UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCIÓN - CHILE

"Implementación de una Base Funcional DevOps para el Desarrollo Colaborativo de la Aplicación EverPlanApp"

ALAN DANIEL MAC-KAY VARGAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO EN INFORMÁTICA

Profesor Guía: Cristian Antonio Lara Valenzuela

Diciembre - 2024

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Implementación de una Base Funcional DevOps para el Desarrollo Colaborativo de la Aplicación EverPlanApp

Nombre del candidato(a): Alan Daniel Mac-kay vargas

Carrera / Grado: Ingeneria Informatica

Campus: Concepcion **;** **Departamento**: Electrónica e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Cristian Antonio Lara Valenzuela, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: **26/05/2026** ; **Firma:** ________________________

Estudiante o Candidato(a):

Fecha: 26/052026 ; **Firma:** ________________________

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Mirando en retrospectiva, resulta difícil resumir en pocas líneas todo lo que significó este camino universitario. Fue una etapa larga, marcada por aprendizajes, desafíos, errores y logros. Por ello, quiero dedicar este trabajo, de manera muy especial, a mi familia, quienes fueron un apoyo constante durante todo este recorrido. Gracias por la paciencia, el cariño y la confianza incondicional, incluso en los momentos de mayor cansancio y dificultad. Su apoyo fue fundamental para poder seguir adelante y cerrar una etapa tan importante de mi vida.

Asimismo, quiero dedicar este trabajo a un compañero de camino que estuvo presente durante todos mis años universitarios. Gracias por estar siempre ahí, por el apoyo constante, la amistad y por compartir cada etapa de este proceso junto a tu familia. Sin duda, su compañía hizo este recorrido mucho más llevadero y significativo. Gracias, Sebastián Gallardo.

AGRADECIMIENTOS

De manera especial, quiero destacar y agradecer a mis compañeros y amigos con quienes compartí gran parte de mi etapa universitaria: Felipe Badilla, Fernando Cruces, Javier Choque, Ignacio Hormazábal, Dismar Quiroz, Alan Mac-Kay, Gabriel Farfán, Matías Torres y Sebastián Gallardo. Gracias por los momentos compartidos, por el apoyo constante, las conversaciones random, las risas y por estar presentes tanto en los buenos momentos como en los más difíciles.

Asimismo, quiero agradecer a Sebastián Gallardo y Michelle Rocha, con quienes desarrollé este proyecto. Durante todo el año trabajamos juntos con muchas ganas y compromiso, buscando hacer algo que realmente tuviera sentido para las personas. El proceso no siempre fue fácil, pero el trabajo en equipo, el apoyo mutuo y las conversaciones que tuvimos hicieron que esta experiencia fuera muy valiosa, tanto en lo académico como en lo personal.

RESUMEN

Esta memoria aborda la implementación de una base funcional DevOps para el desarrollo colaborativo de EverPlanApp, una aplicación diseñada para apoyar la organización académica de estudiantes universitarios neurodivergentes, especialmente aquellos con TDAH o TEA. Desde el rol de estudiante en formación profesional, se identificó la necesidad de establecer una infraestructura técnica que facilitara el trabajo en equipo, mejorara la calidad del código y redujera errores mediante automatización de tareas.

Para ello, se propusieron y desarrollaron procesos esenciales como la estructuración de repositorios en GitHub, la integración y despliegue continuo mediante GitHub Actions, la creación de entornos locales reproducibles con Docker, y la documentación técnica clara para nuevos colaboradores. Esta base técnica permitió validar un flujo de trabajo eficiente y escalable, alineado con buenas prácticas de ingeniería de software. La solución desarrollada representa un aporte significativo tanto para el equipo de desarrollo como para la evolución técnica de EverPlanApp.

Palabras clave: DevOps; CI/CD; Docker.

ABSTRACT

***Abstract*— ABSTRACT**

This thesis addresses the implementation of a functional DevOps foundation for the collaborative development of EverPlanApp, an application designed to support the academic organization of neurodivergent university students, especially those with ADHD or ASD. From the perspective of a student in professional training, the need to establish a technical infrastructure that facilitates teamwork, improves code quality, and reduces errors through task automation was identified.

To achieve this, essential processes were proposed and developed, such as repository structuring in GitHub, continuous integration and deployment through GitHub Actions, the creation of reproducible local environments using Docker, and clear technical documentation for new collaborators. This technical foundation made it possible to validate an efficient and scalable workflow aligned with good software engineering practices. The developed solution represents a significant contribution both to the development team and to the technical evolution of EverPlanApp.

***Keywords*—** DevOps; CI/CD; Docker.

GLOSARIO

Accesibilidad Sensorial: Enfoque de diseño orientado a reducir la sobrecarga cognitiva y sensorial en interfaces digitales.

Amazon EC2 (Elastic Compute Cloud): Servicio de Amazon Web Services que provee instancias virtuales configurables para ejecutar aplicaciones en la nube.

Amazon RDS (Relational Database Service): Servicio administrado de bases de datos relacionales de Amazon Web Services.

API (Application Programming Interface): Conjunto de definiciones y protocolos que permiten la comunicación entre distintos componentes de software.

Arquitectura DevOps: Estructura técnica que integra desarrollo, automatización y operaciones para el ciclo de vida del software.

Backend: Componente del sistema encargado de la lógica de negocio y del acceso a los datos.

Base de Datos: Sistema estructurado para el almacenamiento y gestión de información.

Base Funcional DevOps: Conjunto mínimo operativo de prácticas y herramientas DevOps implementadas en el proyecto.

BitKeeper: Sistema de control de versiones propietario, predecesor del uso de Git en el desarrollo del kernel Linux.

CD (Continuous Delivery / Continuous Deployment): Práctica que garantiza que el software esté listo para ser desplegado de forma automatizada.

CI (Continuous Integration): Práctica DevOps que integra cambios de código de forma frecuente y automatizada.

Cloud Computing (Computación en la Nube): Modelo de provisión de recursos informáticos a través de internet.

Commit: Registro de un conjunto de cambios realizados en un repositorio Git.

Contenedor: Unidad ejecutable aislada que encapsula una aplicación y sus dependencias.

Control de Versiones: Mecanismo para gestionar y rastrear cambios en el código fuente.

DevOps: Enfoque que integra desarrollo y operaciones para automatizar y mejorar el ciclo de vida del software.

Docker Engine: Motor que ejecuta y administra contenedores Docker.

Escalabilidad: Capacidad de un sistema para crecer o adaptarse según la demanda.

Feature Toggle: Mecanismo para activar o desactivar funcionalidades sin desplegar nuevo código.

Flujo de Trabajo (Workflow): Conjunto de reglas que organizan el desarrollo colaborativo.

Frontend: Capa de presentación e interacción con el usuario.

Git: Sistema de control de versiones distribuido.

GitHub: Plataforma de alojamiento de repositorios Git y colaboración.

GitHub Flow: Modelo de flujo de trabajo simple basado en una rama principal y pull requests.

Hotfix: Rama destinada a correcciones urgentes en producción.

IaaS (Infrastructure as a Service): Modelo de provisión de infraestructura en la nube.

Jenkins: Herramienta de automatización utilizada para implementar pipelines CI/CD.

Kernel: Núcleo del sistema operativo compartido por los contenedores.

Main (rama): Rama principal y estable de un repositorio Git.

Máquina Virtual: Entorno virtual que ejecuta un sistema operativo completo.

Pipeline: Flujo automatizado de etapas para construir, probar y desplegar software.

PostgreSQL: Sistema de gestión de bases de datos relacional.

Ramificación (Branching): Creación de líneas paralelas de desarrollo.

RDS (Relational Database Service): Servicio administrado de bases de datos relacionales en AWS.

SSH (Secure Shell): Protocolo seguro para acceso remoto a servidores.

TDAH (Trastorno por Déficit de Atención e Hiperactividad): Condición neurodivergente caracterizada por inatención e impulsividad.

TEA (Trastorno del Espectro Autista): Condición del neurodesarrollo que afecta la comunicación y el comportamiento.

Trunk-Based Development: Modelo de desarrollo centrado en una sola rama principal.

Variables de Entorno (.env): Parámetros externos que configuran el comportamiento de la aplicación.

VCS (Version Control System): Sistema para gestionar versiones de archivos digitales.

INDICE DE CONTENIDOS

Contenido

RESUMEN 4

ABSTRACT 4

GLOSARIO 5

INDICE DE CONTENIDOS 8

INDICE DE FIGURAS 11

INDICE DE TABLAS 11

INTRODUCCIÓN 13

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 14

1.1 Contexto del proyecto 14

1.2 Situación Actual 15

1.3 Actores Involucrados 15

1.4 Justificación 15

1.5 Objetivos 16

2.1 Configurar control de versiones colaborativo con Git y GitHub. 18

2.1.1 Fundamentos del Control de Versiones 18

2.1.2 Sistemas de control de versiones: centralizado vs distribuido 19

2.1.3 Git como sistema de control distribuido 20

2.1.4 Comparación entre Git y otros sistemas 21

2.1.5 El concepto de ramas en Git 23

2.1.6 Modelos de flujo de trabajo con ramas 24

2.1.7 Monorepo y multirepo: enfoques para la organización del código en proyectos modulares 26

2.2 Contenerizar el entorno completo usando Docker y Docker Compose. 27

2.2.1 Docker: tecnología base para entornos aislados y portables 27

2.2.2 Docker como base para flujos DevOps 27

2.2.3 Contenedores en Docker 28

2.2.4 Docker Compose como herramienta de definición del entorno **29**

2.2.5 Comparación entre contenedores y otras tecnologías de virtualización **30**

2.3 Implementar integración y entrega continua (CI/CD) con Jenkins 31

2.3.1 Integración Continua (CI) **31**

2.3.2 Entrega Continua y Despliegue Automatizado (CD) **32**

2.3.3 Pipelines CI/CD **34**

2.4 Despliegue en la nube para entornos DevOps 35

2.4.1 Fundamentos del cómputo en la nube **35**

CAPÍTULO 3: PROPUESTA DE SOLUCION **40**

3.1 Evolución de la arquitectura DevOps de EverPlanApp 40

3.2 Decisiones de infraestructura y entorno 41

3.3 Contenerización del backend: dificultades y ajustes 42

3.5 Despliegue en la nube: configuración, pruebas y problemas .... **Error! Bookmark not defined.**

3.6 Aprendizajes, limitaciones y proyección 46

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN **48**

4.1 Enfoque de validación de la solución 48

4.2 Validación del flujo de Integración y Despliegue Continuo (CI/CD) 49

4.3 Validación de la contenedorización del backend 51

4.4 Validación del despliegue en la nube 52

4.4.1 Configuración del entorno en la nube **52**

4.4.3 Análisis de la validación en la nube **53**

4.5 Análisis de resultados y limitaciones 53

CAPÍTULO 5: CONCLUSIONES **55**

5.1 Cumplimiento de los objetivos y validez de la solución 55

5.2 Aprendizajes obtenidos durante el desarrollo 55

5.3 Limitaciones encontradas 55

5.4 Proyecciones y trabajo futuro 56

REFERENCIAS BIBLIOGRÁFICAS 57

INDICE DE FIGURAS

Figura 1:Ejemplo de flujo de ramas en Git. Fuente: Chacon y Straub (2014), Pro Git (2.ª ed.), capítulo “Branching Workflows”. 23

Figura 2: Diagrama general sobre la arquitectura y propósito de Docker. Fuente: Docker (2024). Recuperado de https://docs.docker.com/get-started/overview/........ 28

Figura 3: Flujo general de Integración Continua y Entrega Continua (CI/CD). Fuente: Adaptado de Humble y Farley (2010)........ 32

Figura 4: Rol del servidor Jenkins dentro de un flujo CI/CD. Fuente: Jenkins Documentation (2024). 33

Figura 5: Estructura general de un pipeline CI/CD. Fuente: Jenkins Documentation (2024). 34

Figura 6: Arquitectura conceptual de la solución DevOps implementada para el backend de EverPlanApp. Fuente: Elaboración propia. 40

Figura 7: Arquitectura conceptual de la solución DevOps implementada para el backend de EverPlanApp. Fuente: Elaboración propia 43

Figura 8: Flujo CI/CD del backend de EverPlanApp. Fuente: Elaboración propia 44

Figura 9: Ejecución exitosa del pipeline CI/CD en Jenkins para EverPlanApp. Fuente: Elaboración propia. 49

Figura 10: Ejecución activa del contenedor Docker del backend de EverPlanApp. Fuente: Elaboración propia. 51

Figura 11: Registro de logs del contenedor Docker del backend desplegado en una instancia EC2, evidenciando el inicio correcto del servicio y su disponibilidad en el puerto 8000. Fuente: Elaboración propia. 53

INDICE DE TABLAS

Tabla 1: Tabla de objetivos específicos. Fuente: Elaboración Propia 17

Tabla 2: Análisis comparativo de sistemas de control de versiones. Fuente: Elaboración Propia .. 22

Tabla 3: Comparación general de modelos de flujo de trabajo con ramas en Git. Fuente: Elaboración propia 25

Tabla 4: Comparación entre contenedores y máquinas virtuales. Fuente: Elaboración propia. 30

Tabla 5: Comparación entre modelos de despliegue en la nube: IaaS, PaaS y VPS. Fuente: Elaboración propia, basada en NIST (2011), Docker (2024) y Humble y Farley (2010). 38

Tabla 6: Pipeline CI/CD implementado para EverPlanApp con Jenkins y Docker Fuente: Elaboración propia. 50

Tabla 7: Validación de la contenedorización del backend de EverPlanApp Fuente: Elaboración propia. 51

INTRODUCCIÓN

La organización académica es un desafío constante para estudiantes universitarios, especialmente para quienes presentan condiciones neurodivergentes como el Trastorno por Déficit de Atención e Hiperactividad (TDAH) o el Trastorno del Espectro Autista (TEA). EverPlanApp surge como una solución tecnológica destinada a mejorar la gestión del tiempo, tareas y rutinas, incorporando principios de accesibilidad sensorial y autorregulación emocional.

