UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Bill: Plataforma para la generación de proyectos con base tecnológica para DevOps

Informe de Proyecto de Grado presentado por

Santiago Cuadrado, Lucas Gonzalez, Maximiliano Gutiérrez

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Laura González
Gustavo Vázquez

Montevideo, 1 de julio de 2025

Agradecimientos

La presentación de esta tesis implica un gran paso hacia la finalización de nuestra etapa como estudiantes de Ingeniería en Computación, en la Facultad de Ingeniería de la UDELAR.

Este proyecto no solo representa el esfuerzo de quienes lo hemos desarrollado, durante el proceso hubo personas que han contribuido a que este momento sea posible desde diferentes espacios y de distintas maneras.

Queremos expresar nuestro más sincero agradecimiento a nuestras familias y amigos, quienes nos han acompañado durante la realización de este proyecto y a lo largo de todo el camino universitario. Su apoyo incondicional ha sido fundamental para llegar hasta aquí.

Estamos muy agradecidos con nuestros tutores Laura González y Gustavo Vázquez, y todos los docentes y estudiantes que asistieron a presentaciones para la validación.

Resumen

En el tiempo reciente, el término DevOps ha tomado gran popularidad en el sector TI. Este término conlleva una serie de prácticas y herramientas que se promueven principalmente con la finalidad de automatizar y agilizar el pasaje del trabajo implementado en Desarrollo (Dev) hacia Operaciones (Ops). Sin embargo, la implementación de técnicas DevOps no es una tarea sencilla, ya que se necesita conocimiento de ambas áreas para lograr el objetivo con éxito.

Como respuesta a esto surgen algunas soluciones como lo son las Plataformas como Servicio (PaaS) o la configuración automática de ciertas herramientas DevOps. Si bien esto es un avance importante, no cubren todos los aspectos para brindar una solución DevOps completa para un proyecto. Por otro lado, muchas de estas opciones son altamente dependientes de tecnologías específicas o proveedores de nube que no las hacen adecuadas para todos los casos. Este proyecto propone una plataforma que apunta a facilitar la adopción de herramientas DevOps mediante la generación automática de sus configuraciones.

Primero, se hizo un análisis de los requerimientos funcionales y no funcionales para la plataforma. Además, se realizó análisis del conocimiento existente sobre plataformas DevOps. Luego, en base a los requerimientos y al conocimiento existente, se propone una plataforma en la cual usuarios con diferentes grados de conocimiento pueden generar soluciones DevOps para sus proyectos. A continuación, se implementó la plataforma definida. Esto involucró el desarrollo de un portal web para que los diferentes usuarios puedan interactuar. Dicha plataforma es capaz de generar soluciones DevOps automáticamente y cuenta con un conjunto de proyectos y componentes nativos para ser reutilizados por los usuarios.

Finalmente, se realizaron verificaciones con casos de estudio y pruebas automatizadas, y se hicieron validaciones con presentaciones en las que concurrieron estudiantes y docentes. Además, se validó la extensibilidad de la plataforma mediante la incorporación de una nueva tecnología.

Palabras clave: DevOps, Plataforma para proyectos con base en DevOps, Automatización, Contenedores, Pipelines, Infraestructura como código, Monitoreo y Logging, Integración continua, Despliegues continuos

Índice general

Capítulo 1

Introducción

En este capítulo se presenta el contexto, motivación y objetivos de este proyecto.

1.1. Contexto

El término DevOps es un concepto que se ha popularizado en el sector TI en los últimos años, aunque aún carece de una definición completamente aceptada. En términos generales, DevOps refiere a un esfuerzo colaborativo y multidisciplinario dentro de una organización, cuyo objetivo es automatizar la entrega continua de nuevas versiones de software, apuntando a garantizar su corrección y confiabilidad [1].

El enfoque DevOps promueve una cultura de colaboración entre los equipos de desarrollo y operaciones, tradicionalmente aislados, haciendo uso de prácticas ágiles y una diversidad de herramientas que buscan automatizar el flujo de las tareas que surgen en Desarrollo (Dev) hacia Operaciones (Ops). En otras palabras, el objetivo principal es mejorar, acelerar y automatizar la transición de lo creado en el entorno de desarrollo hacia el entorno de producción.

La organización **DevOps Research and Assessment** (DORA[1]) es una organización que se dedica a estudiar cómo prácticas de DevOps e ingeniería de software impactan en el rendimiento de las organizaciones a través de estudios anuales; el más conocido es el State of DevOps Report [2].

DORA define cuatro métricas de performance que buscan capturar tanto la velocidad como la estabilidad del software, realizando las mediciones de forma transversal en las organizaciones por sobre mediciones locales [3].

1. **Change lead time**: El tiempo que tarda un cambio de código en desplegarse exitosamente en producción.

[1] https://dora.dev/

2. **Deployment frequency**: Frecuencia con la que se despliegan los cambios de la aplicación en producción.

3. **Change fail rate**: Porcentaje de despliegues que provocan fallos en producción, requiriendo correcciones urgentes o retrocesos.

4. **Failed deployment recovery time**: El tiempo que se tarda en recuperarse de un despliegue fallido.

El reporte más reciente hasta la fecha [4] realizado a partir de una encuesta anual dirigida a profesionales de la tecnología realiza un análisis estadístico empleando *cluster analysis* sobre las cuatro métricas de desempeño definidas. Los equipos del primer clúster reportan un *Change lead time* de menos de un día, realizando *despliegues on-demand*, menos del 5 % de fallos y recuperación en menos de una hora. Por otro lado, los del último clúster reportan *Change lead time* de entre 1 y 6 meses, despliegues mensuales o semestrales, 40 % de fallos y recuperaciones de entre una semana y un mes. Comparativamente, los equipos del primer clúster son 127 veces más rápidos en *Change lead time*, realizan 182 veces más despliegues, tienen 8 veces menos tasa de fallos y son 2293 veces más rápidos en recuperación de fallos en comparación a los equipos del último clúster.

En el libro *Accelerate* [3] se presentan los resultados de un estudio de cuatro años que analiza cómo prácticas de DevOps, ingeniería, gestión y cultura afectan el desempeño en la entrega de software. Estos resultados se basan en datos recolectados de organizaciones de distintos sectores. En la investigación presentada en el libro, se destacan 24 capacidades clave que impactan directamente en la performance de las métricas definidas previamente, algunas de índole organizacional y cultural, otras estrictamente relacionadas con la aplicación de técnicas de DevOps, entre las que se encuentran, **Despliegues automáticos**, **Integración continua (CI)**, **Control de versiones**, **Tests automatizados**, **Despliegues continuos (CD)** y **Monitoreo**.

1.2. Problema y justificación

Aunque los estudios realizados por DORA muestran cómo las prácticas de DevOps mejoran sustancialmente el desempeño del software, su adopción dentro de una organización puede presentar múltiples dificultades, tanto técnicas como culturales. En [5] se realiza una revisión sistemática de la literatura para identificar los principales problemas en la adopción de DevOps dentro de las organizaciones. El estudio muestra que uno de los principales desafíos que enfrentan las organizaciones para la adopción de estas prácticas es la falta de habilidades y conocimientos. Esto es debido a que se requiere de personal capacitado que domine tanto prácticas de desarrollo como de operaciones.

Considerando la complejidad tecnológica que implica la adopción de prácticas DevOps, así como su creciente relevancia en la industria del software, resulta

fundamental realizar un uso adecuado de las herramientas que las implementan. En respuesta a esta necesidad, se propone el desarrollo de una plataforma que reduzca barreras de entrada, permitiendo la adopción automatizada de dichas herramientas sin requerir un nivel avanzado de conocimientos técnicos.

Analizando las soluciones existentes, se identificó que las herramientas similares han sido diseñadas para entornos empresariales específicos, o focalizadas para un entorno tecnológico particular. También, algunas de las opciones más conocidas como Azure DevOps o *GitLab* CI/CD no tienen un enfoque unificado del ciclo de vida completo del software, o están enfocadas en un stack de tecnologías muy específicas, por ende es necesario utilizar otras herramientas para complementar.

Inspirados en soluciones como Fury [6] de Mercado Libre, que automatiza la infraestructura y los *pipelines* de CI/CD para sus desarrolladores internos, el proyecto busca trasladar esta idea adaptada para su uso general. Esto último se debe al hecho de que Fury presenta un enfoque principalmente comercial, ya que fue diseñado para ser utilizada dentro de una organización específica, según sus requerimientos específicos, junto con un nivel de complejidad alto luego de varios años de desarrollo.

1.3. Objetivos

El objetivo general de este proyecto es desarrollar una plataforma que proporcione soluciones tecnológicas integradas orientadas a la implementación de prácticas DevOps, adaptadas a necesidades específicas de usuarios. Esta plataforma busca automatizar la configuración y despliegue de un conjunto de herramientas que aborden las distintas etapas del ciclo de vida del software, desde la integración continua hasta el monitoreo. De esta forma, se apunta a facilitar la adopción de DevOps en los proyectos de los usuarios.

Para cumplir con esto, se plantean los siguientes objetivos específicos:

1. **Realizar un relevamiento de trabajos relacionados**: Este objetivo está destinado a la recopilación y análisis de antecedentes relevantes vinculados a la automatización de prácticas DevOps mediante una plataforma de automatización. Este relevamiento busca contextualizar el proyecto dentro del estado del arte, identificar enfoques existentes, soluciones propuestas y limitaciones comunes, aportando una base conceptual para el desarrollo de la plataforma.

2. **Identificar y especificar requerimientos funcionales y no funcionales de la plataforma**: Abarca la definición de capacidades esenciales de la plataforma, como la generación automática de soluciones, capacidad de incorporar nuevas tecnologías y la personalización de configuraciones.

Asimismo, se consideran propiedades no funcionales como la extensibilidad.

3. **Identificar y especificar requerimientos de la solución generada**: Se busca establecer los lineamientos que deben cumplir las soluciones automatizadas ofrecidas por la plataforma, considerando las necesidades particulares de los usuarios finales. Es necesario tener en cuenta los diferentes perfiles de usuarios y escenarios de uso.

4. **Diseñar la solución generada**: Este objetivo se centra en el diseño de soluciones tecnológicas entregadas al usuario, que integren herramientas y prácticas propias del enfoque DevOps, con el fin de responder a necesidades concretas de los usuarios. El diseño debe considerar los puntos más importantes dentro de las prácticas de DevOps e integrarlos en la solución, aspirando a entregar una solución lo más completa posible.

5. **Diseñar una plataforma extensible que permita la generación de las soluciones**: Se busca que la arquitectura de la plataforma cumpla con los requerimientos relevados, con un diseño modular que facilite la incorporación de nuevas herramientas y tecnologías, promoviendo su evolución. La extensibilidad se establece como una propiedad central, permitiendo que la plataforma pueda mantenerse vigente frente a los cambios tecnológicos y a la aparición de nuevas tecnologías.

6. **Implementar una plataforma que genere soluciones orientadas a las prácticas DevOps**: Este objetivo refiere al desarrollo de un sistema funcional que materialice el diseño propuesto, integrando las herramientas necesarias y automatizando su configuración según los requerimientos definidos.

7. **Incluir componentes nativos, realizar pruebas y validar plataforma**: Este objetivo contempla la incorporación de soluciones DevOps en la plataforma, incorporando una base de tecnologías diversas, abarcando distintos escenarios de uso que permitan demostrar su funcionamiento. Finalmente, se verifica la solución mediante pruebas automáticas y se busca llevar a cabo un proceso de validación orientado a comprobar que la plataforma satisface los requerimientos establecidos, para evaluar su utilidad y pertinencia en contextos reales de adopción.

1.4. Aportes del proyecto

Los principales aportes del proyecto son los siguientes:

1. **Relevamiento de trabajos relacionados**: Se presenta un análisis sistemático de trabajo relacionado vinculado a plataformas orientadas a la automatización de prácticas DevOps. Este relevamiento recopila enfoques

existentes, identifica sus limitaciones y caracteriza los elementos clave considerados en soluciones previas, proporcionando una base documental para el diseño de la plataforma propuesta.

2. **Diseño de la solución entregada al usuario**: Se define un modelo de solución que integra herramientas y prácticas DevOps, alineado con los requerimientos tecnológicos particulares de los usuarios.

3. **Plataforma para la generación de proyectos con base tecnológica para DevOps**: Se desarrolla una plataforma capaz de generar soluciones tecnológicas orientadas a la implementación de prácticas DevOps. La plataforma permite automatizar la creación de estas soluciones, incorporando herramientas para integración continua, despliegues automáticos, gestión de infraestructura, contenedores, monitoreo y logging. Admite la selección y combinación de componentes según las necesidades específicas del usuario, entregando una solución personalizada.

4. **Automatizaciones para la puesta en marcha de la plataforma**: Se realizan las automatizaciones necesarias para poner en funcionamiento en la nube la plataforma desarrollada, buscando que su despliegue sea sencillo y fácilmente replicable.

5. **Componentes nativos de la plataforma**: Se proporciona un conjunto de componentes predefinidos y configurados, listos para su utilización. Estos incluyen herramientas de desarrollo populares, tales como React, Angular, Java EE y Postgres.

1.5. Organización del documento

La estructura del resto del documento es la siguiente:

En el Capítulo 2 se describen los principales conceptos teóricos utilizados en distintas partes del documento.

Luego, en el Capítulo 3 se realiza el análisis de requerimientos de la plataforma, tanto funcionales como no funcionales. En esta sección también se presentan los resultados de la revisión de antecedentes.

El Capítulo 4 realiza una descripción general de la solución planteada, desde un punto de vista conceptual. Se describe desde la «solución DevOps» que recibirá el usuario, hasta la arquitectura de la plataforma.

En el Capítulo 5 se describe la implementación de la solución, las tecnologías específicas utilizadas en la plataforma, junto con un portal web desde el cual los usuarios interactúan con el resto de la plataforma.

A continuación, en el Capítulo 7 y el Capítulo 6, presentamos las instancias de validación y verificación que se realizaron para evaluar en qué medida la solución cumple con los objetivos definidos y funciona de acuerdo a lo esperado.

Por último, en el Capítulo 8 se presentan las conclusiones del trabajo realizado, limitaciones, lecciones aprendidas y posible trabajo a futuro.

Capítulo 2

Marco Teórico

En este Capítulo se brinda un marco teórico con definiciones necesarias para la comprensión del documento.

2.1. Automatización y Entrega Continua

A continuación, se presentan conceptos relacionados con la automatización y prácticas de entrega continua.

2.1.1. Integración Continua

La Integración Continua es una práctica del desarrollo de software que consiste en que los desarrolladores integren frecuentemente cambios al código de un repositorio, normalmente varias veces por día. En estas integraciones se hacen verificaciones automáticas con compilaciones y pruebas automatizadas. Esto permite que se puedan detectar errores rápidamente [7].

2.1.2. Despliegue Continuo

El Despliegue Continuo, o Entrega Continua, es una práctica estrechamente conectada con la Integración Continua. Esta se basa en tener el código siempre listo para hacer despliegues automáticos a un entorno de producción o *staging* (entorno de prueba) del código una vez finalizadas las pruebas automáticas con éxito [7].

2.1.3. Pipelines CI/CD

Los *Pipelines* CI/CD unen las prácticas de Integración Continua y Despliegue Continuo. Los mismos son flujos automatizados que se encargan de compilar, probar y desplegar una aplicación. Estos *pipelines* pueden incluir pasos como validación de código, ejecución de tests, empaquetado y despliegue de servidores.[7].

2.1.4. GitOps

GitOps es utilizado para automatizar el proceso de aprovisionamiento de infraestructura. Para esto, se utiliza la infraestructura como código (IaC) (ver Sección 2.2.1) junto a prácticas recomendadas de DevOps como lo son el control de versiones, la colaboración y *pipelines CI/CD*, aplicadas a la creación y gestión de la infraestructura. El concepto principal de GitOps es utilizar un sistema de control de versiones (ejemplo: Git) como fuente única de la verdad sobre el estado deseado del sistema. Los cambios en la infraestructura se realizan mediante la modificación de los archivos versionados [8].

2.2. Infraestructura y Orquestación

En esta sección se abordan conceptos clave sobre la gestión de infraestructura y su automatización.

2.2.1. Infraestructura como código (IaC)

La infraestructura como código consiste en configurar y aprovisionar la infraestructura de forma automática. Toma las prácticas adoptadas por el desarrollo de software pero aplicadas a la automatización de la infraestructura. El manejo de la infraestructura como código ha sido adoptado por grandes empresas como GitHub, Mozilla, Facebook, Google y Netflix, y su popularidad ha ido creciendo a lo largo de los últimos años [9].

Las herramientas de IaC permiten manejo de la infraestructura como si fuesen datos de software definiéndose generalmente en archivos de configuración. Pudiendo aplicar herramientas de control de versiones como Git, tests automáticos, integración continua, etc.

Estas prácticas permiten que los sistemas de infraestructura sean recursos que puedan ser creados, destruidos, modificados y replicados fácilmente para, por ejemplo, crear un entorno de *staging* idéntico al entorno de producción en cuestión de minutos. También ayudan a regresar rápidamente a versiones anteriores de la infraestructura en caso de errores, permitiendo que sea más fácil la adaptación a los cambios de diseño y a las necesidades propias de la evolución del software [10].

