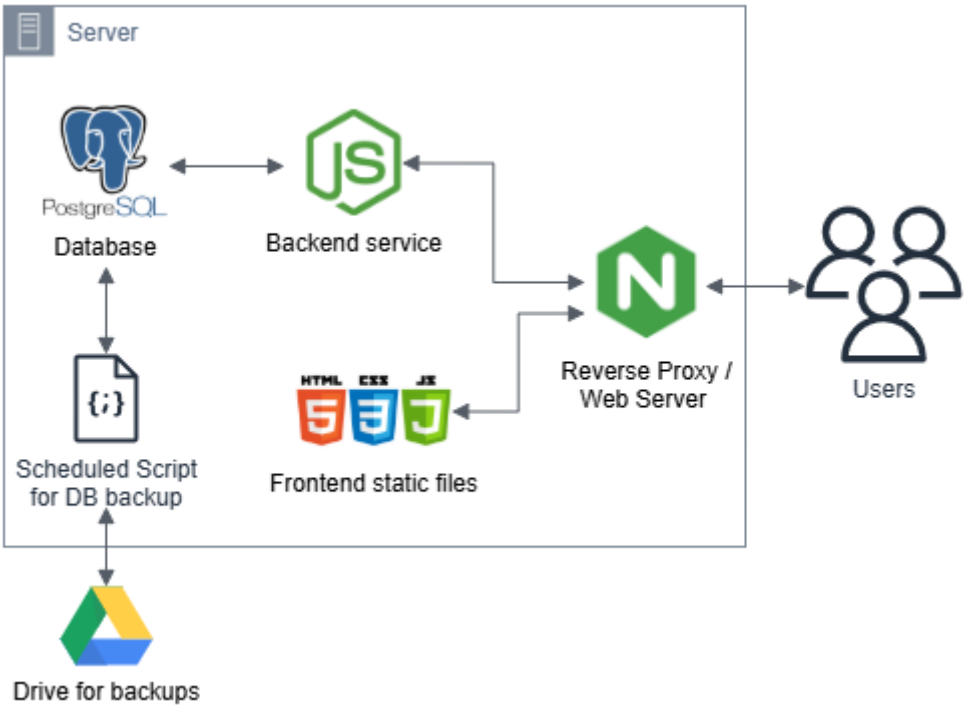

Fig. 25: Imagen de ejemplo de la solución 5.

En iteraciones posteriores del producto, la implementación de nuevas versiones de la aplicación se llevó a cabo ejecutando nuevamente el script de despliegue, que automatiza la actualización con la última versión del código en la rama *master*.

La Fig. 26 muestra una imagen de la aplicación desplegada y en funcionamiento, accesible a través de un navegador.

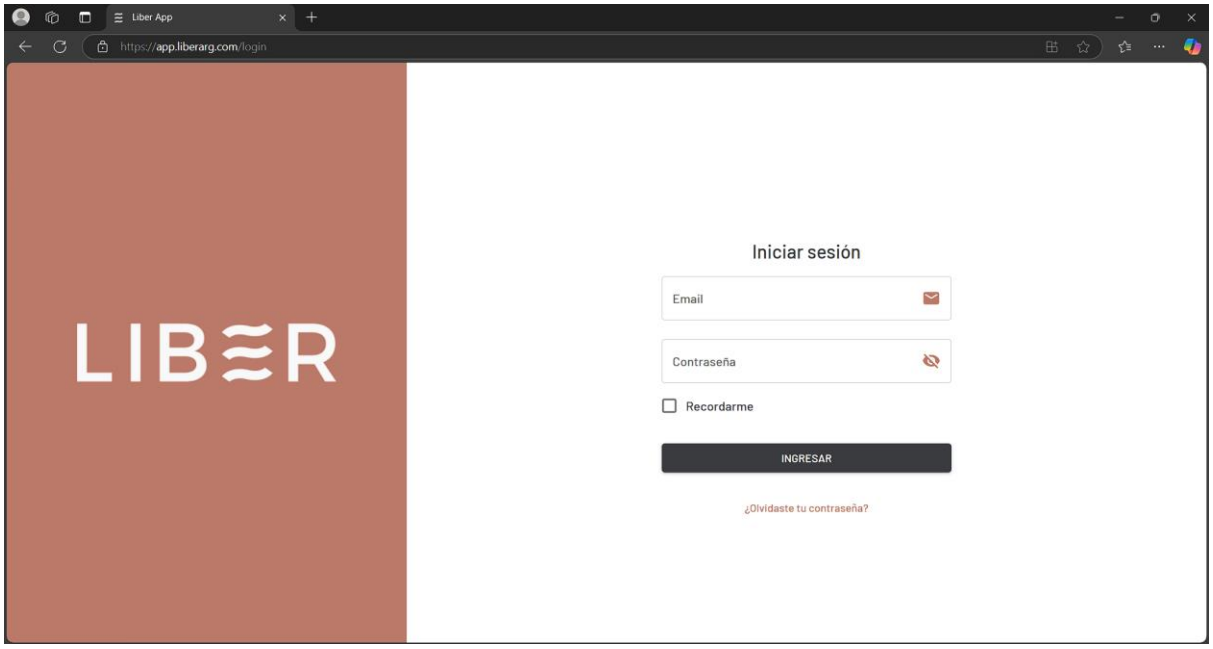

Fig. 26: Imagen de ejemplo de la solución 6.

4.3.2.2 CtesF5 Management

La experiencia en este proyecto fue similar al ejemplo anterior. En este caso, se utilizó una estructura de monorepositorio, donde tanto el backend como el frontend estaban bajo un mismo control de versiones. Para mejorar la gestión del despliegue y la documentación, se agregó una nueva carpeta destinada a los archivos de despliegue y su respectiva documentación. La Fig. 27 ilustra estas carpetas en GitHub, la plataforma de alojamiento de repositorios utilizada para este proyecto.

