Despliegue De Plataforma De Contenerización En GCP Mediante Infraestructura Y Configuración Como Código Para Prácticas De DevOps

Jacob Rodríguez Duque

Sergio Estiven Chica Gallego

Diego Esteban Gaviria

Asesor

Jhon Fernando Sánchez Álvarez

Universidad nacional abierta y a distancia – UNAD

Escuela de ciencias básicas, tecnología e ingeniería ECBTI

Ingeniería de Sistemas

2024

Página de Aceptación

_____

Director Trabajo de Grado

_____ _____

Jurado Jurado

Medellín - 2024

Resumen

Este trabajo presenta el diseño y la implementación de una plataforma de integración continua y despliegue continuo (CI/CD) en Google Cloud Platform (GCP), utilizando herramientas como Terraform, Ansible y Jenkins. El proyecto, llevado a cabo por un equipo compuesto por dos estudiantes de ingeniería de sistemas y uno de ingeniería de telecomunicaciones, tiene como objetivo desarrollar un sistema que automatice la construcción, prueba y despliegue de aplicaciones en la nube de manera eficiente y escalable.

Durante el desarrollo del proyecto, se configuró la infraestructura en GCP utilizando Terraform, lo que permitió definir y gestionar los recursos necesarios de manera eficaz. Además, se implementaron scripts de Ansible para automatizar tareas de configuración y despliegue en los servidores, asegurando una gestión consistente y reproducible del entorno.

La integración continua se logró a través de Jenkins, que se desplegó como un contenedor en GCP, permitiendo la ejecución automática de pipelines de integración y despliegue. Esta configuración permitió a los desarrolladores validar rápidamente los cambios en el código y desplegar nuevas versiones de la aplicación de manera automatizada.

La colaboración interdisciplinaria entre los miembros del equipo fue fundamental para abordar los desafíos técnicos y operativos del proyecto. La combinación de habilidades en ingeniería de sistemas y telecomunicaciones permitió un enfoque integral en el diseño y la implementación de la plataforma de CI/CD.

Abstract

This thesis presents the design and implementation of a continuous integration and continuous deployment (CI/CD) platform on the Google Cloud Platform (GCP) using tools such as Terraform, Ansible, and Jenkins. The project, carried out by a team of two systems engineering students and one telecommunications engineering student, aims to develop a system that efficiently and scalably automates the build, test, and deployment of applications in the cloud.

During the project development, Terraform was used to configure GCP's Infrastructure, allowing for the effective definition and management of necessary resources. Additionally, Ansible scripts were implemented to automate server configuration and deployment tasks, ensuring consistent and reproducible environment management.

Continuous integration was achieved through Jenkins, deployed as a container on GCP, enabling automatic execution of integration and deployment pipelines. This powerful automation configuration allowed developers to quickly validate code changes and deploy new versions of the application automatically, providing a seamless and convenient experience.

Interdisciplinary collaboration among team members was not just a necessity, but a key strength in addressing technical and operational challenges of the project. The combination of skills in systems engineering and telecommunications allowed for a comprehensive approach in designing and implementing the CI/CD platform, highlighting the importance of teamwork in such complex projects.

Tabla de Contenido

Lista de Figuras

Introducción

Las prácticas de CI/CD son esenciales para los equipos tanto de infraestructura como de desarrollo dentro del área de la informática aplicada a la nube. Estas prácticas posibilitan la automatización y rapidez del proceso de instalación, pruebas y despliegue de herramientas, asegurando que los cambios en la infraestructura y las aplicaciones se realicen de manera segura y rápida. En una circunstancia de portabilidad y exigencias dinámicas, donde la escalabilidad, la disponibilidad y la seguridad son esenciales, la capacidad de iterar y cambiar rápidamente la infraestructura y las aplicaciones se hace fundamental el conservar la operatividad y la competencia.

La utilización de CI/CD genera una serie de bondades fundamentales para las áreas de TI. En primer lugar, disminuyen significativamente la dedicación de los equipos de trabajo y los recursos utilizados para gestionar modificaciones en la infraestructura y las aplicaciones. En lugar de realizar acciones manuales como el aprovisionamiento, disponibilización y reconfiguración de los recursos de infraestructura, además de la creación, pruebas y simulacros de estabilidad de la infraestructura, los equipos de TI pueden centrarse en perfeccionar la arquitectura, optimizar recursos y la calidad de configuraciones sobre los mismos, en vez de ejecutar acciones repetitivas y reactivas en la infraestructura.

Además, la actualización constante de código en los equipos propicia una cultura de colaboración y retroalimentación inmediata dentro de estos. Permitir la incorporación

periódica de actualizaciones en la infraestructura y las aplicaciones y ejecutar de manera automática las pruebas de validación, genera una simplificación de la identificación de errores y dificultades en la configuración, dando una respuesta preventiva a situaciones de pérdidas de servicio, las cuales permiten a las personas que corrigen las dificultades contar con una reacción rápida y efectiva.

En el ambiente actual de uso de las nubes para el despliegue de infraestructura y uso de la misma en implementación de aplicaciones, donde los parámetros de complejidad y cambio exigen confiabilidad y rapidez como características comunes, la automatización de CI/CD se torna todavía más importante. Sin embargo, esto a su vez también ha generado nuevos problemas en el ámbito de la administración y el control.

El proyecto, hecho por un grupo de estudiantes de los cuales dos terminan la carrera de ingeniería de sistemas y uno ingeniería de telecomunicaciones, aplican dicha colaboración entre diversas áreas de la informática para abordar los problemas del proyecto desde diferentes puntos de vista técnicos, logrando desde un enfoque eficaz el aprovisionamiento y disponibilización de una plataforma de contenerización que permita a equipos de desarrollo contar con ambientes apropiados para desempeñar sus funciones, dichos ambientes con posibilidad de ser altamente escalables y replicables en tiempos óptimos como el mercado de hoy lo requiere, de esta forma se garantiza la premisa de agilidad y estabilidad de los recursos antes mencionados.

Estructura Del Documento

Introducción:

- Breve presentación del proyecto y su importancia en el contexto actual.
- Descripción de los objetivos del proyecto y la audiencia a la que se dirige.

Estructura del Documento:

- Breve descripción de cómo está organizado el documento y qué secciones y contenido pueden esperar los lectores.

Estado del arte:

- Proporciona un análisis exhaustivo de las investigaciones y desarrollos previos relacionados con el proyecto.

Objetivos del Proyecto:

- Se describe claramente lo que se espera lograr con el proyecto.

Contexto y Justificación del Proyecto:

- Descripción detallada del proyecto, incluyendo su enfoque en la implementación de una plataforma de contenerización en Google Cloud Platform (GCP).
- Explicación de los principios de Infraestructura como Código (IaC) y Configuración como Código (CaC) utilizados en el proyecto.
- Identificación de la necesidad o problema que motiva el proyecto.
- Explicación de por qué es importante implementar una plataforma de contenerización en GCP y cómo esto puede beneficiar a las organizaciones.

Desafíos:

- Se discuten las dificultades y obstáculos que enfrentaremos durante el desarrollo del proyecto.

Alcance del Proyecto:

- Descripción detallada de las tres capas de trabajo del proyecto: infraestructura, disponibilidad y aplicación.
- Explicación de los objetivos y límites del proyecto.

Metodología y Enfoque:

- Descripción de la metodología utilizada para llevar a cabo el proyecto.
- Explicación del enfoque basado en IaC y CaC, y cómo se aplicará en cada una de las capas de trabajo.

Desarrollo del proyecto:

- Sección donde se detallan los pasos y procesos que se llevarán a cabo durante el proyecto.

Beneficios Esperados:

- Resal en GCP puede traer a las organizaciones.
- Enumeración de los beneficios de GKE y Jenkins en este contexto.

Público Objetivo:

- Descripción del público objetivo del proyecto, incluyendo empresas interesadas en optimizar sus procesos de desarrollo e implementación de aplicaciones en la nube.

Diagramas UML:

- Se encontrarán cada uno de los diagramas UML (Unified Modeling Language) que son esenciales para la documentación técnica del proyecto.

Conclusiones:

- Recapitulación de los puntos clave del proyecto.
- Reflexión sobre los resultados esperados y la importancia del proyecto en el contexto actual.

Referencias:

- Lista de todas las fuentes y referencias utilizadas en el documento.

Estado Del Arte

En el contexto de cualquier intento de investigación, es importante dar una explicación fundamental de la información existente y de las líneas tecnológicas actuales en la especialidad. El proyecto en cuestión se preocupa por poner en marcha un sistema en contenerización que use las prácticas de DevOps para implementar una plataforma en el marco de Google Cloud Platform (GCP). Entender la situación presente de los métodos y herramientas relevantes y su ambiente es fundamental para el éxito del proyecto.

La contenerización consiste en una novedosa tecnología que tiene como objetivo principal revolucionar la manera en la que se desarrollan y se implementan las aplicaciones (IBM, 2024). Debido a que es una tecnología que permite generar contenedores de manera gratuita, Docker se convirtió en la herramienta líder de contenerización ya que por su capacidad para agrupar los componentes de una aplicación y sus necesidades en contenedores que son livianos y portables (Docker Inc, 2024). Este incremento facilito a los programadores la creación, distribución y ejecución de aplicaciones en diferentes ambientes, disminuyendo significativamente las dificultades de configuración y armonía.

Al masificarse el uso de contenedores, surgió la necesidad de centralizar la administración de grandes cantidades de ellos de manera eficaz y uniforme. Es en ese momento en donde surge Kubernetes, un sistema de administración de contenedores que está disponible para el público. Kubernetes hace que la operación, el crecimiento y la administración de las aplicaciones que se ejecutan en contenedores sea controlado, así como la distribución de recursos, la equidad de cargas y la posibilidad de control de errores

sean manejables de forma armónica (The Linux Foundation, 2024). Esta habilidad de gestionar recursos a gran escala hizo que Kubernetes se convirtiera en la herramienta objetivo de las empresas que tienen ambientes robustos de contenerización en servidores de la Nube.

Mientras se desarrollaba la tecnología para contenerizar y orquestar, se hizo evidente la necesidad de gestionar el origen del software y de automatizar ciclo de vida del desarrollo. Aquí es donde las herramientas de administración de códigos fuente (SCM) (Software Freedom Conservancy, 2024) como GitLab y las herramientas de integracion y entrega continua (CI/CD) como Jenkins toman importancia.

- GitLab: Brinda una solución combinada que mezcla la administración del fuente Code con la capacidad de CI/CD. Facilita la labor de los programadores en la contribución de los códigos, la realización de comprobaciones y la automatización de pruebas y despliegues.
- Jenkins: Es una herramienta de CI/CD que posibilita la automatización de la integración y entrega periódica (Jenkins, 2024). Jenkins se une con diversas herramientas y métodos, usando Docker y Kubernetes, para conseguir una labor de trabajo automatizada que aumenta la efectividad y calidad del software.

Para obtener el máximo provecho de las tecnologías antes mencionadas, es fundamental aplicar enfoques tecnológicos actualizados como la infraestructura como código (IaC) y la configuración como código (CaC). Estas estrategias posibilitan delimitar y gestionar los componentes de la infraestructura y las configuraciones de manera ágil y repetible.

- **Terraform**: Es una herramienta de IaC que pone a disposición de los usuarios una forma de describir la infraestructura en forma de archivos de configuración que se pueden actualizar y versionar, esto con el fin de facilitar la creación y administración de recursos en GCP de manera más eficaz (HashiCorp Developer, 2024d).
- **Ansible**: Es herramienta que se enfoca en realizar CaC automatizando la configuración y administración de servidores, posibilitando la utilización de configuraciones de manera constante y repetible (Ansible Collaborative, 2024).

En este orden, GCP ofrece una suma de servicios diferentes que complementan los anteriormente mencionados, los cuales facilitan la administración y ejecución de aplicaciones en la Nube, brindando una infraestructura que puede ser escalable, ágil y segura. Dentro de los más importantes servicios se hallan:

- **Google Kubernetes Engine (GKE):** Un servicio que posibilita la ejecución y administración de aplicaciones en contenedores usando Kubernetes (Google Cloud, 2024e).
- **Compute Engine**: Proporciona máquinas virtuales que pueden ser modificadas en función de las necesidades del proyecto (Google Cloud, 2024d).
- **Cloud Storage:** Ofrecen un alojamiento con duración prolongada y de gran disponibilidad para información no planificada (Google Cloud, 2024a).

Terraform

Terraform es una herramienta de código abierto desarrollada por HashiCorp que permite la creación, gestión y versionado de la infraestructura como código (IaC). Utilizando la sintaxis declarativa de HashiCorp Configuration Language (HCL), Terraform facilita la automatización del aprovisionamiento de recursos en diferentes proveedores de nube, como Google Cloud Platform (GCP), AWS, Azure, entre otros (HashiCorp Developer, 2024c)

Ansible

Ansible, también desarrollado por Red Hat, es una plataforma de automatización que simplifica las tareas de configuración, gestión y despliegue de sistemas y aplicaciones. Utilizando un enfoque basado en YAML y SSH, Ansible permite la automatización de procesos de forma sencilla y escalable, lo que lo convierte en una herramienta ideal para la gestión de configuraciones (CaC) y la automatización de la infraestructura (Ansible Collaborative, 2024)

Google Cloud Platform (GCP)

Google Cloud Platform (GCP) es una suite de servicios en la nube ofrecida por Google que proporciona una infraestructura escalable y confiable para la ejecución de aplicaciones y servicios en la nube. Con una amplia gama de servicios, incluyendo cómputo, almacenamiento, bases de datos, inteligencia artificial y machine learning, GCP se ha convertido en una opción popular para el desarrollo y despliegue de aplicaciones en la nube (Google Cloud, 2024c)

Google Kubernetes Engine (GKE)

Google Kubernetes Engine (GKE) es un servicio gestionado de Kubernetes ofrecido por Google Cloud Platform que facilita la implementación, administración y escalado de aplicaciones en contenedores. Utilizando Kubernetes como orquestador de contenedores, GKE proporciona una plataforma robusta y confiable para la ejecución de cargas de trabajo en contenedores Docker (Google Cloud, 2024e)

Kubernetes

Kubernetes es un proyecto de código abierto desarrollado por Google que proporciona una plataforma para la gestión de contenedores a escala. Con características como el aprovisionamiento automático, el escalado horizontal y la gestión de la salud de las aplicaciones, Kubernetes se ha convertido en la plataforma de facto para la implementación y gestión de aplicaciones en contenedores en entornos de producción (The Linux Foundation, 2024)

Docker

Docker es una plataforma de código abierto que permite la creación, distribución y ejecución de aplicaciones en contenedores. Al encapsular aplicaciones y sus dependencias en contenedores ligeros y portátiles, Docker proporciona una forma eficiente de gestionar el ciclo de vida de las aplicaciones y facilita su despliegue en diferentes entornos (Docker Inc, 2024)

Git

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software colaborativo. Al proporcionar un historial completo de cambios,

ramificaciones y fusiones, Git facilita la colaboración entre equipos de desarrollo y la gestión de código fuente en proyectos de cualquier tamaño (Software Freedom Conservancy, 2024)

Jenkins

Jenkins es una herramienta de integración continua (CI) de código abierto que automatiza el proceso de compilación, prueba y despliegue de software. Utilizando pipelines declarativos o scripts personalizados, Jenkins ayuda a garantizar la calidad del software al integrar cambios de código de forma continua y detectar errores temprano en el ciclo de desarrollo (Jenkins, 2024)

Infraestructura como Código (IaC)

La infraestructura como código (IaC) es una metodología que consiste en gestionar y aprovisionar la infraestructura de TI utilizando código y herramientas de automatización. Al tratar la infraestructura como software, IaC permite una gestión más eficiente, reproducible y escalable de los recursos de infraestructura (Red Hat, Inc., 2023)

Configuración como Código (CaC)

La configuración como código (CaC) es una práctica que consiste en definir la configuración de los sistemas y aplicaciones utilizando código, en lugar de configuraciones manuales o basadas en interfaces gráficas. Al gestionar la configuración como código, se facilita la automatización y la reproducibilidad de los entornos de software (Ansible Collaborative, 2024)

CI/CD

La integración continua (CI) y la entrega continua (CD) son prácticas de desarrollo de software que consisten en integrar cambios de código en un repositorio compartido de forma frecuente y automatizada, seguidos de la entrega automatizada y continua de las nuevas versiones del software a entornos de prueba y producción (Red Hat, Inc., 2022b)

DevOps

DevOps es una cultura y metodología que promueve la colaboración entre los equipos de desarrollo (Dev) y operaciones (Ops) para mejorar la velocidad, calidad y seguridad de la entrega de software. Al enfocarse en la automatización, la colaboración y la retroalimentación continua, DevOps permite la entrega de software de manera más rápida y confiable (Red Hat, Inc., 2022a)

CASOS DE USO

Algunas de soluciones planteadas de forma similar que intentan atender temáticas similares desde diferentes puntos de vista son las siguientes:

A continuación, se puede tener una percepción de los administradores de los beneficios que se han tenido en la implementación de IAC con Terraform.