2.2.2. Contenedores

El objetivo principal de los contenedores es hacer las aplicaciones portables y autocontenidas. Son una encapsulación de una aplicación junto con todas sus dependencias. Además de la ejecución en el *cloud* los contenedores también tienen ventajas fuera del *cloud*, permitiendo a los usuarios y desarrolladores descargar

y ejecutar aplicaciones complejas evitando los problemas relacionados a la configuración en un sistema particular. La portabilidad de los contenedores tiene la capacidad de eliminar gran cantidad de problemas relacionados al entorno de ejecución [11].

2.3. Computación en la Nube

Esta sección presenta los conceptos fundamentales de computación en la nube.

2.3.1. Cloud Computing

Cloud Computing es un modelo para habilitar el acceso ubicuo, conveniente y bajo demanda a recursos compartidos de computación configurables, que pueden ser rápidamente aprovisionados y liberados con un bajo esfuerzo de gestión o interacción con el proveedor de servicios. En el mercado actual, existen distintos proveedores de *cloud* que ofrecen estos servicios, algunos de los más conocidos son AWS[1], Azure[2] y Google Cloud[3].

2.3.2. Multicloud

El término multicloud refiere al uso de más de un proveedor de *cloud* en simultáneo para un proyecto o conjunto de proyectos. Esto permite flexibilidad (no se genera una dependencia a un único proveedor), posibilidad de optimización de rendimiento y costos, y usar el mejor *cloud* para cada tarea [12].

2.3.3. Modelos de servicio de la nube

Dentro de los modelos de cloud computing se pueden encontrar Software como Servicio (SaaS), Plataformas como Servicio (PaaS) e Infraestructura como Servicio (IaaS).

En el modelo **SaaS** el consumidor utiliza aplicaciones del proveedor que se ejecutan sobre una infraestructura en la nube. Estas aplicaciones son accesibles desde diversos dispositivos cliente mediante interfaces ligeras como navegadores web. El consumidor no gestiona ni controla la infraestructura subyacente, incluyendo red, servidores, sistemas operativos o almacenamiento, y tiene control limitado sobre configuraciones específicas de la aplicación [13].

En modelo **PaaS** se permite al consumidor desplegar en la infraestructura en la nube aplicaciones propias, creadas con herramientas y lenguajes de programación compatibles con el proveedor. El consumidor no gestiona la infraestructura

[1] https://aws.com/
[2] https://azure.microsoft.com/
[3] https://cloud.google.com/

subyacente, pero mantiene control sobre las aplicaciones desplegadas y en ciertos casos, sobre la configuración del entorno de alojamiento de las mismas [13].

En el modelo **IaaS** el consumidor dispone de capacidad para aprovisionar procesamiento, almacenamiento, redes y otros recursos fundamentales de cómputo. Puede desplegar y ejecutar software arbitrario, incluidos sistemas operativos y aplicaciones. Aunque no gestiona la infraestructura subyacente, sí tiene control sobre sistemas operativos, almacenamiento, aplicaciones y, en ocasiones, componentes de red específicos como firewalls [13].

2.4. Control de Versiones

Git es un sistema de control de versiones distribuido que permite a múltiples desarrolladores trabajar de forma simultánea y coordinada en un mismo proyecto. A diferencia de los sistemas de control centralizado, Git almacena una copia completa del historial del proyecto en cada repositorio local, lo que proporciona mayor resiliencia y autonomía a los usuarios. Entre sus características más destacadas se encuentran su rapidez, flexibilidad para manejar flujos de trabajo no lineales (como ramas y fusiones), y su capacidad para rastrear y revertir cambios. Git es además una herramienta de software libre y de código abierto [14].

2.5. Solución DevOps

En esta sección se presentan herramientas que dan soporte a DevOps y se describe qué se entiende por solución DevOps.

2.5.1. Herramientas de soporte para DevOps

Para poder categorizar las diferentes herramientas de DevOps se utilizó un trabajo desarrollado en el marco de un Módulo de Taller de la Facultad de Ingeniería[4]. En dicho trabajo se presenta una clasificación consolidada de los tipos de herramientas empleadas en entornos DevOps, con una evaluación específica de Kubernetes[5] y de Grafana[6] utilizando el método T-Check[7]. Basándose en la clasificación consolidada, existen actualmente distintas herramientas que dan soporte a DevOps, que pueden categorizarse en 6: Intercambio de Conocimiento,

[4] https://www.fing.edu.uy/es/paginas/modulos-de-taller-y-modulos-de-extension-informacion-y-reglamento
[5] https://kubernetes.io/
[6] https://grafana.com/
[7] https://insights.sei.cmu.edu/documents/2508/2010_015_001_512005.pdf

Gestión de Código Fuente, Build Process, Integración Continua, Automatización de Despliegue y Monitoreo & Logging.

Las herramientas de Intercambio de Conocimiento facilitan la colaboración, documentación y transferencia de información entre miembros de un equipo. Ejemplos: Trello[8] y Notion[9].

La Gestión de Código Fuente incluye herramientas que permiten controlar las versiones del código de manera organizada y sincronizada entre un equipo de desarrolladores. Ejemplos: GitHub[10] y GitLab[11].

Las herramientas de Build Process automatizan el proceso de compilación del código fuente. Su función es asegurar que el software pueda ser construido correctamente junto a sus dependencias y retornar un formato ejecutable del programa. Ejemplos: Maven[12], Webpack[13] y Docker[14].

Las herramientas de Integración Continua detectan errores rápidamente integrando y probando de forma automática cada cambio realizado en el código. Esto permite identificar conflictos y fallos de pruebas antes de desplegar la nueva versión. Ejemplos: GitLab CI/CD[15] y Jenkins[16].

La Automatización de Despliegue permite implementar cambios en los entornos de forma automática con el fin de reducir el error humano y acelerar la entrega. Ejemplos: Terraform[17], Ansible[18], Docker y Kubernetes.

Finalmente, las herramientas de Monitoreo & Logging permiten observar el comportamiento del sistema durante su ejecución para detectar fallos y analizar métricas. Ejemplos: Grafana y Datadog[19].

En la Tabla 2.1, se presentan las 6 categorías con sus respectivos ejemplos.

2.5.2. Solución DevOps

En el marco de este trabajo, una solución DevOps refiere a un entorno listo para desarrollar y desplegar software, compuesto por repositorios con el código base del proyecto, pipelines de integración y despliegue configurados, y la

[8] https://trello.com/
[9] https://notion.so/
[10] https://github.com/
[11] https://gitlab.com/
[12] https://maven.apache.org/
[13] https://webpack.js.org/
[14] https://www.docker.com/
[15] https://docs.gitlab.com/ci/
[16] https://jenkins.io/
[17] https://developer.hashicorp.com/terraform/
[18] https://ansible.com/
[19] https://datadoghq.com/

Tabla 2.1: Tabla de Categorías

Categorías	Ejemplos
Intercambio de Conocimiento	Trello Notion
Gestión de Código Fuente	GitLab GitHub
Build Process	Maven Webpack Docker
Integración Continua	GitLab CI/CD Jenkins
Automatización de Despliegue	Docker Kubernetes Terraform Ansible
Monitoreo & Logging	Grafana DataDog

infraestructura definida como código e integrada a proveedores de *cloud*. Esta solución, adaptada a las decisiones tecnológicas del usuario, permite aplicar prácticas DevOps desde las etapas iniciales del proyecto, cubriendo la mayoría de las categorías previamente mencionadas.

Capítulo 3

Análisis

3.1. Conocimiento Existente

En esta sección se realiza un análisis del conocimiento existente, con la idea es tener una visión general en lo que respecta a herramientas para automatización de soluciones DevOps.

3.1.1. Trabajo Relacionado

En esta sección se realiza la revisión de antecedentes. Los detalles de la revisión se presentan en el Apéndice A.1.

Se realizó una búsqueda en el portal Scopus[1] que dio como resultado 8 documentos. Luego del análisis, se concluye que los siguientes 4 artículos son los que abordan temáticas similares al proyecto:

- **Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects** [15]: Este artículo es altamente relevante para nuestro proyecto, dado que el mismo trata de la automatización de *pipelines* CI/CD, un tema abordado en nuestra solución.
- **Automating Tiny ML Intelligent Sensors DevOps Using Microsoft Azure** [16]: Es otro artículo que presenta cierta relevancia para este proyecto, ya que habla sobre Azure DevOps[2] y conceptos claves como la automatización de *pipelines* y la infraestructura como código.
- **A Case for a New IT Ecosystem: On-The-Fly Computing** [17]: El artículo presenta cierta relevancia, ya que busca que servicios informáticos puedan ser configurados y ejecutados de manera automatizada, facilitando el desarrollo y el despliegue.

[1] https://www.scopus.com/
[2] https://azure.microsoft.com/en-us/products/devops

- **Automated Threat Analysis and Management in a Continuous Integration Pipeline** [18]: Este documento está altamente relacionado con nuestro proyecto, ya que aborda la integración de herramientas automatizadas en *pipelines* de CI/CD, específicamente en *GitLab*, para gestionar amenazas de seguridad y privacidad.

Si se desea profundizar sobre cada artículo, en el Apéndice A.2 se describe con más detalle este análisis.

3.1.2. *Multiexperience Development Platforms*

En esta sección se describen las *Multiexperience Development Platforms (MXDP)* dado que cuentan con características relevantes para el proyecto. Este término surgió en el marco de una presentación intermedia con docentes del LINS[3] donde el objetivo era validar la idea y recibir retroalimentación. Las MXDP ofrecen un conjunto integrado de herramientas de desarrollo frontend y capacidades de *backend for frontend* (BFF), diseñadas para simplificar y acelerar la creación de aplicaciones adaptadas a diversos puntos de contacto digitales y modalidades de interacción [19].

Las *Multiexperience Development Platforms* buscan abstraer a los desarrolladores de la complejidad técnica inherente a la creación de aplicaciones para múltiples plataformas y dispositivos. Estas plataformas proporcionan herramientas preconfiguradas y componentes reutilizables que eliminan la necesidad de implementar soluciones desde cero, permitiendo a los desarrolladores centrarse en la lógica de negocio y en la experiencia del usuario. Al automatizar tareas como la generación de binarios, la integración con servicios backend y la adaptación a distintos puntos de contacto, las MXDP aceleran el desarrollo y garantizan la consistencia entre aplicaciones, optimizando los recursos del equipo de desarrollo. Estos puntos de contacto o *touchpoints* se refieren a la manera en la que se puede interactuar con las aplicaciones, ya sea por comando de voz, escrita, entre otras formas.

Algunos ejemplos que se pueden encontrar en el mercado son los siguientes[4]:

- OutSystems [20]
- Salesforce Lightning Platform [21]
- Firebase (por Google) [22]
- Kinvey [23]
- Power Apps (por Microsoft) [24]

[3] Laboratorio de Integración de Sistemas (LINS), perteneciente al Instituto de Computación (INCO) de la Facultad de Ingeniería (FING) - https://www.fing.edu.uy/inco/grupos/lins/

[4] https://www.gartner.com/reviews/market/multiexperience-development-platforms

- GeneXus (por Globant) [25]

Entre las plataformas mencionadas, **Firebase** y **OutSystems** presentan características que podrían ser comparadas con este proyecto. Firebase, desarrollado por Google, proporciona un ecosistema integrado que incluye bases de datos en tiempo real, autenticación y hosting, con un fuerte enfoque en el backend como servicio (BaaS). Por otro lado, OutSystems se centra en ofrecer herramientas visuales para el desarrollo rápido de aplicaciones, incorporando capacidades de DevOps como la integración continua y el monitoreo de despliegues. Sin embargo, mientras Firebase se especializa en la simplificación de tareas backend y OutSystems en la creación rápida de aplicaciones frontend y backend, el enfoque de este proyecto está en automatizar y estandarizar la configuración de herramientas DevOps e infraestructura como código, cubriendo un espacio menos atendido por estas plataformas y permitiendo mayor control del ciclo de vida del desarrollo de software.

3.2. Análisis de requerimientos

En esta sección se presentan los requerimientos funcionales y no funcionales del proyecto.

3.2.1. Requerimientos funcionales

Este proyecto tiene como objetivo principal el desarrollo de una plataforma tecnológica que proporcione soluciones DevOps, adaptadas a las necesidades específicas de los usuarios.

Se apunta a construir una plataforma con una interfaz gráfica que mitigue el costo de entrada a los desarrolladores para introducir herramientas DevOps a su proyecto. Para ser más específicos, este costo de entrada se trata de la curva de aprendizaje que conlleva la implementación. Este costo se da no solo por el hecho de aprender tecnologías nuevas, sino también por la gran cantidad de propuestas del mercado que hace difícil una elección adecuada.

Con esto en mente, se llegó al siguiente listado de requerimientos funcionales:

1. Iniciar sesión en la plataforma mediante un flujo OAuth con un proveedor de identidad.
2. Gestión de proyectos (o soluciones) DevOps. Crear, modificar y eliminar.
3. Desplegar proyectos.
4. Compartir proyectos y tecnologías con otros usuarios.
5. Extender la plataforma agregando nuevos componentes desde la interfaz.

6. Gestión de claves de acceso de los proveedores de *cloud*.

La gestión de proyectos es la parte central de la plataforma, la plataforma debe permitir a los usuarios crear y gestionar nuevos proyectos para configurar acorde a sus necesidades. Esto es, dejándoles seleccionar qué tecnologías necesitan y cómo se conectan entre ellas, ofreciendo un catálogo de tecnologías disponibles. En base a esto la plataforma debe generar la «solución DevOps», desplegando el proyecto en proveedores de *cloud*, y gestionando el código, las herramientas y la infraestructura en repositorios.

La plataforma debe ser extensible en tecnologías. Para esto, tiene que ser posible agregar nuevas tecnologías y herramientas desde la interfaz, haciendo que otros usuarios puedan extenderla y mantenerla.

Debe existir colaboración entre los usuarios, especialmente entre los que ya poseen un conocimiento de DevOps y los que no. Para esto se requiere que las principales entidades del sistema (proyectos y componentes) puedan opcionalmente ser compartidas entre los usuarios.

3.2.2. Requerimientos no funcionales

Además de los requerimientos funcionales, la plataforma deberá satisfacer algunos requerimientos no funcionales como: usabilidad, seguridad y mantenibilidad.

En lo que respecta a la usabilidad, la idea es desarrollar una interfaz gráfica para los usuarios y que no suponga dificultades al momento de trabajar con la plataforma. A modo de tener una forma visible y ordenada de dichos proyectos, se desarrollará el *Canvas* para la creación de proyectos. Este término se explicará en la Sección 5.1.2.

Por otro lado, es importante tener en cuenta que para la creación y gestión de repositorios y de la infraestructura, es necesaria la manipulación de credenciales de los usuarios. Por eso se tiene en cuenta la seguridad como requerimiento no funcional. Para ingresar y utilizar la plataforma se necesitará estar autenticado y se guardarán todas las credenciales externas e información sensible de manera encriptada.

Finalmente, se buscará desarrollar una plataforma que sea mantenible. Esto quiere decir que el código y la infraestructura del mismo serán diseñados para que puedan extenderse y siguiendo prácticas de desarrollo que la permitan ser comprensible y modificable.

3.3. Resumen y conclusiones

Como conclusión del capítulo, se puede apreciar que por parte de los artículos académicos hay un acercamiento a la automatización de la configuración de herramientas DevOps. Quizás más enfocados en partes específicas del proceso como la creación automática de *pipelines* CI/CD o fuertemente basados en un único proveedor de *cloud* como lo es Azure DevOps. Con esto en mente, se ve que la solución que se propone en este proyecto busca facilitar el desarrollo de software de una manera más general que en los artículos académicos analizados, con la finalidad de incorporar un paquete de herramientas DevOps que ayude a todo el ciclo de vida de la aplicación y sin depender de tecnologías específicas.

En comparación con una plataforma MXDP, el proyecto en el que se está trabajando se centra en la automatización de herramientas DevOps y la creación de infraestructura para facilitar el ciclo de vida de desarrollo de aplicaciones. Mientras que una MXDP se enfoca principalmente en la experiencia de usuario y en la creación de aplicaciones, el proyecto busca automatizar y simplificar los procesos relacionados con el desarrollo y la infraestructura, eliminando la necesidad de configuraciones manuales de forma que los desarrolladores puedan centrarse exclusivamente en el código y las funcionalidades de sus aplicaciones. Firebase y OutSystems presentan capacidades que abordan aspectos parciales de este objetivo, pero ninguna de ellas ofrece un enfoque integral para la automatización DevOps y la infraestructura como código. Ambos enfoques buscan mejorar la eficiencia y la calidad del desarrollo, pero desde perspectivas distintas y complementarias.

Finalmente, se definieron los requerimientos funcionales y no funcionales que describen la plataforma que se busca concluir. Dicha plataforma será implementada como una aplicación web en la que los usuarios tendrán un perfil en donde guardarán y gestionarán sus proyectos. Estos proyectos serán generados en repositorios en el sistema de versionado de código de preferencia del usuario. Dichos repositorios contendrán el código fuente del proyecto, la configuración de las herramientas DevOps y la infraestructura.

Capítulo 4

Plataforma Propuesta

En este capítulo se presenta la plataforma propuesta en este proyecto. En primer lugar, se brinda una descripción general de la plataforma. Luego, se explica el modelo de dominio y la arquitectura de la plataforma. También, se describe la solución generada por la misma, los problemas que se abordan y las decisiones arquitectónicas tomadas.