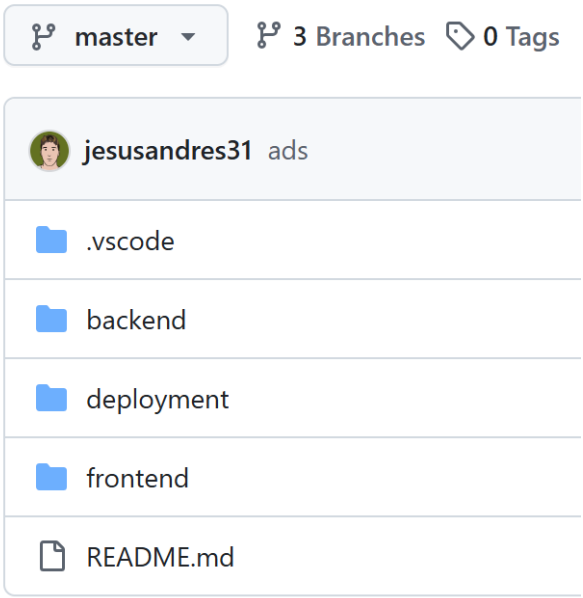

Fig. 27: Imagen de ejemplo de la solución 7.

Para completar el directorio de despliegue, se utilizó una plantilla de la guía, el cual fue adaptado según las necesidades del proyecto. Se ajustaron los scripts y los archivos de configuración del servidor Web y el Proxy inverso de acuerdo con los nombres y requisitos específicos de este caso. El directorio de despliegue quedó estructurado en los siguientes archivos y carpetas:

- **/cronjob**: Contiene archivos para la configuración de tareas programadas en el scheduler del sistema operativo.
    - **automate-db-dump.sh**: Script para realizar copias de seguridad de la base de datos.
- **/nginx**: Archivos de configuración del servidor Web y Proxy inverso.
    - **admin-ctesf5-app.duckdns.org.conf**: Archivo de configuración de Nginx.

- **ctesf5-app.duckdns.org.conf**: Archivo de configuración de Nginx.

- **/pb**: Archivos de configuración del servicio backend/database.

- **pocketbase.service**: Archivo de configuración del servicio backend para Linux.

- **/scripts**: Carpeta que almacena los scripts utilizados en el despliegue y tareas relacionadas.

- **deploy.sh**: Script para la instalación inicial y la implementación de nuevas versiones de la aplicación.

- **private.txt**: Archivo de texto que indica la ubicación de las credenciales y claves del proyecto.

- **README.md**: Documentación con la información necesaria para desplegar la aplicación y configurarla en distintos entornos.

La Fig. 23 muestra la estructura en detalle.

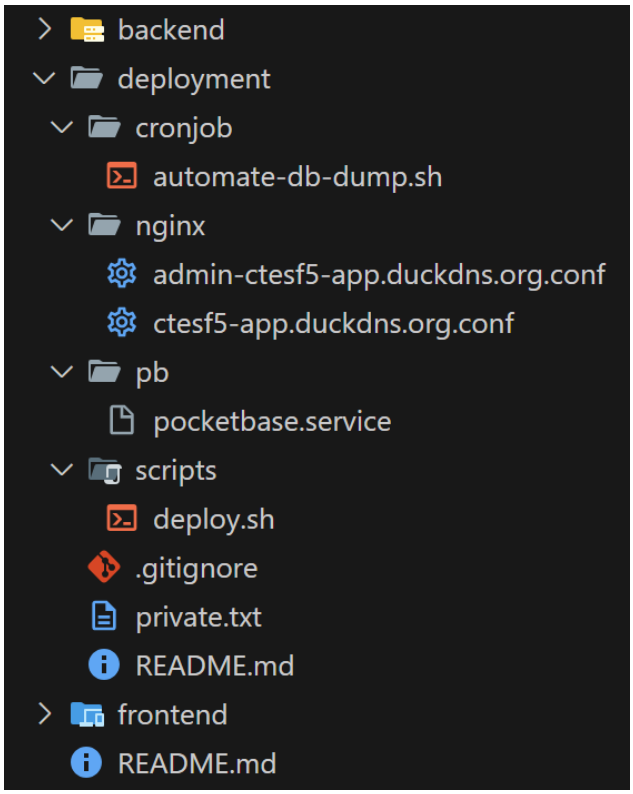

Fig. 28: Imagen de ejemplo de la solución 8.

Al igual que en el proyecto anterior, el despliegue se realizó en una máquina virtual (VPS). Se clonó el repositorio en la VPS de desarrollo y se instalaron las dependencias del proyecto, además de Nginx, que se configuró como Proxy inverso y servidor Web.

Para la gestión de la base de datos, se configuró una tarea programada (crontab) que ejecuta automáticamente un script de backup cada domingo, almacenando la copia en una cuenta de Google Drive. La Fig. 29 presenta un diagrama de la arquitectura desplegada.

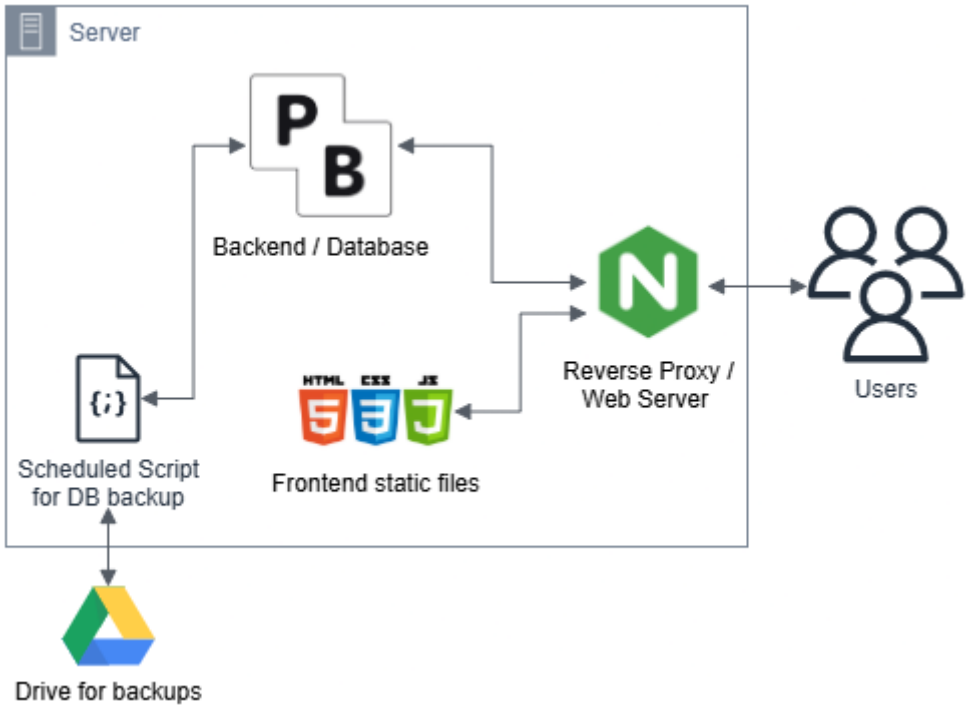

Fig. 29: Imagen de ejemplo de la solución 9.