En este contexto, el subproyecto abordado en la presente memoria consiste en la implementación de una base funcional DevOps que permite establecer procesos básicos de integración continua, despliegue automatizado y gestión de entornos para el desarrollo colaborativo de EverPlanApp. Este enfoque busca optimizar el flujo de trabajo del equipo, reducir errores manuales en los procesos de integración y despliegue, y facilitar la evolución técnica del sistema mediante prácticas alineadas con la ingeniería de software moderna.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Contexto del proyecto

En el contexto de la educación superior, la organización académica representa un desafío constante para gran parte del estudiantado. Esta dificultad se acentúa en estudiantes neurodivergentes, como aquellos que presentan Trastorno por Déficit de Atención e Hiperactividad (TDAH) o Trastorno del Espectro Autista (TEA), quienes suelen enfrentar barreras adicionales relacionadas con la gestión del tiempo, la estructuración de tareas y la autorregulación emocional.

A pesar de la existencia de múltiples herramientas digitales orientadas a la planificación, muchas de ellas no consideran las particularidades cognitivas ni sensoriales de este grupo de estudiantes. Interfaces recargadas, elementos de gamificación o flujos poco intuitivos pueden generar sobrecarga sensorial o desmotivación, dificultando aún más su experiencia académica.

Frente a esta realidad, surge la necesidad de desarrollar soluciones tecnológicas inclusivas, capaces de adaptarse a distintos estilos de aprendizaje y formas de procesar la información. En este escenario, se enmarca el desarrollo de *EverPlanApp*, una aplicación diseñada específicamente para apoyar la planificación académica de estudiantes neurodivergentes.

El objetivo específico que se aborda en este apartado de la memoria es implementar una base técnica DevOps que facilite la automatización y organización eficiente de los procesos de integración, despliegue y gestión del entorno de desarrollo de EverPlanApp. Esta necesidad surge debido a la complejidad del proyecto, que integra múltiples componentes —frontend, backend y base de datos— y requiere mecanismos formales para evitar problemas de coordinación, errores recurrentes durante la integración de cambios y pérdida de trazabilidad en el desarrollo. En este contexto, el subproyecto DevOps se enfoca en crear un entorno colaborativo basado en prácticas de integración continua (CI), despliegue automatizado (CD), control de versiones y gestión de entornos mediante contenedores, utilizando herramientas como Git, Docker y Jenkins. Todo esto con el fin de asegurar un flujo de trabajo ordenado, reproducible y alineado con las mejores prácticas actuales de la ingeniería de software.

1.2 Situación Actual

Actualmente, herramientas como Google Calendar, Trello o Notion son ampliamente utilizadas por estudiantes para organizar sus tareas, rutinas y compromisos académicos. Sin embargo, muchas de estas plataformas pueden representar una barrera para personas con condiciones neurodivergentes, como aquellas dentro del espectro autista. Esto se debe a que algunas de estas aplicaciones incorporan elementos de gamificación o interfaces visualmente cargadas con el objetivo de hacer más atractiva la experiencia de usuario. No obstante, este tipo de estímulos pueden resultar sobreestimulantes o generar rechazo en usuarios autistas, quienes suelen preferir entornos visuales más simples, con menos elementos distractores, colores suaves y estructuras claras. En general, estas aplicaciones carecen de un enfoque inclusivo y un diseño sensorialmente accesible que minimice la sobrecarga cognitiva y emocional.

1.3 Actores Involucrados

Los principales beneficiarios de esta solución son estudiantes universitarios neurodivergentes, quienes enfrentan mayores desafíos en la gestión de sus actividades académicas. No obstante, la aplicación también puede ser útil para cualquier estudiante que requiera apoyo en su organización personal. Asimismo, se consideran actores secundarios a tutores, docentes y profesionales del ámbito psicopedagógico, quienes podrían utilizar la información generada por la aplicación como una herramienta complementaria para fortalecer el acompañamiento académico y la toma de decisiones pedagógicas.

1.4 Justificación

Justificar el desarrollo de una solución como EverPlanApp implica reconocer que muchas aplicaciones actuales, aunque resultan funcionales para el público general, no siempre consideran las necesidades de usuarios neurodivergentes. Interfaces con gamificación, colores intensos o elementos visuales recargados pueden generar sobrecarga sensorial en estudiantes autistas, provocando rechazo en lugar de apoyo. Frente a esta realidad, se vuelve necesario diseñar herramientas más simples, estables y predecibles que realmente acompañen sus procesos de organización.

1.5 Objetivos

Objetivo General:
Desarrollar procesos básicos de integración y despliegue para facilitar el trabajo del equipo de desarrollo y asegurar que EverPlanApp se pueda probar y actualizar de forma ordenada.

Objetivo Específico	Marco Conceptual	Propuesta de Solución
Configurar control de versiones colaborativo con Git y GitHub.	- Control de versiones con Git.- Buenas prácticas en estructuras de repositorios (monorepo vs multirepo).- Convenciones de ramas (main, dev, feature).	- Crear un repositorio con una estructura clara que permita el control de versiones del proyecto, definiendo el backend y la base de datos como componentes gestionados por el subproyecto DevOps.
Contenerizar el entorno completo usando Docker y Docker Compose.	- Virtualización ligera mediante contenedores con Docker.- Definición declarativa del entorno de ejecución de aplicaciones mediante Docker Compose. - Uso de archivos Dockerfile para definir entornos reproducibles y consistentes.	- Crear archivos Dockerfile para backend - Definir un archivo docker-compose.yml para la orquestación del backend en el entorno de desarrollo. - Configurar variables de entorno para la conexión del backend con la base de datos gestionada en RDS. - Permitir el levantamiento del backend de forma local mediante docker-compose up para pruebas y validación.
Implementar integración y entrega continua (CI/CD) con Jenkins	- Automatización de tareas mediante flujos de Integración Continua (CI) y Entrega Continua (CD). - Uso de **Jenkins** como plataforma open source para la definición de pipelines. - Validación automática del código mediante ejecución de pruebas en entornos contenerizados antes del despliegue.	- Configurar un pipeline CI/CD en Jenkins que se ejecute ante eventos *push* en ramas principales del repositorio. - Automatizar la ejecución de pruebas del backend y la validación de la estructura del código. - Integrar etapas de construcción de imágenes Docker y despliegue al servidor en la nube tras validación exitosa. -Asegurar trazabilidad y control en cada integración al repositorio.

Desplegar la aplicación en un entorno de nube configurado.	- Uso de infraestructura en la nube (IaaS o PaaS) para la ejecución de aplicaciones backend. - Entornos Linux remotos como VPS o plataformas como Heroku, Railway, GCP o AWS. - Configuración de red, servicios, puertos y autenticación para acceso remoto.	- Seleccionar y aprovisionar infraestructura cloud basada en Amazon Web Services. - Configurar una instancia EC2 para para la ejecución de los servicios necesarios del proyecto, incluyendo automatización, despliegue y futuras extensiones operativas. - Utilizar RDS como servicio gestionado para la base de datos del sistema. -Configurar acceso SSH desde el repositorio GitHub para automatizar despliegues.

Tabla 1: Tabla de objetivos específicos. Fuente: Elaboración Propia

CAPÍTULO 2: MARCO CONCEPTUAL

El presente marco conceptual tiene como propósito fundamentar los elementos teóricos y técnicos esenciales para la implementación de una base funcional DevOps en el proyecto EverPlanApp, una aplicación orientada a la planificación inclusiva para estudiantes neurodivergentes. La construcción de esta base se sustenta en prácticas modernas de desarrollo colaborativo, integración continua, despliegue automatizado, y gestión de entornos virtualizados.

Para alcanzar estos objetivos, es necesario comprender conceptos clave como el control de versiones, los sistemas distribuidos como Git, la estructuración de flujos de trabajo con ramas, y la automatización de procesos mediante pipelines CI/CD. Asimismo, se abordan principios asociados al uso de contenedores, la documentación técnica reproducible, y las buenas prácticas en gestión de datos.

A continuación, se presenta el desarrollo conceptual de los componentes más relevantes en el contexto de DevOps y su aplicación en EverPlanApp.

2.1 Configurar control de versiones colaborativo con Git y GitHub.

2.1.1 Fundamentos del Control de Versiones

En todo proyecto de desarrollo de software, los archivos están en constante cambio: se agregan funciones, se corrigen errores, se hacen pruebas y mejoras. Sin una forma de registrar estos cambios, el trabajo puede volverse caótico.

El control de versiones es un componente esencial en la ingeniería de software moderna, especialmente en contextos de desarrollo colaborativo como el de EverPlanApp. Un sistema de control de versiones (Version Control System, VCS) permite registrar, organizar y administrar de manera sistemática los cambios realizados sobre archivos digitales (en particular, código fuente) a lo largo del tiempo. Esta tecnología proporciona una estructura que posibilita la comparación entre versiones, la recuperación de estados anteriores, y la coordinación eficiente entre múltiples colaboradores mediante la gestión de ramas y fusiones.

Entre los beneficios más relevantes de utilizar un sistema de control de versiones se encuentran:

- La prevención de sobrescritura de código o pérdida de información crítica
- La posibilidad de restaurar versiones anteriores si se detectan errores
- El mantenimiento de un historial completo de cambios con metadatos (autor, fecha, motivo)
- La coordinación eficiente del trabajo entre distintos integrantes del equipo

Estas características permiten construir un entorno de trabajo replicable, auditable y seguro, especialmente cuando se emplea en conjunto con flujos modernos de desarrollo como la integración y entrega continua (CI/CD).
Para EverPlanApp, el control de versiones representa una solución técnica fundamental para mantener la coherencia del sistema, asegurar la calidad del código, habilitar la trazabilidad de cambios, y permitir la evolución sostenida del software dentro de un equipo distribuido.

2.1.2 Sistemas de control de versiones: centralizado vs distribuido

En el desarrollo de software moderno, especialmente en contextos colaborativos, es fundamental contar con herramientas que gestionen los cambios del código fuente de forma ordenada, segura y eficiente. Para ello se utilizan los sistemas de control de versiones (VCS), que permiten mantener un historial completo del proyecto, identificar quién hizo qué cambios, y facilitar el trabajo simultáneo sin sobrescribir ni perder información. Además de prevenir errores, estos sistemas son clave en la colaboración distribuida, permitiendo incorporar nuevas funcionalidades o corregir fallos sin comprometer la estabilidad del sistema. En síntesis, el control de versiones actúa como una "máquina del tiempo" del software y es base del desarrollo ágil y la integración continua.

Existen principalmente dos enfoques para implementar control de versiones: los sistemas centralizados y los distribuidos.

a) Sistemas centralizados

Un sistema de control de versiones centralizado es una herramienta donde todo el historial del proyecto se guarda en un único servidor. Los desarrolladores deben conectarse a ese servidor para descargar o subir sus cambios. Ejemplos conocidos son Subversion (SVN) y CVS. Aunque facilita la administración, depende completamente del servidor, lo que limita el trabajo offline y representa un riesgo si el servidor falla.

b) Sistemas distribuidos

Un sistema de control de versiones distribuido, como Git, permite que cada desarrollador tenga una copia completa del repositorio, incluyendo archivos y todo el historial. Las operaciones como commits, creación de ramas o revisión de cambios se realizan localmente, sin necesidad de un servidor central. Luego, los cambios pueden sincronizarse con otros repositorios mediante comandos como *push* y *pull*.

En el caso de EverPlanApp, que es desarrollado por un equipo que trabaja en diferentes entornos locales, se decidió utilizar un sistema de control de versiones distribuido como Git. Esta herramienta permite que cada miembro del equipo trabaje de manera independiente, pruebe nuevas funcionalidades en ramas separadas y luego integre sus cambios a través de procesos automatizados de validación y despliegue. Esta dinámica es clave para implementar prácticas de DevOps y garantizar una evolución técnica constante y controlada del sistema.

2.1.3 Git como sistema de control distribuido

Git es actualmente el sistema de control de versiones distribuido más utilizado. Fue creado en 2005 por Linus Torvalds, autor del kernel de Linux, ante la necesidad de reemplazar a BitKeeper por una herramienta rápida, segura y descentralizada. Desde el inicio, Git se diseñó con tres principios clave: velocidad, integridad de datos y soporte para desarrollo distribuido. Estas bases lo diferenciaron de otros sistemas y lo consolidaron como estándar en proyectos de código abierto y entornos empresariales.

