Facultad de Tecnologías Interactivas

Estrategia para el despliegue de servicios en la nube mediante la utilización de herramientas de Infraestructura como Código (IaC)

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Leandro de la Paz Cabrera

Tutor: M. Sc. Gilberto Arias Naranjo

La Habana, diciembre de 2024

DECLARACIÓN DE AUTORÍA

El autor del trabajo de diploma con título **"*Estrategia para el despliegue de servicios en la nube mediante la utilización de herramientas de Infraestructura como Código*"** concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara como único autor de su contenido. Para que así conste firma la presente a los 3 días del mes de diciembre del año 2024.

Leandro de la Paz Cabrera

_____
Firma del Autor

P. As., Lic. Gilberto Arias Naranjo, M.Sc.

_____
Firma del Tutor

DATOS DE CONTACTO

Tutor: P. As., Lic. Gilberto Arias Naranjo, M.Sc.

Graduado de Licenciatura en Ciencias de la Computación de la Universidad Central "Martha Abreu" en el año 2005 y de Máster en Bioinformática en el año 2008. Ostenta la categoría de profesor asistente e impartió clases en la Universidad de Ciencias Informáticas durante 15 años. En la actualidad es Programador Principal y Líder de Proyectos en Dofleini S.R.L.

Correo electrónico: gilberto@dofleinisoftware.com.

AGRADECIMIENTOS

A mi tutor de tesis, por su valiosa guía, paciencia y apoyo en este proceso. Su compromiso y disposición para ofrecer su ayuda han sido esenciales para alcanzar este logro.

A mi familia, amigos y compañeros de trabajo, quienes me alentaron en cada paso del camino.

Un agradecimiento especial a Dofleini S.R.L, empresa que me acogió entre sus filas y ha jugado un papel crucial en mi formación profesional. Es en el seno de esta organización donde se gestó el tema de esta tesis, y estoy profundamente agradecido por la oportunidad de crecer y desarrollarme en un ambiente que fomenta la innovación y el aprendizaje continuo.

RESUMEN

El presente trabajo desarrolla una estrategia para el despliegue de servicios en la nube utilizando herramientas de Infraestructura como Código (IaC), abordando los desafíos actuales de escalabilidad, trazabilidad y consistencia en la gestión de infraestructuras modernas. Se emplearon tecnologías como Terraform, GitLab y Docker, integradas en una propuesta que optimiza la automatización y la reproducibilidad de la infraestructura. Terraform permite gestionar recursos en plataformas de múltiples proveedores, mientras que GitLab asegura un control efectivo de versiones y la ejecución de flujos CI/CD para la infraestructura y las aplicaciones en contenedores Docker. Los resultados obtenidos demuestran la efectividad de esta estrategia para reducir errores manuales, mejorar tiempos de despliegue y garantizar una infraestructura ágil, adaptable y segura para entornos de nube dinámicos.

Palabras clave: Automatización, CI/CD, Despliegue en la nube, Docker, GitLab, Infraestructura como Código, Terraform.

Abstract

This work develops a strategy for deploying cloud services using Infrastructure as Code (IaC) tools, addressing current challenges of scalability, traceability, and consistency in modern infrastructure management. Technologies such as Terraform, GitLab, and Docker were employed, integrated into a proposal that optimizes automation and reproducibility of infrastructure. Terraform enables resource management across multi-provider platforms, while GitLab ensures effective version control and the execution of CI/CD pipelines for infrastructure and containerized applications. The results demonstrate the effectiveness of this strategy in reducing manual errors, improving deployment times, and ensuring an agile, adaptable, and secure infrastructure for dynamic cloud environments.

Keywords: *Automation, CI/CD, Cloud Deployment, Docker, GitLab, Infrastructure as Code, Terraform.*

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo de software ha transformado profundamente la forma en que las organizaciones abordan sus procesos y servicios. Las aplicaciones informáticas se han convertido en el pilar de muchas operaciones empresariales, desde sistemas de gestión interna hasta plataformas de servicio al cliente. Sin embargo, el desarrollo de estas aplicaciones no se limita únicamente a su diseño y programación, sino que también involucra un paso crítico: su implementación y despliegue en infraestructuras que permitan su ejecución de manera eficiente y confiable.