En iteraciones posteriores, la actualización de nuevas versiones de la aplicación se llevó a cabo ejecutando nuevamente el script de despliegue, el cual implementaba automáticamente la última versión del código en la rama master.

La Fig. 30 muestra una imagen de la aplicación desplegada y en funcionamiento, accesible a través de un navegador.

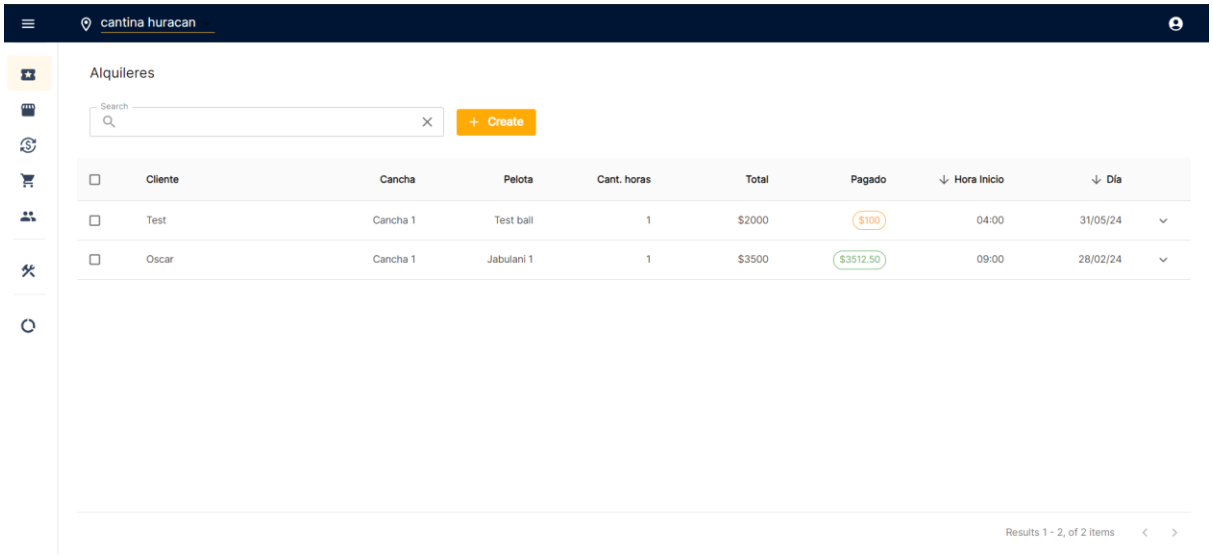

Fig. 30: Imagen de ejemplo de la solución 10.

4.3.2.3 Ddash - Docker Web Dashboard

La experiencia en este proyecto fue similar a los ejemplos anteriores, con la diferencia de que en este caso se utilizaron contenedores para la implementación. Se empleó una estructura de monorepositorio, donde tanto el backend como el frontend estaban bajo un mismo control de versiones. Para mejorar la gestión del despliegue y la documentación, se agregó una nueva carpeta destinada a los archivos de despliegue y su respectiva documentación. La Fig. 31 ilustra estas carpetas en GitHub, la plataforma de alojamiento de repositorios utilizada para este proyecto.

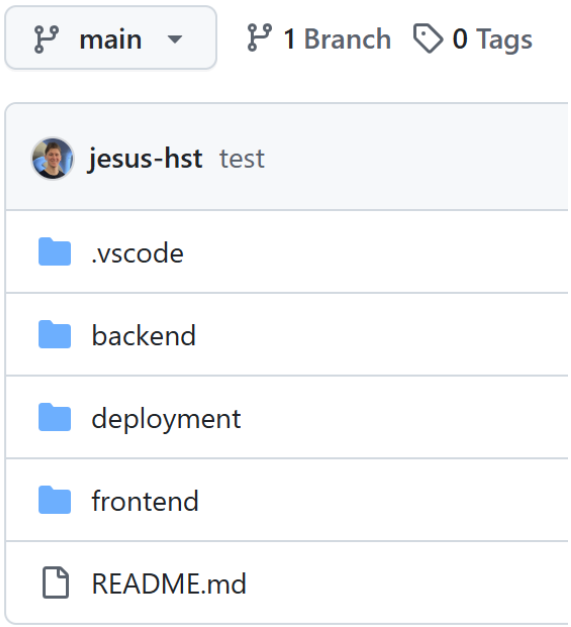

Fig. 31: Imagen de ejemplo de la solución 11.

Para completar esta carpeta, se utilizó una plantilla de la guía, el cual fue adaptado para incluir los scripts y archivos de configuración del servidor Web, el Proxy inverso y, adicionalmente, los archivos Dockerfile y Docker Compose configurados para los distintos entornos (desarrollo, QA y producción). El directorio de despliegue quedó estructurado en los siguientes archivos y carpetas:

- **/nginx**: Archivos de configuración del servidor Web y Proxy inverso.
  - **nginx.conf**: Archivo de configuración de Nginx.
  - **Dockerfile**: Archivo Dockerfile de configuración del contenedor de Nginx.
- **/scripts**: Carpeta que almacena los scripts utilizados en el despliegue y tareas relacionadas.
  - **deploy.sh**: Script para la instalación inicial y la implementación de nuevas versiones de la aplicación.
- **docker-compose.yml**: Archivo de Docker Compose para desplegar y sincronizar los diferentes contenedores/servicios de la aplicación.
- **README.md**: Documentación con la información necesaria para desplegar la aplicación y configurarla en distintos entornos.

La Fig. 32Fig. 23 muestra la estructura en detalle.