Características principales de Git

- Modelo distribuido completo: Cada desarrollador posee una copia completa del repositorio, incluyendo todo su historial. Esto permite trabajar sin conexión, hacer commits y gestionar ramas localmente.
- Rendimiento local optimizado: Operaciones como commit, branch, merge y diff se ejecutan localmente, lo que mejora notablemente el rendimiento frente a sistemas centralizados.
- Integridad con SHA-1: Git garantiza la integridad del historial mediante el uso de hashes SHA-1, que permiten detectar cualquier alteración en los datos.
- Ramificación flexible y ligera: Las ramas en Git son fáciles de crear y administrar. Este enfoque facilita el desarrollo paralelo, la experimentación y los flujos de trabajo ágiles.
- Fusión de cambios precisa: Permite integrar ramas mediante fusiones automáticas o manuales, lo cual es esencial para mantener la coherencia en procesos de integración continua.
- Historial trazable: Cada cambio queda registrado con autor, fecha y mensaje, lo que permite auditorías, revisiones y un seguimiento detallado del desarrollo.

Git en entornos DevOps

La integración de Git con plataformas como GitHub, GitLab o Bitbucket extiende su funcionalidad al permitir control de acceso por roles, revisiones mediante pull/merge requests, automatización de pruebas y despliegues (CI/CD), y gestión de incidencias. En este contexto, Git se convierte en el eje central del flujo de trabajo DevOps, unificando control de versiones, colaboración y automatización en una sola herramienta.

En EverPlanApp, se utiliza Git como herramienta principal para el control de versiones del proyecto, permitiendo gestionar de manera centralizada el código fuente y mantener la trazabilidad de los cambios realizados. En el contexto del subproyecto DevOps, el control de versiones se enfoca principalmente en el backend del sistema, mientras que el frontend móvil y la base de datos gestionada en la nube quedan fuera del proceso de automatización y despliegue. El uso de ramas específicas (como *feature/*, *hotfix/* y *release/*) permite organizar el trabajo colaborativo, facilitar la revisión de cambios y asegurar una integración controlada. Esta estructura se complementa con procesos de automatización mediante Jenkins, garantizando coherencia, trazabilidad y control a lo largo del ciclo de vida del software. Git se selecciona por su madurez, amplia adopción en la industria y alineación con las prácticas DevOps.

2.1.4 Comparación entre Git y otros sistemas

Durante el desarrollo de EverPlanApp, evaluamos distintos modelos de control de versiones para determinar cuál se ajustaba mejor a las necesidades técnicas y organizativas del proyecto. En este análisis consideramos tanto sistemas centralizados como Subversion (SVN), como distribuidos, como Mercurial (Hg) y Git.

Subversion es una herramienta centralizada que permite cierto control lineal del flujo de trabajo, lo cual puede ser útil en equipos pequeños o con estructura jerárquica. Sin embargo, su dependencia constante de un servidor central y su rigidez frente al trabajo paralelo limitan su aplicabilidad en contextos distribuidos como el nuestro.

Por otro lado, Mercurial también es un sistema distribuido, y se destaca por su simplicidad y una curva de aprendizaje algo más suave que la de Git. Sin embargo, al revisar documentación técnica y experiencias de uso en proyectos reales (*Chacon & Straub, 2014; Loeliger & McCullough, 2012*), observamos que la comunidad y el soporte en torno a Mercurial han disminuido en comparación con Git, lo cual representa una desventaja tanto a nivel técnico como formativo.

En base a esta comparación, se decidió utilizar Git como sistema de control de versiones para el proyecto EverPlanApp. Esta elección se fundamenta en los siguientes aspectos:

1. Soporte para trabajo distribuido y asincrónico, lo cual resulta clave en un equipo donde los integrantes trabajan desde distintos entornos y en horarios diversos.
2. Facilidad para ramificar y desarrollar funcionalidades en paralelo sin afectar la rama principal, permitiendo mantener un flujo de desarrollo ordenado y controlado.
3. Integración efectiva con herramientas de automatización CI/CD, como Jenkins, utilizadas para la validación y despliegue del backend del sistema.
4. Escalabilidad técnica y organizacional, considerando el posible crecimiento del proyecto, la incorporación de nuevas funcionalidades y su integración con herramientas del ecosistema DevOps.
5. Amplia comunidad y abundante documentación, lo que facilita la incorporación de nuevos colaboradores y la resolución autónoma de problemas técnicos.
6. Relevancia profesional, dado que el dominio de Git es una competencia ampliamente valorada en el ámbito de la ingeniería de software.

Criterios de comparación

Criterio	Git	Subversion (SVN)	Mercurial (Hg)
Modelo	Distribuido	Centralizado	Distribuido
Velocidad	Alta (operaciones locales)	Media (depende del servidor)	Alta (local)
Ramificación	Muy eficiente, ligera y flexible	Costosa y poco intuitiva	Eficiente, pero menos flexible
Historial local	Completo	Parcial (requiere conexión)	Completo
Soporte y comunidad	Muy amplia	En declive	Más reducido
Curva de aprendizaje	Moderada	Baja	Baja a moderada
Integración DevOps	Excelente	Limitada	Buena, pero menos adoptada

Tabla 2: Análisis comparativo de sistemas de control de versiones. Fuente: Elaboración Propia

En resumen, aunque exploramos distintas opciones, la elección de Git responde tanto a criterios técnicos como pedagógicos. Su adopción se alinea con las buenas prácticas de la industria, y nos permite abordar el desarrollo de EverPlanApp con herramientas modernas, eficientes y adecuadas al contexto colaborativo de nuestro equipo.

2.1.5 El concepto de ramas en Git

Una de las características más destacadas de Git es su modelo de ramificación. A diferencia de otros sistemas de control de versiones, Git permite crear ramas de manera rápida, ligera y eficiente, lo que facilita un desarrollo paralelo y bien organizado.

Una rama en Git actúa como un puntero que señala a un commit específico dentro del historial del repositorio. Cuando trabajas en una rama, los nuevos commits se registran como parte de esa línea de desarrollo, sin alterar el resto del proyecto. Esto te permite aislar cambios relacionados con nuevas funcionalidades, correcciones o pruebas, sin afectar la versión principal.

El sistema de ramificación en Git no solo es técnico, sino que también es conceptual: organiza el trabajo de manera modular, mantiene el código estable en una rama principal (como main o master) y te da la libertad de experimentar en ramas separadas. Luego, estas ramas pueden fusionarse entre sí mediante procesos de fusión (merge), ya sea de forma automática o resolviendo conflictos manualmente.

Este enfoque promueve la adopción de flujos de trabajo bien estructurados, como Git Flow o GitHub Flow, donde cada rama tiene un propósito claro dentro del ciclo de vida del software. La organización de las ramas se complementa con procesos de integración continua, lo que permite automatizar pruebas y validaciones del backend según el estado de cada rama, utilizando herramientas como Jenkins. Esto refuerza el control y la calidad del desarrollo.

Ejemplo básico

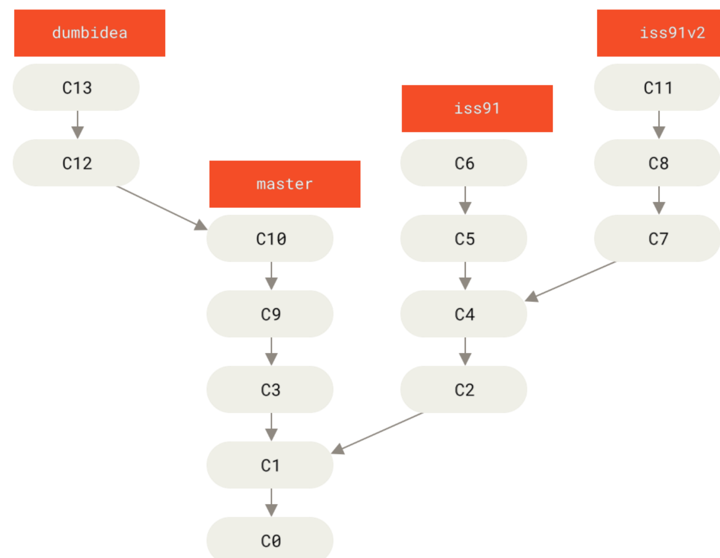

Figura 1:Ejemplo de flujo de ramas en Git. Fuente: Chacon y Straub (2014), Pro Git (2.ª ed.), capítulo “Branching Workflows”.

2.1.6 Modelos de flujo de trabajo con ramas

El modelo de ramificación de Git no solo permite dividir el trabajo en líneas paralelas, sino que también da origen a diferentes flujos de trabajo que organizan el desarrollo colaborativo del software. Estos flujos definen cómo y cuándo se crean ramas, cómo se integran los cambios, y qué reglas se siguen para mantener la estabilidad del proyecto. Lejos de ser simples convenciones, influyen directamente en la eficiencia, escalabilidad y seguridad del proceso de desarrollo (*Chacon & Straub, 2014; Atlassian, 2024*).

Entre los modelos más reconocidos se encuentran Git Flow, GitHub Flow y Trunk-Based Development. A continuación, se describen brevemente sus características principales.

a) Git Flow
Propuesto por Vincent Driessen en 2010, Git Flow es uno de los modelos más estructurados. Se basa en una jerarquía clara de ramas que separan el desarrollo activo, las versiones listas para prueba, y el código en producción. La estructura típica incluye ramas main, develop, feature/*, release/* y hotfix/*. Este modelo es ideal para proyectos con ciclos de desarrollo bien definidos y versiones planificadas.
Ventajas: organización clara, buen aislamiento de cambios, adecuado para entornos con control de calidad riguroso.
Desventajas: puede generar muchas ramas y ser complejo de mantener en equipos pequeños o con despliegue frecuente.

b) GitHub Flow
Es un modelo más simple, enfocado en equipos ágiles y desarrollo continuo. Aquí, cada nueva funcionalidad parte de una rama basada en main. Una vez completada, se abre un *pull request*, se ejecutan pruebas automáticas y, si todo está correcto, se hace el *merge*. Puede acompañarse de despliegue automático en producción.
Ventajas: simplicidad, integración con CI/CD, ideal para proyectos web y móviles.
Desventajas: menos control formal en entornos críticos o sin automatización sólida.

c) Trunk-Based Development
Este modelo propone trabajar todos en una única rama (main o trunk). Las funcionalidades incompletas se integran mediante *feature toggles* (banderas de características), que permiten ocultar cambios aún no listos para producción. Es común en equipos con despliegues múltiples por día.
Ventajas: máxima velocidad de integración, muy alineado con DevOps y CD.
Desventajas: requiere disciplina técnica elevada, buen sistema de pruebas y control estricto de los *toggles*.

Comparación general de modelos

Característica	Git Flow	GitHub Flow	Trunk-Based Dev
Complejidad	Alta	Media	Baja
Número de ramas	Muchas	Pocas	Una
Ideal para	Software versionado	Aplicaciones web/móviles	Equipos DevOps avanzados
Revisión de código	Sí	Sí	Opcional
Despliegue continuo (CD)	No	Sí	Sí
Automatización necesaria	Media	Alta	Muy alta
Requiere feature toggles	No	No	Sí

Tabla 3: Comparación general de modelos de flujo de trabajo con ramas en Git. Fuente: Elaboración propia

En el subproyecto DevOps de EverPlanApp, hemos decidido adoptar un modelo de trabajo híbrido que combina Git Flow y GitHub Flow, adaptándolo a las necesidades académicas y técnicas del proyecto. Este enfoque incluye una rama estable (main) y una rama de integración (develop), además de utilizar ramas feature/ para el desarrollo de funcionalidades de manera aislada.

La automatización de las validaciones y despliegues del backend se incorpora al flujo de trabajo a través de Jenkins, que se activa según el estado de las ramas principales. Este modelo nos permite mantener un orden claro, trazabilidad y una colaboración segura, sin complicar demasiado el proceso como lo haría el Git Flow completo, ni las exigencias del Trunk-Based Development. Se adapta perfectamente a un entorno académico donde hay diferentes niveles de experiencia en el uso de Git.

2.1.7 Monorepo y multirepo: enfoques para la organización del código en proyectos modulares

A medida que los sistemas de software incrementan su complejidad, resulta habitual dividir su desarrollo en componentes especializados, como frontend, backend y base de datos. Esta modularidad plantea la necesidad de definir una estrategia adecuada para la organización y el versionado del código fuente. En este contexto, destacan dos enfoques principales: monorepo y multirepo.

El modelo monorepo centraliza todos los componentes del sistema en un único repositorio, permitiendo gestionar código, configuraciones y documentación de forma unificada. Este enfoque no implica una arquitectura monolítica, sino una centralización a nivel de control de versiones, lo que puede facilitar la coordinación y la automatización en equipos pequeños o proyectos académicos.

Por su parte, el enfoque multirepo separa cada componente del sistema en repositorios independientes, otorgando mayor autonomía a cada módulo, pero aumentando la complejidad de coordinación, versionado y automatización entre ellos.