Históricamente, estas aplicaciones se desplegaban en infraestructuras locales o in-situ, donde las organizaciones eran responsables de adquirir, configurar y mantener los recursos físicos necesarios para ejecutar sus sistemas. Este enfoque, aunque funcional en su momento, trae consigo desventajas significativas: los altos costos iniciales asociados a la compra de hardware, el mantenimiento continuo de equipos, la necesidad de personal especializado y la limitada escalabilidad. En un entorno tecnológico donde la velocidad y la flexibilidad son cruciales, estas limitaciones han llevado a buscar alternativas más eficientes.

En este contexto, los proveedores de servicios en la nube, como Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP), han ganado protagonismo. La computación en la nube permite a las organizaciones acceder a recursos como almacenamiento, redes y capacidad de procesamiento de manera remota y bajo demanda. Las ventajas son claras: eliminación de la inversión inicial en hardware, escalabilidad casi instantánea, flexibilidad para adaptar recursos según las necesidades y modelos de pago basados en el uso, que optimizan los costos operativos (Voorsluys, Broberg y Buyya, 2011). Estas características han convertido a la nube en una opción preferida para la implementación de aplicaciones informáticas.

Sin embargo, a medida que las aplicaciones evolucionan y adoptan arquitecturas más dinámicas y distribuidas, como las basadas en microservicios, su gestión se vuelve cada vez más compleja. La infraestructura que soporta estas aplicaciones debe ser capaz de

adaptarse rápidamente a cambios constantes, lo que añade un nivel significativo de dificultad. Cuando esta gestión se realiza de forma manual, no solo se incrementa el tiempo y el esfuerzo requeridos, sino que también se vuelve propensa a errores humanos. Estas fallas pueden derivar en inconsistencias entre entornos, interrupciones en los servicios y pérdida de trazabilidad en los cambios realizados (Morris, 2021).

En este escenario, se define como **problema de investigación** ¿Cómo garantizar un proceso de despliegue de infraestructuras en la nube que sea ágil, trazable, reproducible y adaptable a las demandas de aplicaciones modernas, reduciendo los riesgos asociados con la gestión manual y evitando la dependencia exclusiva de un único proveedor de nube?

El **objeto de estudio** propuesto es el proceso de despliegue de infraestructuras en la nube.

El **campo de acción** se centra en el diseño e implementación de estrategias de despliegue de infraestructuras en la nube basadas en principios de infraestructura como código (IaC).

Se define como **objetivo general:** Implementar una estrategia de despliegue de infraestructura en la nube para aplicaciones modernas mediante el uso de infraestructura como código, garantizando que sea un proceso ágil y reproducible.

Para alcanzar el objetivo general, se plantean las siguientes tareas de investigación:

- Realizar un análisis de los conceptos y fundamentos de la infraestructura como código y la computación en la nube, destacando sus ventajas y desafíos.
- Comparar diferentes herramientas de infraestructura como código para identificar características y ventajas que sean relevantes para el despliegue de infraestructuras en la nube.
- Diseñar una arquitectura de infraestructura en la nube adecuada para aplicaciones modernas, detallando los componentes esenciales, como servidores, redes, bases de datos y balanceadores de carga.
- Desarrollar una estrategia de despliegue modular basada en principios de infraestructura como código que facilite la escalabilidad y la administración de la infraestructura.

- Implementar la infraestructura diseñada y realizar pruebas orientadas a confirmar el funcionamiento de los componentes desplegados, evaluando aspectos clave como la conectividad, la respuesta de los servicios, la escalabilidad de los recursos y la resiliencia ante fallos.
- Analizar los resultados obtenidos en las pruebas para verificar que la infraestructura desplegada cumpla con los requisitos de funcionamiento y estabilidad, formulando recomendaciones y posibles mejoras en la estrategia de despliegue de IaC.

