Optimización de la infraestructura on-premise de la Facultad de Ingeniería para escalabilidad y la entrega continua del Software con prácticas DevOps

Andrés Felipe Vásquez Ramírez

Informe de práctica presentado para optar al título de Ingeniero de Sistemas

Asesor
José Ignacio López Pérez, Ingeniero de Sistemas

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Antioquia, Colombia
2023

Cita	(Vásquez Ramírez, A. 2023)
Referencia **Estilo APA 7 (2020)**	Vásquez Ramírez, A. (2023). *Optimización de la infraestructura on-premise de la Facultad de Ingeniería para escalabilidad y la entrega continua del Software con prácticas DevOps* [Informe de práctica]. Universidad de Antioquia, Medellín, Colombia.

Dedicatoria

A mí, un recordatorio de mi valentía y perseverancia en la búsqueda de mis sueños. A lo largo de este viaje académico, siempre he soñado con estar en esta alma máter, y hoy celebro el logro de hacerlo realidad. Además, quiero expresar mi más profundo agradecimiento a esa persona especial que estuvo a mi lado al final del camino.

Agradecimientos

Agradezco a mis asesores, José Ignacio López Pérez y Clara Lucia Monsalve, por su inestimable apoyo y orientación; al equipo del DRAI y su jefe Juan Diego Vélez Serna por brindar un ambiente enriquecedor y la autonomía durante el desarrollo de este proyecto; a la Universidad de Antioquia por darme la oportunidad de estar en el Alma Mater y forjarme como profesional. Y finalmente, a todos quienes de alguna manera contribuyeron a este logro.

Tabla de Contenido

Lista de tablas

Lista de Figuras

Siglas, acrónimos y abreviaturas

CI/CD	Continuous Integrations Continuous Delivery
DevOps	Development Operations
DORA	DevOps Research and Assessment
DRAI	Departamento de Recursos de Apoyo e Informática
K8s	Kubernetes
I&O	Infraestructura y Operaciones
SDLC	Systems Development Life Cycle)
SRE	Site Reliability Engineering
TI	Tecnología de la Información
VM	Virtual Machine (Máquina Virtual)

Resumen

Este proyecto tuvo como objetivo general implementar prácticas DevOps para optimizar el datacenter de la Facultad de Ingeniería, buscando mejorar la calidad del software, minimizar los tiempos de despliegue de software y maximizar la productividad operativa. Se utilizó la metodología Kanban ya que puede ser adaptada a las limitaciones de tiempo y recursos, pasando por etapas de análisis, selección de herramientas, diseño de estrategia, implementación y evaluación. Como resultado, se logra una infraestructura más robusta y eficiente que transformó la forma en que se desarrolla y despliega el software en la facultad. Esto se logró a través de la implementación de tecnologías como Kubernetes y la automatización de procesos mediante CI/CD, lo que ha mejorado la eficiencia y confiabilidad de los servicios ofrecidos. El proyecto concluyó con éxito, dejando una serie de recomendaciones y mejores prácticas para optimizar aún más la implementación de la practicas DevOps y la automatización de la infraestructura en el futuro. En resumen, este proyecto proporcionó una base sólida para futuras innovaciones y mejoras en el desarrollo de software y la gestión de la infraestructura en la Facultad de Ingeniería.

Palabras clave: DevOps, Kubernetes, On-Premise, Infraestructura, CI/CD

Abstract

The overarching objective of this study was to implement DevOps practices aimed at optimizing the datacenter of the Faculty of Engineering. This effort sought to enhance software quality, minimize software deployment times, and maximize operational productivity. We employed the Kanban methodology to adapt to resource and time constraints, transitioning through stages of analysis, tool selection, strategy design, implementation, and evaluation. As a result, we achieved a more robust and efficient infrastructure that transformed the way software is developed and deployed within the faculty. This was accomplished through the implementation of modern technologies such as Kubernetes and the automation of processes via CI/CD, enhancing the efficiency and reliability of the services provided. Our project successfully concluded, leaving a series of recommendations and best practices to further optimize the implementation of the DevOps culture and infrastructure automation in the future. In summary, this project laid a solid foundation for future innovations and improvements in software development and infrastructure management within the Faculty of Engineering.

*Keywords***:** DevOps, Kubernetes, On-Premise, CI/CD

Introducción

En un mundo cada vez más digitalizado, la optimización de las infraestructuras de los centros de datos es una necesidad imperante. La Facultad de Ingeniería de la Universidad de Antioquia, enfrentada a desafíos en su datacenter, sobre todo en el desarrollo y despliegue de software y en la gestión de su infraestructura, se encuentra en un punto de inflexión. Este estudio tiene como objetivo implementar prácticas de DevOps para mejorar la eficiencia operativa y la calidad del software en este contexto.

El alcance de este estudio es amplio y abarca la evaluación de la situación actual de la facultad, el desarrollo y fomento de prácticas DevOps, con la selección e implementación de las herramientas DevOps adecuadas, y la evaluación del impacto de dicha implementación. Además, se generarán recomendaciones y mejores prácticas basadas en los resultados obtenidos. Este estudio está sujeto a algunas limitaciones, entre las que se incluyen posibles restricciones de tiempo y recursos, así como la resistencia al cambio que pueda surgir durante el proceso de implementación.

Para abordar estos desafíos, se aplica la metodología Kanban, reconocida por su adaptabilidad y su enfoque en la entrega continua de valor. Esta metodología flexible consta de varias etapas, que incluyen análisis e investigación, selección de herramientas, diseño de estrategia, implementación, evaluación y propuestas de mejora.

La implementación exitosa de una cultura DevOps en la Facultad de Ingeniería tiene el potencial de ser un catalizador en el avance del campo de la gestión de la infraestructura de TI y el desarrollo de software. Esto no solo mejorará la eficiencia operativa y la calidad del software, sino que también proporcionará un modelo para otras instituciones con desafíos similares.

En resumen, este estudio ofrece una visión práctica y efectiva para optimizar la infraestructura de TI y las prácticas de desarrollo de software en la Facultad de Ingeniería. La adopción de una cultura DevOps y la implementación de la automatización de la infraestructura tienen un significado relevante para el avance del campo respectivo y su aplicación tendrá un impacto positivo en la eficiencia y efectividad de los servicios dependientes del datacenter de la facultad.

Planteamiento del problema

La creciente demanda de software en la era digital ha impulsado la necesidad de acelerar la entrega de aplicaciones al mercado. En este contexto, la implementación de la entrega continua del software (*Continuous Delivery*, CD) se ha convertido en una estrategia clave para mejorar el *time to market*, es decir, el tiempo que transcurre desde el inicio del desarrollo hasta la disponibilidad del producto en el mercado. Sin embargo, existen diversas problemáticas asociadas a la implementación efectiva de la entrega continua del software, las cuales deben ser abordadas para lograr los beneficios deseados.

En el contexto actual de la industria TI (Tecnologías de Información), la eficiencia y la optimización de los recursos son fundamentales para el rendimiento y la productividad (Sun et al., 2016). Una de las principales áreas de preocupación en esta esfera es la subutilización de los servidores físicos, lo que resulta en una falta de aprovechamiento de las tecnologías disponibles como las máquinas virtuales (VM) y los contenedores, para potenciar al máximo las características de nuestros servidores. Estas tecnologías, que podrían amplificar notablemente las capacidades de los servidores, no están siendo explotadas de manera óptima (N. Vhatkar & Bhole, 2022).

Es crucial entender dos variables claves en este contexto. La primera, "uso de tecnologías como máquinas virtuales (VM) y contenedores", se refiere a la implementación y el uso de estas tecnologías en una organización de TI. Las VM, por un lado, son sistemas operativos que se ejecutan en hardware compartido, permitiendo así la virtualización de recursos y la posibilidad de ejecutar múltiples sistemas operativos en un solo servidor físico (Goldberg, 1974). Los contenedores, por otro lado, son una forma de virtualización a nivel de sistema operativo que permite ejecutar aplicaciones y sus dependencias en procesos aislados (Docker Inc., 2023).

La segunda variable, "aprovechamiento de las características de los servidores físicos", considera el grado en que una organización utiliza eficientemente los recursos disponibles en sus servidores físicos, incluyendo la CPU, la memoria, el almacenamiento y la red.

Según Gartner (2013), más del 70% de los servidores físicos están subutilizados, un problema alarmante que conlleva a costos operativos innecesarios, un consumo de energía excesivo y una reducción de la eficiencia operativa. A pesar de este escenario desalentador, estudios recientes han destacado el potencial de las VM y los contenedores para mejorar

significativamente la utilización de los servidores. (Scheepers, 2014) encontró que la utilización de los servidores puede incrementarse hasta un 60-80% con el uso de VM y hasta un 80-90% con el uso de contenedores. A pesar de estos hallazgos prometedores, muchos servidores físicos siguen estando subutilizados. La resistencia al cambio, las preocupaciones sobre la seguridad y la fiabilidad de estas tecnologías, así como la falta de habilidades técnicas y conocimientos para implementar y gestionar estas tecnologías podrían estar contribuyendo a este problema (Brynjolfsson & Kahin, 2000).

En el ámbito de las empresas de tecnología de la información (TI), se reconoce que la optimización de los recursos es fundamental para mejorar la eficiencia y la productividad. Sin embargo, para lograr una entrega continua del software aún es necesario adoptar una filosofía *DevOps*, que combina las prácticas de desarrollo de software (Dev) y las operaciones de tecnología de la información (Ops), con el objetivo de mejorar la calidad y la eficiencia del proceso de desarrollo (Bass et al., 2015). Esta sinergia entre el desarrollo y las operaciones puede resultar en una reducción significativa de los tiempos de despliegue y una mejora en la calidad del software.

La "ausencia de la practicas DevOps" se define como la falta de integración y colaboración entre los equipos de desarrollo y operaciones. Esta situación puede resultar en silos de trabajo, falta de comunicación, conflictos y una falta de entendimiento compartido de los objetivos y metas del proyecto (Lwakatare et al., 2015). Una variedad de factores puede contribuir a esta ausencia, incluyendo la falta de formación adecuada, la resistencia al cambio organizacional y la falta de herramientas y tecnologías adecuadas para facilitar la colaboración y la integración.

Uno de los aspectos más afectados por la ausencia de la cultura *DevOps* es la calidad del software y los tiempos de despliegue. Este término se refiere a la capacidad de una organización para desplegar software de alta calidad a un ritmo constante y eficiente (Kim et al., 2021). La calidad del software se mide en términos de su funcionalidad, rendimiento, seguridad y facilidad de uso, mientras que la frecuencia de despliegue se refiere a la rapidez con la que una organización puede desplegar nuevas versiones o actualizaciones del software.

En cuanto a las cifras y estadísticas sobre el tema, según los datos más recientes disponibles, el informe "State of DevOps" de Puppet Labs Puppet y DORA (Puppet, 2021) encontró que las organizaciones que adoptan una cultura DevOps pueden desplegar código 30

veces más a menudo que aquellas que no lo hacen, con un 50% menos de fallos en sus nuevos lanzamientos.

El papel de un ingeniero de Confiabilidad del Sitio (SRE, por sus siglas en inglés) es esencial en cualquier organización que desarrolle software. La metodología SRE, que fue desarrollada por Google en 2003 (Beyer et al., 2016), se centra en la confiabilidad y la estabilidad de las operaciones del sistema, y se complementa bien con la cultura *DevOps*, proporcionando métricas y objetivos claros para mejorar la confiabilidad del sistema y la eficiencia operativa.

Es imperativo que las organizaciones de desarrollo de software comprendan los beneficios de la cultura DevOps y trabajen para superar los desafíos asociados con su implementación. Esta comprensión puede conducir a un mayor éxito en la entrega de software de alta calidad a un ritmo más rápido y eficiente, mejorando la satisfacción del cliente y la competitividad en el mercado.

La implementación de la metodología SRE en combinación con *DevOps* puede mejorar la calidad del software y la eficiencia del despliegue. Sin embargo, existe una brecha de conocimiento significativa en la literatura existente sobre cómo estas dos filosofías pueden integrarse de manera efectiva en las organizaciones de desarrollo de software.

Los ingenieros SRE se enfocan en la creación de sistemas resilientes y autorreparables, lo que puede mejorar la confiabilidad del software y reducir la cantidad de tiempo que los equipos de desarrollo y operaciones dedican a la solución de problemas y la resolución de incidentes (Beyer et al., 2016).

La Facultad de Ingeniería enfrenta desafíos significativos en tres áreas clave: la adopción de prácticas *DevOps,* los tiempos de despliegue de las aplicaciones y la subutilización de los servidores físicos on-premise.

La combinación de estas problemáticas pone en evidencia la necesidad de adoptar una filosofía de desarrollo más moderna, como *DevOps*, y de optimizar el uso de los servidores on-premise a través de tecnologías de virtualización. Sólo así, se podrá mejorar la eficiencia, reducir los tiempos de despliegue de las aplicaciones y garantizar una utilización óptima de los recursos.

El objetivo principal será investigar e implementar estrategias que permitan la adopción efectiva de las prácticas *DevOps* en la Facultad de Ingeniería, con el fin de optimizar los tiempos de despliegue y maximizar la utilización de los servidores físicos on-premise. A través de esta investigación, se espera proporcionar una base sólida para la integración de las filosofías SRE y *DevOps* en el área de I&O.

Objetivos

Objetivo general

Implementar practicas *DevOps* enfocadas en la optimización del *datacenter* de la Facultad de Ingeniería, buscando la calidad del software, minimizando los tiempos de despliegue de software y maximizando la productividad operativa.

Objetivos específicos

- Analizar la situación actual de la Facultad de Ingeniería en términos de desarrollo de software y gestión de la infraestructura, identificando las áreas de mejora y los desafíos existentes.
- Seleccionar herramientas y técnicas adecuadas para la automatización de la infraestructura en la Facultad de Ingeniería, incluyendo el aprovisionamiento de servidores, la configuración de redes y el despliegue de servicios.
- Implementar practicas *DevOps* en la Facultad de Ingeniería, fomentando la colaboración entre los equipos de desarrollo y operaciones y promoviendo la responsabilidad compartida en el ciclo de vida del software.
- Evaluar el impacto de la implementación de *DevOps* y la automatización de la infraestructura en la eficiencia y confiabilidad de los servicios en la Facultad de Ingeniería, considerando indicadores clave como el tiempo de entrega, la calidad del software y la disponibilidad de los servicios.
- Proponer recomendaciones y mejores prácticas basadas en los resultados obtenidos, para optimizar la implementación de una cultura *DevOps* y la automatización de la infraestructura en la Facultad de Ingeniería, con el objetivo de impulsar mejoras continuas en el desarrollo y gestión de los servicios.

Alcance

Evaluación de la Situación Actual: Se llevará a cabo una evaluación exhaustiva de la situación actual de la Facultad de Ingeniería en términos de desarrollo de software e infraestructura. El objetivo principal será identificar áreas de mejora y desafíos existentes. Además, se revisarán las técnicas de desarrollo y despliegue de software actualmente en uso, así como la gestión de servidores físicos y redes.

Desarrollo de la Cultura DevOps: Se promoverá activamente el desarrollo y la implementación de una cultura DevOps en la Facultad de Ingeniería. Esto implicará fomentar una mayor colaboración entre los equipos de desarrollo y operaciones, así como inculcar un sentido de responsabilidad compartida en todas las fases del ciclo de vida del software. Además, se organizarán talleres de formación y sesiones de coaching para los miembros del equipo, con el fin de garantizar su familiaridad y competencia en las prácticas y principios de DevOps.

Selección e Implementación de Herramientas DevOps: Se identificarán y seleccionarán las herramientas y técnicas más adecuadas para la automatización de la infraestructura en la Facultad de Ingeniería. Esto incluirá soluciones para el aprovisionamiento de servidores, la configuración de redes y el despliegue de servicios. Una vez seleccionadas, se supervisará su implementación y se garantizará una integración efectiva con los flujos de trabajo existentes.

Evaluación del Impacto de DevOps: Se realizará una evaluación exhaustiva del impacto de la implementación de DevOps y la automatización de la infraestructura en la eficiencia y confiabilidad de los servicios en la Facultad de Ingeniería. Se considerarán indicadores clave, como el tiempo de entrega, la calidad del software y la disponibilidad de los servicios. Los hallazgos de esta evaluación se documentarán en un informe detallado.

Propuestas de Mejora: Con base en los resultados de la evaluación, se formularán recomendaciones y mejores prácticas para optimizar la implementación de una cultura DevOps y la automatización de la infraestructura en la Facultad de Ingeniería. Estas propuestas serán diseñadas para impulsar mejoras continuas en el desarrollo y la gestión de los servicios, y ayudar así a la Facultad a alcanzar su objetivo de maximizar la productividad operativa.

El alcance del proyecto excluye la adquisición de hardware adicional, el plazo estimado para la finalización del proyecto es de seis meses, comenzando el 1 de septiembre del 2022 y finalizando el 30 de abril de 2023.

Marco teórico

Este marco teórico se propone como una exploración integral de los diversos elementos y tecnologías que conforman el ecosistema actual de desarrollo y administración de software. Aquí, se busca trazar un recorrido que va desde el nivel más básico del sistema, con el estudio de los sistemas operativos Linux, hasta las prácticas y filosofías más sofisticadas y recientes en el campo del desarrollo de software y la administración de sistemas.

La primera sección del marco se enfoca en el Administrador de Servidores Linux o el administrador de TI, en la siguiente sección, se aborda el Ciclo de vida del Software. Este análisis recorre las fases del ciclo de desarrollo de software tradicional, antes de sumergirse en las metodologías ágiles que están revolucionando el mundo del desarrollo de software, proporcionando un nuevo enfoque para adaptarse a las cambiantes necesidades del mercado y mejorar la eficiencia del proceso de desarrollo; continua con la profundización en las Prácticas *DevOps*, un conjunto de prácticas que busca armonizar el desarrollo y las operaciones de TI para mejorar la eficiencia, la calidad del software y la velocidad de entrega y finalmente, el marco concluye con el estudio de la Ingeniería de Confiabilidad del Sitio (SRE), una disciplina que se centra en garantizar la confiabilidad y la estabilidad de las operaciones del sistema, proporcionando una valiosa perspectiva para mantener la alta disponibilidad y el rendimiento en entornos de producción complejos.

Administración de Servidores Linux

El mundo de la informática se encuentra en constante evolución, los *datacenters* juegan un papel crucial en la infraestructura de TI de las organizaciones y uno de los elementos que han surgido en los últimos años ha sido el movimiento DevOps y SRE. Este enfoque busca integrar las tareas de desarrollo y operaciones de los sistemas informáticos para mejorar la eficiencia y la productividad. Un elemento crucial en este contexto es la administración y gestión de servidores Linux, que son un componente esencial de muchas infraestructuras tecnológicas (Kim et al., 2021).

Los servidores Linux juegan un papel fundamental en este contexto. Linux es un sistema operativo (OS) de código abierto que se utiliza comúnmente en entornos de servidores debido a su estabilidad, seguridad y flexibilidad (Torvalds & Diamond, 2002). La administración de

servidores Linux implica la gestión de estas máquinas, lo que incluye tareas como la instalación y configuración de software, la monitorización del rendimiento del sistema, la gestión de usuarios y permisos, y la solución de problemas (Negus, 2020).

Se estima que más del 70% de los servidores web del mundo funcionan con Linux, y este número ha estado creciendo constantemente durante los últimos años (W3techs.com, 2023). A pesar de esto, se ha observado que muchos equipos de operaciones y administración de servidores no están aprovechando completamente las ventajas que ofrece Linux, en parte debido a la falta de habilidades y conocimientos adecuados (The Linux Foundation, 2022).

Sistemas Operativos (OS)

Un sistema operativo (OS) es un programa que gestiona el hardware y el software de una computadora y proporciona servicios a otros programas de software (Tanenbaum & Bos, 2015). Los sistemas operativos son fundamentales para el funcionamiento de los sistemas informáticos y desempeñan diversas funciones claves, como la gestión de la memoria, el control de dispositivos, la planificación de tareas y la gestión de archivos.