En EverPlanApp se adopta una estructura monorepo para el control de versiones del proyecto, integrando los distintos componentes del sistema dentro de un único repositorio. Esta decisión se fundamenta en la simplicidad de gestión, la trazabilidad centralizada de los cambios y la facilidad para mantener una documentación unificada. En el contexto del subproyecto DevOps, esta estructura permite enfocar los procesos de automatización y despliegue exclusivamente en el backend, utilizando Jenkins, mientras que el frontend y la base de datos gestionada en la nube quedan fuera del alcance del pipeline CI/CD.
Este enfoque resulta adecuado para un entorno universitario, ya que reduce la sobrecarga operativa y facilita el trabajo colaborativo sin comprometer las buenas prácticas de ingeniería de software.

2.2 Contenerizar el entorno completo usando Docker y Docker Compose.

2.2.1 Docker: tecnología base para entornos aislados y portables

Docker es una plataforma de virtualización ligera que permite desarrollar, empaquetar y ejecutar aplicaciones dentro de entornos aislados llamados contenedores. Estos contenedores incluyen todo lo necesario para ejecutar el software, como código fuente, dependencias, librerías del sistema, configuraciones y herramientas de ejecución. Esto garantiza que la aplicación se comporte de forma idéntica, sin importar el entorno donde se despliegue .

Docker proporciona un formato estandarizado para construir imágenes a través de un archivo llamado Dockerfile, y cuenta con un motor (Docker Engine) que permite ejecutar, detener, administrar y supervisar los contenedores. Gracias a su arquitectura modular y su ecosistema activo, Docker se ha convertido en una herramienta fundamental dentro de los flujos de trabajo DevOps modernos.

En el contexto de EverPlanApp, Docker se utiliza para la contenerización del **backend** del sistema, asegurando coherencia entre los entornos de desarrollo, pruebas y producción, y facilitando su integración con los procesos de automatización y despliegue definidos en el subproyecto DevOps. El frontend móvil y la base de datos gestionada en la nube quedan fuera del proceso de contenerización.

2.2.2 Docker como base para flujos DevOps

En el mundo de DevOps, Docker no solo se utiliza para contenerizar aplicaciones, sino que se ha convertido en un pilar fundamental para la integración, las pruebas y el despliegue automatizado. Su implementación promueve una arquitectura coherente y reproducible a lo largo de todo el ciclo de desarrollo.

Según su documentación oficial, Docker permite desvincular la aplicación de la infraestructura, lo que facilita una entrega rápida y segura en diferentes entornos, desde máquinas locales hasta servidores de producción.

Entre los principales beneficios de integrar Docker en flujos DevOps se encuentran:

- Portabilidad y consistencia: las imágenes generadas en el entorno de desarrollo pueden ejecutarse sin modificaciones en entornos de prueba o producción.
- Entornos reproducibles y configurables: la definición declarativa de servicios permite estandarizar la ejecución del sistema entre distintos miembros del equipo.

- Automatización de pruebas: es posible crear entornos temporales para la ejecución de pruebas automatizadas y eliminarlos una vez finalizadas.
- Eficiencia operativa: reduce las tareas manuales, fomenta la colaboración entre equipos y se integra perfectamente con los pipelines de CI/CD.
- Escalabilidad técnica: los servicios contenerizados pueden trasladarse fácilmente a la nube sin necesidad de reconfiguración manual.

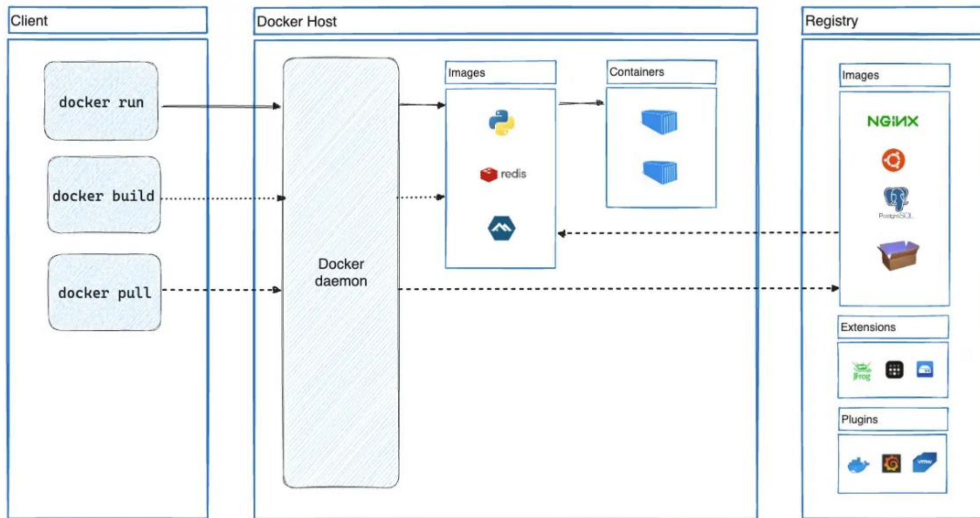

Figura 2: Diagrama general sobre la arquitectura y propósito de Docker. Fuente: Docker (2024). Recuperado de https://docs.docker.com/get-started/overview/

En el subproyecto DevOps de EverPlanApp, Docker se emplea para el empaquetado y ejecución del backend del sistema, permitiendo mantener coherencia entre los entornos de desarrollo, pruebas y producción, y facilitando su integración con los procesos de automatización definidos.

2.2.3 Contenedores en Docker

En el contexto de Docker, un contenedor es una unidad ejecutable que encapsula todo lo necesario para ejecutar una aplicación, incluyendo código, dependencias, configuraciones y librerías. Los contenedores se ejecutan como procesos aislados sobre el sistema operativo anfitrión, compartiendo su kernel, pero manteniendo independencia a nivel de red, sistema de archivos y ejecución.

A diferencia de las máquinas virtuales, los contenedores son más ligeros y eficientes, ya que no requieren un sistema operativo completo por cada instancia. Cada contenedor se genera a partir de una imagen, la cual actúa como una plantilla inmutable que define la estructura del entorno de ejecución.

Las imágenes se construyen mediante un archivo Dockerfile, escrito en un lenguaje declarativo que especifica paso a paso el proceso de construcción del contenedor. Instrucciones comunes incluyen FROM, RUN, COPY, EXPOSE y CMD. Este mecanismo permite versionar y compartir la definición exacta de un servicio, garantizando entornos consistentes entre desarrolladores y sistemas automatizados.

Una de las principales ventajas de los contenedores es su reproducibilidad, ya que una misma imagen puede ejecutarse de forma idéntica en distintos entornos, como desarrollo, pruebas o producción. Además, su carácter efímero facilita su creación y eliminación automática dentro de pipelines CI/CD, permitiendo pruebas controladas y despliegues confiables.

En el subproyecto DevOps de EverPlanApp, los contenedores se utilizan para la ejecución del backend del sistema, asegurando coherencia entre entornos y facilitando su integración con los procesos de automatización y despliegue. El frontend móvil y la base de datos gestionada en la nube quedan fuera del proceso de contenerización definido en este trabajo.

2.2.4 Docker Compose como herramienta de definición del entorno

Docker Compose es una herramienta que permite definir y configurar entornos de ejecución mediante archivos declarativos, facilitando la estandarización y gestión de aplicaciones contenerizadas. A través de un archivo docker-compose.yml, es posible describir de forma estructurada cómo deben ejecutarse uno o más servicios, incluyendo configuraciones como puertos, variables de entorno, dependencias y redes, sin necesidad de especificar comandos manuales de ejecución.

Según la documentación oficial de **Docker**, Docker Compose está orientado a simplificar la definición y ejecución de aplicaciones en entornos locales y de prueba, promoviendo la reproducibilidad y coherencia entre distintos contextos de ejecución. Su enfoque declarativo permite que los entornos definidos puedan versionarse junto al código fuente, facilitando el trabajo colaborativo y reduciendo errores derivados de configuraciones inconsistentes.

En el contexto de las prácticas DevOps, Docker Compose no solo se emplea como un mecanismo de orquestación de múltiples servicios, sino también como una herramienta para estandarizar la ejecución de aplicaciones contenerizadas, incluso cuando estas están compuestas por un único servicio. Este uso resulta especialmente relevante en entornos académicos y de desarrollo, donde se busca asegurar que todos los integrantes del equipo trabajen bajo las mismas condiciones técnicas.

De esta manera, Docker Compose se posiciona como un componente complementario dentro del ecosistema Docker, permitiendo definir de forma clara y reproducible el entorno de ejecución de una aplicación, y sirviendo como base para su integración con procesos de automatización y validación posteriores.

2.2.5 Comparación entre contenedores y otras tecnologías de virtualización

En el ámbito de la ingeniería de software, existen diversas tecnologías orientadas a la creación de entornos aislados para la ejecución de aplicaciones, entre las que destacan las máquinas virtuales y los contenedores. Cada una presenta características particulares en términos de consumo de recursos, nivel de aislamiento y facilidad de automatización.

Las máquinas virtuales permiten ejecutar sistemas operativos completos sobre un hipervisor, ofreciendo un alto nivel de aislamiento, pero a costa de un mayor consumo de recursos y tiempos de arranque más elevados. Este enfoque resulta adecuado para escenarios donde se requiere una separación estricta a nivel de sistema operativo, pero puede resultar poco eficiente para flujos DevOps que demandan rapidez y reproducibilidad.

Por otro lado, los contenedores, al compartir el kernel del sistema anfitrión, ofrecen una alternativa más liviana y eficiente. Esta característica los hace especialmente adecuados para entornos de integración continua y despliegue automatizado, donde la creación y destrucción frecuente de entornos es un requisito fundamental.

Desde una perspectiva DevOps, la utilización de contenedores mediante Docker facilita la estandarización de entornos, la automatización de pruebas y la portabilidad entre distintos contextos de ejecución, lo que justifica su adopción frente a otras soluciones de virtualización tradicionales.

Característica	Máquinas Virtuales	Contenedores (Docker)
Nivel de virtualización	Sistema operativo completo	A nivel de aplicación
Consumo de recursos	Alto	Bajo
Tiempo de arranque	Lento	Rápido
Reproducibilidad	Media	Alta
Automatización CI/CD	Limitada	Alta
Adecuación a DevOps	Media	Alta

Tabla 4: Comparación entre contenedores y máquinas virtuales. Fuente: Elaboración propia.

2.3 Implementar integración y entrega continua (CI/CD) con Jenkins

2.3.1 Integración Continua (CI)

La Integración Continua (Continuous Integration, CI) es una práctica fundamental dentro del enfoque DevOps que consiste en integrar de forma frecuente los cambios realizados en el código fuente dentro de un repositorio compartido. Cada integración activa un conjunto de validaciones automáticas, tales como la compilación del proyecto y la ejecución de pruebas, con el objetivo de detectar errores de manera temprana y reducir riesgos durante el desarrollo.

Según Martin Fowler, "la integración continua busca evitar los problemas asociados a integraciones tardías, promoviendo un flujo de trabajo en el que los desarrolladores integran sus cambios de forma regular y controlada, apoyándose en mecanismos automatizados que validan la estabilidad del sistema" (Fowler, 2006). Esta práctica mejora la calidad del software, facilita la detección de fallos y fortalece la colaboración entre los miembros del equipo.

Desde una perspectiva técnica, la CI se apoya en el uso de sistemas de control de versiones y servidores de automatización que monitorean los cambios en el repositorio y ejecutan procesos definidos previamente. Estas validaciones permiten asegurar que el sistema se mantenga en un estado funcional tras cada modificación, estableciendo una base confiable para etapas posteriores del flujo DevOps, como la entrega o el despliegue automatizado.

En el contexto de este trabajo, la Integración Continua se considera el pilar central del proceso de automatización, ya que permite validar de manera sistemática los cambios incorporados al backend del sistema antes de su liberación, reduciendo errores manuales y asegurando consistencia en el desarrollo.

2.3.2 Entrega Continua y Despliegue Automatizado (CD)

La Entrega Continua (Continuous Delivery, CD) es una práctica que extiende la Integración Continua y tiene como objetivo asegurar que el software se mantenga en un estado siempre listo para ser desplegado. Según Humble y Farley, "la Entrega Continua es la capacidad de poner cambios en producción de manera rápida, segura y sostenible" (Humble & Farley, 2010).

En este enfoque, una vez que el código ha sido integrado y validado mediante procesos de Integración Continua, el sistema avanza a etapas orientadas a la preparación del despliegue, tales como pruebas adicionales, empaquetado y validación del entorno. Estas etapas permiten reducir el riesgo asociado a la liberación de nuevas versiones y mantener un flujo de entrega controlado.

Figura 3: Flujo general de Integración Continua y Entrega Continua (CI/CD). Fuente: Adaptado de Humble y Farley (2010).

Tal como se observa en la Figura 2, la Entrega Continua se articula como una fase posterior a la Integración Continua dentro del ciclo DevOps, estableciendo una transición natural desde la construcción y prueba del software hacia su despliegue, operación y monitoreo. Este flujo refuerza la automatización progresiva del proceso, sin implicar necesariamente la liberación automática a producción.

Es importante distinguir la Entrega Continua del Despliegue Continuo (Continuous Deployment). Mientras la primera garantiza que el sistema esté preparado para ser desplegado en cualquier momento, el segundo implica que cada cambio validado se libera automáticamente a producción. En el contexto de este trabajo, la Entrega Continua se aborda como un marco conceptual que sienta las bases para un despliegue controlado y reproducible, acorde al alcance de una base funcional DevOps.

