

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

**UN GESTOR DE TAREAS BASADO EN KANBAN PARA LA
PLANIFICACIÓN OPERATIVA: UN ENFOQUE ÁGIL PARA LA
GESTIÓN Y ENTREGA CONTINUA DE UN PRODUCTO
SOFTWARE**

**Implementación de las prácticas de DevOps para automatizar el
desarrollo de una aplicación web**

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN
SOFTWARE**

GLENDER ISRAEL MIRANDA CHÁVEZ

glender.miranda@epn.edu.ec

DIRECTOR: JULIO CÉSAR SANDOBALÍN GUAMÁN

julio.sandobalin@epn.edu.ec

DMQ, julio 2024

CERTIFICACIONES

Yo, Glender Miranda declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Glender Miranda

Certifico que el presente trabajo de integración curricular fue desarrollado por Glender Miranda, bajo mi supervisión.

Julio Sandobalín, Ph.D.
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Glender Miranda

Julio Sandobalín, Ph.D.

DEDICATORIA

A mis queridos padres, cuyo apoyo incondicional ha sido el pilar de mi éxito académico y personal. A mis hermanas y hermano, a quienes amo profundamente y sin quienes mi vida no sería la misma. A Dios, por permitirme experimentar tanto los momentos felices como los desafiantes, que han sido fundamentales en mi crecimiento. Y, en general, a la vida misma, por regalarme la oportunidad de sentir y ser parte de algo tan significativo.

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a mis compañeros de equipo, Christian Pazmiño, Bryan Tapia y David León, con quienes tuve el privilegio de colaborar en la realización de este trabajo. Su apoyo y dedicación fueron esenciales para superar los desafíos que enfrentamos juntos.

Agradezco profundamente a mis tutores, Julio Sandobalín, Ph.D., y Carlos Iñiguez, Ph.D., quienes brindaron una orientación invaluable a lo largo de este proceso. Su compromiso y guía me han permitido aprender y crecer significativamente durante el desarrollo de este proyecto.

Finalmente, agradezco a los revisores de este escrito por su disposición a revisar el documento. Valoro profundamente su tiempo y cualquier comentario o sugerencia que puedan ofrecer para contribuir a la mejora y perfección de este trabajo.

ÍNDICE DE CONTENIDO

1.	DESCRIPCIÓN DEL COMPONENTE DESARROLLADO.....	2
1.1	Objetivo general	2
1.2	Objetivos específicos	2
1.3	Alcance	3
1.4	Marco teórico	4
1.4.1.	Enfoques Ágiles	4
1.4.1.1.	Scrum	4
1.4.1.1.1.	Eventos de Scrum	5
1.4.1.1.2.	Roles de Scrum	10
1.4.1.1.3.	Artefactos de Scrum	11
1.4.1.2.	DevOps.....	12
1.4.1.2.1.	Principios	13
1.4.1.2.2.	Prácticas.....	14
1.4.1.2.3.	Infraestructura como código	16
1.4.2.	Azure DevOps.....	17
1.4.2.1.	Azure Boards.....	17
1.4.2.2.	Azure Repos	20
1.4.2.3.	Azure Pipelines	22
1.4.3.	Tableros Kanban.....	24
2.	METODOLOGÍA.....	26
2.1.	Análisis de herramientas DevOps	26
2.2.	Configuración de recursos en Azure Portal.....	29
2.2.1.	Web App Services.....	29
2.2.2.	Bases de datos.....	35
2.3.	Configuración de Azure DevOps.....	39
2.3.1.	Organización y políticas de Azure DevOps.....	39
2.3.2.	Proyecto de Azure DevOps	41
2.4.	Verificación de la implementación de las prácticas y principios DevOps en el desarrollo de Optiplan	50
2.4.1.	Actividades de Scrum.....	50
2.4.1.1.	Planificación del producto	50
2.4.1.2.	Planificación del release	52

2.4.1.3.	Planificación del sprint.....	52
2.4.2.	Implementación	53
3.	RESULTADOS, CONCLUSIONES Y RECOMENDACIONES	76
3.1.	Resultados	76
3.2.	Conclusiones.....	82
3.3.	Recomendaciones.....	83
4.	REFERENCIAS BIBLIOGRÁFICAS	85

RESUMEN

Este trabajo se centra en la aplicación de prácticas y principios DevOps para mejorar el proceso de entrega de un producto software. El componente desarrollado está diseñado para apoyar a un equipo de desarrollo mediante la implementación del enfoque y herramientas DevOps.

La implementación del componente busca promover la comunicación y colaboración dentro del equipo de trabajo a través de la adopción de prácticas y principios DevOps, con el fin de optimizar el tiempo de entrega del software. Las prácticas implementadas incluyen integración continua, despliegue y entrega continuos, mientras que los principios clave abarcan la automatización, la paridad de entornos, la monitorización continua, el feedback y el cambio cultural.

El trabajo incluye la implementación de un encadenamiento de herramientas para soportar la integración, despliegue y entrega continua de software, así como la verificación de la aplicación de las prácticas y principios DevOps en el desarrollo del producto software.

KEYWORDS: DevOps, prácticas DevOps, principios DevOps, integración continua, despliegue continuo, entrega continua, Azure DevOps

ABSTRACT

This work focuses on the application of DevOps practices and principles to enhance the software product delivery process. The developed component is designed to support a development team by implementing DevOps approaches and tools.

The component implementation aims to promote communication and collaboration within the team through the adoption of DevOps practices and principles to optimize software delivery time. Implemented practices include continuous integration, continuous deployment, and continuous delivery, while key principles cover automation, environment parity, continuous monitoring, feedback, and cultural change.

The work includes the implementation of pipelines to support continuous integration, deployment, and delivery of software, as well as the verification of DevOps practices and principles in the software product development.

KEYWORDS: DevOps, DevOps practices, DevOps principles, continuous integration, continuous deployment, continuous delivery, Azure DevOps.

1. DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El componente implementa prácticas y principios DevOps que brinden soporte a un equipo de desarrollo. En este escenario, se implementa las prácticas en el desarrollo de un aplicativo web para la gestión operativa de tareas basado en Kanban.

Las prácticas y principios DevOps ayudan a optimizar el tiempo de entrega del producto software. Las prácticas incluyen la integración, despliegue y entrega continua de incrementos funcionales. Los principios incluyen la automatización, la paridad, la monitorización continua, el feedback y el cambio cultural.

Se utiliza Azure DevOps como una suite de herramientas que apoyan las prácticas de integración, despliegue y entrega continua. Además, Azure DevOps soporta el ciclo de vida del desarrollo de productos software [1].

Se han implementado políticas basadas en las prácticas y principios DevOps en los repositorios del proyecto para garantizar una gestión coherente del código fuente y los artefactos relacionados. Estas políticas incluyen restricciones de acceso, revisiones de código obligatorias, automatización de pruebas previas a la fusión y procedimientos definidos para la gestión de ramas.

Además, el componente implementa un encadenamiento de herramientas que apoya la integración, el despliegue y la entrega continua de un producto software. Esto garantizará una implementación continua y automatizada de los cambios en el ciclo de vida del software, mejorando así el tiempo de entrega y la calidad del proceso de desarrollo en su conjunto.

1.1 Objetivo general

Promover la comunicación y colaboración dentro de un equipo de trabajo mediante prácticas y principios DevOps para optimizar el tiempo de entrega de un producto software.

1.2 Objetivos específicos

1. Implementar un encadenamiento de herramientas que soporten la integración, despliegue y entrega continua de incrementos funcionales de software.
2. Verificar la implementación de las prácticas y principios DevOps en el desarrollo de un producto software.

1.3 Alcance

El componente se enfoca en la implementación de prácticas y principios DevOps para promover la comunicación y la colaboración en el equipo de desarrollo durante la construcción de un producto software. Este componente busca optimizar el tiempo de entrega del producto mediante la aplicación de prácticas DevOps en el proceso de desarrollo del software. Se realizaron pipelines de integración continua para la construcción del aplicativo, asegurando que el código se integre y se pruebe de manera frecuente y automática. Además, se implementaron releases para el despliegue y la entrega continua, facilitando la disponibilidad de nuevas versiones del software.

Para el principio de automatización, se configuraron pipelines con ejecuciones automáticas, evitando así el manejo manual de estas tareas y reduciendo la posibilidad de errores humanos. En cuanto al principio de paridad, se configuraron los ambientes de desarrollo y producción de manera idéntica, asegurando que siempre se ejecute el aplicativo web bajo las mismas condiciones, lo que facilita la identificación y resolución de problemas. Asimismo, se monitoreó el tiempo de despliegue del aplicativo y se registraron problemas y errores en el proceso de despliegue, permitiendo una rápida detección y resolución de incidentes, en línea con el principio de monitorización continua. Se consideró la retroalimentación continua, identificando y aprovechando las oportunidades de mejora en los procesos de integración y despliegue continuos, realizando sesiones de retroalimentación una vez identificadas las áreas de mejora.

La Infraestructura como Código (IaC) se utilizó para definir mediante scripts el aprovisionamiento de infraestructura que soportan las prácticas DevOps. Los scripts se almacenaron en los repositorios del proyecto, asegurando la disponibilidad y consistencia del código fuente y los artefactos relacionados.

Finalmente, no se tomó en cuenta la automatización de un plan de testing exhaustivo para las pruebas automatizadas en el producto software.

1.4 Marco teórico

1.4.1. Enfoques Ágiles

El término “ágil” se describe como “que actúa o se desarrolla con rapidez o prontitud”. Mientras que “agilidad” se define como la cualidad de ser “ágil” [2].

El agilismo en el desarrollo de software comienza en el 2001 con la publicación del manifiesto ágil. El manifiesto ágil es un documento creado en 2001 por diecisiete participantes del área de desarrollo de software. Cuenta con doce principios y cuatro valores para fomentar el desarrollo de software ágil [3].

El manifiesto ágil, en su definición, expresa:

“Estamos descubriendo mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo. A través de este trabajo hemos llegado a valorar:

- Individuos e interacciones **sobre** procesos y herramientas.
- Software funcionando **sobre** documentación exhaustiva.
- Colaboración con el cliente **sobre** negociación de contratos.
- Responder al cambio **sobre** seguir un plan.

Es decir, aunque hay valor en los elementos de la derecha, valoramos más los elementos de la izquierda [3].”

1.4.1.1. Scrum

Kenneth S. Rubin menciona que “Scrum es un enfoque ágil para desarrollar productos y servicios innovadores” [4].

Ken Schwaber y Jeff Sutherland definen a Scrum como “un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos” [5].

Bajo los conceptos antes mencionados podemos definir a Scrum como un enfoque ágil centrado en el desarrollo de productos y servicios innovadores. Se caracteriza por ser un marco de trabajo ligero que ayuda a personas, equipos y organizaciones a resolver problemas complejos mediante soluciones adaptativas, promoviendo la generación de valor continuo.

Scrum se diseñó intencionalmente como un marco incompleto que deja espacio para la creatividad; se describe con los elementos mínimos para ejercer la teoría Scrum. La

suposición es que el conocimiento general de las personas que lo usan lo completará. En lugar de instrucciones estrictas y concretas, se proporcionan reglas generales que gobiernan las relaciones entre las personas y sus interacciones [5].

El marco de trabajo Scrum se basa en tres pilares fundamentales: inspección, adaptación y transparencia.

Inspección

El proceso y los artefactos del marco Scrum deben ser revisados con frecuencia para evitar cualquier defecto indeseable [5].

“La inspección permite la adaptación. La inspección sin adaptación se considera inútil” [5].

Adaptación

A partir de los resultados obtenidos en la inspección, se procede a cambiar o mejorar cualquier proceso o material que no cumpla con los estándares o límites aceptables. Es crucial que los cambios se realicen a la brevedad posible para evitar consecuencias a futuro [5].

Transparencia

El proceso y los artefactos deben ser visibles tanto para el equipo Scrum como para los usuarios finales. Si los artefactos no se tratan de manera transparente, aumenta el riesgo y se reduce el valor en la entrega [5].

“La transparencia permite la inspección. La inspección sin transparencia es engañosa y derrochadora” [5].

1.4.1.1.1. Eventos de Scrum

Los eventos en Scrum son actividades que suceden dentro de la aplicación de Scrum. Cada evento es una oportunidad para mejorar el proceso o los artefactos. Dentro de Scrum se definen cinco eventos fundamentales: Sprint, Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective.

Sprint

Los Sprints son el pilar fundamental de Scrum. Este es un evento que se establece en una duración fija, aproximadamente de una semana hasta un mes [5], y cada uno de ellos debe ser de la misma duración. Cada nuevo sprint sigue a continuación del último [4]. En la Figura 1, se presenta un Sprint dentro del marco Scrum.

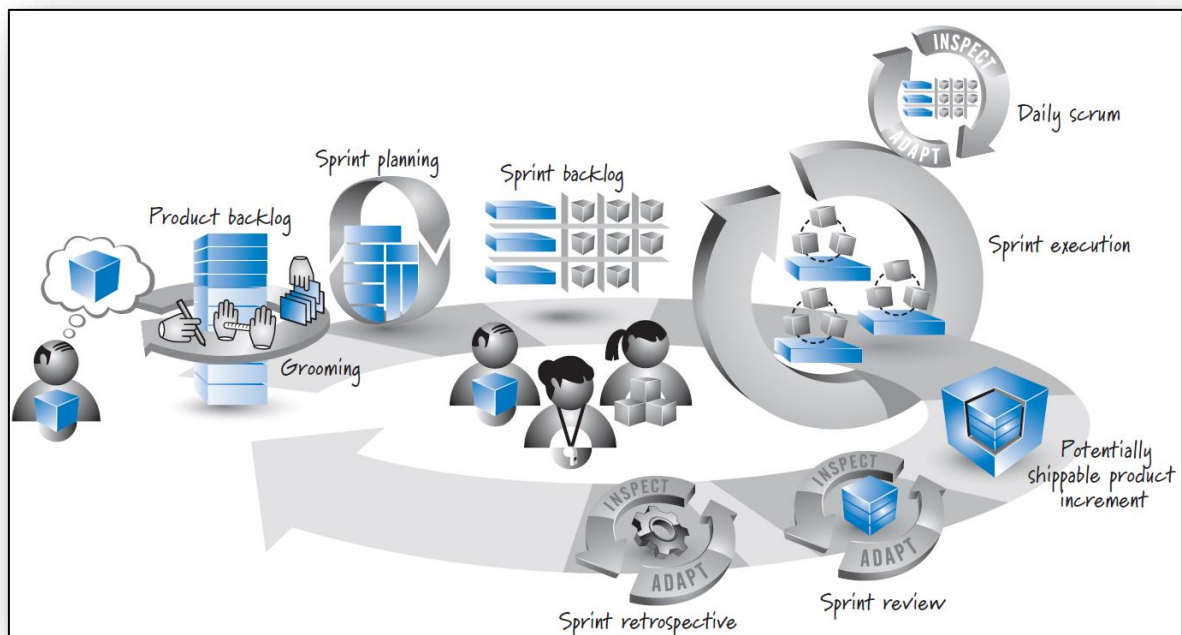


Figura 1. Scrum Framework [4]

Sprint Planning

Durante la planificación del Sprint (ver Figura 2), el equipo de desarrollo junto con el Product Owner se reúnen para definir el objetivo del Sprint. A partir de este objetivo, el equipo de desarrollo escoge las historias de usuario que sean concernientes a dicho objetivo. Las historias de usuario deben ser seleccionadas bajo el criterio de que deben ser completadas en el sprint. El equipo de desarrollo refina las historias de usuario en tareas de implementación para conocer el tiempo aproximado que tomará dicha historia de usuario, de esta manera se gana confianza y una mayor certeza de si una historia de usuario puede ser completada en un Sprint [4].

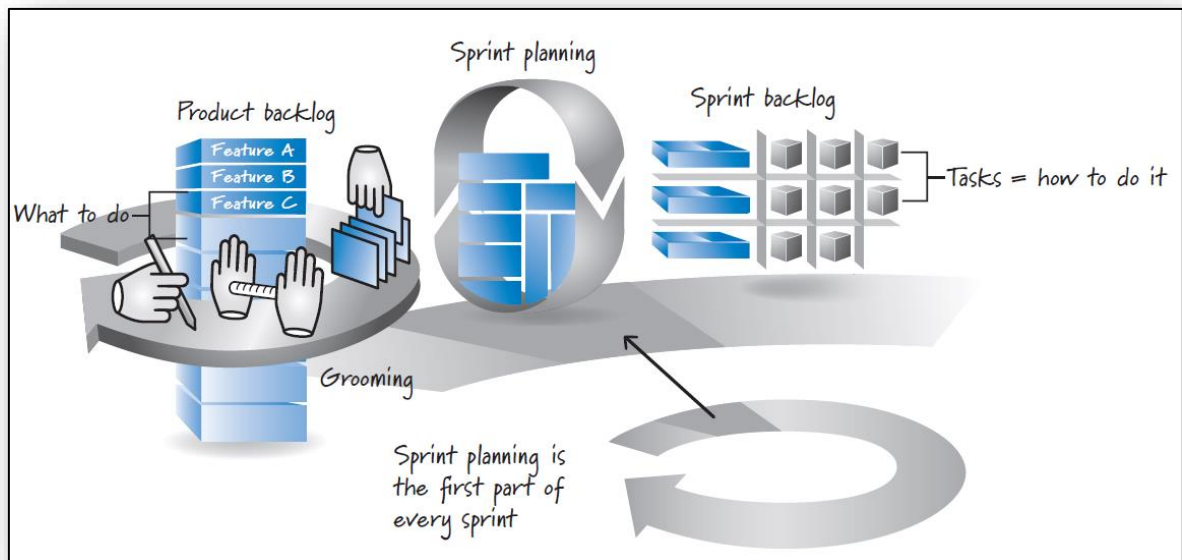


Figura 2. Sprint Planning [4]

Daily Scrum

El Daily Scrum es una actividad que la realiza el equipo de desarrollo, esta es una actividad de inspección donde cada integrante del equipo responde a las siguientes preguntas:

- ¿Qué se completado desde el último daily scrum?
- ¿Qué se planean realizar hasta el próximo daily scrum?
- ¿Existe algún impedimento u obstáculo que evita que avance?

En la Figura 3, se muestra de manera gráfica el proceso del Daily Scrum. La actividad de inspección no debería tomar más de 15 minutos. El Daily Scrum permite al equipo de desarrollo manejar de manera más flexible y efectiva el flujo de trabajo del sprint.

El Daily Scrum no es una actividad para resolver problemas. Después del Daily Scrum se procede a resolver todos estos problemas en la adaptación. El Daily Scrum es una actividad de inspección, sincronización y adaptación para ayudar al equipo de desarrollo a realizar el trabajo de mejor manera [4].

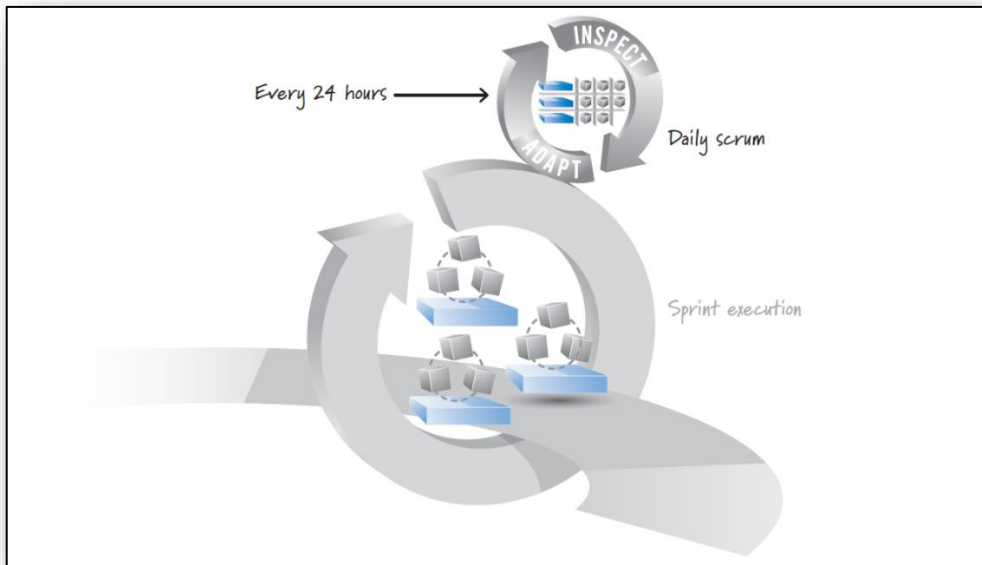


Figura 3. Ejecución del Sprint [4]

Sprint Review

Esta es una actividad de inspección y adaptación que revisa el incremento del producto software que se desarrolló en el sprint. En esta revisión participa el equipo de desarrollo, Product Owner, Scrum Master, Stakeholders, entre otros interesados.

En la Figura 4 se muestra el Sprint Review al final de la ejecución del sprint. Esta actividad permite a las entidades externas tengan una vista clara del producto construido, además de sincronizarse con el esfuerzo de desarrollo y ayudar a guiar el trabajo en la dirección deseada [4].

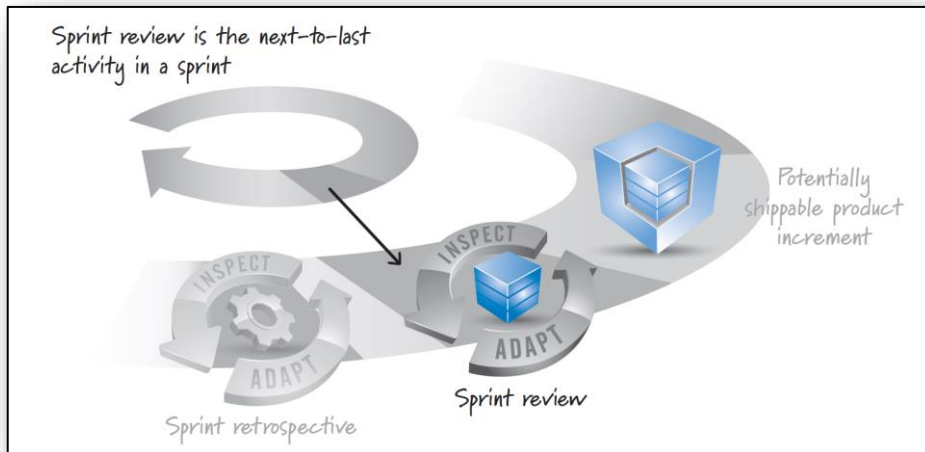


Figura 4. Sprint Review [4]

Sprint Retrospective.

Esta es una actividad de inspección y adaptación que se realiza al final del sprint, usualmente después del Sprint Review. Esta actividad permite la inspección y adaptación del proceso, la Figura 5 detalla el Sprint Review en el Framework de Scrum.

En el Sprint Retrospective, el Scrum Master, el equipo de desarrollo y el Product Owner revisan qué actividades se han realizado de manera adecuada y se deben continuar haciendo, y en qué actividades existen falencias y es necesario mejorar. Al final del Sprint Retrospective, todo el equipo de scrum debe tener identificados los puntos clave a mejorar para tomar acciones en el siguiente sprint [4].

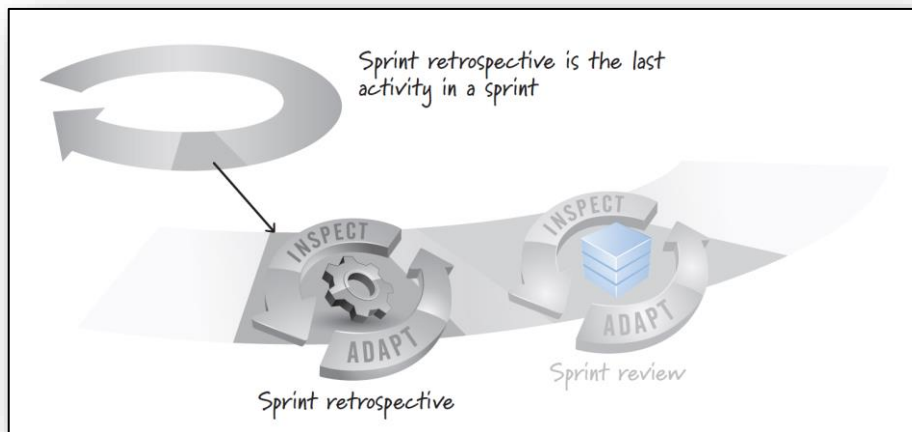


Figura 5. Sprint Retrospective [4]

1.4.1.1.2. Roles de Scrum

Un equipo de Scrum trabaja con tres roles principales:

Scrum Master

El Scrum Master es un facilitador, ayuda al equipo a resolver conflictos y capacita en las prácticas, principios y valores de Scrum. Es un líder que ayuda a los miembros del equipo a llegar a su rendimiento óptimo y enfocarlos en el marco de Scrum.

El Scrum Master elimina impedimentos, tanto internos como externos, que afectan la productividad del equipo. Este rol no se centra en controlar a los miembros del equipo ni hacer las veces de una autoridad de control como un Project Manager [4].

Product Owner

En la gestión de productos, el papel del Product Owner es crucial. Este rol tiene la responsabilidad de tomar decisiones en la construcción y evolución del producto. Su autoridad en el producto ayuda a decidir qué características (o requisitos) se implementarán y en qué orden. Una de sus tareas principales es comunicar y mantener una visión clara de los objetivos del producto software entre todos los participantes del proyecto.

El éxito en la gestión del producto recae en gran medida sobre el Product Owner. En este escenario, el Product Owner debe asegurarse de que se priorice y ejecute el trabajo que genere valor para el usuario final.

Scrum Developers

El equipo de desarrollo está formado de entre cinco y nueve integrantes y es un crisol de talentos diversos. No se trata de individuos aislados en sus respectivas especialidades, sino de un conjunto armónico de profesionales que fusionan sus habilidades para dar vida al producto.

La magia de este equipo radica en su autonomía. Los desarrolladores orquestan su trabajo y deciden cómo abordar los desafíos planteados por el Product Owner. Esta autogestión permite una dinámica fluida y adaptativa, esencial en el cambiante paisaje del desarrollo de software [4].

1.4.1.1.3. Artefactos de Scrum

Product Backlog

El Product Backlog es una lista priorizada y ordenada que contiene requisitos del usuario (es decir, historias de usuario), cambios de los requisitos existentes, mejoras a nivel técnico y resolución de errores.

El Product Owner junto con los Stakeholders se encargan de definir los elementos dentro del Product Backlog. Es fundamental asegurarse que los elementos están colocados en una secuencia correcta, donde los más prioritarios se encuentran en la parte más alta de la lista y van bajando de prioridad hasta la base.

La actividad de crear, refinar y priorizar los elementos del Product Backlog se denomina “preparación del Product Backlog”. Esta práctica es fundamental para mejorar la calidad del Product Backlog y minimizar el desperdicio, asegurando que los elementos estén bien definidos y alineados con las prioridades del proyecto [4].

Sprint Backlog

El Sprint Backlog es un plan para que el equipo de desarrollo comprenda cómo construir el incremento del producto software. Permite al equipo estimar y conocer el tiempo de trabajo a nivel de tareas que van a tener en el sprint.

El sprint backlog se compone del objetivo del sprint, las historias de usuario seleccionadas para el sprint actual y el plan, expresado en tareas, de cómo se van a entregar las nuevas características del producto software [5].

Incremento

Un incremento es un producto software funcional que contiene características que van aumentando a medida que se crean más incrementos. Cada incremento es un paso más al objetivo del producto final. Cada incremento se suma a los incrementos anteriores para que funcionen en conjunto como un producto software completo.

En el Sprint Review, el incremento es presentado a los interesados del producto. Para decidir si el incremento del producto software será presentado en el sprint review, el incremento debe satisfacer la Definición de Terminado (DoD, por sus siglas en inglés de Definition of Done). El DoD es crucial para garantizar que todos los miembros del equipo y las partes interesadas compartan una comprensión uniforme de los criterios que determinan si un incremento del producto software es potencialmente entregable. En el contexto de una revisión de Sprint, cualquier elemento que no cumpla con la Definición de Terminado no se puede presentar ni publicar.