4.1. Descripción general

En esta sección se describe la plataforma propuesta como solución, que lleva el nombre de ***Bill***. La plataforma en cuestión fue creada para ayudar a mitigar las dificultades enfrentadas por desarrolladores y equipos de software al incorporar herramientas DevOps en sus proyectos. Tiene como objetivo facilitar el proceso de configuración de dichas herramientas, permitiendo a los desarrolladores enfocarse en la lógica de negocio y en la entrega de valor a través del código, solucionando varias complejidades técnicas de la infraestructura y despliegues.

Bill está implementada con una arquitectura modular apuntando a facilitar la adopción de nuevas tecnologías. Está compuesta por un módulo de **backend** y un módulo web de **frontend**.

La plataforma abstrae tareas técnicas complejas como la configuración de *pipelines* CI/CD, la creación de infraestructura mediante principios de GitOps y la integración de herramientas de monitoreo. Todo esto, mediante una interfaz gráfica que busca mejorar la experiencia de los usuarios, evitando la configuración de herramientas a quienes no están familiarizados con prácticas DevOps.

El desarrollo de esta plataforma está motivado por la necesidad de facilitar el acceso y el uso de herramientas que actualmente son esenciales en la industria del software. A través de la automatización, *Bill* busca eliminar barreras técnicas y reducir el tiempo y esfuerzo necesarios para implementar entornos de

desarrollo y producción. Se apunta a promover de esta forma una mayor adopción de prácticas DevOps en contextos educativos y profesionales.

En la Figura 4.1, se presenta una visión general de la plataforma.

Por un lado, se pueden diferenciar tres tipos de usuarios: El «Administrador de Proyectos» que crea y despliega proyectos utilizando componentes preexistentes mediante el *Frontend*. Luego, el «Configurador DevOps», que puede utilizar la plataforma de la misma manera que el Administrador de Proyectos, pero también cuenta con el conocimiento necesario para crear nuevos componentes. Estos componentes pueden ser utilizados tanto por el mismo Configurador como también por otros usuarios para ser reutilizados. Por último, el «Desarrollador» también puede cumplir el rol de Administrador de Proyectos, pero principalmente se encarga de trabajar en la lógica de negocio del proyecto utilizando la solución generada por la plataforma.

Continuando con la figura, dentro de Bill se encuentran los *Proyectos* que están conformados por conexiones de *Componentes*. Dichos proyectos son procesados por los módulos encargados de generar los archivos requeridos y de ejecutar las acciones necesarias para crear y configurar tanto los repositorios como la infraestructura, todo esto mediante la comunicación con los sistemas externos.

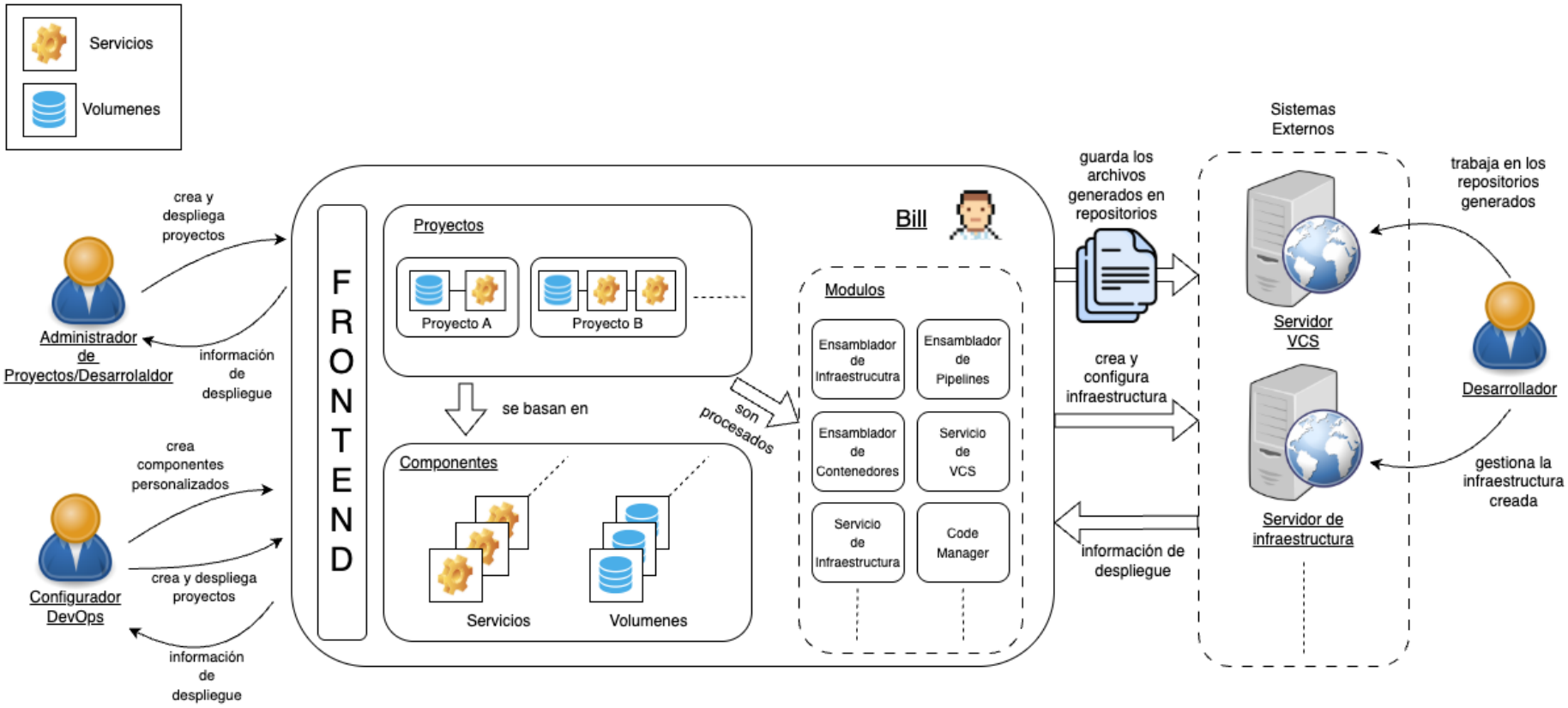

Figura 4.1: Diagrama de la Plataforma Propuesta

4.2. Modelo de Dominio

En la Figura 4.2 se presenta el Modelo de Dominio asociado a la solución.

En esta solución se plantea la entidad de *Usuario* el cual tiene *Variables de Usuario*. Estas variables son utilizadas para acceder y gestionar la infraestructura en los proveedores de *cloud* (Por ejemplo: MiNube y AWS). A su vez, cada usuario puede crear *Proyectos*, *Grupos*, *Componentes* y su configuración. Los proyectos pueden tener múltiples *Despliegues*, lo que permite tener otras configuraciones o incluso distintos proveedores de *cloud*. Los grupos permiten implementar proyectos con componentes desplegados en varios entornos y/o diferentes proveedores en un mismo despliegue, esto quiere decir que la solución soporta proyectos *multicloud*.

Los *Componentes* representan las tecnologías que conforman un proyecto, como bases de datos, frameworks de backend, frontend, entre otros. Continuando la figura, se puede apreciar que cada componente cuenta con *Conexiones*, para poder habilitar los puertos del mismo. También, dichos componentes tienen *Conectores* que se encargan de tener la configuración necesaria para realizar un despliegue del componente. Un ejemplo de estos conectores puede ser el conector de *Docker*, que cuenta con el *Dockerfile* del componente para que pueda ser ejecutado dentro de un contenedor. Los conectores contienen *Variables* que necesita el componente para funcionar correctamente en el conector implementado. Un ejemplo recurrente de estas variables serían las credenciales de usuario, en el caso de que el componente sea una base de datos. Finalmente, los *Ensambladores* son los encargados de generar los archivos de texto necesarios para desplegar cada proyecto. En la Sección 4.3.3 se describe con más detalle el rol que cumplen los ensambladores en la solución.

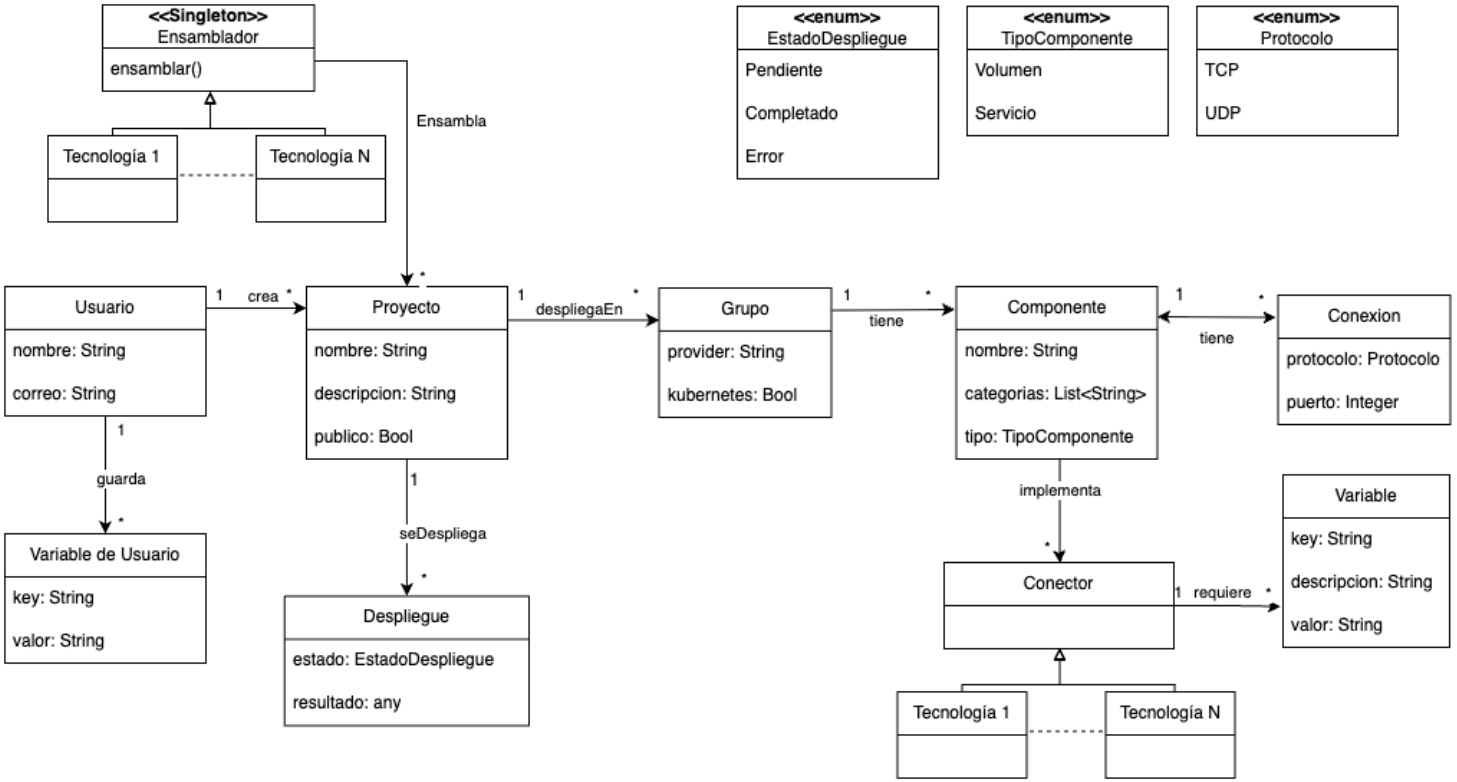

Figura 4.2: Modelo de Dominio de la Solución

4.3. Arquitectura de la Plataforma

En la Figura 4.3 se describe la arquitectura de la solución. Se pueden diferenciar 5 tipos de componentes: el *Frontend*, el *Controlador*, los *Ensambladores*, el *Code Manager* y los *Servicios*.

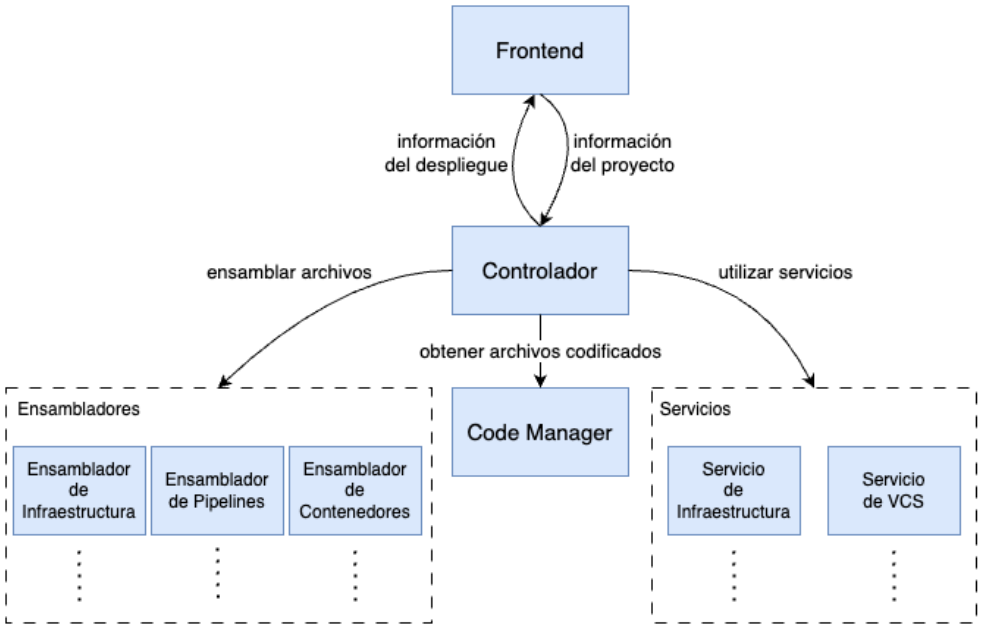

Figura 4.3: Arquitectura de la plataforma (*Bill*)

En las siguientes sub-secciones se describen con más detalle los principales componentes de la plataforma.

4.3.1. Frontend

El módulo Frontend se encarga de la interfaz gráfica que utilizan los usuarios para interactuar con la plataforma. A través de esta interfaz es posible crear proyectos, configurar componentes y gestionar despliegues, entre otras acciones. La comunicación con el Backend se realiza por medio de una serie de API's REST.

4.3.2. Controladores

El módulo Controlador se encarga de orquestar el trabajo que realizan los ensambladores, el *Code Manager*, y los servicios involucrados en la creación y

gestión de un proyecto. Su rol principal es asegurar que se generen, suban y apliquen correctamente los archivos de configuración necesarios para el despliegue. La Figura 4.4 muestra el flujo típico que sigue el Controlador durante este proceso.

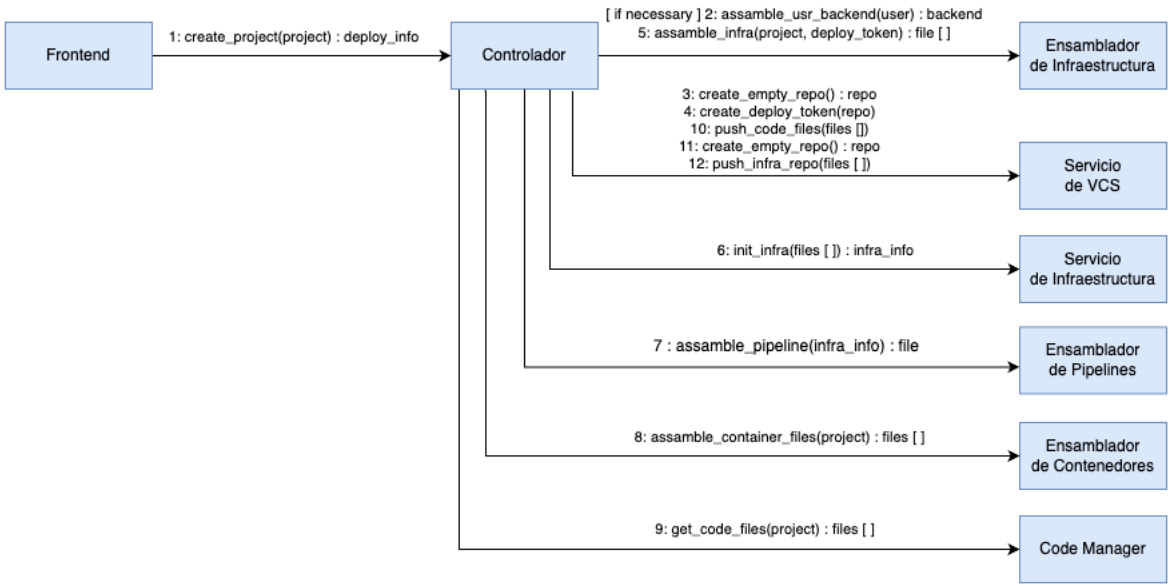

Figura 4.4: Flujo del Controlador para la creación y despliegue de un proyecto.

A continuación se describe el flujo que representa la Figura 4.4. Cada elemento del siguiente listado corresponde uno a uno a los de dicha figura.