> "With Terraform, infrastructure development and deployment that used to take more than a week can now be done in less than 30 minutes. Our teams have the autonomy and authority to build what they need, when they need it on their own."

KÉVIN DEFIVES,

INFORMATION SYSTEM ENGINEER, DECATHLON

(HashiCorp Developer, 2024a)

El manejo de los contenedores se puede ejemplificar de una forma adecuada con la implementación que se realiza por Employers y las soluciones de RedHat para contenerización:

> After evaluating several proprietary and open source solutions, Employers decided to work with trusted vendor Red Hat to build a modern digital services environment. Employers has successfully used Red Hat Enterprise Linux for several years and chose to create its new IT infrastructure using Red Hat OpenShift, running on Amazon Web Services (AWS).
>
> "Compared to the competition, Red Hat really was the clear choice," said Shaw. "With the option for support from a Technical Account Manager and consulting services, Red Hat offers the whole package businesses need for success with new technologies and approaches."
>
> Red Hat OpenShift is an enterprise Kubernetes container platform for reliable operations and management in on-premise, cloud, and edge environments. It automates container infrastructure installation, upgrades, and life-cycle management.
>
> (Red Hat, Inc., 2021)

> Booz Allen Hamilton es una compañía de EEUU la cual con una implementación de kubernetes puede ayudar a sus clientes a se realmente agiles.
>
> Booz Allen Hamilton, which has provided consulting services to the federal government for more than a century, introduced microservices, Docker containers, and AWS to its federal agency clients about five years ago. The next logical step was Kubernetes for orchestration. "Knowing that we had to be really agile and really reliable and scalable, we felt that the only technology that we know that can enable those kinds of things are the ones the CNCF provides," Folkoff says. "One of the things that is always important for the government is to make sure that the things that we build really endure. Using technology that is supported across multiple different companies and has strong governance gives people a lot of confidence." (kubernetes.io, 2017)

En cuanto a la herramienta de integración continua, Jenkins es una excelente alternativa al ser una herramienta open source con una gran comunidad que se encarga de su crecimiento constante y de mejorar cada dia para la solución de inconvenientes los usuarios, a continuación se muestran algunas de las bondades que se tienen al usar Jenkins para realizar integración continua.

- Todo el código fuente era desarrollado y luego testeado, con lo que los despliegues y las pruebas eran muy poco habituales y localizar y corregir errores era muy laborioso. El tiempo de entrega del software se prolongaba.
- Los desarrolladores tenían que esperar al desarrollo de todo el código para poner a prueba sus mejoras.

- El proceso de desarrollo y testeo eran manuales, por lo que era más probable que se produjeran fallos.

(Sentrio Labs S.L, 2021)

Marco Teórico

Los avances en las tecnologías de la información han facilitado el desarrollo de nuevos métodos, aumentando la eficiencia y la automatización del proceso de desarrollo de software. Entre ellos, Infraestructura como Código (IaC) (AWS, 2024b) y Configuración como Código (CaC) destacan por su capacidad para automatizar la gestión y configuración de la infraestructura de TI mediante el uso de código, eliminando así los procesos manuales propensos a errores. En este contexto, Terraform (AWS, 2024b) se considera una herramienta IaC importante en nuestros proyectos, permitiendo la creación, modificación y gestión de infraestructura de forma predecible y eficiente. Ansible, por otro lado, facilita la implementación de CaC, automatizando la configuración de servidores y aplicaciones, asegurando una gestión consistente y repetible del entorno operativo.

Otro aspecto importante del proyecto es el uso de tecnología de contenerización, con Docker a la vanguardia, que permite empaquetar y distribuir aplicaciones en contenedores, que encapsulan todo lo necesario para su ejecución. Esto simplifica las dependencias y mejora la portabilidad de las aplicaciones. La gestión de estos contenedores a escala se realiza a través de Kubernetes, específicamente Google Kubernetes Engine (GKE) en GCP, que organiza la gestión de cargas de trabajo y servicios para garantizar la escalabilidad y la disponibilidad continua.

La integración y despliegue continuos (CI/CD) es una práctica que aumenta la agilidad en el desarrollo de software. Al utilizar Jenkins, las fases de implementación e integración del software se automatizan, lo que facilita ciclos de lanzamiento más cortos y

frecuentes y promueve una cultura de desarrollo ágil. Esto es fundamental para responder rápidamente a los cambios del mercado y mantener la calidad del software en un entorno de producción dinámico.

Además, en este proyecto es crucial utilizar Google Cloud Platform (GCP), que proporciona una infraestructura sólida y segura para implementar y operar la aplicación. Integre con servicios de GCP como Cloud Storage y Compute Engine para respaldar eficazmente la contenerización y las operaciones de CI/CD y beneficiarse de la escalabilidad, elasticidad y seguridad que brinda la nube (safetica, 2024).

Objetivos Del Proyecto

Objetivo General

Implementar una plataforma de contenerización en Google Cloud Platform (GCP) utilizando prácticas de Infraestructura como Código (IaC) y Configuración como Código (CaC), respaldadas por los principios de DevOps.

Objetivos Específicos

Desarrollar una infraestructura en GCP utilizando prácticas avanzadas de Infraestructura como Código (IaC), integrando principios de diseño de sistemas eficientes y robustos para garantizar un entorno de desarrollo ágil y confiable.

Configurar la automatización como código para las aplicaciones en contenedores, adaptando el despliegue y la gestión automática de recursos.

Diseñar una configuración óptima de los componentes en la plataforma de contenerización, haciendo uso de la experiencia en ingeniería de sistemas y aprovechando la experiencia en telecomunicaciones para garantizar una conectividad eficiente y segura entre los servicios desplegados.

Configurar la automatización de CaC para aplicaciones en contenedores, asegurando un despliegue coherente y eficiente, adaptable a variados entornos de desarrollo.

Reducir los tiempos de implementación, minimizar el esfuerzo requerido para la gestión de la infraestructura, y mitigar posibles sobrecostos asociados al despliegue y mantenimiento de la plataforma de contenerización en Google Cloud Platform (GCP).

Contexto Y Justificación

El proyecto surge a modo de respuesta a la apremiante necesidad que tienen las compañías de actualizar su infraestructura y reducir los tiempos de desarrollo y puesta en marcha de sus aplicaciones. En una zona digitalmente evolucionando, las compañías están forzadas a continuar su competencia y proveer servicios de calidad que acaten los requerimientos del público.

En el momento en que se utilizan tecnologías en la nube, se resaltan varias ventajas que hacen que el procedimiento de utilización de herramientas como la implementación en Google Cloud Platform (GCP) (Google Cloud, 2024c) sea primordial. En primer lugar, la infraestructura en la nube tiene una mayor disponibilidad y capacidad de escalado de recursos, no teniendo en cuenta las limitaciones que tienen los recursos de la infraestructura tradicional. Esto asegura que las aplicaciones sean muy accesibles y puedan escalar de manera ágil con el fin de atender la demanda del público.

También, la facilidad de reproducir ambientes en la Nube hace que los grupos de trabajo puedan encontrar errores, realizar actualizaciones y ejecutar pruebas con mayor rapidez y eficiencia. Esto no sólo es beneficioso para las empresas en términos de la calidad y fiabilidad de sus desarrollos, sino que también genera un contexto más interdisciplinario para los grupos de tecnología de la información. La incorporación y el avance de los aplicativos se transforman en acciones colaborativas en las que varios grupos pueden participar de manera activa, esto incrementa la efectividad y la fluidez del trabajo.

La diversidad de los miembros del equipo de trabajo, confirmado por estudiantes de ingenierías de sistemas y de telecomunicaciones, es una característica fundamental que agrega valor al tema del proyecto. La mezcla de entendimientos en las dos áreas de conocimiento posibilita una mayor comprensión de los flujos de implementación, las estructuras de Networking y la comunicación correcta de los servicios de la Nube. Los expertos en sistemas informáticos añaden conocimientos sobre la administración y automatización de infraestructuras, en tanto que los expertos en telecomunicación añaden habilidades en el ámbito de las redes y las comunicaciones, esto resulta en una más grande y mejor administración de la plataforma en la nube.

Otro beneficio importante de la computación en la nube es que permite un suministro constante de nuevos recursos, códigos, estructuras y actualizaciones de seguridad. Esta habilidad les da a las compañías la capacidad de moverse con rapidez y adecuarse a las alteraciones del mercado y de las nuevas tecnologías. La habilidad de ejecutar estas actualizaciones en forma ágil (Red Hat, Inc., 2022b) y con éxito es importante para mantenerse como un competidor ágil y con éxito en un ámbito comercial que se mantiene en transformación.

Adicionalmente, mediante el manejo de los procedimientos de creación e implementación de aplicaciones, se asegura que la información sea escalable y no se pierda a causa de rotaciones de personal o transformaciones en los grupos. El código se transforma en la verdad que registra y automatiza la totalidad de las fases del procedimiento, esto ayuda a la colaboración entre los distintos procesos de las compañías y también simplifica la mejoría continua de los procedimientos.

Desafíos

La implementación de una plataforma de contenerización en Google Cloud Platform (GCP) con prácticas de DevOps es una iniciativa ambiciosa y valiosa. Sin embargo, la criticidad del proyecto radica en las mismas bondades e ideales que se buscan al unificar frentes de trabajo tan distintos como la disponibilización de infraestructura y la creación de flujos de despliegue para equipos de desarrollo. Estos dos frentes, tradicionalmente percibidos como opuestos, representan un desafío significativo en términos de alineación y colaboración.

Desafío de Unificación de Equipos

La infraestructura y el desarrollo de software han sido históricamente áreas con enfoques y prioridades diferentes. Los equipos de infraestructura se centran en la estabilidad, la seguridad y la disponibilidad de los sistemas, mientras que los equipos de desarrollo se enfocan en la innovación, la rapidez y la funcionalidad de las aplicaciones. Esta diferencia de prioridades puede llevar a conflictos y malentendidos.

El objetivo del proyecto es encontrar un punto de equilibrio entre estos dos frentes, garantizando que los componentes y las capas del sistema sean comprensibles y accesibles para todos los actores involucrados. Este equilibrio es crucial para asegurar que el conocimiento y la actualización del código se mantengan de manera efectiva, independientemente de la finalidad específica de cada componente.

Comprensibilidad y Mantenibilidad

Para alcanzar este equilibrio, es fundamental que el desarrollo del proyecto vele por la claridad y la simplicidad en la documentación y en la implementación de los componentes. Cada actor, ya sea de infraestructura o de desarrollo, debe poder entender y trabajar con los componentes del sistema sin barreras significativas de conocimiento.

- **Documentación Clara:** La documentación debe ser exhaustiva y accesible, proporcionando guías claras sobre cómo se configuran y se utilizan las diferentes herramientas y componentes.
- **Simplicidad en el Diseño:** El diseño de la arquitectura y los flujos de trabajo debe ser lo más simple posible, evitando complejidades innecesarias que puedan dificultar la comprensión y el mantenimiento.
- **Capacitación y Colaboración:** Es vital fomentar una cultura de capacitación continua y colaboración entre los equipos. Workshops, sesiones de formación y colaboraciones interdepartamentales pueden ayudar a cerrar las brechas de conocimiento.

Herramientas Bien Definidas y Soportables

Otro aspecto crítico es la elección de herramientas que sean bien definidas y soportables a largo plazo. Las herramientas seleccionadas deben ser robustas, ampliamente utilizadas y con un fuerte soporte de la comunidad o del proveedor. Esto asegura que:

- **Longevidad**: Las herramientas no se vuelvan obsoletas rápidamente y que continúen recibiendo actualizaciones y soporte.

- **Compatibilidad**: Las herramientas sean compatibles entre sí y con otras tecnologías utilizadas en la organización.
- **Escalabilidad**: Las herramientas puedan manejar el crecimiento y la evolución de las necesidades del proyecto.

Alcance Del Proyecto

El alcance de este proyecto se centra en la implementación de una plataforma de contenerización en Google Cloud Platform (GCP), aprovechando los conocimientos interdisciplinarios de ingeniería de sistemas y telecomunicaciones. La estrategia se basará en el uso de prácticas de Infraestructura como Código (IaC) y Configuración como Código (CaC) respaldadas por los principios de DevOps.

El proyecto se dividirá en tres capas de trabajo definidas de la siguiente manera:

Capa de Infraestructura:

Se desarrollará el código necesario para el despliegue de una plataforma de contenerización en Google Cloud Platform (GCP), utilizando prácticas de Infraestructura como Código (IaC) para automatizar la creación y gestión de la infraestructura. Esto incluirá la configuración de redes, almacenamiento y la implementación de servicios fundamentales para la operación de la plataforma. (Google Cloud, 2024)

Capa de Disponibilización:

Se habilitará un entorno de Jenkins en la plataforma de contenerización previamente configurada en GCP. Esto implicará la implementación de scripts de Configuración como Código (CaC) para la configuración automatizada de Jenkins y su integración con los servicios de Google Kubernetes Engine (GKE). El objetivo es proporcionar un entorno de integración continua que agilice el desarrollo y despliegue de aplicaciones contenerizadas en la nube. (Google Cloud, 2024e)

Capa de Aplicación:

Se desarrollará el código de despliegue de una demo de aplicación disponible en la nube utilizando Jenkins y GKE. Esto consistirá en la creación de scripts de Configuración como Código (CaC) para la configuración de la aplicación, contenedor y su despliegue automatizado utilizando el clúster de GKE a través de Jenkins. El propósito es demostrar el funcionamiento práctico de la plataforma implementada y su capacidad para desplegar aplicaciones de manera eficiente y escalable. (Continuous Delivery Foundation, 2024)

El alcance del proyecto incluirá el desarrollo del código necesario para cada una de estas capas de trabajo, así como la aplicación y pruebas de la plataforma en su totalidad. Es importante destacar que no se abordará la creación de aplicaciones con funcionalidades específicas para la plataforma de contenerización, ni la habilitación de funcionalidades avanzadas adicionales a las especificadas anteriormente.

Metodología Y Enfoque

Para ejecutar el proyecto, se adoptará una postura basada en los métodos modernos de creación y distribución de software, con un fuerte énfasis en la implementación y configuración de códigos (IaC y CaC). Este enfoque proporcionará la oportunidad de automatizar la elaboración, disposición y administración de la infraestructura en Google Cloud Platform (GCP), asegurando la consistencia, rapidez y eficiencia de la misma.

El punto de vista adoptado se divide en tres capas claramente identificables: la capa de infraestructura, la capa de disponibilidad y la capa de aplicación. Cada una de estas capas será abordada utilizando los métodos de IaC y CaC (HashiCorp Developer, 2024d), esto se acompañará de las prácticas y procedimientos recomendados por los fabricantes de cada una de las herramientas que serán usadas para el proyecto, lo que permitirá entregar soluciones totalmente automatizadas y escalables.

Revisión Bibliográfica:

- Se realizará una investigación exhaustiva sobre la contenerización, la infraestructura en la nube y las prácticas de DevOps.
- Se identificarán las mejores prácticas y los casos de estudio relevantes en el despliegue de plataformas de contenerización en Google Cloud Platform (GCP).

Diseño de la Arquitectura:

- Se diseñará la arquitectura de la plataforma de contenerización en GCP.
- Se definirán los componentes principales, la infraestructura necesaria y las interacciones entre ellos.

- Se prestará especial atención a la escalabilidad, la seguridad y la eficiencia del sistema.

Desarrollo de la Infraestructura como Código (IaC):

- Se utilizarán herramientas de automatización para definir la infraestructura como código.
- Se escribirán scripts y plantillas para provisionar y configurar los recursos en GCP de manera reproducible y escalable.

Implementación de Prácticas de DevOps:

- Se integrarán prácticas de DevOps en el proceso de desarrollo y despliegue de la plataforma.
- Se establecerán pipelines de CI/CD (Integración Continua/Despliegue Continuo) para automatizar las pruebas y el despliegue de aplicaciones en contenedores.

Despliegue de la Plataforma en GCP:

- Se procederá con el despliegue de la plataforma de contenerización en GCP utilizando la infraestructura y la configuración definidas como código.
- Se realizarán pruebas exhaustivas para garantizar el correcto funcionamiento del sistema en el entorno de producción.

Evaluación y Validación:

- Se evaluará el desempeño de la plataforma desplegada en GCP, comparándola con los objetivos establecidos al principio del proyecto.
- Se realizarán pruebas de carga, seguridad y confiabilidad para validar su funcionamiento.

Desarrollo Del Proyecto

Diseño Del Proyecto

Requerimientos Del Proyecto

- **Acceso a Google Cloud Platform (GCP):**

Despliegue de Infraestructura en GCP: Para utilizar Terraform y desplegar la infraestructura en Google Cloud Platform, se requiere acceso a una cuenta de GCP con los permisos adecuados. Esto le permite crear y administrar los recursos necesarios, como clústeres, redes y servicios en la nube de Kubernetes.

- **Acceso a Herramientas de Desarrollo de Código:**

Terraform y Ansible: Para desarrollar el código de infraestructura como código (IaC) utilizando Terraform y Ansible, es necesario contar con acceso a las herramientas y los entornos de desarrollo adecuados. Esto incluye instalar las últimas versiones de Terraform y Ansible, así como acceder al repositorio de código para almacenar y compartir scripts de configuración.

- **Infraestructura en Google Cloud Platform (GCP):**

Se requiere desplegar un servicio básico de Google Kubernetes Engine (GKE) utilizando Terraform. Este servicio será la base sobre la cual se construirá la plataforma de despliegue de aplicaciones. La configuración de la infraestructura debe realizarse de forma automatizada para garantizar la replicabilidad y la consistencia en todos los entornos.

- **Despliegue de Contenedor en GKE:**

El siguiente paso consiste en implementar un contenedor en el servicio de GKE creado anteriormente. Este contenedor albergará una instancia de Jenkins, una herramienta de automatización ampliamente utilizada en el desarrollo de software. Jenkins facilitará la automatización de procesos clave, como la integración continua y el despliegue continuo, lo que mejorará la eficiencia y la consistencia en el ciclo de vida del desarrollo de software.

- **Automatización con Jenkins:**

Una vez configurado Jenkins en el contenedor desplegado en GKE, se procederá a establecer pipelines de Jenkins para automatizar la integración y el despliegue de la aplicación de ejemplo. Estos pipelines serán parametrizados para permitir la flexibilidad en el despliegue de diferentes versiones de la aplicación, lo que facilitará la gestión y mantenimiento del sistema en el futuro.

- **Despliegue de Aplicación de Ejemplo:**

Finalmente, se desplegará una aplicación de ejemplo disponible en un repositorio gratuito. Esta aplicación estará empaquetada en un contenedor Docker y estará lista para su despliegue en GKE a través de Jenkins. Este paso servirá como una demostración práctica del funcionamiento del sistema y permitirá validar la efectividad de la infraestructura y la automatización implementadas.

Arquitectura De La Solución

La arquitectura propuesta se centra en el despliegue de una plataforma de CI/CD utilizando Google Kubernetes Engine (GKE), Terraform, Ansible y Jenkins.

Componentes Principales:

1- **Google Cloud Platform (GCP):**

Servicios Utilizados: Google Kubernetes Engine (GKE), Cloud Storage, IAM.

Rol: Proveedor de servicios en la nube donde se desplegará la infraestructura y la aplicación.

2- **Terraform**:

Función: Herramienta utilizada para definir y gestionar la infraestructura como código en GCP.

Configuración: Se desarrollará código Terraform para crear y configurar los recursos necesarios en GCP, incluyendo el clúster de GKE, las redes, y los servicios de almacenamiento.

3- **Ansible**:

Función: Utilizado para configurar los servidores y desplegar la aplicación dentro de los contenedores en GKE.

Configuración: Se desarrollarán scripts Ansible para la configuración de los servidores y la automatización del despliegue de la aplicación dentro de los contenedores en GKE.

4- **Jenkins**:

Función: Plataforma de automatización para la integración continua y el despliegue continuo (CI/CD).

Configuración: Se configurará Jenkins para automatizar la integración y el despliegue de la aplicación de ejemplo en GKE. Se desarrollarán pipelines de Jenkins para orquestar estos procesos de manera eficiente.

5- **Aplicación de Ejemplo:**

Función: Aplicación de prueba que será desplegada en GKE.

Configuración: La aplicación estará empaquetada en un contenedor Docker y lista para su despliegue en GKE. Se integrará con Jenkins para automatizar su despliegue y pruebas.

Flujo de Trabajo:

1- **Despliegue de Infraestructura con Terraform:**

Terraform se utilizará para definir y crear la infraestructura necesaria en GCP, incluyendo el clúster de GKE y otros recursos.

2- **Configuración de Servidores con Ansible:**

Ansible se encargará de configurar los servidores y desplegar la aplicación dentro de los contenedores en GKE. Los scripts de Ansible automatizarán este proceso de manera eficiente.

3- **Automatización con Jenkins:**

Jenkins se configurará para automatizar la integración y el despliegue de la aplicación en GKE. Se desarrollarán pipelines de Jenkins para orquestar estos procesos y permitir su ejecución automatizada.

4- **Despliegue de Aplicación de Ejemplo:**

La aplicación de ejemplo será desplegada en GKE utilizando los pipelines de Jenkins. Se realizarán pruebas automáticas para validar su funcionamiento.

Implementación Del Proyecto

Desarrollo del código Terraform:

Proceso de Desarrollo:

- **Definición de Requerimientos:** Revisión de los requerimientos del proyecto para identificar los recursos necesarios en GCP.
- **Configuración del Entorno:** Preparación del entorno de desarrollo local con Terraform y configuración de las credenciales de GCP.
- **Definición de Recursos:** Creación del código Terraform para definir y gestionar la infraestructura en GCP, incluyendo el clúster de GKE, las redes, y los servicios de almacenamiento.
- **Parametrización y Modularización:** Uso de variables y módulos en Terraform para parametrizar la configuración y promover la reutilización del código.
- **Pruebas y Validación:** Ejecución de pruebas locales para validar el código Terraform y asegurar su correcto funcionamiento.
- **Aplicación de Cambios:** Aplicación de los cambios al entorno de GCP mediante la ejecución del comando terraform apply.

Configuración Con Ansible:

Proceso de Desarrollo:

- **Definición de Tareas:** Identificación de las tareas necesarias para la configuración de los servidores y el despliegue de la aplicación en GKE.
- **Creación de Playbooks**: Desarrollo de playbooks de Ansible que contengan las tareas necesarias para la configuración y el despliegue.
- **Parametrización y Modularización:** Parametrización de los playbooks para permitir la configuración flexible de los servidores y la aplicación.
- **Pruebas Locales:** Verificación de los playbooks con pruebas locales para asegurar su correcto funcionamiento.
- **Ejecución Manual:** Ejecución manual de los playbooks desde un equipo local para configurar los servidores y desplegar la aplicación en GKE.

Integración con Jenkins (contenedor en GKE):

Proceso de Configuración:

- **Despliegue de Jenkins en GKE:** Creación de un clúster de GKE y despliegue de Jenkins como un contenedor en el clúster.
- **Configuración de Pipelines:** Configuración de pipelines de Jenkins para automatizar la integración y el despliegue de la aplicación en GKE.

- **Configuración de Credenciales**: Configuración de las credenciales necesarias en Jenkins para acceder a GCP y ejecutar comandos de Terraform y Ansible.
- **Despliegue Automatizado:** Configuración de los pipelines de Jenkins para desplegar automáticamente la aplicación en GKE al detectar cambios en el repositorio de código.

Pruebas Del Proyecto

Pruebas De Infraestructura:

Proceso de Pruebas:

- **Verificación de Recursos**: Se realizó una verificación exhaustiva de los recursos desplegados en Google Cloud Platform (GCP) para asegurar que se crearon correctamente.
- **Conectividad de Red**: Se verificó la conectividad de red entre los diferentes componentes de la infraestructura, incluyendo el clúster de GKE, los servicios de almacenamiento y las instancias de máquinas virtuales.
- **Cumplimiento de Requerimientos**: Se verificó que la infraestructura desplegada en GCP cumpla con los requerimientos del proyecto, incluyendo la configuración de redes, almacenamiento y servicios fundamentales.

Pruebas de integración:

Proceso de Pruebas:

- **Ejecución de Pipelines**: Se ejecutaron los pipelines de Jenkins para asegurar que la integración y el despliegue automático funcionen correctamente.
- **Comprobación de Flujo de Trabajo**: Se verificó que todos los pasos del flujo de trabajo, desde la integración hasta el despliegue, se ejecuten en el orden correcto y sin errores.
- **Integración de Componentes**: Se realizó una prueba de integración para verificar que todos los componentes del proyecto, incluyendo Terraform, Ansible y Jenkins, funcionen juntos de manera adecuada.

Pruebas de Aplicación:

Proceso de Pruebas:

- **Pruebas de Funcionalidad**: Se realizaron pruebas funcionales exhaustivas para verificar que la aplicación de prueba desplegada en GKE funcione según lo esperado.
- **Pruebas de Rendimiento**: Se realizaron pruebas de rendimiento para evaluar el rendimiento de la aplicación bajo diferentes cargas de trabajo y condiciones.

Beneficios Esperados

La implementación de infraestructura como código y su disponibilidad eficiente en Google Cloud Platform (GCP) ofrecerá numerosos beneficios importantes a las compañías que prestan servicios financieros. Estas ventajas se centran en la agilidad, la eficiencia y la calidad en el desarrollo e implementación de soluciones en la nube. A continuación, se presentan los principales beneficios esperados:

- Reducción del tiempo de aprovisionamiento: La automatización del proceso de creación de la infraestructura permitirá que los equipos de desarrollo obtengan rápidamente el entorno necesario para trabajar, acortando así los tiempos de espera y agilizando el inicio de los proyectos.
- Mayor consistencia y reproducibilidad: Al definir la infraestructura como código, se garantizará que todos los entornos de desarrollo, pruebas y producción sean consistentes entre sí, lo que facilitará la detección y solución de problemas y reducirá el riesgo de errores.
- Optimización de recursos: La gestión automatizada de la infraestructura permitirá una mejor utilización de los recursos disponibles en GCP, lo que resultará en ahorros en costos y una mayor eficiencia operativa.
- Facilitación del trabajo colaborativo: La infraestructura como código facilitará el trabajo colaborativo entre los equipos de desarrollo, permitiendo cambios controlados y documentados a través del código, lo que mejorará la comunicación y la transparencia en el proceso.

- Mejora en la velocidad de entrega: La disponibilidad eficiente de la infraestructura agilizará el ciclo de desarrollo y despliegue de soluciones, permitiendo entregas más rápidas y frecuentes de nuevas funcionalidades y actualizaciones a los usuarios finales.

Público Objetivo

El presente proyecto aplicado está dirigido a empresas interesadas en encontrar soluciones para optimizar sus procesos de desarrollo e implementación de aplicaciones en la nube. Esta audiencia puede estar compuesta por empresas emergentes recientemente fundadas, así como por organizaciones bien establecidas que representan diferentes sectores como tecnología, finanzas, comercio electrónico, entre otros.

El objetivo principal se centra en el sector financiero, específicamente en equipos de desarrollo que buscan optimizar su flujo de trabajo y aumentar su productividad en la creación de software para pruebas y entrega. También se apunta a líderes de equipo y gerentes de proyectos que estén dispuestos a introducir los estándares contemporáneos de desarrollo de software y aprovechar al máximo los beneficios de la nube.

Se busca ofrecer una respuesta accesible incluso para aquellos que quizás no estén tan familiarizados con la tecnología, pero que estén abiertos a la idea de adoptar estrategias más ágiles y eficientes en su operación. El objetivo es proporcionar un procedimiento sencillo de implementación de infraestructura en la nube para disfrutar de sus beneficios, reduciendo al mismo tiempo la complejidad.

Análisis De Requisitos Del Sistema

Descripción de Usuarios

Equipos de Desarrollo:

- **Perfil**: Desarrolladores de software que trabajan en la creación, prueba y despliegue de aplicaciones.
- **Necesidades**:
  - Acceso a ambientes ligeros y ágiles para pruebas de código.
  - Facilidad para desplegar aplicaciones de prueba sin necesidad de configurar manualmente la infraestructura.
  - Capacidad para iterar rápidamente sobre las aplicaciones y recibir feedback.
- **Expectativas**:
  - Ambientes de desarrollo consistentes y reproducibles.
  - Herramientas que faciliten la integración continua y la entrega continua (CI/CD).
  - Posibilidad de proporcionar accesos a usuarios finales para pruebas y validaciones.

Equipos de Infraestructura:

- **Perfil**: Administradores de sistemas y arquitectos de infraestructura encargados de gestionar la infraestructura de TI.
- **Necesidades**:

        - Conocimiento básico de las herramientas usadas (GCP, Docker, Kubernetes, Terraform, Ansible).
        - Capacidad para configurar y mantener la infraestructura de soporte para los equipos de desarrollo.
        - Colaboración con los equipos de desarrollo para asegurar la usabilidad y eficiencia de la infraestructura.
    - **Expectativas**:
        - Herramientas que permitan la gestión automatizada de la infraestructura.
        - Facilidad para escalar y ajustar la infraestructura según las necesidades del desarrollo.