2.3.3 Servidores de automatización

Un servidor de automatización es un componente central en los flujos DevOps, encargado de ejecutar de forma automática las tareas definidas dentro de los procesos de Integración Continua y Entrega Continua. Estos servidores permiten orquestar etapas como la construcción del software, la ejecución de pruebas, el empaquetado y la preparación para el despliegue, a partir de eventos generados en el sistema de control de versiones.

En este contexto, Jenkins se presenta como una de las plataformas de automatización más utilizadas en la industria del software. Según su documentación oficial, “Jenkins es un servidor de automatización de código abierto que permite automatizar las partes del desarrollo de software relacionadas con la construcción, las pruebas y el despliegue” (Jenkins, 2024).

Desde una perspectiva conceptual, Jenkins actúa como un intermediario entre el repositorio de código y la infraestructura de ejecución, reaccionando ante cambios en el código fuente y ejecutando pipelines previamente definidos. Su arquitectura extensible y su integración con herramientas como sistemas de control de versiones y tecnologías de contenerización lo convierten en una base sólida para la implementación de flujos CI/CD.

En el marco de esta memoria, el uso de un servidor de automatización resulta fundamental para la implementación de una base funcional DevOps en EverPlanApp, ya que permite centralizar y estandarizar los procesos de validación del backend del sistema, asegurando trazabilidad, repetibilidad y control en cada integración realizada.

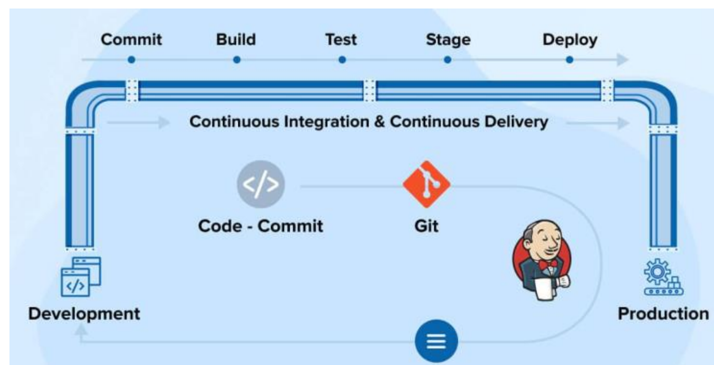

Figura 4: Rol del servidor Jenkins dentro de un flujo CI/CD. Fuente: Jenkins Documentation (2024).

2.3.3 Pipelines CI/CD

Un *pipeline* CI/CD representa un flujo de trabajo automatizado compuesto por una secuencia de etapas que permiten validar, construir y preparar una aplicación para su despliegue. Estas etapas suelen ejecutarse de forma ordenada y reproducible, garantizando que cada cambio en el código fuente sea procesado bajo las mismas condiciones técnicas.

Según la documentación oficial de Jenkins, "un pipeline es un conjunto de instrucciones automatizadas que describen cómo se construye, prueba y despliega el software" (Jenkins, 2024). Esta definición resalta el carácter declarativo de los pipelines, los cuales permiten versionar el flujo de automatización junto al código fuente, fortaleciendo la trazabilidad y el control de cambios.
Desde una perspectiva DevOps, los pipelines CI/CD facilitan la integración de prácticas como la Integración Continua y la Entrega Continua dentro de un proceso unificado. La ejecución automática de etapas como *build*, *test* y *deploy* reduce la intervención manual, minimiza errores humanos y acelera la retroalimentación hacia el equipo de desarrollo.
En el contexto de esta memoria, los pipelines CI/CD constituyen un elemento central para la implementación de una base funcional DevOps en EverPlanApp, ya que permiten estructurar y estandarizar los procesos de validación y preparación del backend del sistema, sentando las bases para un despliegue controlado, repetible y alineado con las buenas prácticas actuales de ingeniería de software.

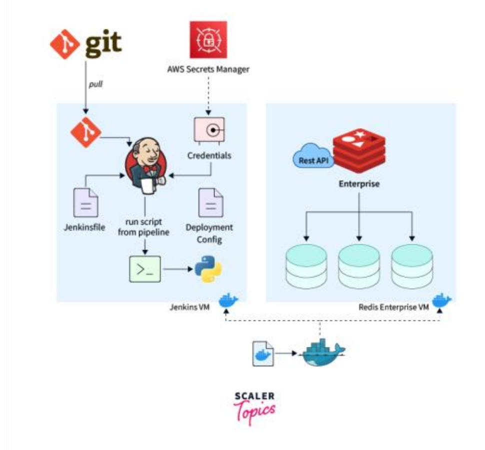

Figura 5: Estructura general de un pipeline CI/CD.
Fuente: Jenkins Documentation (2024).

2.4 Despliegue en la nube para entornos DevOps

2.4.1 Fundamentos del cómputo en la nube

El cómputo en la nube (cloud computing) es un modelo de prestación de servicios informáticos a través de internet que permite acceder, bajo demanda, a recursos como servidores, almacenamiento, bases de datos y aplicaciones, sin necesidad de adquirir ni mantener infraestructura física propia. Este paradigma ha transformado la forma en que se diseñan, despliegan y escalan las aplicaciones modernas.

Según la definición del *National Institute of Standards and Technology (NIST)*, la computación en la nube se caracteriza por cinco atributos esenciales:

1. Autoservicio bajo demanda: los usuarios pueden aprovisionar recursos automáticamente sin intervención humana.
2. Acceso amplio a la red: los servicios están disponibles mediante mecanismos estándares accesibles desde múltiples dispositivos.
3. Agrupamiento de recursos (resource pooling): los recursos del proveedor se comparten entre múltiples usuarios mediante una asignación dinámica.
4. Elasticidad rápida: es posible aumentar o reducir rápidamente los recursos en función de la demanda.
5. Servicio medido: el uso de los recursos se monitoriza y factura en función del consumo.

Modelos de servicio en la nube

Existen tres modelos principales de prestación de servicios en la nube:

- IaaS (Infraestructura como Servicio)
  Proporciona acceso a infraestructura virtualizada (máquinas, redes, almacenamiento). El usuario gestiona el sistema operativo y el software desplegado.
  Ejemplos: Amazon EC2, Google Compute Engine.

- PaaS (Plataforma como Servicio)
  Ofrece una plataforma completa para desarrollar, ejecutar y escalar aplicaciones sin preocuparse por la gestión de servidores o sistemas operativos.
  Ejemplos: Heroku, Railway.

● SaaS (Software como Servicio)
Permite acceder directamente a aplicaciones completas ofrecidas como servicio. El usuario solo interactúa con la aplicación final.
Ejemplos: Google Docs, Gmail, Trello.

Para fines de desarrollo y despliegue técnico, como en EverPlanApp, los modelos IaaS y PaaS resultan más relevantes, ya que permiten ejecutar aplicaciones personalizadas y controlar su comportamiento en diferentes entornos.

Modelos de implementación

La computación en la nube puede ofrecerse bajo diferentes esquemas de implementación:

● Nube pública
Infraestructura compartida y gestionada por un proveedor externo. Es el modelo más común y accesible, utilizado por plataformas como AWS, Azure o Google Cloud.

● Nube privada
Infraestructura dedicada a una sola organización, que puede estar gestionada internamente o por un proveedor especializado. Ofrece mayor control, aunque a mayor costo.

● Nube híbrida
Combinación de nube pública y privada, permitiendo migrar o distribuir servicios según los requisitos de seguridad, rendimiento o costo.

En entornos académicos o de prototipado funcional, como EverPlanApp, el uso de nube pública mediante VPS o plataformas gratuitas PaaS resulta eficiente, accesible y suficiente para simular despliegues reales.

2.4.2 Servicios comunes de despliegue en la nube

La adopción del cómputo en la nube ha impulsado el desarrollo de diversos modelos y plataformas orientadas al despliegue de aplicaciones web y APIs, los cuales difieren en su nivel de abstracción, flexibilidad, complejidad operativa y costos asociados. Desde una perspectiva DevOps, estas diferencias influyen directamente en la capacidad de automatización, control del entorno y escalabilidad de los sistemas.

En proyectos de software modernos, especialmente aquellos que utilizan contenedores, los modelos de Infraestructura como Servicio (IaaS) y Plataforma como Servicio (PaaS) se posicionan como las alternativas más representativas. El modelo IaaS ofrece un mayor control sobre la infraestructura subyacente, permitiendo configurar de forma detallada el sistema operativo, la red y los servicios, lo cual resulta adecuado para escenarios donde se busca aplicar prácticas DevOps más cercanas a entornos reales de producción.

Por otro lado, las soluciones PaaS abstraen gran parte de la gestión de la infraestructura, facilitando el despliegue rápido de aplicaciones y reduciendo la carga operativa del equipo, aunque con menores posibilidades de personalización. Estas plataformas son comúnmente utilizadas en contextos académicos, prototipos o proyectos de menor escala, donde la rapidez de puesta en marcha es prioritaria.

Desde el punto de vista conceptual, ambas aproximaciones representan modelos ampliamente utilizados en la industria del software. Analizar sus características y diferencias permite comprender los criterios que influyen en la elección de una plataforma de despliegue, así como justificar el uso de soluciones que favorezcan la automatización, reproducibilidad y control del entorno, aspectos fundamentales para el desarrollo de una base funcional DevOps.

Tabla comparativa

Criterio	IaaS (Infraestructura como Servicio)	PaaS (Plataforma como Servicio)	VPS
Nivel de control	Alto control sobre SO, red y servicios	Control limitado, infraestructura abstraída	Control total del sistema
Complejidad operativa	Media a alta	Baja	Alta
Automatización DevOps	Alta, integración flexible con CI/CD	Media, dependiente de la plataforma	Alta, pero requiere configuración manual
Escalabilidad	Alta y flexible	Alta, gestionada por la plataforma	Limitada, depende del proveedor
Adecuación a prácticas DevOps	Alta	Media	Alta
Uso típico	Entornos productivos y automatizados	Prototipos y despliegues rápidos	Escenarios de aprendizaje y control total

Tabla 5: Comparación entre modelos de despliegue en la nube: IaaS, PaaS y VPS. Fuente: Elaboración propia, basada en NIST (2011), Docker (2024) y Humble y Farley (2010).

2.4.3 Relación entre el despliegue en la nube y las prácticas DevOps

La adopción de entornos de despliegue en la nube ha tenido un impacto directo en la consolidación de las prácticas DevOps, al permitir la provisión rápida y flexible de infraestructura bajo demanda. A diferencia de los entornos tradicionales on-premise, la nube facilita la creación, modificación y eliminación de recursos de forma automatizada, lo que resulta fundamental para soportar procesos de Integración Continua y Entrega Continua.

Desde una perspectiva DevOps, los modelos de despliegue en la nube favorecen la estandarización de entornos, la reproducibilidad de configuraciones y la automatización de tareas operativas, reduciendo la dependencia de configuraciones manuales y minimizando errores humanos. Estas características permiten establecer flujos de trabajo más ágiles y controlados, alineados con las necesidades de desarrollo y despliegue continuo de software.

En el contexto de esta memoria, el uso de modelos de despliegue en la nube se justifica como un habilitador clave para la implementación de una base funcional DevOps en EverPlanApp. La disponibilidad de infraestructura flexible permite simular escenarios reales de despliegue de backend, integrar procesos automatizados y fortalecer la trazabilidad y el control del ciclo de vida del software, aspectos esenciales para un entorno académico y colaborativo.

CAPÍTULO 3: PROPUESTA DE SOLUCION

3.1 Evolución de la arquitectura DevOps de EverPlanApp

La arquitectura DevOps de EverPlanApp no fue definida de forma definitiva desde el inicio del proyecto, sino que evolucionó progresivamente a medida que se adquiría mayor comprensión sobre las herramientas, la infraestructura en la nube y las necesidades reales del sistema. Esta evolución estuvo fuertemente influenciada por el proceso de aprendizaje práctico asociado al rol DevOps asumido durante el desarrollo del subproyecto.

En una primera etapa, el backend y la base de datos se ejecutaban conjuntamente dentro de una misma instancia EC2, utilizando contenedores Docker administrados mediante Docker Compose. Esta decisión respondió principalmente a la necesidad de contar con una solución simple y rápida de implementar, considerando el conocimiento limitado que se tenía inicialmente sobre servicios administrados en la nube, como Amazon RDS. En este escenario inicial, Docker Compose permitía levantar tanto el backend como la base de datos de forma conjunta, facilitando la conexión entre ambos componentes y reduciendo la complejidad de configuración.