1. Se envía la información necesaria para crear y desplegar el proyecto del usuario.
2. Si el usuario no cuenta con uno, se crea un *backend* para el usuario en un controlador de versiones *online* (VCS). Este *backend* es un repositorio en donde se guardará el código de la infraestructura para el proyecto. Para esto, se le envía la información del usuario al ensamblador y el mismo se encarga de crear lo necesario, según el proveedor elegido.
3. Se crea un repositorio vacío en el VCS del usuario para la infraestructura.
4. Se crea un *token* para realizar el despliegue de la infraestructura.
5. Se crean los archivos que contienen la infraestructura del proyecto en el ensamblador de infraestructura.
6. El servicio de infraestructura se encargará de iniciar y aplicar la infraestructura requerida (un ejemplo de este tipo de servicio es el *Terraform Service*).

7. En el ensamblador de *pipelines* se crean los archivos del *Pipeline CI/CD* de la infraestructura y el código del proyecto.
8. Se ensamblan los archivos para la containerización del proyecto. Es decir, se generan los archivos necesarios para que el proyecto pueda ser ejecutado en contenedores (por ejemplo, se pueden crear los *Dockerfiles* en el caso de que el proyecto utilice *Docker*).
9. Se le solicitan los archivos del proyecto al *Code Manager*
10. Se crea un repositorio vacío para el código del proyecto.
11. Se sube el código del proyecto.
12. Se sube el código de la infraestructura del proyecto.

4.3.3. Ensambladores

Las instancias de *Ensambladores*, como bien lo dice su nombre, se encargan de ensamblar archivos basándose en los distintos componentes que conforman a los proyectos. Un ejemplo de ensamblador sería un `docker_assembler`, que se encarga de generar los *Dockerfiles* y los *docker-compose.yaml* para poder correr el proyecto en contenedores.

4.3.4. Code Manager

El módulo *Code Manager* se encarga de manipular y preparar los archivos que serán subidos al sistema de control de versiones. Más en específico, está destinado a crear una estructura que contenga todo lo necesario para crear y subir el repositorio que contiene el código fuente y los archivos de configuración necesarios para las herramientas DevOps de un proyecto. Se ocupa de codificar en `base64` los archivos y agregarles algunos atributos necesarios para enviarlos. A continuación, se presenta a modo de ejemplo una función del *Code Manager* que crea una estructura para un archivo.

```
def create_git_file(self, path: str, content: str):
    return {
        "action": "create",
        "file_path": path,
        "encoding": "base64",
        "content": base64.b64encode(content.encode()).decode()
    }
```

4.3.5. Servicios

Los Servicios se encargan de abstraer funcionalidades de tecnologías específicas. Esto promueve la arquitectura modular de la plataforma, permitiendo que

la integración de un nuevo servicio o tecnología externa se realice únicamente mediante el desarrollo de un nuevo servicio. Continuando con el ejemplo que se mencionó en la descripción de la Figura 4.4, el *servicio de infraestructura* se podría encargar de ejecutar las acciones que gestionan la infraestructura del proyecto a desplegar. Como otro ejemplo se puede tomar el *servicio de VCS*, que se ocupa de manipular los repositorios remotos y ejecutar funcionalidades de *Git* como lo son el *push*, *commit*, crear repositorios, entre otras.

4.3.6. Diagrama de Despliegue

La Figura 4.5 presenta el diagrama de despliegue de la solución. En este, se pueden apreciar los diferentes componentes de *hardware* y *software* que intervienen.

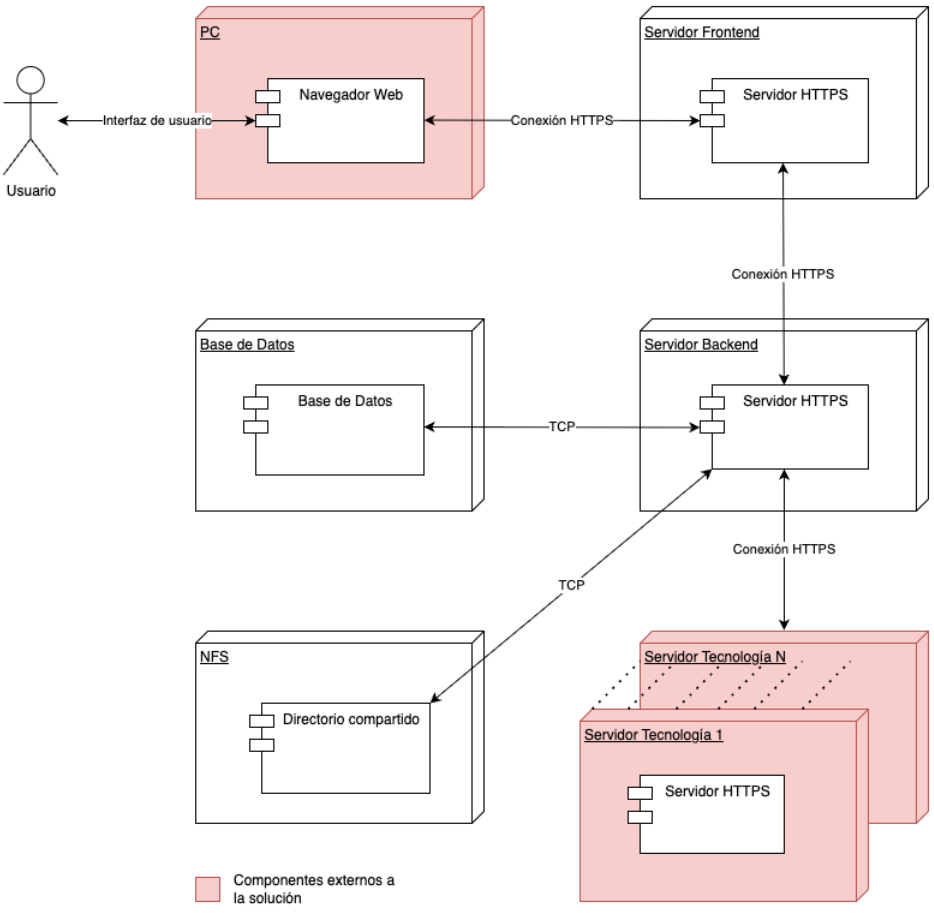

Figura 4.5: Diagrama de Despliegue de la solución

Existen dos tipos de componentes: los internos a la solución que se plantea

en este trabajo y los externos a la misma. Entre los componentes internos se encuentran los servidores de *backend* y *frontend*, una base de datos y un sistema de archivos de red (NFS). El sistema NFS implementado se utiliza para almacenar el código fuente de los componentes, lo cual permite evitar la pérdida de archivos en caso de fallas en el servidor. Además, resulta útil en escenarios de escalado horizontal del servidor *backend*, ya que permite que múltiples instancias accedan a un sistema de archivos compartido. Por otro lado, se tienen los componentes externos que son conformados por una computadora personal (PC) que ejecuta el navegador web y los servidores de tecnología. Entrando en detalle en estos últimos componentes, los servidores de tecnología HTTPS permiten gestionar las diferentes herramientas que forman parte de la solución DevOps. Entre algunos ejemplos de estos servidores se encuentra el de *GitLab* para la creación y gestión de repositorios y *pipelines*, y los servidores de proveedores de *cloud* como MiNube[1] o AWS.

[1] https://minubeantel.uy/

4.4. Solución Generada

La plataforma busca entregar al usuario una solución lo más completa posible integrando diversas tecnologías con el fin de que el usuario obtenga una solución base inicial sólida de la cual partir, implementando diversos principios de DevOps y acortando la barrera de entrada existente en la utilización de estas tecnologías. El usuario es capaz de seleccionar qué tecnologías y componentes quiere en su proyecto, eligiendo una solución completamente personalizada dependiendo de sus necesidades. Para esto el usuario selecciona qué componentes[2] quiere utilizar en su proyecto, cómo se conectan entre sí, qué *cloud* quiere utilizar para desplegar la solución (junto con otras opciones de configuración) y la plataforma se encarga de crear y entregar la solución al usuario.

Como resultado de la ejecución de la plataforma, luego de creado y desplegado el proyecto del usuario, se obtienen dos repositorios: el repositorio de código y el de infraestructura. El repositorio de código contiene los archivos ya configurados del proyecto, listo para desarrollar la lógica de negocio pertinente al proyecto y ser desplegado. Por otro lado, el repositorio de infraestructura contiene toda la infraestructura como código del proyecto. A su vez, la plataforma se encarga de crear y aplicar la infraestructura directamente en los proveedores de infraestructura en el caso que el usuario lo ordene.

A continuación se describen las características que tiene esta solución:

4.4.1. Gestión del código

Para la gestión de código, se utiliza un servidor de **GitLab** como plataforma de alojamiento de código. Para cada proyecto el sistema crea dos repositorios en GitLab. En el *repositorio de código*, del tipo *monorepo*[3], se aloja el código de los componentes seleccionados por el usuario. Adicionalmente, se crea otro repositorio para la infraestructura, se habla más en detalle de dicho repositorio en la Sección 4.4.5 sobre **GitOps**.

4.4.2. Pipelines de CI/CD

Para cada componente se crean *pipelines* de CI/CD, los cuales implementan la integración continua y los despliegues automáticos. Los *pipelines* por defecto cuentan con tres estados, **test**, **build** y **deploy**. Los *pipelines* de integración se ejecutan al integrar nuevo código a la rama principal, mientras que los *pipelines* de deployment se activan automáticamente cuando el usuario realiza una nueva «release» desde la interfaz de *GitLab*. Cada pipeline es distinto dependiendo de

[2]Los componentes son las piezas fundamentales que componen el proyecto, por ejemplo, una aplicación Java, un frontend en React, una base de datos Postgres, un cache con Redis, etc.

[3]Repositorio único que contiene el código fuente de múltiples proyectos o componentes relacionados

los componentes que se han elegido, ya que estos implementan acciones particulares de cada tecnología, pudiendo ser la ejecución de tests unitarios, análisis de dependencias, calidad de código utilizando un *linter*[4], etc.

4.4.3. Contenedores

Cada componente se entrega con su entorno de ejecución encapsulado en un contenedor[5], de forma que se puede empezar a trabajar en el desarrollo sin necesidad de estar haciendo configuraciones locales en cada entorno particular para correr el entorno. Además, queda abierta la posibilidad de realizar despliegues aprovechando los contenedores, por ejemplo utilizando Kubernetes.

4.4.4. Infraestructura

La infraestructura se define usando *Infraestructura como Código* lo que permite su trazabilidad, ya que cada cambio de infraestructura queda guardado en un repositorio *git* con todo el historial de cambios. En este sentido, cuando el usuario avanza con una solución y realiza un despliegue, la plataforma crea la infraestructura definida por los componentes del proyecto. Se configuran las *keys* en los proveedores de *cloud* elegidos y variables de entorno requeridas en el *repositorio de infraestructura*, en donde también se almacena su código.

4.4.5. GitOps

Dado que la infraestructura se define como código, se configura su repositorio como un repositorio **GitOps**. En él se almacena todo lo relacionado a la infraestructura y se incluyen las claves necesarias para su funcionamiento. A partir de esto, se crean los pipelines que permiten gestionar su ciclo de vida: creación, modificación y eliminación directamente desde el repositorio.

4.4.6. Monitoreo y logging

Como última pieza de la solución entregada, se brindan herramientas de monitoreo y *logging*. Las características de la solución con respecto al monitoreo y *logging* dependen de las características y los componentes elegidos, dado que se hace uso de las funcionalidades que ya proveen los proveedores de *cloud* para estos fines. Se profundiza sobre estos casos en el Capítulo 5.2.6.

4.5. Problemáticas Abordadas

En esta sección se describen las principales problemáticas identificadas y abordadas durante el desarrollo del proyecto.

[4]Herramienta que analiza el código fuente para detectar errores, malas prácticas o desviaciones de un estilo definido, ayuda a mantener la calidad y consistencia del código.

[5]Esto queda definido en los Dockerfile y docker-compose dentro del mismo repositorio.

4.5.1. Integración de Distintas Tecnologías

Existe una gran cantidad de combinaciones tecnológicas en un mismo proyecto, lo que presenta un desafío significativo al momento de generar los archivos de configuración necesarios para su despliegue. Esta dificultad se debe a que cada componente puede requerir variables específicas, establecer conexiones con otros servicios, o seguir ciclos de despliegue distintos, entre otras particularidades.

Para abordar esta problemática, se realizó un proceso de abstracción que permitió identificar patrones comunes entre los distintos archivos de configuración. A partir de estas similitudes, se definieron plantillas (*templates*) que facilitan su generación automática, independientemente de las tecnologías involucradas. Esto habilita a que los usuarios puedan integrar nuevas tecnologías creando sus propios *templates*, definiendo variables y reglas de generación en base a ellas.

4.5.2. Reutilización de Componentes

Al momento de crear proyectos, la configuración de cada componente puede generar problemas para los usuarios. Esto se debe al tiempo y el conocimiento que se necesita, como por el retrabajo que genera si se necesita volver a utilizar un componente igual o similar en otros proyectos.

Esto se soluciona con la entidad *Componente* de la plataforma, que permite reutilizar los componentes dentro de diferentes proyectos del usuario creador. También, dichos componentes tienen la posibilidad de ser públicos, concediendo a otros usuarios a utilizarlos sin la necesidad de crear y configurar cada componente desde cero.

4.5.3. Extensibilidad

Diseñar una plataforma con capacidad de extensibilidad representa un desafío importante. Este enfoque busca asegurar que, en el futuro, sea posible incorporar nuevas tecnologías sin necesidad de realizar modificaciones significativas en el código existente.

Para lograr esto, se plantea una arquitectura fuertemente basada en módulos, donde cada módulo se responsabiliza de una parte particular del proceso.

Cuando un proyecto es configurado y desplegado por un usuario, el mismo pasa por una serie de ensambladores independientes los cuales toman como entrada el mismo proyecto, retornando una salida que habitualmente condice a una serie de archivos de configuración.

Es decir, cada tecnología concreta es gestionada mediante conectores de esa tecnología particular, junto con un ensamblador que implementa la lógica de

conexiones entre esos conectores dentro del proyecto. Por lo tanto, agregar soporte a una nueva tecnología de DevOps se basa principalmente en la creación de un nuevo conector que de soporte a esa tecnología, junto con un ensamblador que genere la solución en base a esos conectores.

4.5.4. Creación y Mantenimiento de la Infraestructura

Al momento de crear la infraestructura de un proyecto, uno de los principales desafíos es la diversidad de proveedores de servicios en la nube (*cloud*), ya que cada uno ofrece mecanismos distintos para definir y aprovisionar recursos. Esto obliga al usuario a investigar y adaptar su proyecto según las particularidades de cada proveedor, lo cual puede ser complejo y propenso a errores.

Además, el mantenimiento de la infraestructura plantea problemas de trazabilidad, especialmente cuando se aplican cambios que no quedan registrados de forma clara o cuando es necesario volver a un estado anterior en caso de fallos (lo que se conoce como *rollback*).

Para abordar esta problemática, se utilizó el enfoque de Infraestructura como Código (IaC). De forma similar a la generación de archivos de configuración mediante plantillas, también se generan automáticamente archivos IaC que permiten definir y desplegar la infraestructura del proyecto. Esto facilita su trazabilidad, mejora el control de versiones y permite realizar *rollbacks* de manera más segura y estructurada.

4.5.5. Multicloud

En algunos proyectos, surge la necesidad de desplegar distintos componentes en múltiples proveedores de servicios en la nube. Esto puede deberse a restricciones específicas de cada proveedor, requerimientos de arquitectura o disponibilidad geográfica. Gestionar despliegues en múltiples nubes presenta desafíos como la configuración de credenciales, diferencias en los entornos de ejecución, y la interoperabilidad entre servicios de distintos orígenes.

La plataforma aborda esta problemática permitiendo separar un proyecto en grupos de componentes, donde cada grupo puede asignarse a un proveedor de *cloud* distinto. Esta flexibilidad, combinada con conectores y ensambladores específicos por proveedor, permite abstraer las diferencias y habilitar despliegues distribuidos sin modificar la estructura general del proyecto.

4.6. Decisiones Arquitectónicas

Esta sección trata de las decisiones arquitectónicas relevantes que fueron tomadas al inicio y durante el diseño de la aplicación.

4.6.1. Servicios y Volúmenes

En la etapa inicial del diseño, no se consideraba necesario diferenciar a los componentes entre servicios o volúmenes, cuando la principal diferencia de los mismos es la persistencia de los datos. La definición de componente es general y abarca prácticamente cualquier tecnología que forme parte de un proyecto, desde *frameworks* para desarrollo hasta bases de datos.

Sin embargo, al trabajar con aplicaciones basadas en contenedores, surgió la necesidad de tratar de forma diferenciada a aquellos elementos que requieren persistencia. Los contenedores están diseñados para ser efímeros, es decir, pueden ser destruidos y recreados dinámicamente, lo cual no es adecuado para almacenar datos de forma duradera.

Al diferenciar explícitamente entre servicios y volúmenes, se puede solucionar este problema. Especialmente, esta decisión habilita a modelar adecuadamente a los volúmenes como recursos independientes, con un ciclo de vida diferente al de los servicios.

4.6.2. Grupos de Componentes

Cuando se estaba implementando el despliegue en un proveedor de *cloud*, surgió un problema de arquitectura que no permitía tener despliegues de proyectos tal y como se había pensado en la solución planteada. En especial, este proveedor es el más relevante, ya que se contaba con créditos gratuitos por convenio con la Facultad de Ingeniería (MiNube de Antel).