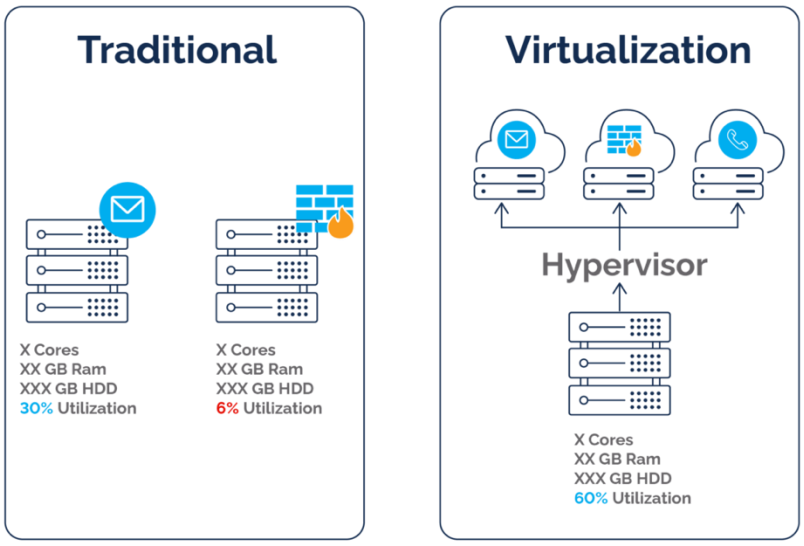
En el ámbito de los *datacenters*, existe un tipo de sistema operativo especializado, este tipo de sistema operativo fue específicamente diseñado para administrar y coordinar recursos a gran escala en un datacenter, abarcando aspectos como la red, la computación y el almacenamiento (Vahldiek-Oberwagner et al., 2018). Los OS para servidores se diferencian de los tradicionales en el sentido de que están concebidos para gestionar simultáneamente decenas de miles de nodos, en lugar de enfocarse en la administración de una única máquina, entre los más utilizados para la gestión de servidores físicos son RHEL (Red Hat Enterprise Linux), SUSE, Windows Server. Estos últimos requieren una licencia comercial, debido a que incluyen un hipervisor, que permite maximizar el aprovechamiento de los recursos y facilita la gestión de las Máquinas Virtuales (VM).

Hipervisor

Un hipervisor, también conocido como monitor de máquina virtual (VM), es un software de sistema que permite la virtualización de hardware. Su función principal es la de crear, ejecutar y administrar múltiples máquinas virtuales independientes en un solo host físico (Popek & Goldberg, 1974). La virtualización ha transformado la forma en que las organizaciones

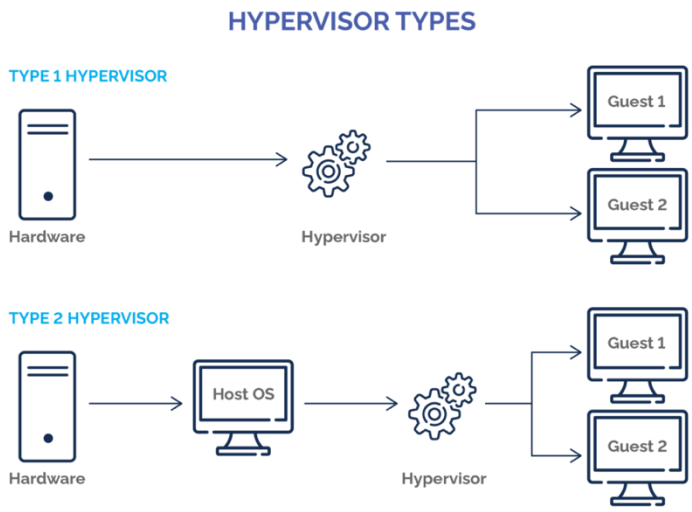
implementan y administran sus recursos de TI, aumentando significativamente la eficiencia operativa y la agilidad del negocio.

Figura 1
Servidor tradicional vs virtualizado



Los hipervisores se clasifican generalmente en dos tipos: Tipo 1 y Tipo 2 (Kivity et al., s. f.). En la figura 2, se puede observar los dos tipos. Los hipervisores de tipo I, también conocidos como hipervisores "nativos" o "bare metal", se ejecutan directamente en el hardware del host. Estos son altamente eficientes ya que tienen acceso directo al hardware subyacente. Ejemplos de este tipo de hipervisores son VMware ESXi, Microsoft Hyper-V y Xen. Por otro lado, los hipervisores de tipo II o "alojados" se ejecutan en un sistema operativo convencional, al igual que cualquier otra aplicación de software. Aunque los hipervisores de tipo 2 tienen una sobrecarga adicional debido a la capa adicional del sistema operativo, ofrecen una mayor flexibilidad y son más fáciles de instalar y administrar. Ejemplos de este tipo son VMware Workstation, KVM, Hyper-v y Oracle VirtualBox.

Figura 2
Tipos de Hipervisores



Un informe de (Grand View Research, 2023) estima que el mercado global de hipervisores alcanzará los $8.9 mil millones de dólares para 2025, lo que demuestra la creciente importancia de la virtualización en los entornos empresariales actuales.

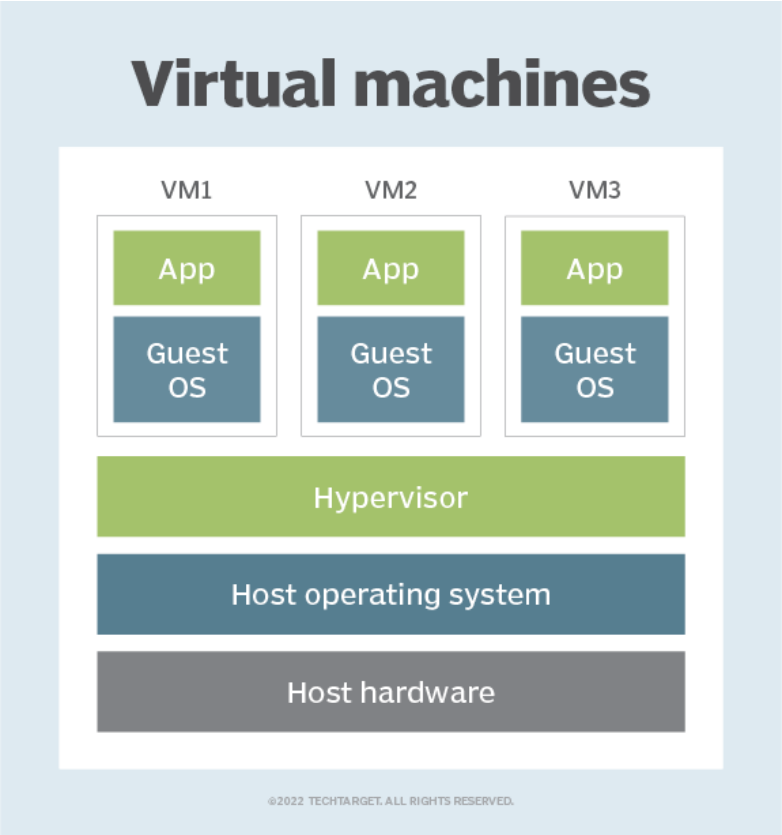
La implementación de un hipervisor puede aportar varios beneficios a una organización. Estos incluyen una mayor utilización de los recursos, ya que múltiples máquinas virtuales pueden compartir un solo host físico, una mayor flexibilidad, dado que las VM pueden ser fácilmente creadas, destruidas, migradas o redimensionadas según sea necesario; y una mayor resistencia, debido a que los fallos pueden estar aislados a una sola VM en lugar de afectar a todo el sistema físico (VMWare, 2023).

Máquinas Virtuales

Las máquinas virtuales (VM) representan una faceta crucial de la infraestructura moderna de TI, facilitando la eficiencia operativa y la agilidad organizacional. Una máquina virtual es una emulación de una computadora que se ejecuta en un hipervisor, imitando una computadora física completa desde el sistema operativo hasta el hardware (Smith & Nair, 2005). Esta emulación permite a los administradores ejecutar múltiples sistemas operativos y aplicaciones en un solo servidor físico, promoviendo la consolidación de servidores y la rápida implementación de nuevos sistemas.

Figura 3

Arquitectura de una Máquina Virtual



Como se observa en la figura 3, las máquinas virtuales están virtualizadas sobre un hipervisor en este caso de clase II, al tener un Guest OS se puede hacer la configuración de todos los elementos de hardware de la VM.

El uso de máquinas virtuales tiene numerosos beneficios. Entre ellos se incluyen la consolidación de servidores, que permite a las organizaciones hacer un uso más eficiente de sus recursos físicos; el aislamiento, que evita que las fallas en una VM afecten a otras; la portabilidad, que permite a las VMs ser trasladadas entre hosts físicos con mínimas interrupciones; y la instalación de diferentes OS de acuerdo a la necesidad indiferente del OS servidor Host (VMWare, 2023). Aunque esto conlleva, como gran desventaja, el consumo de los recursos de forma no óptima, al disponer de instalaciones completas de OS por cada VM.

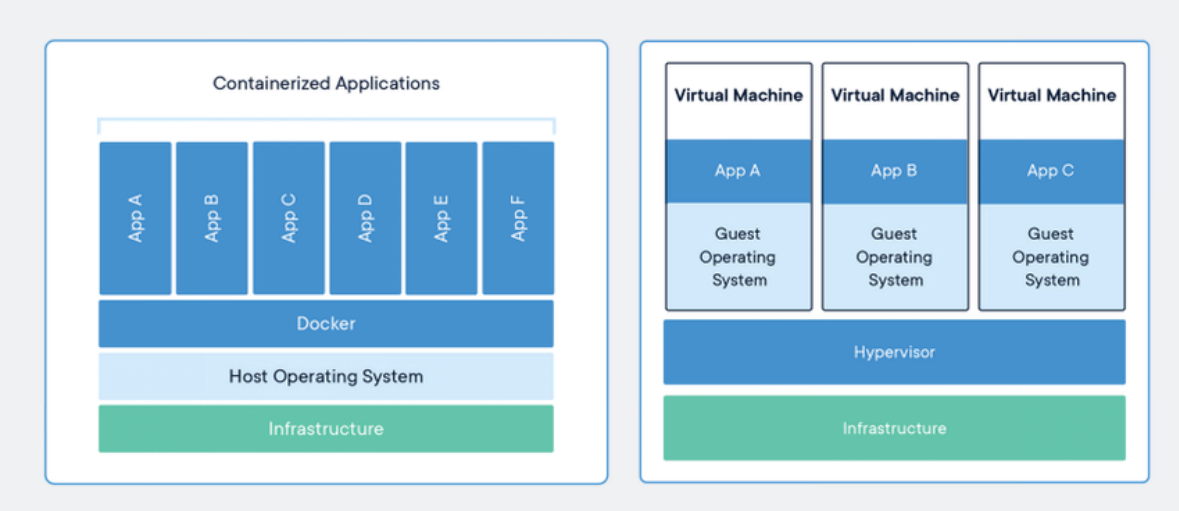
Contenedores

Los contenedores son una tecnología de virtualización a nivel del sistema operativo que permite ejecutar aplicaciones y sus dependencias en procesos autocontenidos y aislados, conocidos como contenedores (Soltesz et al., 2007). A diferencia de las máquinas virtuales, que emulan hardware completo y requieren un sistema operativo completo para cada instancia, los contenedores comparten el *kernel* del sistema operativo del host y aíslan las aplicaciones a nivel de espacio de usuario, resultando en una sobrecarga significativamente reducida y un inicio más rápido.

En la figura 4, se muestran las diferencias conceptuales entre una arquitectura de las VMs y los contenedores (usando Docker como el Engine Container).

Figura 4

Arquitectura VM vs Contenedor



Los contenedores ofrecen una serie de ventajas sobre las máquinas virtuales tradicionales. Estas incluyen una mayor eficiencia en el uso de recursos, ya que los contenedores comparten el *kernel* del sistema operativo y utilizan menos recursos que las VMs; una mayor portabilidad, debido a que los contenedores pueden moverse fácilmente entre diferentes plataformas y entornos; y una mejor escalabilidad, pues los contenedores pueden ser iniciados y detenidos rápidamente.

Existen varios motores de contenedores populares, cada uno con sus propias ventajas y desventajas, como lo son Docker, containerd, podman, LXC, runCDocker, CoreOS RKT, entre

otros. Actualmente la tecnología más adoptada a nivel de contenedores son Docker, podman y containerd. Con la tecnología de contenedores han llegado nuevos retos en los procesos de orquestación, despliegue, mantenimiento y gestión de cada uno de los contenedores o clúster de contenedores.

Docker

Docker es una plataforma de software que permite la automatización del despliegue, la escalabilidad y la gestión de aplicaciones a través del uso de contenedores. Esta tecnología, lanzada por Docker Inc. en 2013, ha transformado la forma en que las organizaciones desarrollan, entregan y mantienen el software (Merkel, 2014).

La innovación central de Docker es la encapsulación de aplicaciones y sus dependencias en unidades autocontenidas y portables llamadas contenedores, que son más ligeras y eficientes que las máquinas virtuales tradicionales. Un contenedor Docker incluye todo lo que una aplicación necesita para ejecutarse, incluyendo el código, las bibliotecas del sistema, las herramientas del sistema y las configuraciones de tiempo de ejecución. Este enfoque asegura que las aplicaciones funcionarán de la misma manera sin importar dónde se implementen, eliminando así el problema comúnmente conocido como "funciona en mi máquina" (Turnbull, 2014).

Docker se basa en la tecnología de contenedores de Linux, pero también proporciona una interfaz de usuario sencilla y una arquitectura de cliente-servidor que facilita la gestión de contenedores. Además, Docker introduce la noción de imágenes de Docker, que son plantillas de sólo lectura utilizadas para crear contenedores. Las imágenes de Docker pueden ser compartidas y distribuidas a través de Docker Hub, un repositorio en línea que también ofrece una amplia variedad de imágenes de la comunidad y oficiales, lo que permite a los desarrolladores aprovechar el trabajo previo y reutilizar el código de manera más efectiva (Docker Inc., s. f.)

Containerd

Containerd es una herramienta esencial en el ecosistema de los contenedores de Docker. Originalmente se creó como parte de Docker, pero posteriormente se separó para convertirse en un proyecto independiente de código abierto, con el objetivo de proporcionar un motor de contenedores mínimo y estable que pudiera ser utilizado por Docker y otras plataformas (Debab & Hidouci, 2021).

Containerd se encarga de las tareas fundamentales de administración de contenedores, como la gestión de imágenes, el ciclo de vida de los contenedores, el almacenamiento y la red. Proporciona una API gRPC, lo que facilita la integración con otras herramientas y tecnologías. Además, Containerd es compatible con la Open Container Initiative (OCI), un estándar para la portabilidad de contenedores entre diferentes plataformas y sistemas operativos.

Esta herramienta ha sido diseñada para ser ligera, con un enfoque en la confiabilidad y el rendimiento. A diferencia de Docker, que incluye características de nivel superior como una interfaz de línea de comandos, una API REST y un sistema de plugins, Containerd se centra exclusivamente en la administración de contenedores a un nivel más bajo. Este enfoque lo hace especialmente adecuado para su uso en entornos de producción y en la nube, donde la simplicidad y la estabilidad son primordiales (Espe et al., 2020).

Orquestación de contenedores

La orquestación de contenedores se refiere al proceso de automatización, coordinación y gestión de la vida útil de los contenedores en infraestructuras de TI (Hightower et al., 2017). Esta práctica se ha convertido en un pilar esencial en entornos *DevOps* debido a su contribución en la eficiencia operativa, la escalabilidad y la resiliencia de las aplicaciones.

Las herramientas de orquestación de contenedores asumen varias funciones esenciales, incluyendo el aprovisionamiento y despliegue de contenedores, el balanceo de carga, el descubrimiento de servicios, la escalabilidad automática y la recuperación de fallos. Además, estas herramientas pueden manejar tareas más avanzadas como la gestión del almacenamiento y la red, la gestión de secretos y la aplicación de políticas de seguridad (Abraham et al., 2020).

Kubernetes, desarrollado originalmente por Google y donado a la Cloud Native Computing Foundation, se ha consolidado como la herramienta de orquestación de contenedores de facto en la industria. Kubernetes utiliza una arquitectura de clúster y ofrece una amplia gama de características que facilitan la orquestación de contenedores a gran escala. Su adopción ha crecido exponencialmente desde su lanzamiento en 2015. Según el informe "Cloud Native Landscape" de (CNCF, 2023), más del 83% de las organizaciones que utilizan contenedores también utilizan Kubernetes.

Kubernetes

Kubernetes, un término griego que significa "piloto" o "navegante", es un sistema de código abierto para automatizar el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores (Hightower et al., 2017). Fue desarrollado por Google en 2014 y actualmente es mantenido por la Cloud Native Computing Foundation (CNCF).

Proporciona un marco para ejecutar sistemas de aplicaciones distribuidas de manera resiliente, con escalamiento horizontal y facilitando las actualizaciones. Las aplicaciones se pueden desplegar en forma de "pods", que son las unidades más pequeñas y sencillas en el modelo de objetos de Kubernetes. Lo que permite llevar actividades de orquestación como son dónde debe ir un pod, cómo se monitoriza, cómo se reinician cuando tengan un problema, cómo escalar de forma automática de 1 a 20 réplicas en función de la carga, y así, una larga lista de más acciones posibles; además, ofrece una abstracción que le permite desplegar las aplicaciones en un clúster sin pensar en las máquinas que lo soportan.

En términos de despliegue, ofrece diversas herramientas y métodos para facilitar esta tarea. Helm, por ejemplo, es un gestor de paquetes para Kubernetes que permite definir, instalar y actualizar aplicaciones complejas. Helm utiliza un formato de empaquetado llamado "charts", que son colecciones de archivos que describen una aplicación relacionada con Kubernetes.

Otra herramienta importante para el despliegue en Kubernetes es Kustomize, que permite la personalización de recursos de aplicaciones para diferentes entornos y escenarios utilizando archivos de configuración de Kubernetes.

Además, existen herramientas de despliegue continuo (CD) específicamente diseñadas para Kubernetes, como *ArgoCD* y Flux. Estas herramientas permiten el despliegue automatizado y la gestión de aplicaciones en Kubernetes utilizando el principio de *GitOps*, donde el estado deseado de la aplicación se describe y versiona en un repositorio Git (Yuen et al., 2021).

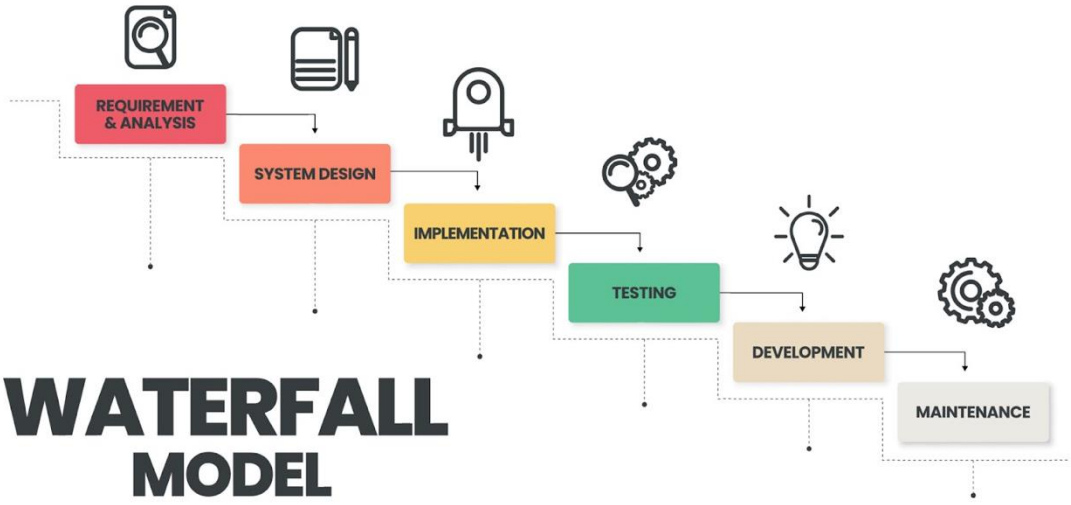
Ciclo de vida del software

Las fases del ciclo de desarrollo de software tradicional

El desarrollo de software es un proceso metódico que involucra una serie de fases discretas. El modelo de ciclo de vida de desarrollo de software (SDLC Systems Development Life Cycle) tradicional, también conocido como modelo en cascada, es una representación conceptual de este proceso. Este modelo, descrito por primera vez por Winston Royce en 1970,

sigue un enfoque secuencial y lineal con fases distintas y bien definidas (Royce, 1987), ver figura 5.