Métodos científicos utilizados en la investigación

Para alcanzar los objetivos de esta investigación, se han empleado una serie de métodos científicos que permiten abordar el problema de forma integral, desde el análisis conceptual hasta la validación práctica de la propuesta. Estos métodos se clasifican en teóricos y empíricos, y su aplicación busca garantizar un enfoque riguroso y fundamentado.

1. Métodos Teóricos

- **Análisis y síntesis:** Este método se emplea para descomponer y estudiar los conceptos de la IaC, la computación en la nube y las prácticas de despliegue en estas. El análisis permite desglosar los componentes críticos de estas tecnologías y comprender sus interacciones, mientras que la síntesis facilita la integración de estos conocimientos para diseñar una solución coherente y adaptada a los requisitos de la nube.
- **Comparación:** La comparación se utiliza para evaluar diversas herramientas de infraestructura como código, identificando sus ventajas y desventajas en el contexto de despliegue en la nube. Este método permite justificar la elección de la herramienta que mejor se adapte a los requerimientos de flexibilidad, escalabilidad y trazabilidad necesarios para el proyecto.
- **Modelado:** A través del modelado, se construyen representaciones de la arquitectura de infraestructura y del flujo de despliegue. Este método permite visualizar y planificar los componentes de la solución propuesta, facilitando la identificación de posibles problemas y ajustes antes de la implementación real.

2. Métodos Empíricos

- **Pruebas experimentales:** Este método se aplica en la fase de validación de la infraestructura propuesta. Consiste en ejecutar pruebas específicas de seguridad y consistencia de la configuración, evaluando aspectos clave como restricciones de acceso, cifrado de datos y ejecución de flujos automatizados en GitLab CI/CD. Estas pruebas permiten verificar el correcto funcionamiento de la infraestructura y su capacidad para gestionar recursos de forma eficiente bajo condiciones controladas.

- **Observación directa:** La observación directa se utiliza durante el proceso de despliegue y validación de la infraestructura. Este método implica registrar y analizar el comportamiento del sistema en tiempo real, identificando patrones de uso, posibles errores y aspectos que requieren ajustes. La observación directa facilita la recolección de datos precisos sobre el desempeño y la estabilidad de la infraestructura.

- **Evaluación comparativa de resultados:** Una vez implementada y probada la infraestructura, se realiza una evaluación comparativa de los resultados obtenidos en las pruebas. Esto permite contrastar el desempeño del sistema con los objetivos planteados, identificando áreas de mejora y validando la efectividad de la estrategia de despliegue.

El presente trabajo comprende tres capítulos:

Capítulo 1: Fundamentación teórica, se presentan los conceptos básicos de computación en la nube, infraestructura como código y sus ventajas para entornos empresariales.

Capítulo 2: Propuesta de solución, describe la arquitectura de infraestructura propuesta para una plataforma de comercio electrónico, así como el diseño de una estrategia de despliegue en la nube que cumpla con los principios de IaC.

Capítulo 3: Validación de la solución propuesta, se realiza la validación de la propuesta mediante pruebas experimentales y análisis de resultados. Se evalúan el desempeño, la escalabilidad y la seguridad de la infraestructura desplegada, contrastando los resultados con los objetivos iniciales.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Este capítulo establece los fundamentos teóricos necesarios para comprender el contexto y los conceptos clave relacionados con el despliegue de infraestructuras en la nube. Se inicia con una exploración detallada de la computación en la nube, abarcando sus modelos de servicio, características principales y beneficios, así como los desafíos asociados con su adopción. Este análisis permite destacar la importancia de infraestructuras escalables, seguras y flexibles en entornos tecnológicos dinámicos.

A continuación, se introduce el concepto de Infraestructura como Código (IaC), un enfoque que ha transformado la gestión de recursos tecnológicos al permitir que estos se configuren y desplieguen de manera reproducible y automatizada. Este paradigma resulta esencial para superar las limitaciones de los métodos tradicionales, especialmente en cuanto a trazabilidad y consistencia.