Usuarios Finales:

- **Perfil**: Usuarios que probarán y validarán las funcionalidades de las aplicaciones desplegadas.
- **Necesidades**:
    - Acceso sencillo y seguro a las aplicaciones para realizar pruebas y proporcionar feedback.
- **Expectativas**:
    - Interacciones fluidas y sin complicaciones con las aplicaciones.
    - Funcionalidades que se alineen con sus necesidades y expectativas.

Requisitos del Sistema

Requisitos de Negocio

- **BN1**: Reducir los costos asociados con la configuración manual de ambientes de prueba y despliegue.
- **BN2**: Agilizar el ciclo de vida de desarrollo y despliegue de software.
- **BN3**: Aumentar la colaboración entre equipos de desarrollo e infraestructura.

Requisitos del Usuario

- **RU1**: Los desarrolladores deben poder desplegar aplicaciones de prueba en menos de 5 minutos.
- **RU2**: Los equipos de infraestructura deben poder gestionar la infraestructura usando herramientas automatizadas.
- **RU3**: Los usuarios finales deben tener acceso a las aplicaciones de prueba sin necesidad de configuraciones adicionales.

Casos de Uso

- **CU1**: **Desplegar aplicación de prueba**.
    - Actores: Desarrollador.
    - Descripción: Un desarrollador despliega una versión de prueba de una aplicación en un ambiente de contenedores.
    - Precondiciones: El código está listo para ser desplegado.
    - Flujo principal:
        1. El desarrollador accede a Jenkins.
        2. Selecciona el proyecto a desplegar.

3. Jenkins ejecuta el pipeline de CI/CD.
4. La aplicación se despliega en GKE.

- Postcondiciones: La aplicación está disponible para pruebas.

- **CU2**: **Gestionar infraestructura**.
  - Actores: Administrador de Infraestructura.
  - Descripción: Un administrador configura y mantiene la infraestructura utilizando Terraform y Ansible.
  - Precondiciones: Acceso a las herramientas necesarias.
  - Flujo principal:
    1. El administrador escribe los scripts de Terraform y Ansible.
    2. Ejecuta los scripts para desplegar y configurar la infraestructura.
    3. Verifica que los recursos se han desplegado correctamente.
  - Postcondiciones: La infraestructura está lista para su uso.

Requisitos No Funcionales Detallados

RNF1: Escalabilidad

- **RNF1-Velocidad**:
  - El sistema debe poder desplegar una nueva aplicación en un entorno de prueba en menos de 30 minutos.
  - Las actualizaciones de las aplicaciones desplegadas deben reflejarse en los entornos de prueba en menos de 3 minutos después de ser aplicadas.
- **RNF1-Capacidad de Respuesta**:

    - La interfaz de usuario de Jenkins debe responder a las acciones del usuario en menos de 2 segundos.
    - Las solicitudes a las aplicaciones desplegadas deben tener un tiempo de respuesta inferior a 200 milisegundos bajo condiciones de carga estándar.

- **RNF1-Eficiencia**:
    - La utilización de recursos de CPU y memoria en los nodos de GKE debe mantenerse por debajo del 70% durante el funcionamiento normal.
    - El sistema debe escalar automáticamente para mantener una eficiencia óptima, añadiendo o eliminando nodos en función de la carga de trabajo sin intervención manual.

RNF2: Disponibilidad

- **RNF2-Disponibilidad**:
    - La infraestructura debe tener una disponibilidad del 99.9%.
    - Los servicios críticos deben ser monitoreados constantemente, y se deben implementar mecanismos de recuperación automática ante fallos para minimizar el tiempo de inactividad.

RNF3: Seguridad

- **RNF3-Protección de Datos**:
    - Todos los datos en tránsito entre los servicios deben estar cifrados usando TLS 1.2 o superior.
    - Los datos sensibles almacenados en la nube deben estar cifrados utilizando cifrado AES-256.

- **RNF3-Acceso al Sistema**:
    - El acceso a Jenkins, GCP y otros componentes del sistema debe estar protegido mediante autenticación multifactor (MFA).
    - Los roles y permisos deben estar configurados siguiendo el principio de menor privilegio, asegurando que los usuarios solo tengan acceso a los recursos necesarios para sus funciones.
    - Deben realizarse auditorías de seguridad trimestrales para identificar y mitigar posibles vulnerabilidades en el sistema.

RNF4: Mantenibilidad

- **RNF4-Facilidad de Mantenimiento**:
    - El código y las configuraciones deben estar bien documentados para facilitar su mantenimiento y actualización.
    - Se deben utilizar estándares de codificación y prácticas recomendadas para asegurar que el código sea limpio, modular y fácil de entender.
- **RNF4-Actualización y Documentación**:
    - Toda la documentación del sistema debe mantenerse actualizada, incluyendo guías de instalación, operación y mantenimiento.
    - Debe existir un sistema de control de versiones para el código y las configuraciones, permitiendo rastrear cambios y revertir a versiones anteriores si es necesario.

Usabilidad

- **RNF-U1: Facilidad de Uso**:

    - La interfaz de usuario de Jenkins debe ser intuitiva y fácil de navegar, con todos los controles claramente etiquetados y accesibles.
    - La documentación del sistema debe ser completa y estar actualizada, proporcionando guías paso a paso para los usuarios en todas las funciones principales del sistema.

- **RNF-U2: Accesibilidad**:
    - La interfaz de usuario debe ser accesible según las pautas WCAG 2.1 nivel AA, asegurando que los usuarios con discapacidades puedan utilizar el sistema sin problemas.
- **RNF-U3: Soporte y Mantenimiento**:
    - Debe existir un sistema de soporte accesible para los usuarios, incluyendo una base de conocimientos en línea, un sistema de tickets de soporte y asistencia en tiempo real durante las horas laborales.

Requisitos de Hardware

Dado que el proyecto utilizará Google Cloud Platform (GCP), los requisitos de hardware se centrarán en los recursos y servicios proporcionados por GCP:

- **Nodos de Kubernetes (GKE)**:
    - **Tipo de Máquina**: n1-standard-2 (2 vCPUs, 7.5 GB RAM) para entornos de desarrollo y prueba.
    - **Tipo de Máquina**: n1-standard-4 (4 vCPUs, 15 GB RAM) para entornos de producción.
    - **Escalabilidad**: Autoscaling habilitado para ajustar el número de nodos según la carga.
- **Almacenamiento**:
    - **Persistent Disks**: SSDs para almacenamiento rápido y eficiente de datos críticos.
- **Redes**:
    - **VPC (Virtual Private Cloud)**: Red privada para comunicación interna entre los servicios.

Requisitos de Software

Los componentes de software necesarios para la implementación y funcionamiento del proyecto son:

- **Google Cloud SDK**: Herramientas de línea de comandos para interactuar con GCP.
- **Terraform**: Para gestionar la infraestructura como código.

    - **Version**: 1.0 o superior.
- **Ansible**: Para la configuración y gestión de los servidores.
    - **Version**: 2.9 o superior.
- **Docker**: Para contenerización de aplicaciones.
    - **Version**: 20.10 o superior.
- **Kubernetes**: Para orquestación de contenedores.
    - **Version**: 1.20 o superior.
- **Jenkins**: Para integración continua y despliegue continuo.
    - **Version**: 2.263 o superior.
- **Git**: Sistema de control de versiones.
    - **Version**: 2.30 o superior.
- **GitHub**: Plataforma de SCM.

Descripción de Interfaces del Sistema

Interfaces Internas

- **Interacción entre Microservicios:**
    - **Protocolos**: HTTP/HTTPS, gRPC.
    - **Formato de Datos**: JSON.
- **Comunicación entre Jenkins y GKE**:
    - **Jenkins Plugins**: Kubernetes plugin para la ejecución de pipelines en nodos GKE.
    - **APIs**: Kubernetes API para la gestión de pods, servicios, y deployments.
- **Interacción entre Terraform y GCP**:

    - **APIs**: Google Cloud API para aprovisionamiento y gestión de recursos.
    - **Scripts**: Archivos HCL (HashiCorp Configuration Language) para definir la infraestructura.

Interfaces Externas

- **Acceso de Usuarios a Jenkins**:
    - **Interfaz Web**: GUI accesible a través de navegadores web para automatización de tareas e integración con otras herramientas.
    - **Seguridad**: Autenticación, roles y permisos.
- **Desarrolladores Interactuando con GitHub**:
    - **Interfaz Web**: Para administración de repositorios.
- **Acceso a Aplicaciones Desplegadas**:
    - **Endpoint URLs**: URLs públicas o privadas para acceder a las aplicaciones en contenedores.

Arquitectura del Sistema

La arquitectura del sistema estará basada en una estructura de microservicios orquestados por Kubernetes en GCP. Los componentes principales incluirán:

- **Control Plane**:
    - GKE (Google Kubernetes Engine) para orquestación de contenedores.
    - Terraform para aprovisionamiento de infraestructura.
    - Ansible para configuración de servidores y servicios.
- **Data Plane**:

    - Pods desplegados en nodos de Kubernetes que contienen las aplicaciones de microservicios.
    - Persistent Volumes para almacenamiento de datos persistentes.
    - ConfigMaps y Secrets para la configuración y gestión de credenciales sensibles.
- **CI/CD Pipeline**:
    - Jenkins como servidor de CI/CD desplegado en un contenedor de GKE.
    - GitHub como sistema de control de versiones y repositorio de código fuente.
    - Jenkins pipeline configurado para automatizar el despliegue de aplicaciones en GKE.
- **Networking**:
    - VPC configurada para la comunicación interna segura entre servicios.

Diagramas de Arquitectura Técnica

A continuación, se describe cómo puede estructurarse un diagrama de arquitectura técnica del proyecto:

1. **Diagrama de Infraestructura**:
    - Representa la arquitectura de red (VPC, subnets).
    - Muestra los nodos de Kubernetes y su configuración (autoscaling, tipos de máquina).
    - Incluye servicios adicionales (Cloud Storage).
2. **Diagrama de Componentes**:

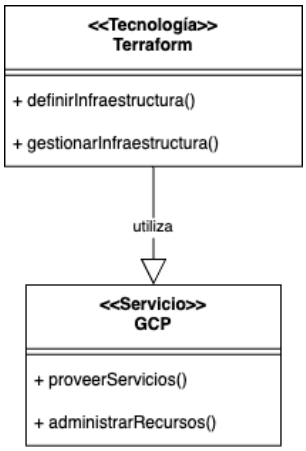
- Detalla los componentes principales del sistema (Jenkins, GitHub, Docker, Kubernetes).
- Muestra cómo interactúan entre sí (APIs, plugins, interfaces web).
- Representa los pipelines de CI/CD.

3. **Diagrama de Flujo de Datos**:
   - Ilustra el flujo de datos desde el desarrollo de código hasta el despliegue en producción.
   - Muestra la interacción entre microservicios.
   - Incluye la recopilación de métricas y logs.

Diagramas UML

Figura 1

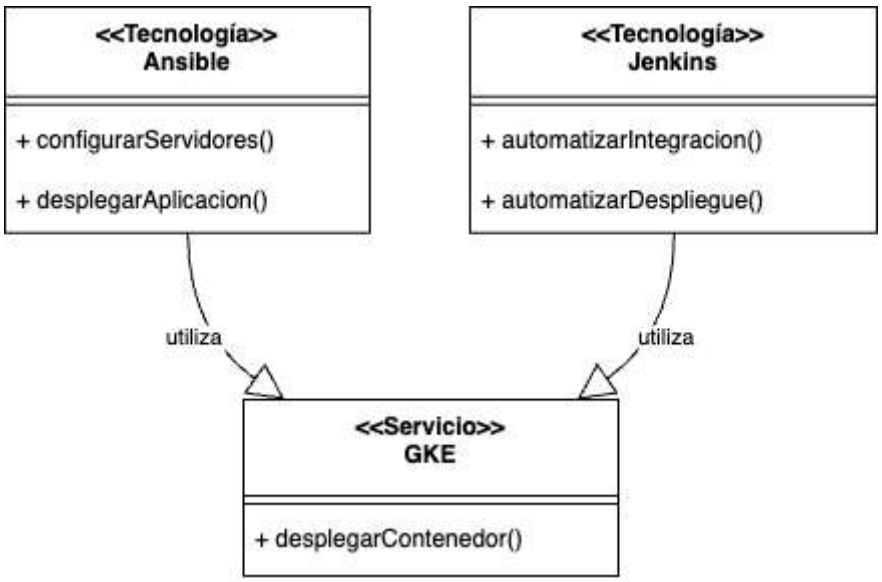
Diagrama UML (Infraestructura)



Nota - Diagrama UML Capa de Infraestructura. *Fuente. Autor*

Figura 2

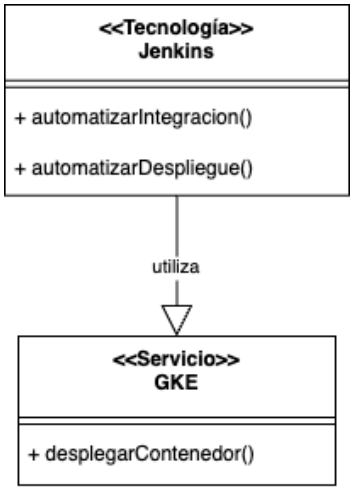
Diagrama UML (Disponibilización)



Nota - Diagrama UML Capa 2 de Disponibilización. *Fuente. Autor*

Figura 3

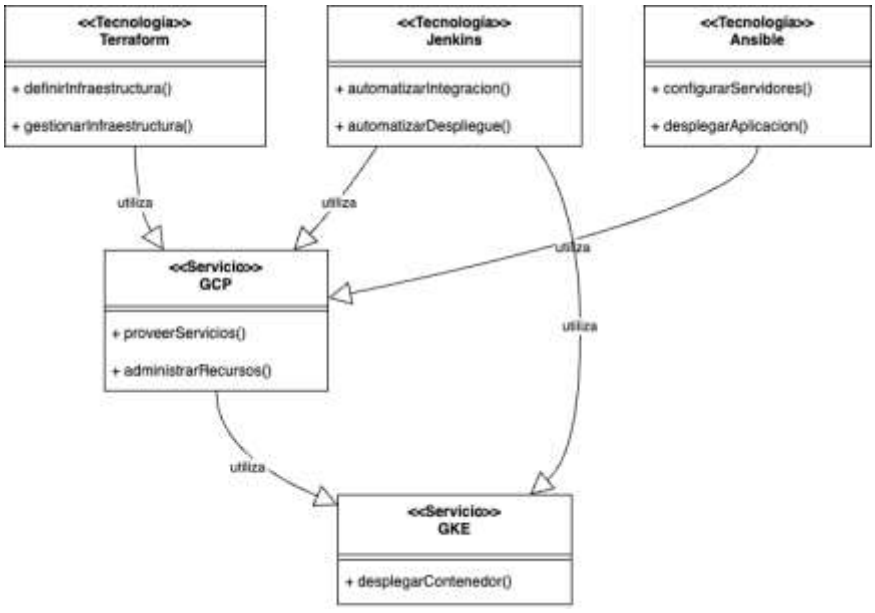
Diagrama Capa 3 (Aplicación)



Nota - Diagrama UML Capa de Aplicación. *Fuente. Autor*

Figura 4

Diagrama General del despliegue



Nota - Diagrama UML General. *Fuente. Autor*

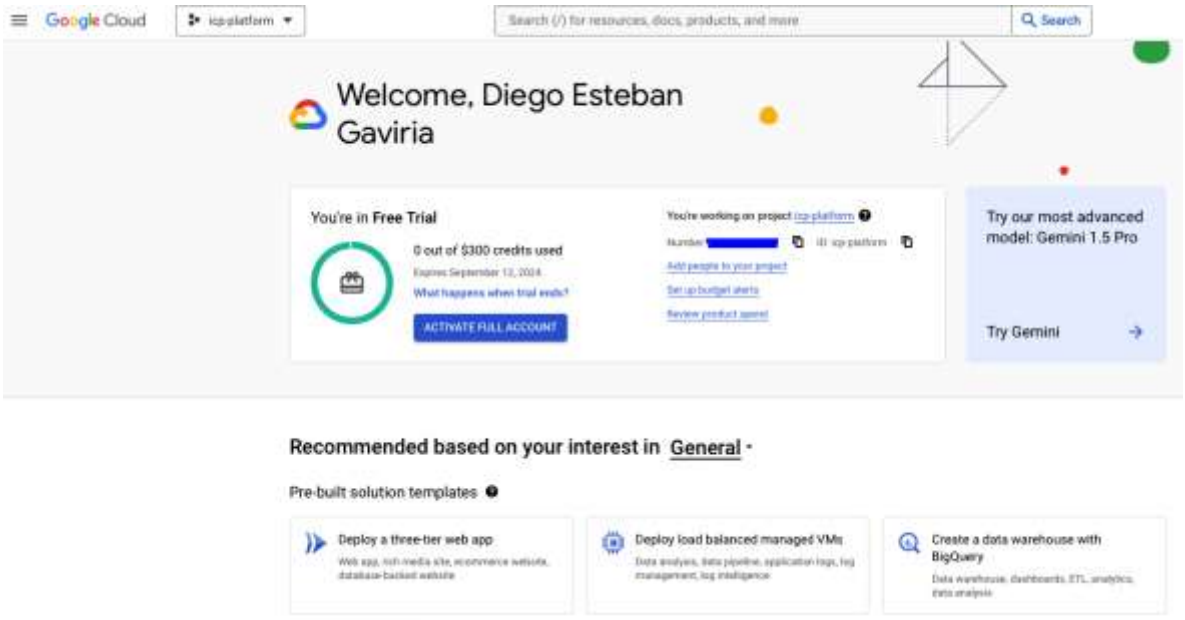
Casos De Prueba

Caso de prueba de exploración cloud**:**

- **Objetivo**: Validar y activar los recursos del cloud que permiten disponibilizar los recursos.
- **Descripción**: Se creó una cuentea en Google Cloud Platform y se activan cada uno de los recursos (Kubernetes Engine, API Cloud Resource Manager) requeridos para el funcionamiento del prototipo para asegurar el funcionamiento de los componentes.
- **Resultados**: El caso de prueba confirmó que se cuentan con los recursos requeridos para el despliegue de los componentes.

Figura 5

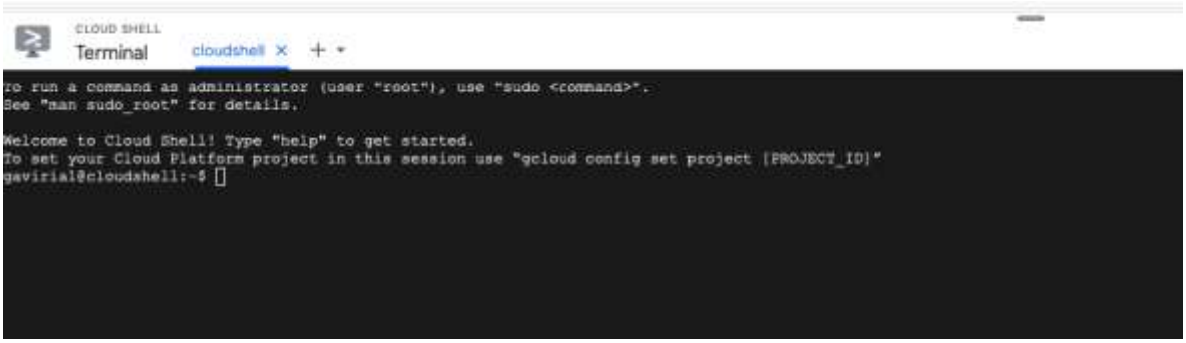
Pantalla de Inicio GCP



Nota - Pantalla de inicio de Google Cloud Platform (GCP). *Fuente. Autor*

Figura 6

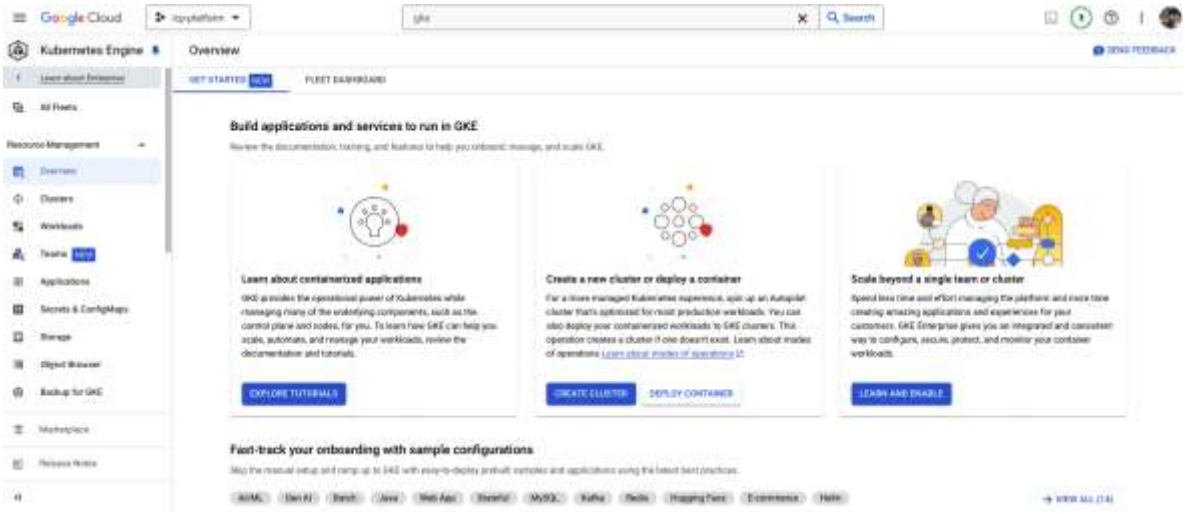
Consola de GCP



Nota - Consola integrada de cloud GCP. *Fuente. Autor*

Figura 7

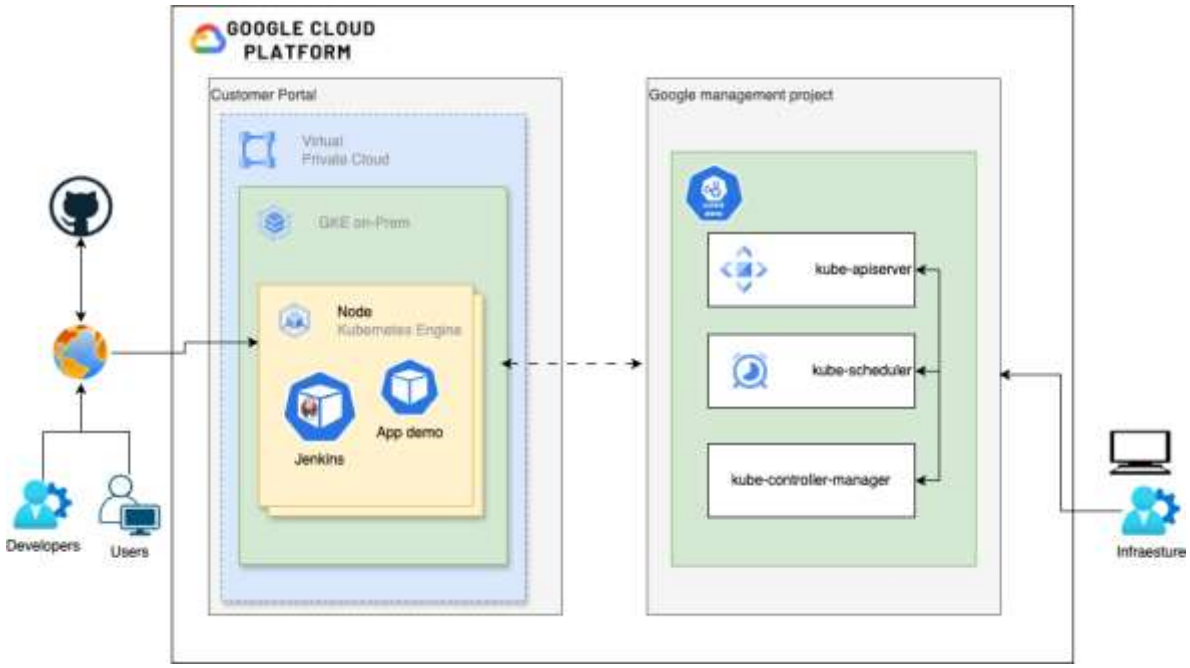
Activacion Servicio Kubernetes



Nota. Activacion de servicio de Kubernetes Engine en GCP. *Fuente. Autor*

Figura 8

Diagrama de Arquitectura



Nota - Diagrama de Arquitectura del proyecto. *Fuente. Autor*

Caso de prueba de contenerización**:**

- **Objetivo**: Validar el uso de Docker para empaquetar aplicaciones.
- **Descripción**: Se creó un contenedor Docker para una aplicación simple (una aplicación web basada un lenguaje de programación estándar) para asegurarse de que todas sus dependencias se instalan correctamente y que puede ejecutarse en cualquier entorno Docker.
- **Resultados**: El caso de prueba confirmó que la aplicación puede ser contenerizada y ejecutada en diferentes entornos sin problemas.

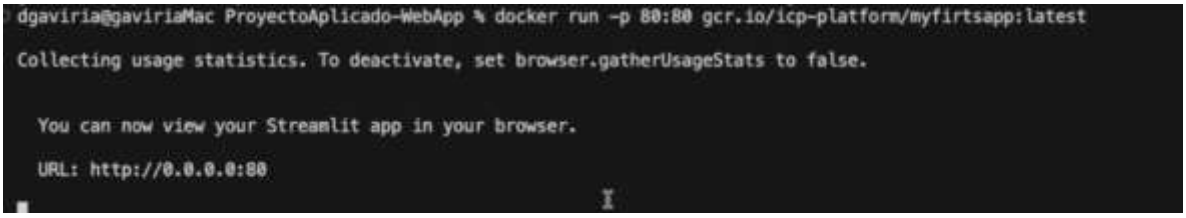
Figura 9

Compilacion de Contendor

Nota. Compilacion de contenedor Docker. *Fuente. Autor*

Figura 10

Validacion Ejecucion Contenedor



Nota. Validacion de ejecucion de contenedor en local. *Fuente. Autor*

Figura 11

Validacion de Ejecucion de contenedores

Nota. Validacion de contenedores en ejecucion desde la consola. *Fuente. Autor*

Figura 12

Ejecucion de la aplicación en local

Nota. Validación de aplicación en ejecución en local. *Fuente. Autor*

Figura 13

Carga Imagen de Contendor

Nota. Carga de imagen de contenedor a registry. *Fuente. Autor*

Figura 14

Finalizacion Carga de Imagen

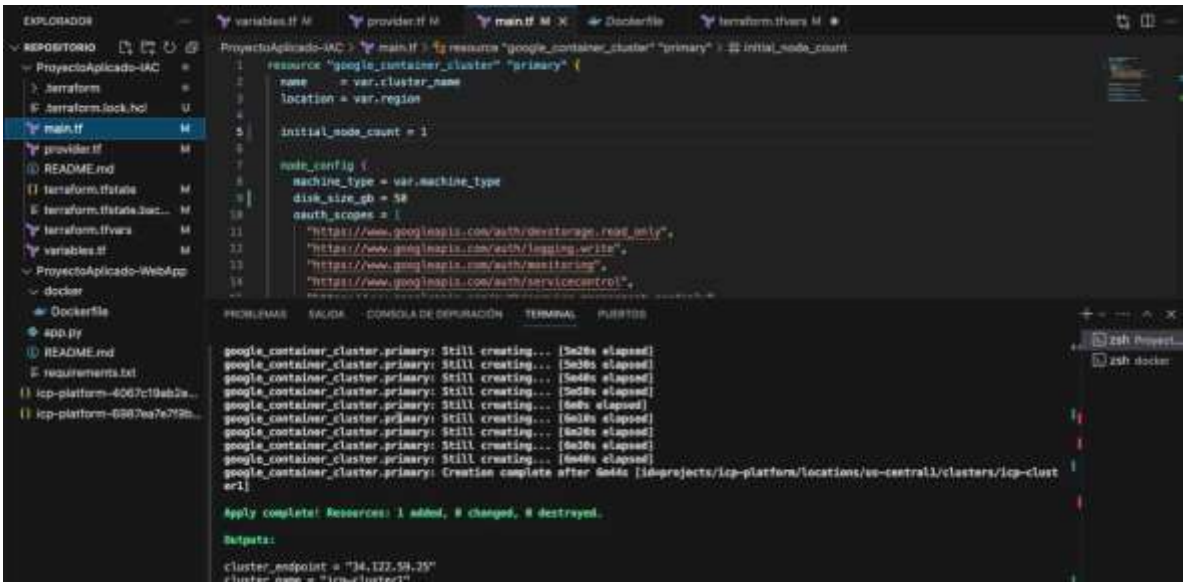
Nota. Finalizacion de carga de imagen a registry. *Fuente. Autor*

Caso de prueba de orquestación**:**

- **Objetivo**: Probar la capacidad de Kubernetes (GKE) para gestionar contenedores a gran escala.
- **Descripción**: Despliegue de la aplicación contenerizada en un clúster de Kubernetes, utilizando un archivo de configuración YAML para definir los despliegues, servicios y configuraciones.
- **Resultados**: Se validó la capacidad de Kubernetes para escalar la aplicación automáticamente y gestionar su ciclo de vida.

Figura 15

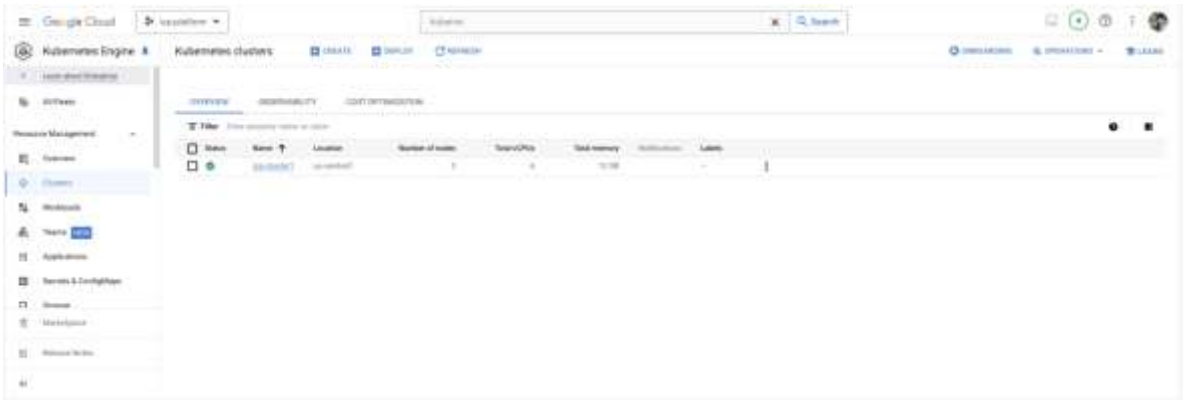
Despliegue Infraestructura como codigo



Nota. Despliegue de infraestructura por código en Visual Studio Code. *Fuente. Autor*

Figura 16

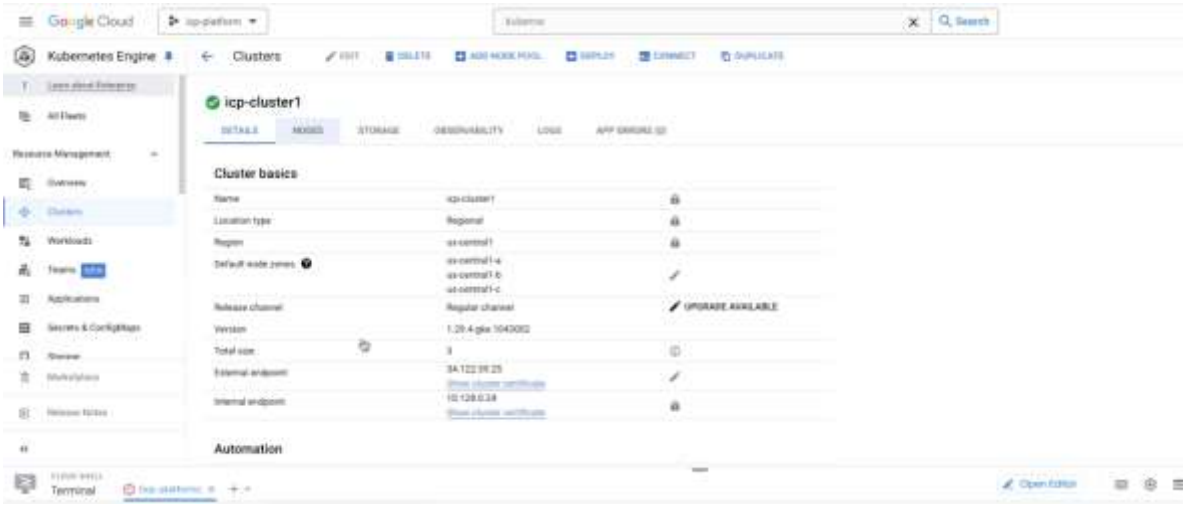
Comprobacion de ejecucion de servicios



Nota. Validación de servicios en ejecución en GCP. *Fuente. Autor*

Figura 17

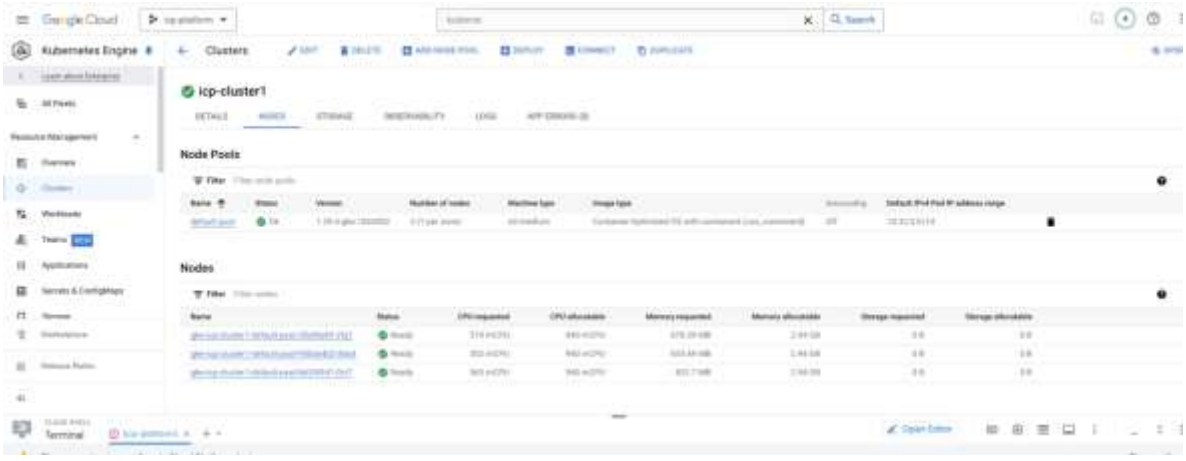
Chequeo de Cluster



Nota. Validación de cluster de kubernetes en GCP. *Fuente. Autor*

Figura 18

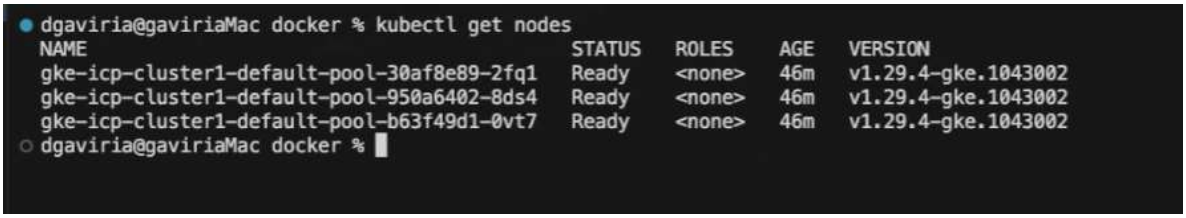
Revision de los nodos de Kubernetes



Nota. Validación de nodos de servicio de kubernetes en GCP. *Fuente. Autor*

Figura 19

Revision de los nodos desde consola



Nota. Validación de nodos desde la consola de comandos. *Fuente. Autor*

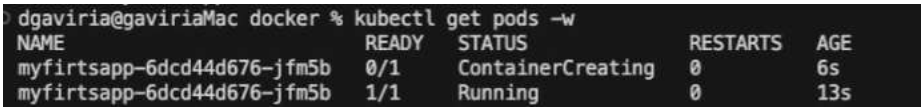
Figura 20

Despliegue de Servicios en la nube

Nota. Despliegue de servicios en nube a traves de la consola. *Fuente. Autor*

Figura 21

Revision de ejecución de contenedores



Nota. Validación de contenedores en ejecución a traves de la conola. *Fuente. Autor*

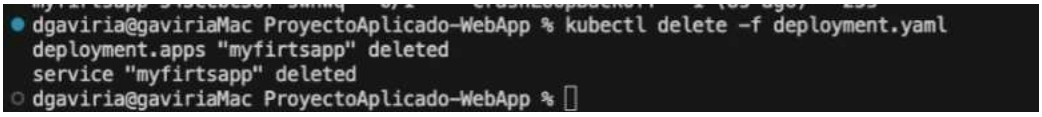
Figura 22

Despliegue de aplicación

Nota. Validación de aplicación desplegada en contenedores. *Fuente. Autor*

Figura 23

Eliminación de recursos en nube



Nota. Eliminación de recursos en nube desde la consola. *Fuente. Autor*

Caso de prueba de CI/CD**:**

- **Objetivo**: Implementar un pipeline básico de CI/CD usando Jenkins y GitHub.
- **Descripción**: Se configuró un repositorio GitHub y un pipeline de Jenkins que realiza las siguientes tareas:

- Compilación del código.
- Pruebas automatizadas.
- Creación de una imagen Docker.
- Despliegue de la imagen en un clúster de GKE.

- **Resultados**: El pipeline automatizó con éxito las etapas de construcción, prueba y despliegue, reduciendo significativamente el tiempo de entrega.

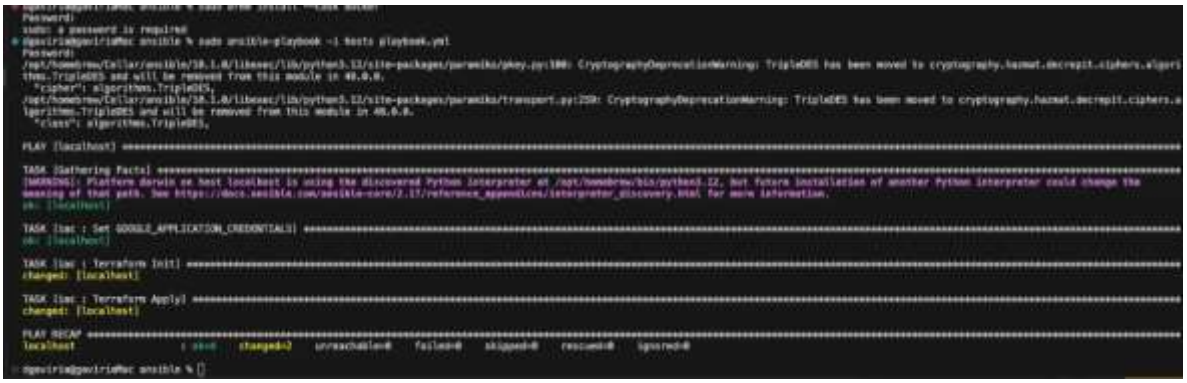
Figura 24

Ejecución de Playbook de Ansible

Nota. Ejecucion de playbook de Ansible para el deploy de la infraestructura. *Fuente. Autor*

Figura 25

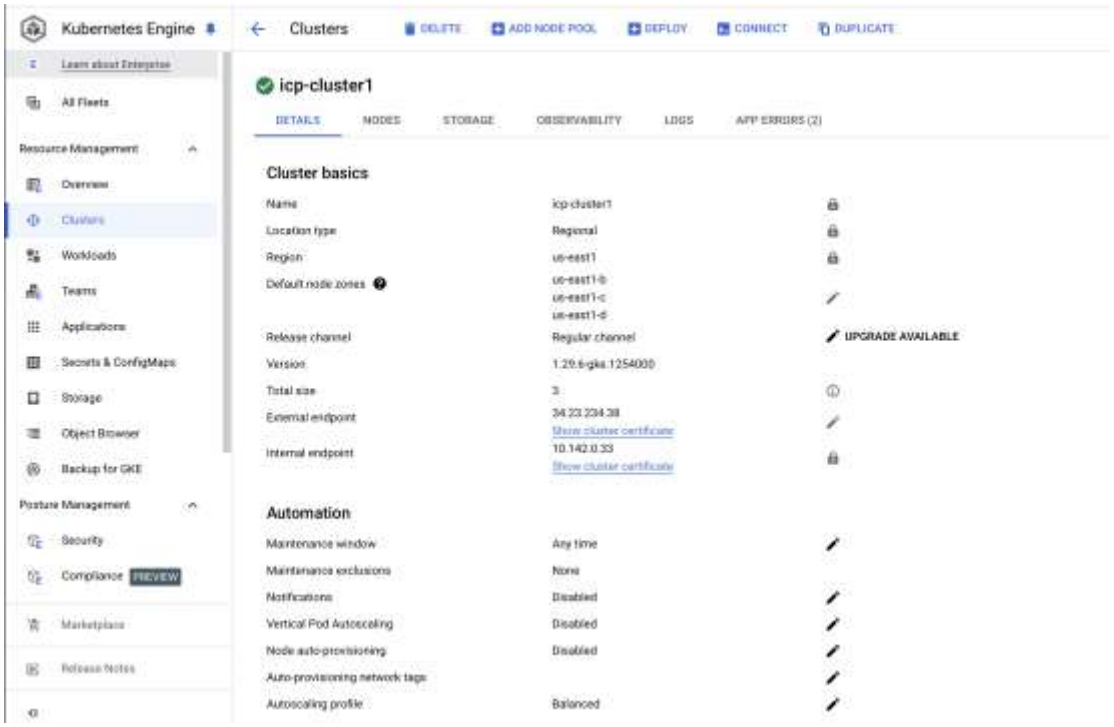
Finalizacion del Despliegue



Nota. Finalizacion del proceso de deploy de infraestructura. *Fuente. Autor*

Figura 26

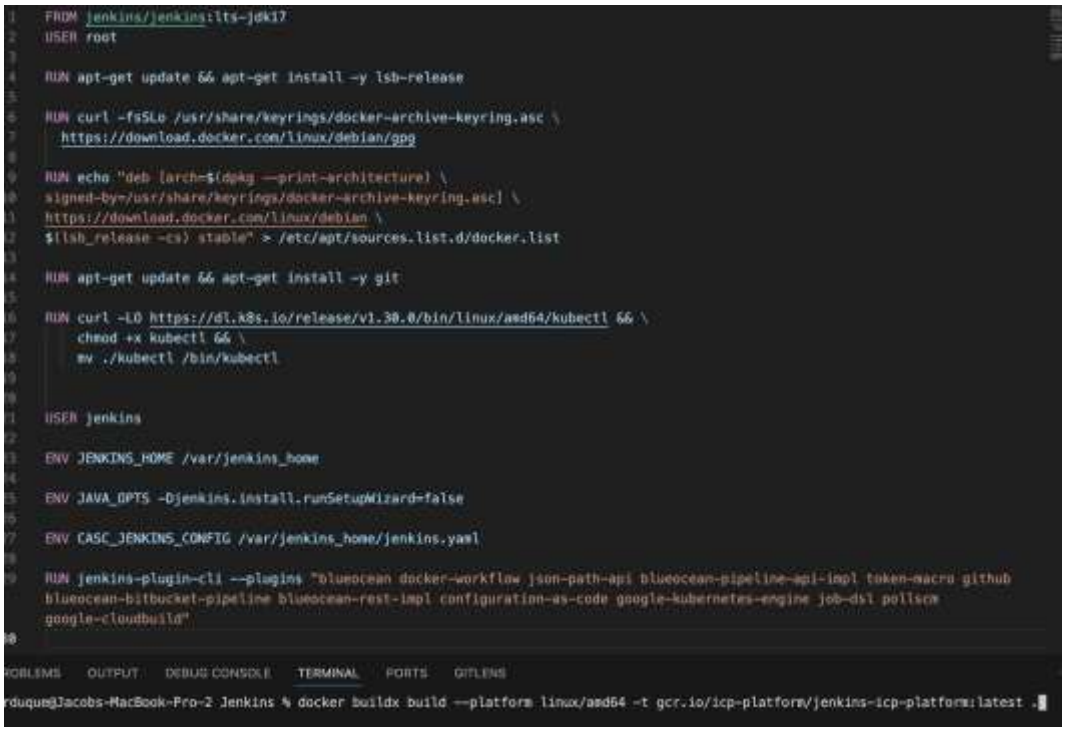
Despliegue de Cluster



Nota. Validacion del cluster desplegado en GCP. *Fuente. Autor*

Figura 27

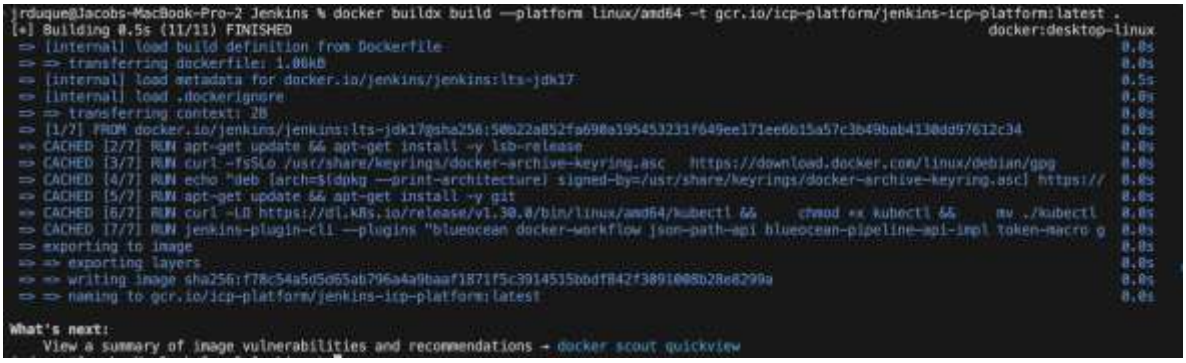
Configuracion de Docker para Jenkins



Nota. Configuracion de Dockerfile para Jenkins a traves de código. *Fuente. Autor*

Figura 28

Compilacion de Imagen para Jenkins



Nota. Compilacion de imagen para jenkins a traves de consola. *Fuente. Autor*

Figura 29

Revisión de Imagen de Jenkins

Nota. Validacion de imagen de jenkins conrrectamente creada. *Fuente. Autor*

Figura 30

Cargue de Imagen de Jenkins al registry

Nota. Proceso de carga de imagen de jenkins al registry. *Fuente. Autor*

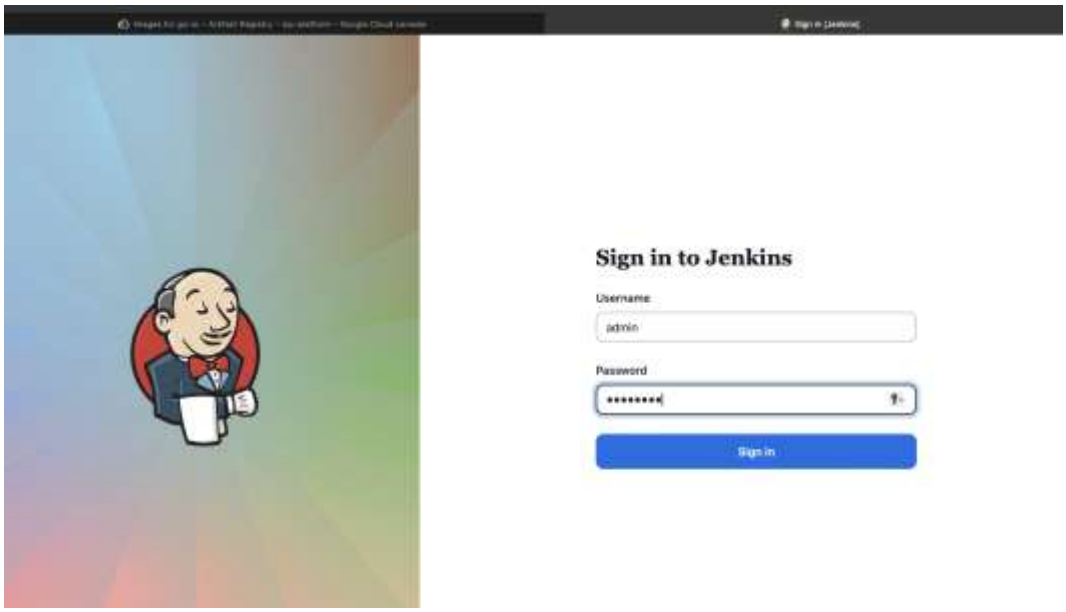
Figura 31

Cargue de servidor Jenkins

Nota. Proceso de carga de servidor Jenkins. *Fuente. Autor*

Figura 32

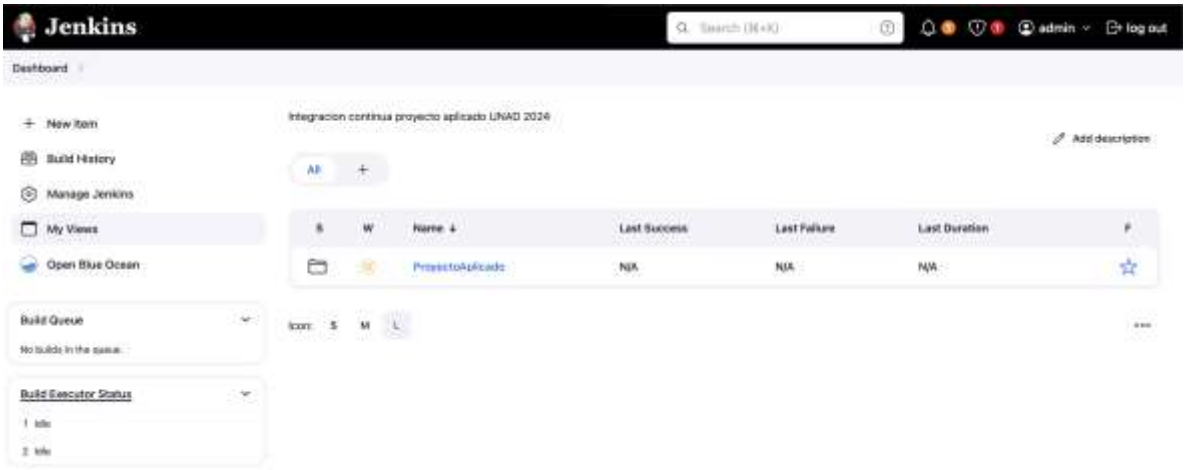
Cargue de Plataforma



Nota. Cargue completo de plataforma y login en Jenkins. *Fuente. Autor*

Figura 33

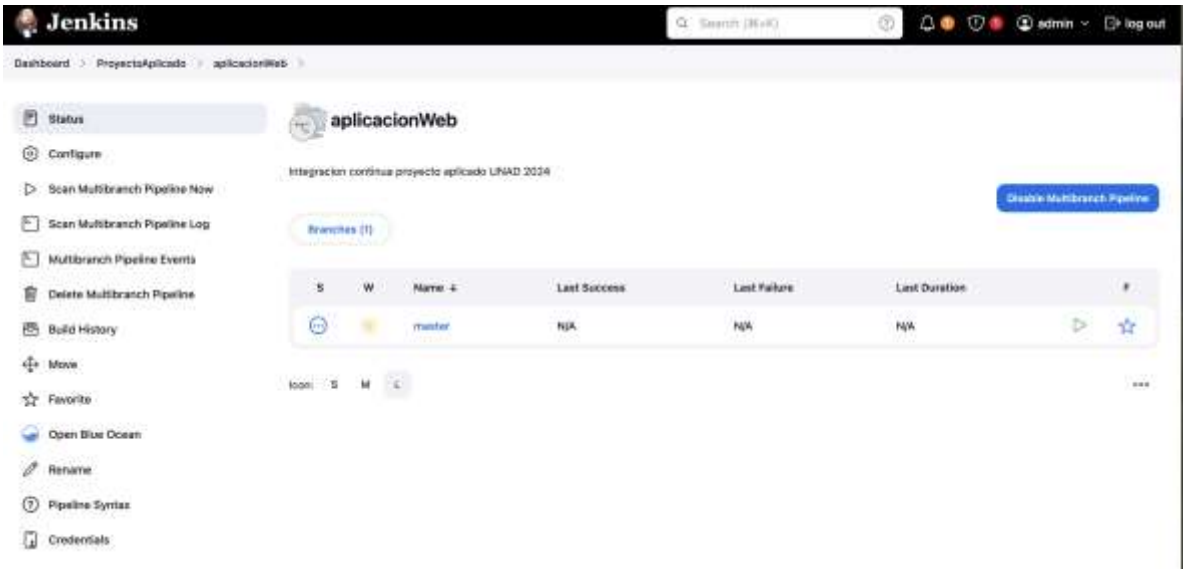
Dashboard Plataforma Jenkins



Nota. Dashboard de plataforma con tareas creadas para la aplicacion de prueba. *Fuente. Autor*

Figura 34

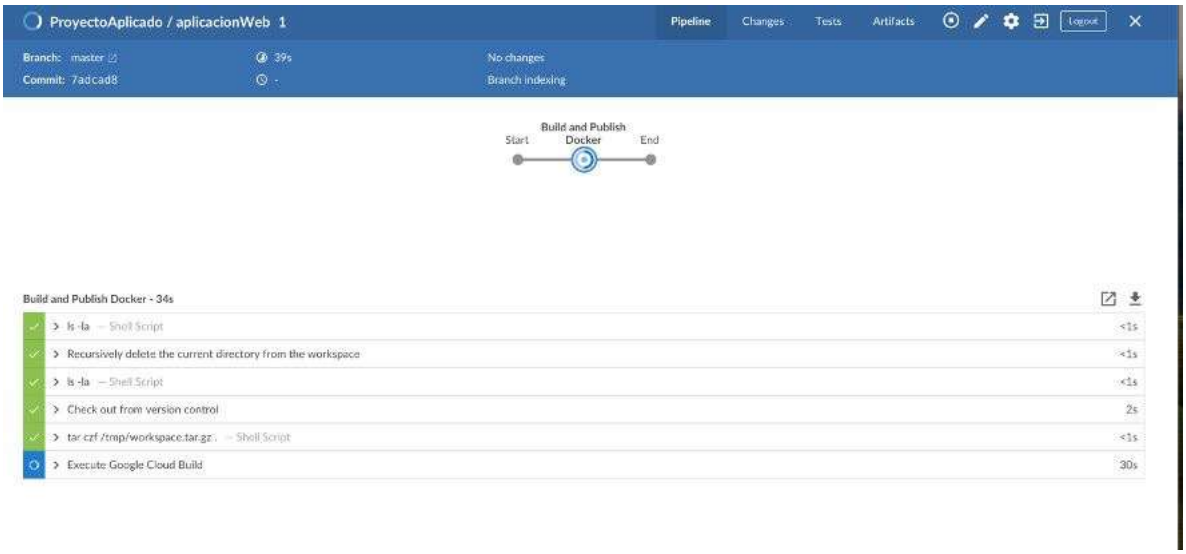
Validacion de Tarea en Jenkins



Nota. Validacion de tarea creada para compilacion y despliegue de aplicacion de prueba. *Fuente. Autor*

Figura 35

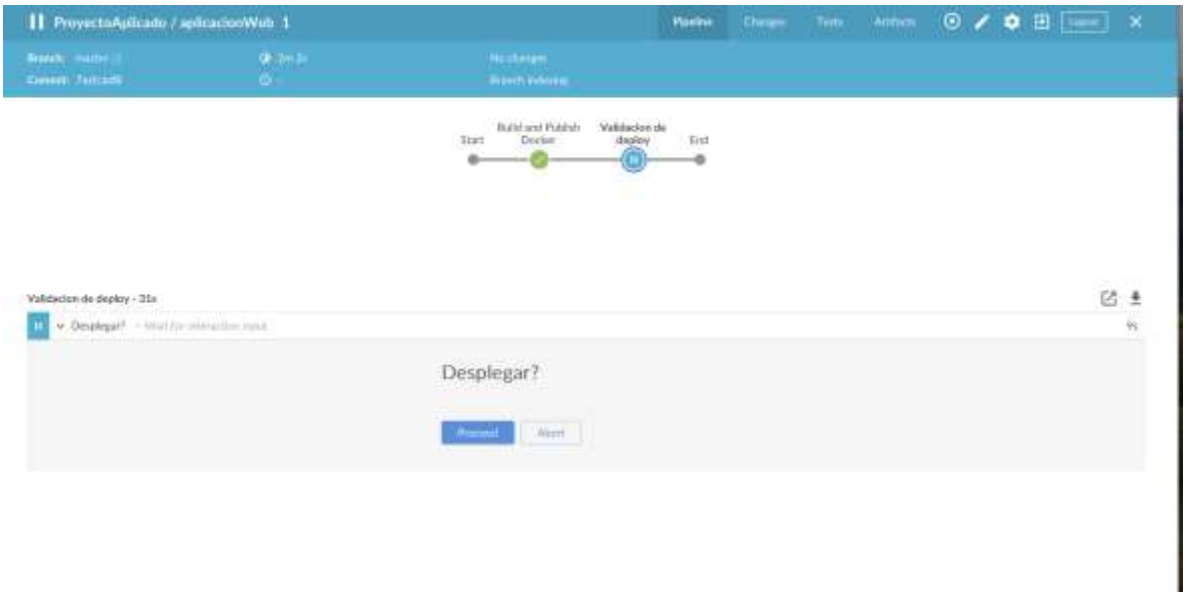
Ejecución primer pipeline



Nota. Ejecucion de primer pipeline de compilacion de aplicación. *Fuente. Autor*

Figura 36

Validacion primer pipeline



Nota. Validacion en el pipeline para desplegar en Jenkins. *Fuente. Autor*

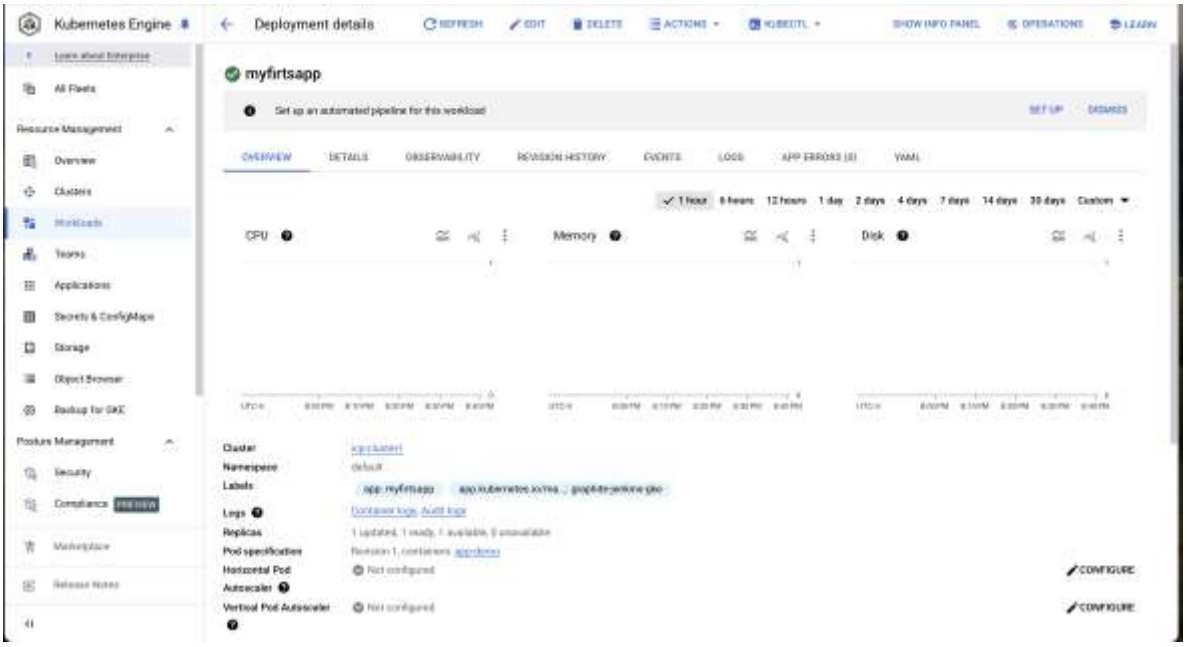
Figura 37

Terminacion del flujo

Nota. Finalizacion del primer flujo de integracion continua en Jenkins. *Fuente. Autor*

Figura 38

Revision de despligue de aplicacion



Nota. Validacion de aplicacion desplegada en cluster de servicio en GCP. *Fuente. Autor*

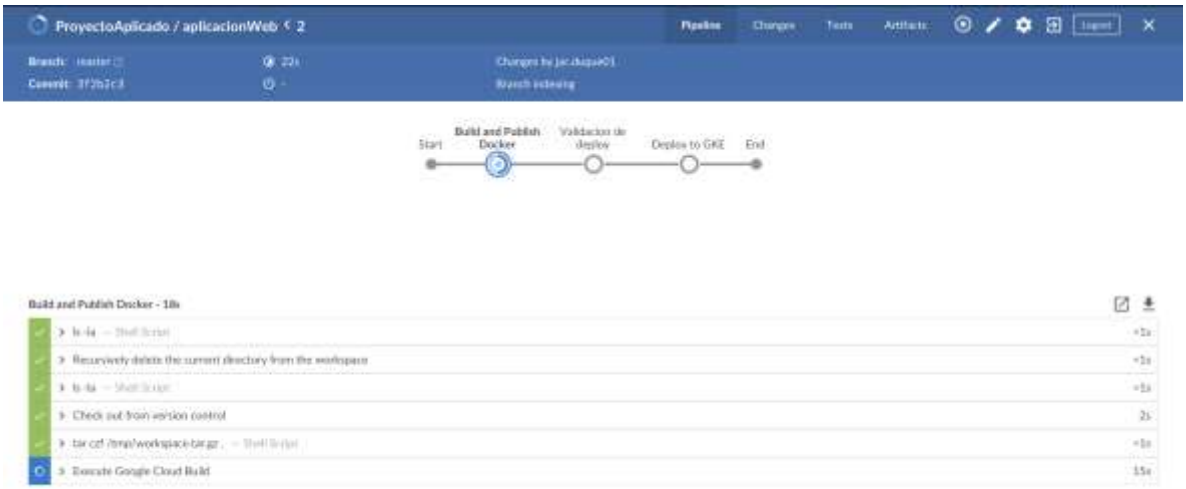
Figura 39

Revision de primera version

Nota. Validacion de primera version de aplicacion desplegada. *Fuente Autor*

Figura 40

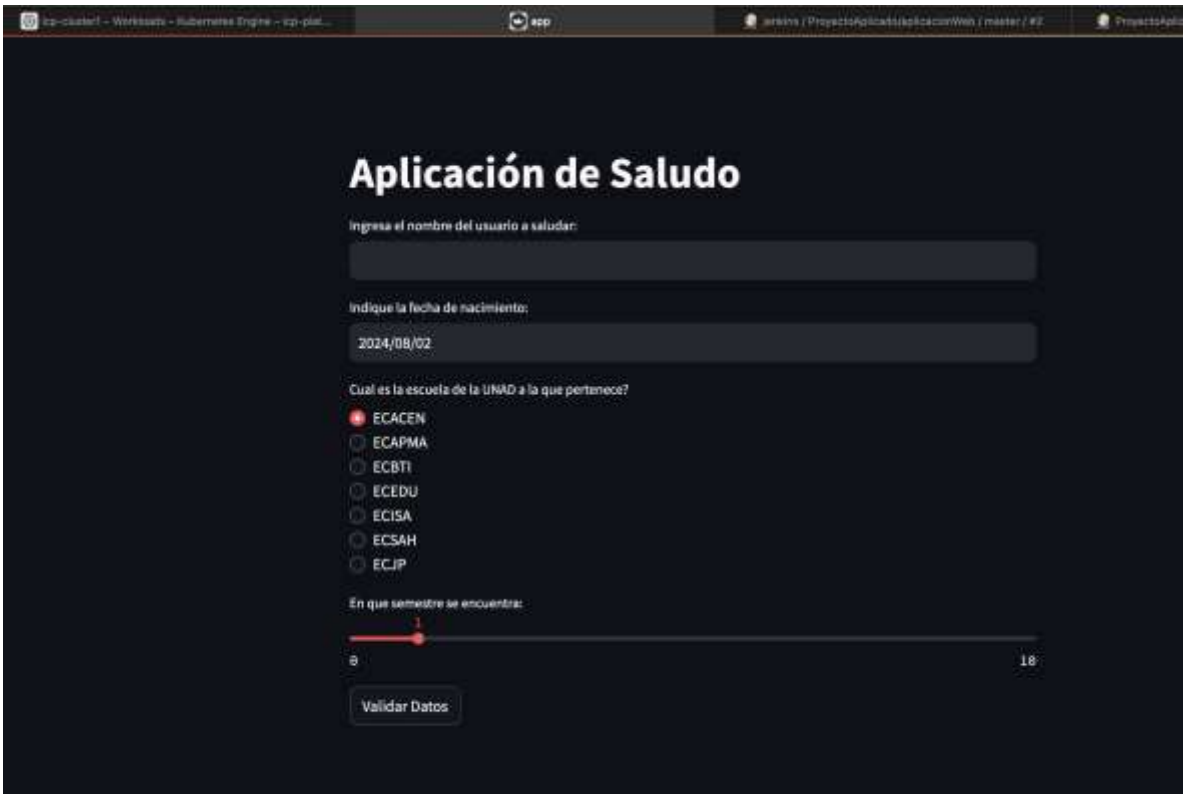
Ejecución segundo pipeline



Nota. Ejecucion de segundo pipeline posterior a cambios en la aplicación. *Fuente. Autor*

Figura 41

Revisión de cambios en Aplicación



Nota. Validacion de cambios en la aplicacion desplegada en la segunda version. *Fuente. Autor*

Resumen de Resultados

Las pruebas descritas cubren los aspectos críticos del despliegue y operación del sistema. La mayoría de los casos de prueba han pasado satisfactoriamente, demostrando que la infraestructura y las herramientas están configuradas y operando correctamente. Las pruebas de escalabilidad y compatibilidad han mostrado que el sistema puede manejar incrementos en la carga de trabajo y resolver problemas de compatibilidad.

Dificultades y soluciones en las pruebas funcionales

Durante la fase de pruebas funcionales, se encontraron varias dificultades que se abordaron y solucionaron de manera efectiva. Un problema significativo surgió al compilar las imágenes de Docker debido a incompatibilidades de arquitectura. Google Kubernetes Engine (GKE) esperaba imágenes compiladas para la arquitectura de Ubuntu, mientras que las imágenes estaban siendo compiladas en una arquitectura M1 de Mac. Este problema se resolvió ejecutando el comando de compilación de Docker con la opción --platform linux/amd64, lo cual garantizó la compatibilidad requerida.

Este ajuste permitió que las imágenes de Docker fueran compatibles con GKE, asegurando que las aplicaciones se desplegaran y ejecutaran correctamente en el clúster de Kubernetes. Este tipo de problemas y soluciones son comunes en proyectos que involucran múltiples plataformas y arquitecturas, y la capacidad para identificar y resolver estos problemas rápidamente es crucial para el éxito del proyecto (Docker Inc, 2023).

Paradigma Utilizado

Para el desarrollo de este proyecto, se ha seguido el **paradigma DevOps**, el cual combina el desarrollo de software (Dev) y las operaciones de TI (Ops) para mejorar la colaboración entre los equipos de desarrollo e infraestructura, así como para automatizar y mejorar el proceso de entrega de software.

Principios del Paradigma DevOps

1. **Colaboración y Comunicación**:
    - El trabajo en conjunto entre los equipos de desarrollo y operaciones es fundamental. Se fomenta la comunicación abierta y el trabajo en equipo para resolver problemas rápidamente.
    - **Ejemplo en el Proyecto**: Los desarrolladores y los ingenieros de sistemas y telecomunicaciones colaboraron estrechamente en la definición de los requisitos y la configuración de la infraestructura.
2. **Automatización**:
    - La automatización de tareas repetitivas y manuales es clave para lograr eficiencia y reducir errores.
    - **Ejemplo en el Proyecto**: Se automatizó el pipeline de CI/CD con Jenkins, permitiendo la construcción, prueba y despliegue automatizados de la aplicación.
3. **Entrega Continua**:

- La capacidad de lanzar nuevas versiones de software de manera rápida y frecuente, asegurando que las mejoras y correcciones lleguen a los usuarios finales en el menor tiempo posible.
- **Ejemplo en el Proyecto**: El uso de Jenkins y Kubernetes permite que nuevas versiones de la aplicación se desplieguen continuamente en un entorno de producción.

Desarrollo del Proceso

Contenerización de la Aplicación

1. **Creación del Dockerfile**:
   - Se desarrolló un Dockerfile que define cómo construir la imagen de Docker de la aplicación. Este archivo incluye instrucciones para instalar dependencias, copiar el código fuente y definir el comando de inicio de la aplicación.
2. **Construcción y Prueba del Contenedor**:
   - Se ejecutaron comandos Docker para construir la imagen y ejecutar el contenedor localmente, asegurando que la aplicación se ejecuta correctamente dentro del contenedor.

Orquestación con Kubernetes

1. **Definición de los Archivos YAML**:

- Se crearon archivos YAML para definir los despliegues y servicios de Kubernetes, especificando detalles como la imagen Docker a usar, el número de réplicas y las configuraciones de red.

2. **Despliegue en GKE**:
   - Se aplicaron los archivos YAML en el clúster de GKE utilizando kubectl, verificando que los pods se crearon y están funcionando correctamente.

Configuración del Pipeline de CI/CD

1. **Configuración del Jenkinsfile**:
   - Se creó un Jenkinsfile que define las etapas del pipeline de CI/CD, desde la construcción y prueba del código hasta la creación de la imagen Docker y el despliegue en Kubernetes.
2. **Integración con GitHub**:
   - Se configuró GitHub para activar el pipeline de Jenkins automáticamente cada vez que se realiza un commit en el repositorio, asegurando una integración continúa fluida.

Conclusiones

En el proyecto hemos llevado a cabo una plataforma de contenerización en Google Cloud Platform (GCP), utilizando métodos de desarrollo de software para conseguir una mayor eficiencia y rapidez en los equipos de desarrollo e infraestructura. A lo largo del desarrollo del mismo, hemos validado diversas herramientas y métodos que no solo apoyan la administración y despliegue de aplicaciones, sino que además aseguran un entorno flexible y escalable para los próximos desarrollos.

La colaboración entre alumnos de carreras de ingenierías de sistemas y de telecomunicación ha sido fundamental para la ejecución de este trabajo. Esta mezcla de conocimientos y habilidades nos ha posibilitado examinar los problemas desde diferentes puntos de vista técnicos, garantizando una visión integral que comprende la creación y administración de infraestructuras además de la manera más eficaz de implementar soluciones de contención y CI/CD.

El sistema creado no sólo ofrece ambientes ligeros y ágiles para los equipos de desarrollo, sino que además posibilita a las compañías disminuir los costos y aumentan la eficiencia al evitar desperdicios en infraestructura física. Los equipos de infraestructura pueden trabajar en conjunto con los equipos de integración continua, debido a que conocen las herramientas preliminares que utilizan.

Además, la habilidad de reproducir ambientes con el fin de pruebas y actualizaciones añade una importancia fundamental, esto le da a los desarrolladores y a los usuarios finales una oportunidad de validar e identificar la funcionalidad antes de que este sea puesto en un entorno productivo.

Referencias

Ansible Collaborative. (2024). *How Ansible works*. https://www.ansible.com/overview/how-ansible-works

AWS. (2024a, junio). *¿Qué es la computación en la nube para móviles?* https://aws.amazon.com/es/what-is/mobile-cloud-computing/

AWS. (2024b, junio). *¿Qué es la infraestructura como código? - Explicación de IaC*. https://aws.amazon.com/es/what-is/iac/

Belcastro, L., Cosentino, C., & Marozzo, F. (2024). Infrastructures for High-Performance Computing: Cloud Infrastructures. En *Reference Module in Life Sciences*. Elsevier. https://doi.org/10.1016/B978-0-323-95502-7.00006-3

Chouliaras, S., & Sotiriadis, S. (2022). Auto-scaling containerized cloud applications: A workload-driven approach. *Simulation Modelling Practice and Theory*, *121*, 102654. https://doi.org/10.1016/j.simpat.2022.102654

Continuous Delivery Foundation. (2024). *Jenkins User Documentation*. https://www.jenkins.io/doc/

Docker Inc. (2023). *Docker buildx build*. https://docs.docker.com/reference/cli/docker/buildx/build/

Docker Inc. (2024). *What is a container?* https://www.docker.com/what-docker

Friyanto, A. (2023). Automation deployment analysis in simulation network infrastructure as code. *AIP Conference Proceedings*, *2510*(1), 1-4. https://doi.org/10.1063/5.0130295

Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J. M., Caro-

Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., & Panojo, F. S. (2024). Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software*, *208*, 111906. https://doi.org/10.1016/j.jss.2023.111906

Google Cloud. (2024a). *Cloud Storage*. https://cloud.google.com/storage

Google Cloud. (2024b). *Containers in Cloud*. https://cloud.google.com/containers?hl=en

Google Cloud. (2024c). *Google Cloud Platform*. https://cloud.google.com/?hl=en

Google Cloud. (2024d). *Google Compute Engine*. https://cloud.google.com/products/compute?hl=en

Google Cloud. (2024e). *Google Kubernetes Engine (GKE)*. https://cloud.google.com/kubernetes-engine?hl=en

Google Cloud. (2024f). *Using Recommendations for Infrastructure as Code*. https://cloud.google.com/recommender/docs/tutorial-iac

Grande, R., Vizcaíno, A., & García, F. O. (2024). Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. *Computer Standards & Interfaces*, *87*, 103767. https://doi.org/10.1016/j.csi.2023.103767

HashiCorp Developer. (2024a). *DECATHLON - CUSTOMER CASE STUDY*. https://www.hashicorp.com/assets/1613776684-hcdecathloncasestudydigital.pdf

HashiCorp Developer. (2024b). *HashiCorp—On top of their game*. https://www.hashicorp.com/case-studies/decathlon?product_intent=terraform

HashiCorp Developer. (2024c). *Terraform provider for Google Cloud*. https://www.terraform.io/docs/providers/google/index.html

HashiCorp Developer. (2024d). *What is Terraform?* https://developer.hashicorp.com/terraform/intro

IBM. (2024). *Que es la contenerización?* https://www.ibm.com/es-es/topics/containerization

Jenkins. (2024). *Jenkins*. https://www.jenkins.io/

kubernetes.io. (2017). *Case study:Booz Allen Hamilton*. https://kubernetes.io/case-studies/booz-allen/

Mishra, A., & Otaiwi, Z. (2020). DevOps and software quality: A systematic mapping. *Computer Science Review*, *38*, 100308. https://doi.org/10.1016/j.cosrev.2020.100308

Red Hat, Inc. (2021, marzo 25). *Employers centralizes insurance apps on Red Hat OpenShift*. https://www.redhat.com/en/resources/employers-case-study

Red Hat, Inc. (2022a, octubre 5). *El concepto de DevOps*. https://www.redhat.com/es/topics/devops

Red Hat, Inc. (2022b, noviembre 5). *La integración y la distribución continuas (CI/CD)*. https://www.redhat.com/es/topics/devops/what-is-ci-cd

Red Hat, Inc. (2023, septiembre 20). *Understanding automation*. https://www.redhat.com/en/topics/automation

safetica. (2024, junio). *Seguridad de datos en la nube*. http://www.safetica.com/

Sentrio Labs S.L. (2021, septiembre 16). *Introducción a Jenkins: ¿qué es, para qué sirve y cómo funciona?* https://sentrio.io/blog/que-es-jenkins/

Software Freedom Conservancy. (2024). *GIT*. https://git-scm.com/

The Linux Foundation. (2024). *¿Qué es Kubernetes?* https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/