Figura 5
Modelo cascada por Winston Royce



Las fases del ciclo de desarrollo de software tradicional se pueden definir de la siguiente manera:

Requisitos: Esta fase implica la identificación y documentación de los requisitos del sistema y del software. Los analistas de sistemas suelen trabajar en estrecha colaboración con los clientes o usuarios finales para entender sus necesidades y expectativas (Sommerville, 2005).
Diseño: Una vez que se han recopilado y validado los requisitos, el equipo de desarrollo comienza a diseñar la solución de software. Esto puede incluir la arquitectura del sistema, la interfaz de usuario, los modelos de datos y los algoritmos de procesamiento (Barghoth et al., 2020).
Implementación/Codificación: En esta fase, los desarrolladores traducen los diseños en código fuente utilizando lenguajes de programación apropiados. Según un informe de («TIOBE Index»,

2021) algunos de los lenguajes de programación más utilizados en esta fase incluyen Java, Python, y C++.

Pruebas: Esta fase implica la verificación del software para detectar y corregir errores. Las pruebas pueden incluir pruebas unitarias, pruebas de integración, pruebas de sistema y pruebas de aceptación (Manaseer et al., 2015).

Mantenimiento: Una vez que el software se ha desplegado y está en uso, puede requerir mantenimiento para corregir errores, mejorar el rendimiento o añadir nuevas características (Pfleeger & Atlee, 2009).

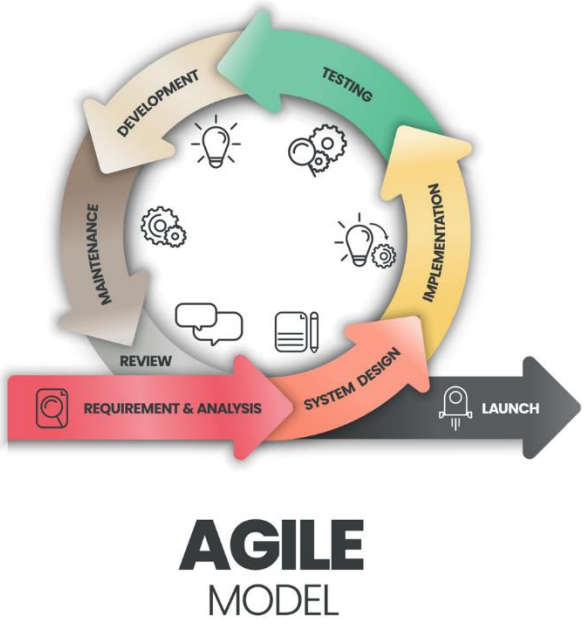
Metodologías ágiles en el desarrollo de software

El desarrollo de software ha evolucionado significativamente en las últimas décadas, con el surgimiento y la adopción de nuevas metodologías en respuesta a los enfoques en cascada en el desarrollo de proyectos de software, que buscan mejorar la eficiencia, la calidad y la satisfacción del cliente.

Una de estas metodologías es el desarrollo de software ágil, que se centra en la colaboración, la adaptabilidad y la entrega continua; en la ***Figura 6*** se ilustra el ciclo de este modelo. Las metodologías ágiles se caracterizan por la entrega incremental y la retroalimentación continua, con el objetivo de proporcionar valor al cliente lo más rápido posible (Beck et al., 2001). Esto contrasta con las metodologías tradicionales de desarrollo de software, como el modelo en cascada, que a menudo requieren una planificación y diseño detallados antes de la implementación (Royce, 1987). Un equipo de desarrolladores, proponen este nuevo enfoque para el desarrollo de software y describen cuatro características fundamentales que se deben priorizar en el desarrollo, a lo cual llamaron manifiesto Ágil (Beck et al., 2001).

- **Las personas y las interacciones** antes que los procesos y las herramientas.
- **El software en funcionamiento** antes que la documentación exhaustiva.
- **La colaboración con el cliente** antes que la negociación contractual.
- **La respuesta ante el cambio** antes que el apego a un plan.

Figura 6
Modelo General de Agile



Las metodologías ágiles incluyen prácticas como Scrum, Kanban, Programación Extrema (XP) y Desarrollo Guiado por Pruebas (TDD), entre otras (Anderson, 2010; Beck et al., 2001; Schwaber & Beedle, 2002). Estas metodologías proporcionan un marco para el desarrollo de software que permite a los equipos adaptarse a los cambios y responder rápidamente a las necesidades del cliente.

El desarrollo de software utilizando metodologías ágiles ofrece una serie de beneficios potenciales, incluyendo una mejora en la calidad del software, una mayor satisfacción del cliente y una mayor capacidad para gestionar el cambio. Sin embargo, también presenta retos que requieren una consideración cuidadosa. A medida que las prácticas ágiles siguen siendo adoptadas y adaptadas, es esencial continuar la investigación en esta área para optimizar su implementación y uso.

Una de las áreas que ha migrado a las metodologías ágiles es la de Operaciones TI, naciendo así nuevos conceptos como *DevOps* y la Ingeniería de Confiabilidad del Sitio (SRE) que han transformado la forma en que se abordan las tareas de Operaciones TI. Estas

metodologías promueven la colaboración entre desarrolladores y equipos de operaciones, utilizando el software como medio para administrar sistemas y automatizar funciones operativas. La implementación de procesos de integración continua y despliegue continuo (CI/CD) agiliza la entrega de nuevo código y mejora la confiabilidad y calidad del software.

Prácticas DevOps

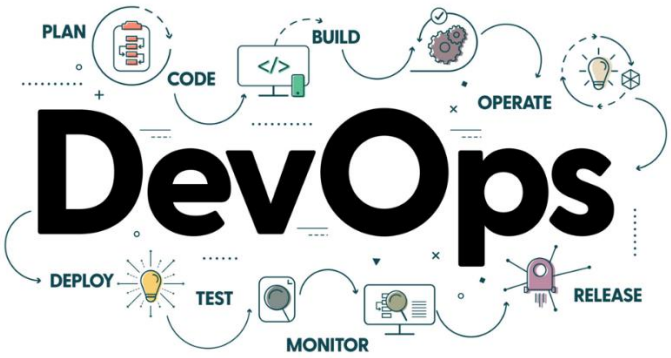
DevOps es un término que surge de la combinación de "desarrollo" y "operaciones", y representa una evolución en la forma en que los equipos de desarrollo de software y operaciones de TI interactúan y colaboran para proporcionar mejores productos y servicios, más rápidamente (Lwakatare et al., 2015).

Según la definición proporcionada por (RedHat, Inc., 2022) *"DevOps es un modo de abordar la cultura, la automatización y el diseño de las plataformas para generar mayor valor empresarial y capacidad de respuesta, mediante la prestación ágil de servicios de alta calidad. Todo eso es posible gracias a la prestación rápida y constante de los servicios de TI. DevOps implica vincular las aplicaciones heredadas con aquellas diseñadas para la nube y las infraestructuras más nuevas."*

El movimiento DevOps comenzó a principios de la década de 2010 como respuesta a la división tradicional entre los equipos de desarrollo de software y operaciones de TI (Kim et al., 2016). Esta división a menudo llevaba a una falta de comunicación y colaboración, lo que resulta en retrasos en la entrega, problemas de calidad y una menor satisfacción del cliente.

Desde una perspectiva teórica, DevOps se constituye de distintas etapas que representan un ciclo completo de desarrollo y operación de software, que se detalla a continuación. Cada paso juega un papel crucial:

Figura 7
Pasos de prácticas DevOps



Planificación (Plan): Este es el primer paso donde se identifican las necesidades del proyecto y se definen los objetivos y metas. Se realiza un entendimiento detallado de las necesidades de los usuarios y se traza un plan detallado para el desarrollo del software.

Codificación (Code): Aquí, los desarrolladores comienzan a escribir el código siguiendo las especificaciones establecidas en la etapa de planificación. Este paso es crucial, ya que es donde los requisitos se transforman en un producto tangible.

Construcción (Build): En esta etapa, se compila el código fuente en código ejecutable. Este proceso también incluye la revisión del código, las pruebas unitarias y la integración del código.

Pruebas (Test): Aquí, se realizan pruebas exhaustivas para asegurarse de que el software funciona según lo previsto y para detectar y solucionar cualquier error o defecto antes del despliegue.

Despliegue (Deploy): Una vez que el software ha pasado con éxito la etapa de prueba, se despliega en el entorno de producción. En DevOps, este proceso se realiza de manera continua y automática para permitir un desarrollo más rápido y eficiente.

Operación (Operate): Este paso implica el mantenimiento continuo y la supervisión del software para asegurar que esté operando de manera eficiente y según lo esperado.

Liberación (Release): Durante esta fase, las nuevas características, actualizaciones y correcciones son liberadas para ser utilizadas por los usuarios finales. Esta etapa también implica la gestión de las versiones y la preparación de las mismas para su despliegue.

Monitoreo (Monitor): Aquí, se supervisa el rendimiento del software y la infraestructura para asegurar que todo funcione como se espera y para detectar y solucionar rápidamente cualquier problema. Este paso es vital para mantener la estabilidad y la fiabilidad del sistema.

Principios clave de DevOps:

Colaboración: Fomentar la comunicación y la colaboración entre los equipos de desarrollo y operaciones, rompiendo las barreras organizativas tradicionales. La colaboración mejora la eficiencia y la calidad del desarrollo de software.

Integración continua: Integrar los cambios de código de forma continua y automática, lo que permite una detección temprana de errores y una entrega más rápida. La integración continua garantiza la estabilidad y la calidad del software.

Despliegue continuo: Automatizar el despliegue de software en entornos de producción de manera rápida y confiable. El despliegue continuo reduce los tiempos de entrega y facilita la implementación de cambios.

CI/CD

La Integración Continua y la Entrega Continua (CI/CD, por sus siglas en inglés) son prácticas que forman un pilar central en el enfoque de DevOps. CI/CD buscan mejorar la calidad del software y acelerar el proceso de entrega, permitiendo a los equipos de desarrollo entregar código en producción de manera segura, rápida y sostenible (Kim et al., 2021) .

La Integración Continua (CI) es la práctica de fusionar todos los cambios de código de los desarrolladores en una línea principal de código compartida varias veces al día. Cada integración se verifica mediante una compilación automatizada y pruebas para detectar errores lo más rápido posible (Duvall et al., 2007) . Esto facilita la detección temprana de problemas de integración, mejora la calidad del software y reduce el tiempo que se tarda en validar y lanzar nuevas actualizaciones de software.

La Entrega Continua (CD), por otro lado, es una extensión de la CI. En CD, todos los cambios de código que pasan la fase de compilación y pruebas son desplegados automáticamente en el entorno de producción, aunque el paso final de "despliegue" puede requerir un clic de aprobación manual (Humble & Farley, 2010) . La entrega continua tiene como objetivo permitir a

los equipos liberar nuevas características y cambios a los usuarios de forma segura, rápida y sostenible.

Es esencial destacar que la CI/CD no es solo un conjunto de herramientas o tecnologías. Es un cambio cultural que implica una mentalidad de colaboración, prácticas de desarrollo y operaciones sólidas, y la automatización de las pruebas y los despliegues (Kim et al., 2021). Este enfoque permite a los equipos de desarrollo responder rápidamente a las necesidades del negocio, reducir el tiempo de inactividad del software y mejorar la satisfacción del usuario.

SRE (Site Reliability Engineering)

Site Reliability Engineering (SRE) es una disciplina que se originó en Google a principios del año 2014, y ha crecido en relevancia en los últimos años debido a su enfoque único para abordar los desafíos de infraestructura y operaciones de software. El SRE combina elementos de ingeniería de software y operaciones de sistemas para crear sistemas altamente escalables y confiables (Beyer et al., 2016).

SRE es una práctica que busca balancear el desarrollo de características nuevas y la confiabilidad del sistema, ya que mantener un sistema 100% disponible puede impedir el desarrollo de nuevas funciones y, por otro lado, enfocarse demasiado en características nuevas puede llevar a la inestabilidad del sistema (Beyer et al., 2016). Para equilibrar esto, los equipos de SRE utilizan una métrica conocida como presupuesto de error, lo que posibilita un cierto grado de tiempo de inactividad permitido que concede el desarrollo de nuevas funciones (Beyer et al., 2016).

Un concepto fundamental en SRE es la automatización. El trabajo manual es propenso a errores, no escala bien y no agrega valor a largo plazo. SRE promueve la automatización en todas las áreas posibles, como la solución de problemas, la configuración del sistema y las pruebas de integridad del sistema (Beyer et al., 2016). Este enfoque permite a los equipos centrarse en tareas de alto valor que requieren entre otras habilidades humanas, creatividad.

Un aspecto distintivo de SRE es su enfoque en la medición y cuantificación de los problemas del sistema. En lugar de confiar en la intuición o en la reacción a los problemas después de que ocurren, los equipos de SRE utilizan un enfoque basado en datos para predecir y prevenir problemas futuros. Esto incluye la monitorización constante del sistema, la recopilación

de datos sobre el rendimiento del sistema y el análisis de estos datos para identificar patrones y tendencias (Beyer et al., 2016).

El objetivo final de SRE es mejorar la fiabilidad del sistema sin ralentizar el desarrollo de nuevas características. Este equilibrio es clave para el éxito de cualquier organización en el rápido mundo del software y la tecnología.

Metodología

En un entorno empresarial altamente competitivo, es esencial contar con una cultura DevOps sólida para lograr una eficiencia operativa y una entrega de software más rápida. No obstante, el proceso de implementación se puede ver afectado por las limitantes de recurso y tiempo; es por ello que el presente proyecto se llevó a cabo a través de la metodología Kanban que es ideal cuando se requiere flexibilidad, adaptación y entrega continua de valor. A continuación, se detallan los pasos de este proceso:

El primer paso para implementar la metodología Kanban en el proyecto fue definir las distintas etapas del flujo de trabajo del proyecto (Anderson, 2010). Estas etapas se basaron en los objetivos específicos del proyecto y en las prácticas generales de la metodología *DevOps*. Las etapas se definieron de la siguiente manera: análisis e investigación, selección de herramientas, diseño de la estrategia, implementación del diseño de las herramientas, evaluación y propuestas de trabajos futuros.

Etapa de Análisis e Investigación

Se inició con la revisión de literatura existente sobre la administración de servidores, técnicas que han aplicado grandes empresas para abordar problemáticas semejantes a la que tenía la Facultad de Ingeniería, la automatización y orquestación de cargas de trabajo, *DevOps*, CI/CD y SRE. Se llevó a cabo una evaluación detallada de la infraestructura física y su modelo actual para el despliegue de aplicaciones, donde se identificaron los desafíos y beneficios potenciales de implementar una cultura *DevOps* orientada a la eficiencia de servidores físicos.

Etapa de Selección de herramientas

Se seleccionaron una serie de herramientas y técnicas para la automatización de la infraestructura en la Facultad de Ingeniería. Esto incluyó las herramientas para el aprovisionamiento de servidores físicos y VMs, la orquestación de contenedores y las técnicas para la implementación de la cultura DevOps.

Etapa de diseño de la estrategia

Se diseñó la nueva infraestructura que maximiza la eficiencia operativa. Se contemplaron aspectos como la virtualización, la utilización de contenedores y la escalabilidad. Se tuvo en cuenta no solo las necesidades actuales, sino también la posibilidad de crecimiento y expansión futura de la carga de trabajo y se diseñó la estrategia DevOps que se va a implementar en el ciclo de desarrollo de software.

Etapa de implementación

Se establecieron varios pasos esenciales para asegurar una transición efectiva hacia una cultura de DevOps. Los detalles de estos pasos se describen a continuación:

Configuración de la Infraestructura: Inicialmente, se procedió con la implementación de la nueva infraestructura de servidores virtuales y contenedores que se había diseñado en las etapas anteriores. Este proceso implicó la configuración de los servidores físicos, la creación de entornos virtuales y la instalación y configuración de los contenedores.

Instalación de Herramientas DevOps: Después de configurar la infraestructura, se instaló y configuró un conjunto de herramientas DevOps seleccionadas para respaldar las prácticas de integración y entrega continua, la gestión de la configuración, el monitoreo y la respuesta a incidentes. Estas herramientas incluían sistemas de control de versiones, servidores de integración continua, sistemas de gestión de configuración y plataformas de monitoreo.

Pruebas de la Infraestructura y las Herramientas: Una vez implementada la infraestructura y las herramientas, se llevó a cabo un conjunto de pruebas exhaustivas para asegurar que todo funcionaba como se esperaba. Esto incluía pruebas de las funciones individuales de las herramientas, así como pruebas de la integración entre las diferentes herramientas y la infraestructura en general.

Formación de los equipos: Una vez que la infraestructura y las herramientas estaban en su lugar y funcionando correctamente, se llevó a cabo la formación de los equipos de desarrollo y operaciones. Esta formación se centró en cómo utilizar la nueva infraestructura y las herramientas y en cómo trabajar juntos de manera más eficaz bajo una cultura DevOps.

Etapa de evaluación

Se llevó a cabo la evaluación del impacto de la implementación de DevOps y la automatización de la infraestructura en la eficiencia y confiabilidad de los servicios en la Facultad de Ingeniería. Para ello se utilizaron indicadores clave como el tiempo de entrega, la calidad del software y la disponibilidad de los servicios para medir el éxito de la implementación usando los Indicadores clave de rendimiento (KPI) y recomendados por el proyecto DORA:

- Tiempo promedio de despliegue del software.
- Tasa de éxito de despliegues.
- Productividad operativa medida en entregas por período.

Etapa de propuestas de mejora

Se desarrollaron una serie de recomendaciones y mejores prácticas basadas en los resultados obtenidos en la etapa de evaluación.

A través, de la aplicación de esta metodología Kanban, se logró implementar con éxito una cultura DevOps eficiente en el uso de servidores físicos, mejorando significativamente los tiempos de despliegue del software y maximizando la productividad operativa.

Resultados

El área de Infraestructura y Operaciones (I&O) desempeña un papel fundamental en el funcionamiento eficiente de cualquier institución y la Facultad de Ingeniería no es una excepción. Sin embargo, es importante realizar un análisis exhaustivo del estado actual de este departamento para identificar posibles áreas de mejora y optimización. En este proyecto, se ha llevado a cabo un acercamiento detallado al estado actual del área de I&O de la Facultad de Ingeniería, con el objetivo de evaluar su rendimiento, identificar posibles desafíos y proponer soluciones innovadoras.

Se inició con la primera fase del proyecto que es Etapa de Investigación y Análisis, durante la investigación se recurre inicialmente a cuatro fuentes bibliográficas que han marcado un hito en el campo del DevOps y el SRE que son “Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations”, ”Site Reliability Engineering”, “The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations” y “Start and Scaling Devops in the Enterprise”, con esta literatura se obtuvo la guía para iniciar con la evaluación del estado actual a nivel de infraestructura y prácticas DevOps.

En este análisis se atacaron cuatro frentes de trabajo que son la evaluación de la infraestructura, revisión de los procesos de despliegue actuales, análisis del sistema de control de versiones y finalmente validar si tiene prácticas de DevOps aplicadas.

Evaluación de la Infraestructura

Se llevó a cabo una evaluación y análisis de la infraestructura on-premise existente, evidenciando que la Facultad de Ingeniería cuenta con 9 servidores físicos que soportaban las operaciones cotidianas. Con el mapeo completo de la infraestructura, que se puede ver en la ***Tabla 1***, se identificó que las cargas de trabajo estaban distribuidas de manera ineficiente entre los servidores. De hecho, el 87% de las cargas de trabajo estaban asignadas a solo el 40% de los servidores físicos, dejando una distribución desbalanceada y en algunos casos, sobrecargando ciertos servidores mientras otros quedaban desaprovechados.

Tabla 1
Distribución servidores On-premise

Nombre	OS	VM	% Utilización
Server I	Windows Server 2008	1	15 %
Server II	RHEL 6.3	0	10 %
Server III	RHEL 7.3	12	60 %
Server IV	Windows Server 2012	0	5 %
Server V	Windows Server 2008	3	10 %
Server VI	Windows Server 2008	0	5 %
Server VII	Windows Server 2012	8	80 %
Server VIII	RHEL 6.3	4	20 %
Server IX	Windows Server 2019	0	10 %
Server X	RHEL 9**	0	0 %

** *Servidor recién adquirido*

Se ha identificado que no todos los servidores están destinados al uso de aplicaciones, específicamente IV, y VI está designado para la licencia en RED, II protocolo de respaldos y IX dedicado al directorio activo, motivo por lo que operan sin VM. En el análisis, se descubrió que los servidores I y II habían alcanzado el final de su ciclo de vida. Además, se identificó que el servidor I albergaba una máquina virtual (VM) en la cual se ejecutaba una aplicación de gran importancia para toda la institución.