Asimismo, se analizan las principales estrategias de despliegue en la nube y las herramientas que soportan IaC, evaluando sus capacidades y limitaciones. Finalmente, el capítulo aborda los componentes fundamentales de una infraestructura típica, con énfasis en su integración para garantizar alta disponibilidad, seguridad y rendimiento. Este marco teórico proporciona los elementos necesarios para fundamentar las decisiones metodológicas y técnicas que se abordarán en los capítulos siguientes.

1.1 Computación en la nube

Definición y características

La computación en la nube es un modelo de tecnología que permite el acceso a recursos de procesamiento, almacenamiento y redes a través de Internet, sin la necesidad de poseer físicamente la infraestructura. En lugar de gestionar centros de datos y servidores propios, las empresas y usuarios pueden rentar estos recursos y utilizarlos en función de sus necesidades, accediendo a ellos de manera remota y pagando únicamente por lo que consumen (Marinescu, 2017; Voorsluys, Broberg y Buyya, 2011).

Este enfoque ofrece varias características clave:

- **Elasticidad:** La computación en la nube permite ajustar los recursos de manera dinámica, aumentando o disminuyendo la capacidad según la demanda. Esto es fundamental para responder a picos de actividad o reducir el consumo durante periodos de menor demanda, lo cual es especialmente útil en sectores como el comercio electrónico y los servicios en línea (Armbrust et al., 2010).
- **Escalabilidad:** A diferencia de la infraestructura tradicional, donde la expansión de capacidad implica inversiones en hardware y tiempo de configuración, la nube permite escalar los servicios fácilmente, tanto verticalmente (aumentando la capacidad de los recursos existentes) como horizontalmente (añadiendo nuevos recursos). Este escalado inmediato es vital para aplicaciones que requieren alta disponibilidad y desempeño constante (Buyya, Vecchiola y Selvi, 2013).
- **Pago por uso:** Uno de los aspectos más atractivos de la computación en la nube es su modelo de pago. Los usuarios pagan solo por los recursos que realmente consumen, en lugar de realizar inversiones significativas en infraestructura. Esto permite a las empresas optimizar sus costos y acceder a infraestructura avanzada sin grandes desembolsos iniciales, beneficiándose de un modelo económico flexible (Voorsluys et al., 2011).
- **Flexibilidad y accesibilidad:** La nube permite el acceso a recursos desde cualquier lugar y dispositivo con conexión a Internet, facilitando un entorno de trabajo remoto y colaborativo. Esta accesibilidad es especialmente importante para empresas con equipos distribuidos y para aquellas que necesitan mantener una continuidad operativa (Marinescu, 2017).
- **Gestión simplificada y automatización:** En lugar de tener que configurar y gestionar servidores y redes de forma manual, los proveedores de nube ofrecen herramientas y servicios que automatizan muchos de estos procesos, reduciendo la carga operativa. Esto permite que las empresas se enfoquen en su actividad principal y no en el mantenimiento de la infraestructura (Buyya et al., 2013).

A diferencia de la infraestructura tradicional, donde los recursos son limitados y los costos de expansión son elevados, la computación en la nube ofrece una flexibilidad incomparable. Las empresas pueden ajustar sus necesidades de infraestructura casi instantáneamente, sin necesidad de adquirir ni configurar hardware adicional. Además, gracias al pago por uso, pueden optimizar su presupuesto en función de su actividad, evitando costos fijos y favoreciendo una estructura de costos variable.

Modelos de servicio en la nube

La computación en la nube ofrece varios modelos de servicio que se adaptan a diferentes necesidades de las organizaciones. Los modelos principales son Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS). Cada uno de estos modelos proporciona un nivel de abstracción distinto y se ajusta a diferentes casos de uso según los requisitos de infraestructura, control y administración que demande la organización (Marinescu, 2017; Buyya, Vecchiola y Selvi, 2013).