Al momento de desplegar, un proyecto puede estar creado con varios nodos que se comunican entre sí. Una arquitectura estándar que utiliza este tipo de topología de nodos sería una página web que cuente con un *backend* y un *frontend*, que se comunican por HTTPS y no necesariamente están «hosteados» en la misma máquina. En este caso de ejemplo, contamos con 2 nodos «centrales».

El problema en sí ocurría una vez que se desplegaban proyectos con más de un nodo «central», ya que por ambiente[6], dicho proveedor solo permite un único nodo «central». Esta característica presenta problemas al momento en que se requiera direccionar tráfico por el *load balancer* principal a distintos nodos, ya

[6] En MiNube, un ambiente o entorno, es una agrupación lógica de componentes. Cada ambiente cuenta con un nombre de dominio dentro de MiNube, un *load balancer* y un nodo central, entre otras características particulares del *cloud*.

que este nodo «central» es el único al que el balanceador apunta, y al que se direcciona el dominio.

Para solucionar este problema, se tuvo que hacer una división por ambiente de los componentes y se crearon los «Grupos» de componentes. Cada grupo de componentes permite tener ambientes diferentes entre ellos y la libertad de poder desplegar proyectos con múltiples nodos centrales.

Por otro lado, aunque no surgió de una necesidad específica ni fue un objetivo inicial, la creación de estos grupos de componentes habilitó la posibilidad de que la plataforma soporte proyectos *multicloud*[7]. Cabe destacar la importancia de esta funcionalidad en el ámbito DevOps, ya que permite que los proyectos se dividan en grupos *hosteados* en diferentes proveedores con una configuración hecha por el mismo usuario en la plataforma.

[7]Proyectos que tienen sus componentes distribuidos en diferentes proveedores de *cloud*. Marco Teórico, Sección 2.3.2

Capítulo 5

Implementación

5.1. Implementación de la plataforma Bill

Esta sección trata de las diferentes tecnologías utilizadas para implementar la solución.

5.1.1. Backend

El *backend* se desarrolló en Python, utilizando Django[1]. La decisión de este *framework* en específico fue por la experiencia que se tenía en el grupo con el mismo. Esta experiencia permitió disminuir la curva de aprendizaje al momento de la implementación. Por otro lado, el *framework* presenta facilidades al momento de diseñar funcionalidades como la autenticación de usuarios, conexiones con bases de datos y manejo de archivos, entre otras.

Para el manejo de los conectores y los ensambladores de los archivos de configuración, se utilizó la tecnología de Jinja[2], la cual es un motor de *templating* que facilita la integración entre conectores. También permite que estos conectores puedan ser definidos en tiempo de ejecución por los usuarios, pudiendo incluso definir nuevas variables para ser utilizadas al momento de desplegar una solución (ejemplo: cantidad de nodos en un clúster, o cantidad de memoria en un servidor), y utilizar variables que provee la plataforma al momento de realizar despliegues.

Finalmente, la plataforma cuenta con un motor de base de datos relacional utilizando la tecnología PostgreSQL[3]. La elección de esta tecnología para la base de datos fue impulsada también por afinidad con la misma y por ser gratuita.

[1] https://www.djangoproject.com/
[2] https://jinja.palletsprojects.com/en/stable/
[3] https://www.postgresql.org/

5.1.2. Portal Web

En esta sección se describe el portal web diseñado para la solución. Se brinda una breve descripción de quiénes serán los usuarios finales de la plataforma y sus principales funcionalidades.

Creación de componentes

Los componentes son una pieza importante en el funcionamiento de la plataforma y alojan una gran parte de la complejidad para la configuración y despliegues de proyectos. Por esta razón, se decidió invertir gran esfuerzo para que la experiencia y fluidez de creación/edición sea la mejor posible. Para esto, se escogió implementar un patrón de «wizard», donde el usuario atraviesa una serie de pasos que lo guían por la configuración.

La Figura 5.1 muestra el primer paso del flujo, en el cual el usuario ingresa los datos básicos del componente: nombre, descripción, una imagen representativa y la visibilidad del mismo, es decir, si será público[4]. A continuación, en la Figura 5.2, el usuario debe especificar si el componente es de tipo servicio o volumen. En caso de tratarse de un servicio, tiene la posibilidad de subir los archivos necesarios mediante el receptor correspondiente.

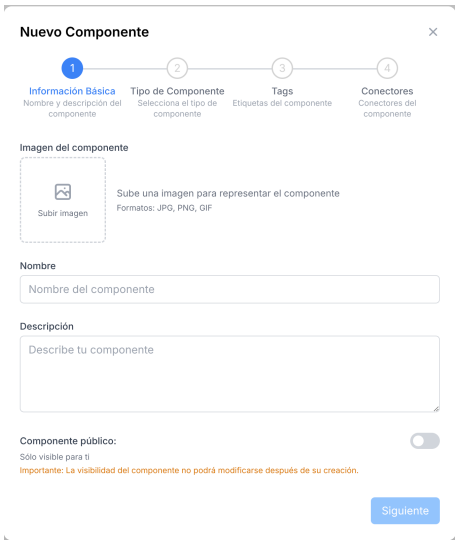

Figura 5.1: Wizard de componentes, Paso 1

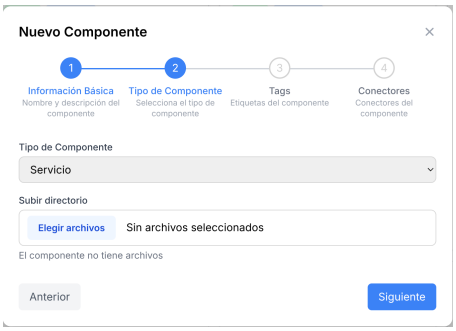

Figura 5.2: Wizard de componentes, Paso 2

En el paso 3, como se muestra en la Figura 5.3, el usuario selecciona una serie de *tags* referentes al componente para facilitar su búsqueda en el listado de

[4]Un componente público puede ser utilizado por otros usuarios en sus propios proyectos.

componentes. En el último paso presentado en la Figura 5.4, el usuario agrega los conectores del componente para que este pueda integrar las distintas tecnologías. Al configurar cada conector el *wizard* muestra una pestaña para cada uno de ellos.

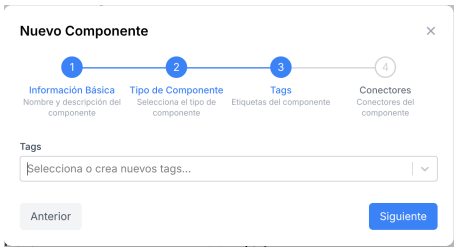

Figura 5.3: Wizard de componentes, Paso 3

Figura 5.4: Wizard de componentes, Paso 4

Creación de proyectos

Para la creación de proyectos, se diseñó una interfaz de usuario que permite configurar un proyecto de forma gráfica mediante diagramas. Con esto en mente, un usuario que desea crear un proyecto indica cómo se relacionan los diferentes componentes de su aplicación a alto nivel en el *Canvas* del creador de proyectos.

Como se puede apreciar en la Figura 5.5, el creador está compuesto por dos partes principales: el *Canvas* y el buscador de componentes. En el buscador, se listan los componentes disponibles para utilizar en el proyecto, pudiendo filtrar por nombre y por tag («frontend», «cache», «database», etc.). Haciendo *click* en uno de estos componentes, se agrega al *Canvas* en donde el usuario los puede distribuir y conectar, uniendo mediante una flecha a otros componentes para formar la arquitectura.

En la parte superior central del *Canvas*, se aprecia una sección para agregar nuevos entornos, en donde opcionalmente se puede seleccionar otro proveedor de *cloud* de forma que se tiene un *Canvas* independiente por entorno y se despliegan juntos al confirmar el despliegue del proyecto. Cada entorno puede llevar su propia configuración y opcionalmente habilitar despliegue con Kubernetes.

Despliegue de un proyecto

Al ir agregando componentes en el *Canvas*, el usuario tiene la opción de guardar el estado de su diagrama para persistir la configuración y volver más tarde.

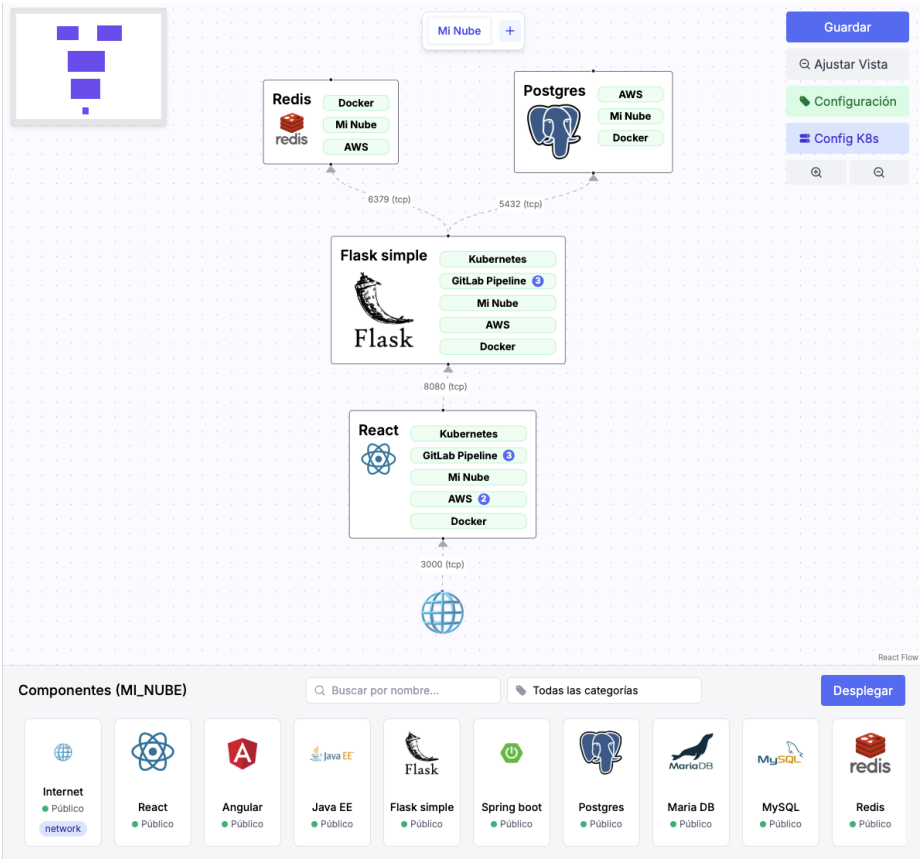

Figura 5.5: *Canvas* del Creador de Proyectos

Luego de haber agregado todos los componentes necesarios y haber reflejado la infraestructura deseada en el *Canvas*, el usuario puede realizar un despliegue con el botón «Desplegar». Inmediatamente después, se iniciará el proceso de despliegue del proyecto en base al estado del *Canvas*.

5.1.3. Despliegue de la solución

Se realizó el despliegue en el proveedor MiNube[5] de Antel, ya que se cuenta con créditos por parte de la universidad para alojar aplicaciones estudiantiles. En el despliegue se utilizó un servidor Apache Python donde se desplegó el *backend*, una base de datos PostgreSQL, y un nodo de almacenamiento NFS. El *backend* almacena una serie de directorios y archivos que corresponden al código de los componentes (de tipo servicio) de los usuarios. Este nodo NFS permite al backend la posibilidad de escalar horizontalmente, ya que no guarda estado en el servidor. Además, el código de los componentes se encuentra en un nodo separado al backend, por lo que en caso de una falla en este no se perdería información. El frontend se desplegó en un nodo Nginx que provee MiNube.

5.1.4. GitLab en la plataforma

Inicialmente la plataforma se integró con el servidor de *GitLab* de la FING, donde debía tener permisos para crear/modificar repositorios del usuario. Esto supone una barrera para que los usuarios puedan probar la solución dado que implica otorgar permisos de lectura y escritura en su cuenta personal de *GitLab* FING, donde muchos alojan sus proyectos personales.

Por esto, se optó por desplegar una instancia nueva e independiente de *GitLab* para el proyecto. Esto se realizó mediante Docker en un servidor de MiNube. Adicionalmente, se configuraron *runners*[6] de forma que se encuentren disponibles en *GitLab* para ser utilizados por los usuarios. Se permitió el acceso a *GitLab* solo a estudiantes o docentes de la FING. Para lograrlo, se configuró un cliente de correo que envía los correos de confirmación automáticos a los correos *@fing.edu.uy* ingresados por los usuarios.

Cabe mencionar que la instancia de *GitLab* utilizada por la plataforma puede ser modificada cambiando las variables de entorno correspondientes.

5.1.5. Prácticas DevOps en la Plataforma Implementada

En el desarrollo de la plataforma se utilizaron diversas prácticas de DevOps. Se utilizaron contenedores Docker tanto para el backend, frontend y la base de datos. Esto permitió al equipo trabajar con el mismo entorno de desarrollo y

[5] https://minubeantel.uy/

[6] En el contexto de integración y entrega continua, los *runners* son agentes de software que se ejecutan en instancias de cómputo determinadas y se encargan de la ejecución de los trabajos (jobs) especificados en los pipelines.

minimizar problemas de configuración.

Se configuró un repositorio en *GitLab* que está integrado con *pipelines* de integración continua, el cual ejecuta tests unitarios sobre el *backend*, chequeos adicionales de Django y realiza el build del frontend. Se realizan despliegues automáticos utilizando una configuración de MiNube, esta configuración detecta cambios en el repositorio y actualiza tanto el *frontend* como el *backend*.

La creación de la infraestructura se realizó mediante *cloud scripting* (ver Sección 5.2.5). De esta forma se apunta a que el proceso de creación pueda ser replicable y la solución pueda ser desplegada nuevamente sin demasiadas complicaciones.

5.2. Implementación de la solución con soporte DevOps

Esta sección trata de las tecnologías particulares que se utilizaron al implementar la solución con soporte para DevOps entregada al usuario.

5.2.1. Contenedores

Como se introdujo en la Sección 2.2.2, los contenedores son un elemento de gran importancia y utilidad al momento de desarrollar y desplegar software. Para esta categoría se utilizó Docker. La elección fue apoyada por dos razones: es una tecnología respaldada por una gran comunidad y también es ampliamente utilizada en el rubro.

5.2.2. Integración de Pipelines CI/CD

En los *pipelines* CI/CD se utilizó CI/CD *Pipelines* de *GitLab*[7]. Esta decisión fue estratégica, debido a que los repositorios de los proyectos del usuario también son desarrollados utilizando *GitLab*, lo que simplifica la creación y uso de los *pipelines*.

5.2.3. Proveedores de Infraestructura

Para elegir a los proveedores de infraestructura se tenía una gran cantidad de opciones. En esta implementación se utilizó AWS[8] y MiNube de Antel[9]. Cada usuario podrá elegir dónde desplegar su aplicación entre estas dos alternativas.

[7] https://docs.gitlab.com/ci/pipelines/
[8] https://aws.amazon.com/es/
[9] https://minubeantel.uy/

5.2.4. Infraestructura como Código (IaC)

Para la infraestructura como código se utilizó la tecnología Terraform[10]. Dicha tecnología es ampliamente aceptada entre los proveedores de infraestructura, lo que permite la reutilización de los archivos de configuración. Esto es útil si surge la necesidad de extender la solución hacia proveedores que no fueron implementados en esta primera versión de la plataforma. En la implementación está disponible el despliegue en AWS utilizando Terraform.

5.2.5. Cloud Scripting

Por otro lado, para el despliegue en MiNube Antel, no se pudo utilizar ninguna tecnología de IaC para los despliegues como el ya mencionado Terraform[11]. Sin embargo, MiNube permite exportar e importar configuraciones de los entornos con archivos *jps*, archivos con una sintaxis de un *yaml* o *json*. Esta tecnología es definida como *Cloud Scripting*[12]. Si bien esto no sustituye a la infraestructura como código, es un acercamiento que permitió tener todos los componentes de cualquier proyecto de MiNube guardados como archivo. A continuación se puede apreciar a modo de ejemplo un archivo *jps* de un entorno Java WildFly[13] con una base de datos PostgreSQL.

```
name: test-project
type: install
nodes:
  - nodeType: postgresql
    nodeGroup: sqldb
    tag: 17.2-almalinux-9
    fixedCloudlets: 3
    cloudlets: 6
    diskLimit: 200G
    scalingMode: STATELESS
    isSLBAccessEnabled: true
  - nodeGroup: cp
    nodeType: wildfly
    tag: 35.0.0.Final-zulujdk-21.0.5-almalinux-9
    fixedCloudlets: 4
    cloudlets: 16
```

Sin embargo, la generación de estos archivos no es suficiente, ya que es necesario importarlos en el proveedor para que las configuraciones sean aplicadas. Para esto, se crearon *pipelines* CI/CD que asignan dichos cambios y realizan

[10] https://developer.hashicorp.com/terraform

[11] Se llegó a esta conclusión luego de leer la documentación de MiNube Antel y de una consulta que se realizó al sistema de tickets de soporte. En dicho ticket se consultó si la plataforma soporta alguna tecnología de IaC y la respuesta fue negativa.

[12] https://docs.cloudscripting.com/

[13] https://www.wildfly.org/

otras acciones necesarias en un despliegue, como lo es instalar dependencias necesarias, cargar variables de entorno y clonar repositorios. Se puede apreciar un ejemplo de estos *pipelines* en el Apéndice B.1, en donde se despliega el entorno WildFly con PostgreSQL mencionado previamente.