Al revisar las máquinas virtuales existentes, se identificó una variedad de aplicaciones con diferentes propósitos y requerimientos de recursos operando en ellas. Esta diversidad, aunque proporcionaba cierta flexibilidad, también presentaba problemas de compatibilidad y eficiencia. Entre ellas se encontraba una VM con alrededor de 120 aplicaciones Java legacy y otra VM con alrededor de 80 aplicaciones de diversas tecnologías.

Se observó que entre las 28 máquinas virtuales solo 2 contaban con Docker, una para el entorno de producción y otra para pruebas. Esto representaba un uso efectivo de la tecnología de contenedores, pero el hecho de que solo se usarán en dos máquinas limitaba su potencial, además

no se encontraban orquestadas y su despliegue era manual usando la línea de comando de Docker.

Finalmente, se notó que algunas aplicaciones estaban instaladas directamente en los servidores físicos. Esta práctica, aunque común, puede generar problemas de dependencias y dificultar las tareas de mantenimiento y actualización.

Revisión de los Procesos de Despliegue

La facultad no disponía de un sistema de despliegue 100% automatizado e implementado, se contaba con una instancia de Jenkins instalada en una VM, pero se utilizaba solo para el despliegue de diez aplicaciones. Además, se observó que las prácticas de DevOps no se estaban aplicando en estas implementaciones. El paradigma DevOps, que combina el desarrollo y las operaciones para mejorar la colaboración y la productividad, puede reducir significativamente el tiempo de despliegue y mejorar la calidad del software (Forsgren, Humble, & Kim, 2018). Según un estudio realizado por Puppet Labs (2021), las organizaciones que implementan prácticas de DevOps pueden desplegar hasta 200 veces más rápido que aquellas que no lo hacen, lo que subraya la importancia de esta metodología.

Análisis del uso de sistema de control de versiones

Para el sistema de control de versiones la Facultad utilizaba GitLab Self-managed, pero su aplicación se limitaba a su función como sistema de control de versiones. GitLab, sin embargo, es más que un simple sistema de control de versiones: ofrece un conjunto integral de herramientas para apoyar las prácticas de DevOps, que incluyen gestión de proyectos, integración continua, despliegue continuo, monitorización y seguridad GitLab.com (2023). La infrautilización de GitLab resultaba en un uso ineficiente de los recursos disponibles y oportunidades perdidas para mejorar la colaboración y la eficiencia.

Al adoptar todas las características de DevOps disponibles en GitLab, se podrá mejorar la eficiencia del desarrollo y despliegue de software en un 35%. Esta cifra se basa en investigaciones que muestran que la adopción de las prácticas de DevOps, como la integración y el despliegue continuo, puede mejorar significativamente la eficiencia y la calidad del software (Forsgren, Humble, & Kim, 2018).

Prácticas DevOps

A pesar de que la Facultad de Ingeniería no cuenta con prácticas DevOps definidas, se analizan las interacciones y los procesos existentes entre los equipos de desarrollo y operaciones para identificar cualquier práctica informal que pueda estar en línea con los principios de DevOps. Se evidencio que muchas aplicaciones aún se desplegaban sin procesos de CI/CD en los cuales el desarrollador creaba el artefacto y este se compartía con el equipo de operaciones para su posterior despliegue; también se observa que se inició la implementación de la herramienta Jenkins para iniciar con procesos de CI/CD, esta estrategia inició con 10 aplicaciones con pipeline creados, estos pipelines contaban con dos pasos que era la compilación y el despliegue únicamente, sin prácticas DevOps.

Durante la etapa de análisis, se descubrió la ausencia de prácticas DevOps y el uso subóptimo de herramientas disponibles. En respuesta a esto, se realizó un análisis exhaustivo y evaluación de diversas herramientas y plataformas. La selección de estas herramientas estuvo guiada por criterios claros: debían ser de código abierto, integrarse fácilmente entre sí y ofrecer una gama de funcionalidades. Con estas herramientas seleccionadas, se alinearon mejor los requerimientos y objetivos del proyecto.

Selección de herramientas

GitLab como el motor fundamental de todo el ciclo de software, aprovechando su capacidad de autogestión y su robusto conjunto de características. GitLab sobresale no sólo como un eficiente sistema de control de versiones, sino también como una potente herramienta de integración continua y despliegue continuo (CI/CD), registro de contenedores, gestor de proyectos eficaz y el sistema de inicio de sesión único (SSO) para las aplicaciones integradas en el ciclo DevOps.

SonarQube como la herramienta esencial para llevar a cabo el Análisis Estático de Seguridad de Software (SAST) en los proyectos de Software. Esta elección se basó en diversos factores críticos: SonarQube es una solución Open Source, ofrece capacidades de autogestión y de manera crucial permite la autenticación con inicio de sesión único (SSO) en conjunción con GitLab.

Portainer como el administrador principal de contenedores, teniendo en cuenta la presencia de contenedores sin orquestar y que la implementación de un orquestador será solo una prueba de concepto (PoC). Portainer permite un control con listas de control de acceso (ACL) de los contenedores. Esta elección se basó en características claves: proyecto Open Source, la capacidad de autogestión y la posibilidad de autenticación mediante inicio de sesión único (SSO) con GitLab.

Kubernetes como el principal orquestador de contenedores. Esta elección estratégica posibilita optimizar y aprovechar eficientemente las capacidades de las máquinas físicas existentes en el entorno universitario. Además, Kubernetes brinda la flexibilidad para escalar las operaciones a medida que evoluciona la carga de trabajo. Con su arquitectura altamente adaptable, es posible añadir nodos en la nube cuando sea necesario. Por tanto, si la demanda de recursos computacionales aumenta, se podría expandir la infraestructura de Kubernetes a la nube, aprovechando la escalabilidad y la robustez de AWS. Este resultado resalta la eficacia de la estrategia elegida, permitiendo manejar los desafíos y demandas cambiantes del ciclo vida del software.

Rancher como administrador principal para Kubernetes. Su elección se basa en la capacidad de autogestión y en el sólido soporte que ofrece la comunidad de usuarios y desarrolladores de Rancher. La habilidad de Rancher para facilitar la administración de clústeres de Kubernetes se ha destacado de manera integral. Gracias a esta decisión, se gestionaron de manera eficiente las operaciones de DevOps y CI/CD, respaldados por una comunidad activa y experta que se esfuerza constantemente por mejorar y optimizar el rendimiento y la funcionalidad de Rancher.

Traefik es el balanceador de carga esencial para la infraestructura de la facultad. Traefik se utiliza tanto para las aplicaciones alojadas en contenedores no orquestados como para el ingress de Kubernetes. Esta estrategia se ha implementado con el objetivo de mantener una homogeneidad entre la infraestructura actual y la prueba de concepto (PoC) del clúster de Kubernetes, facilitando su futura transición a un entorno de producción.

Terraform como solución de Infraestructura como Código (IaC) para la administración de la Máquina Virtual (VM) del proyecto. Esta herramienta proporciona una plataforma efectiva para automatizar la creación, modificación y eliminación de las VMs en función de las necesidades.

Ansible como el principal medio de automatización de la configuración de las Máquinas Virtuales (VM).

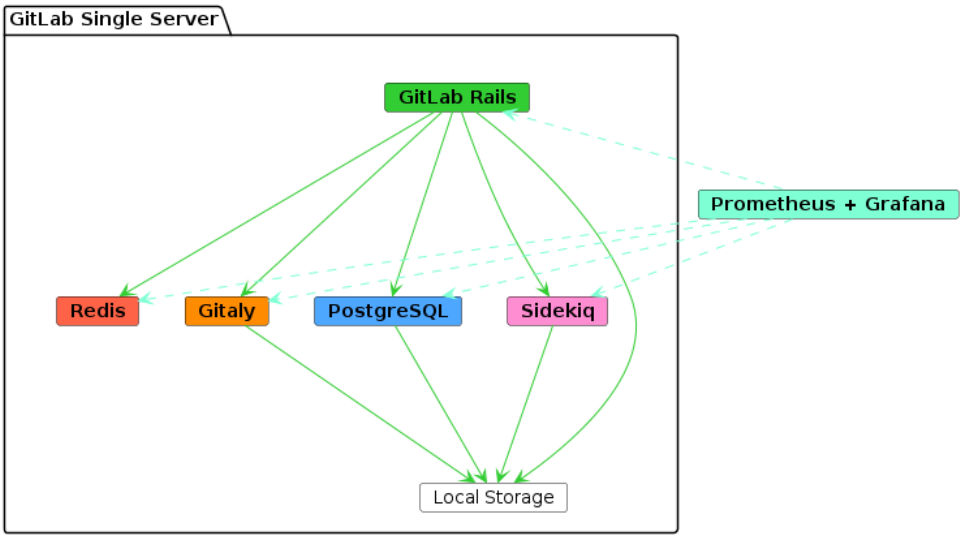
OWASP ZAP como la herramienta principal para llevar a cabo el Análisis Dinámico de Seguridad de Aplicaciones (DAST) en los proyectos de software. Esta decisión representa un resultado crucial en el presente estudio, demostrando la importancia de una evaluación de seguridad en profundidad para garantizar la calidad y la seguridad en los proyectos. OWASP ZAP, al proporcionar pruebas dinámicas de aplicaciones, añade una capa esencial de seguridad en el entorno de DevOps y CI/CD, mejorando la capacidad para detectar y rectificar vulnerabilidades antes de que se conviertan en problemas serios.

Etapa de implementación

Instalación del GitLab Self-hosted

El proyecto se centra en el desarrollo y la operación, conocido como DevOps, y en la integración continua y despliegue continuo, denominados CI/CD. GitLab, es la herramienta que respaldará todas estas operaciones. Se implementó GitLab basándose en la arquitectura de referencia diseñada para soportar hasta 1000 usuarios como se ve en la ***Figura 8***.

Figura 8
Arquitectura GitLab para un servidor



Con base a la arquitectura de referencia y las sugerencias en los requisitos del servidor, se ha configurado una Máquina Virtual (VM) con características específicas para proporcionar un rendimiento óptimo. Esta VM está basada en el sistema operativo Debian 11; una elección popular y probada para entornos de servidor debido a su estabilidad y seguridad; está equipada con 8 Unidades de Procesamiento Central (CPU), esto garantiza que pueda manejar múltiples tareas simultáneamente sin ralentizaciones significativas. Tiene un tamaño de memoria de 16 GB de RAM; esta amplia cantidad de memoria permite el procesamiento de grandes cantidades de datos y la ejecución de múltiples procesos sin comprometer el rendimiento de la máquina, y un almacenamiento de 200 GB; que se hace necesario para alojar tanto la base de código, el registro de contenedores, los artefactos y los datos de usuario, con suficiente margen para futuras expansiones y actualizaciones.

En la ***Figura 9***, ***Figura 10*** y ***Figura 11*** se presentan los resultados de la implementación de GitLab en modo self-hosted. Las capturas de pantalla adjuntas ilustran claramente el resultado de la instalación. Para obtener una descripción más detallada del proceso de instalación remítase al Anexo 1. Instalación del GitLab.

Figura 9

Servidor GitLab configurado

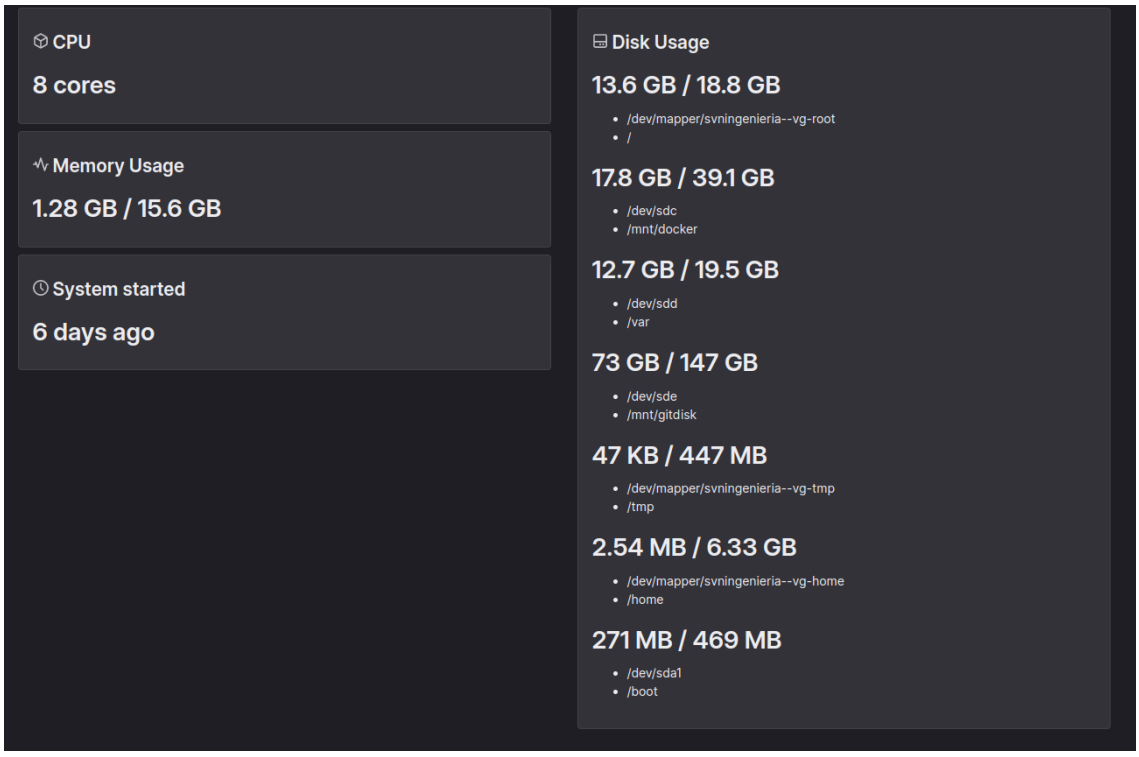


Figura 10
Servidor GitLab CPU y RAM

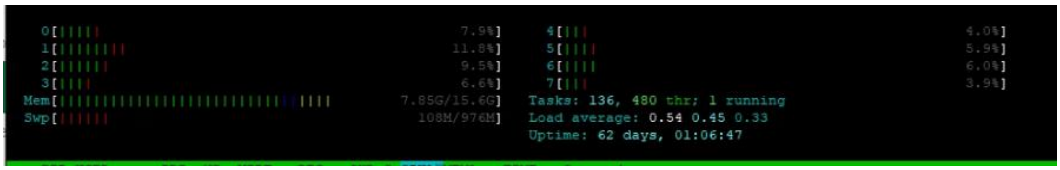
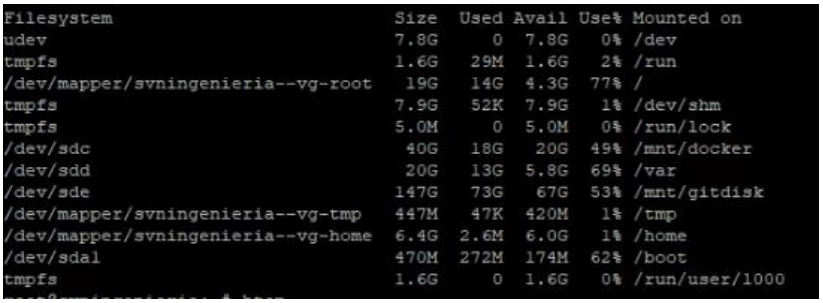


Figura 11
Distribución de discos del servidor de GitLab



Filesystem	Size	Used	Avail	Use%	Mounted on
udev	7.8G	0	7.8G	0%	/dev
tmpfs	1.6G	29M	1.6G	2%	/run
/dev/mapper/svningenieria--vg-root	19G	14G	4.3G	77%	/
tmpfs	7.9G	52K	7.9G	1%	/dev/shm
tmpfs	5.0M	0	5.0M	0%	/run/lock
/dev/sdc	40G	18G	20G	49%	/mnt/docker
/dev/sdd	20G	13G	5.8G	69%	/var
/dev/sde	147G	73G	67G	53%	/mnt/gitdisk
/dev/mapper/svningenieria--vg-tmp	447M	47K	420M	1%	/tmp
/dev/mapper/svningenieria--vg-home	6.4G	2.6M	6.0G	1%	/home
/dev/sda1	470M	272M	174M	62%	/boot
tmpfs	1.6G	0	1.6G	0%	/run/user/1000

Habiendo instalado GitLab exitosamente como se evidencia en la **Figura 12**, el paso a seguir fue implementar y configurar herramientas adicionales para respaldar integralmente el ciclo de vida del software. Inicialmente, se instala el registro de contenedores de GitLab, una funcionalidad crucial que facilita la administración y distribución de imágenes de contenedores.

Se activa el OAuth 2.0 Identity Provider API. Esto permite utilizar GitLab como el proveedor de autenticación OAuth, mejorando así la seguridad y proporcionando una experiencia de usuario más fluida para el equipo de desarrollo.

Por último, se implementaron los GitLab Runners aprovechando los servers I y II para evitar la sobrecarga del GitLab, se puede ver en la ***Figura 13*** que se instalaron 16 instancias. Esta estrategia permite optimizar los recursos y asegurar que el servidor principal de GitLab esté libre para gestionar otras tareas críticas.

Figura 12
Dashboard GitLab al final de la implementación

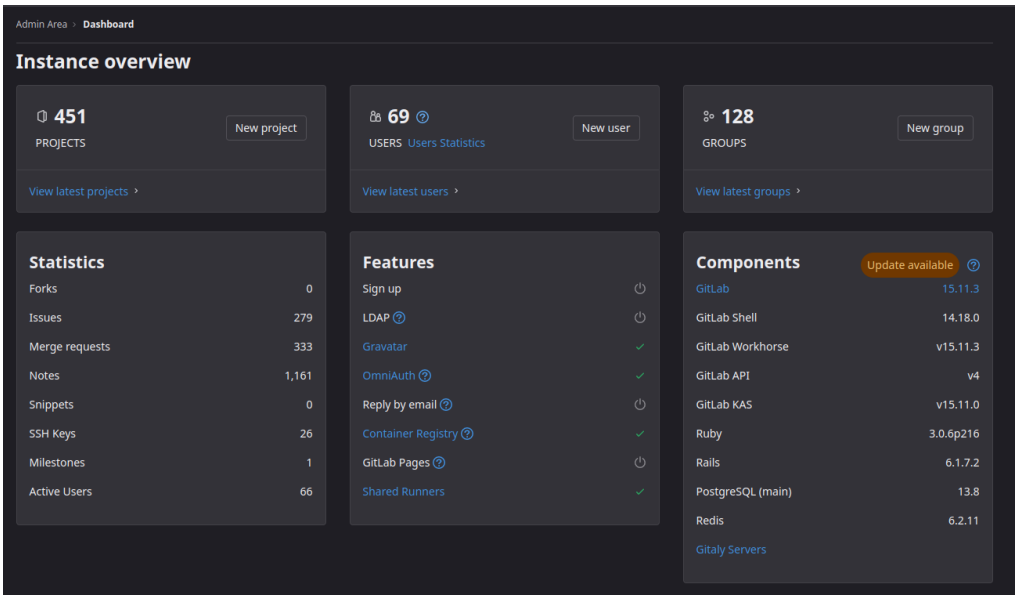
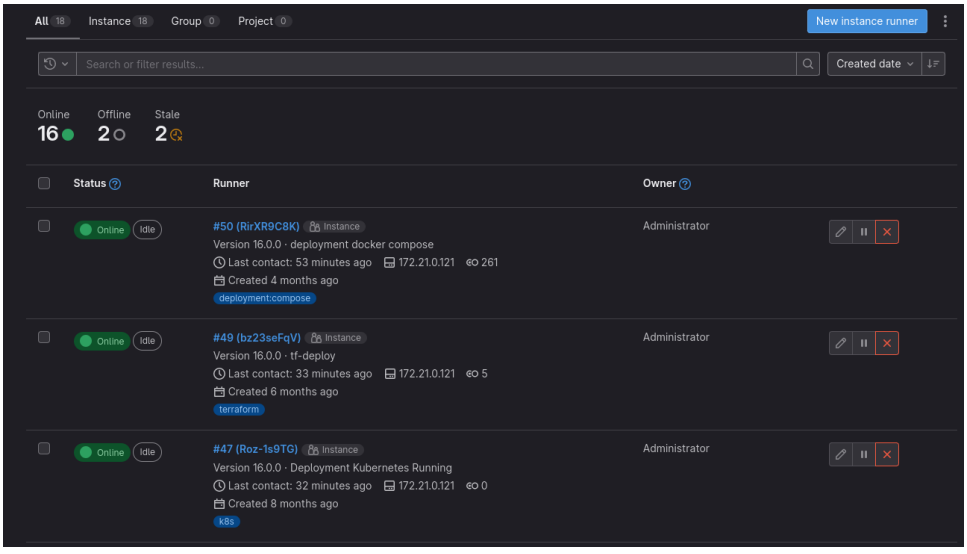


Figura 13
Runner instalados de GitLab



Instalación de herramientas para prácticas DevOps

Portainer

La primera herramienta que se implementó es Portainer. Esta herramienta proporciona un control excepcional sobre los contenedores desplegados tanto en el ambiente de producción como de pruebas. Se detalla los resultados obtenidos:

Instalación de Portainer: Se instala la versión Business Edition con los pasos descritos en el manual de instalación de su página web (Portianer, Inc., s. f.). Esta herramienta se distingue por su capacidad para proporcionar un control superior sobre los contenedores desplegados tanto en el entorno de producción como de pruebas. El proceso de instalación en sí fue sencillo y estandarizado, gracias a las ventajas de trabajar con contenedores, vemos la página de bienvenida en la ***Figura 14***.