- **Infraestructura como Servicio (IaaS):** Este modelo proporciona los componentes básicos de infraestructura, como servidores, almacenamiento y redes, que los usuarios pueden administrar y configurar según sus necesidades. Con IaaS, las empresas pueden controlar el sistema operativo, las aplicaciones y el almacenamiento sin necesidad de invertir en hardware físico. Es ideal para organizaciones que requieren un alto grado de control sobre su infraestructura y buscan una solución escalable y flexible sin encargarse del mantenimiento del hardware.
- **Plataforma como Servicio (PaaS):** PaaS ofrece un entorno de desarrollo completo en la nube que incluye el sistema operativo, bases de datos y herramientas de programación. Los desarrolladores pueden utilizar esta plataforma para construir, probar y desplegar aplicaciones sin preocuparse por la infraestructura subyacente.
- **Software como Servicio (SaaS):** En el modelo SaaS, los proveedores ofrecen aplicaciones listas para usar a través de Internet. Los usuarios pueden acceder a estas aplicaciones desde cualquier dispositivo conectado, sin necesidad de instalar o mantener software.

A continuación se presenta una tabla comparativa de los tres modelos de servicios principales :

Tabla 1: Principales modelos de servicio en la nube [Fuente: Elaboración propia]

Característica	**IaaS (Infraestructura como Servicio)**	**PaaS (Plataforma como Servicio)**	**SaaS (Software como Servicio)**
Nivel de control del usuario	Alto: el usuario administra el sistema operativo, aplicaciones, y configuraciones de red.	Medio: el usuario gestiona las aplicaciones y datos, mientras que el proveedor administra la infraestructura y el sistema operativo.	Bajo: el proveedor gestiona todo, incluyendo la aplicación, el sistema operativo y la infraestructura.
Responsabilidades del proveedor	Solo la infraestructura física (hardware, almacenamiento y red), con soporte en el acceso a los recursos.	Infraestructura y sistema operativo, así como el entorno de desarrollo, bases de datos y middleware.	Toda la infraestructura y la aplicación, incluyendo actualizaciones, seguridad, y mantenimiento.
Uso común	Empresas que necesitan un alto grado de control sobre sus aplicaciones y sistemas operativos, y que buscan una infraestructura escalable sin gestionar hardware.	Desarrollo y despliegue de aplicaciones sin preocuparse por la administración de la infraestructura subyacente.	Aplicaciones empresariales y colaborativas, como correo electrónico, CRM, y software de productividad.
Ventajas	Gran control y flexibilidad; posibilidad de ajustar recursos según demanda; reducción de costos al evitar compra de hardware.	Simplificación del proceso de desarrollo; ahorro de tiempo y recursos en la gestión de infraestructura; entorno optimizado para desarrollo.	Menores costos iniciales; accesibilidad desde cualquier dispositivo con Internet; mantenimiento, actualizaciones y respaldo a cargo del proveedor.
Limitaciones	Requiere conocimientos técnicos para la gestión; la organización es responsable de la seguridad y el mantenimiento del sistema operativo y aplicaciones.	Menos control sobre el sistema operativo y los recursos; dependencia del proveedor para disponibilidad del entorno de desarrollo.	Poco control sobre la infraestructura y el software; el usuario depende del proveedor para cambios, actualizaciones y personalización de la aplicación.
Ejemplos	Amazon EC2, Google Compute Engine, Microsoft Azure Virtual Machines	Google App Engine, Microsoft Azure App Service, Heroku	Google Workspace, Salesforce, Microsoft Office 365

Se decide utilizar una combinación de los modelos IaaS y PaaS basado en la necesidad de flexibilidad y eficiencia en el despliegue de infraestructuras en la nube. Al emplear IaaS como base, se obtiene un alto grado de control sobre la infraestructura, permitiendo personalizar configuraciones específicas de servidores, redes y almacenamiento. Esto es ideal para gestionar aplicaciones críticas y asegurar que los recursos se ajusten a las necesidades del sistema de forma escalable.