5.2.6. Monitoreo y logging

El monitoreo y logging se implementan utilizando las funcionalidades nativas que ofrece cada proveedor de *cloud*. En el caso de MiNube, al ser un entorno administrado por el propio proveedor, ya cuenta con herramientas integradas de monitoreo y logging, por lo que se decide aprovechar estas capacidades en los despliegues realizados sobre su infraestructura. En el caso de AWS, se utiliza específicamente el servicio *CloudWatch*[14], ya que cada componente de la plataforma está diseñado para integrarse con esta herramienta de forma individual.

[14] https://aws.amazon.com/es/cloudwatch/

Capítulo 6

Verificación

Con el fin de verificar que el proyecto cumple con los objetivos propuestos, se plantean casos de estudio reales en los que la plataforma podría ser utilizada.

6.1. Caso de Estudio: Proyecto de TSE

Como primer caso de estudio, se toman las propuestas típicas de la materia Taller de Sistemas Empresariales[1], en donde los estudiantes desarrollan un sistema empresarial aplicando diversas tecnologías. El objetivo de esta materia es formar al estudiante para que sea capaz de desarrollar una aplicación empresarial de mediano y gran porte, integrando diversas tecnologías.

Dado que la implementación del proyecto se realiza en equipos e implica un trabajo colaborativo y simultáneo de desarrollo con despliegues en varios proveedores de *cloud*, la adopción de herramientas de DevOps puede ayudar en los ciclos de entrega de software y facilitar el desarrollo.

Los sistemas que se proponen a nivel de componentes suelen consistir en un componente central en el que se implementa principalmente la lógica de negocio, junto con un *backoffice*, una base de datos relacional, una no relacional (opcional), un *frontend* que puede o no ser parte del componente central y nodos periféricos que se encuentran en otro proveedor de *cloud*.

6.1.1. Componente Central y Bases de Datos

El componente central es un componente de Java empresarial que sigue el arquetipo **wildfly-jakartaee-ear-archetype**. Para definir este componente en el dominio de la solución, procederemos a construir los conectores necesarios.

[1] https://www.fing.edu.uy/es/curso/grado/2018/taller-de-sistemas-empresariales

El proceso comienza con el conector de Docker, que permite empaquetar el componente central de la aplicación empresarial dentro de un contenedor. Este componente requiere, por un lado, un servidor de aplicaciones *Wildfly*, y por otro, la ejecución del proceso de construcción mediante *Maven*. Considerando que el desarrollo de aplicaciones Java ha estado históricamente vinculado al uso de entornos de desarrollo integrados (IDE), se optó por una solución que contemple tanto el empaquetado como la compatibilidad con dichos entornos, facilitando así el trabajo en entornos de desarrollo.

Para esto, en el docker compose parcial del componente central se opta por utilizar dos servicios distintos: el primer servicio se ejecuta en un entorno con *Maven* y Java, el cual realiza el *build* del proyecto. El otro servicio es el servicio que implementa *Wildfly*, este servicio se hace valer del Dockerfile, el cual simplemente levanta el servidor, expone los puertos necesarios y crea un usuario en el servidor *Wildfly* con credenciales provenientes de las variables del conector. Adicionalmente, se define un volumen para las dependencias. De esa forma se evita que se descarguen todas las dependencias cada vez que se realiza el proceso del *build*.

Luego, se vincula un volumen desde la ubicación en la que se encuentra el *ear* al compilar, **ear/target/central.ear** hasta el directorio de *deployments* del servidor *Wildfly*. También, como el servicio que compila tiene un ciclo de vida acotado, se define una dependencia entre servicios, por lo que el servidor se ejecuta por primera vez cuando ya existe el *artifact*. Posteriormente, es necesario ejecutar únicamente el servicio que realiza el *build* para que este refleje los cambios realizados en el código. Por otro lado, es posible compilar a través de IDE y ver reflejados los cambios automáticamente gracias a la configuración del volumen.

En los bloque de código 6.1 y 6.2 se puede ver el Docker compose parcial y su Dockerfile respectivamente, pertenecientes al conector de Docker del componente central.

```
services:
  # Servicio para construir el EAR con Maven
  {{instance_name}}_mvn_builder:
    image: maven:3.9.9-eclipse-temurin-24
    volumes:
      - ./{{instance_name}}/:/app           # Monta el codigo fuente
    local
      - {{instance_name}}_mvn_repo:/root/.m2  # Cache de Maven
    working_dir: /app
    command: sh -c "mvn clean install -DskipTests && chmod -R 777 /app/
    ear/target"

  # Servicio para WildFly
  {{instance_name}}_wildfly:
    build: ./{{instance_name}}
    ports:
      - "8080:8080"
      - "9990:9990"
    volumes:
      - ./{{instance_name}}/ear/target/central.ear:/opt/jboss/wildfly/
    standalone/deployments/central.ear  # Monta el EAR
    depends_on:
      {{instance_name}}_mvn_builder:
        condition: service_completed_successfully

volumes:
  {{instance_name}}_mvn_repo:  # Volumen para cache de Maven
```

Código 6.1: Configuración de docker compose parcial - componente central del ejemplo TSE

```
FROM jboss/wildfly:latest

# Crear usuario admin (sin prompts)
RUN /opt/jboss/wildfly/bin/add-user.sh -u {{wildfly_usr}} -p {{
    wildfly_psw}} --silent

EXPOSE 8080 9990

CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-
    bmanagement", "0.0.0.0"]
```

Código 6.2: Configuración de Dockerfile - componente central del ejemplo TSE

Para realizar un despliegue en MiNube, se añade un conector de esa clase, definiendo en el manifiesto JPS la creación del servidor *Wildfly* en MiNube. En el mismo conector se define un *hook*[2] de *post build*, dicho *hook* se encarga de

[2]En el contexto de MiNube, un *hook* es un script definido por quien configura el despliegue, que se ejecuta en un momento especifico del ciclo de despliegue de una aplicación dentro de

desplegar únicamente el .ear, el cual es el empaquetado que contiene los subproyectos. De lo contrario, el servidor de MiNube intentará desplegar el .ear y el resto de artefactos generados.

Para configurar los *pipelines* de integración y despliegue se agregan dos conectores: uno ejecuta los test unitarios en el *pipeline* de integración y otro que realiza un nuevo despliegue en MiNube. Este último es ejecutado únicamente cuando se realiza una nueva *release* desde la interfaz de *Gitlab*.

Con respecto a la base de datos, se escoge una base de datos PostgreSQL, en la cual son necesarios dos conectores: el conector de Docker en donde se define el docker-compose parcial y el conector de MiNube que se define en el manifiesto. Esta base de datos es desplegada junto con el servidor *Wildfly* cuando se aplique el manifiesto.

6.1.2. Multicloud y Nodos Periféricos

Como se mencionó previamente, el componente central se conecta con otros componentes llamados nodos periféricos. En el marco de los proyectos de TSE, son representados por sistemas externos con los cuales interactúa el componente principal. Estos componentes deben estar desplegados en otro proveedor de *cloud* distinto al del componente principal. Para esto, se hace uso de los grupos de componentes que permiten crear otro grupo que despliega en AWS. Seleccionamos un componente Python con Flask, dado que es liviano y fácil de utilizar, lo que es útil para este caso de simulación de un sistema externo. El nodo periférico se despliega en una única instancia EC2[3] y se integra con un *pipeline* de CD que actualiza la aplicación automáticamente en cada nueva *release*.

6.1.3. Frontend

En el frontend, se seleccionó un componente de React para desplegar en el proveedor principal MiNube. Sin embargo, este componente presenta el siguiente problema: MiNube permite un único componente principal por ambiente. Este componente principal es aquel al que se dirige el tráfico proveniente del dominio que MiNube asigna al crear cada *environment*. También, es el único con el que se puede utilizar el *load balancer* y el cifrado SSL integrado que ofrece. No obstante, estas características son necesarias tanto en el frontend como en el componente central. Para lograrlo, nuevamente se hace uso de los grupos de componentes: se despliega el componente central junto con la base de datos en un grupo y el frontend en otro grupo que también se despliega en MiNube. El componente en React, dado que es un sitio web estático[4] se despliega en un servidor de aplicaciones de NgInx, el cual sirve el contenido de la aplicación

un servidor en MiNube, estos momentos son «pre build», «post build», «pre deploy» y «post deploy»

[3] https://aws.amazon.com/es/ec2/

[4] No se renderiza del lado del servidor

React. Mediante el *pipeline* de deployment, la aplicación se actualiza en cada *release*, volviéndose a compilar y actualizando el contenido servido por NgInx.

6.2. Caso de estudio: Utilización de caso típico en AWS

Esta sección trata sobre el caso de estudio en el que se utiliza AWS como proveedor de *cloud*. En este caso, se toma un proyecto compuesto por tres componentes principales: un backend en *Java Spring Boot*, un frontend en *React* y una base de datos relacional *PostgreSQL*. Al escoger AWS como proveedor para desplegar estos tres componentes, se aprovecha la infraestructura y tecnología que brinda. La idea de presentar el análisis de este caso de uso es brindar un ejemplo que muestre la flexibilidad y adaptación tecnológica de la plataforma, siendo capaz de realizar *deployments* en distintos proveedores, y hacer uso de las tecnologías particulares que más se adaptan para cada componente.

6.2.1. Backend y Base de Datos

Para el backend Spring Boot se busca que sea escalable, resiliente y eficiente. Para esto, el componente aprovecha que ya cuenta con un Dockerfile que permite integrarlo en un contenedor y hacer uso de este en un *cluster ECS*[5]. ECS es una solución manejada de orquestación de contenedores que ofrece AWS. Implementa despliegues de contenedores Docker tanto serverless (versión con Fargate) como en máquinas dedicadas (versión con Provisioning). Esto permite definir reglas de escalado, cantidad de instancias disponibles, junto con otra cantidad de configuraciones.

Para realizar el aprovisionamiento de esta tecnología se utiliza Terraform. La configuración de los conectores del componente se encarga de crear el *cluster*, las reglas de escalado, los roles y políticas necesarias para el funcionamiento del mismo, junto con un *load balancer* que distribuye la carga entre las distintas instancias. Adicionalmente, se crea un Docker Registry[6] en el mismo proveedor de *cloud*, utilizando el servicio de Elastic Container Registry (ECR)[7]. Para realizar los *deployments* en dicho *cluster*, el *pipeline* debe construir el contenedor, subir la imagen al Docker Registry y actualizar el *cluster* para utilizar la nueva versión.

En el bloque de código 6.3 se muestra el contenido del conector del *pipeline* de *deployment*, en su fase de construcción y publicación de una nueva versión de la imagen.

[5]Elastic Container Service: https://aws.amazon.com/es/ecs/
[6]https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-registry/
[7]https://aws.amazon.com/es/ecr/

```
{{instance_name}}_push_registry:
  stage: deploy
  image: docker:24.0.5
  services:
    - docker:24.0.5-dind  # Docker-in-Docker
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
    SHORT_TAG: $CI_COMMIT_SHORT_SHA
    PROD_TAG: prod
    IMAGE_NAME_BASE: $ECR_REPO_URL
    IMAGE_TAG_SHA: $IMAGE_NAME_BASE:$SHORT_TAG
    IMAGE_TAG_PROD: $IMAGE_NAME_BASE:$PROD_TAG

  before_script:
    - apk add --no-cache curl jq python3 py3-pip
    - pip install awscli

  script:
    - echo "Building Docker image..."
    - cd {{instance_name}}
    - aws ecr get-login-password | docker login --username AWS --password
    -stdin $ECR_REPO_URL
    - docker build -t $IMAGE_TAG_SHA .
    - docker tag $IMAGE_TAG_SHA $IMAGE_TAG_PROD
    - docker push $IMAGE_TAG_SHA
    - docker push $IMAGE_TAG_PROD
  rules:
  - if: $CI_COMMIT_TAG
```

Código 6.3: Configuración del pipeline de deployment en AWS, fase de *build* y publicación del contenedor, componente de spring boot

Para la fase de build, se sigue una estrategia que va a permitir realizar *rollbacks*, en caso de que se quiera volver a una versión anterior rápidamente. Como se ve en la configuración del *pipeline* en la fase de build, el contenedor se sube utilizando el hash del commit[8] como tag, ya que es un identificador único y correspondiente a una versión particular dentro del repositorio. A su vez, se le asigna el tag *prod* el cual es el que toma el *cluster* para realizar los despliegues. De esta forma, se consigue que sea fácil actualizar a una nueva versión, dado que no se tiene que realizar cambios en la parte de infraestructura. En cada actualización, es necesario subir una nueva imagen, asignarle el tag *prod* y forzar un *replacement*. En el caso de que algo salga mal es posible realizar un *rollback* asignando al tag *prod* en la versión estable deseada, realizando nuevamente otro *replacement*.

En la fase de actualización del clúster, solo es necesario actualizar las ins-

[8]En el versionado de código, los commits cuentan con un hash (*string* de letras y números aleatorios) como identificador.

tancias, dado que ahora la imagen tageada como *prod* tiene la nueva versión. El código 6.4 muestra el contenido del conector que lo implementa.

```
{{instance_name}}_deploy:
  stage: deploy
  image: python:3.11

  before_script:
    - pip3 install awscli
    - aws --version
  script:
    - echo "Deploying 'prod' image to ECS service..."
    - aws ecs update-service --cluster $ECS_CLUSTER_NAME --service
    $ECS_SERVICE_NAME --force-new-deployment
  needs:
  - job: {{instance_name}}_push_registry
  rules:
  - if: $CI_COMMIT_TAG
```

Código 6.4: Configuracion del pipeline de deployment en AWS, fase de actualizacion del cluster, componente de spring boot

Para la base de datos, se utilizó el servicio de Relational Database Service[9] (RDS), el cual es un servicio manejado por el proveedor que permite la creación de bases de datos relacionales, pudiendo seleccionar entre distintos motores. La misma no presenta grandes dificultades al momento de configurar y provee opciones de escalado, backups, versiones, entre otras funcionalidades. Para esto, se implementó un conector de Terraform que es el encargado de la creación de la infraestructura en AWS.

6.2.2. Frontend

Para el frontend en React, también se buscó que sea escalable y eficiente. Dicho esto, se toma en cuenta el hecho de que se trata con una aplicación web estática. Esto quiere decir que una vez compilada la aplicación, se obtiene una serie de archivos *html, js, y css*, junto con otros archivos estáticos. Todos estos archivos pueden ser servidos por casi cualquier servidor de aplicaciones (ej.: Nginx y Apache HTTP Server), abriendo nuevas posibilidades de despliegue. Con esto en mente, se usará una solución dedicada y optimizada para servir archivos estáticos, almacenando los archivos en un bucket s3[10], el cual es un servicio de almacenamiento de datos. Este sistema se mantiene privado y no puede ser accedido desde internet, ya que para acceder al servicio web lo hacemos por medio del servicio de Cloudfront [11], que se encarga de servir los archivos

[9] https://aws.amazon.com/es/rds/
[10] https://aws.amazon.com/es/s3/
[11] https://aws.amazon.com/cloudfront/

almacenados en el bucket.

El *pipeline* de *deployment* se encarga de realizar el *build* la aplicación en una serie de archivos estáticos, actualizar el contenido del servicio de almacenamiento, e invalidar la cache del servicio del *Cloudfront* para que este tome el contenido del servicio de almacenamiento nuevamente y se actualicen los cambios.

6.3. Pruebas automatizadas

Se desarrollaron pruebas automatizadas (unitarias y de integración) para verificar el correcto funcionamiento del código. Mediante estas pruebas, se alcanzó una cobertura del 84 % del total del código, con un 91 % de cobertura en los módulos de la lógica de negocio.

Capítulo 7

Validación

En este capítulo, se describen las diferentes instancias de validación que se realizaron en el proyecto: una presentación intermedia y una instancia de uso real de la plataforma, con docentes, estudiantes y egresados de la Facultad de Ingeniería.

7.1. Presentación Intermedia

Una vez que el desarrollo del prototipo de la plataforma estuvo en un estado avanzado, se realizó una presentación intermedia del proyecto a docentes del LINS de la Facultad de Ingeniería. La presentación consistió de una parte oral, una demo y una ronda de preguntas, con una duración de una hora.

Como sensación general, la plataforma tuvo una recepción positiva con los oyentes que rápidamente vieron el potencial de la herramienta. A su vez, en la ronda de preguntas, surgieron cuestiones interesantes. Algunas de las más importantes fueron las siguientes:

1. ¿Por qué no se desarrolló un despliegue utilizando Kubernetes?
2. ¿Cuáles fueron los portales en donde se hizo la búsqueda del estado del arte?
3. ¿Se comparó dicha plataforma con las plataformas definidas como *Multi-experience Development Platforms*?
4. ¿Se pueden modificar componentes en proyectos desplegados desde la plataforma y que esos cambios impacten en el mismo?
5. ¿Cómo vamos a evaluar a la plataforma? ¿Se harán pruebas de carga o alguna otra validación?
6. ¿Cuáles son los pendientes que quedaron en la sección de «Trabajo Futuro» que consideramos que son más importantes?

7. ¿Qué costos de mantenimiento tiene la plataforma? Estos costos serían por usuario y por el mismo despliegue de la plataforma para que esté *online*.
8. ¿Se puede borrar o deshacer un despliegue? Eso sería una funcionalidad muy útil en el ámbito DevOps.