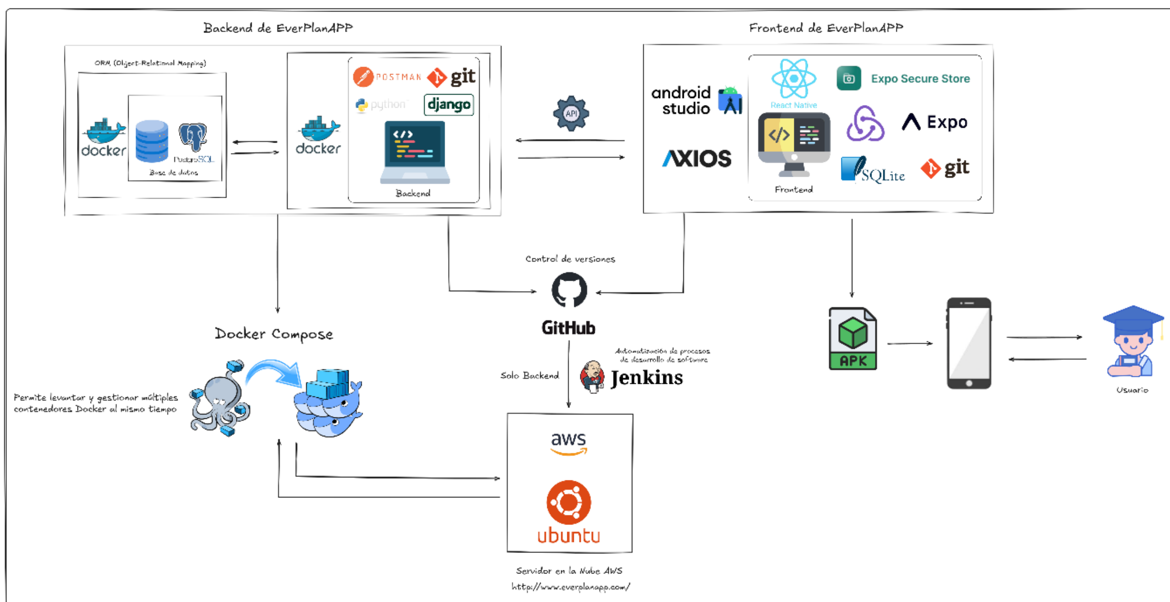

Figura 6: Arquitectura conceptual de la solución DevOps implementada para el backend de EverPlanApp. Fuente: Elaboración propia.

A medida que el proyecto avanzó, comenzaron a evidenciarse limitaciones asociadas a la arquitectura inicial, principalmente en términos de escalabilidad, mantenimiento y recuperación ante fallos, debido a la gestión conjunta del backend y la base de datos. Además, el proceso de despliegue era completamente manual, lo que aumentaba la probabilidad de errores y dificultaba la validación frecuente de cambios.

Durante esta etapa también se presentaron desafíos relacionados con la elección y configuración de la infraestructura en la nube. La adopción de Amazon Web Services respondió tanto a la disponibilidad de un plan académico como a la necesidad de contar con un entorno flexible. La configuración inicial de la instancia EC2 implicó un proceso de aprendizaje relevante, especialmente en aspectos de red, seguridad y conectividad externa.

Frente a estas dificultades, se decidió evolucionar la arquitectura separando la base de datos del backend y migrándola a un servicio administrado mediante Amazon RDS. Paralelamente, se incorporó Jenkins como herramienta de automatización, con el objetivo de avanzar hacia un despliegue más autónomo del backend. La implementación de Jenkins se encuentra en una etapa inicial, orientada a establecer una base funcional de integración continua, la cual se está mejorando progresivamente a medida que se adquiere mayor experiencia con la herramienta y con los flujos de automatización definidos para el proyecto.

Estos cambios requirieron ajustes en la configuración de Docker y Docker Compose, lo que permitió adquirir un mayor entendimiento sobre la contenerización, la automatización de despliegues y el ciclo de vida de las aplicaciones. La arquitectura final adoptada se encuentra más alineada con las prácticas DevOps, presentando una estructura más estable, mantenible y adecuada para la evolución del proyecto en la nube.

3.2 Decisiones de infraestructura y entorno

Las decisiones de infraestructura adoptadas para el subproyecto DevOps de EverPlanApp se definieron de forma progresiva, considerando tanto las necesidades técnicas del sistema como el proceso de aprendizaje asociado al rol DevOps asumido durante el desarrollo. El objetivo principal fue contar con un entorno flexible, configurable y adecuado para experimentar con prácticas de automatización y despliegue en la nube.

La elección de Amazon Web Services como plataforma de infraestructura respondió, en primer lugar, a la disponibilidad de un plan académico proporcionado por la universidad, lo que permitió acceder a servicios reales de nube sin incurrir en costos adicionales. Dentro de AWS, se optó por utilizar una instancia EC2 como entorno principal de ejecución, debido a la posibilidad de configurar manualmente recursos como memoria, almacenamiento y sistema operativoSO, lo que resultó especialmente útil en una etapa inicial de aprendizaje.

La configuración del entorno EC2 implicó enfrentar diversos desafíos técnicos, particularmente en aspectos relacionados con la apertura de puertos, reglas de seguridad, permisos de acceso y conectividad externa. Estas tareas permitieron adquirir un mayor entendimiento sobre el funcionamiento de servidores en la nube y la importancia de una configuración adecuada para garantizar la disponibilidad del sistema. Como parte de este

proceso, se incorporó el uso de un dominio para facilitar el acceso al backend sin requerir cambios constantes de dirección IP.

En relación con la persistencia de datos, se tomó la decisión de utilizar Amazon RDS como servicio administrado para la base de datos PostgreSQL. Esta elección permitió separar la gestión de datos del entorno de cómputo, reduciendo la carga operativa del servidor EC2 y mejorando la estabilidad del sistema. La migración hacia RDS implicó ajustes en la configuración del backend y en las variables de entorno definidas en Docker Compose, representando un aprendizaje relevante en la integración de servicios en la nube.

Por otra parte, Jenkins fue incorporado como herramienta de automatización dentro del mismo entorno EC2, con el objetivo de establecer una base funcional de integración continua. Su implementación se encuentra en una etapa inicial, orientada a automatizar tareas básicas del flujo de desarrollo, y se ha ido mejorando progresivamente a medida que se adquiere mayor experiencia con la herramienta y con los procesos DevOps definidos para el proyecto.

En conjunto, estas decisiones de infraestructura permitieron construir un entorno funcional y coherente con los objetivos del subproyecto, priorizando la comprensión de los componentes, la experimentación controlada y la evolución gradual de la solución DevOps implementada.

3.3 Contenerización del backend: dificultades y ajustes

La contenerización del backend de EverPlanApp representó uno de los desafíos técnicos más relevantes dentro del subproyecto DevOps, especialmente considerando que fue la primera experiencia práctica trabajando con Docker en un entorno de servidor en la nube. El objetivo inicial de este proceso fue lograr un entorno de ejecución consistente que permitiera reducir dependencias del sistema operativo y facilitar futuros despliegues automatizados.

En una primera etapa, la contenerización se abordó mediante la creación de un *Dockerfile* básico para el backend desarrollado en Django. Este archivo definía la imagen base, la instalación de dependencias y la ejecución de la aplicación. Si bien esta configuración permitió levantar el backend de forma funcional, se presentaron dificultades relacionadas con permisos, rutas de archivos y correcta instalación de librerías, lo que requirió múltiples iteraciones y pruebas hasta lograr una configuración estable.

Inicialmente, tanto el backend como la base de datos se ejecutaban dentro de contenedores administrados por Docker Compose. Esta aproximación facilitó la conexión entre ambos servicios durante las primeras fases del proyecto, pero a su vez generó complejidad al momento de adaptar el entorno a una arquitectura más escalable. La posterior migración

de la base de datos a Amazon RDS obligó a modificar la configuración de Docker Compose, ajustando variables de entorno y parámetros de conexión para que el backend pudiera comunicarse correctamente con un servicio externo.

Otro aspecto que presentó dificultades fue la definición adecuada del archivo *docker-compose.yml*, particularmente en lo relacionado con la gestión de redes, puertos y dependencias entre servicios. La resolución de estos problemas permitió adquirir una comprensión más profunda sobre la forma en que Docker Compose coordina múltiples componentes y sobre la importancia de una configuración clara y ordenada para evitar conflictos durante la ejecución.

A pesar de estas dificultades, el proceso de contenerización permitió establecer una base sólida para la ejecución del backend tanto en entornos locales como en el servidor EC2. Desde una perspectiva formativa, este proceso contribuyó significativamente al entendimiento del ciclo de vida de las aplicaciones contenerizadas y a la aplicación práctica de conceptos fundamentales de DevOps, como la reproducibilidad del entorno y la estandarización de despliegues.

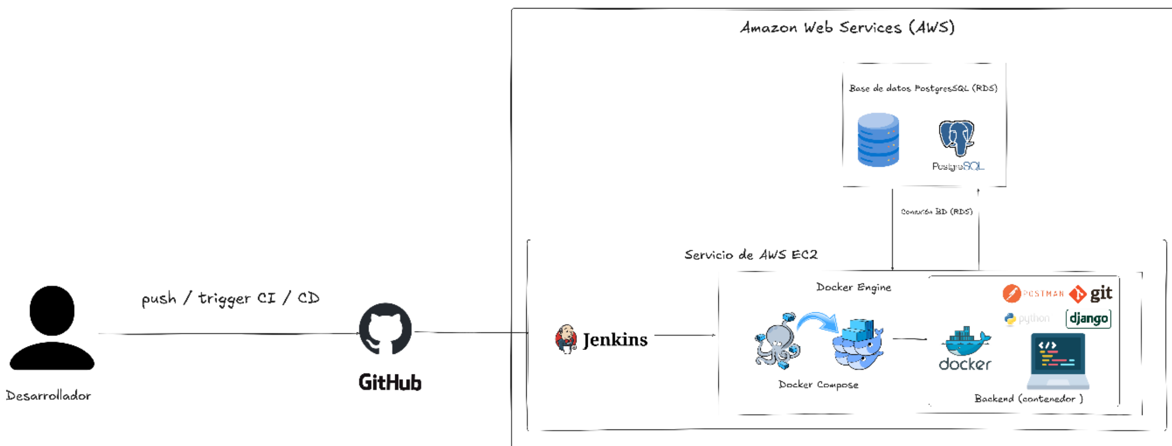

Figura 7: Arquitectura conceptual de la solución DevOps implementada para el backend de EverPlanApp. Fuente: Elaboración propia

3.4 Automatización del flujo CI/CD con Jenkins

Jenkins se encuentra instalado y ejecutándose en la instancia EC2, desde donde orquesta las tareas asociadas al ciclo de integración del backend. El flujo se activa a partir de eventos de actualización en el repositorio GitHub, mediante un mecanismo de *trigger* que permite iniciar el proceso ante nuevos *push* de código.

La implementación actual corresponde a una configuración inicial, orientada a automatizar las etapas básicas del proceso, tales como la obtención del código fuente, la construcción de la imagen Docker a partir del *Dockerfile* y la ejecución del backend mediante Docker Compose. Jenkins no ejecuta directamente la aplicación, sino que coordina la ejecución de comandos sobre el entorno Docker definido en el servidor.

Si bien el pipeline CI/CD no se encuentra completamente optimizado ni incorpora validaciones avanzadas, su implementación permite establecer un flujo reproducible y coherente para la actualización del backend. Este enfoque facilita la mejora progresiva del proceso, permitiendo iterar sobre la configuración del pipeline a medida que se adquiere mayor experiencia y se identifican nuevas necesidades del proyecto.

En el contexto del subproyecto DevOps, Jenkins actúa como un componente clave para la automatización del despliegue, sentando las bases para una integración continua más robusta y escalable en etapas futuras.

CI/CD Workflow – Backend EverPlanApp

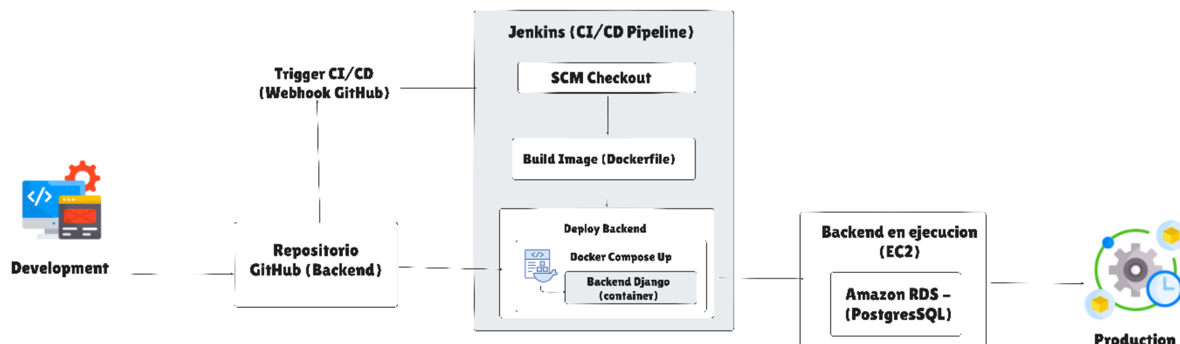

Figura 8: Flujo CI/CD del backend de EverPlanApp. Fuente: Elaboración propia

3.5 Despliegue en la nube: configuración, pruebas y problemas

El despliegue del backend de EverPlanApp se realizó sobre infraestructura en la nube, proceso que estuvo precedido por una etapa exploratoria en la que se evaluaron distintas alternativas de servicios cloud. En una fase inicial, se utilizó una plataforma de nube genérica con fines de autoaprendizaje, la cual permitió comprender conceptos básicos de virtualización, despliegue y acceso remoto. No obstante, esta solución presentaba limitaciones relevantes, principalmente en términos de recursos disponibles y flexibilidad de configuración, lo que restringía la posibilidad de escalar el entorno o replicar escenarios más cercanos a un contexto productivo.