Por otro lado, incorporar PaaS para ciertos servicios específicos, como bases de datos gestionadas o almacenamiento en caché, permite simplificar tareas de administración, seguridad y mantenimiento. PaaS libera a los equipos de la gestión directa de estos componentes, ya que el proveedor garantiza su rendimiento y disponibilidad, lo que permite un enfoque más directo en el desarrollo y la optimización de la aplicación. Esta combinación equilibra el control con la eficiencia operativa, optimizando los recursos y acelerando el despliegue de servicios en un entorno ágil y flexible.

Desafíos del despliegue en la Nube

A pesar de sus numerosas ventajas, el despliegue de infraestructura en la nube plantea varios desafíos técnicos y estratégicos que las organizaciones deben considerar para lograr una implementación exitosa y sostenible. Estos desafíos incluyen la gestión de recursos distribuidos, la seguridad de los datos y el riesgo de dependencia de un solo proveedor (*vendor lock-in*), entre otros.

- **Gestión de recursos distribuidos:** La infraestructura en la nube permite el despliegue de recursos distribuidos en diferentes ubicaciones geográficas, lo cual facilita la escalabilidad y la disponibilidad global. Sin embargo, administrar recursos distribuidos introduce complejidades en términos de sincronización, monitoreo y administración de datos.
- **Seguridad de los datos:** La seguridad de los datos en la nube es una de las principales preocupaciones para las empresas, ya que implica confiar en un proveedor externo para almacenar y proteger información sensible. Los desafíos de seguridad

incluyen el control de acceso, la protección de datos en tránsito y en reposo, y la prevención de ataques cibernéticos.

- **Dependencia de un proveedor específico (*vendor lock-In*):** La adopción de un proveedor de nube específico puede llevar a la dependencia de sus servicios y herramientas, lo que limita la flexibilidad de la organización para migrar a otro proveedor en el futuro. Esta dependencia, conocida como *vendor lock-in*, puede ser problemática si el proveedor cambia sus políticas de precios, servicios o seguridad. Para mitigar este riesgo, es importante diseñar la infraestructura con una estrategia *multi-cloud* o adoptar prácticas que faciliten la portabilidad de sus aplicaciones y datos entre diferentes proveedores (Armbrust et al., 2010).
- **Escalabilidad y adaptabilidad de la infraestructura**: La nube ofrece la posibilidad de escalar los recursos en función de la demanda, pero diseñar una infraestructura que sea verdaderamente escalable y adaptable requiere una planificación cuidadosa. Los sistemas deben estar diseñados para manejar aumentos repentinos en la carga sin degradar su rendimiento, y deben poder adaptarse a diferentes entornos de trabajo. Esto exige una arquitectura flexible y modular que permita añadir o reducir componentes según las necesidades sin afectar la integridad del sistema (Buyya, Vecchiola y Selvi, 2013).

Estos desafíos resaltan la importancia de contar con una infraestructura en la nube escalable, segura y adaptable. La gestión de recursos distribuidos y la implementación de políticas de seguridad adecuadas son esenciales para asegurar el buen funcionamiento de las aplicaciones y la protección de los datos. Además, reducir la dependencia de un solo proveedor y diseñar una infraestructura que pueda escalar de acuerdo con la demanda permite a las organizaciones aprovechar al máximo los beneficios de la computación en la nube mientras minimizan los riesgos asociados.

1.2 Infraestructura como Código

Con el crecimiento de la computación en la nube y la demanda de infraestructuras flexibles y escalables, IaC ha emergido como una práctica esencial para la gestión eficiente de los

recursos de TI. IaC permite definir, desplegar y gestionar la infraestructura mediante código, de manera similar a cómo se desarrollan y versionan las aplicaciones. Esto significa que, en lugar de configurar manualmente cada servidor o recurso, las organizaciones pueden emplear archivos de configuración para automatizar y estandarizar el aprovisionamiento y la administración de infraestructura.