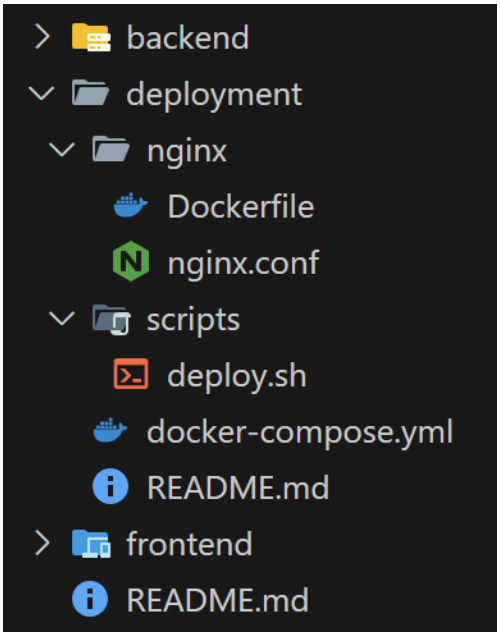

Fig. 32: Imagen de ejemplo de la solución 12.

Al igual que en el proyecto anterior, el despliegue se realizó en una máquina virtual (VPS). Se clonó el repositorio en la VPS de desarrollo, se instalaron las dependencias necesarias, y se configuró Nginx como Proxy inverso y servidor Web. Además, se instaló Docker, permitiendo la gestión de los contenedores y el despliegue de la aplicación dentro de ellos. La Fig. 33 presenta un diagrama de la arquitectura desplegada.

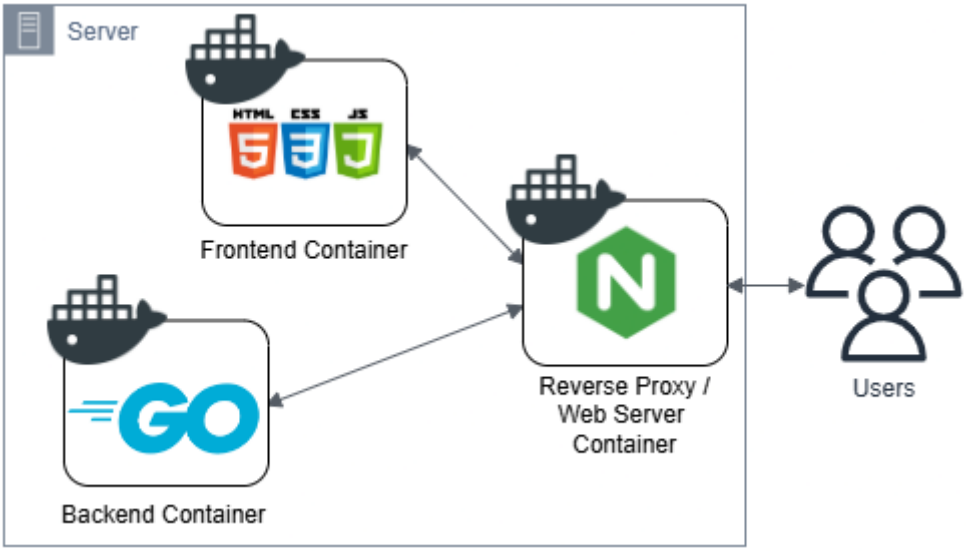

Fig. 33: Imagen de ejemplo de la solución 13.

En iteraciones posteriores, la actualización de nuevas versiones de la aplicación se realizó ejecutando nuevamente el script de despliegue, que automáticamente implementaba la última versión del código en la rama master.

La Fig. 34 muestra una imagen de la aplicación desplegada y en funcionamiento, accesible a través de un navegador.

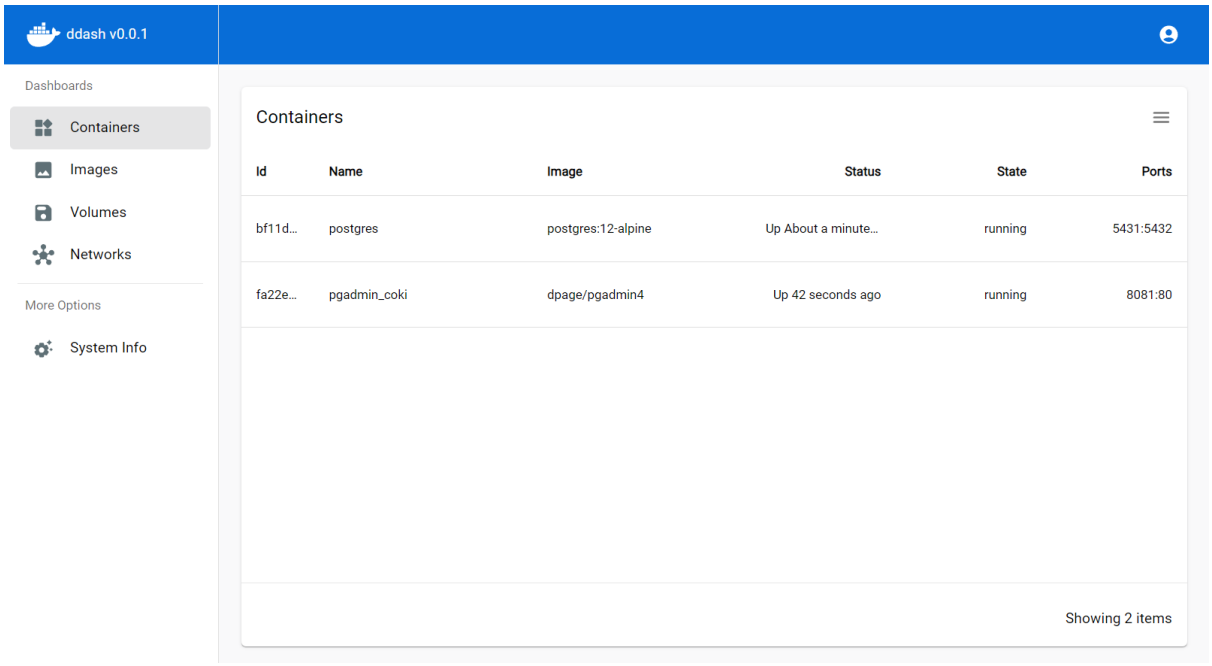

Fig. 34: Imagen de ejemplo de la solución 14.

4.3.3 Comparación del Proceso de Despliegue Con y Sin el Repositorio

Como se mencionó anteriormente, para evaluar si existieron mejoras en el proceso, el procedimiento para cada proyecto se llevó a cabo tanto con, como sin el repositorio de despliegue. En esta sección, se presentan las comparaciones de los resultados obtenidos.