1.4.1.2. DevOps

DevOps es un movimiento o paradigma con un conjunto de prácticas, principios y herramientas que permiten reducir el tiempo de entrega de un producto software y, al mismo tiempo, garantiza la calidad del producto software [6]. DevOps, del inglés "Development & Operations", unifica el desarrollo de software, las operaciones de TI y el aseguramiento de la calidad en un proceso integrado [7], [8].

DevOps fomenta la comunicación y colaboración continua entre desarrolladores y el equipo de operaciones, además, promueve la integración, despliegue y entrega continua de incrementos funcionales de productos software [7], [8].

Las principales ventajas de este paradigma son:

Automatización del proceso de producción de software: Los pipelines de integración continua automatizan el proceso de construcción de cada incremento de software, eliminando las tareas de integración manuales y repetitivas.

Retroalimentación temprana: Los interesados pueden evaluar el progreso del trabajo desde etapas tempranas y tomar decisiones informadas [7].

1.4.1.2.1. Principios

Los principios de DevOps son un conjunto de directrices fundamentales que facilitan la comunicación y colaboración entre los equipos de desarrollo y operaciones. Estos principios están diseñados para mejorar la eficiencia y la calidad en la entrega de software, fomentando una cultura de colaboración, automatización y mejora continua. Existe una variedad de conceptos diferentes presentados por distintos autores acerca de los principios DevOps. Sin embargo, este trabajo presenta los siguientes cinco principios de DevOps implementados en el proyecto:

Principio 1. Automatización

Automatizar los procesos de construcción, despliegue y pruebas del software para garantizar la calidad y una entrega temprana. La automatización evita la repetición de tareas y facilita una entrega continua, lo que permite obtener retroalimentación constante [9].

Principio 2. Paridad

La paridad especifica que todo el hardware en diferentes entornos debe configurarse de la misma manera. La paridad se vuelve crítica para los desarrolladores porque el mismo código puede proporcionar resultados diferentes si se intenta compilar y desplegar en diferentes entornos [10].

Principio 3. Monitorización

La monitorización implica la supervisión constante de los elementos de hardware para el correcto funcionamiento de la aplicación, el pipeline de entrega continua y toda la infraestructura asociada. Este principio es crucial en DevOps porque permite identificar áreas de mejora en el proceso de entrega del software, asegurando así un rendimiento óptimo y la detección temprana de problemas [11].

Principio 4. Feedback

Este principio indica que la retroalimentación ayuda a mejorar los procesos de integración, despliegue y entregas continuas. Es posible obtener feedback interno cuando se despliega código y se detecta problemas luego del despliegue. Puede existir un feedback externo cuando un usuario prueba el producto e informa sobre el nivel de satisfacción que ha

obtenido. Este proceso de retroalimentación es fundamental para mejorar el proceso de construcción y despliegue del producto software [12].

Principio 5. Cambio cultural

Una cultura DevOps se caracteriza por fomentar la comunicación, colaboración e integración de equipos o departamentos que tradicionalmente están separados. En lugar de trabajar con sistemas de tiques tradicionales, los equipos DevOps mantienen conversaciones continuas acerca del producto, abordando requisitos, características, cronogramas y recursos.

DevOps fomenta una cultura de comunicación y colaboración constante, promueve el aprendizaje a partir de los fallos y la cooperación entre departamentos y miembros de un mismo equipo. La comunicación es fluida a través de todo el equipo, utilizando herramientas y canales de colaboración, como Azure DevOps [11]. Además, la cultura DevOps fomenta el respeto y la confianza entre los miembros del equipo. La confianza permite que el equipo trabaje centrado en el mismo objetivo para desarrollar un producto de calidad.

1.4.1.2.2. Prácticas

Las prácticas de DevOps comprenden un conjunto de prácticas y técnicas que ayudan en el desarrollo y operación del software. Estas prácticas buscan fortalecer la colaboración entre los equipos y garantizar la entrega continua de software de calidad. Es importante mencionar que, al igual que los principios, varios autores definen diferentes prácticas para DevOps, pero el presente trabajo se basa en [13] para presentarlas. A continuación, se detallan las prácticas DevOps:

Practica 1. Integración continua

La integración continua, tal como lo define Martin Fowler:

“La Integración Continua es una práctica de desarrollo de software donde cada miembro de un equipo fusiona sus cambios en una base de código junto con los cambios de sus colegas al menos una vez al día. Cada una de estas integraciones es verificada por una compilación automatizada (incluyendo pruebas) para detectar errores de integración lo más rápido posible” [14].

Según lo mencionado por Fowler, dentro de un equipo de desarrollo, es fundamental realizar integraciones *diarias*. En contraste a desarrollar diferentes funcionalidades en

ramas distintas durante semanas, la integración continua promueve pequeñas y frecuentes fusiones de código hacia la rama de integración. Este proceso permite que sea más sencillo identificar los posibles errores o conflictos en el código, evitando pasar grandes cantidades de tiempo en solucionar problemas de integración que surgen debido a integraciones poco frecuentes [15].

La integración continua se realiza de manera automatizada, asegurando que cada integración esté libre de errores y no se suban cambios defectuosos al repositorio.

Practica 2. Despliegue continuo

Una vez que el incremento del producto software se construye, de manera tradicional, se transfiere al equipo de operaciones para su despliegue en los diferentes entornos, tanto de desarrollo como de producción. Inicialmente, los cambios más recientes se despliegan en el entorno de desarrollo para su revisión y aprobación. Posteriormente, se despliegan en el entorno de producción, donde el usuario final puede visualizar las nuevas características funcionales del aplicativo. En este escenario, el despliegue continuo consiste en enviar “automáticamente” el producto software a un entorno determinado para su funcionamiento.

El despliegue y la integración continuos requieren de un pipeline, o encadenamiento de herramientas. Cada pipeline permite la ejecución automatizada de herramientas para la construcción, prueba y despliegue del producto software.

Un pipeline de integración continua permite construir un incremento del producto software y dejarlo en un estado viable o “trabajable”. Entonces, el pipeline de despliegue continuo envía automáticamente el incremento del software a un entorno determinado para su puesta en funcionamiento.

Practica 3. Entrega continua

Similar al despliegue continuo, la entrega continua se refiere a la puesta en funcionamiento del producto software en un entorno determinado, pero de manera “manual”. De esta manera, las personas interesadas pueden visualizar el producto con los últimos cambios.

La entrega continua permite el lanzamiento continuo de incrementos funcionales del producto software, pasando por el proceso de integración continua y pruebas.

1.4.1.2.3. Infraestructura como código

La infraestructura como código, en inglés “Infrastructure as Code” (IaC), es un enfoque que permite la automatización de la configuración y aprovisionamiento de infraestructura basado en técnicas de desarrollo de software. Este enfoque se centra en la consistencia, rutinas repetitivas de aprovisionamiento y los posibles cambios en las configuraciones de un sistema [6].

Los principales beneficios de IaC [16] se listan a continuación:

- Permite aprovisionar infraestructura de TI de una manera más rápida, optimizando el tiempo de entrega de valor del producto.
- Permite reducir el tiempo y esfuerzo de realizar cambios en la infraestructura.
- Aumenta la disponibilidad de los recursos de infraestructura a los usuarios cuando sea necesario.
- Proporciona un conjunto de herramientas utilizables por los departamentos de desarrollo, operaciones u otros interesados.
- Permite crear recursos de infraestructura que sean seguros, confiables y de costo efectivo.
- Permite que los controles de cumplimiento normativo, gobernanza y seguridad sean visibles.
- Permite que los problemas presentados se puedan resolver de manera más rápida y efectiva.

1.4.2. Azure DevOps

Azure DevOps es un conjunto de herramientas que ofrece una amplia gama de servicios para equipos de desarrollo y operaciones, permitiendo gestionar de manera efectiva el ciclo de vida del software. Facilita la planificación, colaboración en el desarrollo de código, y la construcción y despliegue de productos de software. A continuación, se describen los principales servicios que ofrece Azure DevOps [7].

1.4.2.1. Azure Boards

Azure Boards permite a los equipos planificar y gestionar proyectos utilizando enfoques ágiles como Scrum y Kanban, tal como se muestra en la Figura 6. Entre sus funcionalidades se incluyen:

- Gestión del backlog de un proyecto.
- Administración de historias de usuario con diferentes niveles de abstracción.
- Visualización y gestión de sprints (o iteraciones), donde se pueden refinar las historias de usuario en tareas de implementación.
- Retrospectivas del sprint [7].

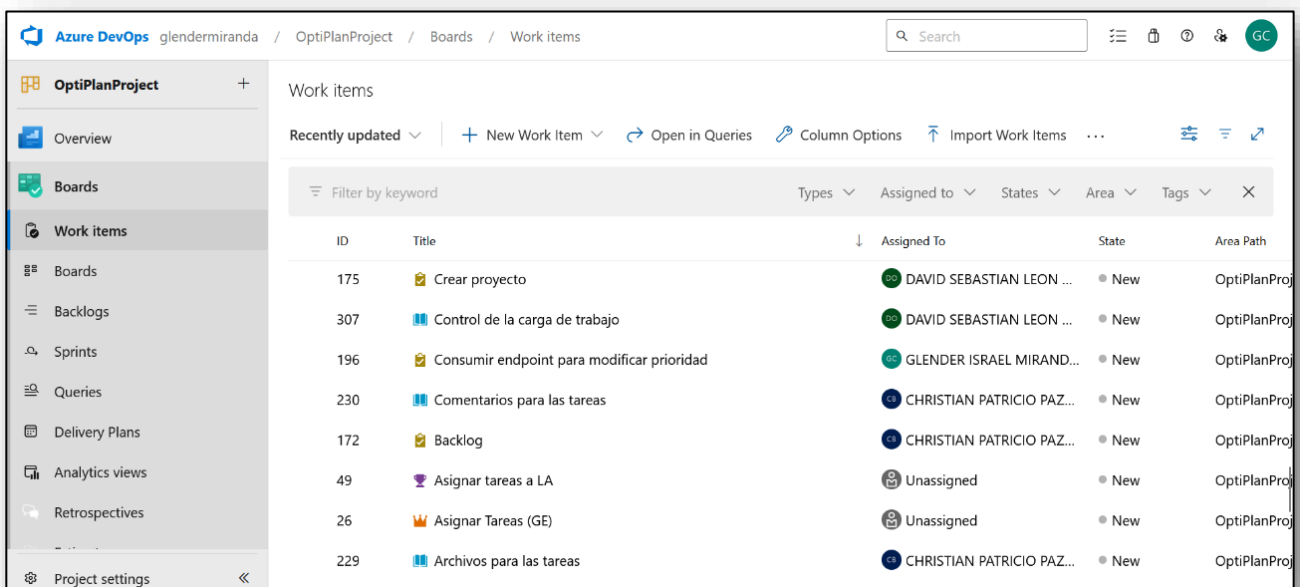


Figura 6. Azure DevOps: Boards

En la sección “Backlogs” se presenta una visión general de los requisitos, el cual agrupa todas las historias de usuario que han sido definidas para el proyecto. Este backlog es esencial para la planificación y priorización de los sprints o iteraciones. La Figura 7 muestra la pantalla con las historias de usuario del proyecto.

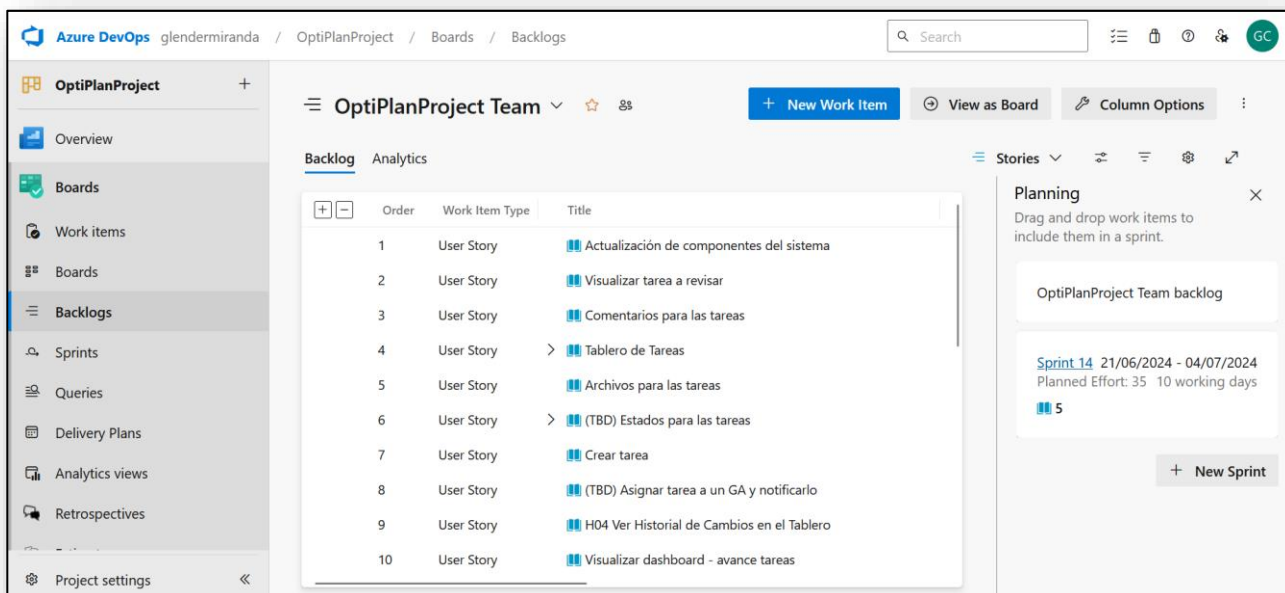


Figura 7. Azure DevOps: Backlogs

En la sección “Sprints” se gestionan el Sprint Backlog (historias de usuario y tareas) mediante en un tablero Kanban. Esto permite una mejor transparencia de las actividades realizadas por los equipos. La Figura 8 presenta la sección Sprints dentro de Azure DevOps.

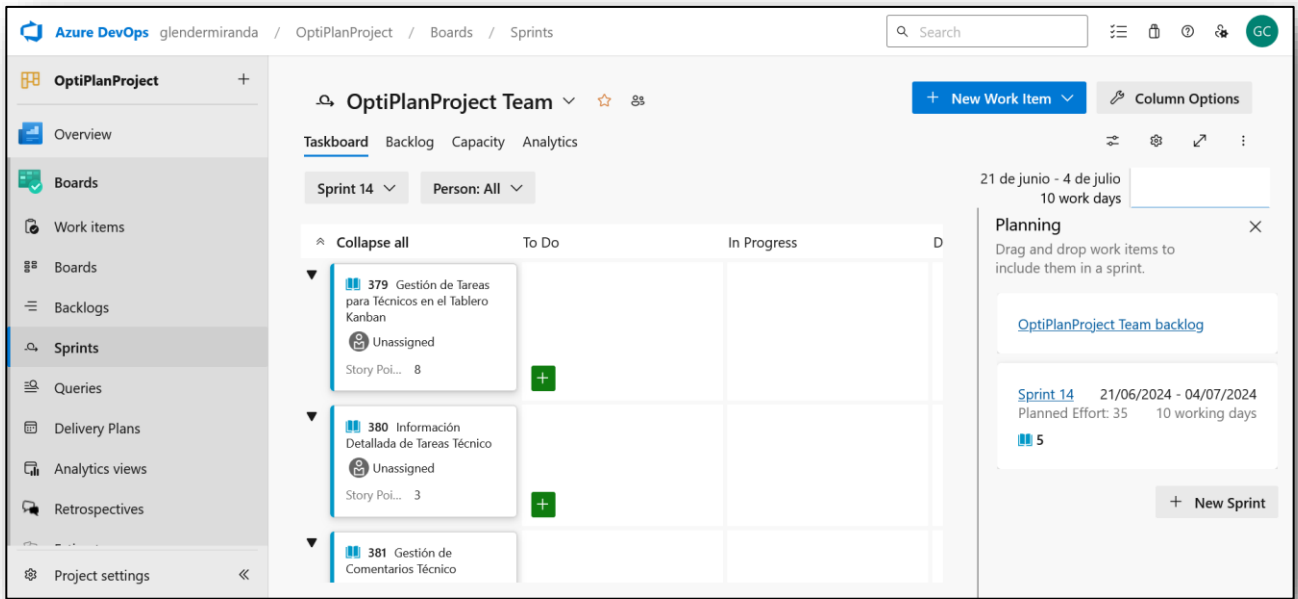


Figura 8. Azure DevOps: Sprints

En la sección “Retrospectives” se muestra el tablero usado para la retrospectiva de cada sprint. Se puede visualizar este apartado en la Figura 9. El tablero mostrado en esta sección sirve como un punto focal para registrar y analizar los comentarios y observaciones recogidos durante la retrospectiva.

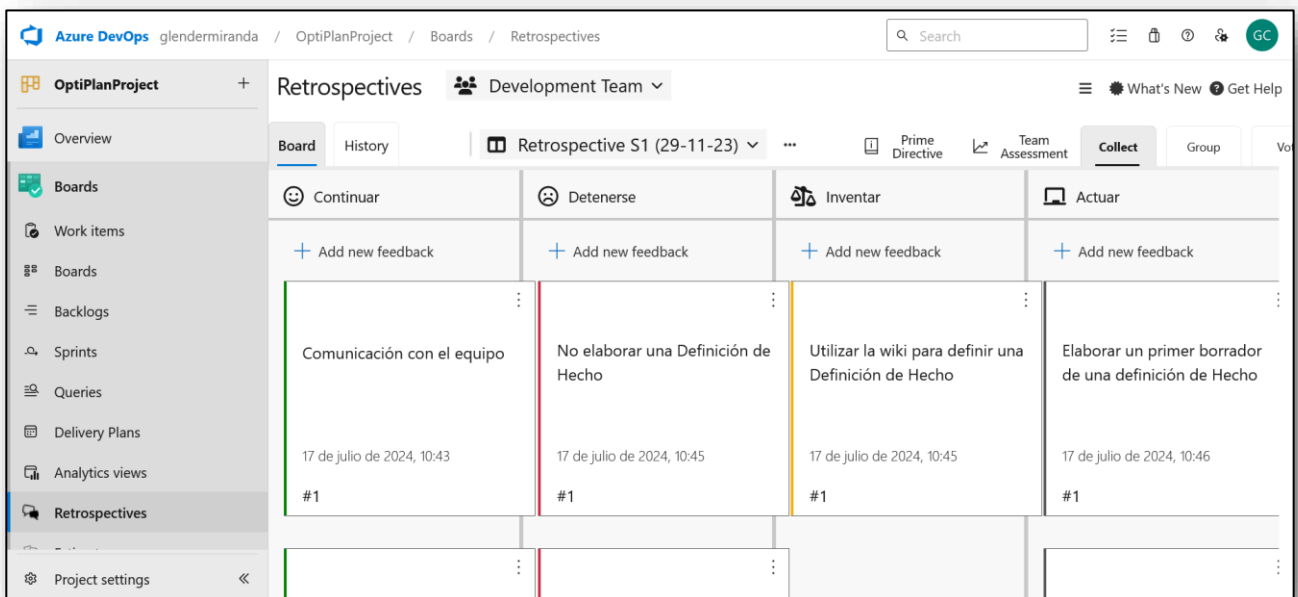


Figura 9. Azure DevOps: Retrospectives

1.4.2.2. Azure Repos

Azure Repos proporciona repositorios Git privados, ofreciendo herramientas para manejar el código, como la gestión de ramas, visualización de commits y creación de pull requests [7]. La Figura 10 muestra la ventana principal de los repositorios.

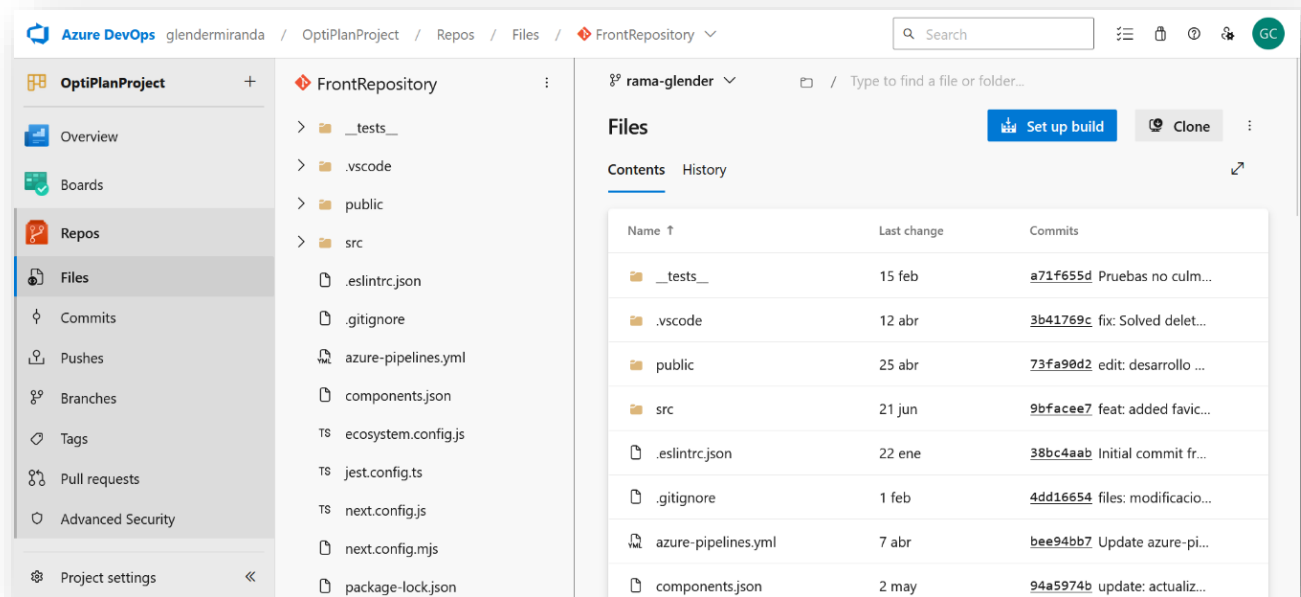


Figura 10. Azure DevOps: Repos

Dentro de los repositorios, la sección “Commits” proporciona una representación gráfica detallada de todos los cambios realizados a lo largo del tiempo en las distintas ramas del repositorio. Este gráfico es una herramienta crucial para visualizar el historial de modificaciones, facilitando el seguimiento de las versiones y la colaboración en el desarrollo. La Figura 11 detalla la sección mencionada.

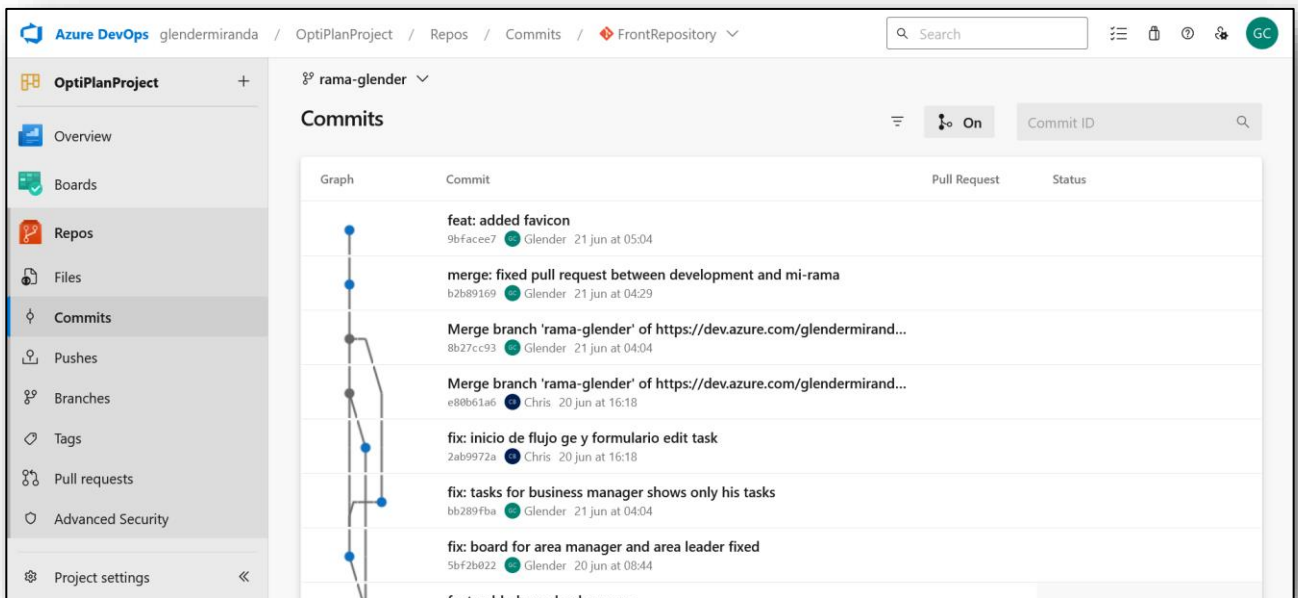


Figura 11. Azure DevOps: Commits

La sección “Pushes” ofrece una vista detallada de los cambios que han sido subidos desde un repositorio local hacia los repositorios de Azure DevOps. Este apartado es fundamental para monitorizar las actualizaciones y sincronizaciones entre los repositorios locales y remotos, garantizando que todos los cambios se reflejen adecuadamente en el entorno de desarrollo compartido. En la Figura 12 se proporciona una vista del apartado Pushes en Azure DevOps.

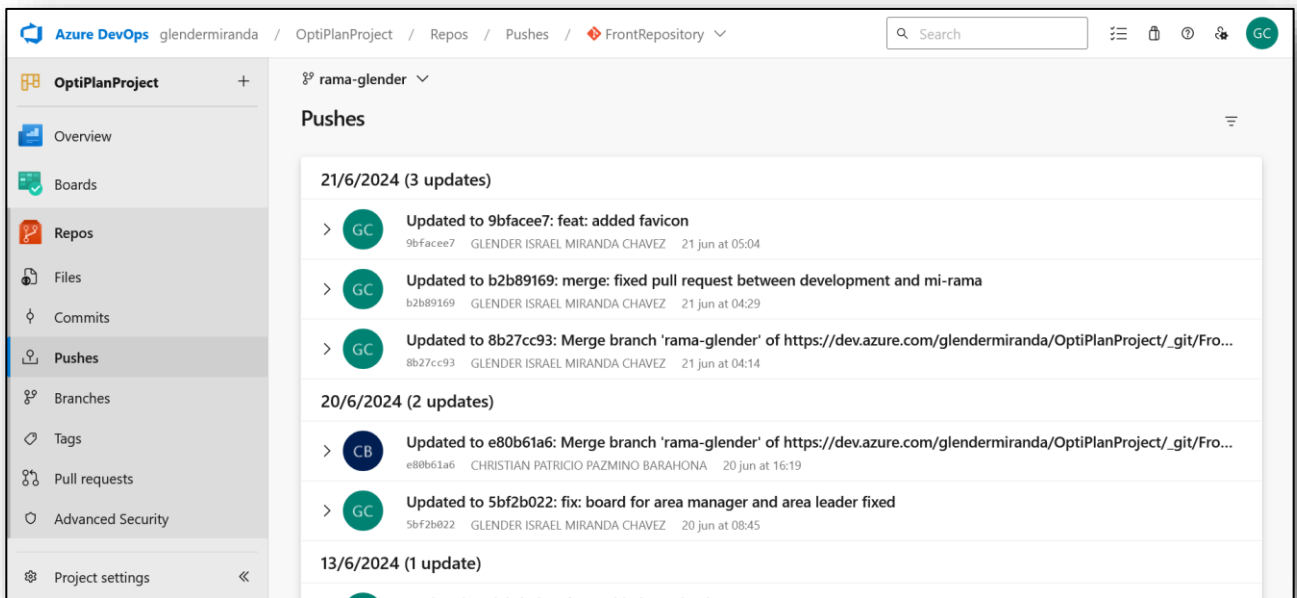


Figura 12. Azure DevOps: Pushes

1.4.2.3. Azure Pipelines

Azure Pipelines permite la construcción, prueba y despliegue automático de código. Admite varios sistemas de control de versiones, como Azure Repos, Git, TFVS y GitHub (ver Figura 13).

Es compatible con múltiples lenguajes de programación, incluyendo Java, JavaScript, Node.js, Python, .NET y C++. Las aplicaciones pueden desplegarse en diversos entornos, como máquinas virtuales, contenedores, servicios de Azure y servicios locales (on-premises) [7].

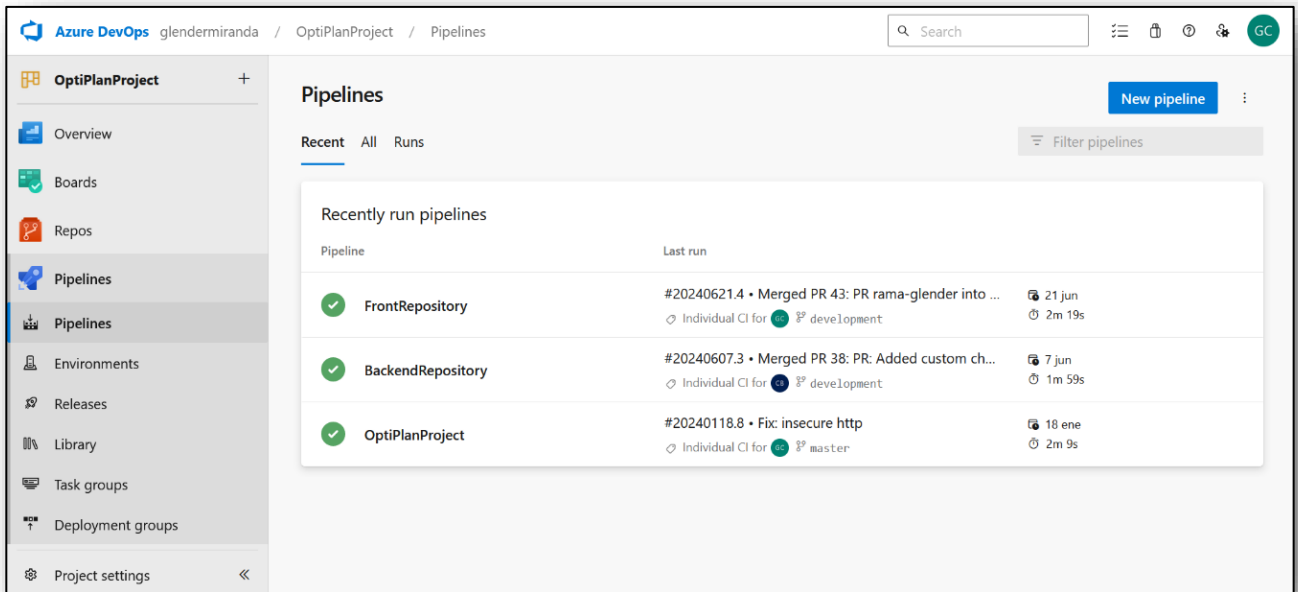


Figura 13. Azure DevOps: Azure Pipelines

En la sección “Releases” se presenta la configuración del proceso de despliegue a los distintos entornos de desarrollo y producción, tanto para el backend como para el frontend del proyecto. Esta sección es fundamental para gestionar y controlar las liberaciones del software y asegurar que las actualizaciones se implementen de manera efectiva y sin interrupciones. En la Figura 14 se visualiza la sección mencionada.

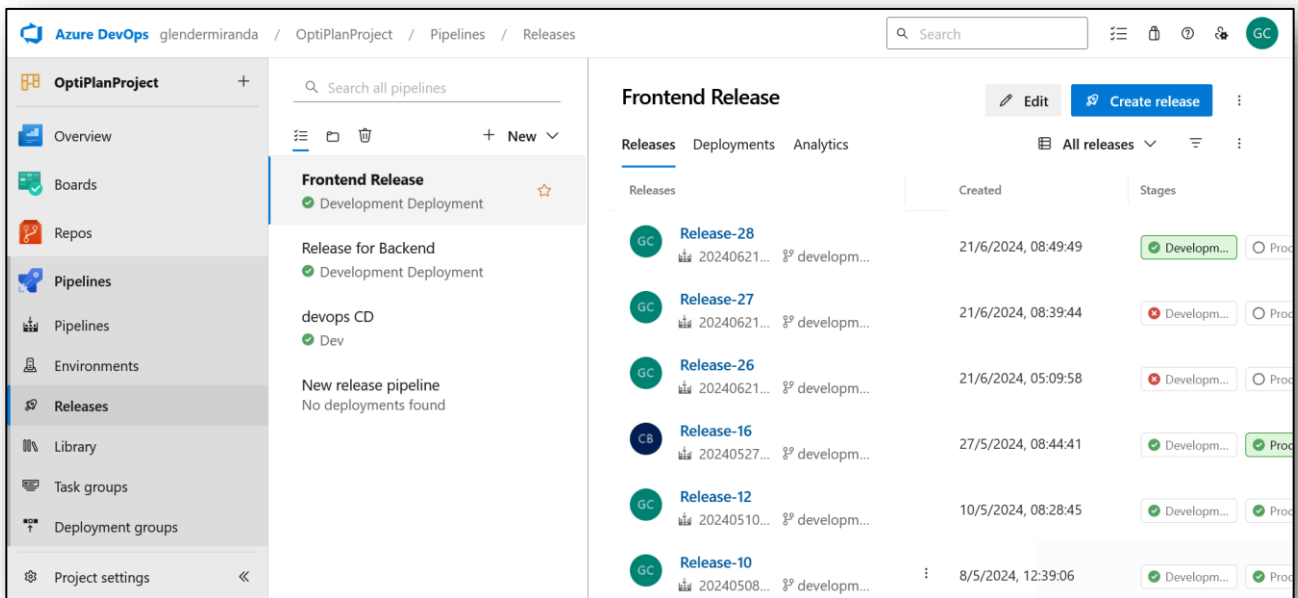


Figura 14. Azure DevOps: Releases

1.4.3. Tableros Kanban

Kanban es un enfoque ágil que facilita la visualización y gestión del flujo de tareas dentro de un grupo de trabajo. Originaria de Japón y utilizada por primera vez en la industria automotriz por Toyota, Kanban ha evolucionado para convertirse en una herramienta esencial en la gestión de proyectos de software y otros sectores. Este enfoque permite iniciar flujos de trabajo y dividirlos en estados, proporcionando un control claro del avance de un proyecto [17].

Características Principales de Kanban:

- **Visualización del Trabajo:** Los tableros Kanban utilizan tarjetas para representar tareas y columnas representa el estado de las tareas, lo que permite a los equipos visualizar todo el trabajo en progreso.
- **Flujo de Trabajo Organizado:** Kanban ayuda a mantener un flujo de trabajo constante y organizado. Las tareas se mueven de una columna a otra, reflejando su estado desde "Pendiente" hasta "Completado", lo que facilita la identificación de cuellos de botella.

- Flexibilidad: A diferencia de otros enfoques ágiles como Scrum, Kanban no requiere ciclos de trabajo fijos o iteraciones. Los equipos pueden agregar nuevas tareas en cualquier momento, lo que ofrece una gran flexibilidad.
- Límites de Trabajo en Proceso (WIP): Kanban permite establecer límites en la cantidad de trabajo en proceso en cada columna, lo que ayuda a evitar la sobrecarga de trabajo.
- Mejora Continua: La metodología Kanban fomenta la revisión y mejora continua de los procesos. Los equipos analizan regularmente su flujo de trabajo para identificar y eliminar ineficiencias.
- Entrega Justo a Tiempo (JIT): Kanban en el desarrollo de software se centra en entregar tareas y productos justo a tiempo, es decir, cuando se lo necesita, mejorando la eficiencia y reduciendo los tiempos de espera [17].

2. METODOLOGÍA

2.1. Análisis de herramientas DevOps

La preparación de la metodología para este proyecto comienza con un análisis de las herramientas DevOps disponibles. El objetivo es seleccionar la herramienta más adecuada para gestionar repositorios, implementar integración continua o CI (CI del inglés Continuous Integration) y entrega continua o CD (CD del inglés Continuous Delivery), y configurar la infraestructura necesaria. La selección de herramientas se basa en varios criterios, que se detallan en la Tabla 1. A continuación, se mencionan las herramientas analizadas:

Tabla 1. Comparación de herramientas DevOps

Comparación de herramientas DevOps				
Criterio	Azure DevOps	Jenkins	GitLab	AWS CodePipelines
Gestión de Repositorios	Integrado con Git y TFVC	Integración con Git y SVN	Integrado con Git	Integración con Git y CodeCommit
CI/CD	Pipelines integrados con YAML	Extensible con plugins	Pipelines integrados con YAML	Pipelines integrados con YAML
Configuración de Infraestructura	Integración con Azure Resource Manager (ARM)	Integración mediante plugins	Integración con Terraform y Kubernetes	Integración con CloudFormation y Terraform
Soporte de Servidor	Servicio en la nube y On-premises	On-premises	Servicio en la nube y On-premises	Servicio en la nube
Licencias	Gratuito para equipos pequeños, modelo de pago para equipos grandes	Gratuito, modelo de pago para soporte adicional	Gratuito para equipos pequeños, modelo de pago para equipos grandes	Pago por uso
Facilidad de Uso	Interfaz gráfica intuitiva, fácil de configurar	Requiere configuración manual y conocimiento de plugins	Interfaz gráfica intuitiva, fácil de configurar	Interfaz gráfica intuitiva, fácil de configurar

Integración con otras herramientas	Integración nativa con herramientas de Azure y Microsoft	Amplia integración con diversas herramientas mediante plugins	Integración nativa con herramientas de GitLab y terceros	Integración nativa con servicios de AWS
Escalabilidad	Alta escalabilidad en la nube	Escalable según la infraestructura subyacente	Alta escalabilidad en la nube	Alta escalabilidad en la nube
Soporte y Comunidad	Soporte oficial de Microsoft, amplia comunidad	Amplia comunidad, soporte mediante comunidad y pago adicional	Soporte oficial de GitLab, amplia comunidad	Soporte oficial de AWS, amplia comunidad
Seguridad	Seguridad integrada con servicios de Azure	Seguridad configurable mediante plugins	Seguridad integrada con servicios de GitLab	Seguridad integrada con servicios de AWS

Se comparó con la herramienta Jenkins¹, la cual es un servidor de automatización que se puede utilizar para automatizar tareas de CI y CD. Cuenta con una variedad de paquetes en los que puede ser instalado, sin embargo, es principalmente de uso on-premise, por lo que el uso de servicios en la nube es limitado.

GitLab² es una herramienta que permite gestionar el ciclo de vida del desarrollo de software. Puede gestionar repositorios, CI y CD, accesos y seguridades, entre otros. Es muy versátil, tiene una interfaz intuitiva y soporta el uso de pipelines en formato YML, además de ser un servicio en la nube.

El servicio de entrega continua de AWS CodePipeline³ permite de manera sencilla y automática realizar la construcción, pruebas y despliegues de aplicaciones directamente en la nube. Este servicio permite el uso de recursos de AWS, el uso de dichos recursos tiene un costo mensual, además de que este servicio se centra más que nada en proceso de entrega y despliegue.

¹ <https://www.jenkins.io/doc>

² <https://about.gitlab.com>

³ <https://docs.aws.amazon.com/codepipeline>

Azure DevOps es una herramienta de Microsoft que proporciona una completa suite de servicios que ayudan a todo el equipo de Scrum a gestionar no solo los procesos de integración y entrega del producto, sino, también al manejo de la planificación del proyecto.

Azure DevOps cuenta con una amplia gama de servicios que permiten trabajar de manera colaborativa y ordenada. Además, el licenciamiento es muy flexible y gratuito para equipos pequeños. Esta suite cuenta con integraciones con los recursos de Azure, por lo que es mucho más sencillo trabajar con los despliegues y entregas del aplicativo.

Cada herramienta se evaluó en términos de su capacidad para gestionar repositorios, soportar CI/CD, y facilitar la configuración de la infraestructura. En la Tabla 1, se presenta un resumen de los criterios de evaluación utilizados, tales como el soporte de servidor, licencias, y otros aspectos relevantes. Para más detalles sobre cada herramienta, se puede consultar su página oficial (véase pie de página para enlaces).

Azure DevOps se destaca como la mejor opción debido a su completa suite de servicios que no solo facilitan la integración y entrega continua (CI/CD), sino que también abordan la planificación y gestión del proyecto, esencial para equipos que utilizan Scrum. A diferencia de Jenkins, que se centra en entornos on-premise, y AWS CodePipeline, que se especializa en la entrega y despliegue en la nube, Azure DevOps ofrece una integración perfecta con los recursos de Azure, lo que simplifica enormemente los despliegues y entregas de aplicaciones.

Además, GitLab, aunque versátil y con una interfaz intuitiva, no ofrece la misma profundidad de integración con Azure, lo que puede ser crucial para proyectos que ya utilizan la infraestructura de Microsoft. Azure DevOps también proporciona un licenciamiento flexible y gratuito para equipos pequeños, facilitando la colaboración y organización del trabajo.

2.2. Configuración de recursos en Azure Portal

Para aprovisionar los recursos necesarios para los procesos de integración y despliegue, se utilizó la plataforma Azure Portal. Esta plataforma ofrece una amplia variedad de servicios para diferentes tecnologías. En nuestro caso, estamos trabajando con NextJS y NestJS, que son frameworks basados en React. Por lo tanto, fue necesario crear y configurar los recursos de manera que sean compatibles con estas tecnologías.

Además, se necesitaron recursos para almacenar los datos, para lo cual se utilizó un servidor flexible de bases de datos PostgreSQL. Para desplegar tanto el frontend como el backend, se emplearon los servicios de aplicaciones web de Azure (Web App Services). Esto aseguró que todas las partes del proyecto estuvieran correctamente alojadas y gestionadas dentro de la misma plataforma.

2.2.1. Web App Services

Azure Web App Service es una plataforma que permite alojar aplicaciones web, API REST y backends. Admite múltiples lenguajes de programación como .NET, Java, Node.js, PHP y Python, y se puede ejecutar en sistemas operativos Windows y Linux [18].

Los Web App Services, de aquí en adelante denominados “Web Apps” para este trabajo escrito, se consumen en base al proceso que se utiliza, dependiendo del plan que se escoja, se puede llegar a tener un mayor consumo de créditos de Azure.

Para nuestro caso, fue necesario crear cuatro Web Apps, a continuación, se detallan estos recursos en la Tabla 2:

Tabla 2. Web App Services

Nombre	Tipo	Ambiente	Dominio
frontprod-optiplanepn	Frontend	Producción	https://frontprod-optiplanepn.azurewebsites.net
frontdev-optiplanepn	Frontend	Desarrollo	https://frontdev-optiplanepn.azurewebsites.net
backprod-optiplanepn	Backend	Producción	https://backprod-optiplanepn.azurewebsites.net
backdev-optiplanepn	Backend	Desarrollo	https://backdev-optiplanepn.azurewebsites.net

La configuración de estos recursos se realiza de manera similar, en la Figura 15 se muestra la vista general de un Web App en la plataforma de Azure Portal. En esta vista se habilitan configuraciones básicas como ejecutar o parar el servicio.

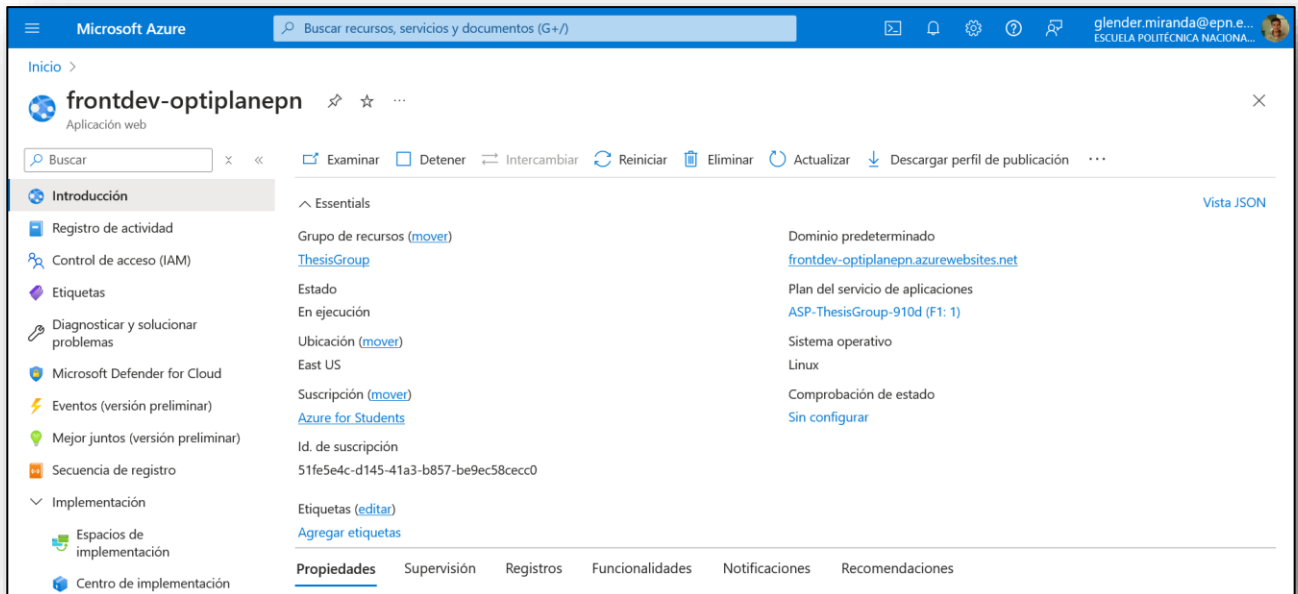


Figura 15. Vista general de un Web App Service

Una de las configuraciones que se han realizado se encuentra en el centro de implementación. En la Figura 16 se muestra la conexión entre los pipelines de Azure DevOps con el recurso. Esta conexión permite desplegar el aplicativo de manera automática una vez se ha construido.

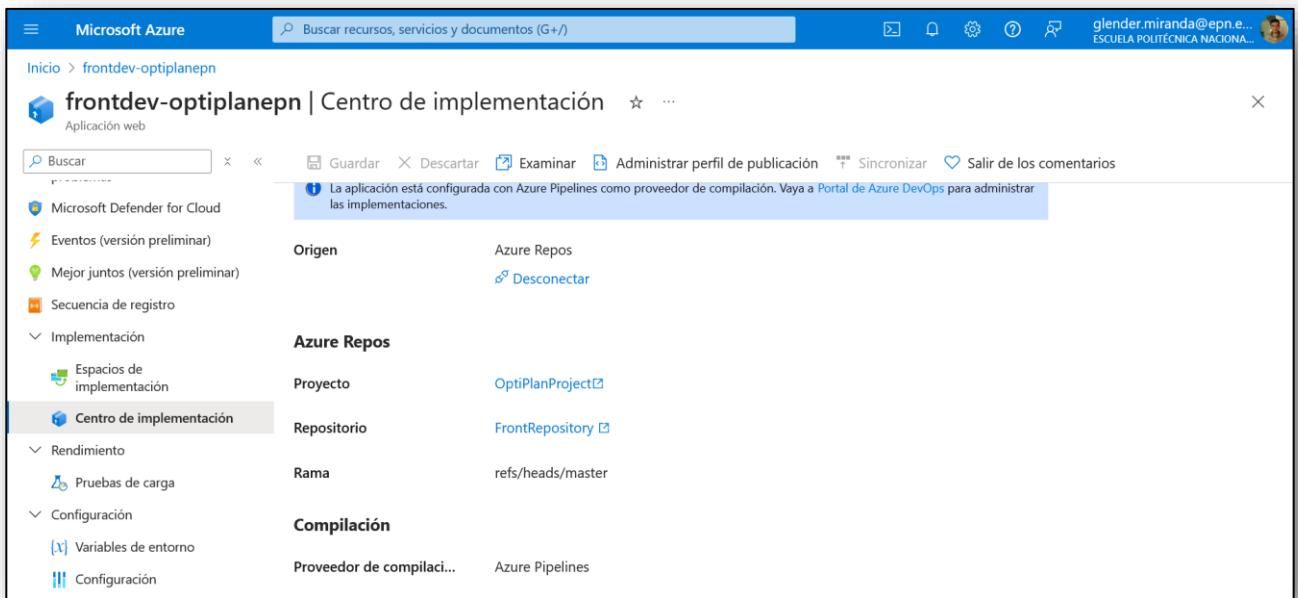


Figura 16. Centro de implementación del Web App

Una de las configuraciones más importantes implementadas, es el establecimiento de las variables de entorno del Web App. En la Figura 17 se visualiza las variables de entorno utilizadas, de las cuales se destacan la dirección URL del servicio de backend (NEXT_PUBLIC_BACKEND_URL), y la clave secreta para realizar el inicio de sesión en el servicio de backend (NEXTAUTH_SECRET).

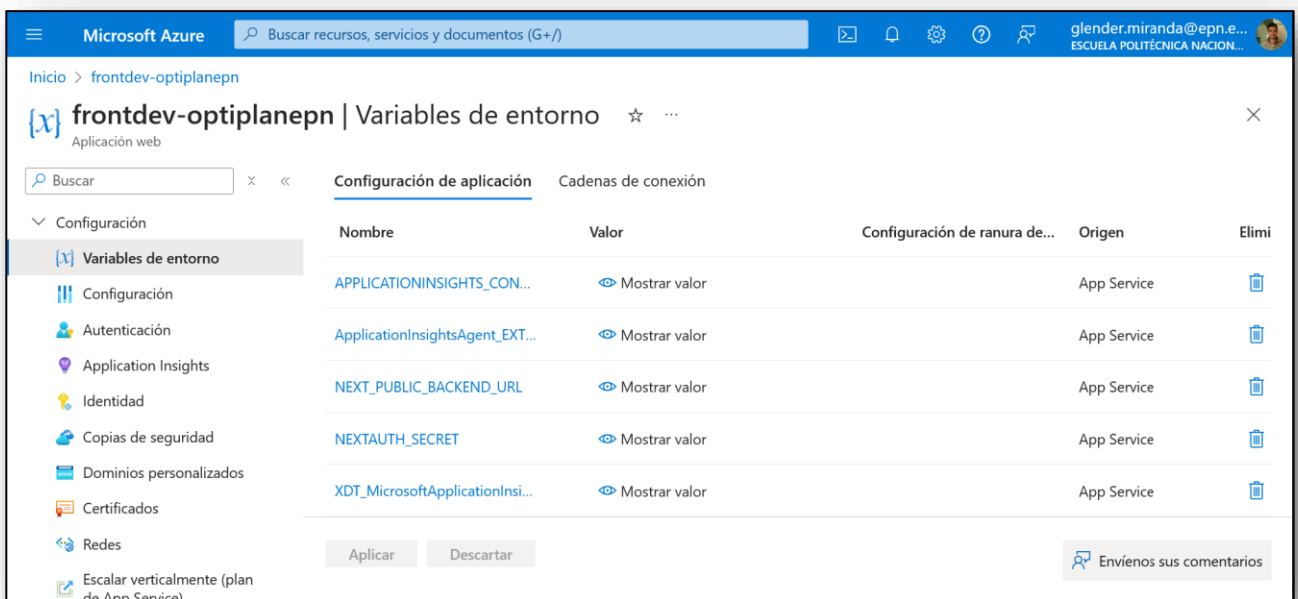


Figura 17. Variables de entorno del recurso Web App

Para asegurar la compatibilidad con el aplicativo a desplegar, es necesario establecer la pila o entorno de ejecución. Todos los Web Apps que se han aprovisionado están corriendo en la misma versión: Node 18 LTS. La Figura 18 presenta las configuraciones generales de los Web Apps implementados.

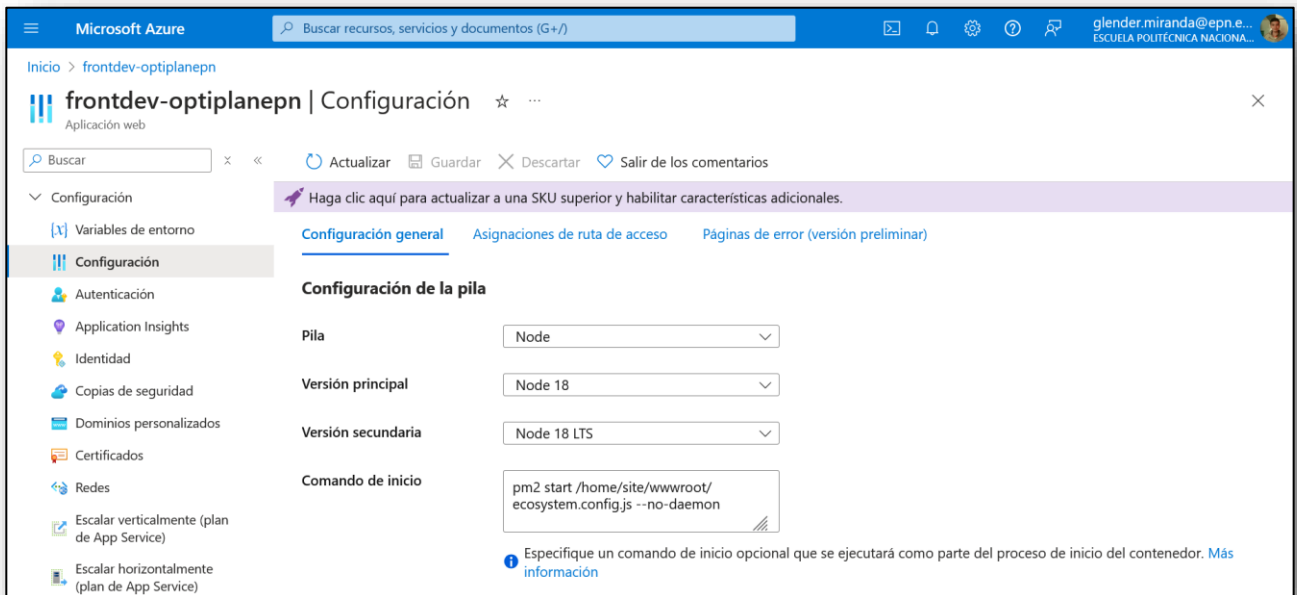


Figura 18. Configuración de la pila del recurso de Web App

Para que el aplicativo comience a funcionar, se proporciona un comando que permite el inicio del servicio web en el contenedor del Web App. En la Figura 19 se presenta el código por el cual es posible el despliegue del aplicativo.

El código exporta un objeto de configuración en la línea 1, que contiene un array de aplicaciones en la línea 2. En la línea 3, se define el nombre de la aplicación como "property-dev". La línea 4 especifica el script de Next.js a ejecutar. Los argumentos para el script se configuran en la línea 6, estableciendo el puerto con la variable de entorno PORT o el puerto 3000 por defecto. La línea 7 indica que la aplicación no debe reiniciarse automáticamente al detectar cambios en los archivos, mientras que la línea 8 permite el reinicio automático si la aplicación se detiene.


```
1  module.exports = {
2  apps: [
3    {
4      name: "property-dev",
5      script: "./node_modules/next/dist/bin/next",
6      args: "start -p " + (process.env.PORT || 3000),
7      watch: false,
8      autorestart: true,
9    },
10 ],
11 };
```

GLENDER ISRAEL MIRANDA CHAVEZ, 6 months ago

Figura 19. Archivo de despliegue del aplicativo

El intercambio de recursos de origen cruzado (CORS, del inglés Cross-Origin Resource Sharing) es un método que permite a un servidor especificar los dominios a los que está permitido acceder a la carga de sus recursos. Es un mecanismo que funciona como una verificación previa para que los navegadores puedan realizar solicitudes al servidor [19].

Dentro de los servicios de Web Apps, es necesario especificar que las solicitudes realizadas por cualquier navegador sean aceptadas por el recurso. Por ello, es fundamental permitir todos los orígenes desconocidos haciendo uso del carácter "*", dicho carácter indica la aceptación de cualquier origen hacia el recurso. La Figura 20 indica la definición de esta configuración.

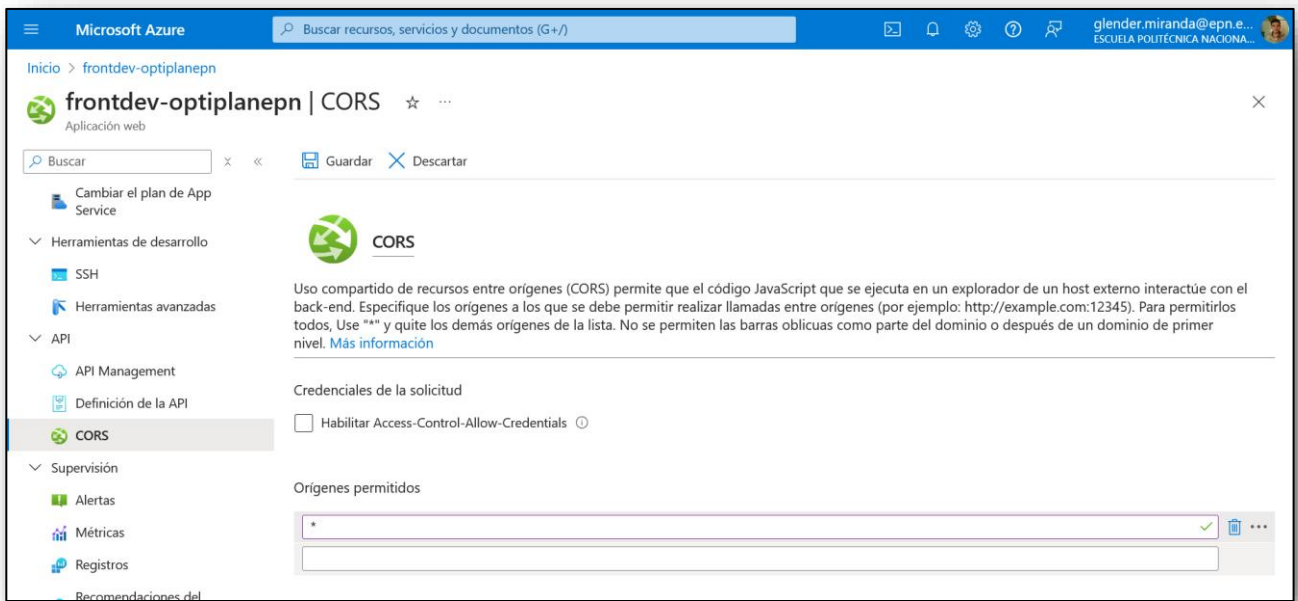


Figura 20. Configuración de CORS del recurso de Web App

Una herramienta que ofrecen los Web Apps para el monitoreo es la Secuencia de registro. Esta herramienta funciona como una consola de monitorización que permite conocer el estado del aplicativo en tiempo de ejecución. De esta manera, podemos visualizar, como se muestra en la Figura 21, los posibles errores de despliegue y obtener información del cual puede ser el problema.

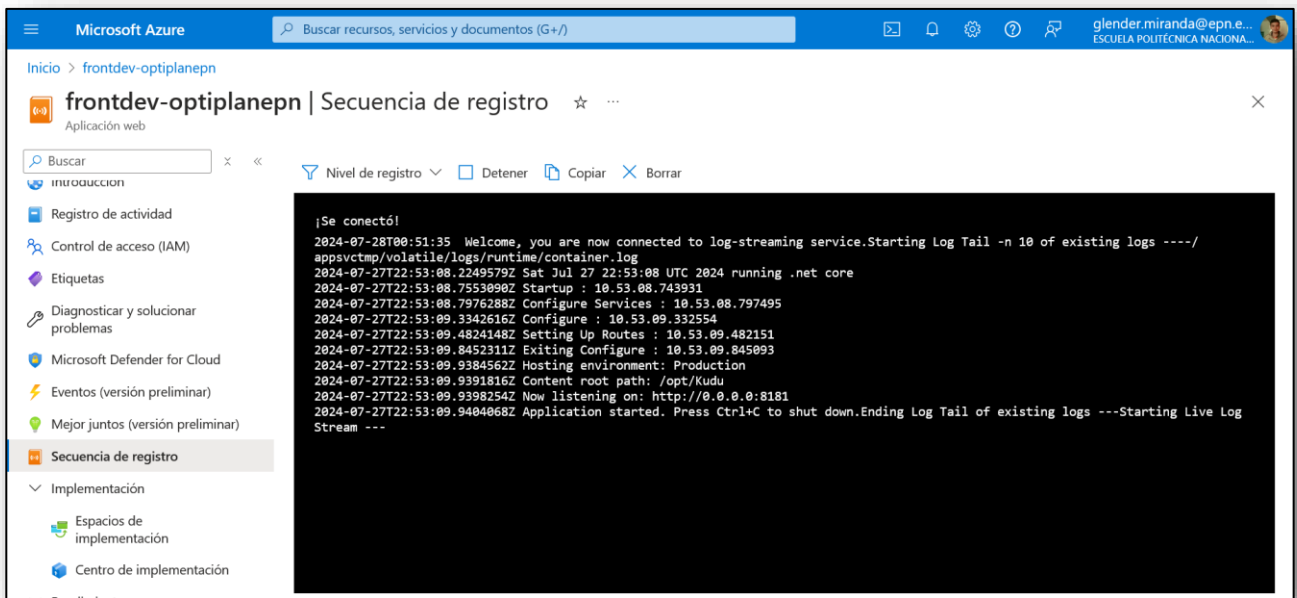


Figura 21. Secuencia de registro del recurso de Web App

2.2.2. Bases de datos

PostgreSQL fue la base de datos que se utilizó para almacenar toda la información generada por la aplicación. Esta base de datos es compatible con NestJS, el framework usado para el backend. Además, dentro de los recursos de Azure, PostgreSQL es una opción más económica en comparación con otras bases de datos.

En este caso, se ha hecho uso de dos recursos para las bases de datos de los diferentes ambientes. La Tabla 3 muestra las características básicas de dichos recursos.

Tabla 3. Bases de datos

Nombre	Tipo	Ambiente	Dominio
backprod-optiplanepn-server	Backend	Producción	https://backprod-optiplanepn-server.postgres.database.azure.com
backdev-optiplanepn-server	Backend	Desarrollo	https://backdev-optiplanepn-server.postgres.database.azure.com

Los servidores y bases de datos están configurados de la misma manera, diferenciándose solo en el entorno. La base de datos de producción se empleará para almacenar los datos críticos del negocio, mientras que la base de datos de desarrollo se utilizará para probar

funcionalidades durante el desarrollo de la aplicación. La Figura 22 muestra la página principal del recurso en el portal de Azure, destacando algunas configuraciones básicas como el encendido o apagado del servidor.

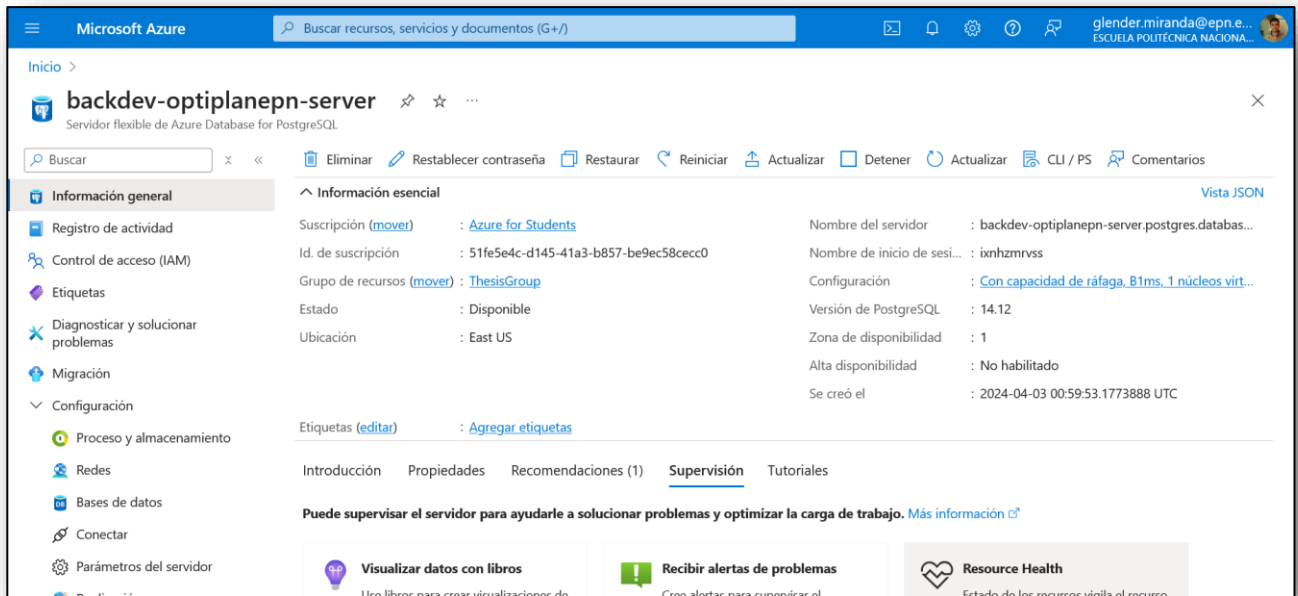


Figura 22. Configuraciones básicas del motor de base de datos

En la Figura 23 se muestra las bases de datos que se han creado para este servidor, además de las creadas por defecto, destaca la base de datos backdev-optiplanepn-database, la cual es la base de datos principal.

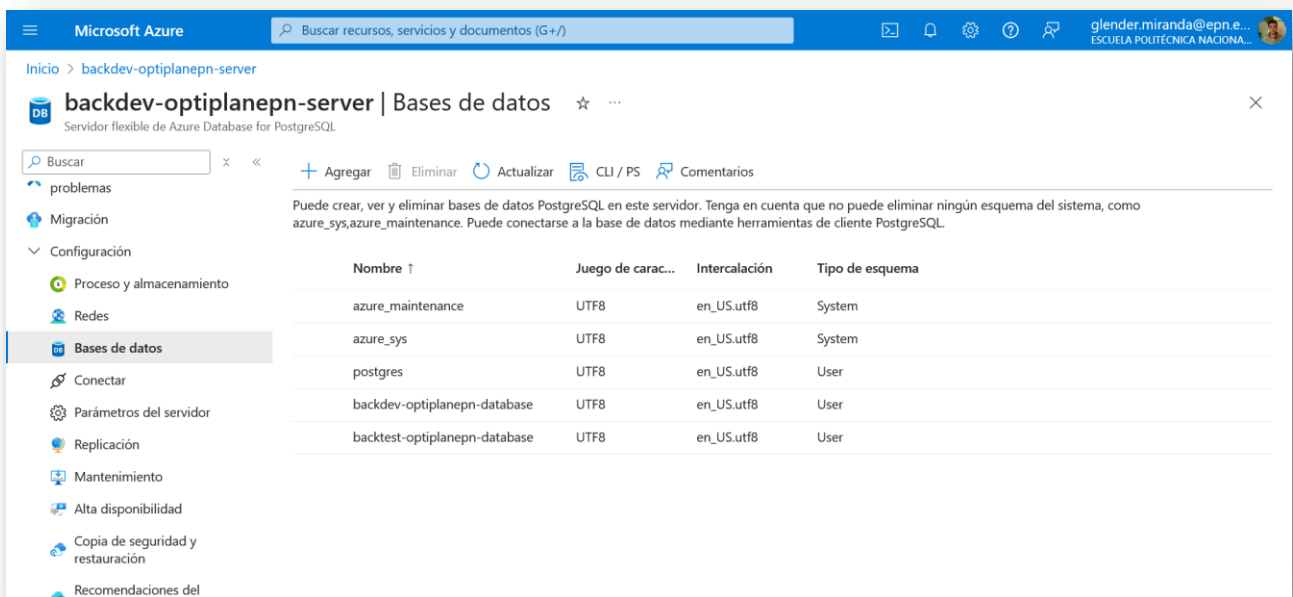


Figura 23. Bases de datos del servidor flexible de PostgreSQL

El propio recurso ofrece las credenciales para poder conectarse a la base de datos. La Figura 24 indica la sección “Conectar”, dicha sección permite seleccionar la base de datos a conectar y las credenciales de acceso.

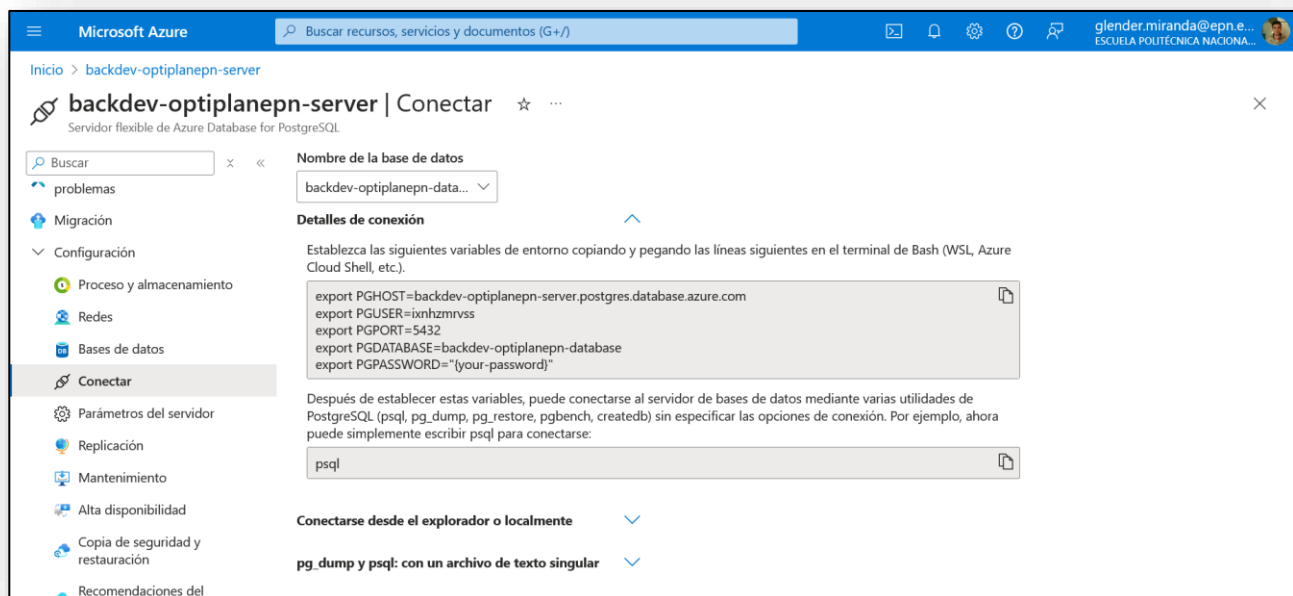


Figura 24. Conexión a las bases de datos PostgreSQL

Una configuración muy importante para poder desplegar el Web App es la definición de la extensión “UUID_OSSP”. Esta extensión permite a las bases de datos utilizar valores “uuid” como identificadores únicos para los datos en la base de datos. Sin esta configuración, surgirán errores al iniciar el servidor. La Figura 25 muestra la habilitación de la extensión mencionada.

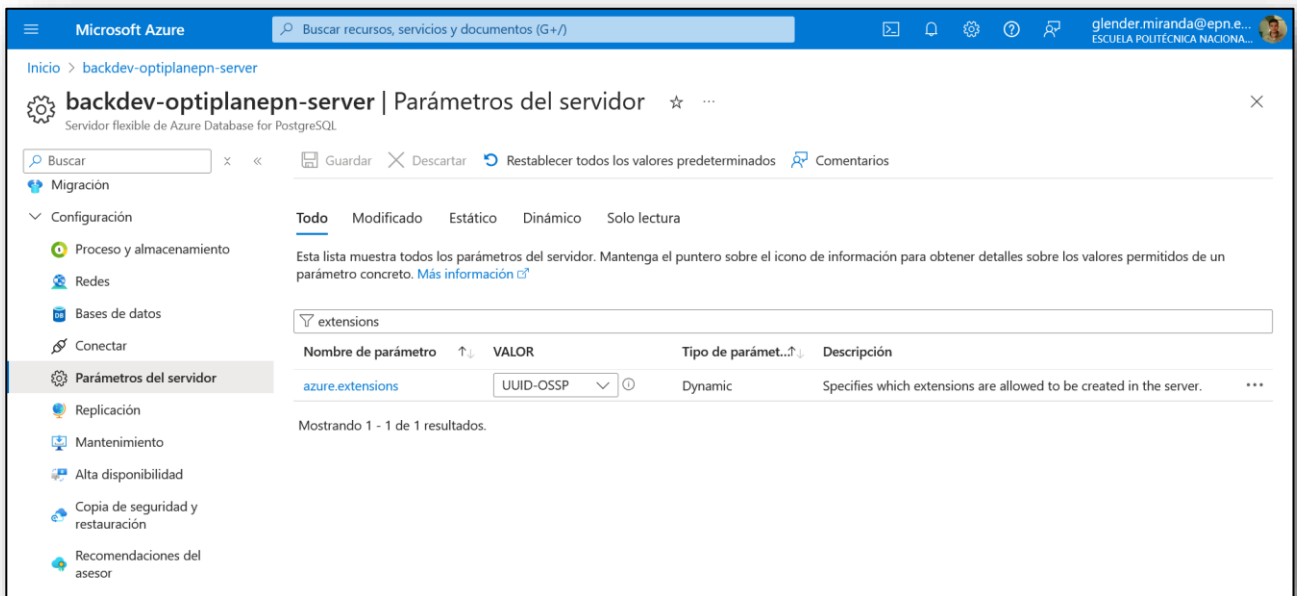


Figura 25. Parámetros del servidor de base de datos

2.3. Configuración de Azure DevOps

2.3.1. Organización y políticas de Azure DevOps

Una organización en Azure DevOps es una plataforma donde los equipos pueden colaborar y gestionar proyectos de software. Permite planificar y realizar un seguimiento de todas las tareas, trabajar en múltiples proyectos y gestionar los accesos de administradores y colaboradores, asignando distintos niveles de permisos y acciones según sea necesario [7].

La Figura 26 muestra la organización “glendermiranda”, dicho nombre fue creado por defecto la primera vez se creó la organización. Dentro de la organización existen varios proyectos que pueden ser gestionados de manera global en la configuración general de la organización.

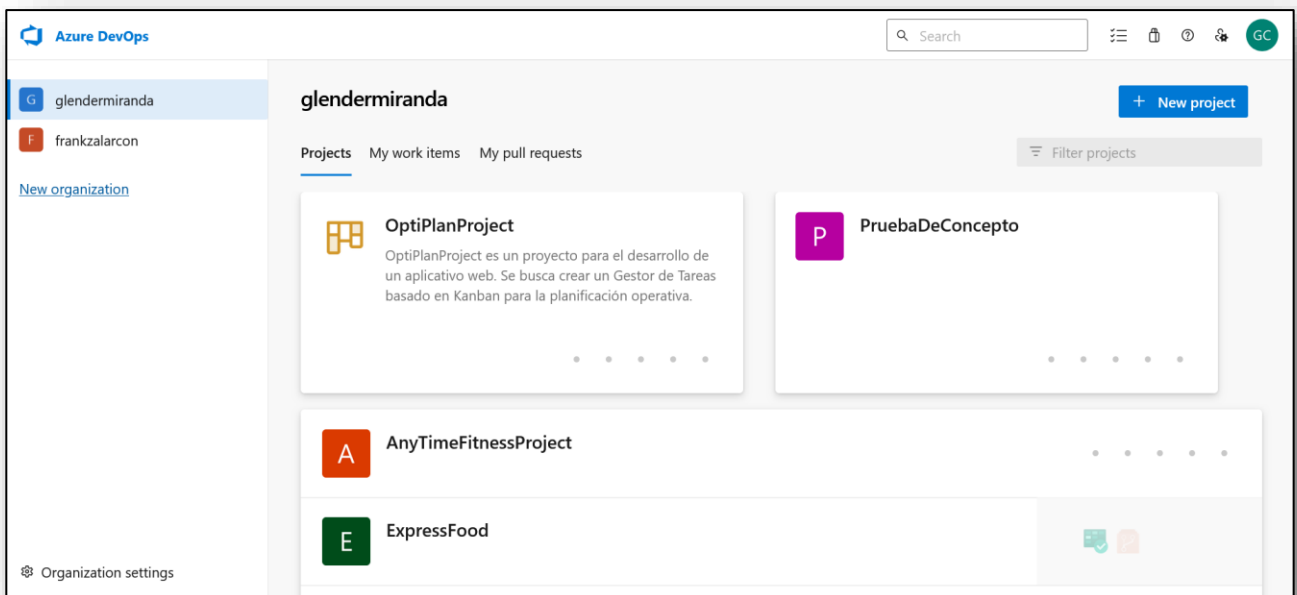


Figura 26. Organización en Azure DevOps

La Figura 27 muestra los miembros de la organización, el nivel de acceso para cada uno es “Basic”, el cual permite acceso a la mayoría de las características de la organización [20]. Se debe tomar en cuenta que, como máximo, en la versión gratuita de Azure DevOps únicamente se pueden añadir cinco usuarios al nivel de acceso “Basic”.

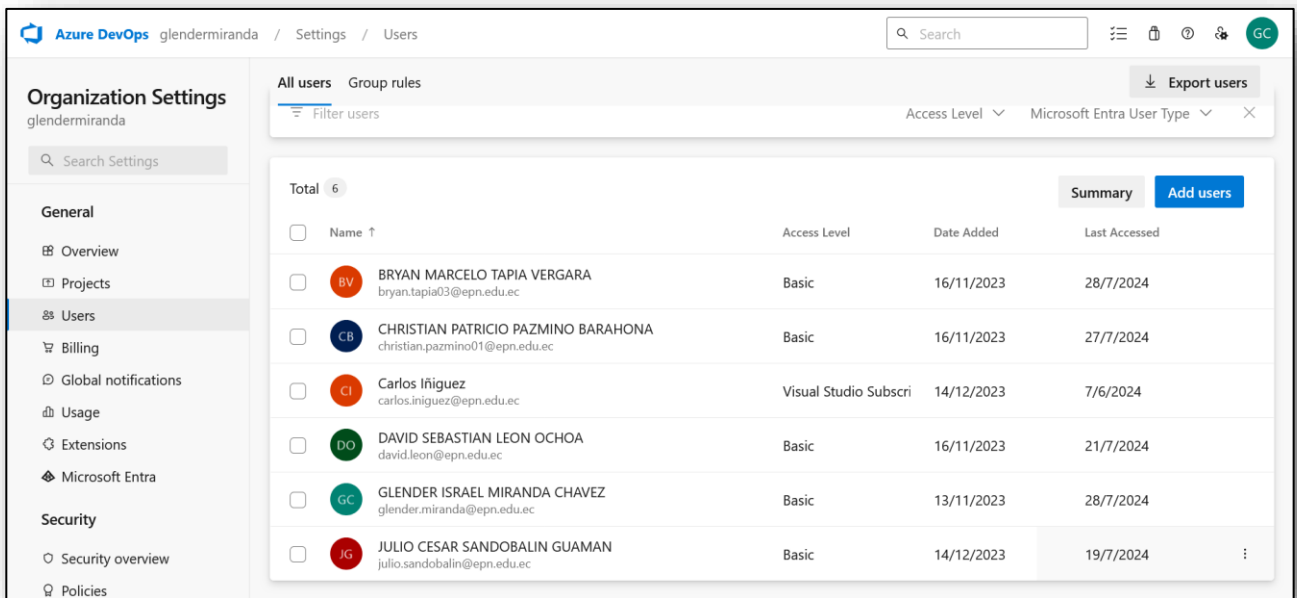


Figura 27. Usuarios de la organización

Se añadieron las extensiones “Estimate” y “Retrospectives” a la organización para poder utilizar estas opciones en los proyectos. La Figura 28 muestra las extensiones instaladas, dichas extensiones fueron utilizadas por el equipo a través del desarrollo de los sprints.

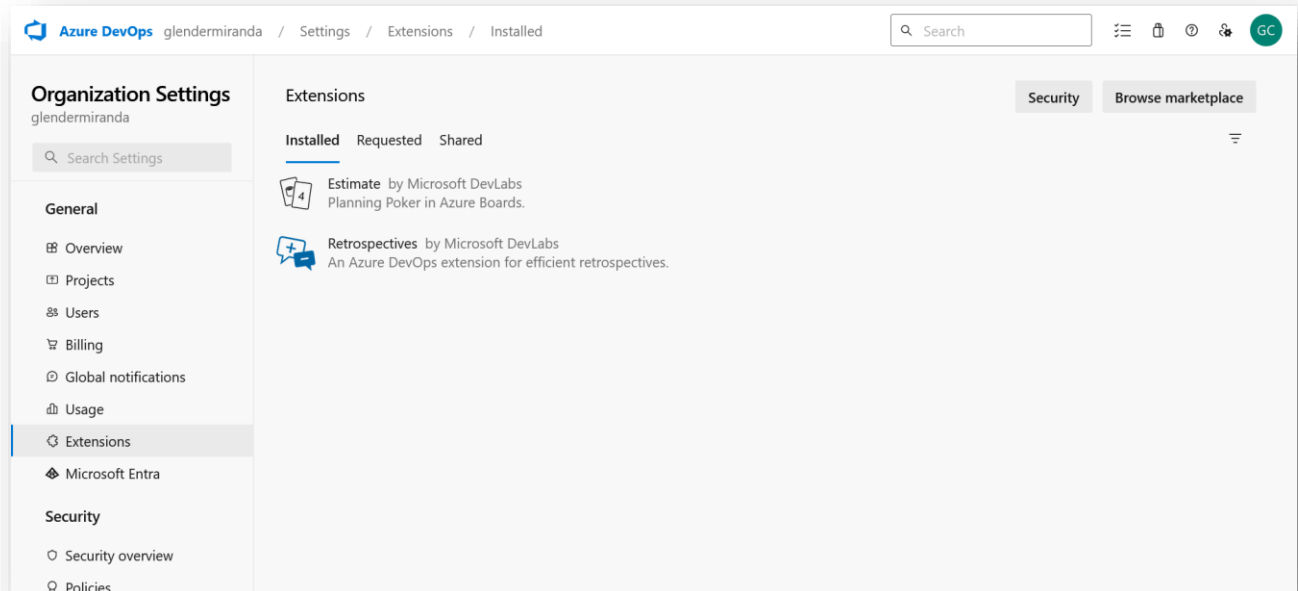


Figura 28. Extensiones de la organización de Azure DevOps

Dentro de la configuración de los pipelines de los proyectos de la organización, es fundamental inhabilitar la opción “Deshabilitar la creación de releases clásicos”, tal como se muestra en la . Esta opción impide la creación de releases en una interfaz interactiva dentro de Azure DevOps, por lo que para poder utilizar esta función y facilitar la visualización de los releases, se optó por inhabilitar esta opción.

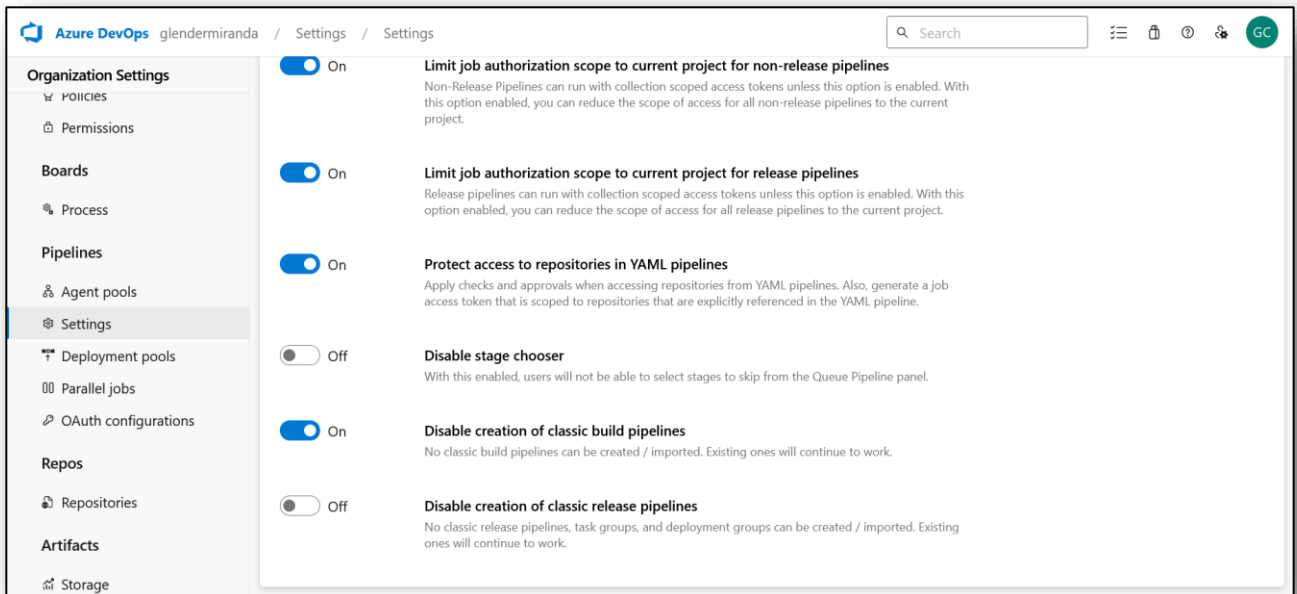


Figura 29. Configuración de los pipelines de los proyectos de la organización

2.3.2. Proyecto de Azure DevOps

Dentro de la organización se creó el proyecto “OptiPlanProject”, este es el lugar donde se trabajará con el marco de trabajo para desarrollar la aplicación web, se gestionarán los repositorios y se realizará los procesos de integración y despliegue continuos.

Dentro del proyecto, se encuentra la sección “Overview” que muestra aspectos básicos del proyecto actual. La Figura 30 exhibe el resumen del proyecto, en este apartado se muestra el título del proyecto, la descripción, el estado del proyecto y los miembros del proyecto.

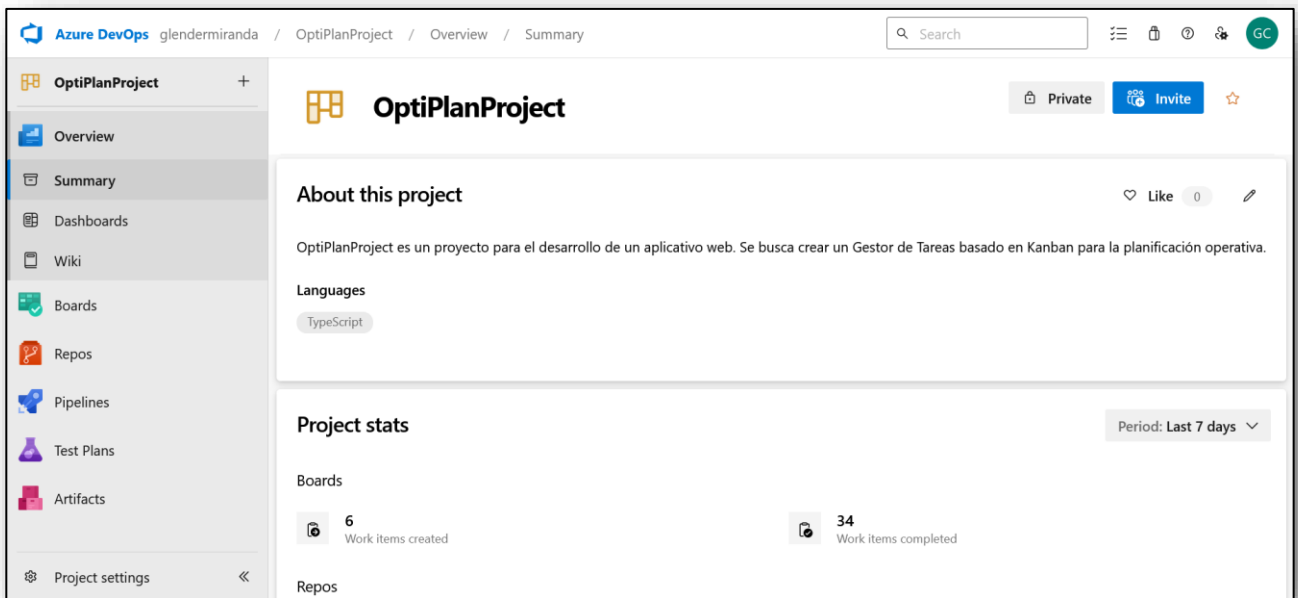


Figura 30. Resumen del proyecto de Azure DevOps

La Figura 31 presenta las configuraciones básicas para el proyecto como el nombre, la descripción, la imagen y el proceso. Un proceso se refiere a la plantilla que permite manejar los elementos de trabajo del proyecto. El proceso que se usará para el proyecto es Agile, ya que esta es la que soporta la mayoría de los términos de metodologías ágiles [21].

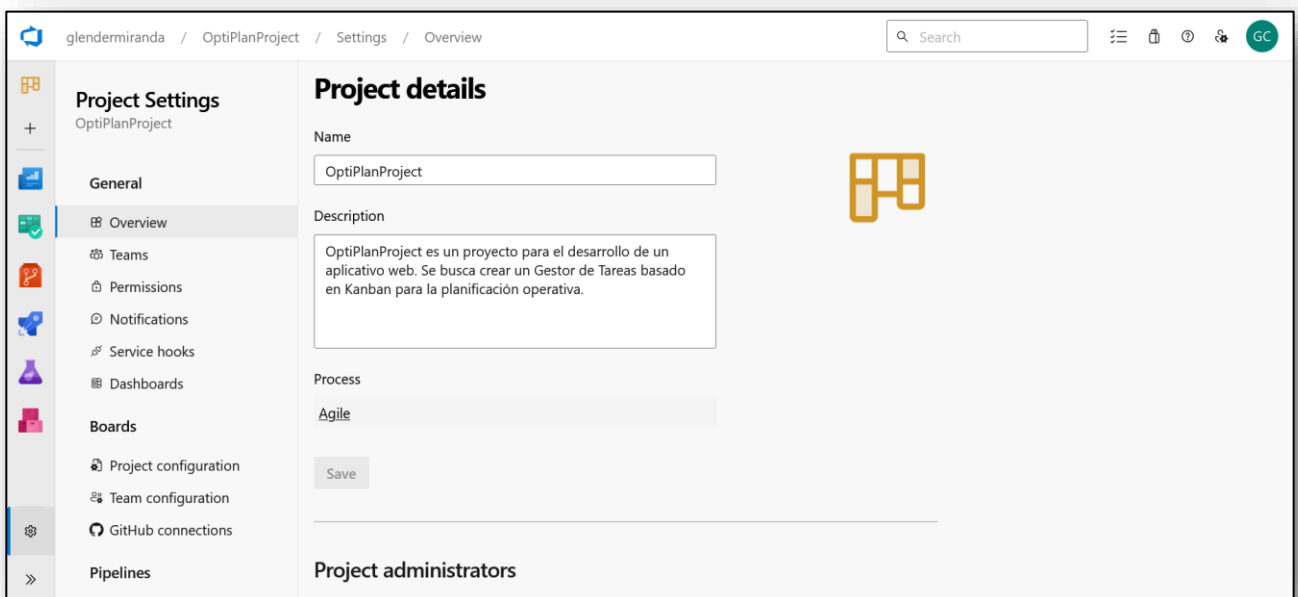


Figura 31. Configuración básica del proyecto Optiplan

En la sección Equipos, como lo muestra la Figura 32, se definen los miembros de los equipos del proyecto. Estos equipos pueden ser gestionados más tarde para conceder permisos a todos los miembros de manera conjunta.

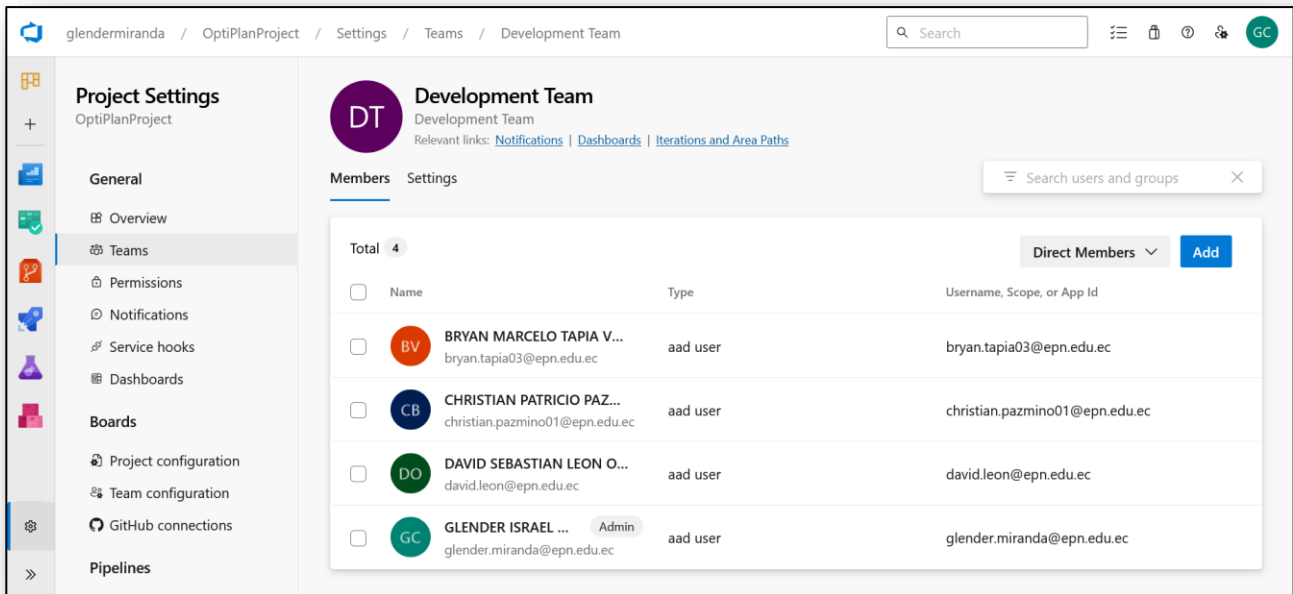


Figura 32. Equipos del proyecto Optiplan

En la sección “Permisos”, se encuentran las configuraciones de los permisos otorgados a los miembros o equipos para acceder a realizar ciertas acciones del proyecto. Como se muestra en la Figura 33, el equipo de desarrollo tiene establecido una configuración de permisos, esta configuración está realizada de tal manera que el equipo de desarrollo pueda realizar las opciones detalladas en la Tabla 4.

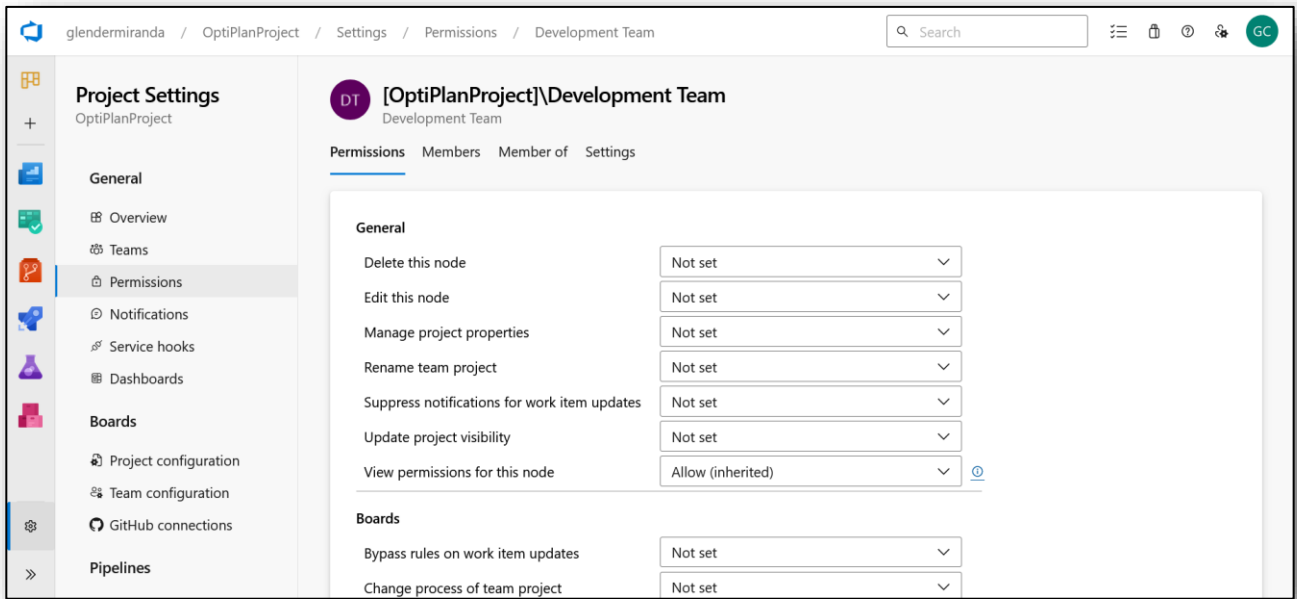


Figura 33. Permisos del proyecto Optiplan

La Tabla 4 detalla los permisos de Azure DevOps organizados en cuatro categorías principales: General, Boards, Analytics y Test Plans. Cada fila de la tabla incluye el nombre del permiso, su valor (ya sea "Allow" para permitido o "No Set" para no establecido), y una breve descripción de lo que implica el permiso. Por ejemplo, en la categoría General, el permiso "View permissions for this node" está permitido, lo que significa que los usuarios pueden ver los permisos asignados a ese nodo.

Tabla 4. Permisos del equipo del proyecto Optiplan

Nombre	Valor	Descripción
General		
Delete this node	No Set	Eliminar este nodo
Edit this node	No Set	Editar este nodo
Manage project properties	No Set	Gestionar propiedades del proyecto
Rename team project	No Set	Renombrar proyecto del equipo
Suppress notifications for work item updates	No Set	Suprimir notificaciones para actualizaciones de elementos de trabajo
Update project visibility	No Set	Actualizar visibilidad del proyecto
View permissions for this node	Allow	Ver permisos para este nodo

Boards		
Bypass rules on work item updates	No Set	Omitir reglas en actualizaciones de elementos de trabajo
Change process of team project	No Set	Cambiar el proceso del proyecto del equipo
Create tag definition	Allow	Crear definición de etiqueta
Delete and restore work items	Allow	Eliminar y restaurar elementos de trabajo
Move work items out of this project	No Set	Mover elementos de trabajo fuera de este proyecto
Permanently delete work items	No Set	Eliminar permanentemente elementos de trabajo
Analytics		
Delete shared Analytics views	Allow	Eliminar vistas compartidas de Analytics
Edit shared Analytics views	Allow	Editar vistas compartidas de Analytics
View analytics	Allow	Ver Analytics
Test Plans		
Create test runs	Allow	Crear ejecuciones de prueba
Delete test runs	Allow	Eliminar ejecuciones de prueba
Manage test configurations	Allow	Gestionar configuraciones de prueba
Manage test environments	Allow	Gestionar entornos de prueba
View test runs	Allow	Ver ejecuciones de prueba

Los repositorios del proyecto están configurados con opciones que permiten manejar de la mejor manera cada uno de ellos. La Figura 34 presenta la sección de configuración del repositorio de backend, esta configuración es la misma en el repositorio de frontend.

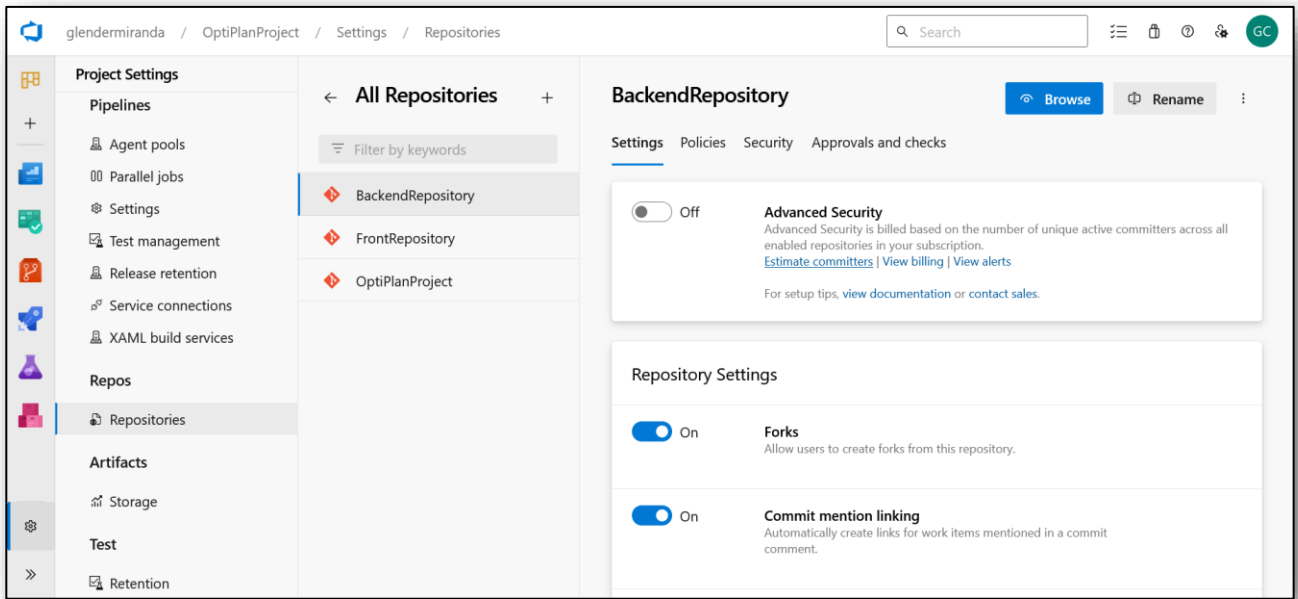


Figura 34. Configuración de los repositorios del proyecto Optiplan

La Tabla 5 presenta una serie de configuraciones de repositorio con sus respectivos valores y descripciones. Estas configuraciones permiten a los usuarios realizar diversas acciones dentro de un repositorio.

Las configuraciones incluyen permisos para crear forks, enlazar y resolver elementos de trabajo en comentarios de commits, y gestionar permisos de ramas. También incluye preferencias de transición de elementos de trabajo, modo de voto estricto, y la herencia del modo de creación de pull requests.

Tabla 5. Configuración de los repositorios

Nombre	Valor	Descripción
Forks	On	Permitir a los usuarios crear forks desde este repositorio.
Commit mention linking	On	Crear automáticamente enlaces para los elementos de trabajo mencionados en un comentario de commit.
Commit mention work item resolution	On	Permitir que las menciones en los comentarios de commits cierren elementos de trabajo (por ejemplo, "Fixes #123").

Work item transition preferences	On	Recordar las preferencias del usuario para completar elementos de trabajo con pull requests.
Permissions management	On	Permitir a los usuarios gestionar permisos para las ramas que ellos crearon.
Strict Vote Mode	Off	Habilitar el Modo de Voto Estricto para el repositorio, lo cual requiere permiso de Contribuir para votar en Pull Requests.
Inherit PR creation mode	On	Cuando está habilitado, el modo de creación de nuevas pull requests se hereda del proyecto. Cuando está deshabilitado, el modo de creación de nuevas pull requests se establece en el nivel actual y puede diferir de la configuración del proyecto.

Dentro de las configuraciones de los repositorios, se encuentran las políticas de los repositorios. Estas políticas se aplican para garantizar la calidad del código, la seguridad y apearse a estándares del proyecto. Una vez que alguna política es implementada, los cambios que se realicen hacia la rama con la política aplicada deberán realizarse mediante Pull Requests.

La primera política implementada es “Requerir un mínimo número de revisores”. Esta política se asegura que, al realizar un Pull Request, debe ser necesario que al menos un miembro del equipo apruebe dicha solicitud de cambios. La Figura 35 muestra la política implementada en la rama “development”. Las ramas “master” y “development”, tanto para frontend y backend, están configuradas de la misma manera.

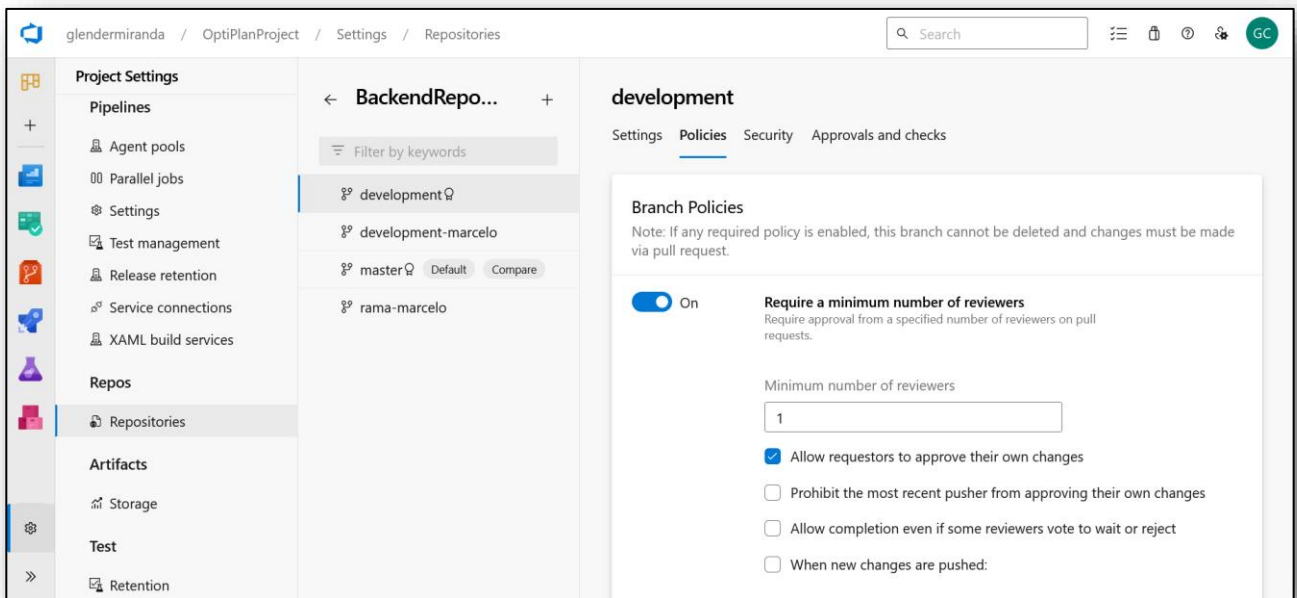


Figura 35. Política: Requerir un mínimo número de revisores

La segunda política implementada es la “Validación de la construcción”. Esta política se asegura que el aplicativo este correctamente construido y haya pasado por el proceso de verificación de la calidad del código. Al realizarse esta política, se ejecuta el pipeline de construcción de la aplicación, ya sea de frontend o backend. De esta forma, se puede garantizar que el código que se va a subir a la rama no tenga errores de sintaxis. La muestra la implementación de esta política en los repositorios del proyecto.

La tercera política implementada es “Incluir revisores automáticamente”. Esta política añade revisores que tengan que ver con el repositorio. En el caso del backend, se añade al o los desarrolladores involucrados con la creación de código de dicho repositorio. La presenta uno de los miembros del equipo involucrado del desarrollo backend.

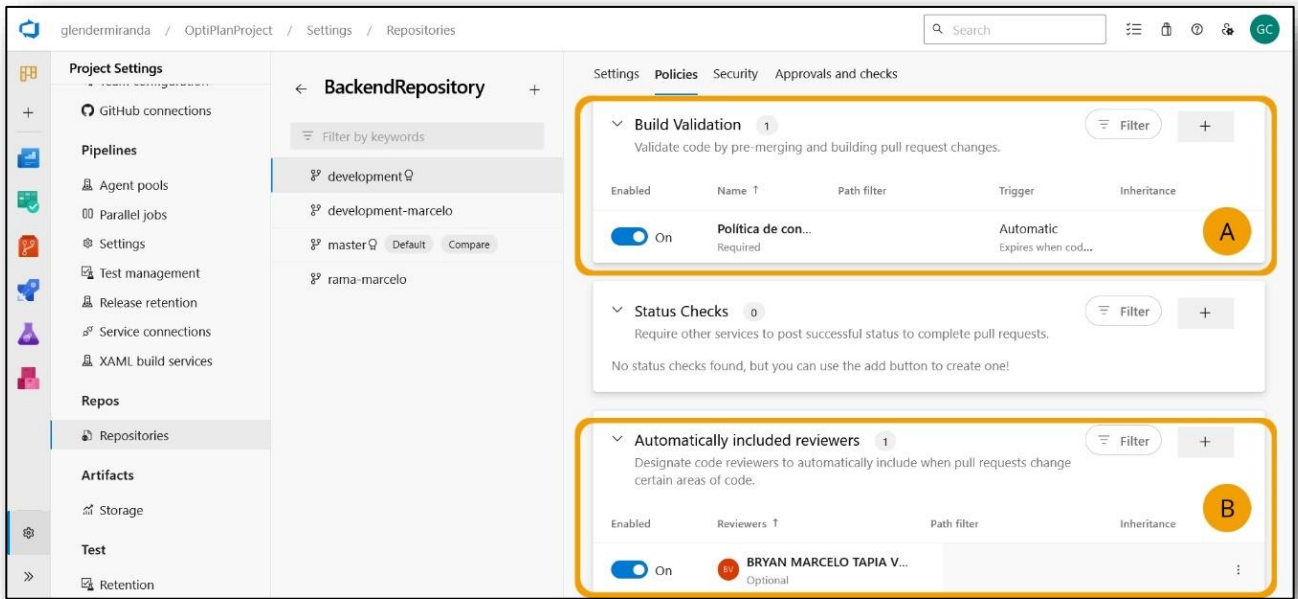


Figura 36. Política: Validación de construcción e inclusión automática de revisores

2.4. Verificación de la implementación de las prácticas y principios DevOps en el desarrollo de Optiplan

En el transcurso de este proyecto, se colaboró activamente en el desarrollo del aplicativo, trabajando estrechamente con el equipo de desarrollo como un miembro más. Además de contribuir al desarrollo, el objetivo fue asegurar que las prácticas y principios DevOps se implementaran correctamente. También se verificó que las integraciones y despliegues continuos se realizaran sin contratiempos.

A continuación, se presentarán las actividades de Scrum realizadas, así como la implementación y verificación de las prácticas y principios DevOps.

2.4.1. Actividades de Scrum

2.4.1.1. Planificación del producto

En la planificación del producto, se realizó el análisis del área de valor de los stakeholders, la visión del producto, el producto backlog de alto nivel y el producto roadmap. A continuación, se presentan algunos de estos aspectos importantes de la planificación.

Visión del producto

La visión del producto es una descripción de todas las áreas en las que los interesados pueden llegar a obtener valor [4]. Para nuestro producto, la visión se presenta a continuación:

*Para las entidades públicas que buscan gestionar su planificación operativa, reducir tiempos de entrega y aumentar la transparencia de sus proyectos, el **Gestor Operativo Optiplan** es una aplicación web para administrar las tareas operativas mediante tableros Kanban que facilita el flujo de las tareas mediante un enfoque visual y colaborativo a diferencia de las hojas de cálculo en Microsoft Excel y otros softwares genéricos de planificación como Jira, Trello o Asana.*

Nuestro producto permite una gestión y supervisión integral de tareas desde la gerencia hasta el nivel operativo, con funcionalidades como el control del flujo de trabajo, notificaciones, carga de archivos a la tareas y seguimiento continuo del progreso de las tareas del proyecto.

Product Backlog de Alto Nivel

En la Tabla 6 se muestran las historias de usuario de alto nivel, también llamadas épicas, para el proyecto. Esta visión permite tener una idea general de las funcionalidades que se van a desarrollar.

Tabla 6. Product backlog de alto nivel

	Título	Descripción
1	Gestión Integral de Tableros Kanban	Como usuario quiero tener una gestión integral de mis tableros Kanban para organizarlos de manera lógica.
2	Gestión Integral de Tareas	Como usuario quiero gestionar tareas de los proyectos para controlar su flujo a través de las columnas del tablero Kanban.
3	Control y Priorización de Tareas	Como usuario quiero tener una vista completa y ordenada de las tareas pendientes en cada proyecto para priorizar y gestionar el trabajo.
4	Control y Seguimiento del Flujo de Trabajo	Como usuario quiero utilizar un gestor visual de tareas para evaluar visualmente el progreso de las tareas.
5	Sistema de Notificaciones	Como usuario quiero recibir notificaciones sobre la finalización de tareas con algún comentario adicional para controlar el progreso y los cambios en el proyecto.
6	Gestión de Documentos	Como usuario quiero poder cargar archivos a las tareas para tener una documentación de respaldo.
7	Acceso Basado en Roles	Como administrador quiero un acceso basado en roles jerárquicos para asegurar que cada usuario tenga permisos adecuados según su rol.
8	Generación de Reportes	Como usuario quiero generar reportes y visualizar gráficas sobre el progreso y desempeño del proyecto para tomar decisiones informadas.

2.4.1.2. Planificación del release

La planificación del release fue dividida en dos releases, cada uno con una duración de siete sprints. En la Tabla 7 y la Tabla 8 se muestran las fechas de realización de ambos releases, tomando en cuenta que la fecha de finalización de todo el proyecto es el 4 de julio de 2024.

Tabla 7. Planificación del Release 1

	Fecha de inicio	Fecha de fin
RELEASE 1	16/11/2023	29/2/2024
Sprint 1	16/11/2023	29/11/2023
Sprint 2	30/11/2023	14/12/2023
Sprint 3	15/12/2023	4/1/2024
Sprint 4	5/1/2024	18/1/2024
Sprint 5	19/1/2024	1/2/2024
Sprint 6	2/2/2024	15/2/2024
Sprint 7	16/2/2024	29/2/2024

Tabla 8. Planificación del Release 2

	Fecha de inicio	Fecha de fin
RELEASE 2	26/3/2024	4/7/2024
Sprint 8	26/3/2024	11/4/2024
Sprint 9	12/4/2024	25/4/2024
Sprint 10	26/4/2024	9/5/2024
Sprint 11	10/5/2024	23/5/2024
Sprint 12	24/5/2024	6/6/2024
Sprint 13	7/6/2024	20/6/2024
Sprint 14	21/6/2024	4/7/2024

2.4.1.3. Planificación del sprint

En la Tabla 9 se presenta cada uno de los objetivos definidos para cada sprint. Cada objetivo permite la selección de historias de usuario a trabajar en cada sprint, de manera que se prioriza el trabajo del equipo enfocado en el objetivo.

Tabla 9. Planificación de los sprints

SPRINT	OBJETIVO
Sprint 1	Crear tableros Kanban
Sprint 2	Crear flujos de tareas en Kanban
Sprint 3	Controlar el acceso a los tableros Kanban
Sprint 4	Mejorar el flujo de trabajo de los tableros Kanban
Sprint 5	Crear proyectos para las tareas operativas
Sprint 6	Mejorar los flujos de trabajo de las tareas operativas
Sprint 7	Visualizar proyectos y tareas operativas
Sprint 8	Descubrir el flujo de tareas para todos los roles
Sprint 9	Gestionar las tareas del Gestor Empresarial
Sprint 10	Visualizar el flujo de tareas del Gestor Empresarial
Sprint 11	Gestionar las tareas del Gestor de Área
Sprint 12	Visualizar el flujo de tareas del Gestor de Área
Sprint 13	Gestionar las tareas del Líder de Área
Sprint 14	Gestionar la ejecución de tareas del Técnico

2.4.2. Implementación

En esta sección se detalla la metodología aplicada durante el desarrollo del proyecto, centrándose en la implementación de prácticas y principios de DevOps. La metodología especifica las actividades realizadas.

Investigación y elección de herramienta DevOps

En el sprint 1 se inició una investigación exhaustiva de herramientas DevOps disponibles en el mercado para identificar cuáles serían más adecuadas para el proyecto.

Se llevó a cabo un análisis comparativo de diversas herramientas DevOps, considerando aspectos como funcionalidad, integración con otros servicios, facilidad de uso, costo y soporte. Se evaluaron herramientas como Jenkins, Azure DevOps, GitLab CI/CD, y AWS

CodePipelines. Se realizaron reuniones con el equipo para discutir las opciones y se consultaron casos de estudio y opiniones de la comunidad.

Esta actividad aplicó las siguientes prácticas y principios:

- **Colaboración y Comunicación:** Se promovió una comunicación activa y continua entre los miembros del equipo para tomar decisiones informadas.

Configuración de los Ambientes de Trabajo en las Máquinas de Desarrollo

En el sprint 1, el equipo de desarrollo llevó a cabo la configuración de los entornos de trabajo en sus máquinas locales. Esta acción fue esencial para garantizar la consistencia entre los entornos de desarrollo, pruebas y producción, y para adherirse al principio de paridad.

La configuración de los entornos de trabajo en las máquinas de desarrollo incluyó la instalación y configuración de todas las herramientas necesarias para el desarrollo del proyecto, así como la configuración de variables de entorno y dependencias específicas del proyecto. Este proceso se llevó a cabo para asegurar que todos los desarrolladores trabajaran en entornos idénticos, minimizando las diferencias que pudieran surgir entre los distintos ambientes.

Los pasos principales realizados incluyeron:

- **Instalación de Herramientas de Desarrollo:** Se instalaron herramientas esenciales como Node.js, Docker, y los entornos de desarrollo integrados (IDEs) recomendados, como Visual Studio Code.
- **Configuración de Variables de Entorno:** Se definieron y configuraron variables de entorno necesarias para el proyecto, como las URLs del backend, claves secretas, y configuraciones específicas para diferentes servicios utilizados.
- **Clonación de Repositorios:** Se clonaron los repositorios de código del proyecto desde el sistema de control de versiones, asegurando que todos los desarrolladores trabajaran con el mismo código base.
- **Instalación de Dependencias:** Se ejecutaron los comandos necesarios para instalar todas las dependencias del proyecto.
- **Configuración de Docker:** Se configuraron contenedores Docker para asegurar que todos los servicios necesarios, como bases de datos y servicios de terceros,

estuvieran disponibles y configurados de manera consistente en todas las máquinas de desarrollo.

- **Configuración de Bases de Datos Locales:** Se configuraron instancias locales de bases de datos para permitir pruebas y desarrollo sin afectar las bases de datos de producción.

Esta actividad aplicó las siguientes prácticas y principios:

- **Consistencia:** Al asegurar que todos los desarrolladores trabajaran en entornos idénticos, se minimizó la posibilidad de errores y discrepancias entre los diferentes ambientes de trabajo.
- **Paridad:** La paridad entre los entornos de desarrollo, pruebas y producción se mantuvo al asegurar que las configuraciones fueran consistentes en todos los entornos, lo que facilita la detección y solución de problemas de manera temprana.

Configuración de Repositorios y Gestión de Código

En el sprint 1 se configuraron los repositorios de código fuente y se establecieron políticas de gestión de ramas.

Se crearon repositorios en GitHub y se configuraron políticas de ramas para garantizar una gestión eficiente del código. Se establecieron ramas principales como 'master' y 'development' donde se mantuvo el código del proyecto.

En la Figura 37 y Figura 38 se muestran los repositorios de cada uno de los componentes de desarrollo.

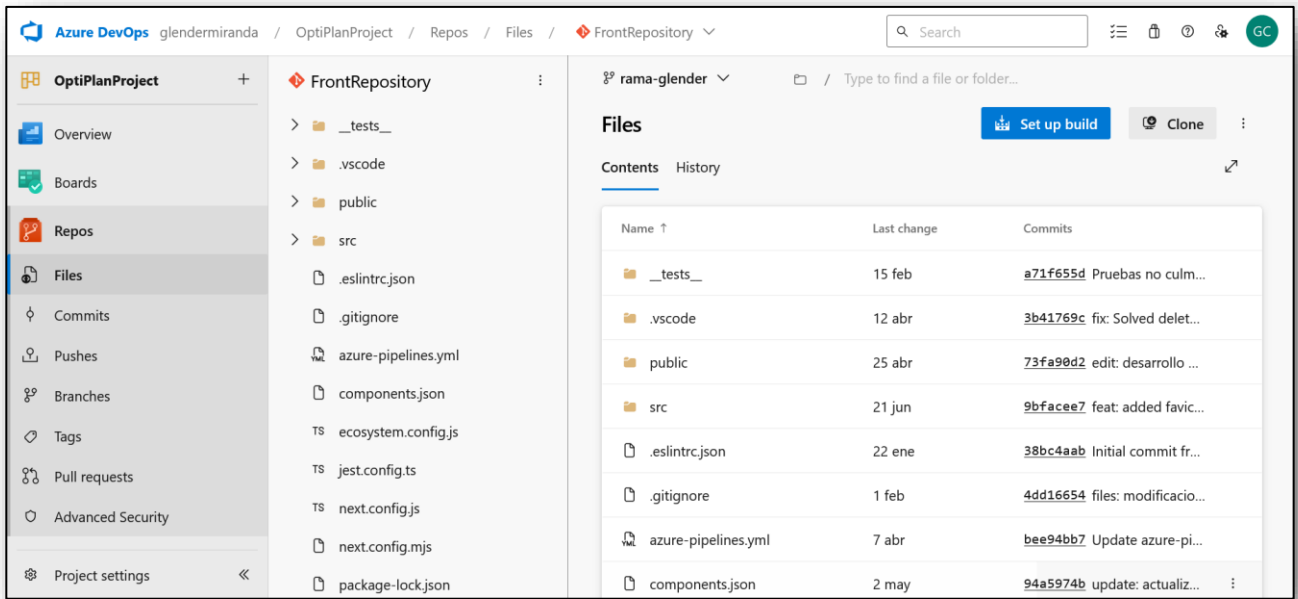


Figura 37. Repositorio para Frontend en Azure DevOps

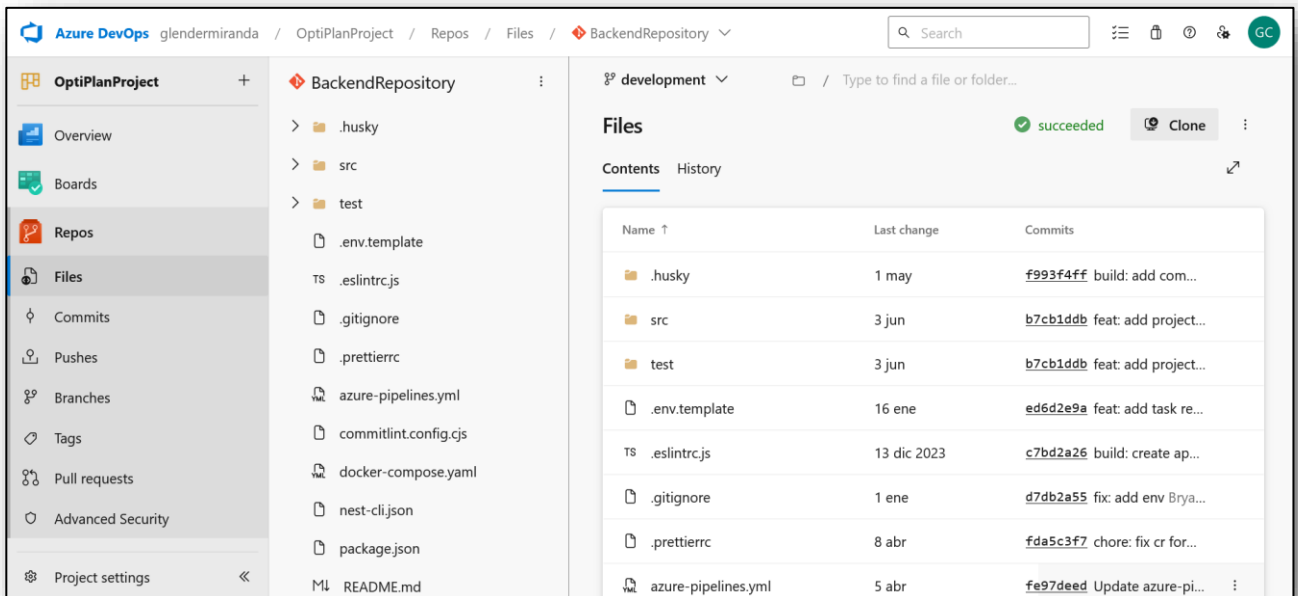


Figura 38. Repositorio para Backend en Azure DevOps

Dentro de cada repositorio de configuraron las políticas necesarias, las cuales se listan a continuación:

Permitir a los usuarios administrar los permisos de sus propias ramas: Esta política permite que los miembros del equipo puedan realizar cualquier acción a las ramas que han creado.

Requerir un mínimo número de revisores: Esta política se definió para las ramas por defecto “master” para asegurarse que, al realizar cambios, se requiera de al menos la aprobación de dos usuarios para realizar dichas modificaciones.

Revisar los elementos de trabajo enlazados: Esta política permite crear una trazabilidad con las tareas de implementación al realizar pull requests.

Política de construcción de la aplicación: Esta política se asegura de que el cambio que se va a realizar es seguro para desplegar. La política apunta a un pipeline simple con las instrucciones para construir el aplicativo.

Esta actividad aplicó las siguientes prácticas y principios:

Control de Versiones: Uso de Git para el control de versiones, asegurando un historial claro y rastreable de cambios.

Revisión de Código: Implementación de revisiones de código para mantener la calidad y coherencia del código.

Configuración de Pipelines de construcción y despliegue para los ambientes de Desarrollo.

En el sprint 3, se enfocó en la configuración de los pipelines de CI/CD para los entornos de desarrollo.

Se realizó una configuración de pipelines utilizando Azure DevOps para la integración y despliegue continuo. Se definieron pasos para la compilación y despliegue en el entorno de desarrollo.

En un principio, tanto la fase de construcción como de despliegue se encontraban en el mismo pipeline, por lo que dentro del código YML del pipeline se definió una etapa llamada “Deploy” para desplegar el Web App. A continuación, se detalla la sección del código en específico de dicha fase en el pipeline para backend en la Figura 39:

```

1 - stage: Deploy
2   displayName: Deploy stage
3   dependsOn: Build
4   condition: succeeded()
5   jobs:
6     - deployment: Deploy
7       displayName: Deploy
8       environment: $(environmentName)
9
10      pool:
11        vmImage: $(vmImageName)
12      strategy:
13        runOnce:
14          deploy:
15            steps:
16              - task: DownloadBuildArtifacts@1
17                displayName: "Download the artifact"
18                inputs:
19                  buildType: "current"
20                  downloadType: "specific"
21                  itemPattern: "**/*.zip"
22                  downloadPath: "$(System.ArtifactsDirectory)"
23
24              - task: AzureWebApp@1
25                displayName: "Azure Web App Deploy:"
26                env:
27                  DB_HOST: $(ELP_HOST)
28                  DB_NAME: $(ELP_NAME)
29                  DB_PASSWORD: $(ELP_PASSWORD)
30                  DB_PORT: $(ELP_PORT)
31                  DB_USERNAME: $(ELP_USERNAME)
32                  STAGE: $(STAGE)
33                  JWT_SECRET: $(JWT_SECRET)
34
35                inputs:
36                  azureSubscription: $(azureSubscription)
37                  appType: webAppLinux
38                  appName: $(webAppName)
39                  runtimeStack: "NODE|18"
40                  package: $(System.ArtifactsDirectory)/**/*.zip
41                  startUpCommand: "pm2 start /home/site/wwwroot/a/dist/main.js --no-daemon"

```

Figura 39. Etapa de despliegue en el pipeline de backend

Dentro del Código proporcionado se indica la fase de despliegue del aplicativo. En un principio, el aplicativo se desplegaba solamente en los ambientes de desarrollo. A partir de la línea 5 a la línea 11, se detallan las características de la fase, como el nombre y la imagen del sistema operativo que se va a utilizar para desplegar el aplicativo. De la línea 16 a la 22, se define el artefacto que va a ser desplegado, en este caso dicho artefacto es el resultado del aplicativo construido en fases anteriores. A partir de la línea 24, se definen las variables de entorno necesarias para desplegar el aplicativo y los valores de entrada

tales como el id de la suscripción de Azure, el ambiente de aplicación, el paquete y el comando de inicio del aplicativo.

Esta actividad aplicó las siguientes prácticas y principios:

- **Integración Continua:** Automatización de la integración del código en el repositorio central varias veces al día.
- **Despliegue Continuo:** Automatización del despliegue de aplicaciones en entornos de desarrollo para pruebas inmediatas.

Resolución de problemas con el despliegue de backend

En el sprint 3, se detectó un problema crítico relacionado con la función “uuid_generate_v4” en la base de datos PostgreSQL. Este problema afectaba la generación de UUIDs, lo que es esencial para la identificación única de registros en la base de datos. Se evaluaron diversas opciones para solucionar este problema y asegurar el correcto funcionamiento de la base de datos y la aplicación en general.

El problema específico estaba relacionado con la función “uuid_generate_v4”, utilizada para generar identificadores únicos universales (UUIDs). La Figura 40 muestra el error en la consola de la secuencia de registro del aplicativo web en Azure. El problema se originaba debido a la falta de la extensión uuid-osspl en la configuración del servidor de la base de datos Postgres.

```

2024-01-10T17:02:02.695543015Z: [INFO] 17:02:02 0|main | QueryFailedError: function uuid_generate_v4() does not exist
2024-01-10T17:02:02.745477111Z: [INFO] 17:02:02 0|main |   at PostgresQueryRunner.query (/home/site/wwwroot/a/dist
/src/driver/postgres/PostgresQueryRunner.ts:331:19)
2024-01-10T17:02:02.745506112Z: [INFO] 17:02:02 0|main |   at processTicksAndRejections (node:internal/process
/task_queues:96:5)
2024-01-10T17:02:02.745512412Z: [INFO] 17:02:02 0|main |   at PostgresQueryRunner.executeQueries (/home/site/wwwroot
/a/dist/src/query-runner/BaseQueryRunner.ts:660:13)
2024-01-10T17:02:02.745517712Z: [INFO] 17:02:02 0|main |   at PostgresQueryRunner.createTable (/home/site/wwwroot/a/dist
/src/driver/postgres/PostgresQueryRunner.ts:605:9)
2024-01-10T17:02:02.745522512Z: [INFO] 17:02:02 0|main |   at RdbmsSchemaBuilder.createNewTables (/home/site/wwwroot
/a/dist/src/schema-builder/RdbmsSchemaBuilder.ts:632:13)
2024-01-10T17:02:02.745527712Z: [INFO] 17:02:02 0|main |   at
RdbmsSchemaBuilder.executeSchemaSyncOperationsInProperOrder (/home/site/wwwroot/a/dist/src/schema-builder
/RdbmsSchemaBuilder.ts:225:9)
2024-01-10T17:02:02.745532412Z: [INFO] 17:02:02 0|main |   at RdbmsSchemaBuilder.build (/home/site/wwwroot/a/dist
/src/schema-builder/RdbmsSchemaBuilder.ts:95:13)
2024-01-10T17:02:02.745537013Z: [INFO] 17:02:02 0|main |   at DataSource.synchronize (/home/site/wwwroot/a/dist
/src/data-source/DataSource.ts:339:9)
2024-01-10T17:02:02.745541513Z: [INFO] 17:02:02 0|main |   at DataSource.initialize (/home/site/wwwroot/a/dist/src/data-
source/DataSource.ts:277:43)

```

Figura 40. Consola de monitorización: Error con `uuid_generate_v4`

Para abordar este problema, se consideraron las siguientes opciones:

- **Reinstalación de “`uuid_generate_v4`”:** Volver a instalar la función podría resolver problemas temporales de configuración o corrupción.
- **Crear una base de datos Postgres limpia:** Esto aseguraría que cualquier problema subyacente con la base de datos actual se elimine.
- **Crear una función de generación de IDs propios:** Implementar una solución personalizada para generar UUIDs.
- **Migrar a otro tipo de base de datos:** Evaluar la posibilidad de cambiar a una base de datos diferente que ofrezca mejor soporte o rendimiento.
- **Utilizar VM para levantar los servidores:** Levantar los servidores de base de datos en máquinas virtuales podría ofrecer un control más granular sobre la configuración y las extensiones instaladas.

Finalmente, junto con el equipo de desarrollo, se determinó que la solución más eficiente y menos disruptiva era habilitar la extensión “`uuid-oss`” en la configuración del servidor de la base de datos (ver Figura 41). Esta extensión es necesaria para que la función “`uuid_generate_v4`” funcione correctamente.

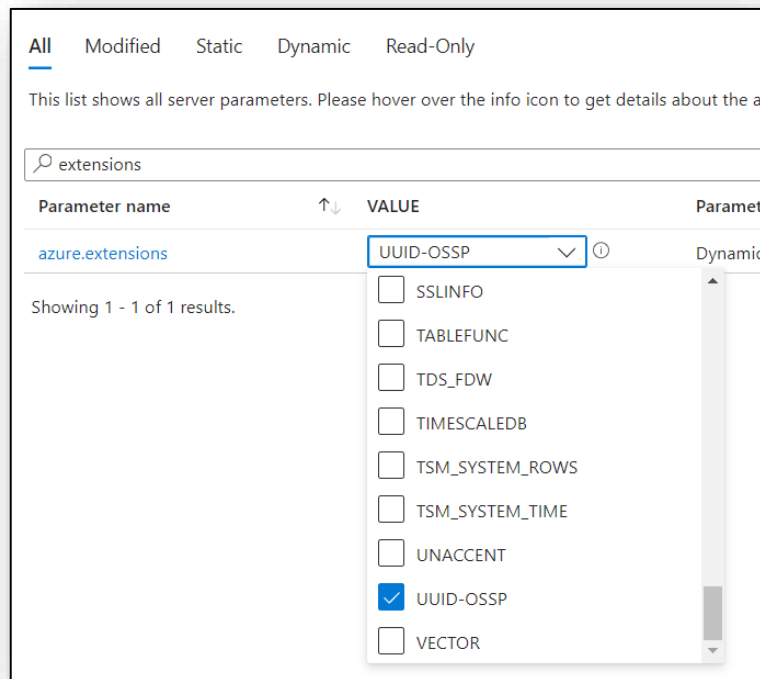


Figura 41. Adición de la extensión UUID_OSSP a la base de datos

Esta actividad aplicó las siguientes prácticas y principios:

Automatización: Asegurar que la configuración de la extensión “uuid-oss” se mantenga como parte del script de aprovisionamiento de la base de datos, automatizando su instalación en cualquier despliegue futuro.

Manejo de Configuración: Asegurar que todas las configuraciones críticas, como la habilitación de extensiones, estén bien documentadas y gestionadas en los scripts de configuración del servidor.

Feedback Continuo: Resolver el problema de manera eficiente permitió que el equipo recibiera feedback rápidamente sobre la efectividad de la solución, asegurando que el sistema volviera a funcionar correctamente sin interrupciones prolongadas.

Resolución de problemas en tiempo de ejecución.

En el sprint 3, el equipo de desarrollo identificó un problema crítico relacionado con el tiempo de respuesta del backend al iniciarse en el entorno de Azure. El tiempo de respuesta superaba los 230 segundos máximos permitidos por la arquitectura de Azure, lo que

resultaba en fallos de despliegue y disponibilidad del servicio. La muestra la consola de monitorización donde se especifica el error en tiempo de ejecución.

El problema surgía porque el backend no estaba configurando el puerto de escucha mediante una variable de entorno específica de Azure. En lugar de permitir que Azure configurara automáticamente el puerto 8080, el backend estaba intentando configurar su propio puerto, lo que causaba demoras significativas en el tiempo de respuesta.

```
[38;5;3m[NestApplication] [39m[32mNest application successfully started[39m[38;5;3m +30ms[39m
2024-01-13T01:49:49.055Z ERROR - Container backendoptiplanbd_1_2c3173df for site backendoptiplanbd did not start within
expected time limit. Elapsed time = 230.8867404 sec
2024-01-13T01:49:49.064Z ERROR - Container backendoptiplanbd_1_2c3173df didn't respond to HTTP pings on port: 8080, failing
site start. See container logs for debugging.
2024-01-13T01:49:50.003Z INFO - Stopping site backendoptiplanbd because it failed during startup.
```

Figura 42. Consola de monitorización: Error con el tiempo de espera del recurso

Para solucionar este problema, se decidió ajustar la configuración del backend para que no especificara ningún puerto explícitamente, permitiendo que Azure gestionara la configuración del puerto por defecto. Azure establece el puerto 8080 como el puerto de escucha predeterminado al detectar que no se ha configurado una variable de entorno para el puerto.

Esta actividad aplicó las siguientes prácticas y principios:

- **Automatización y Configuración Correcta:** Asegurar que la configuración del puerto sea gestionada automáticamente por Azure, reduciendo el riesgo de errores de configuración manual.
- **Eficiencia:** Al eliminar la configuración manual del puerto, se redujo significativamente el tiempo de respuesta del backend, asegurando que se inicie dentro del tiempo permitido por la arquitectura de Azure.
- **Paridad de Ambientes:** Asegurar que la configuración del entorno de producción en Azure sea consistente con las mejores prácticas y configuraciones predeterminadas, manteniendo la paridad entre los entornos de desarrollo y producción.
- **Monitoreo y Mejora Continua:** Implementar un monitoreo constante del tiempo de respuesta y realizar ajustes necesarios para asegurar el rendimiento óptimo del backend.

Implementación de mejoras en los Pipelines

En el cuarto sprint, se realizaron optimizaciones a los pipelines de integración y despliegue para mejorar el tiempo en el que el artefacto realizara el proceso de carga al ambiente de desarrollo.

Dentro de los pipelines de integración del backend, existía una tarea que realizaba el proceso de despliegue del aplicativo al entorno de desarrollo. El problema con esta tarea era antes de desplegar debía descargar el artefacto directamente de la carpeta 'drop', dicho directorio es una carpeta donde se almacenan los artefactos construidos en las fases anteriores. El aplicativo, al construirse, creaba una carpeta con todas las dependencias necesarias, a continuación, se creaba el artefacto, comprimiendo en formato ZIP la carpeta con las dependencias. Este proceso mantenía la carpeta original dentro de 'drop', generando que, al descargarse todo el artefacto, se descargara también la carpeta sin comprimir de las dependencias. Esto ocasionaba que el despliegue llegase a demorarse hasta una hora.

Para resolver esta situación se planteó el uso de una nueva tarea en el pipeline que elimine de los artefactos la carpeta de dependencias sin comprimir. A continuación, se presenta la sección mencionada en la :

```
1 - task: DeleteFiles@1
2 displayName: 'Delete dist folder from ArtifactStagingDirectory'
3 inputs:
4   SourceFolder: '$(Build.ArtifactStagingDirectory)'
5   Contents: '/dist'
```

Figura 43. Tarea para eliminar artefactos innecesarios del pipeline

En la línea 4 del código, se define la carpeta donde se desea eliminar el contenido y en la línea 5 se define el directorio en específico a eliminar.

De esta manera los pipelines de integración continua generan un artefacto más limpio, permitiendo al despliegue un mejor rendimiento para subir las nuevas características a los recursos.

Esta actividad aplicó las siguientes prácticas y principios:

Automatización: La adición de la tarea DeleteFiles@1 automatiza la eliminación de archivos innecesarios, asegurando que solo se desplieguen los artefactos necesarios. Esto reduce el tiempo de despliegue y minimiza el riesgo de errores manuales.

Integración Continua: Manteniendo un pipeline limpio y eficiente, se asegura que las nuevas versiones del aplicativo se integren y desplieguen de manera continua y rápida, facilitando la entrega continua.

Feedback Continuo: La implementación de esta optimización permite recibir feedback más rápido sobre los despliegues, lo que contribuye a un ciclo de desarrollo más ágil y responsivo.

Verificación y monitorización en las herramientas avanzadas de los recursos.

En el Sprint 6 se realizó una monitorización de los recursos de los ambientes. Dentro de la configuración avanzada de los recursos de Azure es posible visualizar todas las herramientas avanzadas como se presenta en la Figura 44.

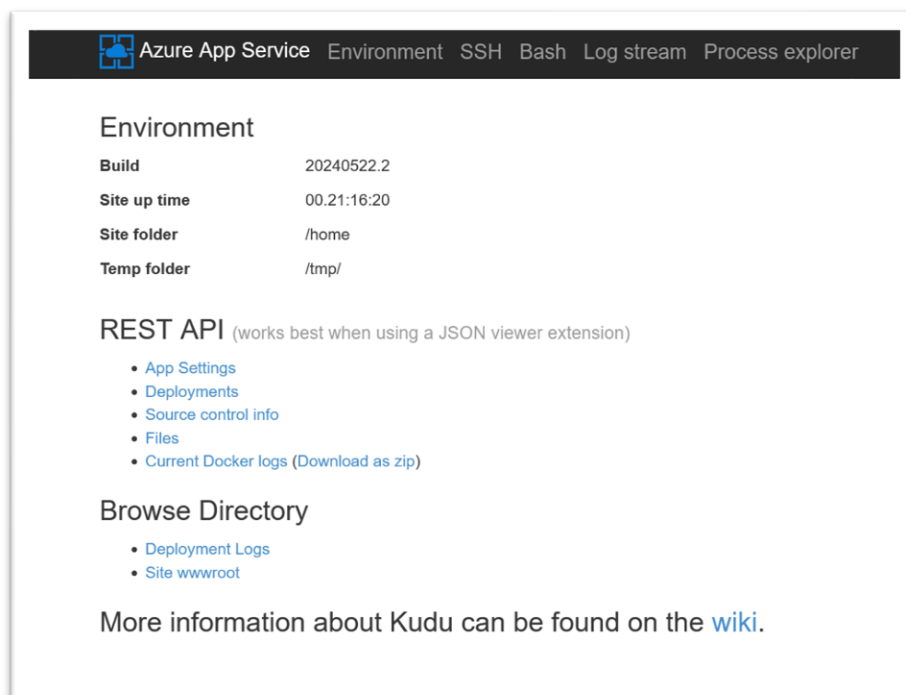
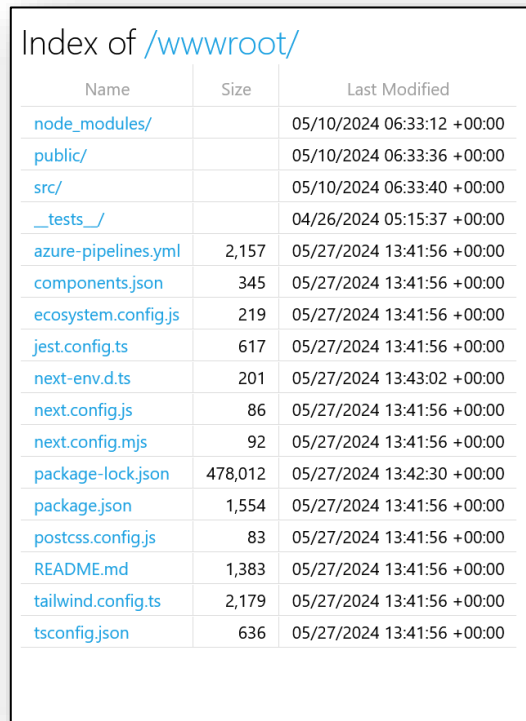


Figura 44. Herramientas avanzadas de los recursos de Azure

Dentro de estas herramientas avanzadas se encuentran dos de las opciones principales, el sitio y el Bash del recurso. El sitio del recurso muestra los archivos que se encuentran

desplegados y que permiten la ejecución de los servicios. La Figura 45 muestra los archivos que se encuentran en el recurso.



Index of /wwwroot/

Name	Size	Last Modified
node_modules/		05/10/2024 06:33:12 +00:00
public/		05/10/2024 06:33:36 +00:00
src/		05/10/2024 06:33:40 +00:00
__tests__		04/26/2024 05:15:37 +00:00
azure-pipelines.yml	2,157	05/27/2024 13:41:56 +00:00
components.json	345	05/27/2024 13:41:56 +00:00
ecosystem.config.js	219	05/27/2024 13:41:56 +00:00
jest.config.ts	617	05/27/2024 13:41:56 +00:00
next-env.d.ts	201	05/27/2024 13:43:02 +00:00
next.config.js	86	05/27/2024 13:41:56 +00:00
next.config.mjs	92	05/27/2024 13:41:56 +00:00
package-lock.json	478,012	05/27/2024 13:42:30 +00:00
package.json	1,554	05/27/2024 13:41:56 +00:00
postcss.config.js	83	05/27/2024 13:41:56 +00:00
README.md	1,383	05/27/2024 13:41:56 +00:00
tailwind.config.ts	2,179	05/27/2024 13:41:56 +00:00
tsconfig.json	636	05/27/2024 13:41:56 +00:00

Figura 45. Sitio del recurso de Azure

En la Figura 46, se proporciona el Bash del recurso. Esta es una consola de Linux por la cual es posible visualizar las configuraciones y archivos del sistema del recurso. Esta consola forma una parte importante para realizar comprobaciones de los archivos y disponibilidad del recurso.

```
Azure App Service Environment SSH Bash Log stream Process explorer

DEBUG CONSOLE | AZURE APP SERVICE ON LINUX

Documentation: http://aka.ms/webapp-linux
Kudu Version : 20240522.2
Commit      : 6ea5b2dd86b0954053a7e59b1638a4c8f096802f

kudu_ssh_user@frontprod- kudu_460fd68774:/$ ls
lsASP.NET DeploymentLogStream LogFiles      site ud66f9ef6acd52ff1829606
kudu_ssh_user@frontprod- kudu_460fd68774:/$ cd site/wwwroot
kudu_ssh_user@frontprod- kudu_460fd68774:~/site/wwwroot$ ls
README.md      components.json  next-env.d.ts   node_modules    postcss.config.js  tailwind.config.ts
tests          ecosystem.config.js  next.config.js  package-lock.json  public             tsconfig.json
azure-pipelines.yml  jest.config.ts  next.config.mjs  package.json     src
kudu_ssh_user@frontprod- kudu_460fd68774:~/site/wwwroot$
```

Figura 46. Bash del recurso de Azure

Esta actividad aplicó las siguientes prácticas y principios:

Monitoreo Continuo: La verificación y monitorización continua de los recursos en Azure permite detectar y solucionar problemas de manera proactiva, asegurando la estabilidad y el rendimiento óptimo de los servicios. Esto se alinea con las mejores prácticas de DevOps, donde el monitoreo constante es crucial para mantener la salud del sistema.

Transparencia y Visibilidad: Utilizar las herramientas avanzadas de Azure, como el sitio del recurso y el Bash del recurso, proporciona una visibilidad clara sobre el estado de los archivos y configuraciones del sistema. Esto asegura que el equipo tenga una comprensión completa del estado actual de los recursos y pueda tomar decisiones informadas.

Migración a NextJS

En el sprint 5, se decidió cambiar el framework utilizado para el frontend del proyecto, pasando de una tecnología anterior a NextJS. Este cambio se debió a la necesidad de mejorar la eficiencia y capacidad del frontend. La migración implicó la adaptación de los

pipelines de CI/CD para soportar la nueva estructura y procesos de construcción y despliegue.

Se realizó una investigación para conocer cómo realizar la construcción y despliegue del aplicativo en NextJS. A continuación, se realizaron los cambios pertinentes.

En cuanto a la construcción, el Framework NextJS usa el mismo ambiente y pasos para construir el aplicativo. A continuación, en la Figura 47, se presenta la fase de construcción del aplicativo frontend:

```
1  - stage: Build
2    displayName: Build stage
3    jobs:
4      - job: Build
5        displayName: Build
6        pool:
7          vmImage: $(vmImageName)
8
9        steps:
10       - task: NodeTool@0
11         inputs:
12           versionSpec: "20.x"
13           displayName: "Install Node.js"
14
15       - script: |
16         npm install
17         npm run build
18         displayName: "npm install and build"
19         env:
20           NEXT_PUBLIC_BACKEND_URL: $(NEXT_PUBLIC_BACKEND_URL)
21           NEXTAUTH_SECRET: $(NEXTAUTH_SECRET)
22
23       - task: ArchiveFiles@2
24         displayName: "Archive files"
25         inputs:
26           rootFolderOrFile: "$(System.DefaultWorkingDirectory)"
27           includeRootFolder: false
28           archiveType: zip
29           archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
30           replaceExistingArchive: true
31
32       - publish: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
33         artifact: drop
```

Figura 47. Cambio en el pipeline en la migración a NextJS

En las líneas 1-2, se establece la etapa de construcción y se le asigna un nombre de visualización. En las líneas 3-4, se define el trabajo dentro de esta etapa, también con un nombre de visualización. Luego, en las líneas 5-6, se especifica la imagen de la máquina virtual a utilizar para el trabajo, utilizando una variable definida previamente (`$(vmImageName)`). Las líneas 8-9 agregan una tarea para instalar Node.js, especificando la versión 20.x (`NodeTool@0`).

En las líneas 11-16, se ejecuta un script que instala las dependencias de Node.js (`npm install`) y construye la aplicación (`npm run build`), estableciendo variables de entorno necesarias para la construcción. Posteriormente, en las líneas 18-24, se añade una tarea para archivar los archivos generados durante la construcción en un archivo zip (`ArchiveFiles@2`). Finalmente, en las líneas 26-27, se publica el archivo zip generado como un artefacto en el pipeline. Esta configuración automatiza el proceso de instalación de dependencias, construcción y archivado de la aplicación para su posterior despliegue.

Esta actividad aplicó las siguientes prácticas y principios:

Automatización: La configuración del pipeline automatiza todo el proceso de instalación de dependencias, construcción y archivado, reduciendo la intervención manual y el riesgo de errores.

Integración Continua: La migración a NextJS se integró de manera continua en el flujo de trabajo existente, permitiendo una transición sin interrupciones y asegurando que cada cambio se construya y pruebe automáticamente.

Despliegue Continuo: Al publicar los artefactos construidos automáticamente, se facilita el despliegue continuo, mejorando la eficiencia y velocidad con que las nuevas características y correcciones se ponen en producción.

Flexibilidad y Adaptabilidad: La capacidad de adaptarse rápidamente a un nuevo framework como NextJS demuestra la flexibilidad del equipo y su disposición para adoptar tecnologías que mejoren el producto.

Feedback Continuo: Al mantener un pipeline eficiente y bien configurado, se recibe feedback rápido sobre la calidad y funcionalidad del código, permitiendo iterar y mejorar continuamente.

Cambio de enfoque a releases

En el segundo release, sprint 1, se tomó la decisión de cambiar el enfoque que se estaba siguiendo. Anteriormente, tanto las integraciones como los despliegues se estaban realizando en el mismo pipeline. Se optó por separar el encadenamiento de herramientas y los despliegues.

Dentro de Azure DevOps, se utilizó la sección “Releases”, donde se realizó la configuración de los despliegues y entregas continuas. La Figura 48 muestra los releases para frontend y backend para realizar los despliegues y entregas continuas de manera automática.

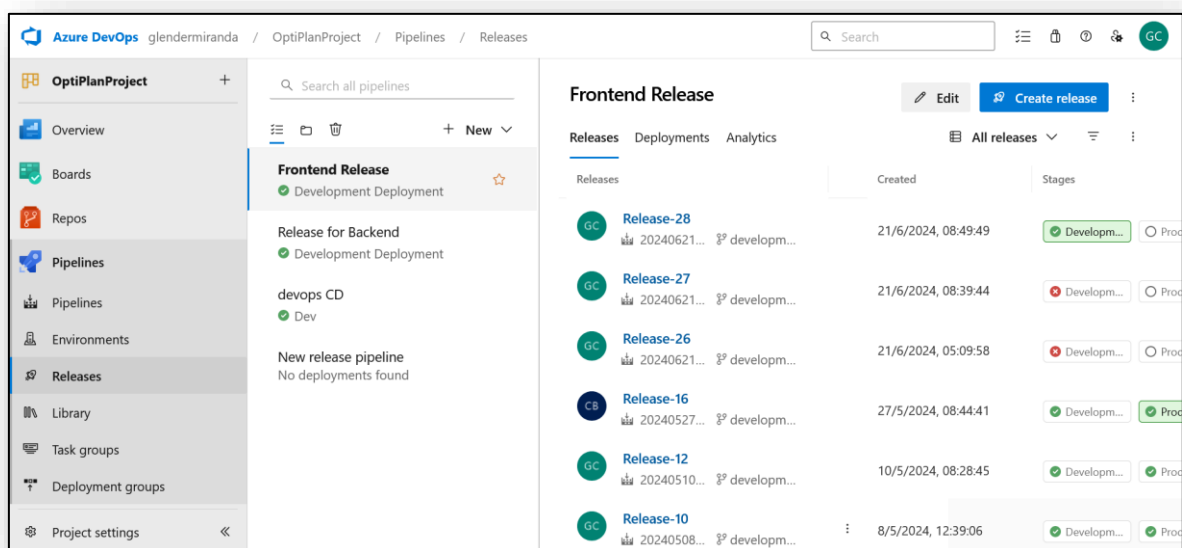


Figura 48. Separación de Pipelines y Releases

Cada uno de los releases se realizaron siguiendo una norma que dice que se deben realizar despliegues cada vez que exista un pull request. De esta manera, cada vez que un miembro del equipo suba un cambio desde su rama a la rama de desarrollo, se realizará la integración y construcción del aplicativo y, una vez finalizado, se realizará un despliegue de manera automática para visualizar el aplicativo desplegado en el ambiente de desarrollo.

La segunda parte, es la entrega continua, esta se realizó una vez cada semana, y antes de la presentación del producto software en el sprint review. De esta manera se despliega el aplicativo en los ambientes de producción listos para ser usado por los usuarios finales.

En el Anexo 1, se encuentran capturas de pantalla del aplicativo desplegado tanto en los entornos de desarrollo como de producción.

Esta actividad aplicó las siguientes prácticas y principios:

Despliegue Continuo: Al separar las integraciones y los despliegues, se implementó una práctica fundamental de DevOps, el despliegue continuo (CD). Esto permite que cada cambio en el código se despliegue automáticamente en el entorno de desarrollo, asegurando que los desarrolladores puedan ver y probar sus cambios en un entorno realista inmediatamente después de realizar un pull request.

Entrega Continua: La entrega continua (CD) se asegura de que el software esté siempre en un estado listo para ser desplegado en producción. Al realizar entregas semanales y antes de cada sprint review, se garantiza que el software esté siempre en su mejor estado posible para la revisión y uso por parte de los usuarios finales.

Configuración estable de pipelines y releases:

A partir del segundo release, se obtuvo una configuración estable y completamente funcional de los pipelines y releases.

En el repositorio para frontend se configuraron los pipelines necesarios para automatizar la construcción, prueba y despliegue del frontend desarrollado en NextJS. Estos pipelines aseguran que cada cambio realizado en el código se someta a pruebas y validaciones automáticas antes de su despliegue.

Se configuraron los pipelines necesarios para automatizar la construcción y despliegue del frontend desarrollado en NextJS. Estos pipelines aseguran que cada cambio realizado en el código se someta a validaciones automáticas antes de su despliegue. La Figura 49 presenta el pipeline de construcción del frontend. Las primeras líneas (líneas 1 a 6) muestran los desencadenantes del pipeline, dicho desencadenante depende de la rama que obtiene el cambio (development o master).

```

1  trigger:
2    branches:
3      include:
4        - master
5      exclude:
6        - development
7
8  variables:
9    # azureSubscription: ')'
10   # webAppName: 'optiplanepn'
11   # environmentName: 'production-front'
12   vmImageName: 'ubuntu-latest'
13
14  stages:
15  - stage: Build
16    displayName: Build stage
17    jobs:
18  - job: Build
19    displayName: Build
20    pool:
21      vmImage: $(vmImageName)
22
23    steps:
24  - task: NodeTool@0
25    inputs:
26      versionSpec: '20.x'
27      displayName: 'Install Node.js'
28
29  - script: |
30      npm install
31      npm run build
32      displayName: 'npm install and build'
33  env:
34      NEXT_PUBLIC_BACKEND_URL: $(PROD_NEXT_PUBLIC_BACKEND_URL)
35      NEXTAUTH_SECRET: $(NEXTAUTH_SECRET)
36
37  - task: ArchiveFiles@2
38    displayName: 'Archive files'
39    inputs:
40      rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
41      includeRootFolder: false
42      archiveType: zip
43      archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
44      replaceExistingArchive: true
45
46  - publish: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
47    artifact: drop

```

Figura 49. Pipeline de integración continua para frontend

En la línea 8 a la 12 se encuentran las variables del pipeline, estas pueden ser utilizadas para almacenar valores comúnmente usados en el código. A partir de la línea 14, se

encuentra la fase de construcción, dentro de esta fase se encuentran los pasos para instalar el ambiente (línea 23 a 27), los comandos necesarios para construir el aplicativo (línea 29 a 35), y la publicación del artefacto (a partir de la línea 37).

Para el backend desarrollado en NestJS, se configuraron pipelines específicos que automatizan el proceso de construcción, prueba y despliegue, tal como se muestra en la Figura 50. Estas configuraciones aseguran que la lógica del backend se implemente de manera eficiente y confiable. Este pipeline define la fase de construcción desde la línea 14, en esta fase se realiza todo el proceso de selección del ambiente en Node, los comandos de instalación de dependencias y construcción (líneas 28 a 31), la definición de las variables de entorno (líneas 32 a 39) y el manejo del artefacto para el despliegue (a partir de la línea 41).

```

6 trigger:| 41
7 - master 42
8 43
9 variables: 44
10 # Agent VM image name 45
11 vmImageName: 'ubuntu-latest' 46
12 47
13 stages: 48
14 - stage: Build 49
15 displayName: Build stage 50
16 jobs: 51
17 - job: Build 52
18 displayName: Build 53
19 pool: 54
20 vmImage: $(vmImageName) 55
21 56
22 steps: 57
23 - task: NodeTool@0 58
24 inputs: 59
25 versionSpec: '20.x' 60
26 displayName: 'Install Node.js' 61
27 62
28 - script: | 63
29 yarn install 64
30 yarn build 65
31 displayName: 'yarn install and build' 66
32 env: 67
33 DB_HOST: $(PROD_HOST) 68
34 DB_NAME: $(PROD_DATABASE) 69
35 DB_PASSWORD: $(PROD_PASSWORD) 70
36 DB_PORT: $(PROD_PORT) 71
37 DB_USERNAME: $(PROD_USERNAME) 72
38 STAGE: $(STAGE) 73
39 JWT_SECRET: $(JWT_SECRET) 74
40 75
41 - task: CopyFiles@2 76
42 displayName: 'Copy package*.json to dist folder' 77
43 inputs: 78
44 Contents: 'package*.json'
45 TargetFolder: 'dist'
46
47 - script: |
48 yarn install
49 displayName: 'yarn install inside dist folder'
50 workingDirectory: 'dist'
51
52 - task: CopyFiles@2
53 displayName: 'Copy dist files to ArtifactStagingDirectory'
54 inputs:
55 Contents: 'dist/**'
56 TargetFolder: '$(Build.ArtifactStagingDirectory)'
57
58 - task: ArchiveFiles@2
59 displayName: 'Build zip file'
60 inputs:
61 rootFolderOrFile: '$(Build.ArtifactStagingDirectory)'
62 includeRootFolder: true
63 archiveType: zip
64 archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
65 replaceExistingArchive: true
66
67 - task: DeleteFiles@1
68 displayName: 'Delete dist folder from ArtifactStagingDirectory'
69 inputs:
70 SourceFolder: '$(Build.ArtifactStagingDirectory)'
71 Contents: '/dist'
72
73 - task: PublishBuildArtifacts@1
74 displayName: 'Publish Artifact: drop'
75 inputs:
76 PathToPublish: '$(Build.ArtifactStagingDirectory)'
77 ArtifactName: 'drop'
78

```

Figura 50. Pipeline de integración continua para backend

La configuración de releases asegura que los despliegues se realicen de manera ordenada y controlada, permitiendo un flujo continuo de entrega y despliegue.

Para el frontend, se crearon dos fases de release:

- Despliegue Continuo: Se realiza en el entorno de desarrollo (backdev-optiplanepn).
- Entrega Continua: Se realiza en el entorno de producción (backprod-optiplanepn).

La Figura 51 muestra la configuración general del release para frontend. En esta sección se configuró el despliegue continuo cada vez que el pipeline de integración continua genere un nuevo artefacto. De manera similar, se tiene un desencadenador manual para desplegar el aplicativo a un entorno de producción cuando se lo requiera.

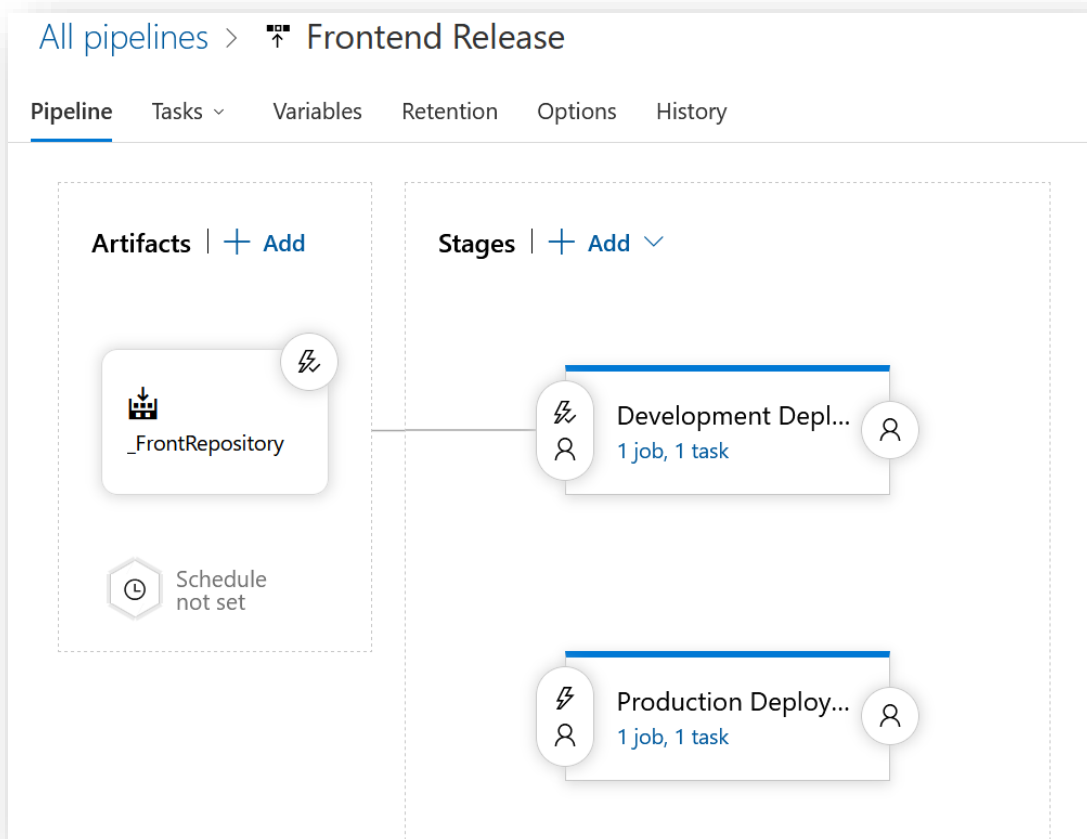


Figura 51. Release de despliegue y entrega continua para frontend

De manera similar, para el backend se configuraron fases de release que permiten el despliegue continuo en el entorno de desarrollo y la entrega continua en el entorno de producción, garantizando que las nuevas funcionalidades y correcciones se desplieguen

de manera controlada y sin interrupciones. La Figura 52 proporciona la vista principal de la configuración del release para backend.

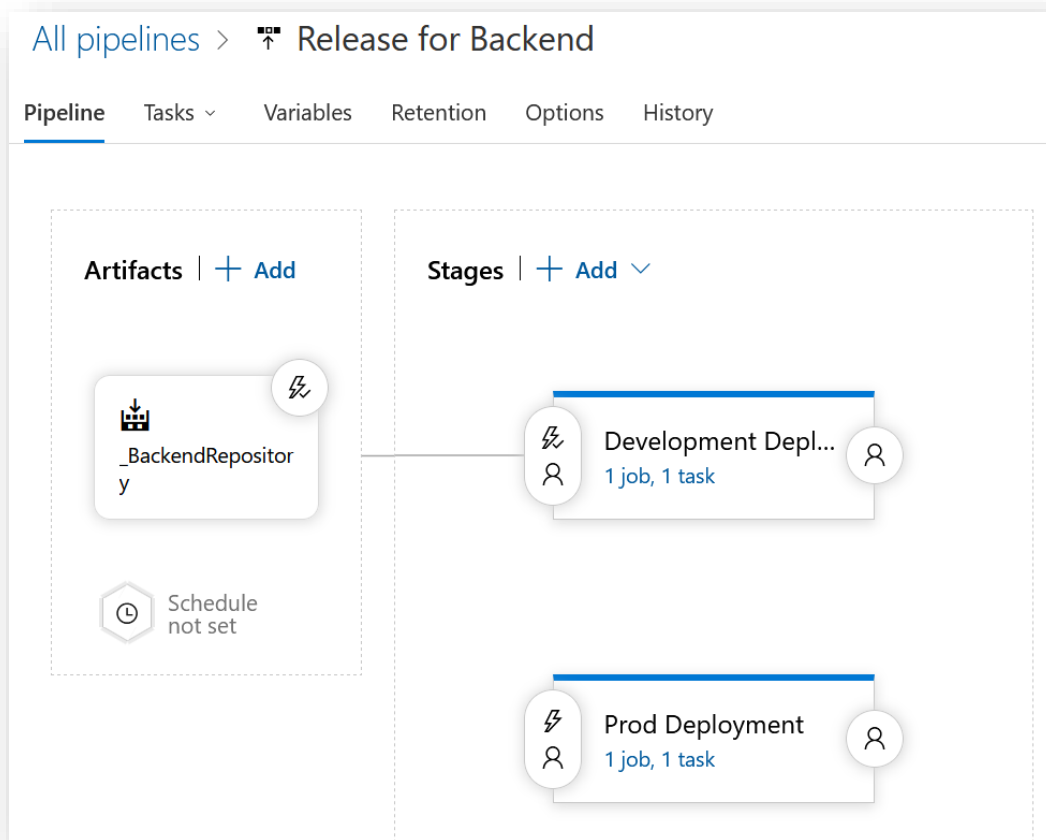


Figura 52. Release de despliegue y entrega continua para backend

El release para backend está configurado de manera similar al de frontend. El despliegue continuo se realiza cada vez que existe un nuevo artefacto y, cada vez que sea necesario, se puede realizar un despliegue manual al ambiente de producción.

Dentro de los releases, es importante usar variables de entorno para realizar de manera correcta los despliegues. Por esta razón, se debe definir estas variables, tal como se muestra en la Figura 53, todos los valores involucrados en el despliegue del aplicativo deben ser ingresados en esta sección. Además, debido a que los releases se realizan tanto para desarrollo como para producción, se pueden dividir las variables en los distintos ambientes.

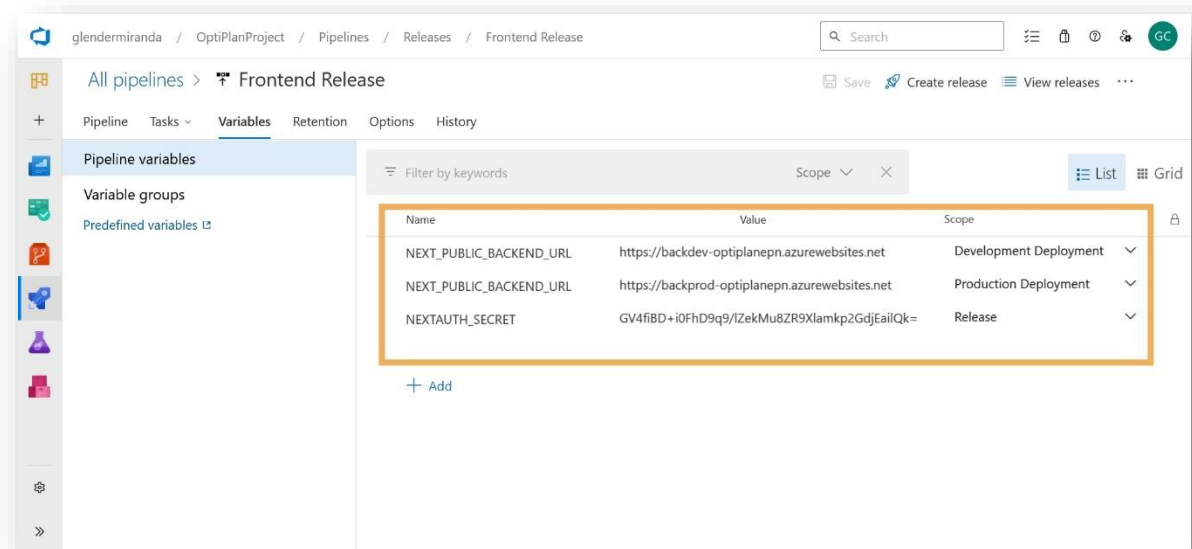


Figura 53. Variables de entorno para releases

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1. Resultados

Verificación del tiempo de despliegue y entrega del aplicativo.

En varios sprints se realizaron comprobaciones de los tiempos en los que el aplicativo desplegaba un recurso. Si bien la velocidad de despliegue está apegada a la disponibilidad del recurso y el tipo de plan utilizado, estos recursos demoran su despliegue por la configuración de su pipeline.

Se realizó una mejora en el pipeline de integración que permitía un mejor desempeño al momento de la implementación. En la se muestra un ejemplo de uno de los despliegues que se realizaron antes de la mejora del pipeline de despliegue, se puede observar como el tiempo de despliegue es de aproximadamente 22 minutos. En la se muestra el despliegue con el pipeline de construcción actualizado, la duración del despliegue es de apenas dos minutos. Es importante tomar en cuenta que, aunque suele variar el tiempo de despliegue, en promedio se disminuyó el tiempo en el que los pipelines construyen y despliegan el aplicativo.

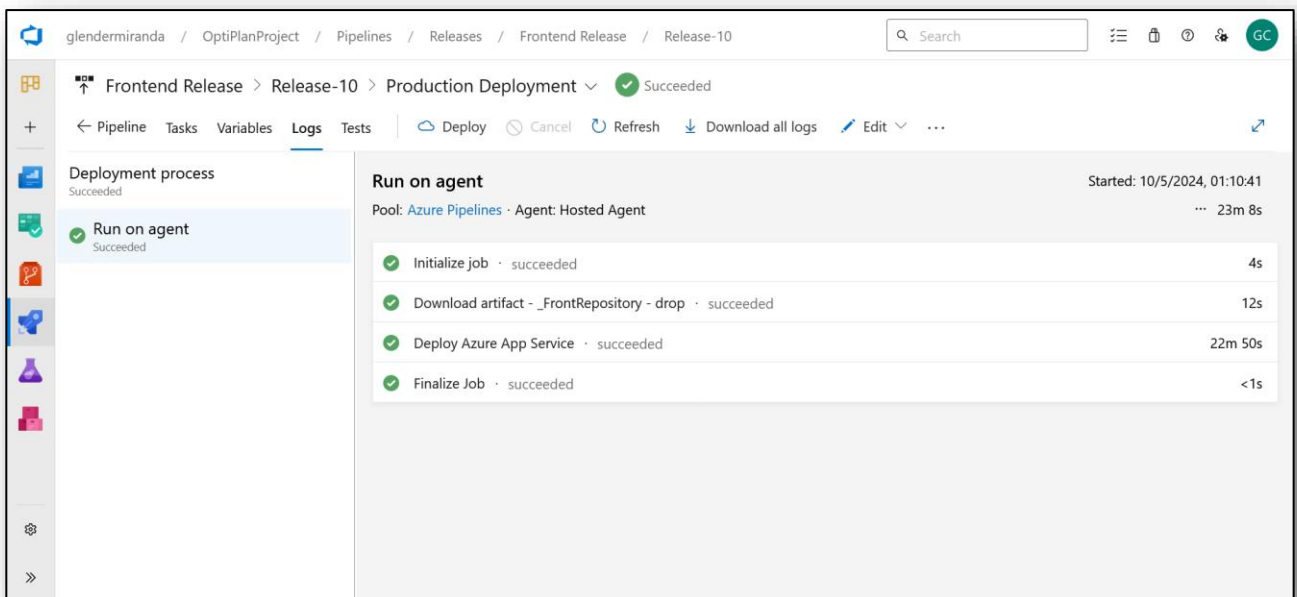


Figura 54. Despliegue del aplicativo con el pipeline de construcción antiguo

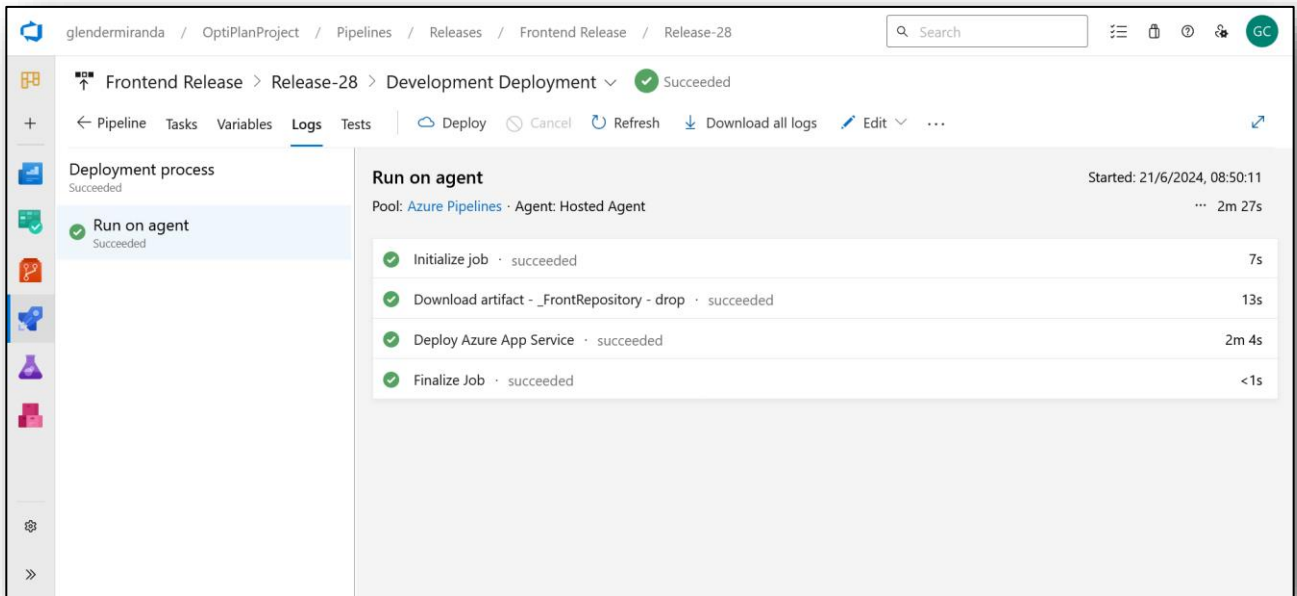


Figura 55. Despliegue del aplicativo con el pipeline de despliegue actual

Colaboración del equipo de desarrollo en las integraciones y pull requests

En los diferentes sprints se realizaron actividades que mejoraron la comunicación y colaboración de equipo de trabajo en el proyecto. Una de ellas fue la comunicación a través de los comentarios de los Pull Request en Azure DevOps.

Cada vez que se realiza un Pull Request, se tiene a una persona encargada de dicha solicitud de cambios, por lo general se trata del creador del Pull Request. Una vez que se han realizado los procesos de validación de las políticas, como la Política de construcción de la aplicación, se puede realizar comentarios como se muestra en la . Este es un espacio donde los usuarios pueden hablar de aspectos importantes del código, el despliegue o la integración de la aplicación con el cambio realizado. Una vez que todos los miembros estén de acuerdo, se puede configurar el Pull Request para que el cambio se suba a la rama de destino de manera automática.

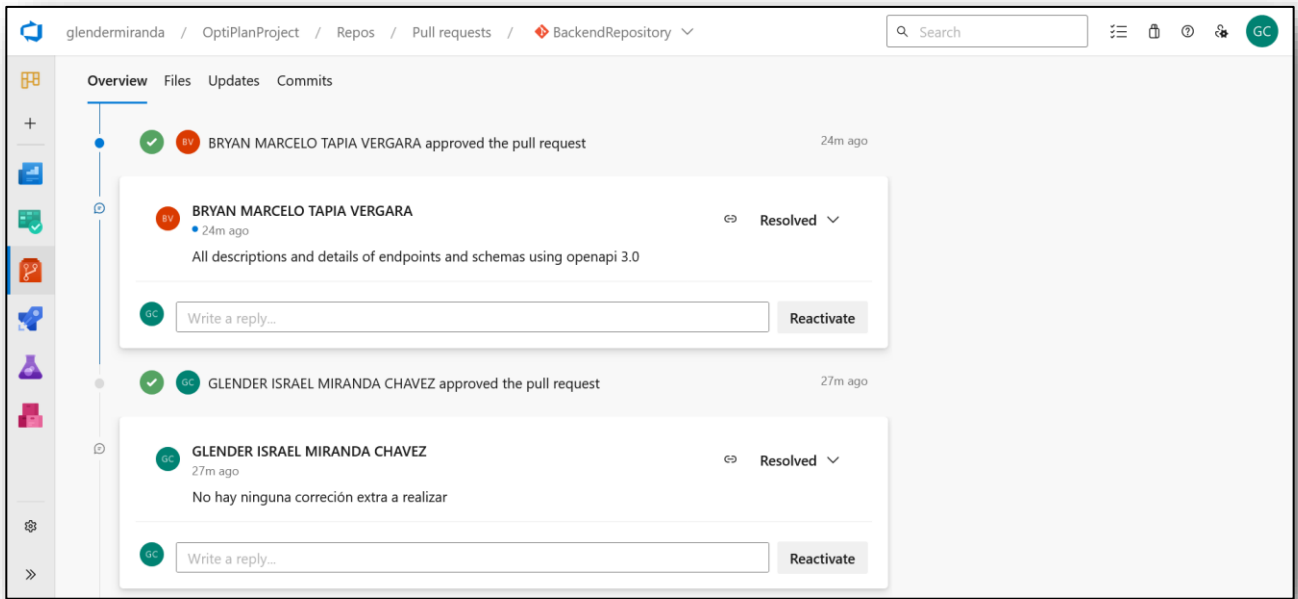


Figura 56. Colaboración del equipo de trabajo en los comentarios de un pull request

Despliegues y entregas continuas

El proceso de despliegue abarcó tanto el frontend como el backend en los ambientes de desarrollo y producción. Primero, se implementaron y verificaron los despliegues en el ambiente de desarrollo, asegurando que todos los componentes funcionaran correctamente. Luego, se replicaron estos pasos en el ambiente de producción, donde se realizaron pruebas adicionales para asegurar que el sistema estuviera completamente operativo para los usuarios finales. Esta fase incluyó la confirmación de que todos los recursos estaban activos y optimizados para un rendimiento óptimo. En la Figura 57 se muestra el aplicativo de frontend desplegado en el entorno de desarrollo. En la se muestra el aplicativo de backend desplegado en el entorno de desarrollo, el servicio está probado en el aplicativo de Postman para visualizar las respuestas del API.

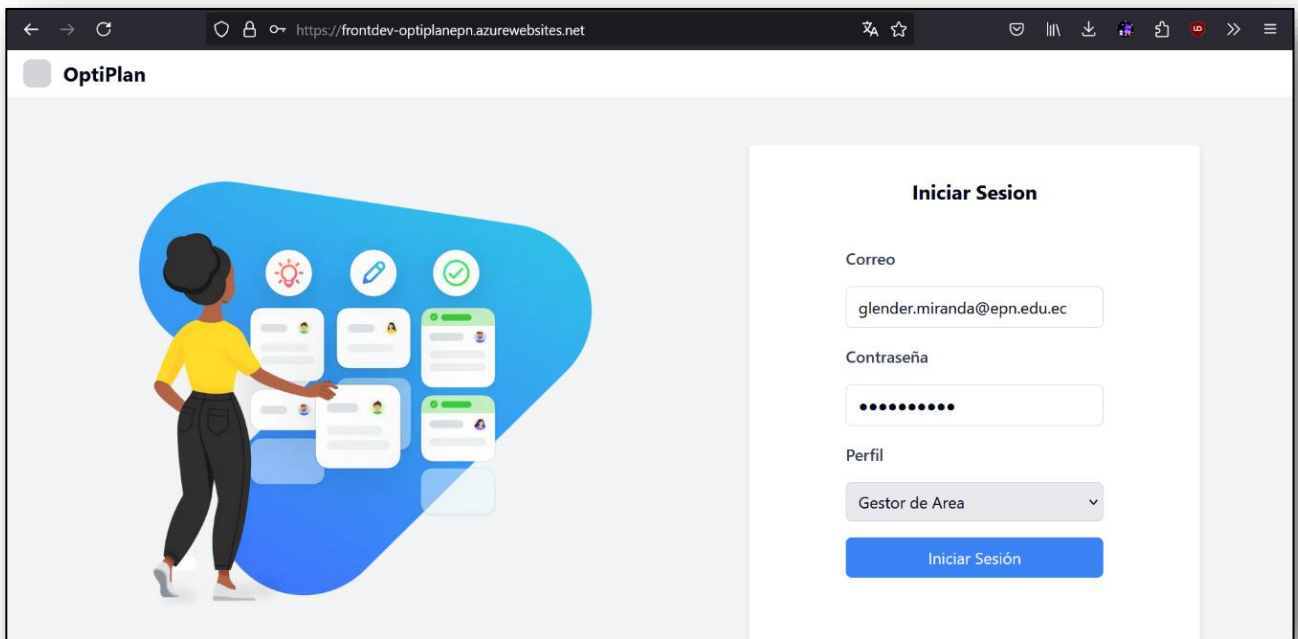


Figura 57. Aplicativo web desplegado en el entorno de desarrollo

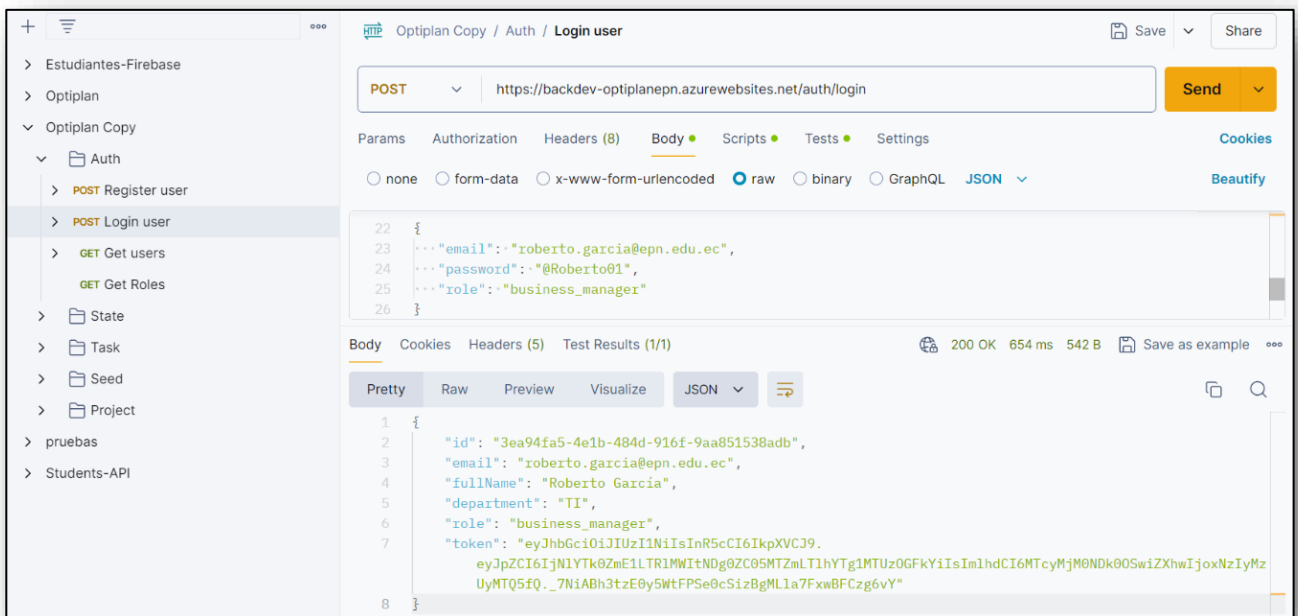


Figura 58. Servicio de backend desplegado en el entorno de desarrollo

En el ambiente de producción, el despliegue del frontend y backend se realizó con especial atención para garantizar que el sistema estuviera listo para su uso final por los usuarios.

Se llevaron a cabo pruebas adicionales para validar la funcionalidad y el rendimiento en condiciones reales de uso. La implementación exitosa en producción permitió que las nuevas características y mejoras fueran accesibles de inmediato a los usuarios finales, optimizando la experiencia del cliente y asegurando que el sistema operara sin problemas. Esta etapa finalizó con la verificación de que todos los recursos estaban correctamente configurados y funcionando como se esperaba. En la Figura 59 se muestra el aplicativo frontend desplegado en el entorno de producción y en la Figura 60 se muestra el servicio backend desplegado en producción.

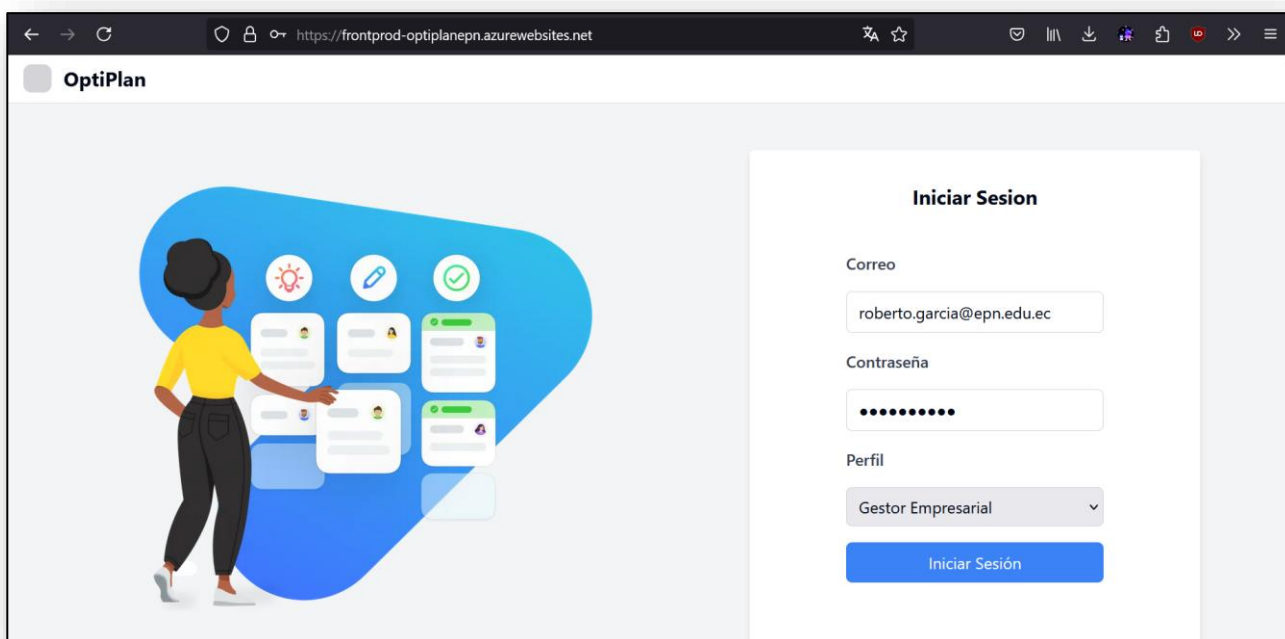


Figura 59. Aplicativo web desplegado en el entorno de producción

3.2. Conclusiones

El componente del proyecto ha logrado cumplir dos objetivos específicos de manera efectiva.

Primero, se implementó con éxito una cadena de herramientas para la integración, despliegue y entrega continua del software. La configuración de los pipelines de Azure DevOps permitió la integración y construcción fluida de las aplicaciones, tanto el backend como el frontend. Se estableció un proceso de despliegue continuo que facilitó la actualización automática de nuevas funcionalidades en el entorno de desarrollo. Adicionalmente, los despliegues en el entorno de producción se realizaron semanalmente, típicamente los domingos por la madrugada, para evitar interferencias con los usuarios. La correcta configuración de los entornos de desarrollo y producción, junto con la implementación de políticas y ajustes adecuados, garantizó que los despliegues se realizaran sin problemas, asegurando una integración limpia y eficiente.

Segundo, Se logró verificar y aplicar las prácticas y principios DevOps durante el desarrollo del software. La participación en el equipo de desarrollo permitió identificar y resolver errores y conflictos durante el proceso de entrega. Un conocimiento detallado del código y los artefactos en desarrollo facilitó la corrección de problemas en las herramientas de integración. La colaboración continua con el equipo de trabajo fue crucial para definir y mejorar los pipelines de integración continua, ya que el feedback recibido ayudó a ajustar y optimizar los procesos. Esto permitió resolver los problemas identificados, garantizando una integración continua efectiva y alineada con las mejores prácticas de DevOps.

La correcta configuración de los pipelines y el trabajo en equipo han sido clave para superar los desafíos y asegurar una integración sin errores. La experiencia adquirida y las lecciones aprendidas durante el proyecto ofrecen una base sólida para futuras implementaciones y mejoras en el proceso de desarrollo y entrega continua. Estos resultados destacan la importancia de una colaboración efectiva y una planificación cuidadosa para lograr un desarrollo de software ágil y eficiente.

3.3. Recomendaciones

Durante el desarrollo del componente, se notó que utilizar recursos en la nube para el desarrollo local generaba costos elevados debido a su modelo de pago por uso. En lugar de utilizar estos recursos para el trabajo diario de desarrollo, se sugiere utilizar entornos locales o herramientas que permitan el desarrollo sin depender constantemente de los recursos en la nube. En nuestro caso, el equipo usó el API del backend en la nube para la codificación en cada máquina local. Esto resultó en un alto consumo de créditos, ya que cada cambio en el código activaba el recurso. Por lo tanto, es más eficiente limitar el uso de estos recursos a momentos específicos, como las pruebas finales o la integración, en lugar de usarlos continuamente durante el desarrollo.

Se identificó que se crearon más recursos de los necesarios, como dos servidores de bases de datos para desarrollo y producción. Consolidar estos recursos en un solo servidor, siempre que sea posible, puede mejorar la eficiencia y reducir los costos operativos. Utilizar un único servidor para ambas bases de datos no solo simplifica la administración, sino que también puede llevar a una reducción en el costo de almacenamiento y en el mantenimiento del sistema. Evaluar la viabilidad de esta consolidación y ajustar la infraestructura en función de las necesidades reales puede optimizar los recursos y reducir costos innecesarios.

Es fundamental gestionar adecuadamente los recursos en Azure Portal, ya que estos generan costos incluso cuando están apagados. Por ejemplo, el almacenamiento en una base de datos sigue generando costos, aunque el servidor esté inactivo. Para manejar estos costos de manera más eficiente, se recomienda utilizar herramientas de aprovisionamiento de infraestructura como Terraform. Esta herramienta permite crear y eliminar recursos de forma automatizada y bajo demanda, asegurando que solo se mantengan activos los recursos necesarios en el momento preciso. Terraform facilita la gestión de configuraciones y ayuda a evitar gastos innecesarios al eliminar recursos no utilizados.

Existen diversas opciones para el despliegue de aplicaciones web y servidores en Azure Portal que pueden ser más económicas que los recursos tradicionales utilizados. Opciones como máquinas virtuales, contenedores Docker o funciones de Azure pueden ofrecer costos más bajos y una mayor flexibilidad. Las máquinas virtuales permiten una infraestructura más ajustable, mientras que los contenedores Docker facilitan la implementación y escalabilidad de aplicaciones. Las funciones de Azure, por otro lado,

ofrecen un modelo de pago por uso basado en eventos, que puede resultar más económico para ciertas aplicaciones. Evaluar y comparar estas alternativas puede llevar a una optimización significativa en los costos de infraestructura.

En la gestión de pipelines y releases, es beneficioso contar con el apoyo del equipo de desarrollo. Ellos pueden ofrecer perspectivas adicionales y soluciones basadas en su experiencia, lo que puede ser crucial para resolver problemas de manera más eficiente. La colaboración puede reducir el tiempo invertido en la documentación y en la identificación de errores, ya que los miembros del equipo aportan conocimientos especializados y experiencia práctica. Esta cooperación no solo ayuda a resolver problemas más rápidamente, sino que también mejora el proceso de desarrollo al integrar diferentes puntos de vista y enfoques.

4. REFERENCIAS BIBLIOGRÁFICAS

- [1] Microsoft Learn, “¿Qué es Azure DevOps? - Azure DevOps | Microsoft Learn.” Accessed: May 19, 2024. [Online]. Available: <https://learn.microsoft.com/es-es/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>
- [2] RAE, “ágil | Definición | Diccionario de la lengua española | RAE - ASALE.” Accessed: Jul. 20, 2024. [Online]. Available: <https://dle.rae.es/%C3%A1gil>
- [3] M. Fowler *et al.*, “Manifiesto for Agile Software Development.” Accessed: Jul. 19, 2024. [Online]. Available: <https://agilemanifesto.org/>
- [4] K. Rubin, “Praise for Essential Scrum,” Jul. 2012.
- [5] K. Schwaber and J. Sutherland, “Scrum Guide Spanish Latin South American.”
- [6] J. César and S. Guamán, “MoCIP: Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube TESIS DOCTORAL,” 2020.
- [7] S. Zaal, S. Demiliani, A. Malik, O’Reilly for Higher Education (Firm), and an O. M. Company. Safari, *Azure DevOps Explained*. 2020.
- [8] R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer, “What is DevOps? A systematic mapping study on definitions and practices,” *ACM International Conference Proceeding Series*, vol. 24-May-2016, May 2016, doi: 10.1145/2962695.2962707.
- [9] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, “Towards a benefits dependency network for DevOps based on a systematic literature review,” *Journal of Software: Evolution and Process*, vol. 30, no. 11, p. 07, Nov. 2018, doi: 10.1002/SMR.1957.
- [10] A. R. F. J. A. A. Morales, H. Yasar, and A. Volkman, “Implementing DevOps Practices in Highly Regulated Environments,” ACM, 2018. [Online]. Available: https://doi.org/10.475/123_4
- [11] M. Hornbeek, *Engineering DevOps*. 2019.
- [12] O. Krancher, P. Luther, and M. Jost, “Key Affordances of Platform-as-a-Service: Self-Organization and Continuous Feedback,” *Journal of Management Information Systems*, vol. 35, no. 3, pp. 776–812, Jul. 2018, doi: 10.1080/07421222.2018.1481636.
- [13] S. Sharma, “The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise,” 2017.
- [14] M. Fowler, “Continuous Integration.” Accessed: Jun. 28, 2024. [Online]. Available: <https://www.martinfowler.com/articles/continuousIntegration.html>
- [15] J. Davis and R. Daniels, “Effective DevOps,” 2016.
- [16] K. Morris, *Infrastructure as Code*. 2021. Accessed: Jul. 19, 2024. [Online]. Available: https://books.google.es/books?hl=en&lr=&id=UW4NEAAQBAJ&oi=fnd&pg=PP1&dq=infrastructure+as+code&ots=cdkMUO1FQn&sig=J5r26_tXbgMCJiDLaXLHvXoaB2w#v=onepage&q=infrastructure%20as%20code&f=false
- [17] E. Corona, F. Eros, and P. Diee, “A Review of Lean-Kanban Approaches in the Software Development,” 2013.

- [18] Microsoft Learn, "Información general - Azure App Service | Microsoft Learn." Accessed: Jul. 26, 2024. [Online]. Available: <https://learn.microsoft.com/es-mx/azure/app-service/overview>
- [19] MDN web docs, "Intercambio de recursos de origen cruzado (CORS) - HTTP | MDN." Accessed: Jul. 26, 2024. [Online]. Available: <https://developer.mozilla.org/es/docs/Web/HTTP/CORS>
- [20] Microsoft Learn, "About access levels - Azure DevOps | Microsoft Learn." Accessed: Jul. 27, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/azure/devops/organizations/security/access-levels?view=azure-devops>
- [21] Microsoft Learn, "Default processes and process templates - Azure Boards | Microsoft Learn." Accessed: Jul. 27, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops&tabs=agile-process>