A raíz de la pregunta 1 sobre *Kubernetes*, el equipo decidió implementar un despliegue de estas características en la plataforma. Esta decisión también fue impulsada por la gran popularidad que tiene dicha herramienta en el sector DevOps. Dada la importancia de incorporar esta tecnología en la plataforma y considerando que esta incorporación funciona como validación de la expansibilidad de la plataforma. La incorporación se detalla en la Sección 7.2.

Para resolver o responder las preguntas 2 y 3 se agregó y desarrolló la información que se menciona cuando se habla del conocimiento existente en la Sección 3.1.

Sobre la pregunta 4, la respuesta corta es «no». El caso de uso principal, sería utilizar la plataforma solo en el momento de la creación del proyecto y a partir de ahí desarrollar y/o hacer las configuraciones específicas en el caso de que sea necesario. Sin embargo, sin duda que sería una funcionalidad muy interesante para dejar en la sección «Trabajo Futuro», por su utilidad y también por la complejidad que llevaría desarrollarla.

Siguiendo con la pregunta 5, la evaluación de la plataforma se llevó a cabo con los siguientes tres métodos: utilizar y desplegar un caso de uso de un proyecto real de Taller de Sistemas Empresariales (TSE), utilizar y desplegar un proyecto genérico para poder probar las principales funcionalidades de la plataforma y como último método de evaluación darle acceso a estudiantes o docentes de la Facultad de Ingeniería para que puedan usar la plataforma.

La pregunta 6 se responderá en la Sección 8.4 que trata sobre el «Trabajo Futuro».

En lo que respecta al mantenimiento, como lo cuestiona la pregunta 7, la plataforma puede ser desplegada con una infraestructura estándar como sería la de una página web con un servidor *backend* y otro *frontend*. Por otro lado, cada proyecto que creen los usuarios, son desplegados utilizando las credenciales personales de cada uno. Estos gastos no quedan contemplados dentro de la plataforma. En conclusión, los costos de mantenimiento de la plataforma serían bajos.

Finalmente, en la pregunta 8 se plantea algo muy importante en DevOps. Borrar o deshacer un despliegue es esencial cuando determinada configuración de la infraestructura no funciona y es necesario hacer cambios y volver a desplegar. Para poder satisfacer este requerimiento, la plataforma lo soluciona mediante la creación de un *pipeline* de «Destrucción de Despliegue» al momento de configurar el proyecto. Este *pipeline* se puede activar manualmente desde el proveedor

de *Git* en donde se está guardando el repositorio del proyecto.

7.2. Extensión de la herramienta: Implementación de un despliegue de Kubernetes

Uno de los requerimientos fundamentales de la plataforma es que sea extensible, para que esta se mantenga actualizada frente a los avances y cambios de la industria tecnológica. Este punto fue un factor clave en el diseño de la arquitectura, mencionado en secciones anteriores.

Cuando se necesitan agregar nuevos componentes, nuevas versiones de los mismos, nuevos *pipelines* o realizar despliegues en nuevas tecnologías, estos se pueden realizar desde la interfaz de usuario, modificando los componentes y sus conectores. Sin embargo, cuando se quiere agregar una tecnología que actúa sobre múltiples componentes en simultáneo, se necesita extender la plataforma mediante la creación de un nuevo ensamblador y posiblemente un nuevo tipo de conector.

Durante la presentación intermedia, surgió la pregunta sobre el despliegue en Kubernetes que es una tecnología que ha ganado mucha popularidad en los últimos años, especialmente en el sector de DevOps por su importancia. La misma permite muchas características deseables en el ciclo de vida de una aplicación tales como escalabilidad, resiliencia, administración automatizada de contenedores, despliegues continuos y balanceo de carga.

La idea fue tomada en consideración, pero en una primera instancia se descartó, dado que el equipo no contaba con experiencia en dicha tecnología, teniendo en cuenta además el avance y los tiempos del proyecto. Pero dada la importancia de Kubernetes y como el proyecto se encontraba en una etapa en la que ya se contaba con un producto funcional, esta implementación serviría como validación de la arquitectura modular extensible. Por lo que se tomó la decisión de implementar Kubernetes en la plataforma.

Para lograr implementar despliegues en Kubernetes se necesita, por un lado, poder crear de forma automática clusters de Kubernetes en los proveedores de *cloud* utilizados, se necesita desplegar los componentes dentro de contenedores Docker y además poder crear los Docker registries donde estos contenedores serán publicados. Se necesitará también una serie de manifiestos que definen la arquitectura del sistema en Kubernetes, referencias a las imágenes de docker, conexiones entre componentes dentro del cluster, exposición de servicios a internet, entre otras configuraciones. Todo esto integrado con *pipelines* de CI/CD que permitan la integración continua y despliegues automáticos, y por último mantener el flujo GitOps en el manejo de la infraestructura.

7.2.1. Proceso de extensión

Dado que el equipo no contaba con experiencia en esta tecnología, primero fue necesario un estudio de la misma y la familiarización en la práctica, creando algunos despliegues manuales de prueba. Luego, se realizó la extensión en la plataforma, tomando un total de 33 horas aproximadamente (11 horas por integrante del equipo).

Para crear los manifiestos que definen la infraestructura en Kubernetes, fue necesario crear un nuevo ensamblador, junto con un nuevo conector, que define algunas configuraciones para cada componente, principalmente relacionadas a la cantidad de réplicas y escalabilidad. Este ensamblador, al igual que el resto, toma el proyecto como entrada y su salida resulta en una serie de archivos (manifiestos de Kubernetes para definir la infraestructura). En dichos archivos se definen los recursos de *deployment*, *service*, *autoscaling*, y el *ingress* en caso de ser necesario. Esta serie de manifiestos queda definida por los componentes que forman el proyecto, las conexiones entre ellos y sus conectores[1].

Dado el caso especial de Kubernetes, en el que es necesario no solo cambiar el tipo de despliegue sino también la infraestructura, fue necesario modificar los ensambladores encargados de crear la infraestructura en los proveedores, agregándoles un método nuevo. Este método se encarga de crear un cluster y un registry en el proveedor de *cloud* correspondiente.

Aprovechando el ensamblador de Docker y el hecho de que los componentes ya están definidos dentro de contenedores, se adapta el ensamblador de *pipelines* para que, en caso de tratarse de un despliegue sobre Kubernetes, genere automáticamente un pipeline adecuado. Este pipeline se encarga de realizar la construcción del contenedor para cada componente, subirlo al registry, asignarle el tag correspondiente y actualizar los nodos del clúster para desplegar la nueva versión.

7.3. Presentación de feedback

Ya concluida la implementación del proyecto, se organizó una nueva presentación con docentes de LINS y personas interesadas en la propuesta. Nuevamente consistió de una instancia oral para hablar de la motivación, propósito y conceptos de la solución, seguido de una demostración de un flujo completo de despliegue utilizando Bill y concluyendo con una instancia de preguntas y

[1] En esta versión, se decide dejar por fuera del cluster los volúmenes, no pudiendo desplegar bases de datos o memorias cache dentro del cluster, aunque estas se pueden desplegar sin problema en el mismo proveedor

comentarios.

Finalizadas las preguntas, se le compartió a los participantes una encuesta para evaluar la presentación. La encuesta realizada se encuentra en el Apéndice C.1. Un total de once personas completaron la encuesta. A continuación algunos de los resultados más relevantes.

Para empezar, se logra ver que la presentación fue bien recibida en términos de claridad, ya que el 100 % de los participantes la calificó como «Clara» o «Muy clara», lo que indica que el contenido fue accesible y comprensible para el público.

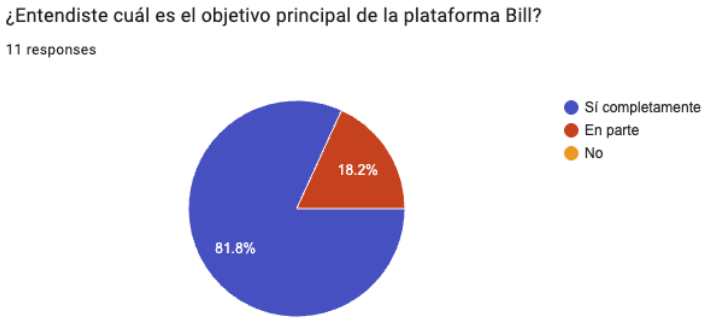

Figura 7.1: Facilidad de comprensión

En cuanto a los aspectos de la solución, los encuestados destacaron especialmente la interfaz de usuario y su velocidad para desplegar proyectos, así como también la posibilidad de realizar despliegues en varias nubes.

Al consultar sobre los contextos útiles para la aplicación, se marcó la utilidad principalmente en contextos educativos y profesionales.

En el ámbito académico, se destacó en particular la posibilidad de poder acercar a los estudiantes a una infraestructura más completa e implementar prácticas DevOps mediante automatizaciones, introducción a la infraestructura como código y monitoreo. Respecto al ámbito profesional, se mencionó el beneficio de estandarizar patrones de código, configuraciones (ej.: de *pipelines* y de infraestructura) y herramientas de desarrollo (lenguajes de programación y librerías).

En cuanto al grado de innovación, la mayoría de los participantes calificaron la propuesta con valores de entre 3 y 5 en una escala de 1 a 5, indicando que la solución fue percibida como innovadora.

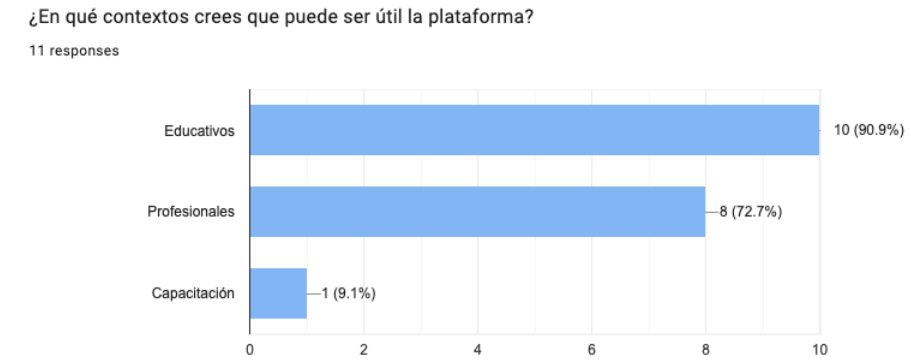

Figura 7.2: Contexto de uso

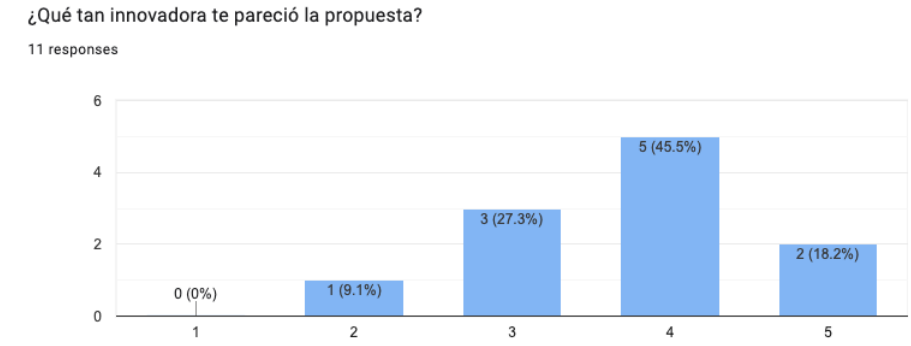

Figura 7.3: Innovación de propuesta

Finalmente, las respuestas abiertas de la encuesta reflejaron un alto nivel de interés y valoración positiva hacia la propuesta. Varios participantes la destacaron como un trabajo «Muy bueno» y «De gran utilidad». Al mismo tiempo surgieron sugerencias concretas para trabajo futuro, como la gestión de múltiples ambientes de desarrollo, aspectos de seguridad a tener en cuenta e integración con inteligencia artificial.

Capítulo 8

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones generales del proyecto, evaluando los resultados obtenidos en función de los objetivos propuestos. También, se hará una reflexión de la plataforma desarrollada, sus limitaciones actuales y las oportunidades a mejora. Por otro lado, se propondrán posibles líneas de trabajo a futuro.

8.1. Principales Conclusiones del Proyecto

El proyecto logró cumplir con los objetivos planteados en un inicio, a raíz de esto, se tuvo como resultado la implementación de una plataforma completamente funcional que permite la creación de múltiples soluciones DevOps de forma dinámica, adaptándose a las necesidades de los proyectos de los usuarios.

A lo largo del desarrollo, fue posible comprobar la posibilidad de abstraer las complejidades propias de las herramientas DevOps. Esto permitió crear una arquitectura que ha demostrado ser modular y extensible. Sumándole una implementación de interfaz gráfica, habilita a usuarios con diferentes niveles de conocimiento, utilizar la plataforma de manera sencilla y sin perder flexibilidad ni capacidad de personalización.

Finalmente, este proyecto da como resultado a Bill, una plataforma generadora de soluciones DevOps automatizadas, manifestando que es posible reducir la barrera de entrada a estas prácticas. Si bien existen aspectos a mejorar, se cuenta ahora con una plataforma base, lo que abre las puertas a nuevas funcionalidades, casos de uso y oportunidades de mejora.

8.2. Limitaciones de la solución Actual

A pesar de los logros alcanzados, la solución planteada presenta algunas limitaciones.

Entre ellas, se encuentra la imposibilidad de probarla en la totalidad de escenarios que se pueden plantear. Dado el alcance del proyecto y la gran cantidad de situaciones que se encuentran tanto en el ámbito académico como empresarial, puede que haya proyectos de los usuarios que nuestra solución no pueda soportar tal como está. Esto se podría mitigar realizando una validación más extensiva con múltiples usuarios y casos de usos.

Por otro lado, se tienen limitaciones con tecnologías que no se llegaron a implementar en la plataforma y que aún así son utilizadas frecuentemente. Algunas de ellas se mencionarán en la Sección 8.4 sobre el Trabajo a Futuro.

Finalmente, en lo que respecta a la extensibilidad, se pueden presentar limitaciones. Esto se debe a que es necesario de conocimiento técnico, tanto de las herramientas DevOps como de la solución, para poder desarrollar o agregar nuevas tecnologías.

8.3. Lecciones Aprendidas

A lo largo del proyecto se adquirieron múltiples aprendizajes técnicos y metodológicos. Por un lado, a nivel técnico, se profundizó en el diseño de arquitecturas, así como en la configuración de múltiples herramientas DevOps. Yendo más allá de estas configuraciones, se logró automatizar la creación de las mismas, logrando otro grado de entendimiento. También, se aprendió sobre los desafíos que conllevan desarrollar una experiencia de usuario simplificada sin perder funcionalidades.

En cuanto a la metodología del proceso, se destaca el aprendizaje de validar la solución de forma temprana como lo fue en la presentación intermedia. Esa retroalimentación de potenciales usuarios reales, ayudó a tomar decisiones que aporten valor y ajustar el rumbo del proyecto. Por otro lado, resultó clave mantener un enfoque incremental del desarrollo, dividiendo y refinando tareas para completar los objetivos en el tiempo estimado, lo que concluyó en la definición de un alcance apropiado del prototipo implementado.

8.4. Trabajo a Futuro

En lo que respecta al trabajo a futuro, se puede dividir en dos principales lineamientos que se pueden seguir: a nivel de solución y a nivel de implementación.

A nivel de solución, una de las principales funcionalidades que quedó pendiente fue la de editar proyectos desde la misma plataforma. Esto trae diferentes complejidades, ya que es necesario mantener un estado de los proyectos actualizados, ya sea si se edita manualmente desde el repositorio o si se hace desde la misma plataforma. Esto extendería el caso de uso de la solución planteada para que sea aplicada a lo largo de toda la vida del proyecto, y no limitarse solamente a utilizarla al inicio del mismo.

A nivel de implementación, la plataforma desarrollada podría extenderse a más tecnologías para aumentar su utilidad. Un ejemplo de estas tecnologías, serían nuevos proveedores de *cloud*, ya que actualmente se limita a soportar MiNube y AWS. Otro ejemplo sería el desarrollo de un *log in* de usuario utilizando GitHub, ya que es ampliamente empleado por desarrolladores. Por otro lado, sería útil realizar una validación más extensa con usuarios reales, e implementar las correcciones necesarias para mejorar a nivel de interfaz de usuario y arquitectura.

Referencias

Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.

Dora — get better at getting better. https://dora.dev/. Accessed: 2025-01-01.

Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 1st edition, 2018.

2024 accelerate state of devops report, 2024. Available online. A decade of research into DevOps capabilities and outcomes.

J. A. V. M. K. Jayakody and W.M.J.I. Wijayanayake. Challenges for adopting devops in information technology projects. In *2021 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, volume 4, pages 203–210, 2021.

Juliano Marcos Martins. The technological evolution at mercado libre: from the monolith to the multicloud platform, 2024.

Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

¿qué es gitops? https://about.gitlab.com/es/topics/gitops/. Accessed: 2025-01-01.

Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77, 2019.

K. Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016.

Adrian Mouat. *Using Docker, developing and deploying software with containers*. 2016.

What is multicloud? — ibm. https://www.ibm.com/think/topics/multicloud. Accessed: 2025-01-01.

Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, USA, 2011.

Git. https://git-scm.com/. Accessed: 2025-01-01.

Ionut-Catalin Donca, Ovidiu Petru Stan, Marius Misaros, Dan Gota, and Liviu Miclea. Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors*, 22(12):4637, 2022.