4.3.3.1 Cantidad de Desarrolladores Involucrados en el Despliegue

La cantidad de desarrolladores involucrados en cada proyecto varió de entre dos a cinco, considerando únicamente al personal encargado del desarrollo de la aplicación Web. No se incluyeron diseñadores, desarrolladores móviles ni gerentes en este análisis.

La Tabla 5 muestra el número total de desarrolladores por proyecto y la cantidad de ellos que participaron en el proceso de despliegue o contribuyeron de alguna manera en el mismo, tanto con, como sin el uso del repositorio de despliegue:

Tabla 5: Desarrolladores involucrados en el despliegue por proyecto.

Nombre del Proyecto	Total de Desarrolladores	Desarrolladores Involucrados (Sin Repositorio de Despliegue)	Desarrolladores Involucrados (Con Repositorio de Despliegue)
Liber App	5	2	3
CtesF5 Management	3	1	3
Ddash - Docker Web Dashboard	2	1	2

Asimismo, en la Fig. 35 se aprecia visualmente el aumento en la cantidad de desarrolladores involucrados por proyecto.

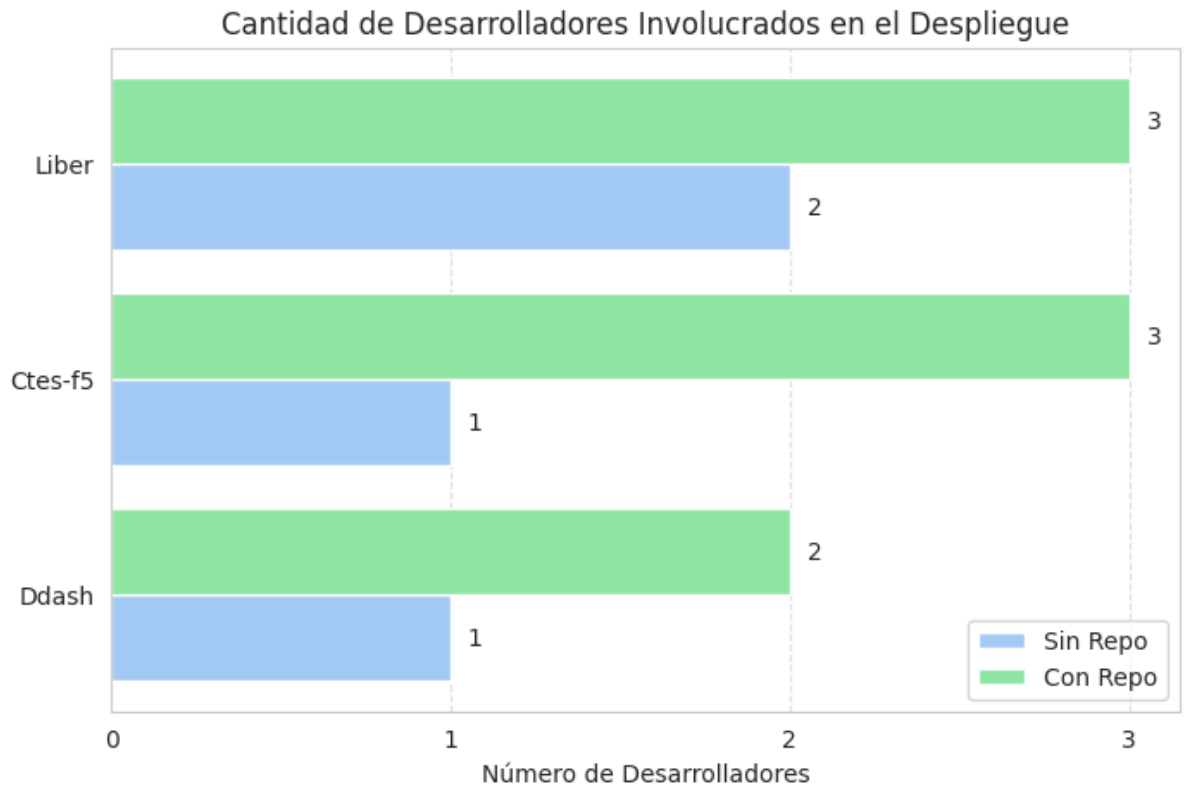

Fig. 35: Cantidad de desarrolladores involucrados en el despliegue.

Según los datos obtenidos en el gráfico, la cantidad de desarrolladores involucrados o con conocimiento del proceso de despliegue, en general se duplicó, evidenciando una mayor participación del equipo de desarrollo en tareas operativas clave.

4.3.3.1 Tiempo de Despliegue (Inicial y de Nuevas Versiones)

En cuanto al tiempo de despliegue, se recopiló información a partir de consultas a los equipos de desarrollo sobre el tiempo promedio, en minutos, requerido para desplegar la aplicación, tanto en su implementación inicial como en la publicación de nuevas versiones. Los resultados se presentan en la Fig. 36.