Ante estas restricciones, y considerando la necesidad de profundizar en prácticas DevOps aplicadas a un proyecto real, se optó por migrar hacia Amazon Web Services (AWS). Esta decisión estuvo motivada tanto por la disponibilidad de un plan académico ofrecido por la universidad como por el acceso a una infraestructura más completa, ampliamente utilizada en entornos profesionales. Asimismo, el uso de AWS permitió complementar el aprendizaje autodidacta con contenidos abordados en asignaturas relacionadas con computación en la nube, fortaleciendo la comprensión técnica del entorno.

El despliegue final se realizó mediante una instancia EC2 con sistema operativo Ubuntu Server, seleccionada por su alto grado de configurabilidad y control sobre los recursos del sistema. En este entorno se configuraron aspectos esenciales como reglas de seguridad, apertura de puertos, acceso remoto y conexión con una base de datos PostgreSQL administrada a través de Amazon RDS, asegurando que el backend fuera el único componente con acceso directo a la capa de persistencia.

Durante esta etapa se llevaron a cabo pruebas tanto manuales como automatizadas. Inicialmente, el backend fue desplegado manualmente utilizando Docker y Docker Compose, con el objetivo de validar la correcta construcción de imágenes, el levantamiento de contenedores y la comunicación con la base de datos. Posteriormente, estas tareas fueron progresivamente integradas a un pipeline de Jenkins, actualmente en proceso de mejora, orientado a reducir la intervención manual y aumentar la consistencia del despliegue.

El proceso no estuvo exento de dificultades técnicas. Se presentaron problemas asociados a la configuración de red, permisos de acceso, variables de entorno y conectividad con el servicio RDS, así como ajustes necesarios en los archivos Dockerfile y docker-compose.yml. La resolución de estos inconvenientes implicó un trabajo iterativo de análisis, pruebas y corrección, contribuyendo significativamente al entendimiento del funcionamiento de los servicios cloud y de la contenerización en un entorno real.

En conjunto, esta experiencia permitió consolidar un entorno de despliegue funcional para EverPlanApp y evidenció una evolución progresiva en el manejo de infraestructura en la nube. Desde una perspectiva formativa, el tránsito desde plataformas limitadas de autoestudio hacia una infraestructura profesional como AWS refleja un proceso de aprendizaje gradual, alineado con los objetivos del subproyecto y con las prácticas fundamentales de DevOps.

3.6 Aprendizajes, limitaciones y proyección

El desarrollo del subproyecto DevOps para EverPlanApp representó una instancia de aprendizaje significativo, tanto a nivel técnico como metodológico. A lo largo del proceso de diseño e implementación de la infraestructura, fue posible adquirir una comprensión práctica sobre el ciclo de vida de una aplicación desplegada en la nube, integrando conceptos de control de versiones, contenerización, automatización y gestión de entornos.

Uno de los principales aprendizajes estuvo relacionado con la configuración y administración de infraestructura en la nube, particularmente en el uso de servicios de Amazon Web Services. La experiencia de trabajar con instancias EC2 y bases de datos administradas mediante Amazon RDS permitió comprender la importancia de la separación de responsabilidades, la correcta configuración de seguridad y la necesidad de contar con entornos reproducibles. Asimismo, la implementación progresiva de Docker y Docker Compose facilitó el entendimiento de la contenerización como una herramienta clave para estandarizar la ejecución del backend.

En cuanto a la automatización, la incorporación de Jenkins permitió establecer una base funcional de integración y despliegue continuo. Si bien el pipeline implementado se encuentra en una etapa inicial, su desarrollo contribuyó a comprender el rol de la automatización dentro de los flujos DevOps y la relevancia de reducir tareas manuales en procesos repetitivos. Esta experiencia evidenció la importancia de iterar y mejorar progresivamente los pipelines, en lugar de buscar soluciones completamente acabadas desde el inicio.

No obstante, el subproyecto presenta ciertas limitaciones propias de su alcance académico. Entre ellas se encuentran la ausencia de pruebas automatizadas dentro del pipeline, la falta de mecanismos avanzados de monitoreo y observabilidad, y la inexistencia de estrategias formales de escalabilidad o alta disponibilidad. Estas limitaciones no afectan el cumplimiento de los objetivos definidos, pero sí delimitan el nivel de madurez alcanzado por la solución implementada.

Como proyección, se identifica la posibilidad de fortalecer el subproyecto mediante la incorporación de etapas adicionales al pipeline CI/CD, tales como validaciones automáticas, controles de calidad del código y mecanismos de despliegue más robustos. Asimismo, se

podría avanzar hacia una mayor automatización de la infraestructura y la integración de herramientas de monitoreo, con el fin de mejorar la estabilidad y mantenibilidad del sistema. En este sentido, la base DevOps implementada sienta un punto de partida sólido para futuras mejoras y extensiones del proyecto.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

En el presente capítulo se aborda la validación de la solución propuesta, con el objetivo de comprobar que la implementación desarrollada cumple con los requerimientos definidos para el subproyecto y resulta adecuada para el entorno en el cual fue aplicada. La validación se centra en verificar el correcto funcionamiento de la base funcional DevOps implementada, considerando aspectos como la automatización del proceso de integración y despliegue continuo, la contenedorización del backend y el despliegue del sistema en un entorno de nube.

Dado que el proyecto corresponde a un prototipo académico, la validación se realiza mediante pruebas técnicas y operativas, apoyadas en la ejecución controlada de los procesos implementados, el análisis de resultados obtenidos y la observación directa del comportamiento del sistema. Este enfoque permite demostrar la viabilidad de la solución y su aporte al desarrollo colaborativo y a la evolución técnica de EverPlanApp, aun cuando no se disponga de métricas de uso en un entorno productivo real.

4.1 Enfoque de validación de la solución

La validación de la solución se planteó a partir de una estrategia práctica y técnica, coherente con el carácter académico del subproyecto y con el alcance de la base funcional DevOps implementada. El objetivo principal fue verificar que los procesos definidos operaran de forma correcta, reproducible y coherente con los objetivos planteados en la propuesta de solución.

La estrategia de validación se basó en los siguientes enfoques:

- Validación funcional: comprobación del correcto despliegue y ejecución del backend contenedorizado, asegurando la disponibilidad del servicio y la correcta conexión con la base de datos en la nube.
- Validación del proceso CI/CD: verificación del disparo automático del pipeline ante cambios en el repositorio, incluyendo la obtención del código, construcción de la imagen Docker y despliegue del backend.
- Validación del entorno: comprobación de que la solución puede ser levantada nuevamente bajo las mismas configuraciones, garantizando consistencia entre ejecuciones.
- Validación operativa: observación directa del comportamiento del sistema frente a cambios controlados, como actualizaciones del código backend.

Esta estrategia permitió evaluar la solución desde una perspectiva técnica y operativa, priorizando la viabilidad, estabilidad y automatización del proceso, más que métricas de rendimiento avanzadas propias de entornos productivos a gran escala.

4.2 Validación del flujo de Integración y Despliegue Continuo (CI/CD)

La validación del flujo de Integración y Despliegue Continuo se realizó mediante la ejecución controlada del pipeline configurado en Jenkins, con el objetivo de verificar el comportamiento del sistema ante modificaciones en el código fuente del backend y su posterior despliegue en el entorno productivo. Este proceso permitió evaluar la automatización, estabilidad y continuidad del servicio, priorizando la correcta integración de las herramientas DevOps seleccionadas.

Durante la validación, se efectuaron cambios controlados en el repositorio GitHub, los cuales fueron enviados mediante *push*, activando automáticamente el pipeline configurado. A partir de esta acción, Jenkins ejecutó de forma secuencial las distintas etapas definidas, sin intervención manual, asegurando la trazabilidad y repetibilidad del proceso.

La Figura 9 presenta la ejecución exitosa del pipeline CI/CD en Jenkins, donde se visualizan claramente las etapas de *checkout* del código, construcción de la imagen Docker, despliegue mediante Docker Compose y acciones posteriores de verificación.

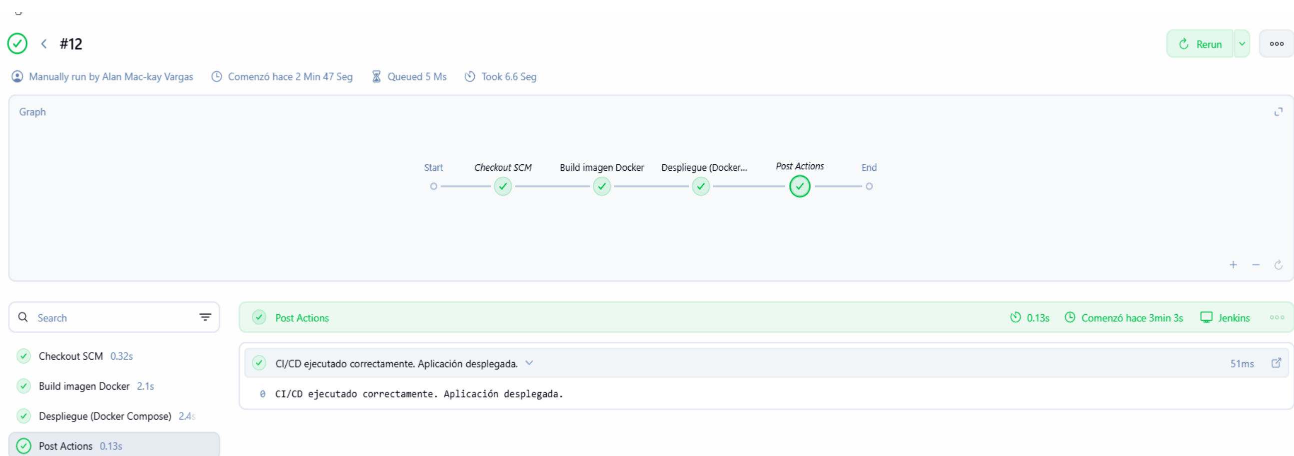

Figura 9: Ejecución exitosa del pipeline CI/CD en Jenkins para EverPlanApp. Fuente: Elaboración propia.

La imagen evidencia que todas las etapas del pipeline finalizaron correctamente, confirmando que Jenkins logró integrarse de manera adecuada con el repositorio GitHub y con el motor de contenedores Docker. Asimismo, se valida que el backend fue reconstruido y desplegado exitosamente en la instancia AWS EC2, demostrando que los cambios realizados en el código se reflejan automáticamente en el entorno de ejecución.

Complementariamente, la Tabla 6 resume los principales componentes que conforman el pipeline CI/CD implementado, junto con las herramientas utilizadas, su función dentro del flujo y el resultado obtenido tras la ejecución.

Componente	**Herramienta**	**Función**	**Resultado**
Control de versiones	Git / GitHub	Gestión del código fuente y control de cambios	Repositorio centralizado
CI/CD	Jenkins	Automatización de build y despliegue	Pipeline ejecutado con éxito
Contenedores	Docker	Construcción de imagen del backend	Imagen Docker generada
Orquestación local	Docker Compose	Despliegue automatizado del backend	Servicio levantado automáticamente
Infraestructura	AWS EC2	Ejecución del entorno DevOps	Entorno funcional en la nube
Variables de entorno	.env	Separación de configuración sensible	Seguridad y flexibilidad

Tabla 6: Pipeline CI/CD implementado para EverPlanApp con Jenkins y Docker Fuente: Elaboración propia.

La tabla permite identificar de forma estructurada el rol de cada tecnología dentro del flujo DevOps. Git y GitHub cumplen la función de control de versiones y gestión del código fuente; Jenkins actúa como el orquestador del proceso de automatización; Docker se encarga de la construcción de la imagen del backend; Docker Compose facilita el despliegue automatizado del servicio; y AWS EC2 proporciona la infraestructura necesaria para la ejecución del sistema. Adicionalmente, el uso de variables de entorno mediante archivos *.env* contribuye a la separación de la configuración sensible del código, mejorando la seguridad y flexibilidad de la solución.

En conjunto, la validación realizada confirma que el pipeline CI/CD implementado cumple satisfactoriamente con los objetivos definidos, permitiendo una integración continua y un despliegue automatizado confiable. Este enfoque reduce errores manuales, mejora la consistencia entre entornos y establece una base funcional DevOps sólida para la evolución futura de EverPlanApp.

4.3 Validación de la contenedorización del backend

La validación de la contenedorización del backend de EverPlanApp tuvo como objetivo comprobar que el servicio puede ejecutarse de forma consistente, reproducible y aislada del entorno subyacente, utilizando tecnologías de contenedores. Esta validación se enfoca exclusivamente en el backend del sistema, dado que la base de datos fue externalizada a un servicio administrado (Amazon RDS), tal como se definió en la propuesta de solución.

En el contexto de esta validación, se verificó que el backend se construye correctamente a partir de su imagen Docker, que el contenedor puede levantarse sin intervención manual y que el servicio queda disponible a través del puerto configurado. Asimismo, se comprobó que el proceso es compatible con la automatización definida en el pipeline CI/CD, permitiendo que Jenkins gestione el ciclo de vida del contenedor durante el despliegue.

El backend se ejecuta como un único servicio contenerizado, utilizando una imagen construida desde un Dockerfile específico del proyecto. Durante las pruebas realizadas, se validó que la imagen se genera sin errores y que el contenedor resultante se inicia correctamente bajo el nombre definido para el entorno de desarrollo, permitiendo identificar claramente su estado y funcionamiento.