Chandrasekar Vuppalapati, Anitha Ilapakurti, Karthik Chillara, Sharat Kedari, and Vanaja Mamidi. Automating tiny ml intelligent sensors devops using microsoft azure. In *2020 ieee international conference on big data (big data)*, pages 2375–2384. IEEE, 2020.

Holger Karl, Dennis Kundisch, Friedhelm Meyer auf der Heide, and Heike Wehrheim. A case for a new it ecosystem: On-the-fly computing. *Business & Information Systems Engineering*, 62:467–481, 2020.

Laurens Sion, Dimitri Van Landuyt, Koen Yskout, Stef Verreydt, and Wouter Joosen. Automated threat analysis and management in a continuous integration pipeline. In *2021 IEEE Secure Development Conference (SecDev)*, pages 30–37. IEEE, 2021.

Multiexperience development platforms (mxdp). https://www.gartner.com/en/information-technology/glossary/multiexperience-development-platforms-mxdp. Accessed: 2025-01-01.

Outsystems official website. https://www.outsystems.com/. Accessed: 2025-01-01.

Salesforce lightning platform. https://www.salesforce.com/eu/products/platform/lightning/. Accessed: 2025-01-01.

Firebase by google. https://firebase.google.com/. Accessed: 2025-01-01.

Kinvey developer center - core overview. https://devcenter.kinvey.com/rest/guides/core-overview. Accessed: 2025-01-01.

Microsoft power apps. https://www.microsoft.com/en-us/power-platform/products/power-apps. Accessed: 2025-01-01.

Genexus - sitio oficial. https://www.genexus.com/es/. Accessed: 2025-01-01.

Aneta Poniszewska-Marańda and Ewa Czechowska. Kubernetes cluster for automating software production environment. *Sensors*, 21(5):1910, 2021.

Kasim Oztoprak, Yusuf Kursat Tuncel, and Ismail Butun. Technological transformation of telco operators towards seamless iot edge-cloud continuum. *Sensors*, 23(2):1004, 2023.

Ravi Shankar Jha, Priti Ranjan Sahoo, and Smrutirekha. Relevance of disruptive technologies led knowledge management system and practices for msme. In *ICT Systems and Sustainability: Proceedings of ICT4SD 2021, Volume 1*, pages 139–147. Springer, 2022.

Grigori Fursin. Collective knowledge: organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A*, 379(2197):20200211, 2021.

Apendice A

La siguiente tabla fue creada en el marco de un Módulo Taller en la Facultad de Ingeniería por el alumno Santiago Cuadrado.

Categoría de Fuente 1 [19]	Categoría de Fuente 2 [16]	Ejemplos [Menciones en fuentes]	Actores	Objetivos	Conceptos
Intercambio de Conocimiento	Colaboración	Trello [S1] Taiga [S5] GitHub Projects [S5] Jira [S5] Notion [S5]	Todos	Colaboración humana	Cultura de la colaboración Intercambio de conocimiento Rompiendo silos Colaboración entre departamentos
Gestión de Código Fuente	Gestión de Código Fuente	GitLab [S3,S4] GitHub [S2,S4] BitBucket [S2,S4]	Dev/Ops	Colaboración humana Entrega continua	Versionado Cultura de la colaboración Intercambio de conocimiento Rompiendo silos Colaboración entre departamentos
Build Process	Build Contenerización Testing	Gradle [S1] Maven [S1] JUnit [S1,S3] SonarQube[S1,S3] Docker [S1,S3,S4] Dagger [S5]	Dev	Entrega continua	Release engineering Entrega continua Automatización Automatización del Testing, Exactitud Análisis estático
Integración Continua	Integración Continua Testing	GitLab CI [S1,S3,S4] Jenkins [S1,S2,S3,S4] GitHub Actions [S4]	Dev/Ops	Entrega continua	Proceso de liberación frecuente y confiable Release engineering Integración continua Pipeline de despliegue Entrega continua, Automatización Gestión de artefactos
Automatización de Despliegue	Despliegue Continuo Configuración y aprovisionamiento IaaS/PaaS Contenerización Gestión de Bases de Datos	Docker [S1,S3,S4] Kubernetes [S3,S4] Terraform [S3,S4] Ansible [S2,S3,S4] Helm [S5] Chef [S1,S4] Puppet [S1,S2,S4]	Dev/Ops	Entrega continua Fiabilidad	Proceso de liberación frecuente y confiable Release engineering Gestión de la configuración Entrega continua Infraestructura como código Virtualización, Contenerización Cloud services, Automatización
Monitoreo & Logging	Logging Seguridad Monitoreo	Grafana [S3,S4] Prometheus [S1,S4] Zabbix [S1,S4] DataDog [S3,S4]	Ops/Dev	Fiabilidad	Lo construyes, lo ejecutas Soporte fuera de horario para desarrolladores Monitoreo continuo del tiempo de ejecución Rendimiento, disponibilidad, escalabilidad Resiliencia, confiabilidad, automatización Métricas, alertas, experimentos Gestión de logs, Seguridad

Tabla A.1: Tabla de categorías consolidada

A.1. Búsqueda en Scopus

Las *keywords* utilizadas fueron «DevOps AND Automation AND Platform». Para filtrar la búsqueda se tomaron los siguientes parámetros:

- Documentos con año de publicación mayor o igual que 2020.
- Contenidos en las áreas de "Computer Science" y "Engineering".
- Tipos de documentos: "Conference paper" y "Article".
- Idioma Inglés.
- El documento ha sido citado 10 o más veces por otros documentos.

Dicha configuración, dio como resultado los siguientes 8 documentos (el orden en la lista se encuentra desde el más citado al menos):

1. Kubernetes cluster for automating software production environment [26]
2. Technological Transformation of Telco Operators towards Seamless IoT Edge-Cloud Continuum [27]
3. Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects [15]
4. Automating Tiny ML Intelligent Sensors DevOps Using Microsoft Azure [16]
5. Relevance of Disruptive Technologies Led Knowledge Management System and Practices for MSME [28]
6. A Case for a New IT Ecosystem: On-The-Fly Computing [17]
7. Collective knowledge: Organizing research projects as a database of reusable components and portable workflows with common interfaces [29]
8. Automated Threat Analysis and Management in a Continuous Integration Pipeline [18]

A.2. Análisis de Artículos Relacionados

Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects

Este artículo trata sobre una solución automatizada para generar y configurar de *pipelines* CI/CD en proyectos ágiles, la misma está basada en scripts en Bash y utiliza herramientas como Docker, Kubernetes, Helm y *GitLab* CI/CD. El documento explica cómo se construye, prueba y despliega software de manera eficiente, utilizando técnicas como el caché de capas de Docker para optimizar tiempos y costos, y la ejecución en instancias económicas de AWS para reducir

gastos de infraestructura.

Esta solución que se presenta tiene relación con este proyecto, ya que se comparte el objetivo de simplificar la configuración y mantenimiento de herramientas DevOps, en este caso *pipelines* CI/CD, mediante la automatización.

Algunas ideas destacadas que podrían tomarse en cuenta para nuestro proyecto son: la implementación de un caché de capas de Docker para acelerar los tiempos de construcción y la ejecución de *pipelines* en instancias económicas para optimizar costos. Aunque esta última recae en un caso específico de utilizar a AWS como proveedor de *cloud*.

En conclusión, este artículo brinda conceptos y técnicas prácticas que pueden servir como inspiración para el desarrollo de la plataforma en la que se basa este proyecto. También en dicho documento se tiene enfoque en la automatización y la eficiencia se conectan estrechamente con nuestros objetivos. Sin embargo, hay que tener en cuenta que la finalidad de la solución de este documento es la automatización de la creación de *pipelines*, mientras que nuestra solución busca ir un poco más allá y facilitar la configuración de otras herramientas y/o prácticas.

Automated Threat Analysis and Management in a Continuous Integration Pipeline

En este artículo se propone el prototipo Continuous Threat Analysis and Management (CTAM). Dicho CTAM es una herramienta que puede ser integrada en *pipelines* de integración continua con la finalidad de automatizar el análisis, la gestión y el monitoreo de amenazas de seguridad durante el desarrollo de software. Este prototipo utiliza modelos que se basan en el flujo de los datos, para esto se utilizan diagramas y luego se almacenan junto al código fuente del proyecto. Cuando se realizan cambios en el repositorio, automáticamente se ejecutan trabajos en el *pipeline* CI que generan modelos de amenazas, evalúan riesgos y finalmente son mostrados en un panel.

La solución presentada se alinea con los objetivos de este proyecto, ya que comparte el enfoque en la automatización dentro de los *pipelines* CI/CD y la integración de herramientas avanzadas. Además, su metodología para mantener modelos actualizados y realizar análisis iterativos puede servir como inspiración para implementar funcionalidades similares en la plataforma propuesta. Un aspecto particularmente valioso es la capacidad de CTAM de ofrecer retroalimentación en cada commit, promoviendo decisiones basadas en datos y facilitando el seguimiento del progreso en la mitigación de amenazas.

Entre las ideas que podrían adoptarse se destacan:

- **Mantenimiento de modelos arquitectónicos**: Incorporar modelos DFD

junto al código fuente permite un análisis constante y actualizado de la seguridad.

- **Paneles interactivos de monitoreo**: Proporcionar a los desarrolladores una visión clara del impacto de sus cambios en términos de seguridad y privacidad.
- **Evaluación de riesgos por commit**: Implementar una herramienta que calcule y reporte riesgos inherentes y residuales después de cada cambio en el código.

En conclusión, este artículo refuerza la importancia de incorporar prácticas de análisis continuo de seguridad en los *pipelines* CI/CD. La solución CTAM no solo destaca por su enfoque innovador en la gestión de amenazas, sino también por su capacidad de evolucionar junto con el desarrollo del sistema. Su implementación en este proyecto podría elevar significativamente la robustez y confiabilidad de la plataforma, fortaleciendo su propuesta de valor.

Automating Tiny ML Intelligent Sensors DevOps Using Microsoft Azure

Este artículo presenta un marco de trabajo que utiliza Microsoft Azure DevOps para automatizar el desarrollo y despliegue de sensores inteligentes Tiny ML, diseñados para ayudar a pequeños agricultores en la industria lechera. Se destaca el uso de Azure DevOps como una plataforma robusta y versátil para construir, probar y desplegar aplicaciones nativas y no nativas de la nube, incorporando técnicas como la infraestructura como código (IaC) y la integración continua (CI) y entrega continua (CD). La propuesta explora cómo la automatización, combinada con Azure DevOps, puede reducir los costos operativos y hacer accesible la inteligencia artificial (IA) en sectores rurales sub atendidos, fomentando un futuro más sostenible.

Aunque este artículo aborda un caso de uso específico, su relación con este proyecto radica en su enfoque en la automatización y la integración de tecnologías modernas para simplificar tareas complejas. Azure DevOps, con sus principios de CI/CD e IaC, proporciona un modelo útil para estructurar los procesos en una plataforma DevOps, ofreciendo consistencia, escalabilidad y eficiencia. Además, la capacidad de gestionar herramientas mediante *pipelines* automatizados resuena con los objetivos de este proyecto, que busca optimizar la configuración de herramientas DevOps.

A Case for a New IT Ecosystem: On-The-Fly Computing

En el artículo «On-The-Fly Computing» se explora un enfoque en donde servicios informáticos pueden configurarse y ejecutarse de manera automática,

lo que permite componer soluciones personalizadas a partir de componentes disponibles en el mercado digital. Tiene como objetivo reducir la complejidad del desarrollo de software ofreciendo una infraestructura donde los usuarios puedan seleccionar y combinar servicios según sus necesidades. A diferencia de este enfoque general, nuestra solución se centra específicamente en la automatización de herramientas DevOps para la gestión de infraestructura y despliegue de software, proporcionando una solución pre configurada que optimiza el ciclo de vida del desarrollo sin depender de un catálogo (el mercado digital que se menciona anteriormente) de proveedores de servicio como lo plantea «On-The-Fly Computing».

Entre las ideas destacadas que podrían adoptarse se encuentran:

- **Automatización a gran escala**: Usar un enfoque basado en IaC para estandarizar y automatizar la configuración de entornos en múltiples plataformas.
- **Integración de monitoreo continuo**: Implementar sistemas para monitorear la eficacia y el rendimiento de los *pipelines* en tiempo real.
- **Sostenibilidad operativa**: Incorporar prácticas que reduzcan costos operativos, haciéndolas accesibles incluso para organizaciones con recursos limitados.

En conclusión, este artículo subraya la relevancia de las tecnologías de automatización y su capacidad para democratizar el acceso a soluciones tecnológicas avanzadas. La implementación de principios y técnicas descritas en esta investigación podría enriquecer significativamente el diseño de la plataforma propuesta, alineándose con sus objetivos de eficiencia y accesibilidad.

Apendice B

B.1. Ejemplo de Pipeline CI/CD utilizado para desplegar un entorno WildFly con Cloud Scripting

```
stages:
  - pre_deploy
  - apply_manifest
  - post_deploy
test-project_apply_manifest:
  stage: apply_manifest
  image: alpine
  script:
    - echo "Iniciando despliegue..."
    - apk add --no-cache jq curl
    - export ENCODED_JSP=$(jq -sRr @uri < "test-project.manifest.jps")
    - 'curl -X POST "https://app.elasticloud.uy/1.0/marketplace/jps/rest/
    install"
      -G -d "session=$SESSION_KEY" -d "shortdomain=$ENV_NAME" -d "jps=$
    ENCODED_JSP" '
java_ee_add_repo:
  image: alpine
  stage: post_deploy
  before_script:
    - apk add --no-cache curl jq
  script:
    - NODE_GROUP="cp"
    - CONTEXT="ROOT"
    - |
      # Crear un archivo temporal con el script
      cat > post_deploy_script.sh << 'EOSCRIPT'
      #!/bin/bash
      BASE_DIR="/var/lib/jelastic/PROJECTS"
      for env_dir in "$BASE_DIR"/*; do
        if [ -d "$env_dir" ]; then
```

```
        echo "Procesando: $env_dir"
        WEB_TARGET="$env_dir/java_ee/web/target"
        EJB_TARGET="$env_dir/java_ee/ejb/target"
        EAR_TARGET="$env_dir/java_ee/ear/target"
        # Eliminar directorios target (si existen)
        rm -rf "$WEB_TARGET" || echo "No se pudo eliminar $WEB_TARGET"
        rm -rf "$EJB_TARGET" || echo "No se pudo eliminar $EJB_TARGET"
        # Copiar contenido del EAR a WEB y EJB
        if [ -d "$EAR_TARGET" ]; then
          cp -R "$EAR_TARGET/" "$WEB_TARGET/" || echo "Fallo al copiar
    a $WEB_TARGET"
          cp -R "$EAR_TARGET/" "$EJB_TARGET/" || echo "Fallo al copiar
    a $EJB_TARGET"
        else
          echo "No se encontr EAR target en $env_dir"
        fi
      fi
    done
    EOSCRIPT
  - |
    # Crear el JSON del hook y codificarlo para URL
    HOOKS_JSON=$(jq -n --arg script "$(cat post_deploy_script.sh)" '{
    postBuild: $script}')
    ENCODED_HOOKS=$(echo "$HOOKS_JSON" | jq -sRr @uri)
  - |
    # Preparar el REPO como JSON y codificarlo para URL
    ENCODED_REPO=$(echo "$REPO" | jq -sRr @uri)
  - |
    # Preparar los SETTINGS como JSON y codificarlo para URL
    SETTINGS_JSON='{"autoResolveConflict":true,"autoUpdate":false,"zdt
    ":false,"workDir":"java_ee/"}'
    ENCODED_SETTINGS=$(echo "$SETTINGS_JSON" | jq -sRr @uri)
  - |
    # Construir la URL correctamente
    DEPLOY_URL="https://app.elasticloud.uy/1.0/environment/deployment/
    rest/deploy?envName=${ENV_NAME}&repo=${ENCODED_REPO}&hooks=${ENCODED
    _HOOKS}&context=${CONTEXT}&nodeGroup=${NODE_GROUP}&settings=$ENCODED
    _SETTINGS&session=${SESSION_KEY}"
  - curl -X POST "$DEPLOY_URL"
```

Apendice C

C.1. Cuestionario realizado luego de la demo de la plataforma en el seminario del LINS.

A continuación se presentaran las preguntas que conformaban el cuestionario que completaron las personas que asistieron al seminario del LINS.

1. **¿Qué tan clara te resultó la presentación de la plataforma?**
   Opciones: Muy clara, Clara, Neutral, Confusa, Muy confusa
2. **¿Entendiste cuál es el objetivo principal de la plataforma Bill?**
   Opciones: Sí completamente, En parte, No
3. **¿Qué parte del funcionamiento te resultó más interesante?**
   Respuesta abierta
4. **¿En qué contextos crees que puede ser útil la plataforma?**
   Opciones: Educativos, Profesionales, Otro
5. **¿Qué posibles usos le ves a la plataforma dentro de tu ámbito académico o profesional?**
   Respuesta abierta
6. **¿Qué tan innovadora te pareció la propuesta?**
   Escala del 1 al 5
   1 = Nada innovadora
   5 = Muy innovadora
7. **¿Qué mejorarías o qué te gustaría ver en versiones futuras?**
   Respuesta abierta
8. **¿Querés dejar algún comentario o duda adicional?**
   Respuesta abierta