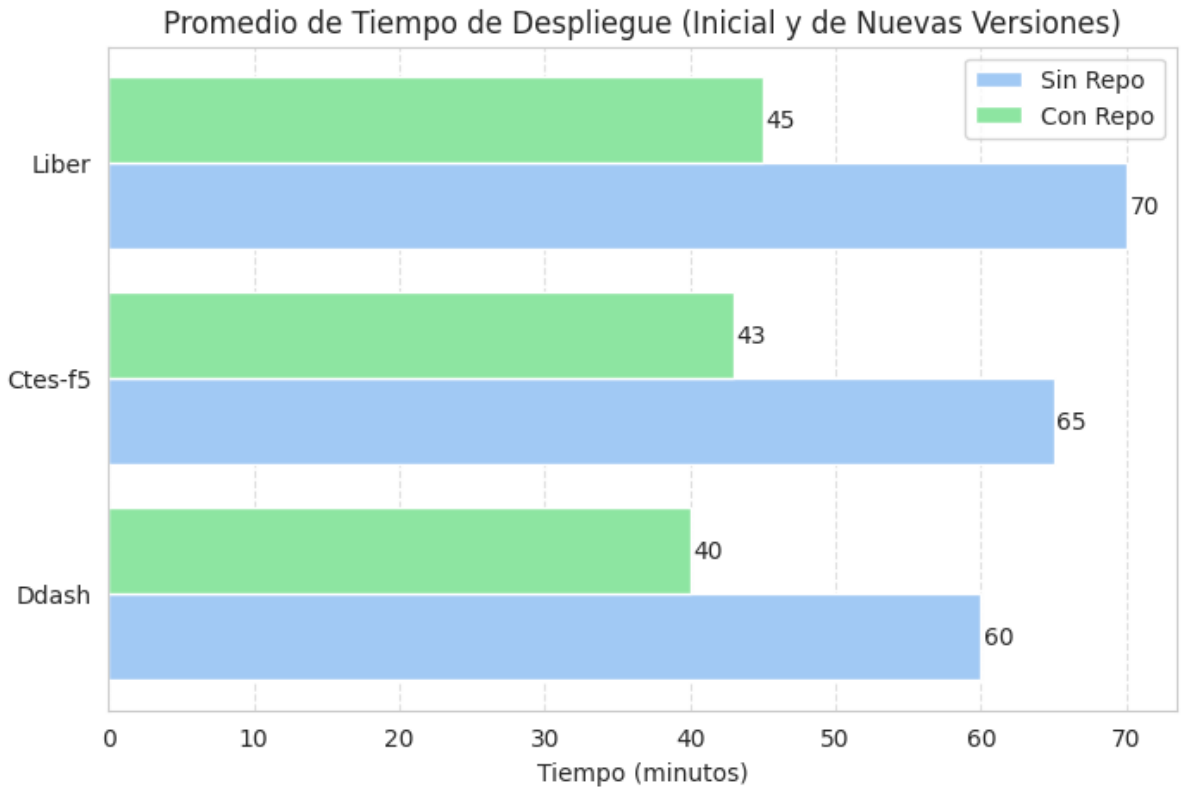

Fig. 36: Tiempo promedio de despliegue (minutos).

Según los datos obtenidos, se observa una reducción aproximada del 35 % en el tiempo requerido para llevar a cabo el proceso de despliegue, lo que refleja una mejora significativa en términos de eficiencia operativa.

4.3.3.1 Tiempo de Migración entre Entornos (Desarrollo, Prueba, y Producción)

De manera similar a la métrica anterior, se registró el tiempo promedio, en minutos, requerido para la migración del proyecto entre entornos (de desarrollo a pruebas y de pruebas a producción), basado en la experiencia reportada por los equipos para cada proyecto. Los resultados se presentan en la Fig. 37.

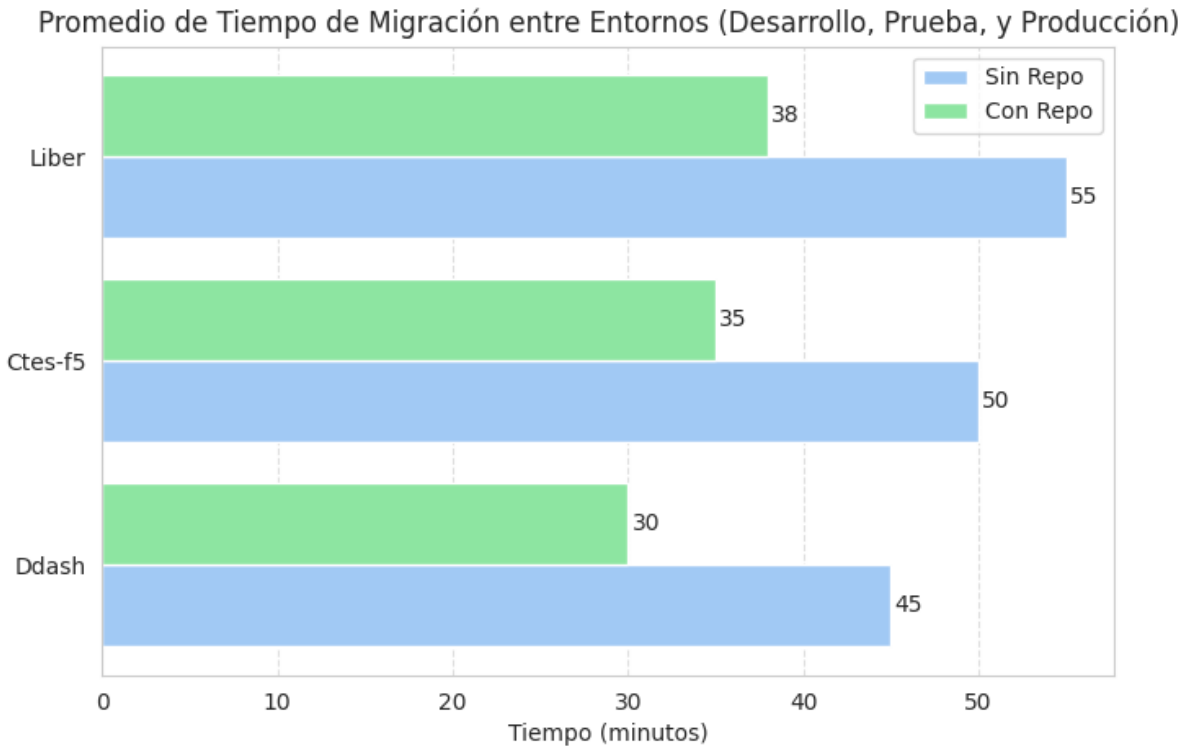

Fig. 37: Tiempo promedio de migración (minutos).

Según los datos obtenidos en el gráfico, se evidencia una reducción aproximada del 30 % en el tiempo necesario para realizar las migraciones entre entornos, lo cual contribuye a una mayor agilidad en el ciclo de vida del desarrollo.

4.3.4 Conclusiones de la Implementación en Entornos Reales

La validación de la solución en entornos reales ha demostrado la efectividad de la guía para la creación de repositorios de despliegue basados en GitOps en proyectos de pequeña escala. La implementación en tres proyectos con diferentes stacks tecnológicos evidenció la flexibilidad y adaptabilidad de la metodología.
A continuación, se destacan las principales ventajas identificadas durante este proceso:

- Repetibilidad y consistencia en distintos entornos: La metodología GitOps garantizó procesos consistentes y repetibles en todos los entornos, además de facilitar la participación de un mayor número de desarrolladores en el proceso de despliegue.

- Ahorro de tiempo en la transición entre entornos: La documentación del proceso de despliegue y la creación de scripts de automatización desde el primer uso redujeron

el tiempo necesario para configurar y realizar el despliegue en distintos entornos, logrando una disminución de aproximadamente un 30%.