La Figura 10 muestra la ejecución activa del contenedor del backend, evidenciando que el servicio se encuentra en estado operativo tras el despliegue automatizado.

```
STATUS          PORTS                                          NAMES
Up 16 minutes   0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp   everplan-backend-dev

CONTAINER ID   IMAGE                COMMAND                  CREATED
94eddf039a37   everplanapp-backend  "sh -c ' python mana…"   About an hour ago
```

Figura 10: Ejecución activa del contenedor Docker del backend de EverPlanApp. Fuente: Elaboración propia.

Como complemento a la evidencia visual, la Tabla 7 resume los principales parámetros validados durante el proceso de contenedorización del backend.

Servicio	Imagen Docker	Puerto expuesto	Estado
Backend EverPlanApp	everplan-backend-dev	8000	En ejecución

Tabla 7: Validación de la contenedorización del backend de EverPlanApp Fuente: Elaboración propia.

Los resultados obtenidos confirman que la contenedorización del backend cumple con los criterios definidos para la validación: el entorno es reproducible, el servicio se ejecuta de manera aislada y su despliegue puede ser automatizado mediante el pipeline CI/CD. Esta validación demuestra que Docker permite estandarizar la ejecución del backend, reduciendo dependencias del sistema anfitrión y facilitando su operación en un entorno de nube.

En consecuencia, la contenedorización del backend se considera válida y coherente con los objetivos del subproyecto DevOps, constituyendo un componente clave para la estabilidad, mantenibilidad y futura escalabilidad de EverPlanApp.

4.4 Validación del despliegue en la nube

El objetivo de este apartado es validar que la solución implementada puede ser desplegada y ejecutada correctamente en un entorno de infraestructura en la nube, utilizando servicios de Amazon Web Services (AWS). La validación se centra en comprobar la correcta configuración del entorno, la ejecución del backend dentro de la instancia EC2 y su funcionamiento operativo tras el despliegue automatizado.

4.4.1 Configuración del entorno en la nube

En esta etapa se validó la correcta configuración de la infraestructura base necesaria para ejecutar la solución. Para ello, se utilizó una instancia **Amazon EC2** bajo el esquema de capa gratuita, configurada con sistema operativo Linux y destinada exclusivamente a alojar el backend de la aplicación.

La arquitectura implementada considera la separación de responsabilidades entre los componentes del sistema, manteniendo el backend desplegado en la instancia EC2 y la base de datos alojada en un servicio administrado **Amazon RDS**, lo que permite mejorar la mantenibilidad, escalabilidad y seguridad de la solución.

Los principales elementos configurados fueron:

- Instancia EC2 tipo *t3.micro*.
- Sistema operativo Ubuntu.
- Acceso remoto mediante SSH.
- Exposición del puerto 8000 para el backend.
- Integración con Jenkins para el despliegue automatizado.
- Variables de entorno gestionadas mediante archivo .env.

4.4.2 Pruebas de acceso y funcionamiento

Una vez configurado el entorno, se procedió a validar la ejecución del backend desplegado dentro de la instancia EC2. Esta validación se realizó mediante la revisión de los registros de ejecución del contenedor Docker generado y levantado automáticamente por el pipeline CI/CD.

La evidencia se obtuvo utilizando el comando docker logs, el cual permitió verificar que:

- El contenedor del backend se encuentra en ejecución.
- La aplicación Django inicia correctamente.
- Las migraciones de la base de datos se aplican sin errores.

- El servicio queda disponible en el puerto 8000.

```
ubuntu@ip-10-0-7-49:~$ sudo docker logs everplan-backend-dev
Operations to perform:
  Apply all migrations: admin, auth, auth_app, contenttypes, horarioPersonal_app, notificacion_app, rutinas_app, sessions, tarea_app, token_blacklist
Running migrations:
  No migrations to apply.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 10, 2026 - 21:03:43
Django version 5.2, using settings 'EverPlanApp.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Figura 11: Registro de logs del contenedor Docker del backend desplegado en una instancia EC2, evidenciando el inicio correcto del servicio y su disponibilidad en el puerto 8000. Fuente: Elaboración propia.

4.4.3 Análisis de la validación en la nube

La ejecución exitosa del backend dentro de la instancia EC2 confirma que la solución puede ser desplegada correctamente en un entorno cloud real, manteniendo la separación entre infraestructura, aplicación y datos. La integración entre Jenkins, Docker y AWS permitió automatizar completamente el proceso de despliegue, reduciendo la intervención manual y los posibles errores asociados.

Si bien el despliegue se realizó con fines académicos y bajo restricciones propias de la capa gratuita, la validación demuestra que la arquitectura implementada es funcional y escalable, pudiendo adaptarse a entornos productivos mediante el uso de configuraciones adicionales de seguridad, monitoreo y balanceo de carga.

4.5 Análisis de resultados y limitaciones

La validación realizada permitió comprobar que la solución propuesta cumple con los objetivos técnicos definidos para esta memoria, demostrando la viabilidad de implementar una base funcional DevOps orientada a la automatización del ciclo de vida del backend de EverPlanApp.

En primer lugar, se validó exitosamente el flujo de Integración y Despliegue Continuo (CI/CD), comprobando que los cambios realizados en el repositorio GitHub activan automáticamente el pipeline configurado en Jenkins, sin intervención manual. Este pipeline ejecutó de forma correcta las etapas de obtención del código, construcción de la imagen Docker y despliegue del contenedor mediante Docker Compose en la instancia EC2, asegurando consistencia y reproducibilidad en cada ejecución.

Asimismo, la contenedorización del backend permitió estandarizar el entorno de ejecución, eliminando dependencias del sistema operativo anfitrión y facilitando el despliegue automatizado. El uso de variables de entorno mediante archivos .env permitió separar la configuración sensible del código fuente, alineándose con buenas prácticas DevOps y de seguridad básica.

Respecto al despliegue en la nube, se validó la correcta ejecución del backend en una instancia EC2 dentro de la capa gratuita de AWS, así como su comunicación con una base

de datos PostgreSQL alojada en Amazon RDS. La aplicación de migraciones y el arranque exitoso del servidor backend evidencian la correcta integración entre los componentes de la infraestructura.

No obstante, se identifican algunas limitaciones propias del alcance del proyecto y del entorno utilizado. En primer lugar, la solución fue desplegada utilizando recursos de la capa gratuita de AWS, lo que implica restricciones en capacidad de cómputo, memoria y almacenamiento. Estas limitaciones impiden evaluar aspectos como alta disponibilidad, tolerancia a fallos o escalabilidad automática en escenarios de carga real.

Además, el pipeline implementado se enfoca en una automatización básica orientada al despliegue, sin incorporar etapas avanzadas de validación como pruebas automatizadas, análisis de calidad de código o escaneo de seguridad, las cuales serían recomendables en un entorno productivo. Del mismo modo, el backend se ejecuta utilizando el servidor de desarrollo de Django, lo que resulta adecuado para fines académicos, pero no representa una configuración óptima para producción.

Finalmente, aunque la solución cumple con los objetivos definidos para esta memoria, quedan como líneas de trabajo futuras la incorporación de monitoreo, logging centralizado, gestión de secretos más robusta y la evolución del pipeline hacia un enfoque DevSecOps completo.

En conclusión, los resultados obtenidos validan la propuesta como una base funcional DevOps, demostrando mejoras concretas en automatización, orden arquitectónico y flujo de trabajo colaborativo, sentando las bases para una futura evolución técnica de EverPlanApp.

CAPÍTULO 5: CONCLUSIONES

El desarrollo de la presente memoria significó un proceso desafiante y formativo, tanto desde el punto de vista técnico como personal. La implementación de una base funcional DevOps para EverPlanApp no solo permitió cumplir con los objetivos planteados, sino que también evidenció la complejidad real que conlleva automatizar y gestionar el ciclo de vida del software en un entorno controlado y reproducible.

A lo largo del proyecto se transitó desde un escenario inicial sin experiencia previa en prácticas DevOps hacia la construcción de una solución funcional que integra control de versiones, automatización de procesos, contenerización y despliegue sobre infraestructura en la nube. Este recorrido implicó enfrentar errores, limitaciones técnicas y decisiones de diseño que fortalecieron la comprensión del funcionamiento real de estas herramientas y su impacto en el desarrollo colaborativo.

5.1 Cumplimiento de los objetivos y validez de la solución

Los objetivos definidos al inicio de la memoria fueron alcanzados de manera satisfactoria. Se logró implementar un pipeline CI/CD utilizando Jenkins, el cual permitió automatizar el flujo de integración y despliegue de EverPlanApp, reduciendo la intervención manual y aumentando la consistencia del proceso. La utilización de Docker y Docker Compose facilitó la estandarización del entorno de ejecución, permitiendo reproducir el despliegue de la aplicación de forma confiable.

La incorporación de servicios en la nube mediante AWS permitió separar responsabilidades entre componentes, ejecutar el pipeline en un entorno real y validar el funcionamiento de la solución en condiciones cercanas a producción. En conjunto, estos elementos demuestran que la solución propuesta es funcional, coherente con el alcance definido y técnicamente válida como base DevOps para el proyecto.

5.2 Aprendizajes obtenidos durante el desarrollo

Uno de los principales aportes de este trabajo fue el aprendizaje adquirido durante su desarrollo. Al no contar inicialmente con conocimientos profundos en cultura DevOps, cada etapa del proyecto representó una oportunidad para comprender nuevas formas de trabajo, arquitecturas y procesos. Desde la gestión del código hasta la automatización del despliegue, el proyecto permitió internalizar la importancia de la integración continua, la trazabilidad de los cambios y la reducción de errores manuales.

Este proceso de aprendizaje no fue lineal ni inmediato, sino que estuvo marcado por pruebas, fallos y ajustes constantes. Sin embargo, cada dificultad enfrentada contribuyó a consolidar una visión más clara sobre cómo se estructura un flujo DevOps y cómo este impacta positivamente en la calidad y mantenibilidad del software. La experiencia resultó enriquecedora y motivadora, reforzando el interés por profundizar en esta área.

5.3 Limitaciones encontradas

Durante el desarrollo de la memoria se identificaron diversas limitaciones, principalmente asociadas al uso de infraestructura en la nube. Los costos derivados de estos servicios pueden

resultar elevados a nivel personal, lo que condiciona la posibilidad de escalar la solución o mantener entornos más robustos de forma permanente. No obstante, esta restricción forma parte del aprendizaje obtenido, ya que permitió comprender la importancia de planificar adecuadamente los recursos y evaluar su impacto económico.

Asimismo, el pipeline implementado se enfocó en los procesos esenciales de integración y despliegue, dejando fuera etapas más avanzadas como pruebas automatizadas, análisis de seguridad o monitoreo continuo. Estas decisiones se tomaron en función del alcance académico del proyecto y del tiempo disponible para su implementación.

5.4 Proyecciones y trabajo futuro

Como proyección futura, se plantea la evolución de la solución hacia una automatización más completa y segura. La incorporación de pruebas automatizadas, herramientas de monitoreo y mecanismos de escalabilidad permitiría mejorar la confiabilidad del sistema y facilitar la detección temprana de fallos. De igual forma, el uso más avanzado de servicios administrados en AWS podría optimizar la disponibilidad y el rendimiento de la aplicación.

Estas mejoras permitirían transformar la base DevOps implementada en una plataforma más madura, preparada para enfrentar escenarios de mayor complejidad y carga, manteniendo siempre los principios de automatización, control y mejora continua.

REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS BIBLIOGRÁFICAS

Atlassian. (2024). *Git workflows*. Recuperado de https://www.atlassian.com/git/tutorials/comparing-workflows

Atlassian. (2024). *Monorepo vs. Multirepo: How to choose*. Recuperado de https://www.atlassian.com/git/tutorials/monorepos

Chacon, S., & Straub, B. (2014). *Pro Git* (2ª ed.). Apress. https://git-scm.com/book/en/v2

Docker. (2024). *Docker Compose overview*. Documentación oficial. Recuperado de https://docs.docker.com/compose/

Docker. (2024). *Dockerfile reference*. Documentación oficial. Recuperado de https://docs.docker.com/engine/reference/builder/

Docker. (2024). *What is a container?* Documentación oficial. Recuperado de https://docs.docker.com/get-started/

Docker. (2024). *What is Docker?* Documentación oficial. Recuperado de https://docs.docker.com/get-started/overview/

Fowler, M. (2006). *Continuous Integration*. Recuperado de https://martinfowler.com/articles/continuousIntegration.html

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.

Jenkins. (2024). *Pipeline*. Documentación oficial. Recuperado de https://www.jenkins.io/doc/book/pipeline/

Jenkins. (2024). *What is Jenkins?* Documentación oficial. Recuperado de https://www.jenkins.io/doc/

Amazon Web Services. (2024). *AWS documentation*. Recuperado de https://docs.aws.amazon.com

Amazon Web Services. (2024). *AWS EC2 User Guide*. Recuperado de https://docs.aws.amazon.com/ec2/

Loeliger, J., & McCullough, M. (2012). *Version Control with Git*. O'Reilly Media.

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing* (Special Publication 800-145). National Institute of Standards and Technology. Recuperado de https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf