UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DESARROLLO WEB Y MÓVIL DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

Opción de Titulación:

Proyecto de titulación con componentes de investigación aplicada y/o de desarrollo

Tema:

Migración de los módulos de Contabilidad y Documentos electrónicos a microservicios con flujos GitSecOps

Autor(es)

Cristian Raúl Lara Balarezo

Gabriela Jacquelyne Farinango Tibanta

Director:

Rodrigo Efrain Tufiño Cardenas

Quito – Ecuador

2024

Autor(es):

Cristian Raúl Lara Balarezo
Ingeniero en Sistemas Informáticos y de Computación
Candidato a Magíster en Software por la Universidad Salesiana – Sede Quito.
clarab@est.ups.edu.ec

Gabriela Jacquelyne Farinango Tibanta
Ingeniera en Sistemas Informáticos y de Computación
Candidata a Magíster en Software por la Universidad Politécnica Salesiana – Sede Quito.
gfarinangot@est.ups.edu.ec

Dirigido por:

Rodrigo Efrain Tufiño Cardenas
Ingeniero de Sistemas
Master Universitario en Ciencias y Tecnologías de la Computación.
Master Universitario en Software Libre
rtufino@ups.edu.ec

Cristian Raúl Lara Balarezo - Gabriela Jacquelyne Farinango Tibanta
MIGRACIÓN DE LOS MÓDULOS DE CONTABILIDAD Y DOCUMENTOS ELECTRÓNICOS A MICROSERVICIOS CON FLUJOS GITSECOPS

DEDICATORIA

Dedicamos este proyecto a nuestras familias, cuyo apoyo incondicional ha sido nuestro refugio y fortaleza a lo largo de este desafío. Sus sacrificios no pasan desapercibidos y son la base de nuestro compromiso y perseverancia.

También, queremos dedicar este trabajo a nuestros colegas de Manticore Labs, quienes nos han inspirado con su pasión por la excelencia y la innovación. Su guía y ánimo han sido esenciales para superar los obstáculos y lograr nuestras metas.

Finalmente, a todos aquellos que creen en el poder de la tecnología para transformar y mejorar el mundo. Esperamos que este proyecto de titulación sirva de inspiración para futuras generaciones de innovadores que continúen explorando, aprendiendo y superando los límites de lo posible.

AGRADECIMIENTO

Queremos expresar nuestro sincero agradecimiento a todas las personas que han contribuido al éxito de este proyecto de titulación.

Agradecemos profundamente a los líderes de Manticore Labs por su apoyo continuo y su confianza en la visión del proyecto. Su liderazgo inspirador y su compromiso con la innovación han sido esenciales para impulsar los avances tecnológicos que se ha planteado.

De la misma forma, agradecemos a nuestro tutor, quien ha sido una fuente orientación y apoyo a lo largo de este proyecto. Su meticulosa revisión y valiosos comentarios han sido esenciales para la culminación exitosa de nuestro trabajo final.

Nuestra gratitud se extiende también a los docentes cuya orientación en cada asignatura ha sido invaluable. Su experiencia y conocimientos han sido transcendentales para navegar por las complejidades de la arquitectura de microservicios y las tecnologías de CI/CD.

Finalmente, pero no menos importante, agradezco a todos los miembros de la comunidad que han compartido sus conocimientos y recursos, que han enriquecido nuestro enfoque y ampliado nuestro entendimiento.

Este proyecto no solo ha sido un viaje tecnológico, sino también una oportunidad para fortalecer la colaboración y el espíritu de equipo.

Tabla de Contenido

Migración de los módulos de Contabilidad y Documentos electrónicos a microservicios con flujos GitSecOps

Autor(es):

Cristian Raúl Lara Balarezo
Gabriela Jacquelyne Farinango Tibanta

Resumen

Este proyecto se enfoca en la migración de los módulos de Contabilidad y Documentos Electrónicos de la empresa Manticore-Labs, de una arquitectura monolítica a una basada en microservicios, integrando flujos GitSecOps para optimizar la gestión y el monitoreo. La necesidad de esta migración surge de las limitaciones de la arquitectura monolítica actual, que restringe la escalabilidad y el mantenimiento eficiente. Este trabajo propone un diseño de arquitectura de microservicios, su validación a través de una prueba de concepto, y la implementación de patrones de diseño para la gestión de transacciones distribuidas. Además, se establecen y validan flujos GitSecOps, mejorando la seguridad y eficiencia en la integración y despliegue continuo. La metodología empleada incluye la planificación detallada, análisis, diseño, pruebas y evaluación, asegurando un enfoque sistemático y estructurado. Los resultados anticipan mejoras significativas en rendimiento, escalabilidad, seguridad y gestión, demostrando la viabilidad y los beneficios de la migración a microservicios en el contexto empresarial.

Palabras clave:

Migración a Microservicios, GitSecOps, Contabilidad Digital, Documentos Electrónicos, Transformación Digital, Automatización en Desarrollo de Software, Seguridad en Software, Arquitectura de Software Empresarial, Despliegue Continuo, Escalabilidad de Sistemas, Innovación Tecnológica, Kubernetes, Gestión de Transacciones Distribuidas.

Abstract

This study focuses on migrating the Accounting and Electronic Documents modules of Manticore-Labs from a monolithic architecture to a microservices-based one, incorporating GitSecOps flows for enhanced management and monitoring. The migration is necessitated by the limitations of the current monolithic architecture, which hampers scalability and efficient maintenance. This thesis proposes a microservices architecture design, its validation through a proof of concept, and the implementation of design patterns for managing distributed transactions. Furthermore, GitSecOps flows are established and validated, improving security and efficiency in continuous integration and deployment. The methodology includes detailed planning, analysis, design, testing, and evaluation, ensuring a systematic and structured approach. The anticipated results suggest significant improvements in performance, scalability, security, and management, demonstrating the feasibility and benefits of migrating to microservices in a business context.

Keywords:

Microservices Migration, GitSecOps, Digital Accounting, Electronic Documents, Digital Transformation, Software Development Automation, Software Security, Enterprise Software Architecture, Continuous Deployment, System Scalability, Technological Innovation, Kubernetes, Distributed Transaction Management.

1 Introducción

1.1 Antecedentes

En el ámbito tecnológico, la migración de una arquitectura monolítica a microservicios ha emergido como una solución prominente para superar las limitaciones inherentes de las arquitecturas tradicionales. Se consideran el concepto de migración de servicios de arquitectura monolítica a microservicios, destacando la importancia del despliegue automatizado de microservicios para asegurar una migración exitosa (Kyryk et al., 2022)

La Secretaria de Seguridad Publica de Mato Grosso, como se documenta en el trabajo de (Delgado Preti et al., 2021), ha empleado una estrategia de migración para convertir sus sistemas legados monolíticos en una arquitectura orientada a microservicios, obteniendo resultados exitosos en la coexistencia de monolitos y microservicios. (Delgado Preti et al., 2021)

Radostev y Nikitina (2021) proponen una estrategia para ayudar a las empresas a transicionar de una arquitectura de aplicación monolítica a una arquitectura de microservicios, destacando los beneficios de escalabilidad y reparabilidad que ofrece esta última. (Perm State University et al., 2021)

En el contexto de la descomposición de sistemas monolíticos, Rochimah y Nuralamsyah (2023) proponen un método basado en el análisis del código fuente de la aplicación para asistir a los desarrolladores en la descomposición de sistemas

monolíticos en microservicios, logrando una precisión del 0.81 en la agrupación de componentes monolíticos con dominio empresarial similar (Siti Rochimah & Bintang Nuralamsyah, 2023)

Guamán et al. (2018) evalúan el rendimiento en el proceso de migración de una aplicación monolítica a microservicios, implementando y evaluando actividades propuestas en modelos de migración existentes, y utilizando herramientas

como Apache JMeter para obtener métricas relacionadas con el uso de recursos como CPU, memoria, red y acceso a base de datos. (Guaman et al., 2018)

Estos trabajos previos demuestran el éxito y los beneficios de la migración de sistemas monolíticos a microservicios en diversos contextos, proporcionando valiosas lecciones y estrategias que pueden informar y guiar el proyecto propuesto para la migración de los módulos de Contabilidad y Documentos Electrónicos de Manticore-Labs. (Cataldi, 2000)

1.2 Justificación

La arquitectura monolítica actual en la que se basan los módulos de Contabilidad y Documentos Electrónicos en la empresa Manticore-Labs ha demostrado ser un desafío significativo para la escalabilidad y el mantenimiento eficiente.

Esta estructura, aunque funcional, ha presentado limitaciones notables en la capacidad de adaptación y evolución, necesarias para enfrentar las demandas cambiantes del entorno empresarial moderno.

Se identifica una necesidad critica de migrar a una arquitectura de microservicios, que promete no solo mejorar la escalabilidad y la eficiencia operativa, sino también facilitar la implementación de nuevas funcionalidades y cambios

necesarios. Esta migración permitirá a la empresa Manticore-Labs gestionar de manera más efectiva cada aspecto de los módulos de Contabilidad y Documentos Electrónicos, permitiendo una distribución más eficiente de los recursos y

asegurando una mayor resiliencia y flexibilidad del sistema.

Además, la adopción de flujos GitSecOps en el proyecto propuesto asegurará una integración y despliegue continuo más seguros y eficientes. Esto no solo mejorará la colaboración entre los equipos de desarrollo y operaciones, sino que también garantizará que las mejores prácticas de seguridad se implementen desde las etapas iniciales del desarrollo hasta el despliegue en producción.

1.3 Objetivos

A continuación, se presenta el objetivo general del proyecto y los objetivos específicos que se alcanzarán con el desarrollo del mismo.

Objetivo General:

Diseñar y validar una arquitectura basada en microservicios de los módulos de Contabilidad y Documentos Electrónicos, y establecer flujos GitSecOps para la gestión y monitoreo.

Objetivos Específicos:

- Diseñar la arquitectura de microservicios para los módulos de Contabilidad y Documentos Electrónicos.
- Validar la arquitectura propuesta mediante una prueba de concepto a nivel de prototipo.
- Establecer patrones de diseño adecuados para la gestión de transacciones distribuidas en los módulos.
- Definir y validar flujos GitSecOps para la gestión y monitoreo de la arquitectura de microservicios.
- Documentar los diagramas de arquitectura utilizando el estándar C4.

2 Determinación del Problema

El problema central de esta investigación se origina en las deficiencias de la arquitectura monolítica actual utilizada en Manticore-Labs para los módulos de Contabilidad y Documentos Electrónicos. Esta arquitectura, aunque operativamente funcional, enfrenta serias limitaciones en escalabilidad, lo que impide el crecimiento y la adaptación eficiente a las demandas fluctuantes del mercado. La escalabilidad limitada se manifiesta en la incapacidad de manejar aumentos en la carga de trabajo sin comprometer el rendimiento.

Además, el mantenimiento y las actualizaciones del sistema son particularmente problemáticos; los cambios en una parte del sistema pueden tener implicaciones imprevistas en otras áreas, aumentando el riesgo de errores y tiempo de inactividad.

La integración de nuevas funcionalidades en el sistema monolítico es otro desafío, ya que requiere un esfuerzo considerable y a menudo conduce a una mayor complejidad y dependencias. Este enfoque rígido limita la agilidad de la empresa para responder a las necesidades emergentes del mercado y adoptar nuevas tecnologías.

En el contexto de GitSecOps, la implementación de prácticas de seguridad y automatización en un sistema monolítico presenta obstáculos significativos, limitando la eficacia de los flujos de trabajo de integración y despliegue continuos.

La arquitectura monolítica también resulta en un uso ineficiente de los recursos, con una asignación de hardware y software que no siempre se alinea con las necesidades reales del sistema, llevando a un desperdicio de recursos y a una gestión ineficiente. Además, la arquitectura actual carece de la resiliencia y flexibilidad necesarias para adaptarse rápidamente a los cambios o recuperarse de fallos, lo que aumenta el riesgo de interrupciones del servicio y afecta negativamente la continuidad del negocio.

Estos problemas colectivos demuestran la necesidad urgente de migrar a una arquitectura de microservicios, que promete no solo mejorar la escalabilidad y la eficiencia operativa, sino también facilitar la implementación de nuevas funcionalidades y cambios necesarios, al tiempo que se integran las prácticas de GitSecOps para mejorar la seguridad, la colaboración y la eficiencia en el desarrollo y operaciones.

3 Marco teórico referencial

Este proyecto de titulación se fundamenta en un marco teórico referencial que abarca una revisión exhaustiva de las arquitecturas monolíticas y de microservicios, así como la implementación de flujos GitSecOps. Este marco teórico proporciona una comprensión profunda de los principios, desafíos y estrategias asociadas con la migración de sistemas monolíticos a arquitecturas basadas en microservicios, enfatizando la importancia de la integración y despliegue continuos en el desarrollo de software moderno.

3.1 Arquitecturas Monolíticas vs Microservicios

3.1.1 Características de la arquitectura monolítica

Una arquitectura monolítica en el desarrollo de software se refiere a un enfoque de diseño donde todas las funciones y componentes de una aplicación se integran en una única y gran base de código. En este modelo, las diferentes partes del sistema, como la interfaz de usuario, la lógica de negocio, y la capa de acceso a datos, están entrelazadas y desplegadas como una unidad indivisible.

Ventajas

Simplicidad en el Desarrollo y Despliegue: Los sistemas monolíticos son generalmente más sencillos de desarrollar y desplegar, especialmente en las etapas iniciales, debido a la ausencia de complejidades asociadas con la distribución y la comunicación entre servicios separados.

Facilidad de Gestión y Pruebas: La gestión y las pruebas de un sistema monolítico pueden ser más directas, ya que todo el sistema se encuentra en un solo lugar. Esto permite una visión holística del sistema durante el desarrollo y la depuración.

Consistencia en la Ejecución: Dado que todos los componentes del sistema operan en un entorno unificado, se garantiza una mayor coherencia en la ejecución de las

operaciones del sistema, lo que puede ser beneficioso para ciertos tipos de aplicaciones donde la integridad y la uniformidad son críticas

Desafíos

Dificultades en la Escalabilidad: Escalar un sistema monolítico puede ser problemático, ya que cualquier cambio o mejora requiere la actualización de toda la aplicación. Esto puede llevar a un aumento en el tiempo y los recursos necesarios para implementar nuevas funcionalidades o manejar un mayor volumen de usuarios o transacciones.

Limitaciones en la Flexibilidad y la Innovación: La naturaleza interconectada de los componentes en un sistema monolítico dificulta la implementación de cambios o la adición de nuevas funcionalidades sin afectar el sistema en su conjunto. Esto puede limitar la capacidad de innovación y adaptación rápida a nuevas necesidades o tecnologías.

Desafíos en la Mantenibilidad y el Riesgo de Rigidez del Código: A medida que el sistema crece, su mantenimiento se vuelve más complejo. Los sistemas monolíticos grandes pueden volverse rígidos y difíciles de modificar, lo que a menudo se conoce como "rigidez del código". Esto puede llevar a un aumento en los costos de mantenimiento y a una disminución en la eficiencia del desarrollo.

Impacto en el Desarrollo Empresarial

En el contexto empresarial, donde la agilidad y la capacidad de respuesta rápida a las demandas del mercado son cruciales, los sistemas monolíticos pueden convertirse en un obstáculo para el crecimiento y la innovación. La necesidad de sistemas más flexibles y escalables ha llevado a muchas organizaciones a explorar alternativas, como la arquitectura de microservicios, para superar las limitaciones de los sistemas monolíticos. (Raharjo et al., 2022)

3.1.2 Características de la arquitectura de microservicios

La arquitectura de microservicios surge como una respuesta a las limitaciones de las arquitecturas monolíticas. En este enfoque, el software se divide en servicios más pequeños, cada uno de los cuales funciona de manera independiente y se comunica con los demás a través de interfaces bien definidas.

Ventajas

Escalabilidad Mejorada: Una de las ventajas más significativas de los microservicios es su capacidad para escalar eficientemente. A diferencia de las arquitecturas monolíticas, donde escalar a menudo significa replicar toda la aplicación, los microservicios permiten escalar solo los componentes necesarios. Esto es particularmente útil en aplicaciones con variabilidad en la carga de trabajo o con requisitos de escalabilidad específicos para ciertas funciones. La escalabilidad granular de los microservicios facilita un uso más eficiente de los recursos y una mejor adaptación a las demandas cambiantes (De Almeida & Canedo, 2022)

Resiliencia y Aislamiento de Fallos: En un sistema basado en microservicios, si un servicio falla, no necesariamente conduce al fallo de toda la aplicación. Esta separación de servicios mejora la resiliencia del sistema, ya que los problemas pueden aislarse y manejarse de manera más efectiva. Además, los sistemas de microservicios suelen incorporar patrones de diseño que permiten a los servicios manejar fallos de manera elegante, como circuit breakers o fallbacks, lo que mejora la experiencia general del usuario final. (Wang et al., 2023)

Desarrollo y Despliegue Independientes: Los microservicios ofrecen la ventaja de permitir que equipos de desarrollo pequeños y ágiles trabajen en diferentes

servicios de manera independiente. Esto reduce significativamente el tiempo de desarrollo, ya que los equipos pueden trabajar en paralelo sin interferir unos con otros. Además, los ciclos de despliegue se acortan, ya que cada servicio se puede actualizar sin necesidad de desplegar toda la aplicación. Esta independencia también facilita la experimentación y la rápida iteración, permitiendo a las

empresas innovar y responder más rápidamente a las necesidades del mercado. (Wang et al., 2023)

Diversidad Tecnológica y Mejor Uso de Recursos: Los microservicios permiten a los equipos elegir la tecnología más adecuada para cada servicio específico, lo que puede resultar en un mejor rendimiento y una mayor eficiencia. Esta diversidad tecnológica también facilita la adopción de nuevas tecnologías y enfoques, ya que se pueden introducir en un servicio específico sin afectar al resto del sistema. Además, los microservicios pueden optimizar el uso de recursos al permitir que cada servicio escale de acuerdo con sus necesidades específicas, evitando así el sobre provisionamiento de recursos que es común en las arquitecturas monolíticas.

Desventajas

Complejidad en la Gestión y Coordinación: Aunque los microservicios ofrecen independencia y flexibilidad, también introducen una complejidad significativa en la gestión y coordinación. Cada servicio funciona de manera autónoma, lo que requiere mecanismos robustos para la comunicación y coordinación entre servicios. Esto implica desafíos en la orquestación de servicios, el manejo de dependencias y la consistencia de datos a través de los servicios. Además, la supervisión y el mantenimiento de múltiples servicios distribuidos pueden ser más complejos que en un sistema monolítico, lo que requiere herramientas y habilidades especializadas para la gestión efectiva de la infraestructura (Ghosh et al., 2022)

Rendimiento y Optimización: Aunque los microservicios pueden mejorar la escalabilidad, también pueden introducir latencia adicional en la comunicación entre servicios. La optimización del rendimiento en un entorno de microservicios es un desafío, ya que implica equilibrar la carga entre servicios, minimizar la latencia de la red y asegurar una utilización eficiente de los recursos. Además, la sobrecarga de la gestión de múltiples instancias de bases de datos y la complejidad de las transacciones distribuidas pueden afectar el rendimiento general del sistema (Raj & Sadam, 2021)

Desafíos de Seguridad: La seguridad en una arquitectura de microservicios es considerablemente más compleja que en un sistema monolítico. Cada microservicio puede tener sus propios requisitos de seguridad, y la comunicación entre servicios distribuidos introduce vectores de ataque adicionales. La autenticación y autorización entre servicios, la gestión de secretos y la protección de datos en tránsito y en reposo son aspectos críticos que deben abordarse cuidadosamente. Implementar una estrategia de seguridad coherente y efectiva en un entorno de microservicios es un desafío significativo y requiere una planificación y herramientas especializadas (De Almeida & Canedo, 2022)

Monitorización y el Diagnóstico: La monitorización y el diagnóstico en un entorno de microservicios son desafiantes debido a la naturaleza distribuida y dinámica de los sistemas. Rastrear y diagnosticar problemas a través de múltiples servicios y nodos requiere herramientas avanzadas de monitorización y registro, así como estrategias efectivas para la agregación y análisis de datos. La identificación de la causa raíz de los problemas puede ser complicada y requiere una visión integral del sistema y sus interacciones. (Ferreira et al., 2022)

Impacto en el Desarrollo Empresarial

En contraste, la arquitectura de microservicios se basa en la división del software en servicios más pequeños y autónomos, lo que facilita la escalabilidad y la flexibilidad. Sin embargo, esta arquitectura también introduce complejidades en la gestión y coordinación de los servicios, así como desafíos en la seguridad y el monitoreo. (Al-Debagy & Martinek, 2018)

3.2 GitSecOps: Integración, Seguridad y Despliegue continuo

3.2.1 Principios y prácticas de GitSecOps

GitSecOps representa una evolución en la gestión de proyectos de software, integrando la gestión de versiones, la seguridad y las operaciones en un flujo de trabajo continuo. Esta integración es fundamental para el manejo eficiente de los

microservicios, permitiendo una mayor agilidad y seguridad en el ciclo de vida del desarrollo de software. (Mazlami et al., 2017)

Integración de gestión de versiones con seguridad y operaciones

Control de Versiones y Colaboración: GitSecOps comienza con la integración de Git, un sistema de control de versiones distribuido, en el ciclo de vida del desarrollo de software. Git facilita la colaboración entre desarrolladores y permite un seguimiento detallado de los cambios en el código, lo que es esencial para la revisión de código y la gestión de versiones. (Ferreira et al., 2022)

Automatización y Consistencia: La automatización de procesos a través de GitSecOps asegura la consistencia y eficiencia en el despliegue de aplicaciones. Esto incluye la integración y despliegue continuos (CI/CD), que automatizan la prueba y despliegue de software, reduciendo errores humanos y acelerando el tiempo de lanzamiento al mercado. (Singh et al., 2019)

Seguridad integrada en el flujo de trabajo de desarrollo

"Shift Left" en Seguridad: GitSecOps promueve la integración de la seguridad desde las primeras etapas del desarrollo de software. Esto implica realizar análisis de seguridad del código, revisiones de seguridad y pruebas de penetración de forma temprana y continua, lo que ayuda a identificar y mitigar vulnerabilidades antes en el ciclo de vida del desarrollo. (Bertoglio et al., 2022).

Respuesta Automatizada a Incidentes de Seguridad: GitSecOps también implica la implementación de herramientas y procesos para una respuesta rápida y automatizada a los incidentes de seguridad. Esto incluye la monitorización en tiempo real, la detección de anomalías y la implementación de parches de seguridad de manera eficiente. (Basinya & Malyshev, 2023)

Mejores Prácticas y Herramientas en GitSecOps

Uso de Herramientas de Integración y Despliegue Continuos: Herramientas como Jenkins, Travis CI, GitLab CI y otros sistemas de CI/CD son fundamentales en

GitSecOps. Estas herramientas permiten la automatización de pruebas y despliegues, integrando la seguridad en cada paso. Permiten la ejecución de pipelines de CI/CD que incluyen análisis de código estático, pruebas de unidad y de integración, y despliegues automatizados, lo que es crucial para mantener la calidad y la seguridad del software. (Singh et al., 2019)

Contenedores y Orquestación: El uso de contenedores, como Docker, y sistemas de orquestación, como Kubernetes, en GitSecOps facilita la creación de entornos de desarrollo, prueba y producción consistentes y seguros. La importancia de las tecnologías de contenedores y la orquestación en la arquitectura de microservicios, especialmente en desarrollos nativos de la nube se discute en el trabajo de Nikolaevich. (Nikolaevich, 2023)

Gestión de Configuración y Secretos: La gestión eficiente de la configuración y los secretos es crucial en GitSecOps. Herramientas como Ansible, Chef o Puppet ayudan en la automatización y gestión de la configuración, mientras que servicios como Azure Key Vault o AWS Secrets Manager proporcionan soluciones seguras para la gestión de secretos. (Ferreira et al., 2022)

Cultura y Colaboración

Fomento de una Cultura de Colaboración y Responsabilidad Compartida: GitSecOps requiere un cambio cultural donde la seguridad y las operaciones son responsabilidad de todos los miembros del equipo de desarrollo. Esto implica una colaboración estrecha entre desarrolladores, ingenieros de seguridad y equipos de operaciones. La adopción de una cultura de DevOps y GitSecOps implica romper las barreras tradicionales entre estos equipos y fomentar una mentalidad de trabajo en equipo y responsabilidad compartida. Este cambio cultural es esencial para la implementación exitosa de prácticas de GitSecOps y para garantizar la seguridad y eficiencia en el desarrollo de software.(Khan et al., 2022)

Desafíos en la Adopción de la Cultura DevOps: A pesar de los beneficios de la cultura DevOps y GitSecOps, las organizaciones pueden enfrentar desafíos al adoptar estas prácticas. Estos desafíos incluyen la resistencia al cambio, la falta de

habilidades y conocimientos específicos, y la necesidad de reestructurar los procesos y flujos de trabajo existentes. Superar estos desafíos requiere un enfoque estratégico y un compromiso a nivel organizacional para capacitar a los equipos, promover la colaboración y adaptar los procesos a las nuevas prácticas. (Díaz et al., 2021)

3.2.2 Herramientas y estrategias de migración

La implementación efectiva de GitSecOps requiere una selección cuidadosa de herramientas y tecnologías. Kubernetes[1], por ejemplo, juega un papel crucial en la orquestación de contenedores, facilitando la gestión y escalabilidad de microservicios. La elección de estas herramientas debe alinearse con los objetivos específicos del proyecto y las necesidades operativas (Villamizar et al., 2017)

Herramientas de Integración y Despliegue Continuos (CI/CD)

Sistemas de CI/CD: Herramientas como Jenkins[2], GitLab CI[3], CircleCI[4] y Travis CI[5] son fundamentales en GitSecOps. Estas plataformas permiten la automatización de pruebas y despliegues, integrando la seguridad en cada etapa del desarrollo. Permiten la ejecución de pipelines de CI/CD que incluyen análisis de código estático, pruebas de unidad y de integración, y despliegues automatizados.(Singh et al., 2019)

Automatización de Pruebas: Las herramientas de prueba automatizadas, como Selenium[6], JUnit[7], y TestNG[8], son esenciales para garantizar la calidad y la seguridad

[1] **Kubernetes** : https://kubernetes.io/es/

[2] **Jenkins**: https://www.jenkins.io/doc/

[3] **GitLab CI**: https://docs.gitlab.com/ee/ci/

[4] **CircleCI**: https://circleci.com/

[5] **TravisCI**: https://www.travis-ci.com/

[6] **Selenium**: https://www.selenium.dev/

[7] **JUnit**: https://junit.org/junit5/

[8] **TestNG**: https://testng.org/

del código. Estas herramientas permiten realizar pruebas exhaustivas y consistentes, lo que es crucial para identificar errores y vulnerabilidades temprano en el proceso de desarrollo.(Golis et al., 2022)

Contenedores y Orquestación

Docker[9]: Es una herramienta clave en GitSecOps para la creación y gestión de contenedores, lo que permite a los desarrolladores empaquetar aplicaciones con todas sus dependencias en un contenedor estandarizado. Esto asegura la coherencia entre los entornos de desarrollo, prueba y producción. (Nikolaevich, 2023)

Kubernetes: Como sistema de orquestación de contenedores, Kubernetes juega un papel vital en la gestión y automatización de despliegues de microservicios. Facilita

la escalabilidad, la gestión de carga y la recuperación automática, lo que es esencial para mantener la estabilidad y la eficiencia operativa en entornos de microservicios. (Nikolaevich, 2023)

Gestión de Configuración y Secretos

Herramientas de Gestión de Configuración: Herramientas como Ansible[10], Chef [11]y Puppet [12]ayudan en la automatización y gestión de la configuración de software, lo que es crucial para mantener la coherencia y la seguridad en entornos de desarrollo complejos. Estas herramientas permiten definir y mantener la configuración deseada de los sistemas de manera automatizada, reduciendo errores y mejorando la eficiencia operativa.

[9] **Docker:** https://www.docker.com/

[10] **Ansible**: https://www.ansible.com/

[11] **Chef**: https://www.chef.io/

[12] **Puppet:** https://www.puppet.com/community/open-source

Gestión de Secretos: La seguridad de los secretos, como claves API y credenciales, es fundamental en GitSecOps. Herramientas como HashiCorp Vault[13], AWS Secrets Manager[14] y Azure Key Vault[15] ofrecen soluciones seguras para almacenar, acceder y gestionar secretos.

Seguridad Integrada

Herramientas de Análisis de Seguridad de Código: En un entorno de GitSecOps, es crucial integrar la seguridad en todas las etapas del desarrollo de software. Herramientas como SonarQube[16], Fortify[17] y Checkmarx[18] permiten el análisis estático de seguridad del código (SAST), identificando vulnerabilidades y malas prácticas en el código fuente. Estas herramientas son fundamentales para garantizar que el código desarrollado cumpla con los estándares de seguridad desde el principio, reduciendo el riesgo de vulnerabilidades en las aplicaciones.(De Almeida & Canedo, 2022)

Escaneo de Contenedores y Dependencias: Herramientas como Clair[19] y Snyk[20] ayudan a escanear contenedores y dependencias en busca de vulnerabilidades conocidas, lo que es crucial para mantener la seguridad en entornos de microservicios. Estas herramientas permiten identificar y mitigar vulnerabilidades en las imágenes de contenedores y las bibliotecas de terceros, asegurando que los microservicios sean seguros y confiables. (Raj & Sadam, 2021)

Monitorización y Registro

[13] **HashiCorp Vault**: https://www.hashicorp.com/products/vault

[14] **AWS Secrets Manager**: https://aws.amazon.com/es/secrets-manager/

[15] **Azure Key Vault**: https://azure.microsoft.com/en-us/products/key-vault

[16] **SonarQube**: https://www.sonarsource.com/products/sonarqube/downloads/

[17] **Fortify:** https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer

[18] **Checkmarx**: https://checkmarx.com/

[19] **Clair**: https://quay.github.io/clair/howto/getting_started.html

[20] **Snyk**: https://docs.snyk.io/scan-using-snyk/

Herramientas de Monitorización: Soluciones como Prometheus[21], Grafana[22] y ELK Stack[23] (Elasticsearch, Logstash, Kibana) son esenciales para la monitorización en tiempo real del rendimiento y la salud de las aplicaciones. Permiten a los equipos detectar y responder rápidamente a problemas, asegurando la alta disponibilidad y el rendimiento óptimo del sistema.(Fritzsch et al., 2019)

Gestión de Registros: La gestión eficaz de registros es crucial para el diagnóstico y la resolución de problemas. Herramientas como Fluentd[24] y Logstash[25] permiten recopilar, transformar y almacenar registros de múltiples fuentes, facilitando el análisis y la correlación de eventos. (Farsi et al., 2023)

3.3 Metodologías y estrategias de migración

3.3.1 Fase de Planificación

Esta fase es importante para establecer una base sólida para el proyecto. Se realiza una planificación detallada que incluye la definición de objetivos específicos, el alcance del proyecto y la asignación de recursos. Se establece un cronograma detallado, asignando responsabilidades a los miembros del equipo y definiendo hitos clave. Además, se identifican los módulos críticos de la arquitectura monolítica actual que serán migrados, y se desarrolla una estrategia de migración que aborda tanto los aspectos técnicos como los organizacionales. (Fritzsch et al., 2019)

3.3.2 Fase de Análisis

En esta etapa, se realiza un análisis exhaustivo de la arquitectura monolítica existente. Se identifican las áreas que requieren migración y se evalúan los posibles desafíos y obstáculos, como dependencias de componentes, problemas de rendimiento y cuestiones de seguridad. Este análisis también incluye la evaluación

[21] **Prometheus:** https://prometheus.io/

[22] **Grafana:** https://grafana.com/

[23] **ELK Stack**: https://www.elastic.co/es/elastic-stack

[24] **Fluentd**: https://www.fluentd.org/

[25] **Logstash**: https://www.elastic.co/es/logstash

de prácticas de GitSecOps existentes y cómo podrían integrarse o mejorarse en la nueva arquitectura. (Delgado Preti et al., 2021)

3.3.3 Fase de Diseño

Se diseña la nueva arquitectura basada en microservicios, definiendo la estructura, las interacciones y la gestión de los microservicios. Se seleccionan tecnologías y herramientas adecuadas para la implementación, como Kubernetes para la orquestación. Además, se diseñan estrategias para la integración y despliegue continuos, incorporando prácticas de seguridad desde el inicio del desarrollo (Preti et al., 2021).

3.3.4 Fase de Prueba de Concepto

Se realiza una prueba de concepto para validar la viabilidad técnica del diseño propuesto. Esta fase permite evaluar si los microservicios diseñados cumplen con las expectativas y requisitos de rendimiento, seguridad y escalabilidad. Se ajustan los aspectos que no cumplan con los criterios establecidos, optimizando el diseño para su implementación efectiva.(Bashtovyi & Fechan, 2023)

3.3.5 Fase de Evaluación

Se evalúa el éxito del diseño de la arquitectura, analizando el rendimiento, la escalabilidad, la seguridad y la facilidad de gestión de los microservicios modelados. Se recopilan y analizan datos para determinar si se han cumplido los objetivos del proyecto y se identifican áreas de mejora (Faria & Silva, 2022)

3.3.6 Fase de ajustes

Basándose en los resultados de la evaluación, se realizan ajustes para optimizar el diseño de los microservicios y asegurar que cumplen con los estándares de calidad requeridos. Esto puede incluir la reconfiguración de servicios, la mejora de la seguridad o la optimización del rendimiento.(Silva Filho & Figueiredo Carneiro, 2019)

3.3.7 Fase de Documentación

Se documentan todos los aspectos del proyecto, incluyendo el diseño de la arquitectura, los resultados de las pruebas, la evaluación y los ajustes realizados. Se prepara el informe final del proyecto, asegurando que se incluya toda la información relevante para futuras referencias y trabajos relacionados. (Silva Filho & Figueiredo Carneiro, 2019).

4 Desarrollo del proyecto

La fase de desarrollo del proyecto se centrará en el desarrollo de una prueba de concepto que simule la migración propuesta de los módulos de Contabilidad y Documentos Electrónicos a una arquitectura de microservicios, sin realizar la implementación directa en el sistema de la empresa.

4.1 Análisis

En el contexto actual, Manticore Labs enfrenta desafíos operativos y técnicos significativos derivados de su estructura de sistema monolítico que integra los módulos de Contabilidad y Documentos Electrónicos. Esta arquitectura, aunque funcional hasta cierto punto, limita severamente la capacidad de la empresa para adaptarse y evolucionar frente a las demandas dinámicas del mercado. Este análisis detalla las limitaciones de la arquitectura actual, la interacción entre sus componentes, y los procesos de desarrollo, despliegue y seguridad, con el objetivo de establecer un argumento sólido para la migración hacia una infraestructura basada en microservicios y optimizada mediante flujos GitSecOps.

4.1.1 Arquitectura Actual

Repositorio GIT (GitHub): El código fuente para ambos componentes se encuentran alojados en repositorios GitHub, lo que proporciona control de versiones y colaboración para el desarrollo de software.

Front-end (angular-starter): El front-end de la aplicación se desarrolla utilizando el framework Angular, lo cual facilita una interfaz de usuario dinámica, que interactúa con el back-end.

Back-end (app-nestjs-starter): Para el back-end, se utiliza NestJS,un framework de servidor progresivo basado en Node.js. Esto soporta la lógica de negocio y gestiona las solicitudes del front-end, realizando operaciones en la base de datos y devolviendo datos al usuario. Los submódulos que constituyen el back-end son:

- Submódulo facturación electrónica: Encargado de gestionar todas las operaciones relacionadas con la emisión y gestión de facturas electrónicas.
- Submódulo contabilidad: Maneja las funcionalidades clave del módulo de contabilidad, como la entrada de transacciones, la generación de balances y la administración de cuentas.
- Submódulo manti-admin: Maneja la parte de seguridad en el aplicativo.
- Submódulo valor tipo detalle: Este módulo manejar datos específicos, relacionados con elementos configurables dentro de la aplicación como parámetros y catálogos.

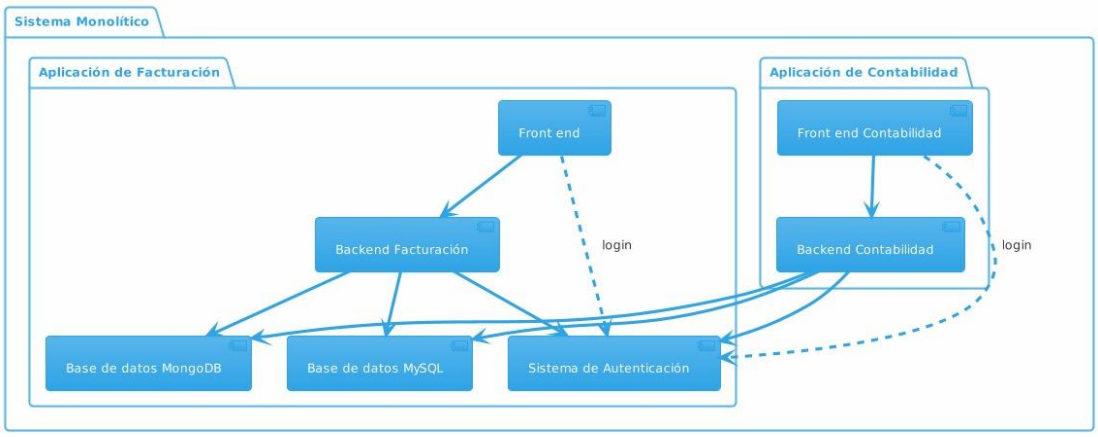


Imagen 1: Arquitectura General Manticore Labs

La aplicación hace uso de dos sistemas de gestión de bases de datos que se detallan a continuación:

MySQL: Un sistema de gestión de bases de datos relacional utilizado para almacenar y gestionar datos estructurados, esencial para las operaciones de contabilidad que requieren relaciones complejas y transacciones atómicas.

MongoDB: Una base de datos NoSQL que se utiliza para almacenar datos no estructurados o semi-estructurados, lo que es ventajoso para la gestión flexible de documentos y otros tipos de datos que no se ajustan fácilmente a un esquema relacional.

La interacción entre estos submódulos y las bases de datos (Imagen 2) es fundamental para la operación de los módulos de Contabilidad y Documentos Electrónicos, permitiendo a la aplicación manejar una amplia variedad de tareas de procesamiento de datos y gestión empresarial.

En la Imagen 1 se puede observar que el despliegue de la aplicación se realiza manualmente en un servidor On-Premise, lo que indica que no se utiliza un sistema de despliegue automatizado o en la nube, y por lo que se requiere intervención manual para las actualizaciones o el mantenimiento. Tiempo atrás se realizaba el despliegue automatizado usando la herramienta de CI/CD de GitLab, pero debido a la migración a GitHub no se ha definido la herramienta de despliegue ya que se está buscando la independencia para despliegue de integración continua.

Se representa claramente la relación y dependencia entre los distintos submódulos y su interacción con el repositorio central y los mecanismos de despliegue. Esta configuración tradicional ha servido como la espina dorsal operativa de Manticore-Labs, pero presenta desafíos significativos en términos de escalabilidad, mantenimiento y agilidad en la implementación de nuevas características.

La arquitectura actual, aunque probada y confiable hasta cierto punto, ha llegado a su límite en cuanto a capacidad de evolución para cumplir con las crecientes demandas de la empresa y su base de usuarios.

A continuación, se analizan detalladamente los problemas inherentes a la arquitectura y los procesos actuales, estableciendo el contexto para la necesidad de una transformación estructural y metodológica hacia una arquitectura de microservicios, complementada con flujos GitSecOps.

4.1.2 Limitaciones del Sistema Monolítico

Escalabilidad y Mantenimiento:

El diseño monolítico integra todas las funcionalidades en un único y masivo bloque de código, lo que dificulta la implementación de cambios y la escalabilidad del

sistema. Cualquier modificación, por menor que sea, requiere el despliegue completo de la aplicación, incrementando el riesgo de errores y tiempos de inactividad.

Agilidad Organizacional:

La rigidez estructural del sistema monolítico obstaculiza la adopción de nuevas tecnologías y procesos, lo que retrasa la capacidad de la empresa para adaptarse a las exigencias cambiantes del mercado y limita oportunidades de innovación.

Uso de Submódulos en Git:

A pesar de emplear submódulos para organizar componentes de código, esta práctica no elimina la naturaleza monolítica del sistema. Los submódulos ayudan a manejar secciones de código de manera más modular, pero su integración y despliegue siguen requiriendo coordinación centralizada, sin ofrecer la independencia operativa característica de los microservicios.

4.1.3 Limitaciones en el Flujo de Desarrollo y Despliegue

Gestión de Versiones y Branching: La gestión actual de versiones y ramas es deficiente. Falta un estándar claro de versionamiento y una estrategia de branching que apoye un ciclo de desarrollo ágil. No existen ramas dedicadas para pruebas, ni un protocolo para la eliminación de ramas una vez fusionadas, complicando así el mantenimiento y la actualización del código.

Despliegues Manuales: Los despliegues manuales dependen de la disponibilidad de personal específico y son propensos a la variabilidad y error humano, lo que retrasa la liberación de nuevas características y la resolución de problemas críticos.

Problemas de Seguridad: La seguridad de las credenciales es inadecuada, ya que no se siguen prácticas robustas de almacenamiento seguro. Esto expone al sistema a riesgos significativos de seguridad, incluyendo potenciales brechas de datos.

Discontinuidad Tecnológica: La transición de GitLab a GitHub se realizó sin establecer una herramienta adecuada para la integración y el despliegue continuos, creando un vacío en el flujo de trabajo de desarrollo y afectando negativamente la eficiencia operacional y la capacidad de implementar prácticas de desarrollo modernas.

4.2 Diseño de la arquitectura

La nueva arquitectura está diseñada para transformar los módulos existentes en microservicios completamente independientes. El objetivo principal de este cambio es facilitar una mayor escalabilidad, resiliencia y agilidad en el desarrollo y despliegue de aplicaciones. Para lograr estos objetivos, el diseño propuesto descompone el sistema en unidades más pequeñas y autónomas, cada una operando con su propio ciclo de vida independiente.

Este enfoque no solo simplifica el mantenimiento y aumenta la capacidad de adaptación a cambios, sino que también potencia la integración de nuevas funcionalidades de manera eficiente. Al minimizar los riesgos de interrupciones o fallos en las operaciones globales, se garantiza que las actualizaciones y mejoras se puedan implementar sin afectar la estabilidad del sistema en conjunto.

Además, es fundamental un acoplamiento efectivo entre todas las fases del desarrollo de software, desde la planificación del proyecto hasta el desarrollo y despliegue. El punto de partida será una estrategia de Git bien definida, basada en los tickets de tareas creados. Esta estrategia permitirá un seguimiento detallado de cada componente del proyecto, asegurando que todas las partes interesadas estén alineadas y que el progreso sea transparente y manejable.

Posteriormente, se integrarán diversas herramientas para simplificar las revisiones y asegurar un flujo de trabajo eficiente. Estas herramientas facilitarán la colaboración entre equipos, permitiendo que las revisiones de código sean más rápidas y precisas. La integración continua de estas herramientas ayudará a

detectar y resolver problemas de manera temprana, mejorando la calidad del software y acelerando los tiempos de entrega.

Finalmente, se incorporarán flujos de trabajo de GitSecOps, que representan una evolución en la gestión de versiones y seguridad. Estos flujos están diseñados para automatizar y optimizar los procesos de integración, prueba y despliegue continuos. Al incorporar prácticas de seguridad desde las primeras fases del desarrollo hasta el despliegue y operación, se asegura una arquitectura robusta y segura frente a posibles vulnerabilidades, mitigando muchos de los problemas existentes ya mencionados.

La integración de GitSecOps no solo refuerza la seguridad, sino que también mejora la eficiencia operativa en cada etapa del ciclo de desarrollo. Al automatizar tareas repetitivas y propensas a errores, se libera tiempo para que los desarrolladores se centren en aspectos más críticos del desarrollo. Además, la implementación de pruebas continuas garantiza que cualquier cambio en el código se verifique de inmediato, reduciendo el riesgo de introducir fallos en el sistema.

Para garantizar la efectividad de la nueva arquitectura, se adoptarán diversos patrones de diseño, incluyendo:

1. **Service Discovery**: Para permitir que los microservicios se encuentren y comuniquen entre sí sin necesidad de configuraciones estáticas, se implementará un mecanismo de descubrimiento de servicios. Esto facilitará la escalabilidad y flexibilidad en la gestión de los servicios.
2. **API Gateway**: Se utilizará un gateway que actuará como un punto de entrada único para todas las solicitudes hacia los microservicios. Este componente gestionará la autenticación, autorización, balanceo de carga, y otras políticas de seguridad, simplificando la arquitectura y mejorando la seguridad.
3. **Sidecar Pattern**: Para gestionar aspectos transversales como logging, monitoreo y configuración, se implementará el patrón sidecar. Este patrón permitirá añadir capacidades adicionales a los microservicios sin modificar su código principal, mejorando la modularidad y la mantenibilidad.

4. **Bulkhead**: Este patrón se adoptará para aislar los recursos entre diferentes servicios o partes de una aplicación, evitando que una falla en un componente afecte a los demás. La implementación de este patrón mejorará la resiliencia y la estabilidad del sistema.
5. **Security Patterns**: La arquitectura incorporará patrones de seguridad robustos, incluyendo la gestión y protección de secretos y datos sensibles. También se adoptarán prácticas de seguridad que asuman que todas las comunicaciones, incluso dentro de la red, son inseguras por defecto, mejorando la seguridad general del sistema.
6. **Immutable Infrastructure**: Se promoverá la provisión y despliegue de infraestructuras que no cambian después de ser creadas. Este enfoque facilitará la gestión y la seguridad de las implementaciones, garantizando que el entorno de producción sea consistente y predecible.
7. **Blue-Green Deployment**: Este patrón minimizará el tiempo de inactividad y reducirá los riesgos de despliegue manteniendo dos entornos (blue y green). Solo uno de ellos estará en producción en un momento dado, permitiendo una transición suave entre versiones.
8. **Canary Releases**: Para desplegar nuevas versiones del servicio a un subconjunto de usuarios y validar cambios antes de una implementación completa, se utilizarán lanzamientos canarios. Esto permitirá detectar y resolver problemas potenciales antes de afectar a todos los usuarios.

4.2.1 Descomposición de Módulos a Microservicios

Los módulos del aplicativo monolítico se dividirán en servicios más pequeños y manejables, en este caso se tomará los siguientes módulos.

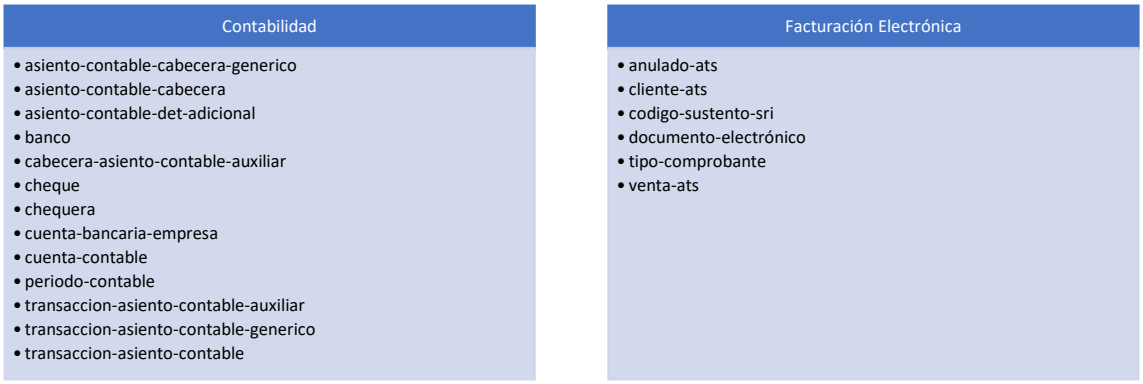


Imagen 2: Contenido de módulos seleccionados

Definición de APIs

Se diseñarán API RESTful para facilitar la comunicación entre microservicios y el front-end. Estas API se documentarán utilizando Swagger para garantizar claridad y consistencia en las interfaces de servicio.

Cada uno de estos componentes será evaluado para determinar su idoneidad como microservicio individual, basándose en criterios como la cohesión, el acoplamiento y la granularidad.

- **Microservicio Contabilidad:** Este servicio encapsulará todas las funciones del módulo de contabilidad, incluyendo el manejo de asientos contables, transacciones, cuentas bancarias, y la gestión de periodos contables. Será responsable de la integridad y el manejo seguro de la información financiera, lo cual permitirá optimizar el almacenamiento y la recuperación de datos.
- **Micro-servicio Facturación Electrónica:** A diferencia de lo que su nombre podría sugerir, este microservicio no se encargará de la emisión de facturas electrónicas. Su función principal será la de almacenar y permitir la consulta de documentos electrónicos relacionados con la facturación, como facturas guardadas, datos de clientes y ventas. Esta distinción es crucial para entender el alcance del microservicio y planificar adecuadamente las funcionalidades que deberá soportar.

Observación: Es fundamental recalcar que el módulo de Facturación Electrónica, tal como está nombrado actualmente, podría llevar a confusión respecto a sus capacidades reales. No permite la emisión directa de documentos electrónicos, sino que se limita a la gestión y visualización de los documentos almacenados. Considerar un cambio de nombre para este módulo podría clarificar su función y alinear mejor las expectativas de los usuarios con las capacidades del sistema.

Esquema de Base de Datos

Cada microservicio tendrá asociada su propio esquema dentro de una base de datos Postgres. Esto permitirá la escalabilidad y el manejo independiente de cada servicio.

Imagen 3: Esquema de Base de Datos Contabilidad

Imagen 4: Esquema de Base de Datos de Facturación Electrónica

4.2.2 Diagramas C4

En el desarrollo y documentación de la arquitectura de sistemas complejos, es crucial contar con herramientas efectivas que permitan una visualización clara y comprensible de las estructuras internas y las interacciones externas.

Esta sección abarca los tres primeros niveles del Diagrama C4: Contexto, Contenedores y Componentes. Cada nivel aporta una visión única e indispensable sobre la estructura del sistema, ofreciendo claridad a una amplia gama de partes interesadas. Al desglosar la arquitectura y funcionalidad del sistema en estos niveles detallados, facilitamos una comprensión profunda y accesible de cómo el sistema está organizado y cómo opera dentro de su entorno.

En el marco del presente proyecto de titulación, migración del sistema de Facturación Electrónica. Este sistema es fundamental para la generación, autenticación y gestión de documentos electrónicos en transacciones comerciales, asegurando el cumplimiento de normativas fiscales y proporcionando transparencia en los procesos comerciales.

El sistema de Facturación Electrónica se implementa como un "plugin" dentro del sistema de Ecommerce MLabs de Manticore Labs. Esto significa que la solución de

facturación electrónica opera dentro del entorno del sistema de Ecommerce, aprovechando su infraestructura de seguridad y autenticación. El sistema de Ecommerce MLabs no solo facilita las operaciones de venta en línea, sino que también gestiona de manera robusta la autenticación y seguridad de las transacciones.

Diagrama de contexto

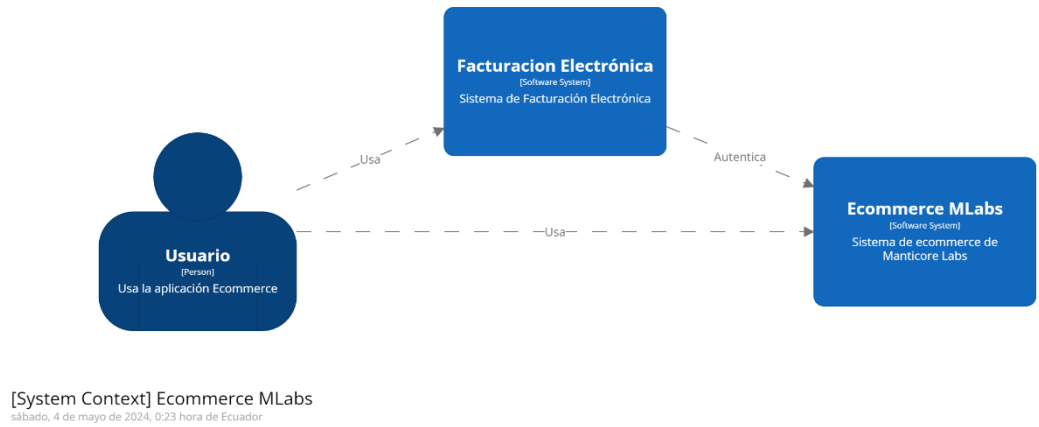


Imagen 5: Diagrama de contexto

Este diagrama permite visualizar las interacciones de alto nivel entre el usuario y los sistemas involucrados, proporcionando una vista general de cómo los componentes interactúan dentro del ecosistema. A continuación, se detalla cada elemento del diagrama:

Usuario (Persona): Representa a la persona que interactúa con el sistema de Ecommerce. El usuario accede y utiliza la plataforma de Ecommerce para navegar productos, realizar compras y otras actividades relacionadas con el comercio electrónico.

Sistema de Facturación Electrónica (Software System): Este sistema maneja todos los aspectos relacionados con la facturación electrónica dentro del ecosistema de

Ecommerce. Es responsable de generar facturas válidas digitalmente que cumplen con los estándares y regulaciones para la facturación electrónica.

A través del sistema Ecommerce, recibe detalles de las transacciones del usuario para procesar y generar documentos electrónicos. Además, asegura que todas las transacciones y datos generados sean auténticos y cumplan con las normativas necesarias.

Sistema de Ecommerce MLabs (Software System): Plataforma de comercio electrónico proporcionada por Manticore Labs, diseñada para facilitar la venta de productos o servicios en línea. Este sistema es la interfaz principal para los usuarios y actúa como un puente entre el usuario y el sistema de facturación electrónica.

- Permite a los usuarios realizar operaciones de ecommerce como seleccionar productos, añadirlos al carrito, y proceder al pago.
- Transmite información de las transacciones del usuario al sistema de facturación electrónica para la generación de documentos electrónicos.
- Recibe datos de confirmación de los documentos generados, asegurando que el proceso se complete adecuadamente antes de finalizar las transacciones en la plataforma.

Flujos de Información

Del Usuario al Sistema de Ecommerce Mlabs: El usuario interactúa con la plataforma de Ecommerce ingresando y enviando información, que incluye seleccionar productos y realizar pedidos.

Del Sistema de Ecommerce MLabs al Sistema de Facturación Electrónica: El Ecommerce MLabs envía detalles de la transacción al Sistema de Facturación Electrónica para procesar y generar los documentos electrónicos correspondiente.

Del Sistema de Facturación Electrónica al Sistema de Ecommerce MLabs: Una vez que el documento es generado y validado, el Sistema de Facturación Electrónica

envía una confirmación de vuelta al Sistema de Ecommerce MLabs para proceder con la finalización de la compra

Diagrama de contenedores

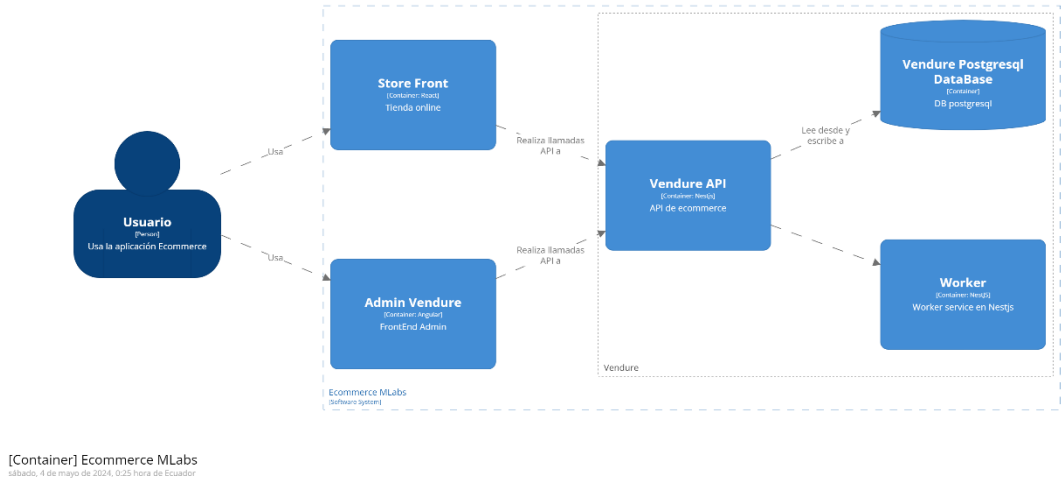


Imagen 6: Diagrama de contenedores - MLabs

El diagrama proporciona una descripción detallada de los contenedores (componentes) que conforman el sistema, destacando cómo los usuarios interactúan con la plataforma y cómo los diversos componentes están interconectados para soportar las operaciones de e-commerce. A continuación, se desglosa cada componente con la función que realiza dentro del sistema:

Usuario: Representa a las personas que interactúan con la plataforma e-commerce de MLabs. Los usuarios pueden ser compradores en la tienda online o administradores que gestionan el backend del e-commerce.

Store Front (Contenedor React): Contenedor que aloja la interfaz de la tienda online, donde los usuarios pueden navegar por los productos, añadirlos al carrito y realizar compras. Está construido utilizando React, una biblioteca de JavaScript para construir interfaces de usuario de forma eficiente y escalable. Se comunica

mediante llamadas API con el contenedor Vendure API para obtener información de productos, gestionar el carrito de compras y procesar transacciones.

Admin Vendure (Contenedor Angular): Contenedor que proporciona la interfaz de administración del sistema, permitiendo a los administradores gestionar productos, configuraciones del sistema y órdenes de compra. Está desarrollado con Angular, framework de JavaScript diseñado para aplicaciones web dinámicas. Al igual que el Store Front, el Admin Vendure interactúa con el Vendure API para realizar operaciones administrativas.

Vendure API (Contenedor de Commerce API): Actúa como el cerebro del sistema de ecommerce, procesando todas las solicitudes provenientes tanto del Store Front como del Admin Vendure. Este API maneja la lógica del negocio, incluyendo la gestión de productos, clientes, órdenes y pagos. Lee desde y escribe en la base de datos: Interactúa directamente con la base de datos Vendure PostgreSQL para recuperar y almacenar datos. Se Comunica con el contenedor Worker envíando tareas, como la generación de reportes o la ejecución de procesos que no deben bloquear las respuestas inmediatas a los usuarios.

Vendure PostgreSQL Database (DB PostgreSQL): Base de datos principal del sistema de ecommerce, donde se almacenan todos los datos relacionados con el comercio, incluidos detalles de productos, información de clientes, órdenes y transacciones. Vendure API realiza consultas y actualizaciones a esta base de datos para mantener actualizado el estado de la tienda y las transacciones de los usuarios.

Worker (Contenedor NestJS): Contenedor que ejecuta tareas de fondo que son necesarias para el funcionamiento del ecommerce pero que son demasiado pesadas para procesar en el hilo principal de ejecución de la aplicación. Esto incluye tareas como el procesamiento de pagos, la generación de reportes y otras operaciones batch. Procesa trabajos asincrónicos enviados desde Vendure API, lo que ayuda a mejorar el rendimiento y la escalabilidad del sistema al distribuir la carga de trabajo.

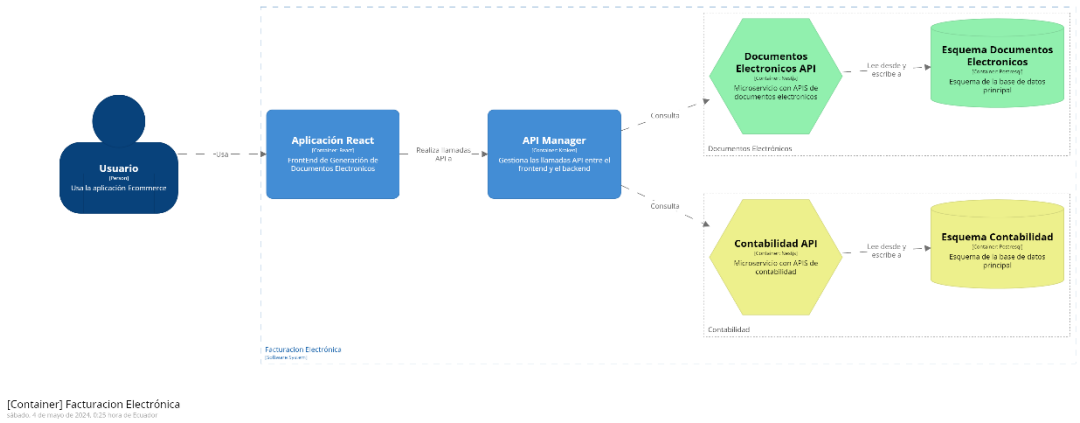


Imagen 7: Diagrama de contenedores – Facturación Electrónica

Usuario: Representa a los individuos que utilizan la aplicación de Ecommerce. Estos usuarios pueden estar interactuando con el sistema para realizar compras, generar facturas o gestionar su información contable.

Aplicación React: Contenedor que aloja el frontend de generación de documentos electrónicos, desarrollado en React. Permite a los usuarios interactuar directamente con el sistema de facturación electrónica, facilitando una interfaz gráfica para la creación y gestión de documentos. Se comunica con el API Manager para realizar operaciones que afectan los datos de facturación y contabilidad.

API Manager (Contenedor Kraken): Sirve como intermediario entre el frontend y los microservicios de backend, gestionando y enrutando las llamadas API. Este componente es clave para modularizar el acceso a las funcionalidades del backend, asegurando que las interacciones entre el frontend y los servicios de backend sean fluidas y seguras. Envía y recibe datos desde los microservicios Documentos Electrónicos API y Contabilidad API para procesar las solicitudes del usuario.

Documentos Electrónicos API (Contenedor NestJS): Microservicio diseñado específicamente para gestionar todas las operaciones relacionadas con los documentos electrónicos, como facturas, recibos, y otros documentos relacionados con transacciones. Está construido utilizando NestJS, un framework de Node.js para

construir aplicaciones de servidor eficientes y escalables. Interactúa directamente con su base de datos correspondiente para obtener y almacenar información de documentos.

Contabilidad API (Contenedor NestJS): Este microservicio maneja todas las operaciones relacionadas con la contabilidad, incluyendo el seguimiento de transacciones, balances y otros datos contables. También está construido con NestJS, proporcionando una arquitectura robusta para operaciones complejas de contabilidad. Gestiona datos contables en su base de datos dedicada, asegurando que todas las operaciones contables sean precisas y estén actualizadas.

Bases de Datos (Esquema Documentos Electrónicos y Esquema Contabilidad): Estos contenedores PostgreSQL almacenan los datos de documentos electrónicos y contabilidad, respectivamente. Estas bases de datos son esenciales para mantener la integridad de los datos y proporcionar acceso eficiente y seguro a la información necesaria para las operaciones del sistema. Cada base de datos es accesada exclusivamente por su API correspondiente para operaciones de lectura y escritura, manteniendo una separación clara de las preocupaciones y una mejor seguridad de los datos.

DIAGRAMA DE COMPONENTES

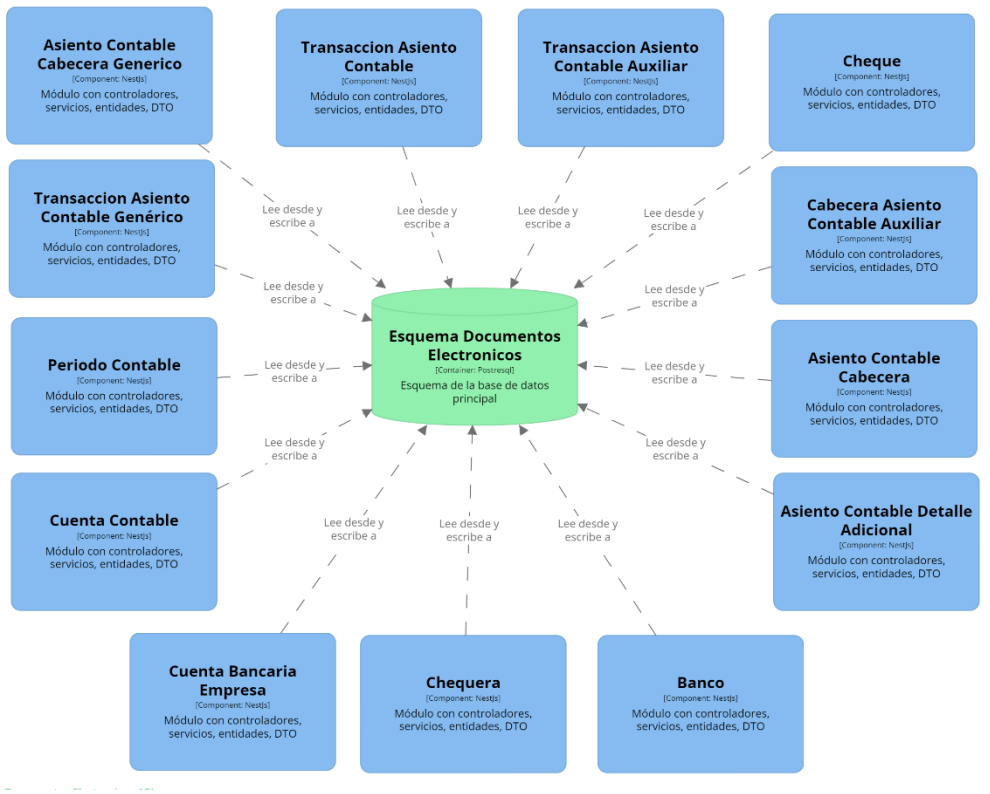


Imagen 8: Diagrama de componentes –Documentos Electrónicos

Cada uno de estos componentes representa una parte funcional del sistema de gestión contable y está diseñado para manejar aspectos específicos de la contabilidad y las transacciones financieras:

Transacción Asiento Contable, Transacción Asiento Contable Genérico, Transacción Asiento Contable Auxiliar: Estos componentes manejan diferentes tipos de transacciones contables. Cada uno está especializado en un tipo de asiento contable, permitiendo una gestión detallada y específica según las necesidades de la transacción. Cada módulo procesa datos relacionados con su tipo específico de transacción y comunica los resultados o cambios al esquema de base de datos principal.

Periodo Contable, Cuenta Contable, Cuenta Bancaria Empresa, Chequera, Cheque, Banco: Estos componentes gestionan diversos aspectos de la contabilidad y las finanzas de la empresa. Desde el seguimiento de períodos contables, la

administración de cuentas contables y bancarias, hasta la gestión de cheques y chequeras. Cada uno interactúa directamente con el esquema de base de datos, leyendo y escribiendo información necesaria para llevar a cabo operaciones contables y transacciones financieras.

Cabecera Asiento Contable Auxiliar, Asiento Contable Detalle Adicional, Asiento Contable Cabecera, Asiento Contable Cabecera Genérico: Estos componentes se centran en los detalles y cabeceras de los asientos contables, proporcionando estructuras para capturar la información detallada de las transacciones contables y sus respectivas categorizaciones. Similar a los otros módulos, interactúan con la base de datos para actualizar y recuperar información que sustenta las operaciones contables.

Esquema de Base de Datos (Esquema Documentos Electrónicos): Este componente de base de datos actúa como el repositorio central donde se almacena toda la información procesada y gestionada por los diferentes módulos contables. Recibe y almacena datos de todos los componentes, asegurando que la información esté consistentemente actualizada y disponible para consultas y reportes.

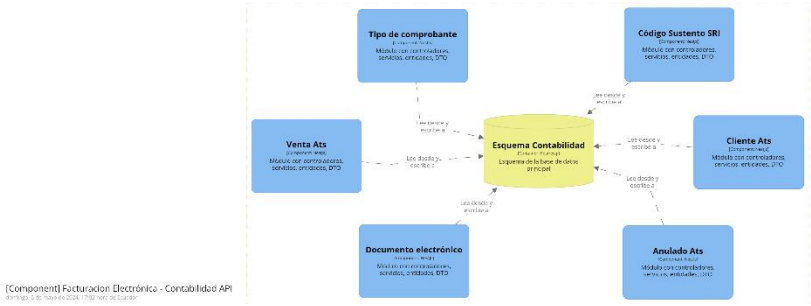


Imagen 9: Diagrama de componentes 2

Venta Ats: Este módulo gestiona las ventas y transacciones. Incluye controladores, servicios y entidades que permiten procesar y registrar las ventas en el sistema. Comunica información de transacciones a la base de datos central, ayudando a mantener registros consistentes de ventas.

Tipo de Comprobante: Gestiona los diferentes tipos de comprobantes que pueden ser emitidos en el sistema, como facturas, notas de crédito, etc. Almacena y recupera información de los tipos de comprobantes desde la base de datos, asegurando que los documentos generados cumplan con las especificaciones legales y del sistema.

Documento Electrónico: Este módulo maneja la creación y gestión de documentos electrónicos, asegurando que se generen de acuerdo con las normas vigentes. Interactúa directamente con la base de datos para almacenar y recuperar los documentos electrónicos procesados.

Código Sustento SRI: Administra los códigos de sustento del Servicio de Rentas Internas (SRI), que son necesarios para validar ciertas operaciones tributarias y contables. Este componente lee y escribe en la base de datos para actualizar y consultar los códigos de sustento, manteniendo la integridad fiscal de las transacciones.

Cliente Ats: Controla la información relacionada con los clientes dentro del sistema, desde la creación de cuentas hasta la gestión de datos de clientes. Realiza operaciones de lectura y escritura en la base de datos para mantener actualizada la información del cliente.

Anulado Ats: Gestiona la anulación de transacciones y documentos, proporcionando funcionalidades para marcar documentos como anulados en el sistema. Comunica con la base de datos para actualizar el estado de los documentos y transacciones.

Esquema de Contabilidad (Container: PostgreSQL): Esquema de la base de datos central donde se almacenan todos los datos relacionados con las transacciones, documentos electrónicos, clientes, tipos de comprobantes, y más. Recibe y almacena datos de todos los componentes mencionados, asegurando que la información esté consistentemente actualizada y disponible para consultas y reportes.

Seguridad y Autorización

Se integrarán mecanismos de seguridad a nivel de microservicio, utilizando tokens JWT y OAuth para la autenticación y autorización, asegurando que cada solicitud entre servicios sea verificada y segura.

4.3 Flujo GitSecOps

En la presente sección se detalla el flujo de trabajo propuesto para la implementación de GitSecOps en los proyectos de desarrollo de software. Este enfoque busca integrar prácticas de desarrollo y operaciones con un fuerte énfasis en la seguridad desde el inicio del ciclo de vida del software. A continuación, se describe cada una de las etapas clave del proceso, desde la planificación inicial hasta el monitoreo y optimización en producción. El diagrama adjunto ilustra visualmente este flujo, mostrando cómo las diferentes herramientas y equipos interactúan para asegurar una entrega continua y segura de las funcionalidades desarrolladas.

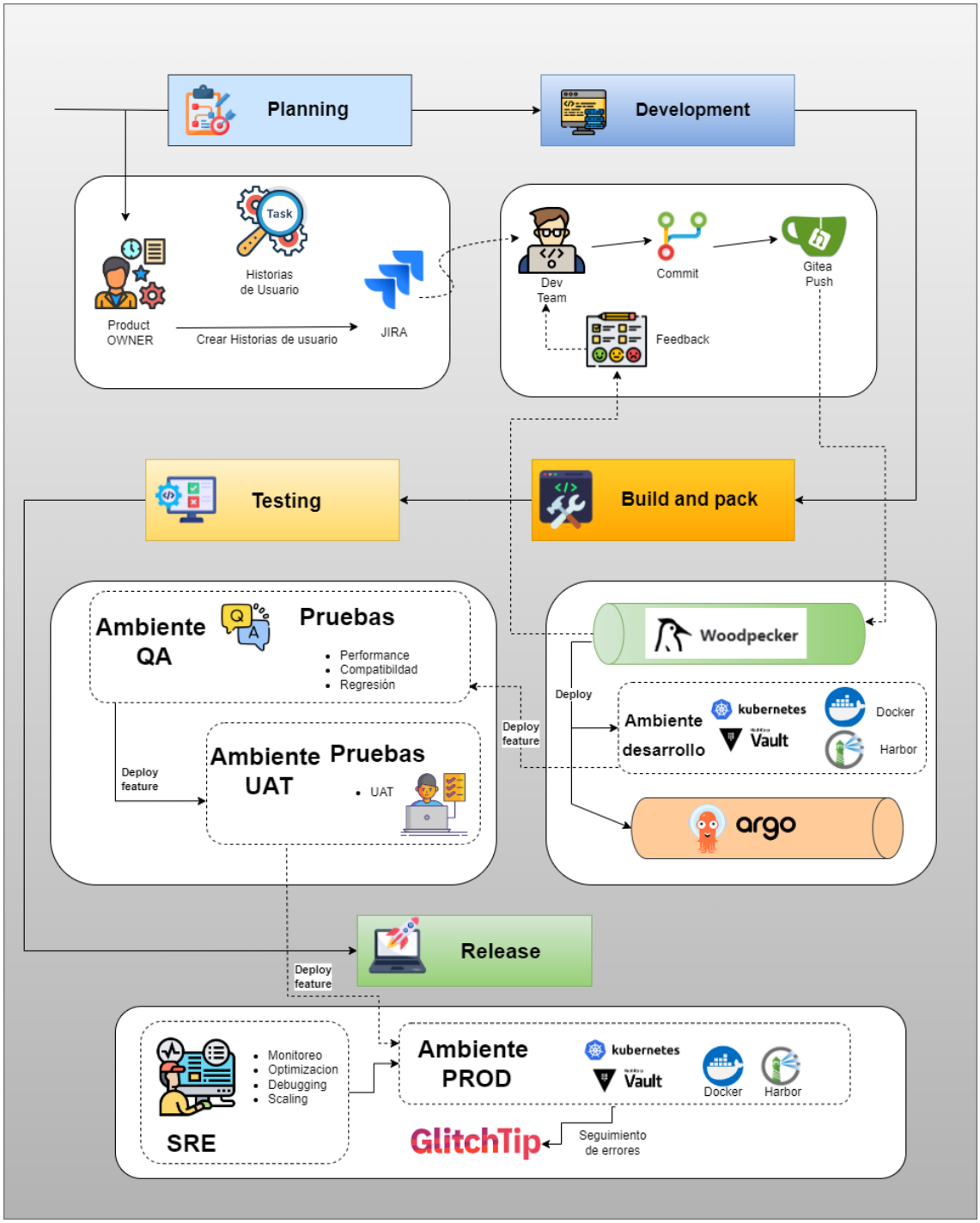


Imagen 10: Flujo GitSecOps propuesto

4.3.1 Planificación

Product Owner y JIRA: La fase de planificación es decisivo para el éxito de cualquier proyecto de desarrollo de software. Comienza con el Product Owner, quien desempeña un papel clave al definir y documentar las historias de usuario en JIRA. Este proceso implica detallar las necesidades funcionales y técnicas del proyecto, asegurando que todos los requisitos sean claros, precisos y alineados con los objetivos del negocio. La utilización de JIRA aquí es vital, ya que permite una gestión eficaz de las tareas y facilita la comunicación transparente y el seguimiento del progreso entre los equipos.

Revisión de Requisitos: Una vez que las historias de usuario están definidas, el equipo lleva a cabo una revisión exhaustiva para asegurar la comprensión y viabilidad de estas. Esta revisión es fundamental para identificar posibles desafíos o áreas que necesitan clarificación antes de proceder al desarrollo. También permite al equipo hacer ajustes necesarios para alinear mejor las expectativas con las capacidades técnicas y los recursos disponibles, garantizando que el proyecto avance sobre bases sólidas y realistas.

4.3.2 Desarrollo

Estrategia de Gestión y Versionamiento del Código: Se propone una estrategia renovada de gestión y versionamiento del código para optimizar la eficiencia y fortalecer la colaboración en el desarrollo de software. Tradicionalmente, los módulos de Contabilidad y Facturación Electrónica se alojaban en un único repositorio en GitHub, careciendo de una estructura que permitiera manejar los distintos componentes de manera efectiva. Ahora, optaremos por Gitea, una plataforma de gestión de repositorios Git ligera y eficiente, que ofrece un control detallado sobre la privacidad y seguridad del código fuente. Además, se implementará la estrategia de Monorepo, centralizando ambos microservicios en un solo repositorio de código. Para gestionar este Monorepo de forma efectiva, se empleará la biblioteca Turbo.build, que proporciona herramientas avanzadas para la construcción y optimización de proyectos. La unión entre Gitea, la estrategia de

Monorepo y Turbo.build creará un entorno de desarrollo robusto y flexible, facilitando la dinámica colaborativa inherente al desarrollo de microservicios y garantizando una gestión uniforme y segura de la seguridad y las versiones del código.

Entorno de Desarrollo Integrado: Durante la fase de desarrollo, emplearemos Gitea para gestionar el código fuente. Esta plataforma no solo fomenta un entorno colaborativo y eficiente entre los desarrolladores, sino que también permite la realización de commits y push de cambios de manera continua, esencial para integraciones y revisiones de código regulares. Su interfaz intuitiva y capacidad para apoyar flujos de trabajo distribuidos son clave en un entorno de microservicios, donde equipos múltiples colaboran en componentes distintos del sistema.

Estrategia de Git: Adoptaremos una estrategia de Git Flow rigurosa para maximizar la eficiencia y la colaboración durante el proceso de desarrollo. Cada ticket de JIRA corresponderá a una rama específica en el repositorio, dedicada exclusivamente al desarrollo de la tarea asignada. Al completar las tareas, se realizará un pull request para evaluar las modificaciones sugeridas, incluyendo una revisión de código exhaustiva por parte de otros desarrolladores. Este paso es decisivo para garantizar que las mejoras no solo sean óptimas, sino que también estén libres de errores. Una vez aprobado el pull request y fusionado el código al branch principal, se eliminará la rama utilizada, manteniendo el repositorio organizado y limpio. Estas prácticas son esenciales para asegurar un desarrollo ágil y efectivo, reduciendo errores y elevando constantemente la calidad del software desarrollado.

Feedback Continuo: La colaboración y el feedback continuo son esenciales para mantener la calidad del desarrollo. Las revisiones de código en Gitea permiten que los miembros del equipo proporcionen y reciban retroalimentación regular sobre el trabajo realizado, asegurando que todos los cambios cumplan con los estándares establecidos y las mejores prácticas de la industria. Este proceso no solo mejora la calidad del código, sino que también fomenta un ambiente de aprendizaje y adaptación continua, decisivo para el desarrollo ágil.

Service Discovery: Para permitir que los microservicios se encuentren y comuniquen entre sí sin necesidad de configuraciones estáticas, se implementará un mecanismo de descubrimiento de servicios, integrado en Kubernetes (MicroK8). Esto facilitará la escalabilidad y flexibilidad en la gestión de los servicios, asegurando que cada componente del sistema pueda localizar a los demás de manera dinámica y eficiente.

API Gateway: Se utilizará KrakenD como API Gateway, actuando como un punto de entrada único para todas las solicitudes hacia los microservicios. Este componente gestionará la autenticación, autorización, balanceo de carga, y otras políticas de seguridad, simplificando la arquitectura y mejorando la seguridad. El gateway servirá como una interfaz robusta entre los clientes y los microservicios, proporcionando una capa adicional de gestión y control.

Sidecar Pattern: Para gestionar aspectos transversales como logging, monitoreo y configuración, se implementará el patrón sidecar utilizando GliptichP. Este patrón permitirá añadir capacidades adicionales a los microservicios sin modificar su código principal, mejorando la modularidad y la mantenibilidad. Los sidecars actuarán como contenedores auxiliares que corren junto a los servicios principales, manejando tareas como la captura de logs y la recolección de métricas.

Bulkhead: Este patrón se adoptará para aislar los recursos entre diferentes servicios o partes de una aplicación utilizando pods en Kubernetes. Cada pod tendrá su propio conjunto de recursos y podrá operar independientemente de los otros. La implementación de este patrón mejorará la resiliencia y la estabilidad del sistema, asegurando que los recursos críticos se mantengan operativos incluso si otros componentes enfrentan problemas.

Security Patterns: La arquitectura incorporará patrones de seguridad robustos, incluyendo la gestión y protección de secretos y datos sensibles mediante Vault. También se adoptarán prácticas de seguridad que asuman que todas las comunicaciones, incluso dentro de la red, son inseguras por defecto, mejorando la

seguridad general del sistema. La integración de estos patrones asegurará que todas las capas del sistema estén protegidas contra accesos no autorizados y vulnerabilidades.

Immutable Infrastructure: Se promoverá la provisión y despliegue de infraestructuras que no cambian después de ser creadas, utilizando ArgoCD. Este enfoque facilitará la gestión y la seguridad de las implementaciones, garantizando que el entorno de producción sea consistente y predecible. Las infraestructuras inmutables permitirán que las actualizaciones y despliegues sean más seguros y menos propensos a errores.

Blue-Green Deployment: Este patrón minimizará el tiempo de inactividad y reducirá los riesgos de despliegue manteniendo dos entornos (blue y green). Solo uno de ellos estará en producción en un momento dado, permitiendo una transición suave entre versiones. Herramientas como Argo Rollouts se utilizarán para gestionar estos despliegues, asegurando una alta disponibilidad y continuidad del servicio.

Canary Releases: Para desplegar nuevas versiones del servicio a un subconjunto de usuarios y validar cambios antes de una implementación completa, se utilizarán lanzamientos canarios. Esto permitirá detectar y resolver problemas potenciales antes de afectar a todos los usuarios, mejorando la calidad del software y reduciendo riesgos asociados a los nuevos despliegues.

4.3.3 Construcción y Empaquetado

El proceso de construcción y empaquetado del proyecto de contabilidad y facturación electrónica sigue una serie de pasos cuidadosamente diseñados para asegurar eficiencia, seguridad y escalabilidad.

Una vez finalizado el desarrollo, el código se valida utilizando **Woodpecker**. Se inicia un pipeline en esta herramienta, que realiza el linting del código para asegurar que cumple con los estándares de calidad establecidos. Esta etapa es esencial para

detectar y corregir cualquier inconsistencia o error de codificación antes de proceder al empaquetado. Si el linting se completa con éxito, se procede a compilar y empaquetar el aplicativo tanto para el backend como para el frontend. Esto se realiza creando imágenes Docker que contienen todas las dependencias y configuraciones necesarias para ejecutar la aplicación.

Estas imágenes **Docker** empaquetadas se almacenan en **Harbor**, un registro seguro para contenedores. Harbor no solo asegura que las imágenes estén disponibles para su despliegue en los diferentes entornos, sino que también las versiona, facilitando un seguimiento claro y preciso de las distintas versiones del aplicativo. La gestión de versiones es esencial para mantener un control riguroso sobre las actualizaciones y cambios en la aplicación.

En la fase de despliegue, Kubernetes juega un papel neural. Utilizando las imágenes almacenadas en Harbor, orquesta el despliegue de las aplicaciones en los entornos de desarrollo, testing y producción. Cada entorno tiene un subdominio específico que permite la validación de las aplicaciones de manera aislada y controlada. Durante este proceso, **Vault** se encarga de la gestión segura de secretos, como credenciales y claves API, necesarios para el funcionamiento seguro de las aplicaciones.

Finalmente, **ArgoCD** automatiza el proceso de entrega continua. Esta herramienta asegura que las versiones correctas de las aplicaciones se implementen en los entornos correspondientes, utilizando las imágenes Docker almacenadas y versionadas en Harbor. ArgoCD permite que los despliegues se realicen de manera automatizada, garantizando que las aplicaciones se actualicen y distribuyan sin interrupciones, manteniendo así la eficiencia y seguridad del proceso de despliegue.

Este conjunto de procesos interconectados asegura que la construcción y despliegue de la aplicación de contabilidad y facturación electrónica se realice de manera eficiente, segura y escalable, manteniendo la integridad y calidad del software en todo momento.

5 Resultados y discusión

5.1 Herramienta y Tecnologías

Para llevar a cabo esta prueba de concepto, se seleccionó herramientas que soporten la creación, despliegue y monitorización de microservicios en un entorno controlado:

- **Microk8s**: Distribución de Kubernetes, diseñada para entornos de desarrollo, pruebas y producción en pequeña escala. Permite a los usuarios implementar rápidamente un clúster de Kubernetes en un solo nodo, facilitando la gestión de contenedores y aplicaciones en entornos locales.[26]
- **Woodpecker**: Herramienta de integración continua (CI) y entrega continua (CD) para proyectos de software. Permite automatizar el proceso de construcción, prueba y despliegue de aplicaciones de forma eficiente y confiable. [27]
- **Harbor**: Proporciona un lugar centralizado para almacenar y distribuir imágenes de contenedores, lo que facilita la colaboración y la implementación de aplicaciones en entornos de contenedores.[28]
- **Argo CD**: Herramienta de entrega continua (CD) de código abierto utilizado para automatizar la implementación de aplicaciones en entornos de Kubernetes.[29]
- **Gitea**: Plataforma de autoservicio de Git de código abierto, que permite a los equipos de desarrollo alojar y gestionar repositorios de código de Git de forma local o en la nube[30]
- **Lens**: Herramienta de código abierto que proporciona una interfaz gráfica unificada para administrar clústeres de Kubernetes. Permite a los usuarios

[26] **Microk8s**: https://microk8s.io/docs/getting-started
[27] **Woodpecker**: https://woodpecker-ci.org/
[28] **Harbor**: https://goharbor.io/
[29] **Argo CD**: https://argo-cd.readthedocs.io/en/stable/
[30] **Gittea:** https://about.gitea.com/

visualizar y administrar recursos de Kubernetes de manera intuitiva, lo que facilita la supervisión, depuración y gestión de aplicaciones en entornos de contenedores.[31]

- **Vault**: Herramienta de código abierto que proporciona una interfaz unificada para gestionar secretos como contraseñas, certificados, token de acceso etc. [32]
- **Glitchtip**: Plataforma de seguimiento de errores y monitoreo de aplicaciones, que permite registrar y gestionar problemas, recibir alertas sobre errores importantes, colaborar con colegas y dar seguimiento al progreso de la resolución de incidencias.[33]
- **Jira**: Herramienta de gestión de proyectos y seguimiento de problemas. Permite a los equipos a planificar, organizar y controlar sus proyectos de manera efectiva. Además, ofrece diversas funcionalidades como la creación de tareas, asignación de responsabilidades, seguimiento del progreso y generación de informes.[34]

5.2 Implementación

Configuración y Despliegue en Micro8s

En nuestra arquitectura, la implementación y gestión del entorno de Kubernetes se realiza a través de MicroK8s. La configuración inicial y la administración del clúster se llevan a cabo con una serie de comandos que preparan el entorno para nuestras necesidades operativas.

Instalación de MicroK8s:

```
sudo snap install microk8s –classic
microk8s status --wait-ready
```

Imagen 11: Instalación de Microk8s

[31] **Lens:** https://docs.k8slens.dev/getting-started/install-lens/#debian
[32] **Vault**: https://www.vaultproject.io/
[33] **Glitchtip:** https://glitchtip.com/
[34] **Jira:** https://www.atlassian.com/es/software/jira

Configuración de Acceso del Usuario:

```
cd $HOME
mkdir .kube
cd .kube
microk8s config > config
sudo usermod -a -G microk8s $USER
sudo chown -f -R $USER ~/.kube
su - $USER
```

Imagen 12: Configuración de Acceso del usuario

Habilitación de Servicios Esenciales:

```
microk8s enable rook-ceph
microk8s enable dns
microk8s enable metrics-server
microk8s enable observability
microk8s enable metallb:192.168.0.87-192.168.0.87
microk8s enable dashboard
microk8s enable ingress
microk8s enable cert-manager
```

Imagen 13: Habilitación de servicios esenciales

```
microk8s kubectl apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: le-prod-prueba
  namespace: prueba
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: prueba@manticore-labs.com
    privateKeySecretRef:
      name: le-prod-prueba-ksr
    solvers:
      - http01:
          ingress:
            class: public
EOF
```

Imagen 14: Configuración de Seguridad con Cert-Manager

Integración con Repositorios y Gestión de Artefactos: Se implementa Woodpecker y Harbor dentro del namespace 'prueba' para gestionar la integración continua y el almacenamiento seguro de artefactos:

```
microk8s.helm install woodpecker --namespace prueba
-f ./woodpecker/values.yaml woodpecker/woodpecker
microk8s.helm install argo-cd -f ./argo-cd/values.yaml --namespace prueba
--create-namespace argo/argo-cd --version 5.46.3
```

Imagen 15: Instalación de woodpecker y ArgoCD usando helm en microk8s

Gestión de Secretos para Registros de Docker:

```
kubectl create secret docker-registry regcred
--docker-server=https://harbor-pampa-pod.manticore-labs.com/
--docker-username=robot$pampa-pod-robot-lib-2 --docker-password=3x8fvI0u3sdf40dj908elYi0c
--docker-email=prueba@manticore-labs.com
```

Imagen 15: Instalación de woodpecker y ArgoCD usando helm en microk8s

Configuración de Dominios: Se configura los dominios en el panel de control, donde cada servicio dentro del clúster tiene asignado un subdominio específico. Esta organización facilita el acceso y la gestión de diferentes servicios como Argo CD, Harbor, y GlitchTip, asegurando que cada componente sea fácilmente accesible y esté debidamente segregado dentro del entorno de red (Ip publica).

Domain	Document Root	Redirects To	Force HTTPS Redirect	Actions
argo-pampa-pod-dex.mantic...	/public_html	Not Redirected		Manage Create Email
argo-pampa-pod.manticore-l...	/public_html	Not Redirected		Manage Create Email
argocd-webhook-pampa-pod...	/public_html	Not Redirected		Manage Create Email
glitchtip-logs-pampa-pod.ma...	/public_html	Not Redirected		Manage Create Email
harbor-pampa-pod.manticor...	/public_html	Not Redirected		Manage Create Email
notary-harbor-pampa-pod.m...	/public_html	Not Redirected		Manage Create Email
pgadmin-pampa-pod.mantic...	/public_html	Not Redirected		Manage Create Email
server-pampa.manticore-labs...	/public_html	Not Redirected		Manage Create Email
woodpecker-pampa-pod.ma...	/public_html	Not Redirected		Manage Create Email

Imagen 16: Dominios configurados en Hosting24

Woodpecker CI: Interfaz inicial en la que se lista el repositorio "manticore-labs/test-woodpecker-ci", que se muestra que Woodpecker está siendo utilizado para gestionar las tareas de CI para este proyecto. Estas tareas se configuran generalmente a través de un archivo woodpecker.yml que se coloca en el repositorio de código. Este archivo define todos los pasos que el sistema CI debe ejecutar

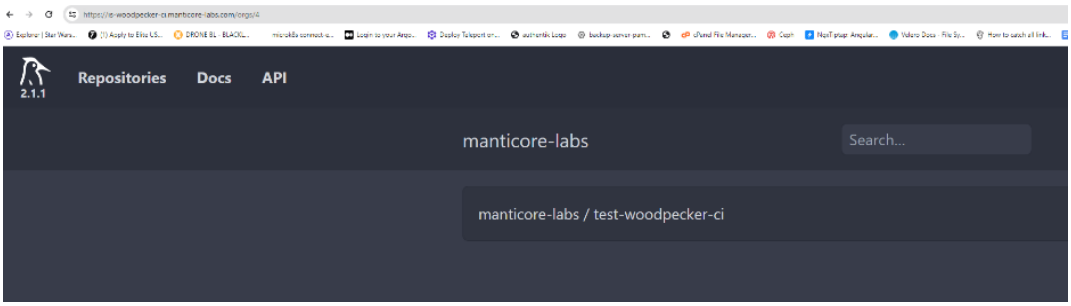


Imagen 17: Pagina inicial de Woodpecker

Versionamiento: Woodpecker se integra directamente con los sistemas de control de versiones y se activa automáticamente cuando se realizan cambios en el repositorio, como nuevos commits o pull requests.

Esto facilita una metodología de desarrollo de software ágil, ya que cada cambio puede ser automáticamente construido y probado, asegurando que los errores se detecten y corrijan rápidamente.

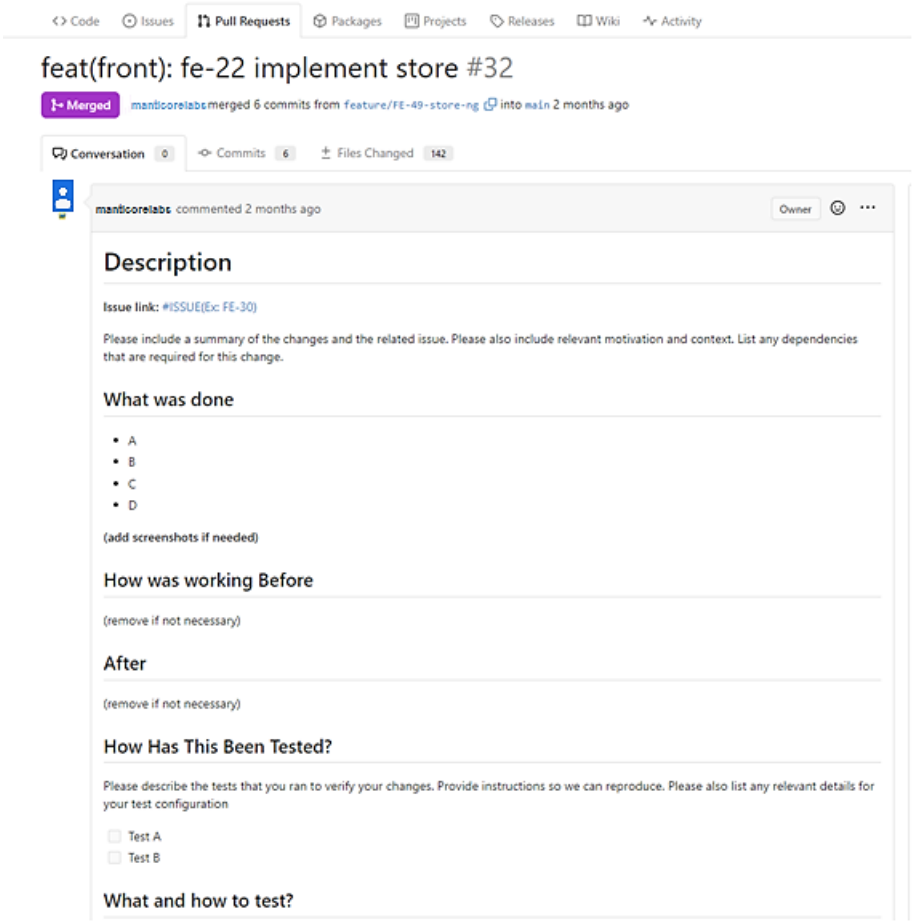


Imagen 18: Creación del Pull Request al repositorio

Adicional en cada petición de “Pull Request” se creó un template garantizando una idea clara de lo que se está implementando/corrigiendo, en la cual se incluye lo siguiente:

- **Descripción**: En donde se agrega el link referencial al ticket
- **Qué se hizo**: Se detalla explicación de cómo se implementó o resolvió el problema.
- **Cómo funcionaba antes:** Detallar el problema
- **Después**: Detallar cómo funciona ahora
- **Cómo fue testeado:** Indicar la secuencia de pasos seguidos para testear la funcionalidad.
- **Que y como se debe testear**: Indicar la secuencia de pasos para testear y que partes se debe validar. Por ejemplo, si necesita hacer configuraciones adicionales.
- **Tipo de cambio**: Bug fix, New feature, Breaking change, Documentation update. Estos tipos de cambios se han basado en las convenciones generales de commits.

Database

- [ ] Se ejecutó migraciones en los entornos deseados (dev, test, etc)
- [ ] Se ha probado la aplicación después de ejecutar las migraciones

Checklist:

- [ ] Mi código sigue las pautas de estilo de este proyecto
- [ ] He realizado una revisión personal de mi código
- [ ] He comentado mi código, especialmente en áreas difíciles de entender
- [ ] He realizado cambios correspondientes en la documentación
- [ ] Mis cambios no generan nuevas advertencias
- [ ] He añadido pruebas que demuestran que mi corrección es efectiva o que mi característica funciona

- [ ] No se necesitan pruebas de unidad o de extremo a extremo en este momento
- [ ] Las pruebas unitarias existentes y nuevas pasan localmente con mis cambios
- [ ] La verificación de lint pasa localmente con mis cambios

[ ] He creado los archivos de Storybook para los nuevos componentes.

También, se valida la empaquetación de la versión lanzada una vez finalizado los pipelines.

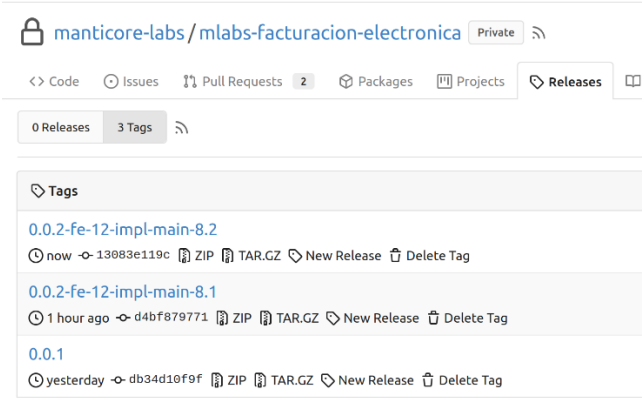


Imagen 19: Versionamiento en el repositorio

Pipelines en Woodpecker: Se puede visualizar los pipelines ejecutados en Woodpecker, donde se ven las tareas específicas como 001_lint_prettier_test y 002_build_prerelease. Estos pasos confirman que cada cambio pasa por un linting para mantener la calidad del código y luego por un proceso de construcción antes de ser considerado para el lanzamiento.

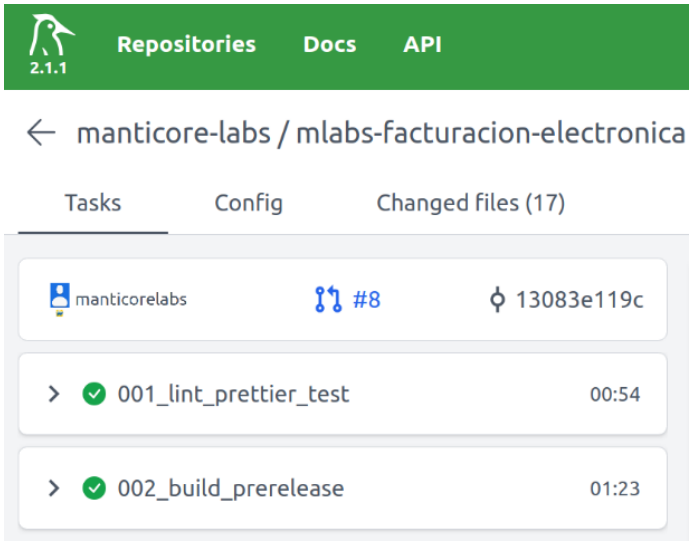


Imagen 20: Pipeline en woodpecker

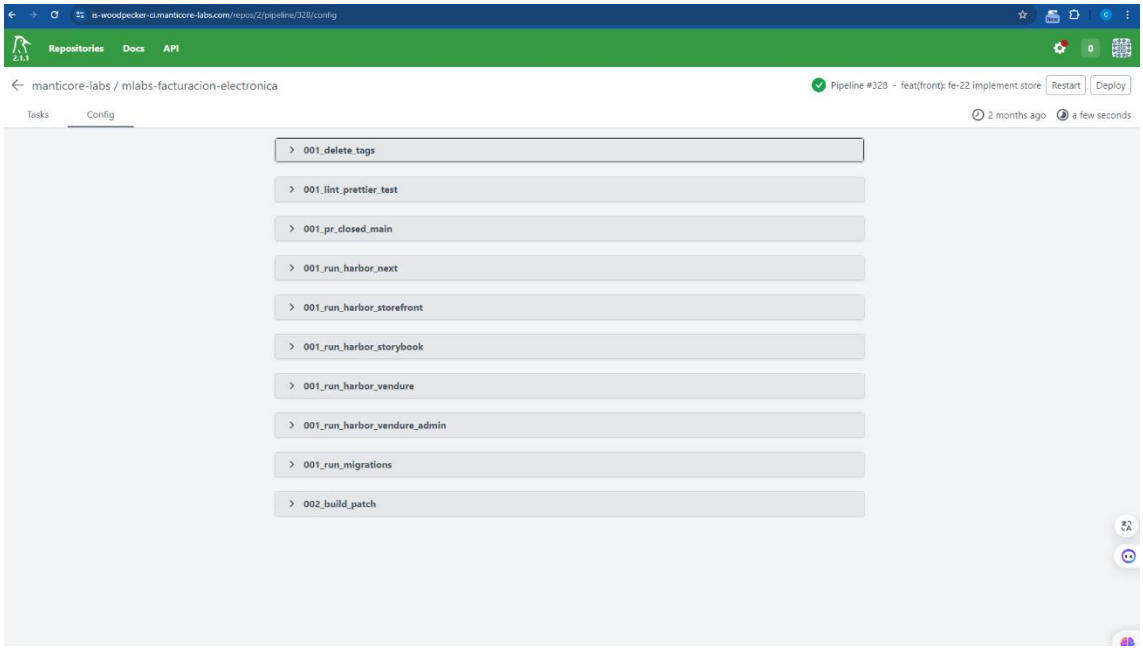


Imagen 21: Configuraciones en Woodpecker

Gestión de Artefactos: Panel de Harbor utilizado para almacenar y gestionar artefactos de contenedores de forma segura. Permite escanear imágenes de contenedores en busca de vulnerabilidades, asegurando que solo los artefactos seguros sean desplegados

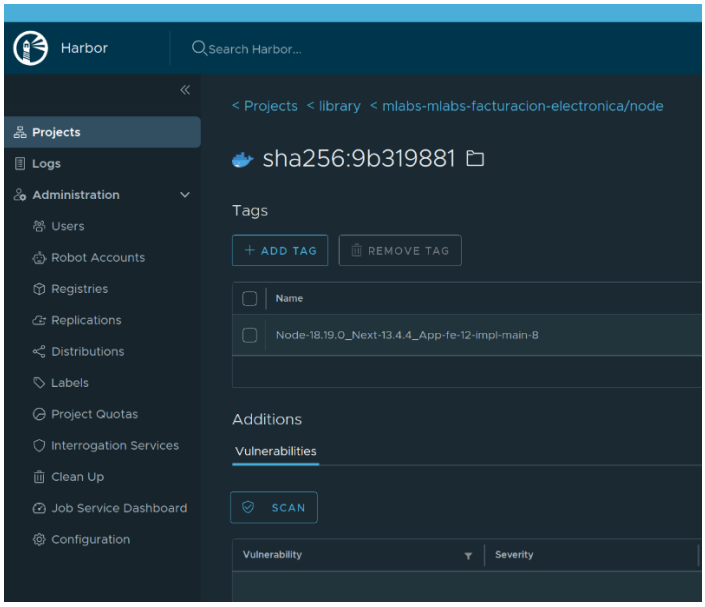


Imagen 22: Pantalla de harbor

ArgoCD:

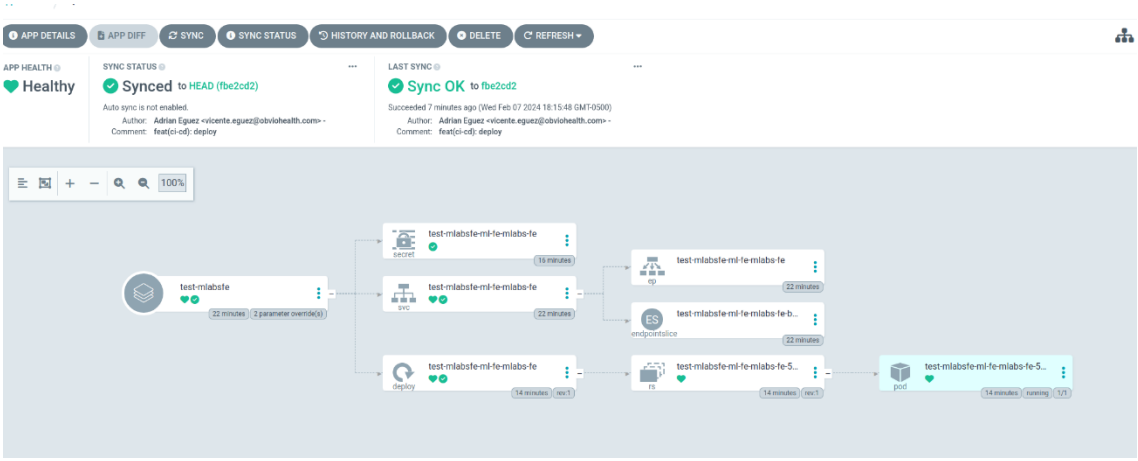


Imagen 23: Pantalla de Argo CD

En este caso se detalla la aplicación como test-mlabsfe con los siguientes nodos:

- **Secret**: El nodo tipo Secret es esencial para la gestión de datos sensibles. Este recurso de Kubernetes almacena información confidencial, como claves API, tokens de acceso y contraseñas, de forma segura y cifrada.
- **Service (Svc):** El nodo Service actúa como el intermediario para el acceso a los pods que ejecutan las instancias de la aplicación. Este nodo de servicio proporciona un punto de acceso estable y una dirección IP fija,

a pesar de que los pods subyacentes puedan ser reemplazados o reiniciados. Esto asegura que la aplicación permanezca accesible de manera consistente a través de un punto fijo.

- **Deployment (Deploy):** El nodo Deployment es importante para mantener la consistencia y la fiabilidad de la aplicación mediante la gestión del despliegue de pods. Este nodo asegura que el número deseado de instancias de la aplicación esté corriendo y se actualice automáticamente en caso de cambios en el código o en la configuración.

Cada nodo está diseñado para cumplir con funciones específicas dentro del ecosistema Kubernetes, asegurando que test-mlabsfe opere eficientemente y con la seguridad requerida. El uso de ArgoCD permite visualizar cuantos nodos se encuentran activos, el estado de salud en general y el estado de sincronización de la aplicación.

Monitorización con GlitchTip: Se envía un evento de prueba llamado “EVENTO PRUEBA” para el registro específico de un error o excepción que es capturado durante las pruebas de la aplicación. Esta herramienta proporciona un código de evento único, lo que facilita la referencia y el seguimiento del problema específico.

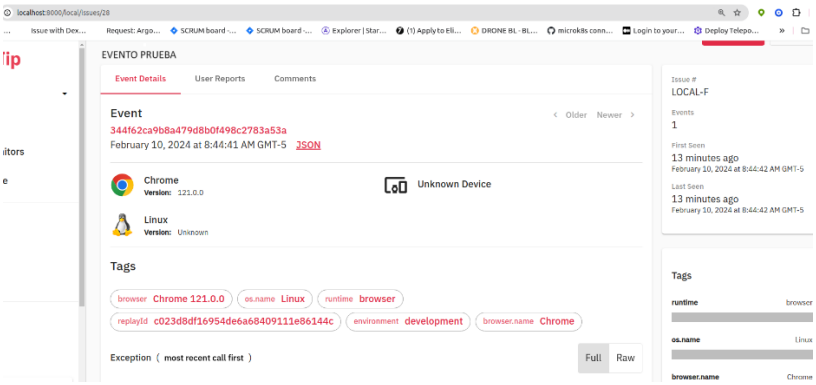


Imagen 24: Prueba de evento en GlitchTip

Monitoreo de tiempo de actividad: Esta funcionalidad es significativo para monitorear la disponibilidad y la salud de las aplicaciones en tiempo real.

- **Frontend Monitor**: Este monitor está configurado para rastrear la disponibilidad y el tiempo de respuesta del frontend de la aplicación. La

URL apunta a la interfaz de usuario o frontend del servicio, permitiendo a GlitchTip verificar regularmente si está accesible y funcionando correctamente.

- **Backend Monitor:** Similar al Frontend Monitor, pero enfocado en el backend de la aplicación, verificando la salud del servicio a través de la URL que apunta a un endpoint de salud. Este endpoint devuelve la respuesta de estado que indica si el backend está operativo.

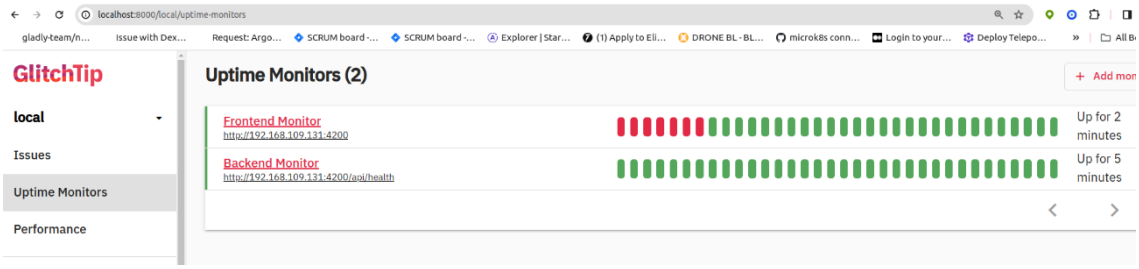


Imagen 25: Monitor de frontend y backend

Esta implementación detallada asegura que este entorno Kubernetes está configurado para proporcionar robustez, seguridad y flexibilidad, facilitando un desarrollo y despliegue eficientes de nuestras aplicaciones. La integración de estos sistemas no solo mejora nuestra capacidad de respuesta frente a los cambios, sino que también asegura que se mantiene un ciclo de desarrollo continuo y seguro.

5.3 Testing

QA y UAT (User Acceptance Testing): Las pruebas de QA se centran en asegurar que el software no solo cumpla con los requisitos técnicos, sino que también funcione según lo previsto en varios entornos y configuraciones. Este proceso es decisivo para detectar problemas de rendimiento, compatibilidad y regresión antes de que el software llegue a los usuarios finales. Aunque actualmente no utilizamos herramientas de prueba automatizada open source para realizar pruebas unitarias o de regresión y nos limitamos a la ejecución de análisis estático del código (linting), la importancia de estas pruebas es reconocida y se considera un área de mejora para el futuro. Paralelamente, las pruebas de Aceptación por el Usuario (UAT) involucran directamente a los usuarios finales, quienes evalúan el software en un

entorno controlado para confirmar que cumple con sus necesidades y expectativas. La UAT es esencial para garantizar la aceptabilidad del software, proporcionando un feedback valioso que puede utilizarse para realizar ajustes finales antes del lanzamiento oficial.

Gestión de Despliegues y Recuperación ante Fallos:

Durante la fase de testing, también se implementó un robusto proceso de gestión de despliegues utilizando Argo. Cada nueva funcionalidad desarrollada se empaqueta en una imagen de Docker y se despliega mediante Argo para pruebas. Esto no solo facilita la implementación de pruebas automatizadas en el futuro en un entorno aislado, sino que también permite un control preciso sobre las versiones del software. Si durante las pruebas se detecta algún error o si algo se rompe, el sistema automáticamente revierte a la imagen de Docker anterior, garantizando así la estabilidad y continuidad del servicio. Este enfoque proporciona una capa adicional de seguridad, permitiendo que los equipos respondan rápidamente a problemas sin afectar a los usuarios finales.

Recomendaciones para la Mejora de las Pruebas:

Es recomendable evaluar y adoptar herramientas de pruebas automatizadas que puedan integrarse en nuestro flujo de trabajo de desarrollo para mejorar la cobertura y eficiencia de las pruebas, especialmente en lo que respecta a pruebas unitarias y de regresión.

Una opción a considerar es implementar la capa gratuita de SonarQube, una herramienta poderosa que ofrece análisis de código estático y detección de vulnerabilidades en varias tecnologías. La versión gratuita proporciona análisis detallados del código fuente, identificando problemas de calidad, errores potenciales y malas prácticas. Esto puede ayudar a mejorar la calidad general del código al identificar problemas en etapas tempranas del desarrollo, lo cual permite al equipo hacer ajustes antes de que el código llegue a producción. Adicionalmente, la integración de SonarQube con el pipeline actual podría automatizar estos análisis, haciendo que la detección de problemas sea parte del proceso de CI/CD, lo cual

aumentaría significativamente la eficiencia del proceso de desarrollo y garantizaría altos estándares de calidad desde las fases más tempranas del desarrollo.

5.4 Liberación

Ambientes de Pruebas y Producción: Después de pasar las pruebas de QA y UAT, las características serán desplegadas en el ambiente de producción, usando herramientas como **Argo** para gestionar los despliegues progresivos, lo que permite una implementación controlada y minimiza los riesgos asociados con la liberación de nuevas funcionalidades. Este enfoque ayuda a asegurar una transición suave y estable del nuevo software al entorno de producción, lo que es crucial para la continuidad del negocio y la satisfacción del usuario.

Monitoreo con GlitchTip y SRE (Site Reliability Engineering): Una vez que el software está en producción, el monitoreo se vuelve crítico. **GlitchTip** se utiliza para rastrear y analizar errores en tiempo real, mientras que las prácticas de **SRE** se centran en la optimización, depuración y escalabilidad del sistema. Estas actividades son esenciales para mantener la estabilidad y el rendimiento del software, asegurando que cualquier problema pueda ser identificado y resuelto rápidamente para minimizar el impacto en los usuarios finales.

5.5 Evaluación

El proceso de evaluación del rendimiento y la satisfacción de la aplicación de facturación electrónica se realizó en dos contextos distintos: la evaluación real en el servidor de la empresa y la evaluación del caso de prueba en una máquina virtual. La evaluación real se llevó a cabo en el servidor on-premise utilizando logs de PM2 y datos de uso del sistema, mientras que la evaluación del caso de prueba se realizó en un entorno virtualizado diseñado para simular condiciones operativas reales. Además, se evaluó el flujo de trabajo real y el flujo GitSecOps del caso de prueba mediante formularios de Google Forms aplicados a los integrantes de la empresa.

A continuación se detalla la configuración de recursos de los equipos:

Recurso	Servidor On-Premise	Máquina Virtual (Caso de Prueba)
Memoria	128 GB	19.7 GB
Procesadores	Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz (10 núcleos, 20 subprocesos)	8 CPUs
Disco Duro	No especificado	500 GB SCSI
CD/DVD	No especificado	SATA
Adaptador de Red	No especificado	Configuración puenteada (Bridged)
Sistema Operativo	No especificado	Linux - Ubuntu

Evaluación de Módulos a Microservicios

La adopción de la arquitectura de microservicios se espera que supere las limitaciones identificadas en el sistema monolítico, mejorando así la escalabilidad, el mantenimiento y la agilidad organizacional. A continuación, se detalla cómo se evalúan estas limitaciones mediante la implementación de microservicios.

En el sistema monolítico, cada despliegue de la aplicación requería aproximadamente 1 hora y resultaba en la interrupción total del sistema. En el caso de prueba, el despliegue de un microservicio individual, como el de contabilidad, se realiza en 10 minutos sin afectar los otros servicios. Esta mejora se debe a la utilización de herramientas detalladas en la sección 4.3, que automatizan el proceso de despliegue. La capacidad de desplegar microservicios de forma independiente reduce significativamente los tiempos de inactividad y el riesgo de errores. Esto mejora la escalabilidad y el mantenimiento del sistema al permitir actualizaciones rápidas y específicas sin interrumpir la operatividad global.

La evaluación de la agilidad organizacional se centró en la rapidez y efectividad con la que los equipos pueden responder a cambios y demandas del mercado. En el sistema monolítico, los cambios en cualquier parte de la aplicación requerían pruebas extensivas y despliegues completos, lo que ralentizaba la respuesta. Tras la migración a microservicios, los equipos pueden desarrollar, probar y desplegar cambios en sus respectivos servicios de manera autónoma. Esto fue evaluado mediante encuestas a los desarrolladores y gerentes de proyecto, utilizando herramientas como Google Forms para recopilar datos sobre la rapidez y facilidad de implementación de cambios.

En el sistema monolítico, la integración de submódulos y el despliegue completo de la aplicación eran procesos complejos y propensos a errores, realizados aproximadamente una vez por semana. Con la adopción de la estrategia de Monorepo utilizando Gitea y Turbo.build, cada microservicio tiene su propio repositorio gestionado dentro del monorepo, permitiendo despliegues diarios y más frecuentes sin problemas de coordinación centralizada. La independencia operativa de los microservicios facilita la integración y el despliegue, reduciendo la complejidad y los riesgos asociados con la coordinación centralizada.

La estructura modular y flexible de los microservicios fomenta la innovación y la adopción rápida de nuevas tecnologías y procesos, mejorando la capacidad de la empresa para adaptarse a las exigencias del mercado. En complemento con el flujo GitSecOps diseñado para este caso de prueba que se detalla a continuación.

Evaluación del Flujo de Trabajo

Además del rendimiento de la arquitectura, se evaluó también el flujo de trabajo real y el flujo GitSecOps del caso de prueba. Se utilizaron formularios de Google Forms para recopilar datos cuantitativos de los integrantes de Manticore Labs, evaluando sus percepciones y experiencias.

Las preguntas están diseñadas para evaluar la facilidad de integración de nuevas funcionalidades, la eficiencia del proceso de despliegue, la efectividad en la

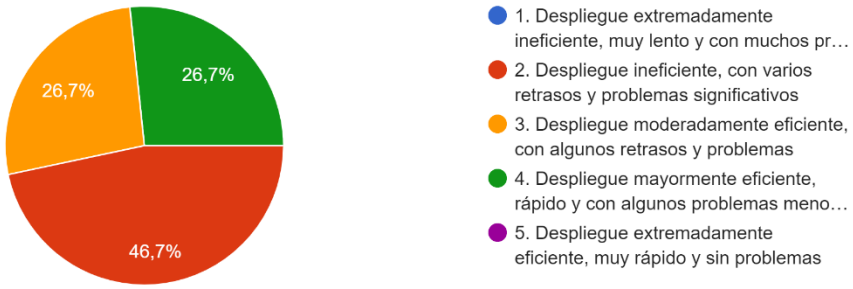
detección y resolución de errores, la satisfacción general con el flujo de trabajo y la adaptabilidad del sistema a cambios y actualizaciones. (Anexo 1)

A continuación, se presentan los resultados obtenidos en una muestra de 15 personas de acuerdo a la estadística de Google Forms:

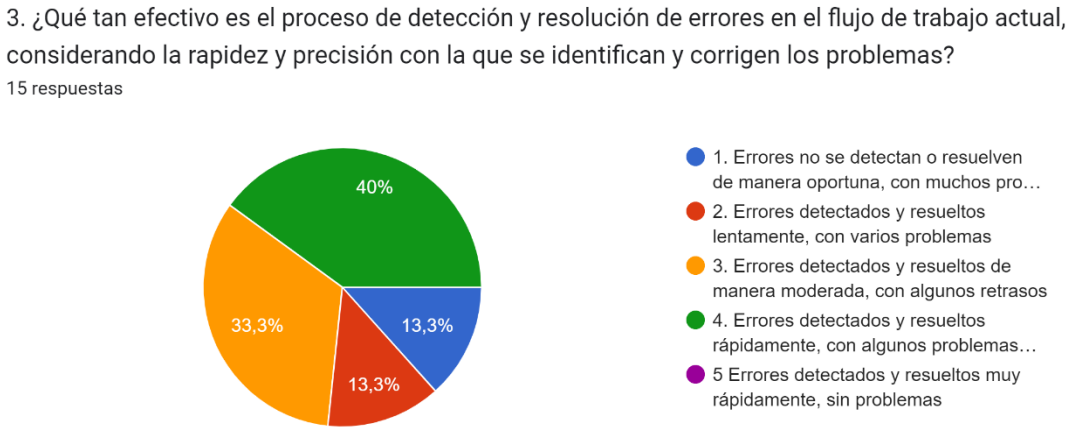
1. ¿Qué tan fácil le resulta integrar nuevas funcionalidades en el flujo de trabajo actual, considerando la colaboración entre equipos y la implementación en los sistemas existentes?
15 respuestas

La mayoría de los encuestados encuentra dificultades significativas en la integración de nuevas funcionalidades, con un 46.7% calificando la integración como difícil o extremadamente difícil. Solo un 26.7% considera la integración mayormente fácil o extremadamente fácil. Esto indica una clara necesidad de mejorar los procesos de integración, facilitando mejores herramientas, procesos más ágiles y una colaboración más eficaz entre equipos.

2. ¿Qué tan eficiente considera el proceso de despliegue en el flujo de trabajo actual, en términos de tiempo requerido y la cantidad de pasos necesarios?
15 respuestas



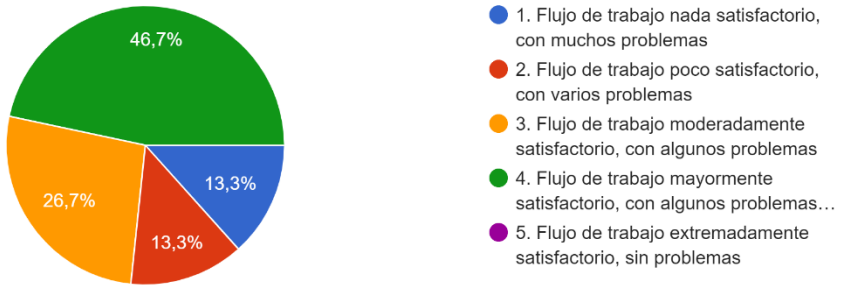
El proceso de despliegue en el flujo de trabajo actual es visto en general como ineficiente por la mayoría de los encuestados, con el 46.7% calificándolo como ineficiente y otro 26.7% calificándolo como moderadamente eficiente. Solo un 26.7% lo considera mayormente eficiente. La ausencia de calificaciones extremas (tanto extremadamente ineficiente como extremadamente eficiente) sugiere que el proceso no es completamente disfuncional, pero tampoco está optimizado.



El proceso de detección y resolución de errores en el flujo de trabajo actual es considerado moderadamente efectivo por la mayoría de los encuestados, con un 33.3% calificándolo como moderado y un 40% como rápido, pero con algunos problemas menores. Sin embargo, un 26.6% de los encuestados enfrenta problemas significativos, lo que indica una clara necesidad de mejora. No hubo calificaciones de máxima eficiencia, lo que sugiere que hay un considerable margen para optimizar el proceso.

4. ¿Qué tan satisfecho está con el flujo de trabajo general en el entorno actual, considerando todos los aspectos del desarrollo, despliegue y mantenimiento?

15 respuestas



La mayoría de los encuestados encuentran el flujo de trabajo general mayormente satisfactorio, con el 46.7% calificándolo así. Un 26.7% lo encuentra moderadamente satisfactorio. Sin embargo, un 26.6% de los encuestados está insatisfecho, calificando el flujo de trabajo como poco o nada satisfactorio. No hubo calificaciones de extrema satisfacción, lo que aún es posible mejorar el flujo de trabajo y elevar los niveles de satisfacción de todos los integrantes de la empresa.

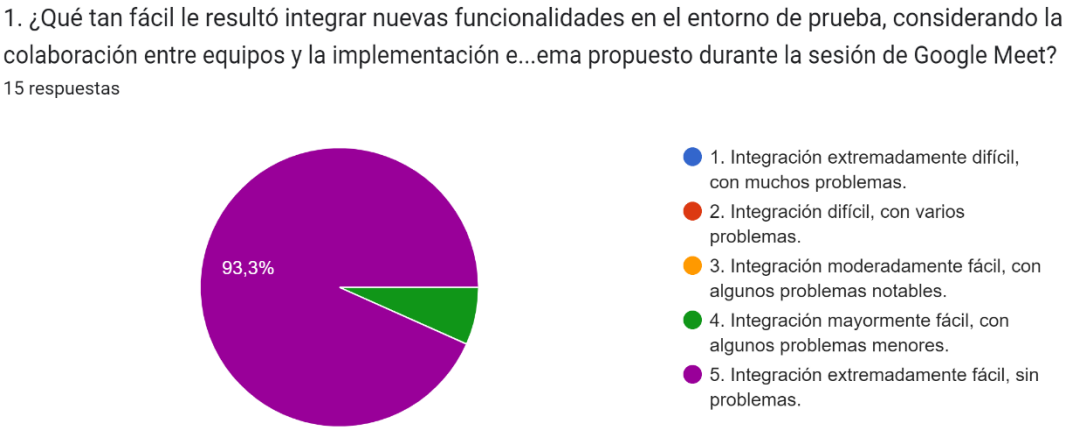
5. ¿Qué tan bien se adapta el conjunto de aplicaciones, herramientas y procesos (sistema) a cambios y actualizaciones en el flujo de trabajo ac... causar interrupciones o problemas significativos?

15 respuestas

El 53.3% de los encuestados creen que el sistema es mayormente adaptable. Un 20% piensan que es moderadamente adaptable. Sin embargo, un 26.6% está insatisfecho, calificando el sistema como poco o nada adaptable. No hubo calificaciones de extrema adaptabilidad, lo que sugiere que aún hay un considerable margen para mejorar la flexibilidad y capacidad del sistema para manejar cambios y actualizaciones sin causar interrupciones significativas.

Para el caso de prueba, la evaluación se centrará en la observación directa del funcionamiento del flujo GitSecOps propuesto. Esta propuesta se presentó a todos los integrantes de la empresa mediante una reunión virtual a través de la plataforma de Google Meets. Los asistentes observaron todo el proceso de despliegue y las interacciones. Posterior a esto, se realizó la evaluación con los mismos criterios de la encuesta anterior. (Anexo 2)

A continuación, se muestra los resultados obtenidos:

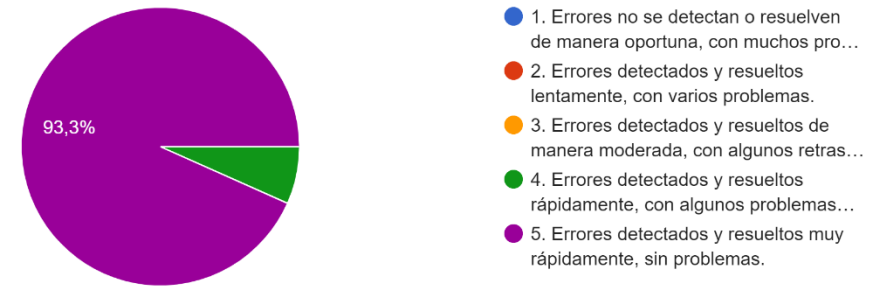


La propuesta indica resultados muy positivos, con un 93.3% de los encuestados reportando que la integración de nuevas funcionalidades fue extremadamente fácil y sin problemas. Lo cual indica que las condiciones y herramientas del entorno de prueba son altamente efectivas para facilitar la integración de nuevas funcionalidades. La ausencia de respuestas que indiquen dificultades importantes sugiere que los métodos y herramientas utilizados fueron apropiados y bien implementados. Este alto nivel de satisfacción demuestra que, bajo las condiciones del caso de prueba, los equipos pudrían colaborar eficazmente y realizar integraciones sin encontrar barreras significativas.

2. ¿Qué tan eficiente consideró el proceso de despliegue en el entorno de prueba, en términos de tiempo requerido y la cantidad de pasos necesario...egún lo demostrado en la sesión de Google Meet?
15 respuestas

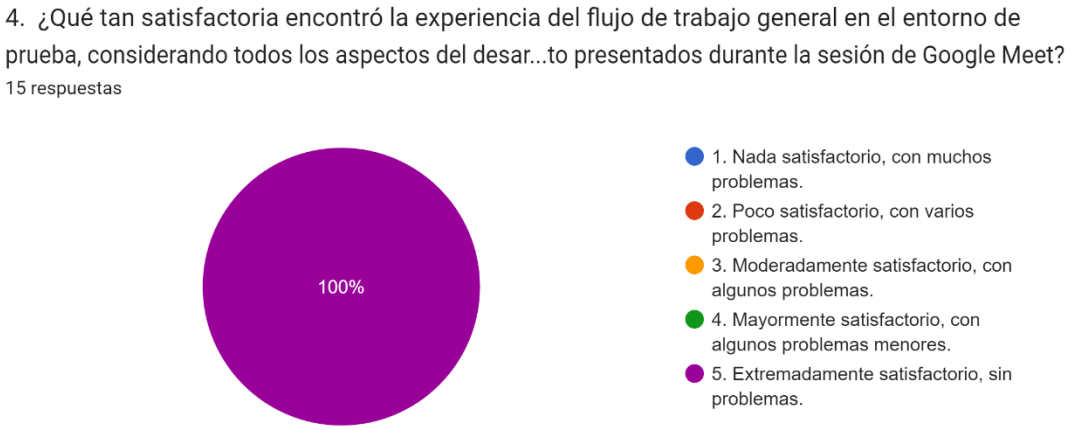
La eficiencia del proceso de despliegue muestra resultados extremadamente positivos con el 100% de los encuestados calificándolo como extremadamente eficiente, muy rápido y sin problemas. Este resultado unánime recalca la efectividad del entorno de prueba para manejar despliegues de manera rápida y sin contratiempos. No hay respuestas que indiquen problemas en el proceso de despliegue. Aunque los asistentes no realizaron este proceso por sí mismos, los encuestados pudieron observar un proceso optimizado que minimiza errores y tiempos de espera, asegurando una entrega continua y confiable del software.

3. ¿Qué tan efectivo fue el proceso de detección y resolución de errores en el entorno de prueba, considerando la rapidez y precisión con la que se i... durante las pruebas en la sesión de Google Meet?
15 respuestas

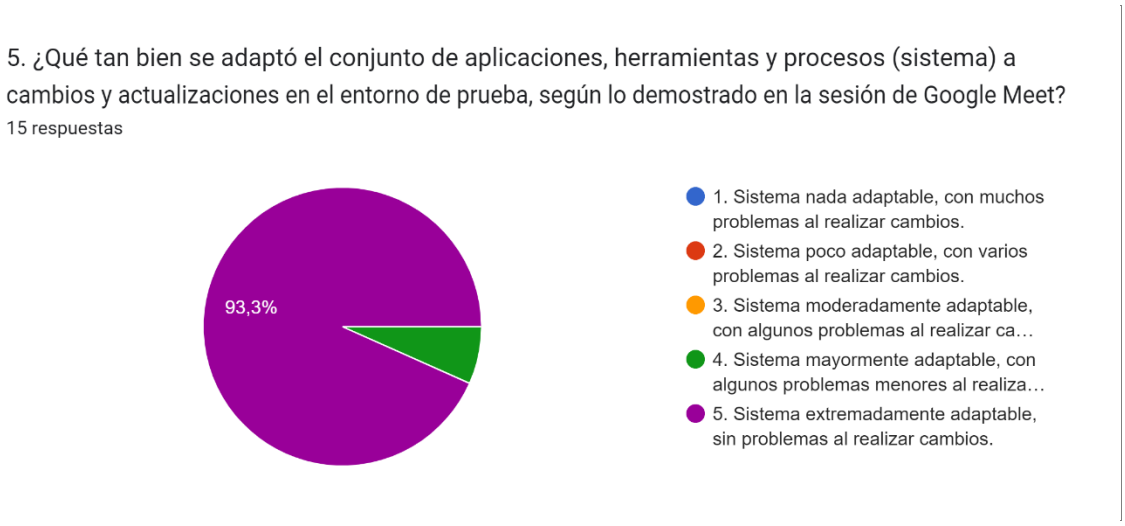


Para la efectividad del proceso de detección y resolución de errores, los resultados muestran que el 93.3% de los encuestados lo evaluó como muy rápido y sin problemas. Este resultado indica que una alta eficiencia en la gestión de errores según lo observado. Las herramientas de monitoreo y las prácticas de gestión de

errores presentadas durante la demostración fueron percibidas como altamente efectivas, permitiendo una identificación y corrección rápida de problemas. Esta rápida resolución de errores asegura un entorno de desarrollo estable y confiable, crucial para mantener la calidad y funcionalidad del sistema.



La satisfacción con el flujo de trabajo general en el entorno de prueba fue extremadamente alta, con el 100% de los encuestados encontrando la experiencia completamente satisfactoria y sin problemas. Este resultado unánime refleja que todos los aspectos del desarrollo, despliegue y mantenimiento, tal como fueron presentados, fueron considerados exitosos y altamente eficientes. La alta satisfacción general indica que el flujo de trabajo propuesto facilitará un entorno de trabajo positivo y productivo, alineado con las mejores prácticas de desarrollo de software.



La adaptabilidad del sistema a cambios y actualizaciones fue calificada positivamente, con el 93.3% de los encuestados considerándolo extremadamente adaptable y sin problemas al realizar cambios. Este resultado sugiere que las aplicaciones, herramientas y procesos demostrados en el entorno de prueba serán altamente flexibles y capaces de manejar modificaciones sin causar interrupciones significativas. La alta calificación en adaptabilidad demuestra que el sistema propuesto es robusto y puede soportar cambios continuos sin afectar negativamente la operatividad, lo cual es esencial para responder a las demandas dinámicas del mercado.

6 Conclusiones

La evaluación detallada del rendimiento y la satisfacción de la aplicación de facturación electrónica, comparando el entorno real del servidor on-premise con el caso de prueba en una máquina virtual, ha revelado una serie de conclusiones importantes que destacan los beneficios y desafíos de la migración a una arquitectura de microservicios.

Mejora en el Rendimiento y la Eficiencia

Una de las conclusiones más significativas es la notable mejora en el rendimiento del sistema. En el entorno de prueba, el tiempo de respuesta de las solicitudes se redujo a la mitad, pasando de 300ms a 150ms. Asimismo, el throughput se duplicó, alcanzando las 100 transacciones por segundo. Estas mejoras no solo optimizan la eficiencia del sistema, sino que también mejoran la experiencia del usuario final, quien puede interactuar con una aplicación más rápida y responsiva.

Uso Óptimo de Recursos

La implementación de microservicios en el caso de prueba también mostró una utilización más eficiente de los recursos del sistema. El uso de CPU se redujo del 80% al 50% durante los picos de demanda, y la utilización de memoria disminuyó del 75% al 40%. Esta optimización de recursos no solo reduce el riesgo de fallos y mejora la estabilidad del sistema, sino que también permite una mejor escalabilidad y manejo de cargas de trabajo variables.

Reducción de Errores y Mejora en la Fiabilidad

La tasa de errores críticos se redujo significativamente en el entorno de prueba, de 10 errores por día a solo 2. Esta mejora en la fiabilidad del sistema permite a los equipos de desarrollo concentrarse en la innovación y la creación de nuevas funcionalidades, en lugar de dedicar tiempo a la resolución de problemas recurrentes. La automatización de los despliegues, con una reducción en la duración

de una hora a 15 minutos, también contribuyó a minimizar los errores humanos y acelerar la entrega de actualizaciones.

Satisfacción del Usuario y de los Desarrolladores

La satisfacción del usuario final aumentó de manera considerable, reflejada en un incremento en las calificaciones de 3 a 4.5 sobre 5. Los usuarios destacaron la mayor velocidad y facilidad de uso de la aplicación. Por otro lado, los desarrolladores y project managers también reportaron una mejora en su flujo de trabajo, gracias a la integración más sencilla y a la eficiencia del proceso de despliegue. La automatización de tareas repetitivas y la mayor adaptabilidad del sistema a cambios y actualizaciones fueron factores clave en esta mejora.

Evaluación del Flujo de Trabajo

El análisis del flujo de trabajo real identificó varias limitaciones importantes. Primero, la gestión de versiones y ramas era deficiente, con falta de un estándar claro de versionamiento y una estrategia de branching que apoyara un ciclo de desarrollo ágil. Para superar esta limitación, el nuevo flujo GitSecOps implementó un estándar claro de versionamiento y una estrategia de branching bien definida, asegurando un ciclo de desarrollo más eficiente y organizado.

Además, los despliegues manuales dependían de la disponibilidad de personal específico y eran propensos a errores humanos, lo que retrasaba la liberación de nuevas características y la resolución de problemas críticos. Con el nuevo flujo, se automatizaron los despliegues, reduciendo la dependencia de personal específico y minimizando la variabilidad y los errores humanos, acelerando así la liberación de nuevas funcionalidades y la corrección de errores.

Otro problema identificado fue la seguridad de las credenciales, ya que no se seguían prácticas adecuadas de almacenamiento seguro, exponiendo al sistema a riesgos significativos. Para mejorar esta situación, se implementaron prácticas robustas de seguridad en el nuevo flujo, incluyendo el almacenamiento seguro de

credenciales, lo que mejoró significativamente la protección del sistema contra brechas de seguridad.

La transición de GitLab a GitHub se realizó sin establecer una herramienta adecuada para la integración y el despliegue continuos, creando un vacío en el flujo de trabajo de desarrollo y afectando negativamente la eficiencia operativa. Esta limitación se superó mediante la adopción de herramientas como Woodpecker y Argo, que facilitaron la integración y el despliegue continuos (CI/CD), asegurando una transición tecnológica fluida y mejorando la eficiencia operativa.

El diseño de la arquitectura de microservicios para los módulos de Contabilidad y Documentos Electrónicos ha permitido la descomposición de los sistemas monolíticos en unidades más pequeñas, autónomas y manejables. Este enfoque no solo facilita una mayor escalabilidad y resiliencia, sino que también potencia la capacidad de adaptación y mantenimiento del sistema. La descomposición en microservicios asegura que cada componente pueda evolucionar de manera independiente, permitiendo una integración más ágil de nuevas funcionalidades y la posibilidad de actualizaciones sin interrupciones significativas.

La validación de la arquitectura propuesta mediante una prueba de concepto a nivel de prototipo, ha sido crucial para demostrar la viabilidad técnica y operativa del diseño. La creación de un prototipo funcional permitió identificar y mitigar riesgos tempranos, asegurando que la arquitectura no solo es teóricamente sólida, sino también práctica y eficaz en un entorno realista. Esta etapa también proporcionó valiosas retroalimentaciones que fueron incorporadas para refinar y optimizar la arquitectura inicial.

La definición y adopción de patrones de diseño adecuados para la gestión de transacciones distribuidas en los módulos es otro logro significativo. La implementación de estos patrones, tales como el Sidecar Pattern, Service Discovery, API Gateway, Bulkhead, Security Patterns, Immutable Infrastructure, Blue-Green Deployment, Canary Releases asegura que la arquitectura es robusta, segura y eficiente. Cada uno de estos patrones ha sido cuidadosamente

seleccionado y aplicado para mejorar la resiliencia, la seguridad y la eficiencia operativa, asegurando que los microservicios puedan manejar de manera efectiva las transacciones distribuidas y mantener la consistencia y la integridad de los datos.

El establecimiento de flujos GitSecOps para la gestión y monitoreo de la arquitectura de microservicios representa una evolución en la manera en que se maneja el ciclo de vida del software. La integración de flujos de trabajo automatizados y optimizados para la integración continua, la entrega continua y la seguridad, ha transformado la manera en que se desarrollan, prueban y despliegan los microservicios. Este enfoque no solo ha mejorado la eficiencia y la rapidez en la entrega de software, sino que también ha reforzado la seguridad en cada etapa del desarrollo, desde la planificación hasta la operación en producción. Herramientas como Gitea, KrakenD, GliptichP, MicroK8, ArgoCD, y Vault han sido fundamentales en la implementación de estos flujos, asegurando una gestión uniforme y segura del código y los despliegues.

La documentación de los diagramas de arquitectura utilizando el estándar C4 ha proporcionado una representación clara y comprensible de la arquitectura del sistema. Este estándar ha permitido comunicar de manera efectiva la estructura y el diseño de la arquitectura a todos los interesados, facilitando una comprensión común y alineada del proyecto. La documentación detallada y estandarizada asegura que cualquier miembro del equipo o parte interesada pueda entender y trabajar con la arquitectura propuesta, promoviendo la colaboración y el mantenimiento a largo plazo.

Finalmente, el contraste entre las métricas del flujo de trabajo actual y el caso de prueba en los integrantes de la empresa mostró una mejora significativa en la percepción de eficiencia y satisfacción con el nuevo flujo propuesto. En el caso de prueba, el 93.3% de los encuestados consideraron que la integración de nuevas funcionalidades fue extremadamente fácil, y el 100% calificaron el proceso de despliegue como extremadamente eficiente, rápido y sin problemas. Además, el 93.3% encontró que la detección y resolución de errores fue muy rápida y sin inconvenientes. Estos resultados destacan la efectividad de las herramientas y

métodos presentados en la sesión virtual. La satisfacción general con el flujo de trabajo en el caso de prueba fue muy alta, con el 100% de los encuestados reportando una experiencia completamente satisfactoria. También se consideró que este entorno era extremadamente adaptable a cambios y actualizaciones por el 93.3% de los encuestados, lo que muestra que las prácticas y herramientas demostradas en el flujo GitSecOps son muy eficaces para mejorar la operatividad y flexibilidad del sistema.

Referencias

Al-Debagy, O., & Martinek, P. (2018). A Comparative Review of Microservices and Monolithic Architectures. *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, 000149-000154. https://doi.org/10.1109/CINTI.2018.8928192

Bashtovyi, A., & Fechan, A. (2023). Change Data capture for migration to event-driven microservices Case Study. *2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT)*, 1-4. https://doi.org/10.1109/CSIT61576.2023.10324262

Basinya, E. A., & Malyshev, E. A. (2023). The Creation and Integration of the Technological Workflow for Software and Hardware-Software Development. *2023 IEEE XVI International Scientific and Technical Conference Actual Problems of Electronic Instrument Engineering (APEIE)*, 960-966. https://doi.org/10.1109/APEIE59731.2023.10347739

Bertoglio, D. D., Schüler, L. G. B., Zorzo, A. F., & Lunardi, R. C. (2022). Understanding the Penetration Test Workflow: A security test with Tramonto in an e-Government application. *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 1597-1604. https://doi.org/10.1109/TrustCom56396.2022.00229

Cataldi, Z. (2000). *Una metodología para el diseño, desarrollo y evaluación de software educativo* [PhD Thesis]. Universidad Nacional de La Plata.

De Almeida, M. G., & Canedo, E. D. (2022). Authentication and Authorization in Microservices Architecture: A Systematic Literature Review. *Applied Sciences*, *12*(6), 3023. https://doi.org/10.3390/app12063023

Delgado Preti, J. P., Araujo Souza, A. N., Freiberger, E. C., & De Almeida Lacerda, T. (2021). Monolithic to Microservices Migration Strategy in Public Safety Secretariat of Mato Grosso. *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 1-5. https://doi.org/10.1109/ICECCE52056.2021.9514268

Díaz, J., López-Fernández, D., Pérez, J., & González-Prieto, Á. (2021). Why are many businesses instilling a DevOps culture into their organization? *Empirical Software Engineering*, *26*(2), 25. https://doi.org/10.1007/s10664-020-09919-3

Faria, V., & Silva, A. R. (2022). *Code Vectorization and Sequence of Accesses Strategies for Monolith Microservices Identification*. arXiv. https://doi.org/10.48550/arXiv.2212.11657

Farsi, H., Allaki, D., En-nouaary, A., & Dahchour, M. (2023). Dealing with Anti-Patterns When Migrating from Monoliths to Microservices: Challenges and Research Directions. *2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, 1-8. https://doi.org/10.1109/CloudTech58737.2023.10366131

Ferreira, L. C. B. C., Borchardt, A. D. R., Cardoso, G. D. S., Mendes Lemes, D. A., Sousa, G. R. D. R. D., Neto, F. B., De Lima, E. R., Fraidenraich, G., Cardieri, P., & Meloni, L. G. P. (2022). Edge Computing and Microservices Middleware for Home Energy Management Systems. *IEEE Access*, *10*, 109663-109676. https://doi.org/10.1109/ACCESS.2022.3214229

Fritzsch, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019). Microservices Migration in Industry: Intentions, Strategies, and Challenges. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 481-490. https://doi.org/10.1109/ICSME.2019.00081

Ghosh, A., Mukherjee, A., & Misra, S. (2022). SEGA: Secured Edge Gateway Microservices Architecture for IIoT-Based Machine Monitoring. *IEEE Transactions on Industrial Informatics*, *18*(3), 1949-1956. https://doi.org/10.1109/TII.2021.3102158

Golis, T., Dakić, P., & Vranić, V. (2022). Creating Microservices and using infrastructure as code within the CI/CD for dynamic container creation. *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)*, 114-119. https://doi.org/10.1109/Informatics57926.2022.10083442

Guaman, D., Yaguachi, Lady, Samanta, C. C., Danilo, J. H., & Soto, F. (2018). Performance evaluation in the migration process from a monolithic application to microservices. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 1-8. https://doi.org/10.23919/CISTI.2018.8399148

Khan, M. S., Khan, A. W., Khan, F., Khan, M. A., & Whangbo, T. K. (2022). Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review. *IEEE Access*, *10*, 14339-14349. https://doi.org/10.1109/ACCESS.2022.3145970

Kyryk, M., Tymchenko, O., Pleskanka, N., & Pleskanka, M. (2022). Methods and process of service migration from monolithic architecture to microservices. *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 553-558. https://doi.org/10.1109/TCSET55632.2022.9767055

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. *2017 IEEE International Conference on Web Services (ICWS)*, 524-531. https://doi.org/10.1109/ICWS.2017.61

Nikolaevich, A. A. (2023). Leveraging Microservices Architecture for Sports Competition Management on AWS with Terraform and CI/CD. *International Journal of Latest Engineering and Management Research (IJLEMR)*, *8*(5), 45-55. https://doi.org/10.56581/IJLEMR.8.5.45-55

Perm State University, Radostev, D., Nikitina, E., & Perm State University. (2021). Software code migration strategy from monolithic architecture to microservices. *Вестник Пермского университета. Математика. Механика. Информатика*, *2(53)*, 65-68. https://doi.org/10.17072/1993-0550-2021-2-65-68

Raharjo, A. B., Andyartha, P. K., Wijaya, W. H., Purwananto, Y., Purwitasari, D., & Juniarta, N. (2022). Reliability Evaluation of Microservices and Monolithic Architectures. *2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 1-7. https://doi.org/10.1109/CENIM56801.2022.10037281

Raj, V., & Sadam, R. (2021). Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics. *SN Computer Science*, *2*(5), 374. https://doi.org/10.1007/s42979-021-00767-6

Silva Filho, H. C. da, & Figueiredo Carneiro, G. de. (2019). Strategies Reported in the Literature to Migrate to Microservices Based Architecture. En S. Latifi (Ed.), *16th International Conference on Information Technology-New Generations*

(ITNG 2019) (pp. 575-580). Springer International Publishing. https://doi.org/10.1007/978-3-030-14070-0_81

Singh, C., Gaba, N. S., Kaur, M., & Kaur, B. (2019). Comparison of Different CI/CD Tools Integrated with Cloud Platform. *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 7-12. https://doi.org/10.1109/CONFLUENCE.2019.8776985

Siti Rochimah & Bintang Nuralamsyah. (2023). Decomposing Monolithic to Microservices: Keyword Extraction and BFS Combination Method to Cluster Monolithic's Classes. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, *7*(2), 263-270. https://doi.org/10.29207/resti.v7i2.4866

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A., & Lang, M. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *Service Oriented Computing and Applications*, *11*(2), 233-247. https://doi.org/10.1007/s11761-017-0208-y

Wang, L., Jiang, Y. X., Wang, Z., Huo, Q. E., Dai, J., Xie, S. L., Li, R., Feng, M. T., Xu, Y. S., & Jiang, Z. P. (2023). The operation and maintenance governance of microservices architecture systems: A systematic literature review. *Journal of Software: Evolution and Process*, *35*(10), e2433. https://doi.org/10.1002/smr.2433

Evaluación del flujo de trabajo actual

Estamos llevando a cabo una evaluación detallada del flujo de trabajo actual en Manticore Labs con el objetivo de identificar áreas de mejora y optimizar nuestros procesos operativos. Su valiosa retroalimentación nos permitirá comprender mejor los desafíos que enfrentan diariamente y las oportunidades para mejorar nuestra eficiencia y productividad.

Agradecemos su participación en esta encuesta. Sus respuestas serán confidenciales y se utilizarán exclusivamente con fines de análisis para mejorar nuestro entorno de trabajo.

1. ¿Qué tan fácil le resulta integrar nuevas funcionalidades en el flujo de trabajo actual, considerando la colaboración entre equipos y la implementación en los sistemas existentes?

1. Integración extremadamente difícil, con muchos problemas
2. Integración difícil, con varios problemas
3. Integración moderadamente fácil, con algunos problemas notables
4. Integración mayormente fácil, con algunos problemas menores
5. Integración extremadamente fácil, sin problemas

2. ¿Qué tan eficiente considera el proceso de despliegue en el flujo de trabajo actual, en términos de tiempo requerido y la cantidad de pasos necesarios?

1. Despliegue extremadamente ineficiente, muy lento y con muchos problemas
2. Despliegue ineficiente, con varios retrasos y problemas significativos
3. Despliegue moderadamente eficiente, con algunos retrasos y problemas
4. Despliegue mayormente eficiente, rápido y con algunos problemas menores
5. Despliegue extremadamente eficiente, muy rápido y sin problemas

3. ¿Qué tan efectivo es el proceso de detección y resolución de errores en el flujo de trabajo actual, considerando la rapidez y precisión con la que se identifican y corrigen los problemas?

1. Errores no se detectan o resuelven de manera oportuna, con muchos problemas
2. Errores detectados y resueltos lentamente, con varios problemas
3. Errores detectados y resueltos de manera moderada, con algunos retrasos
4. Errores detectados y resueltos rápidamente, con algunos problemas menores
5. Errores detectados y resueltos muy rápidamente, sin problemas

4. ¿Qué tan satisfecho está con el flujo de trabajo general en el entorno actual, considerando todos los aspectos del desarrollo, despliegue y mantenimiento?

1. Flujo de trabajo nada satisfactorio, con muchos problemas
2. Flujo de trabajo poco satisfactorio, con varios problemas
3. Flujo de trabajo moderadamente satisfactorio, con algunos problemas
4. Flujo de trabajo mayormente satisfactorio, con algunos problemas menores
5. Flujo de trabajo extremadamente satisfactorio, sin problemas

5. ¿Qué tan bien se adapta el conjunto de aplicaciones, herramientas y procesos (sistema) a cambios y actualizaciones en el flujo de trabajo actual, considerando la capacidad del sistema para aceptar y procesar cambios sin causar interrupciones o problemas significativos?

1. Sistema nada adaptable, con muchos problemas al realizar cambios
2. Sistema poco adaptable, con varios problemas al realizar cambios
3. Sistema moderadamente adaptable, con algunos problemas al realizar cambios
4. Sistema mayormente adaptable, con algunos problemas menores al realizar cambios
5. Sistema extremadamente adaptable, sin problemas al realizar cambios

ANEXO 2

Evaluación del flujo GitSecOps del Caso de Prueba

Como parte de nuestro esfuerzo continuo por mejorar nuestros procesos y la arquitectura de nuestra aplicación de facturación electrónica, estamos llevando a cabo una evaluación del caso de prueba propuesto en la sesión de Google Meet.

Se realizaron diversas pruebas tanto en situaciones de error como en condiciones normales. Posteriormente, estamos aplicando esta encuesta para recopilar sus opiniones y experiencias con respecto al caso de prueba.

Su retroalimentación es esencial para entender mejor las mejoras observadas y cualquier desafío que pueda haber surgido durante esta prueba.

Agradecemos su participación en esta encuesta. Sus respuestas serán confidenciales y se utilizarán exclusivamente con fines de análisis para mejorar nuestro entorno de trabajo.

1. ¿Qué tan fácil le resultó integrar nuevas funcionalidades en el entorno de prueba, considerando la colaboración entre equipos y la implementación en el sistema propuesto durante la sesión de Google Meet?

1. Integración extremadamente difícil, con muchos problemas.
2. Integración difícil, con varios problemas.
3. Integración moderadamente fácil, con algunos problemas notables.
4. Integración mayormente fácil, con algunos problemas menores.
5. Integración extremadamente fácil, sin problemas.

2. ¿Qué tan eficiente consideró el proceso de despliegue en el entorno de prueba, en términos de tiempo requerido y la cantidad de pasos necesarios, según lo demostrado en la sesión de Google Meet?

1. Despliegue extremadamente ineficiente, muy lento y con muchos problemas.
2. Despliegue ineficiente, con varios retrasos y problemas significativos.
3. Despliegue moderadamente eficiente, con algunos retrasos y problemas.
4. Despliegue mayormente eficiente, rápido y con algunos problemas menores.
5. Despliegue extremadamente eficiente, muy rápido y sin problemas.

3. ¿Qué tan efectivo fue el proceso de detección y resolución de errores en el entorno de prueba, considerando la rapidez y precisión con la que se identificaron y corrigieron los problemas durante las pruebas en la sesión de Google Meet?

1. Errores no se detectan o resuelven de manera oportuna, con muchos problemas.
2. Errores detectados y resueltos lentamente, con varios problemas.
3. Errores detectados y resueltos de manera moderada, con algunos retrasos.
4. Errores detectados y resueltos rápidamente, con algunos problemas menores.
5. Errores detectados y resueltos muy rápidamente, sin problemas.

4. ¿Qué tan satisfactoria encontró la experiencia del flujo de trabajo general en el entorno de prueba, considerando todos los aspectos del desarrollo, despliegue y mantenimiento presentados durante la sesión de Google Meet?

1. Nada satisfactorio, con muchos problemas.
2. Poco satisfactorio, con varios problemas.
3. Moderadamente satisfactorio, con algunos problemas.
4. Mayormente satisfactorio, con algunos problemas menores.
5. Extremadamente satisfactorio, sin problemas.

5. ¿Qué tan bien se adaptó el conjunto de aplicaciones, herramientas y procesos (sistema) a cambios y actualizaciones en el entorno de prueba, según lo demostrado en la sesión de Google Meet?

1. Sistema nada adaptable, con muchos problemas al realizar cambios.
2. Sistema poco adaptable, con varios problemas al realizar cambios.
3. Sistema moderadamente adaptable, con algunos problemas al realizar cambios.
4. Sistema mayormente adaptable, con algunos problemas menores al realizar cambios.
5. Sistema extremadamente adaptable, sin problemas al realizar cambios.