- Automatización de tareas críticas: La adecuada estructuración de los repositorios permitió automatizar tareas como despliegues y copias de seguridad, mejorando la eficiencia, minimizando errores humanos, y resultando en un ahorro de tiempo de aproximadamente un 35%.

En conjunto, estos resultados subrayan el potencial de esta solución para optimizar los procesos de despliegue, permitiendo una gestión más eficiente y predecible de las aplicaciones, con una notable mejora en la calidad y velocidad de los despliegues.

5. Conclusiones y trabajos futuros

En este capítulo, se presenta la conclusión de este trabajo final de maestría. Se evalúa el cumplimiento de los objetivos establecidos y se resumen los logros obtenidos. Además, se mencionan posibles direcciones para futuras investigaciones.

5.1 Conclusiones

Este trabajo final de maestría abordó los desafíos del despliegue de aplicaciones Web en equipos pequeños de desarrollo de software, proponiendo una solución basada en la metodología GitOps para automatizar, documentar y optimizar la gestión del conocimiento a lo largo del ciclo de vida del proyecto. Se partió del análisis exhaustivo de las buenas prácticas de la industria y de la investigación de los métodos de despliegue más comunes en distintos stacks tecnológicos, lo que permitió identificar y adaptar actividades utilizadas tradicionalmente en equipos de gran tamaño para su aplicación en entornos más reducidos.
El proceso investigativo permitió consolidar el conocimiento adquirido en la creación de una guía detallada, que describe cómo estructurar un repositorio base de despliegue y adaptarlo al stack tecnológico de cada proyecto. La construcción iterativa e incremental de esta guía, basada en un ciclo de vida evolutivo, facilitó su ajuste a las necesidades reales de los equipos de desarrollo, y su implementación práctica en un equipo de una empresa local permitió validar la efectividad del enfoque propuesto, demostrando mejoras en la eficiencia, en la reducción de tiempos y en la colaboración interna.
A continuación, se evalúa el cumplimiento de cada uno de los objetivos específicos planteados en la tesis:

(OE1) *Analizar las buenas prácticas de la industria y seleccionar actividades aplicables en equipos pequeños:* Se realizó un análisis exhaustivo que identificó y adaptó las prácticas más efectivas de la industria, adecuándolas a las características y limitaciones de equipos de menor tamaño.

(OE2) *Definir una estrategia de despliegue basada en GitOps, estableciendo un repositorio centralizado para almacenar archivos de despliegue:* Se diseñó una estrategia integral fundamentada en GitOps, lo que permitió la creación de una guía para la creación un repositorio centralizado que integra archivos de configuración, scripts de automatización y documentación. La estrategia definida facilitó el manejo de despliegues y fortaleció la gestión del conocimiento.

(OE3) *Investigar los métodos de despliegue más utilizados en distintos stacks y elaborar una guía detallada con pasos clave, mejores prácticas y configuraciones recomendadas:* La investigación abarcó una variedad de métodos y herramientas utilizados en la industria, lo que permitió elaborar una guía detallada y adaptable a distintos entornos tecnológicos. La guía resultante contiene pasos claros, recomendaciones y configuraciones ajustadas a las necesidades de cada proyecto.

(OE4) *Desarrollar la guía de forma iterativa e incremental y publicarla en línea como recurso de código abierto*: La guía fue construida mediante un proceso iterativo que permitió su mejora continua, ajustándose a los requerimientos y feedback de los equipos de desarrollo. Su publicación en línea como recurso de código abierto facilita su actualización y difusión entre la comunidad.

(OE5) *Evaluar y optimizar el procedimiento propuesto mediante su implementación en un equipo real, identificando oportunidades de mejora*: La implementación práctica en un equipo de desarrollo de una empresa local permitió validar la estrategia propuesta. Los resultados demostraron que el uso de un repositorio centralizado simplifica el proceso de despliegue, reduce significativamente los tiempos de ejecución, actualización y migración, y mejora la colaboración entre los miembros del equipo.

Se concluye entonces que el trabajo aportó una solución viable y adaptable para el despliegue de aplicaciones Web en entornos de pequeña y mediana escala, fortaleciendo la gestión del conocimiento y la colaboración en equipos de desarrollo. Estos logros sientan las bases para futuras mejoras y para la expansión del enfoque propuesto en otros contextos de la industria del software.

5.2 Trabajos Futuros

Entre las posibles tareas en las que nos podemos enfocar para seguir mejorando la solución presentada, se destacan:

Ampliación de la Base de Conocimiento: Seguir incorporando ejemplos y casos prácticos a la guía para abordar una mayor diversidad de situaciones y tecnologías, manteniendo el contenido siempre actualizado. Además, continuar investigando y evaluando herramientas emergentes en el despliegue de aplicaciones Web, profundizando en prácticas DevOps para mejorar la eficiencia y la colaboración, y compartiendo estos conocimientos para su adopción por la comunidad.

Automatización y Mejora de la Experiencia del Usuario: Desarrollar herramientas, como un CLI o un script en línea, que permitan generar repositorios de despliegue para proyectos de manera automática, simplificando el proceso para los usuarios.

Continuar la Investigación sobre la Aplicación de GitOps en Equipos Pequeños**:** Profundizar en cómo estos equipos pueden adoptar GitOps de manera eficiente, teniendo en cuenta sus limitaciones y necesidades específicas. Asimismo, investigar estrategias para facilitar la creación de documentación en los procesos de despliegue, basándose en los desafíos identificados en la literatura académica utilizada en este trabajo.