Adición de Nodos: Una vez instalado Portainer, se procede a agregar los nodos de los servidores de prueba y producción como se ve en la ***Figura 15***. Este paso es crucial para asegurar que todas las máquinas que componen la infraestructura estén representadas y puedan ser administradas a través de Portainer.

Configuración de Acceso: El siguiente paso fue configurar el acceso a Portainer utilizando las credenciales de GitLab. Este proceso garantiza una gestión segura y centralizada de los derechos de acceso, lo que facilita enormemente la administración de la infraestructura.

Adición del Repositorio de Contenedores de GitLab: A continuación, se configuró el repositorio de contenedores de GitLab a Portainer ***Figura 16***. Este paso permite mantener un enlace directo y constante con el lugar donde se almacenan y actualizan las imágenes de los contenedores, permitiendo implementar actualizaciones y nuevas versiones de manera eficiente.

Configuración de la Lista de Control de Acceso (ACL): Por último, se configuró una Lista de Control de Acceso (ACL) para gestionar el ingreso a los proyectos de cada desarrollador. Para ello, se utilizó las etiquetas (labels) de los contenedores ***Figura 17***, lo que permite asignar de manera precisa y segura los niveles de acceso adecuados para cada miembro del equipo. Esto garantiza que cada desarrollador tenga acceso solo a los recursos que necesita, aumentando la seguridad y aislamiento de los proyectos.

Figura 14
Página principal del portainer con acceso SSO

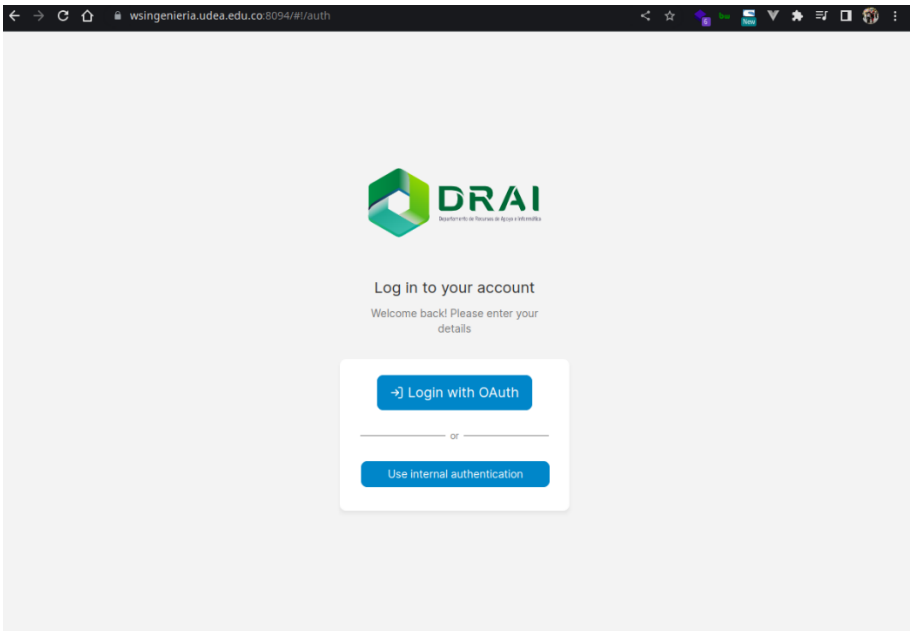


Figura 15
Nodos instalados en el portainer

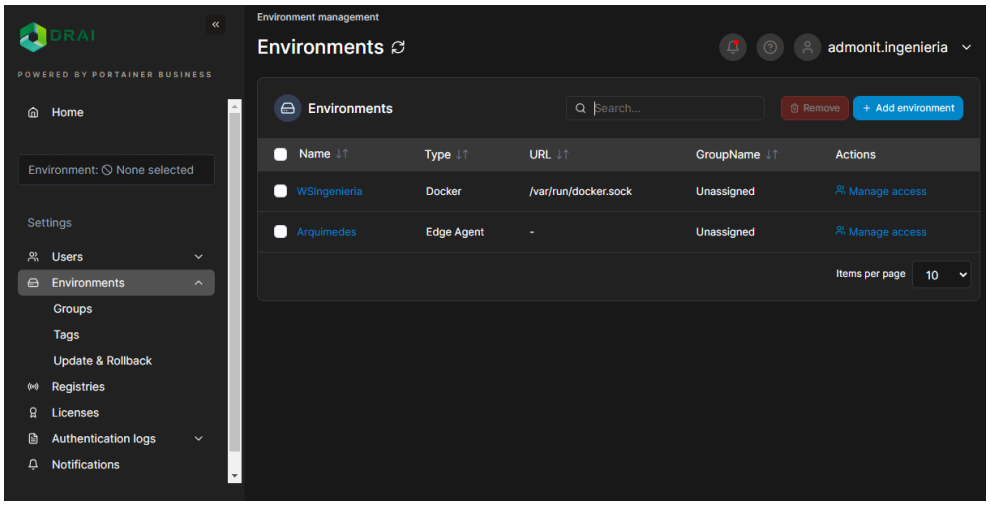


Figura 16
Integración del portainer y GitLab para lanzar stack

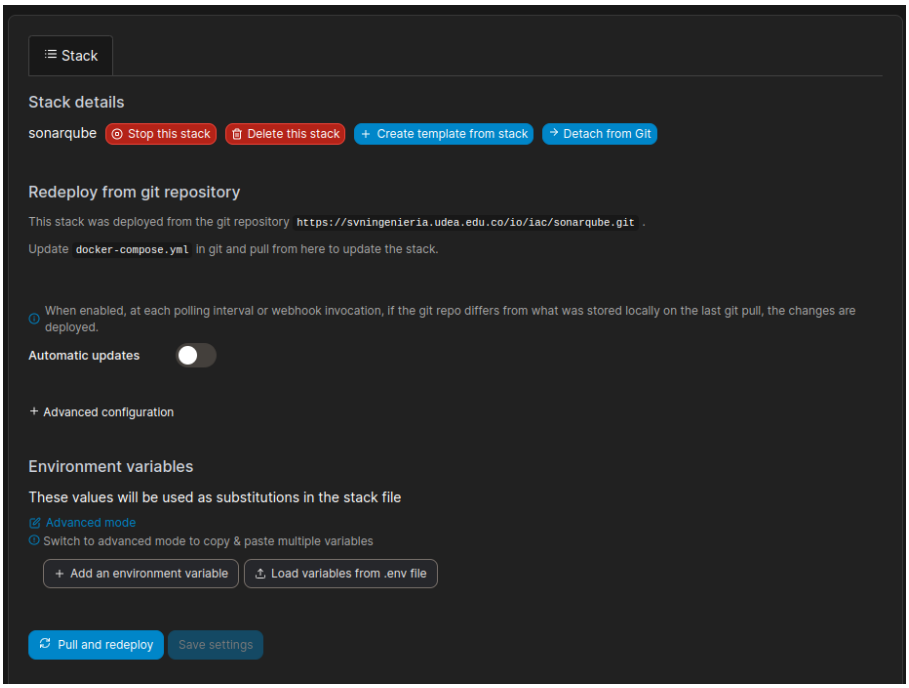


Figura 17
Control de acceso mediante labels

io.portainer.accesscontrol.teams	po
io.portainer.accesscontrol.users	alejandro.perez1, catalina.cespedes

Recapitulando, se ha logrado una implementación efectiva de Portainer en la infraestructura, para así administrar eficazmente los contenedores y mejorar significativamente el flujo de trabajo entre el equipo de desarrollo y de operaciones.

SonarQube

Como parte fundamental de las prácticas DevOps y con el objetivo de mantener la alta calidad del software, se ha integrado SonarQube en el ciclo de vida del software. Este paso es esencial, pues permite realizar un análisis estático del código fuente (SAST), una metodología que facilita la detección temprana de vulnerabilidades y errores antes de la implementación del software.

SonarQube proporciona una visión detallada de la deuda técnica, es decir, ayuda a identificar y cuantificar los problemas del código que, si no se resuelven a tiempo, podrían aumentar los costos y retrasar el desarrollo en el futuro. Además de las capacidades de análisis de código, también permite tener control sobre la cobertura del código a nivel de pruebas unitarias e integración. Esto significa que se puede asegurar que cada línea de código, cada función y cada módulo se sometan a pruebas rigurosas para garantizar su correcto funcionamiento. Gracias a esto, se puede construir software robusto y confiable, cumpliendo así con los altos estándares de calidad.

Puesto que se busca maximizar la eficiencia de la infraestructura, se decidió instalar SonarQube en un contenedor Docker. Esta decisión apoya el esfuerzo por mantener un entorno manejable y escalable. Haciendo uso de Portainer como la herramienta de stack, se logra un despliegue eficiente del contenedor ***Figura 18Figura 18***. Para mantener un registro ordenado y facilitar la gestión de las versiones de SonarQube, el archivo docker-compose en GitLab ***Figura 19***. Esta estrategia permite administrar y actualizar SonarQube de manera fácil y organizada, garantizando su disponibilidad y funcionamiento óptimo.

Figura 18

Página principal del SonarQube con acceso SSO (GitLab)

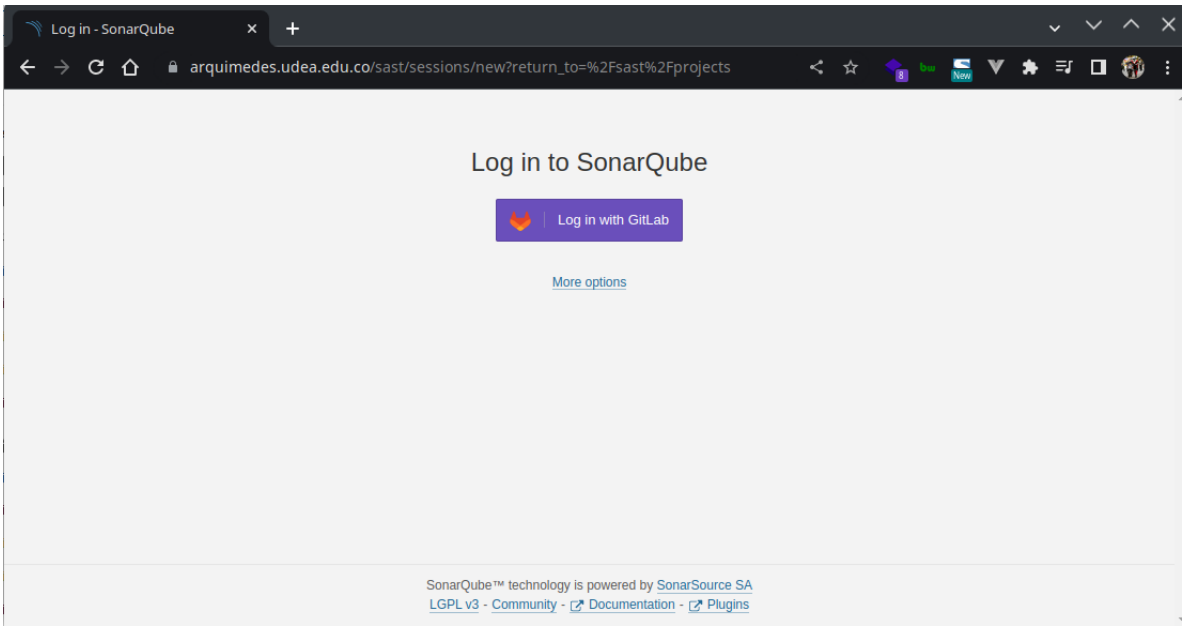
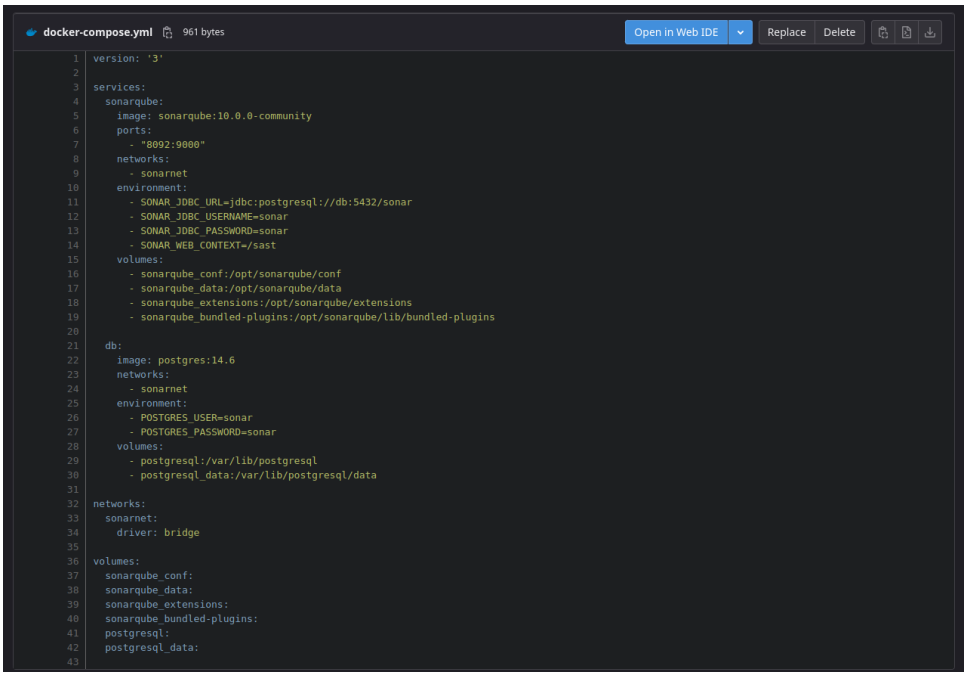


Figura 19

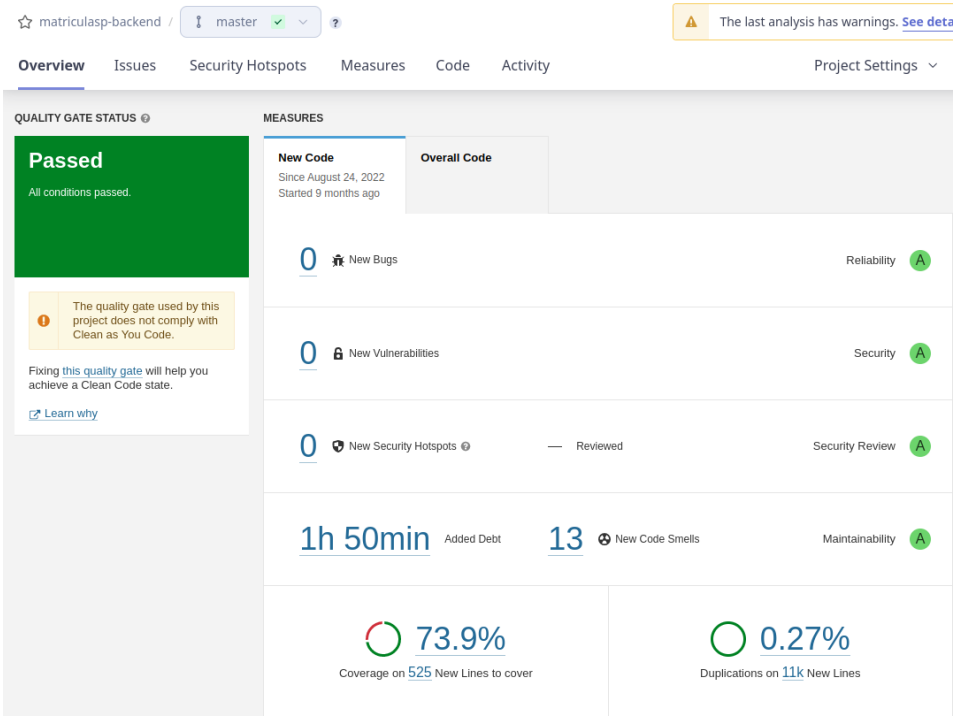
Docker Compose para la instalación de Sonar en portainer



Finalmente se configura SonarQube en proyectos reales, tal como se puede apreciar en la ***Figura 20*** donde se exhibe un análisis de código estático en un proyecto.

Figura 20

Resultado de pruebas SAST de un proyecto donde se implementó SonarQube



Implementación de CI/CD con prácticas DevOps

Una vez instalado y configurado GitLab, junto con otras herramientas clave que respaldan el ciclo de vida del software como SonarQube, OWASP ZAP, Portainer, se inició la tarea de implementar pipelines de CI/CD en todos los proyectos actuales en desarrollo y de forma progresiva, en las aplicaciones legacy de la Facultad de Ingeniería. La creación e implementación de estos pipelines se abordó con una visión centrada en la simplicidad y mantenibilidad, adoptando un enfoque agnóstico a la tecnología utilizada en cada proyecto. Para ello, se diseñó un sistema de plantillas que permitía abstraer los stages y los jobs necesarios para cada proyecto.

Este enfoque flexible y adaptable garantiza que el sistema de CI/CD puede aplicarse a cualquier tipo de proyecto, independientemente si es backend, frontend, microservicios o aplicaciones legacy o incluso si la infraestructura donde será desplegado es on-premise, k8s o AWS. Las plantillas no sólo simplifican el proceso de configuración de cada pipeline, sino que también fomentan la coherencia y la estandarización de los procesos de desarrollo y despliegue, lo que a su vez mejora la eficiencia y la efectividad de los esfuerzos de desarrollo de software. Se

puede apreciar cómo se organizó las plantillas ***Figura 21*** y cómo se vería un archivo `gitlab-ci.yml` configurado en un proyecto actual ***Figura 22 Figura 23***. Las plantillas están diseñadas para ser fácilmente comprensibles y mantenibles, lo que facilita su uso y adaptación por parte de los miembros del equipo.

Figura 21

Proyecto GitLab con los templates de CI/CD

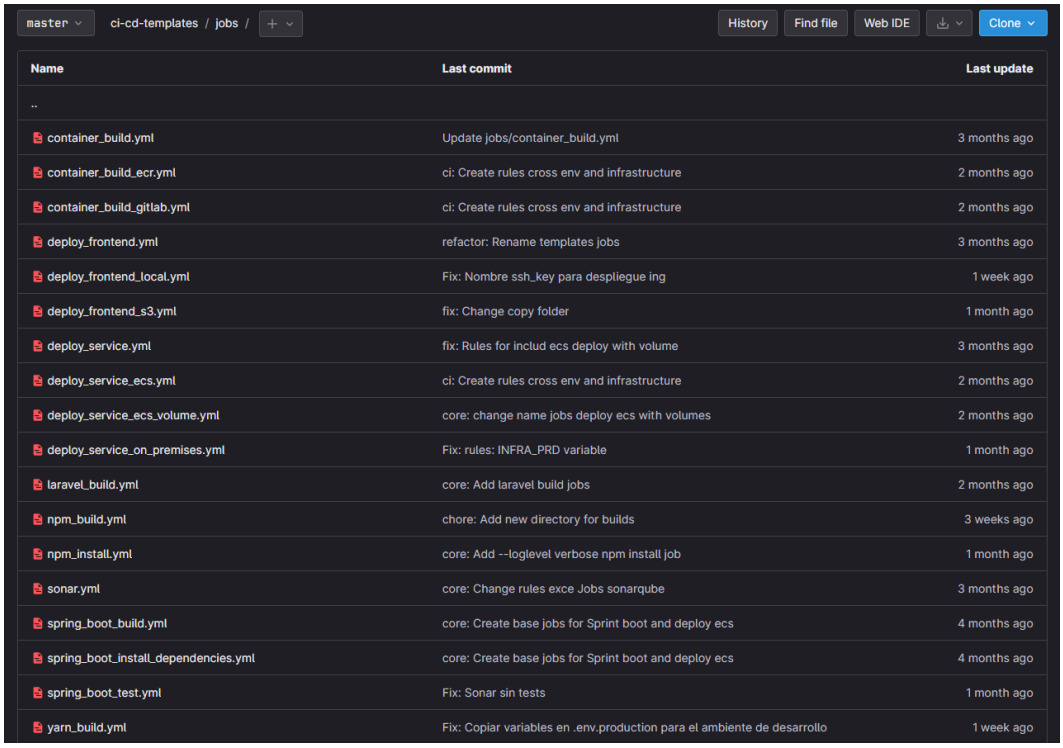
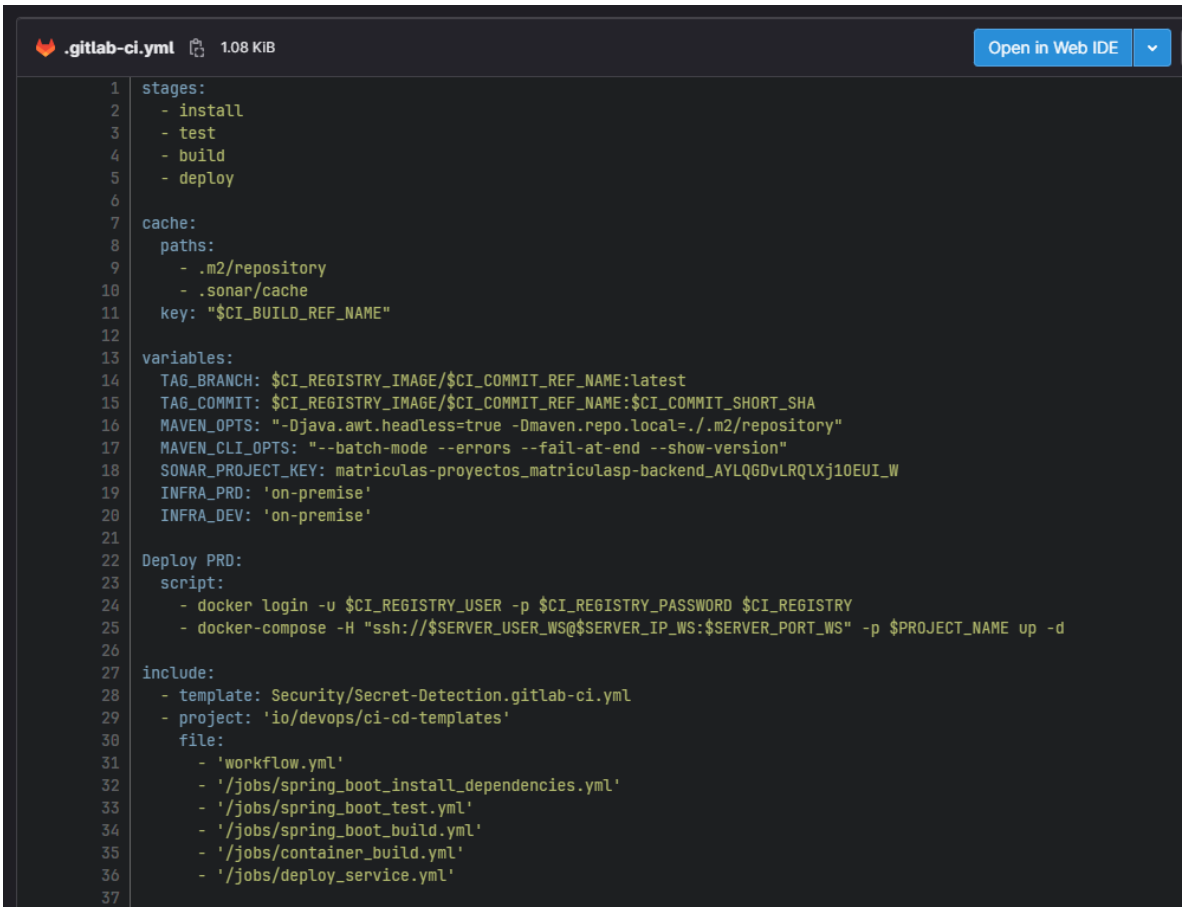


Figura 22

Pipeline de un proyecto backend usando el sistema de plantillas

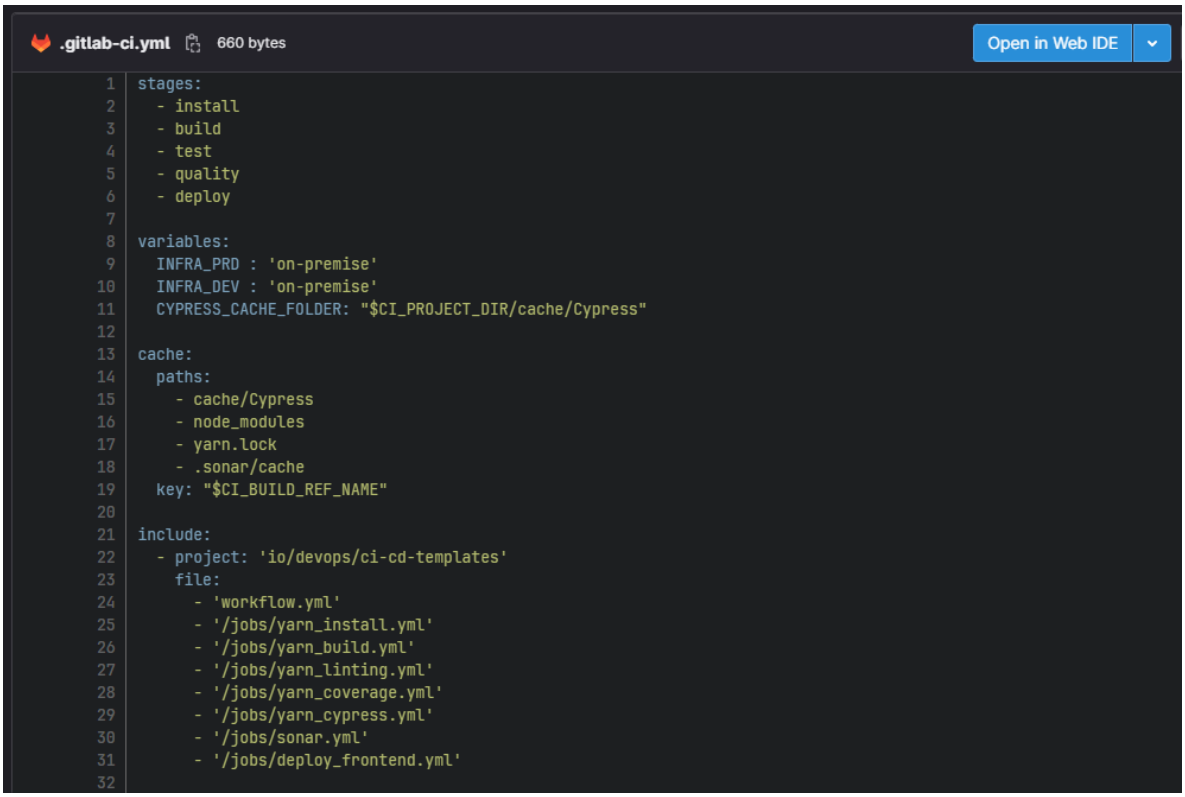


.gitlab-ci.yml 1.08 KiB — Open in Web IDE

```yaml
stages:
  - install
  - test
  - build
  - deploy

cache:
  paths:
    - .m2/repository
    - .sonar/cache
  key: "$CI_BUILD_REF_NAME"

variables:
  TAG_BRANCH: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:latest
  TAG_COMMIT: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:$CI_COMMIT_SHORT_SHA
  MAVEN_OPTS: "-Djava.awt.headless=true -Dmaven.repo.local=./.m2/repository"
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version"
  SONAR_PROJECT_KEY: matriculas-proyectos_matriculasp-backend_AYLQGDvLRQlXj1OEUI_W
  INFRA_PRD: 'on-premise'
  INFRA_DEV: 'on-premise'

Deploy PRD:
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker-compose -H "ssh://$SERVER_USER_WS@$SERVER_IP_WS:$SERVER_PORT_WS" -p $PROJECT_NAME up -d

include:
  - template: Security/Secret-Detection.gitlab-ci.yml
  - project: 'io/devops/ci-cd-templates'
    file:
      - 'workflow.yml'
      - '/jobs/spring_boot_install_dependencies.yml'
      - '/jobs/spring_boot_test.yml'
      - '/jobs/spring_boot_build.yml'
      - '/jobs/container_build.yml'
      - '/jobs/deploy_service.yml'
```

Figura 23

Pipeline de un proyecto frontend usando el sistema de plantillas



Ahora bien, al observar la infraestructura actual, se puede ver claramente cómo se ha transformado y evolucionado. Ahora cuenta con una serie de pipelines de CI/CD en funcionamiento, integrados de manera eficaz con las herramientas clave del ciclo de vida de software. Esta evolución ha sido un proceso gradual, con la implementación inicial de pipelines de CI/CD en los proyectos en desarrollo, seguida de la extensión gradual de esta implementación a las aplicaciones legacy. A lo largo de este proceso, se ha ido ajustando y refinando las plantillas y configuraciones para asegurar de que se adapten a las necesidades específicas de cada proyecto, al mismo tiempo que se mantiene una estructura y una coherencia generales en todos los pipelines de CI/CD.

Aprovechamiento de la infraestructura on-premise

Tras la instalación satisfactoria de GitLab y las demás herramientas para la práctica DevOps, se inicia con la instalación y configuración de una serie de herramientas adicionales que jugarán un papel importante en el proyecto. Que es la optimización de los recursos físicos, para

ello, se ha creado una prueba de concepto (PoC) de un clúster de Kubernetes que se ejecuta en los servidores on-premise de la Facultad de Ingeniería.

Para la implementación de Kubernetes se desarrolló de forma estratégica, utilizando un enfoque escalonado para asegurar una integración efectiva. Este proceso implicó el uso de Terraform para automatizar la creación de Máquinas Virtuales (VM) y la utilización de tres servidores físicos (III, IX y X) para alojar estas VM.

Preparación de los Servidores Físicos

Se configuraron tres servidores físicos, todos bajo el OS Red Hat Enterprise Linux (RHEL). Además de configurar el sistema operativo, se instalan las dependencias necesarias de la Máquina Virtual de Kernel (KVM) en cada servidor ***Figura 24***. La KVM es una solución de virtualización que permite crear múltiples servidores virtuales dentro de un único servidor físico. En términos más simples, proporciona el aumentar la capacidad de alojamiento y escalabilidad, al hacer que cada máquina virtual funcione como un servidor autónomo.

Para finalizar la fase de preparación, se instaló Terraform, una herramienta que proporciona la capacidad de tratar la infraestructura como código (IaC) ***Figura 25***. Esta tecnología posibilitó automatizar la creación y configuración de las máquinas virtuales, optimizando el proceso de despliegue.

Figura 24
KVM instalado en servidor físico con OS RHEL

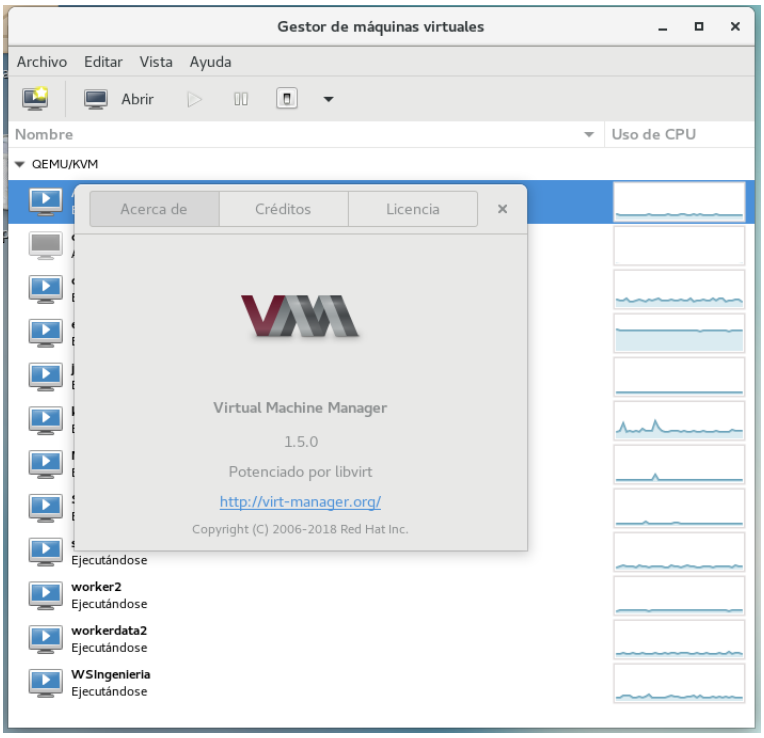
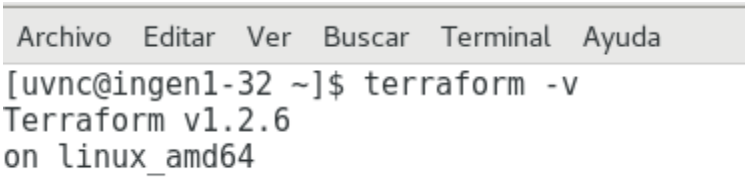


Figura 25
Terraform instalado en servidor físico

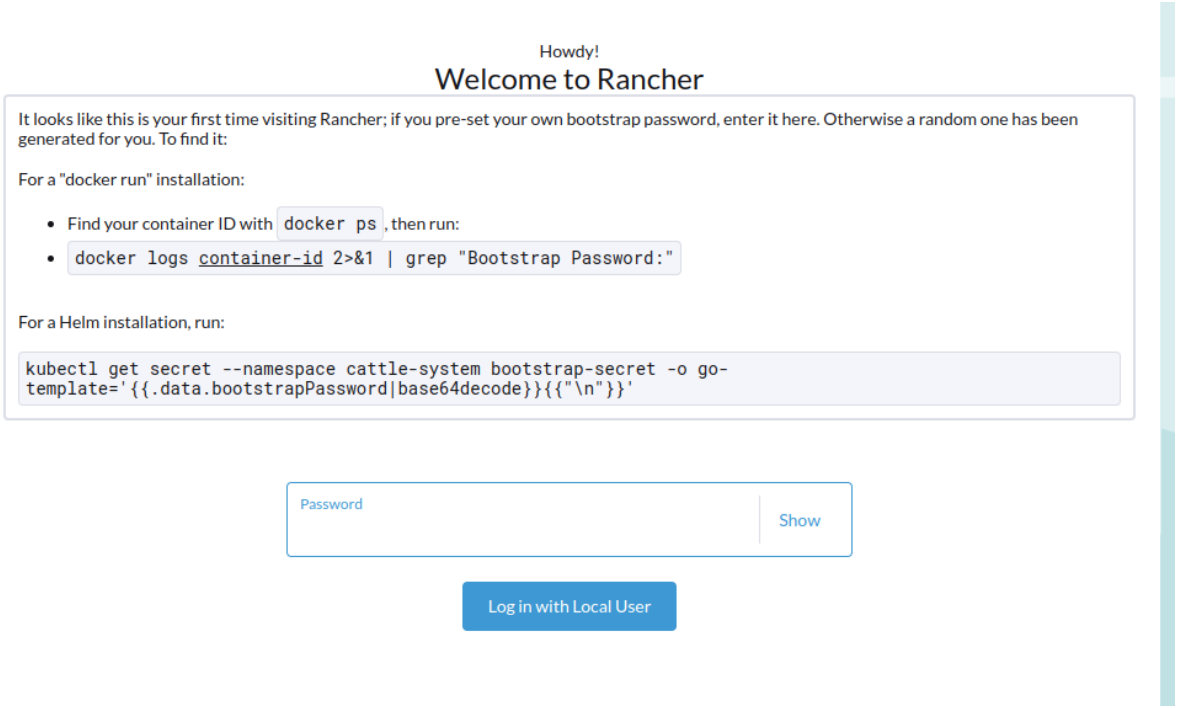


Instalación de Rancher

Para la creación del clúster de Kubernetes, es necesario tener un sistema que ayude a administrar el clúster, en este caso se eligió Rancher. El primer paso fue la ejecución del contenedor de Docker de Rancher en la VM que se designó para este propósito. Este paso fue bastante sencillo, ya que Docker permite un despliegue rápido y fácil con un solo comando.

Para conocer más del proceso de instalación se puede dirigir al **Anexo 2. Instalación del Rancher**

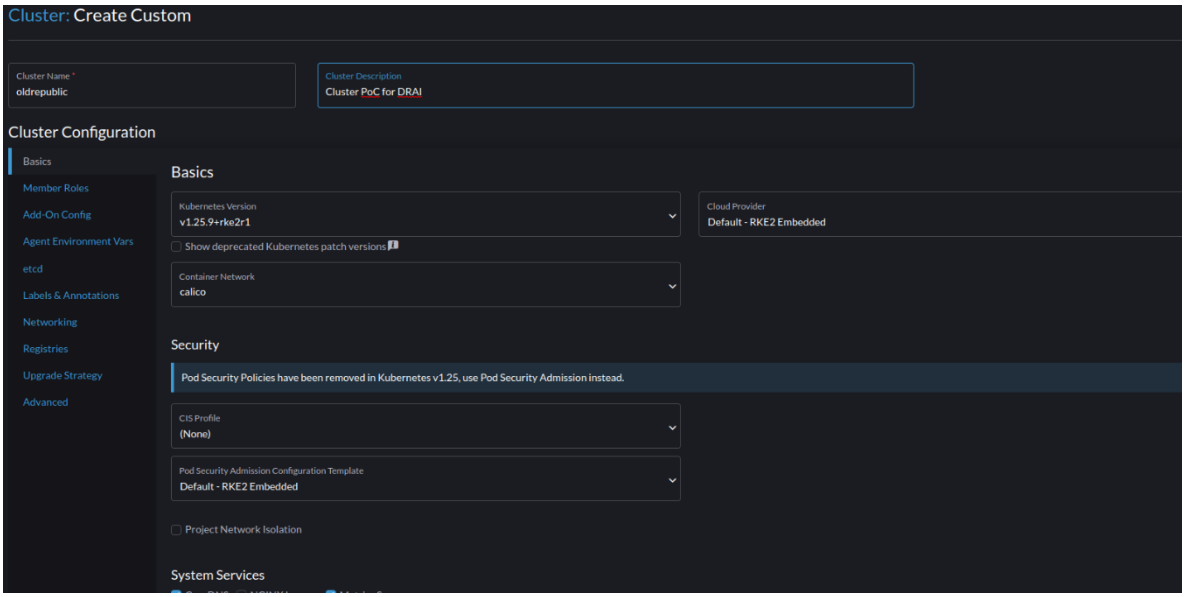
Figura 26
Rancher instalado satisfactoriamente



Tras la exitosa implementación de Rancher, el siguiente paso es la creación de un nuevo clúster. Para la Prueba de Concepto (PoC), se opta por un clúster de tipo personalizado con versión v1.25.9+k3s1 de Kubernetes, así pudiendo utilizar containerd como el motor de contenedores ***Figura 27***. Esta opción se eligió ya que se puede prescindir de la API de Docker, y se aprovecha un rendimiento superior y una menor sobrecarga.

Figura 27

Creación de nuevo clúster de Kubernetes



Una vez creado el clúster, se registran los nodos correspondientes en él. Este proceso, que podría parecer complejo, se simplificó considerablemente gracias a las funcionalidades de Rancher. Todo lo que se necesitaba hacer era ejecutar el comando correspondiente en cada nodo, acompañado de su respectiva bandera (flag). Esto se hizo para especificar el papel que cada nodo desempeñaría en el clúster como en la ***Figura 28***.

Figura 28

Registro de nodos en el Clúster creado

Dentro de la estrategia de automatización, se empleó Terraform para simplificar y agilizar el proceso de configuración de los nodos, con una estructura ajustada a la infraestructura como se ve en la ***Figura 29***. Esto eliminó la necesidad de entrar manualmente en cada máquina virtual después de ser aprovisionada, el plan de Terraform elaborado fue diseñado para aceptar variables específicas - el servidor, el token, el ca-checksum y el tipo de nodo - para configurar automáticamente cada nodo***Figura 30 Figura 30*** **y** ***Figura 31***. Además, también se incluyó una funcionalidad para seleccionar automáticamente una dirección IP de un pool predefinido para su asignación al nodo. Esto ayudo a reducir aún más la intervención manual y aumentar la eficiencia del proceso.

Figura 29
Proyecto GitLab con los archivos de Terraform

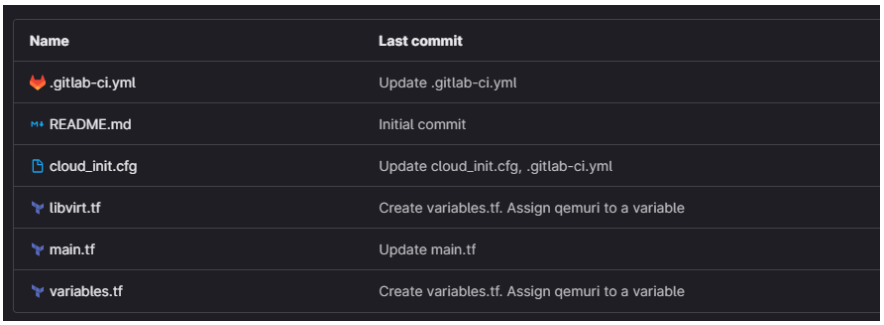


Figura 30
Inicio de Proyecto de Terraform

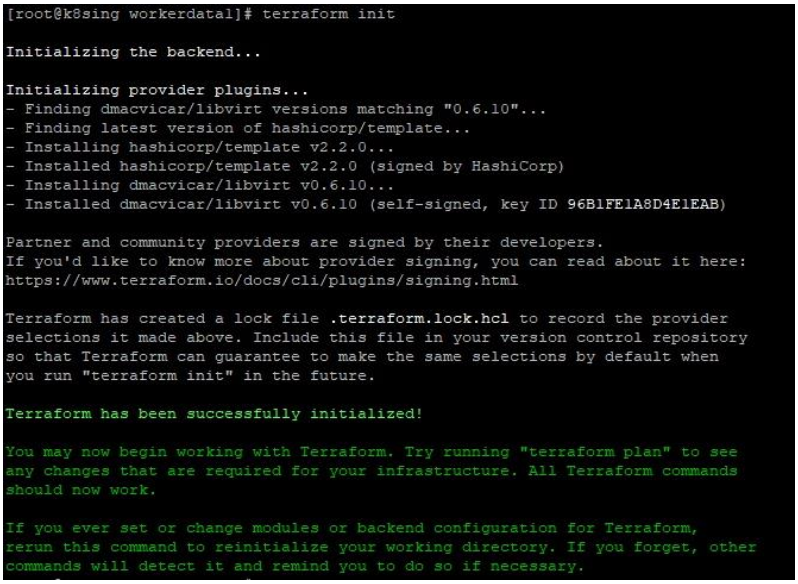
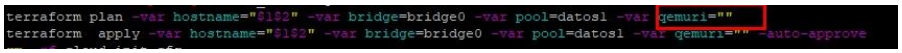


Figura 31
Creación de una VM con Terraform



Las siguientes figuras proporcionan una visión global de cómo queda configurado el clúster, ilustrando la disposición de los nodos y el despliegue de recursos.

Figura 32
Nodos registrados en el Cluster

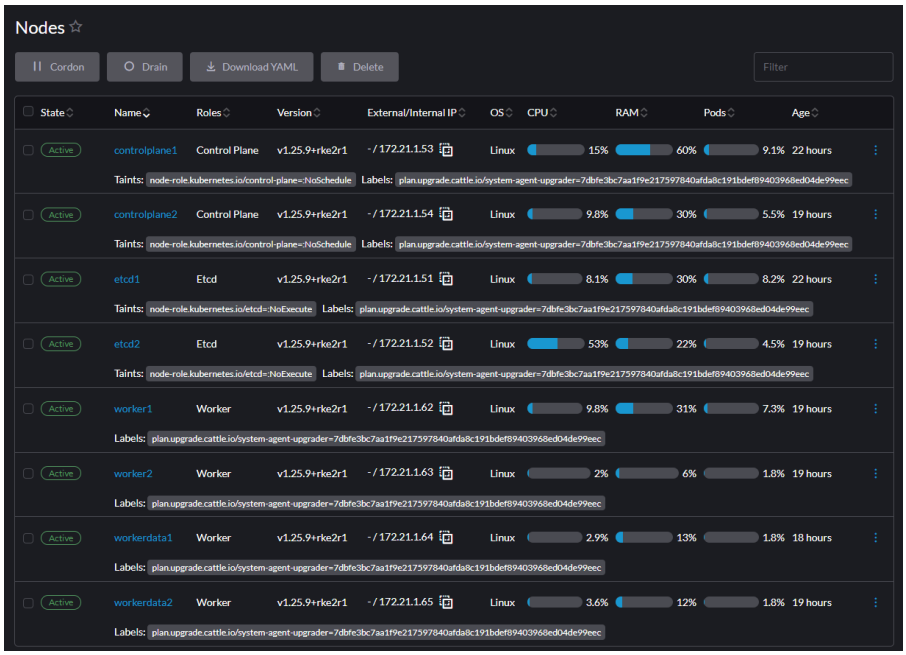
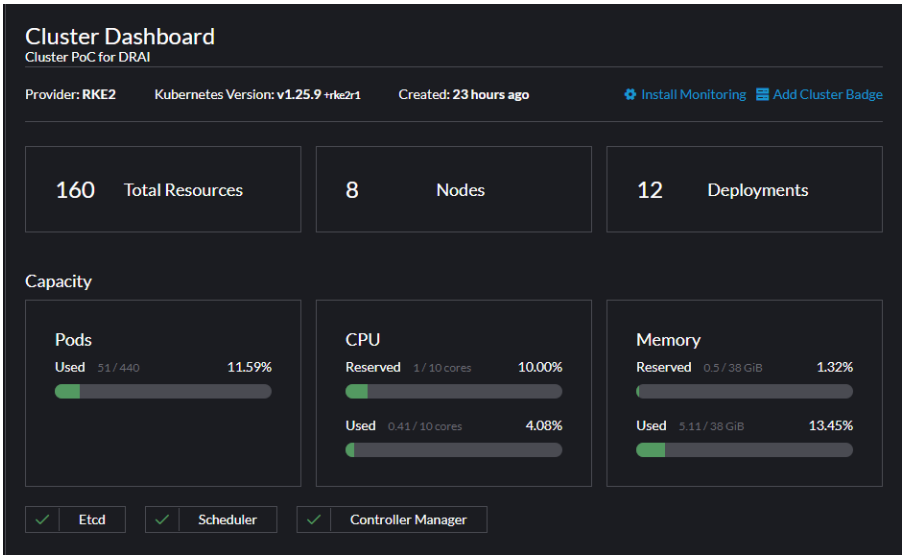


Figura 33
Características del Clúster



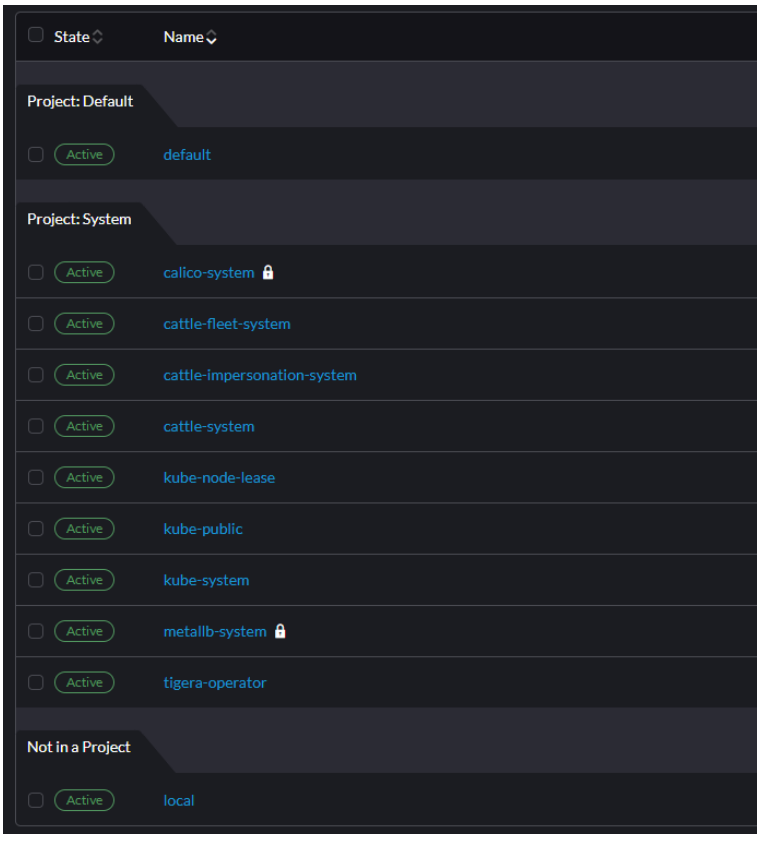
Después de establecer el clúster, se realizó la configuración para adaptar el clúster a las necesidades específicas. El primer paso en este sentido fue instalar un servicio de balanceador de carga. La decisión de seleccionar MetalLB en este punto se basó en el hecho de que el clúster se

encuentra en una infraestructura "on-premise". Antes de abordar las imágenes del resultado, vamos a detallar el proceso seguido para llegar a este punto.

La configuración de MetalLB se inició consultando y siguiendo las recomendaciones expuestas en la documentación oficial. Particularmente, utilizamos Kustomize para la creación del manifiesto de Kubernetes. Kustomize es una herramienta esencial que permite manejar manifiestos de Kubernetes de una forma más estructurada, lo que facilitó enormemente el proceso de configuración. Dada la imposibilidad de acceder a los routers de la universidad, se tuvo que optar por una configuración Layer 2 mode (ARP/NDP) para MetalLB.

Para finalizar la configuración de MetalLB, se creó un manifiesto de ConfigMap. En este manifiesto se asigna 4 direcciones IP para el balanceador de carga, con este manifiesto, informamos a MetalLB sobre qué direcciones IP puede utilizar para asignar a los servicios de tipo LoadBalancer, lo que permite optimizar el uso de los recursos de red disponibles.

Figura 34
Servicio MetalB en Namespace System



Después de instalar MetalLB como se vio en la ***Figura 34***, se continúa con la implementación de del ingress controller, en este caso, sería Traefik Kubernetes.

El primer componente a instalar fue el IngressRoute, que es la implementación de Traefik de un router HTTP para Kubernetes. Esto permite manejar de manera eficiente el tráfico HTTP que entra al clúster de Kubernetes proporcionando una alta disponibilidad.

Seguidamente, se configura el control de acceso basado en roles (RBAC) para este CRD. El uso del RBAC en Kubernetes permite controlar quién puede realizar acciones en determinados recursos y en qué circunstancias. Es una característica esencial que ayuda a mantener el clúster seguro y protegido. El siguiente paso fue la creación de un secreto que almacena los certificados para los dominios que el ingress va gestionar. El primer dominio configurado es *.udea.edu.co, este secreto asegura que el ingress pueda proporcionar una conexión segura y encriptada para todas las aplicaciones bajo este dominio.

Finalmente, se instaló el servicio y su respectivo despliegue para lanzar Traefik como IngressRoute ***Figura 35*** ***Figura 36***. Cabe mencionar que todos los pasos que se siguieron fueron tomados de la documentación oficial del proyecto Traefik (Traefik, 2023).

Figura 35

Servicio de ingress Traefik desplegado como LoadBalancer

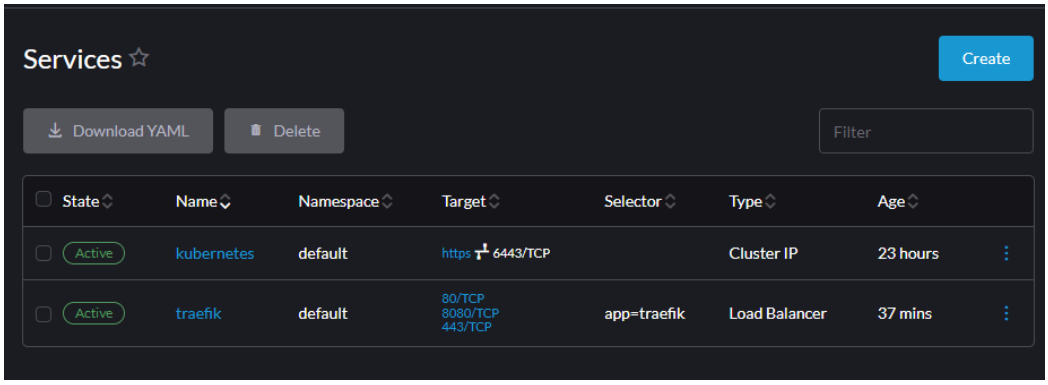
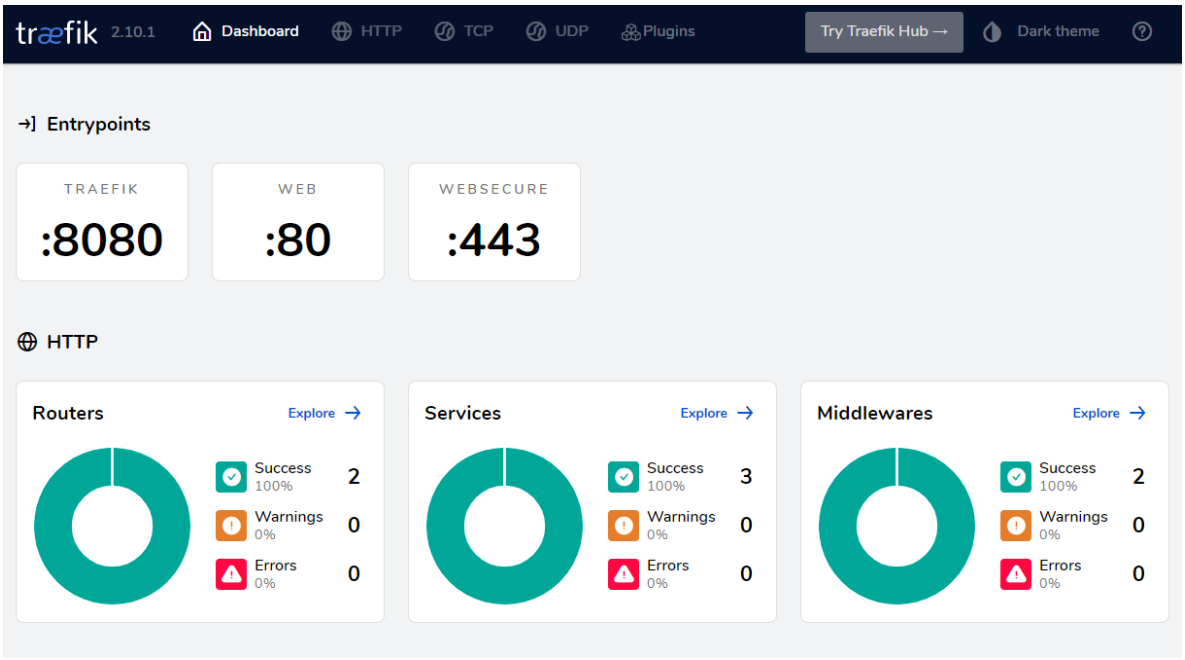


Figura 36
Dashboard del Traefik del Cluster

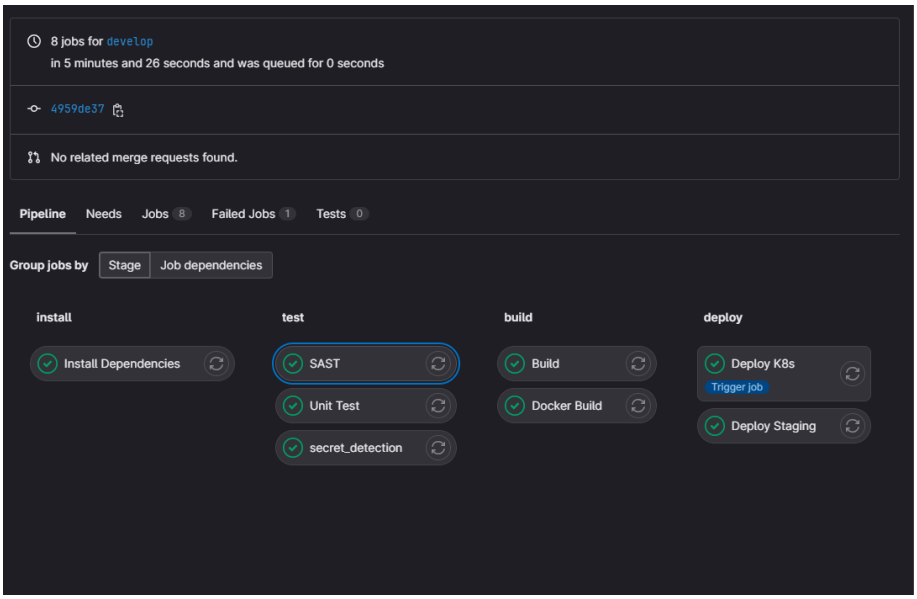


Una vez se tuvo el funcionamiento del clúster de Kubernetes, se da el siguiente paso, que fue su integración dentro de un pipeline de CI/CD de un proyecto actual. Este proyecto resultó ser una excelente oportunidad para llevar a cabo una prueba exhaustiva de la robustez y fiabilidad del nuevo clúster.

Para ello, se creó un pipeline que vemos en la ***Figura 37*** que operaba en paralelo tanto en la infraestructura actual como en la recién creada de Kubernetes. Con esta configuración se pudo comparar directamente el rendimiento y la eficacia de ambas infraestructuras bajo las mismas condiciones de carga de trabajo y demanda de recursos.

Este método de despliegue paralelo, además de proporcionar información valiosa sobre la capacidad del nuevo clúster, ofreció la posibilidad de realizar una transición suave y controlada hacia la nueva infraestructura. En lugar de cambiar de repente todos los servicios a esta, se puede hacer de manera gradual, asegurando en cada paso que los servicios funcionaran correctamente y que se mantuviera la continuidad del servicio para nuestros usuarios.

Figura 37
Pipeline Implementado en un proyecto real



Al final de este proyecto, el resultado ha sido una infraestructura robusta y eficiente que ha transformado la forma la que se desarrolla y despliega software en la Facultad de Ingeniería de la Universidad de Antioquia. Gracias a la implementación de tecnologías modernas como Kubernetes y a la automatización de los procesos a través de CI/CD, para el equipo de desarrollo, esto se traduce en una mayor agilidad y menor tiempo dedicado a tareas repetitivas y tediosas, permitiéndoles centrarse en lo que mejor saben hacer: escribir código de calidad. Para los clientes y usuarios, se traduce en una entrega de software más rápida y confiable, con menos errores y un tiempo de inactividad reducido. Comparado con el estado anterior, la diferencia es notoria. Antes, cada despliegue era una tarea laboriosa y llena de incertidumbre. Ahora, con las nuevas herramientas y procesos, los despliegues son rápidos, predecibles y seguros. Además, el uso de contenedores y orquestación ha permitido una mejor utilización de los recursos, reduciendo costos y aumentando la eficiencia. También se observó una disminución en los errores de despliegue y en el tiempo de inactividad del sistema, lo que contribuye a un mejor servicio al cliente y una mayor fiabilidad. Mirando hacia el futuro, el próximo objetivo es implementar un sistema de monitoreo robusto para la infraestructura, lo que permitirá detectar y solucionar problemas de manera más eficiente y proactiva. Además, buscar adoptar los informes DORA

(DevOps Research and Assessment), un estándar de la industria para evaluar el rendimiento de las operaciones de DevOps.

Estas son las cifras que hemos alcanzado desde la ejecución del proyecto. Con 451 proyectos en marcha, se apoya a una comunidad de 69 usuarios, incluyendo desarrolladores, arquitectos funcionales y propietarios de productos. Se ha gestionado más de 200 issues a través de GitLab. Además, se han realizado más de 300 solicitudes de merge de código, reflejando un flujo de trabajo activo y colaborativo. Como un hito particularmente destacado, se ha ejecutado más de 2500 pipelines como vemos en las estadísticas del GitLab en la Figura 38 y Figura 39, lo cual testimonia el volumen de trabajo y el compromiso con la integración y entrega continua que se ha llevado a cabo. Estas cifras hablan por sí solas, reflejando el valor tangible que este proyecto aporto a la Facultad de Ingeniería de la Universidad de Antioquia.

Figura 38
Estadísticas DevOps en GitLab

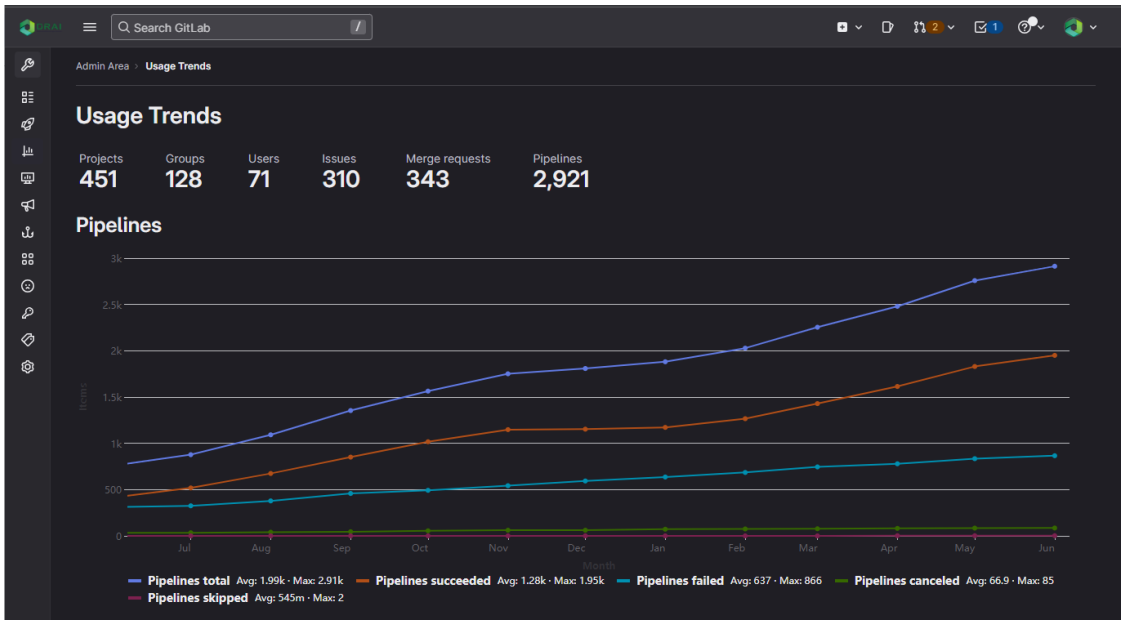
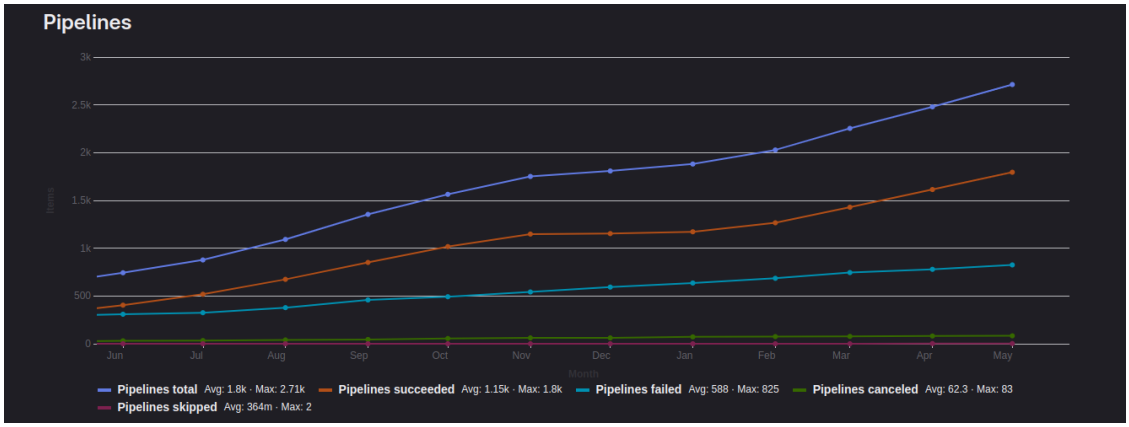


Figura 39

Pipelines ejecutas desde la implementación



Conclusiones

El proyecto planteado con el objetivo de implementar prácticas DevOps centradas en la optimización del datacenter de la Facultad de Ingeniería ha culminado exitosamente, y se ha traducido en avances tangibles hacia la mejora de la calidad del software, la reducción de los tiempos de despliegue de software, y la maximización de la productividad operativa.

Se ha realizado un análisis exhaustivo de la situación actual de la Facultad de Ingeniería en términos de desarrollo de software y gestión de la infraestructura. Este análisis ha permitido identificar áreas de mejora y desafíos existentes, proporcionando una base sólida para nuestras intervenciones futuras.

Herramientas y técnicas adecuadas para la automatización de la infraestructura han sido cuidadosamente seleccionadas y aplicadas, incluyendo el aprovisionamiento de servidores, la configuración de redes, y el despliegue de servicios. Este proceso ha desbloqueado eficiencias operativas, permitiendo a la Facultad de Ingeniería reducir el tiempo y el esfuerzo necesario para mantener y administrar su infraestructura de IT.

Además, se han implementado prácticas DevOps, promoviendo una cultura de colaboración entre los equipos de desarrollo y operaciones y fomentando la responsabilidad compartida en el ciclo de vida del software. Esto ha llevado a mejoras significativas en la eficiencia del proceso de desarrollo, así como a un aumento en la calidad del software producido.

El impacto de estas intervenciones ha sido medido y evaluado, considerando indicadores clave como el tiempo de entrega, la calidad del software, y la disponibilidad de los servicios. Los resultados han demostrado claramente que la implementación de DevOps y la automatización de la infraestructura han aumentado la eficiencia y la confiabilidad de los servicios de la Facultad de Ingeniería.

Finalmente, con base en los resultados obtenidos, se han propuesto recomendaciones y mejores prácticas para optimizar la implementación de una cultura DevOps y la automatización de la infraestructura. Estas recomendaciones ayudarán a la Facultad de Ingeniería a continuar impulsando mejoras en el desarrollo y la gestión de los servicios, asegurando que se puedan mantener y superar los logros obtenidos hasta ahora.

En conclusión, este proyecto ha logrado exitosamente sus objetivos, proporcionando a la Facultad de Ingeniería una hoja de ruta sólida para la mejora continua en el desarrollo de

software y la gestión de la infraestructura. Las lecciones aprendidas y los avances realizados aquí sentarán las bases para futuras innovaciones y mejoras, permitiendo a la Facultad de Ingeniería mantener su posición como líder en su campo.

Trabajo Futuro

Como resultado de este proyecto, se han obtenido importantes hallazgos y desarrollos, pero aún quedan numerosas oportunidades para explorar y mejorar aún más los procesos y sistemas existentes. En vista de estos resultados y oportunidades, se propone el siguiente trabajo futuro.

El primer aspecto que se identifica para el futuro trabajo es la implementación en producción del clúster de Kubernetes. Aunque en el proyecto actual se ha configurado y utilizado Kubernetes en un entorno de pruebas, su implementación en un entorno de producción plantea retos y oportunidades adicionales. Esto no solo permitiría a la organización aprovechar al máximo las ventajas de la orquestación de contenedores en un entorno real, sino que también proporcionaría valiosos aprendizajes para mejorar y optimizar futuras implementaciones.

En esta implementación en producción, se propone utilizar Rancher como administrador de Kubernetes, sin embargo, en lugar de ejecutar Rancher como un contenedor como se hace típicamente, se propone instalarlo usando Helm. Helm es un gestor de paquetes para Kubernetes que facilita la gestión y despliegue de aplicaciones en el clúster de Kubernetes. Este enfoque podría proporcionar más control y flexibilidad en la gestión del clúster, al tiempo que aprovecha las ventajas de Helm en términos de gestión de configuración y despliegue de aplicaciones.

Además, para mejorar aún más la gestión del almacenamiento en el clúster de Kubernetes, se propone agregar un sistema de archivos como Longhorn. Longhorn es una plataforma de almacenamiento distribuido y ligero para Kubernetes, que se centra en la simplicidad y la facilidad de uso. Con Longhorn, se puede manejar de manera eficiente la persistencia de los datos en el clúster de Kubernetes, permitiendo una mayor resiliencia y flexibilidad.

En un nivel más amplio, se identifica una oportunidad para crear un sistema para la gestión del área de Infraestructura y Operaciones (I&O). En el contexto actual, la gestión de I&O implica una amplia gama de tareas, muchas de las cuales podrían beneficiarse de una mayor automatización y estandarización.

Uno de los elementos clave de este sistema de gestión propuesto sería la automatización del ciclo de vida de las máquinas virtuales. Esto podría incluir la creación, modificación, monitoreo y eliminación de las VMs, lo que permitiría un mejor uso de los recursos, reduciría el

tiempo necesario para gestionar las VMs y mejoraría la consistencia y la fiabilidad de las operaciones.

Otro aspecto crítico sería facilitar la solicitud de entornos por parte de los desarrolladores. Actualmente, este proceso puede ser manual y propenso a errores, lo que puede llevar a retrasos y problemas de comunicación. Mediante la automatización y estandarización de este proceso, los desarrolladores podrían solicitar y obtener los entornos que necesitan de manera más rápida y fiable.

Finalmente, la gestión de los entornos efímeros, aquellos que se crean para una tarea específica y se eliminan cuando ya no son necesarios, es otra área que se beneficiaría de la automatización y una mejor gestión. Esto no sólo permitiría un uso más eficiente de los recursos, sino que también ayudaría a mantener un entorno de trabajo más limpio y organizado.

En resumen, el trabajo futuro se centra en llevar los resultados del proyecto actual al siguiente nivel, a través de la implementación en producción del clúster de Kubernetes, la mejora de la gestión del almacenamiento con Longhorn, y la creación de un sistema para la gestión del área de I&O. A través de estos esfuerzos, la organización podrá continuar mejorando su eficiencia y productividad, al tiempo que abre nuevas oportunidades para la innovación y la mejora continua.

Referencias

Abraham, S., Paul, A. K., Khan, R. I. S., & Butt, A. R. (2020). On the use of containers in high performance computing environments. *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*.

Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.

Barghoth, M. E., Salah, A., & Ismail, M. A. (2020). A comprehensive software project management framework. *J. Comput. Commun.*, *08*(03), 86-102.

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley Educational.

Beck, K., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S., van Bennekum, A., Hunt, A., Schwaber, K., Cockburn, A., Jeffries, R., Sutherland, J., Cunningham, W., Kern, J., Thomas, D., Fowler, M., & Marick, B. (2001). Manifesto for Agile Software Development. En *Agilemanifesto.org*. http://agilemanifesto.org/

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering*. O'Reilly Media.

Brynjolfsson, E., & Kahin, B. (Eds.). (2000). *Understanding the digital economy: Data, tools, and research*. The MIT Press.

CNCF (Ed.). (2023). Cloud Native Landscape. En *Cloud Native Landscape*. https://landscape.cncf.io/

Debab, R., & Hidouci, W. K. (2021). Containers Runtimes War: A Comparative Study. En *Advances in Intelligent Systems and Computing* (pp. 135-161). Springer International Publishing.

Docker Inc. (2023). Docker overview. En *Docker Documentation*. https://docs.docker.com/get-started/overview/

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Addison-Wesley Educational.

Espe, L., Jindal, A., Podolskiy, V., & Gerndt, M. (2020). Performance Evaluation of Container Runtimes. *Proceedings of the 10th International Conference on Cloud Computing and Services Science*.

Goldberg, R. P. (1974). Survey of virtual machine research. *Computer (Long Beach Calif.)*, *7*(6), 34-45.

Grand View Research. (2023). Virtual reality market size worth $87.0 billion by 2030. En *Grandviewresearch.com*. https://www.grandviewresearch.com/press-release/global-virtual-reality-vr-market

Hightower, K., Burns, B., & Beda, J. (2017). *Kubernetes—Up and Running*. O'Reilly Media.

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley Educational.

Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The devops handbook the devops handbook: How to create world-class agility, reliability, & security in technology organizations* (2.ª ed.). It Revolution Press.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., & Liguori, A. (s. f.). *kvm: The Linux Virtual Machine Monitor*. https://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2015). Dimensions of DevOps. En *Lecture Notes in Business Information Processing* (pp. 212-217). Springer International Publishing.

Manaseer, S., Manasir, W., Alshraideh, M., Hashish, N. A., & Adwan, O. (2015). Automatic test data generation for java card applications using genetic algorithm. *J. Softw. Eng. Appl.*, *08*(12), 603-616.

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.*, 239.

N. Vhatkar, K., & Bhole, G. P. (2022). Optimal container resource allocation in cloud architecture: A new hybrid model. *J. King Saud Univ. - Comput. Inf. Sci.*, *34*(5), 1906-1918.

Negus, C. (2020). *Linux Bible* (10.ª ed.). Standards Information Network.

Pfleeger, S. L., & Atlee, J. M. (2009). *Software engineering: Theory and practice: United States edition* (4.ª ed.). Pearson.

Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Commun. ACM*, *17*(7), 412-421.

Portianer, Inc. (s. f.). *Get started with Portainer* [Manual]. Get started with Portainer. https://install.portainer.io/

Puppet. (2021). *Stateof DevOps Report*.

RedHat, Inc. (2022). *El concepto de DevOps*. Redhat.com. https://www.redhat.com/es/topics/devops

Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques. *Proceedings of the 9th international conference on Software Engineering*, 328-338.

Scheepers, T. (2014). Virtualization and containerization of application infrastructure: A comparison. En *Thijs.ai*. https://thijs.ai/papers/scheepers-virtualization-containerization.pdf

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Prentice Hall.

Smith, J. E., & Nair, R. (2005). *Virtual machines: Versatile platforms for systems and processes*. Morgan Kaufmann Publishers.

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *Oper. Syst. Rev.*, *41*(3), 275-287.

Sommerville, I. (2005). *Ingenieria del software* (7.ª ed.). Pearson Educacion.

Sun, X., Ansari, N., & Wang, R. (2016). Optimizing resource utilization of a data center. *IEEE Commun. Surv. Tutor.*, *18*(4), 2822-2846.

Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4. ed). Prentice Hall.

The Linux Foundation. (2022). *Open Source Jobs Report*.

TIOBE index. (2021). En *TIOBE*. https://www.tiobe.com/tiobe-index/

Torvalds, L., & Diamond, D. (2002). *Just for fun: The story of an accidental revolutionary*. HarperBusiness.

Traefik. (2023). *Kubernetes IngressRoute & Traefik CRD - Traefik*. https://doc.traefik.io/traefik/providers/kubernetes-crd/

Turnbull, J. (2014). *The Docker book*. Lulu.com.

Vahldiek-Oberwagner, A., Elnikety, E., Duarte, N. O., Sammler, M., Druschel, P., & Garg, D. (2018). *ERIM: Secure, efficient in-process isolation with memory protection keys (MPK)*.

VMWare. (2023). Virtualization technology & virtual machine software: What is virtualization? En *VMware*. https://www.vmware.com/solutions/virtualization.html

W3techs.com. (2023). Usage statistics and market share of operating systems for websites, may 2023. En *W3techs.com*. https://w3techs.com/technologies/overview/operating_system

Anexos

Anexo 1. Instalación del GitLab

A continuación, se presentan los pasos básicos para la instalación de GitLab utilizando la distribución Omnibus:

Preparación del sistema

En primer lugar, actualizamos los paquetes existentes del sistema Debian 11 para garantizar que todas las dependencias estuvieran actualizadas.

```
sudo apt-get update
sudo apt-get install -y curl openssh-server ca-certificates perl
```

Dependencias necesarias

Aseguramos que el sistema contará con las dependencias necesarias para Postfix, que GitLab utilizará para el envío de correos electrónicos.

```
apt-get install -y postfix
```

Instalación de GitLab

Descargamos e instalamos el paquete de GitLab.

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-
ee/script.deb.sh | bash
EXTERNAL_URL="https://svningenieria.udea.edu.co" apt-get install gitlab-
ee
```

Busque el nombre de host e inicie sesión

En este caso que no se asigna una contraseña personalizada en la instalación, se generó una contraseña aleatoriamente y se almacenará durante 24 horas en */etc/gitlab/initial_root_password*

Para obtener información más detallada sobre la instalación de GitLab utilizando la distribución Omnibus, recomiendo remitirse a la documentación oficial de GitLab. En su documentación, encontrarás guías paso a paso, tutoriales y referencias técnicas que te ayudarán a realizar una instalación exitosa y configurar GitLab según tus necesidades específicas.

Puedes acceder a la documentación de GitLab en el siguiente enlace: **https://docs.gitlab.com/**

Dentro de la documentación, busca la sección relacionada con la instalación de GitLab utilizando la distribución Omnibus. Allí encontrarás instrucciones detalladas, requisitos del sistema, opciones de configuración y recomendaciones para llevar a cabo una instalación exitosa. Además, la documentación de GitLab también ofrece información adicional sobre el uso de GitLab, gestión de proyectos, integración con otras herramientas y características avanzadas. Explora la documentación para obtener una visión completa de todas las capacidades que GitLab ofrece.

Recuerda que la documentación oficial es una excelente fuente de información confiable y actualizada, proporcionada por los desarrolladores de GitLab, y te brindará una guía sólida para llevar a cabo la instalación de GitLab de manera efectiva.

En el proyecto se instala algunas características adicionales que su documentación se encuentra en el portal de docs.gitlab.com

Para la instalación de los Runners en GitLab, se puede seguir la documentación oficial que proporciona instrucciones detalladas. La documentación se encuentra en el siguiente enlace: **https://docs.gitlab.com/runner/install/**

Para la instalación del Container Registry en GitLab, se puede seguir la documentación oficial que proporciona instrucciones detalladas. La documentación se encuentra en el siguiente enlace: **https://docs.gitlab.com/ee/administration/packages/container_registry.html**

Para la instalación del soporte de autenticación mediante OAuth 2.0 en GitLab, se puede seguir la documentación oficial proporcionada en **https://docs.gitlab.com/ee/integration/oauth_provider.html**

Anexo 2. Instalación del Rancher

Para la instalación del Rancher en versión Docker, se requieren las siguientes configuraciones:

Instalación de Docker: En el nodo donde deseas instalar Rancher, se recomienda utilizar una distribución de Linux como Rancher OS, Ubuntu o Debian. En este caso, se utilizará Debian como base. Puedes instalar Docker ejecutando el siguiente comando:

```
curl https://releases.rancher.com/install-docker/20.10.sh | sh
```

Este comando descargará e instalará Docker en el sistema. Si deseas seguir las mejores prácticas de instalación de Docker, puedes consultar la documentación oficial en https://docs.docker.com/install/.

Configuración de sysctl: Es necesario configurar el parámetro net.bridge.bridge-nf-call-iptables en cada nodo. Esta configuración permite el enrutamiento de paquetes entre interfaces de red. Puedes realizar esta configuración ejecutando el siguiente comando:

```
sysctl net.bridge.bridge-nf-call-iptables=1
```

Esto asegurará que el tráfico de red se enrutará correctamente dentro del clúster de Kubernetes.

Configuración de cada nodo: Ahora para cada nodo es necesario hacer las siguientes configuraciones.

Configuración del tráfico de red con iptables: Puedes utilizar la siguiente configuración para permitir el tráfico de red en el puerto 6443, que es utilizado por Rancher:

```
sudo iptables -A INPUT -p tcp --dport 6443 -j ACCEPT
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
sudo iptables -F
```

Sin embargo, se recomienda revisar y utilizar los puertos definidos en la documentación de Rancher para un mejor control de acceso. Puedes consultar la documentación en https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/installation-requirements/port-requirements

Configuración del servicio de sshd: Para habilitar la conexión TCP Forwarding en el servicio sshd, descomenta la línea "AllowTcpForwarding yes" en el archivo */etc/ssh/sshd_config*. Debe quedar de la siguiente manera:

```
# Elimina el comentario la siguiente línea
AllowTcpForwarding yes
```

Configuración de los cgroups: Es necesario configurar los cgroups para el uso de Docker o containerd. Puedes realizar esta configuración ejecutando los siguientes comandos:

```
sudo sysctl -w net/netfilter/nf_conntrack_max=131072
sudo sysctl net/netfilter/nf_conntrack_max
```

Esto establecerá el valor máximo de nf_conntrack en 131072.

Al realizar estas configuraciones en cada nodo del clúster de Kubernetes

Para obtener mayor detalle y una guía paso a paso, te recomiendo consultar la documentación oficial de Rancher. La documentación proporciona información detallada sobre la instalación de Rancher en sus diferentes versiones, incluyendo configuraciones adicionales y requisitos específicos.

Puedes acceder a la documentación oficial de Rancher en el siguiente enlace: https://rancher.com/docs/