Referencias

[1]. N. Tankovi´c et al. “Transforming Vertical Web Applications Into Elastic Cloud Applications”. 2015. IEEE

[2]. “Promoting Jira configuration from development to production”. [Online]. Available: https://confluence.atlassian.com/adminjiraserver/promoting-jira-configuration-from-development-to-production-1167739614.html

[3]. “What is CI/CD?”. [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[4]. “What is a Container?”. [Online]. Available: https://www.docker.com/resources/what-container

[5]. “Site Reliability Engineering”. [Online]. Available: https://sre.google/sre-book/introduction/

[6]. "A Lightweight Guide to the Theory and Practice of Scrum”. Version 2.0. 2012 Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde

[7]. J. Zini. “Guía y repositorio para el despliegue de aplicaciones Web en servidores Linux utilizando contenedores”. Universidad Nacional del Nordeste. Corrientes, Argentina. 2023

[8]. “What is Git?”. [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

[9]. A. M. Joy. "Performance Comparison Between Linux Containers and Virtual Machines”. International Conference on Advances in Computer Engineering and Applications (ICACEA). Ghaziabad, India. 2015

[10]. “Docker”. [Online]. Available: https://docs.docker.com/get-started/docker-overview/

[11]. “Docker: Run your app in production”. [Online]. Available: https://docs.docker.com/get-started/orchestration/

[12]. “Jenkins”. [Online]. Available: https://www.jenkins.io/

[13]. “GitHub Actions”. [Online]. Available: https://github.com/features/actions

[14]. “CircleCI”. [Online]. Available: https://circleci.com/

[15]. “Travis CI”. [Online]. Available: https://www.travis-ci.com/

[16]. “OpenGitOps”. [Online]. Available: https://opengitops.dev/

[17]. “Node”. [Online]. Available: https://nodejs.org/

[18]. “React”. [Online]. Available: https://react.dev/

[19]. “PostgreSQL”. [Online]. Available: https://www.postgresql.org/

[20]. A. Dagnino. "An Evolutionary Lifecycle Model with Agile Practices for Software Development at ABB". ABB US Corporate Research Center. Raleigh, NC, USA. 2002

[21]. I. Nonaka & H. Takeuchi. "The Knowledge-Creating Company". USA. Oxford University. 1995

[22]. F. Beetz & S. Harrer. "GitOps: The Evolution of DevOps?". IEEE Software Volume: 39, Issue: 4, July-Aug. 2022

[23]. "Cloud Native Computing Foundation (CNCF)". [Online]. Available: https://www.cncf.io/

[24]. T. A. Limoncelli. "GitOps: a path to more self-service IT". Communications of the ACM Volume 61, Number 9. 2018

[25]. "LXC: Linux Containers". [Online]. Available: https://linuxcontainers.org/

[26]. "Podman Container". [Online]. Available: https://podman.io/

[27]. Y. Li, & Y. Xia. "Auto-scaling Web applications in hybrid cloud based on Docker". 5th International Conference on Computer Science and Network Technology. 2016

[28]. "Container and virtualization tools". [Online]. Available: https://sre.google/sre-book/introduction/

[29]. "What is Docker Hub?". [Online]. Available: https://www.docker.com/products/docker-hub/

[30]. "GitLab Container Registry". [Online]. Available: https://docs.gitlab.com/ee/user/packages/container_registry/

[31]. J. Shah, & D. Dubaria. "Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform". IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). 2019

[32]. "Red Hat". [Online]. Available: https://www.redhat.com

[33]. "AWS | Cloud Computing". [Online]. Available: https://aws.amazon.com/

[34]. "Microsoft Azure: Cloud computing services". [Online]. Available: https://azure.microsoft.com/

[35]. "Google Cloud Platform". [Online]. Available: https://cloud.google.com/

[36]. M. Villamizar et al. "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud". 2015. IEEE

[37]. "What is Infrastructure as Code (IaC)?". [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

[38]. B. Piedade, J. P. Dias & F. F. Correia. "An Empirical Study on Visual Programming Docker Compose Configurations". Faculty of Engineering, University of Porto INESC TEC Porto, Portugal. 2020

[39]. M. Virmani. "Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery". Fifth international conference on Innovative Computing Technology (INTECH 2015). 2015

[40]. "Patterns for Managing Source Code Branches". [Online]. Available: https://martinfowler.com/articles/branching-patterns.html

[41]. "GitLab CI/CD". [Online]. Available: https://docs.gitlab.com/ci/

[42]. T. Theunissen, S. Hoppenbrouwers, & S. Overbeek. "Approaches for Documentation in Continuous Software Development". Netherlands, Arnhem, University of Applied Sciences, 2022.

[43]. I.Bucena & M. Kirikova. "Simplifying the DevOps Adoption Process". Riga Technical University, 2017.

[44]. A. Nordström. "Exploring GitOps for Smaller-Scale Applications". Sweden, Åbo Akademi, Fakulteten för naturvetenskaper och teknik, 2024.

[45]. M. Virmani. "Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery". Fifth international conference on Innovative Computing Technology (INTECH), 2015.

[46]. Y. Li & Y. Xia. "Auto-Scaling Web Applications in Hybrid Cloud Based on Docker". Beihang University, Beijing, China. 5th International Conference on Computer Science and Network Technology (ICCSNT), 2016.

[47]. "Kubernetes". [Online]. Available: https://kubernetes.io/

[48]. "Argo CD". [Online]. Available: https://argo-cd.readthedocs.io/

[49]. G. Kim, P. Debois, & J. Willis. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations". Portland, OR: IT Revolution Press, 2016.

[50]. "Helm". [Online]. Available: https://helm.sh/

[51]. "AWS Elastic Load Balancing". [Online]. Available: https://aws.amazon.com/elasticloadbalancing/

[52]. "Linkerd". [Online]. Available: https://linkerd.io/

[53]. "Istio". [Online]. Available: https://istio.io/

[54]. "Cloudformation". [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

[55]. "Terraform". [Online]. Available: https://www.terraform.io/

[56]. "Apache Kafka". [Online]. Available: https://kafka.apache.org/

[57]. "RabbitMQ". [Online]. Available: https://www.rabbitmq.com/

[58]. “Confluence”. [Online]. Available: https://www.atlassian.com/es/software/confluence

[59]. “NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy”. [Online]. Available: https://www.nginx.com/

[60]. “CertBot”. [Online]. Available: https://certbot.eff.org/

[61]. “Let’s Encrypt”. [Online]. Available: https://letsencrypt.org/

[62]. “Bash”. [Online]. Available: https://opensource.com/resources/what-bash

[63]. “Google Drive”. [Online]. Available: https://www.google.com/drive/

[64]. “Bitwarden”. [Online]. Available: https://bitwarden.com/

[65]. “Docusaurus”. [Online]. Available: https://docusaurus.io/

[66]. “GitHub”. [Online]. Available: https://github.com/

[67]. “Introducing ChatGPT”. [Online]. Available: https://openai.com/index/chatgpt/

[68]. “GitHub Pages”. [Online]. Available: https://pages.github.com/

[69]. “Netlify”. [Online]. Available: https://www.netlify.com/

[70]. “Cloudflare Pages”. [Online]. Available: https://pages.cloudflare.com/