En las siguientes secciones, se abordarán la definición y los principios de IaC, sus ventajas y los componentes clave.

Definición y principios de la IaC

La Infraestructura como Código (IaC) es una práctica que permite gestionar y aprovisionar recursos de infraestructura mediante archivos de configuración, eliminando configuraciones manuales y permitiendo el despliegue de infraestructura de forma automática y consistente. Según Morris (2021), “IaC permite que la infraestructura se gestione de manera similar al código de software, proporcionando un enfoque reproducible y automatizable para el despliegue de entornos de TI”. Al definir la infraestructura en código, es posible reproducir entornos de manera controlada y escalable, lo que facilita el despliegue y la administración de recursos en entornos de nube.

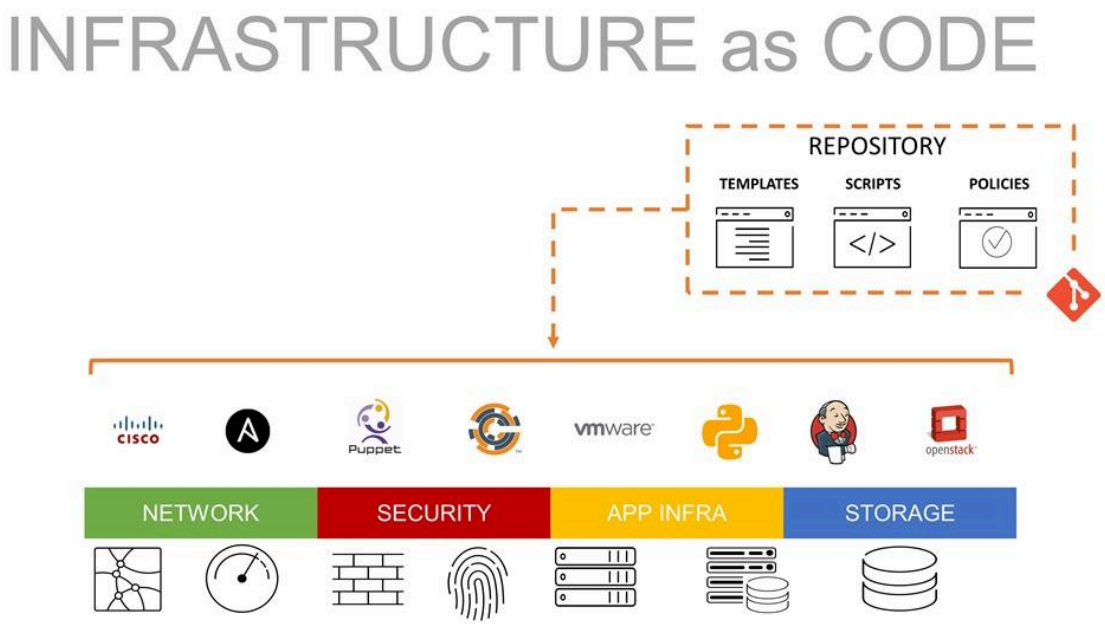

Figura 1: Separación de los recursos de configuración del hardware o software subyacente [Tomado de: https://www.f5.com/es_es/glossary/infrastructure-as-code-iac]

La Figura 1 hace referencia a un principio fundamental de Infraestructura como Código (IaC): la separación de los recursos de configuración del hardware o software subyacente. Este enfoque permite que las configuraciones de infraestructura se almacenen, compartan y gestionen como si fueran código, utilizando herramientas y repositorios de control de versiones como Git. Esto facilita la colaboración entre equipos, la trazabilidad de los cambios y la consistencia en la administración de la infraestructura.

Principios de la IaC:

1. **Repetibilidad:** IaC asegura que cada despliegue sea idéntico en diferentes entornos (desarrollo y producción), garantizando consistencia y reduciendo el riesgo de errores de configuración (Marinescu, 2017).
2. **Trazabilidad:** Los archivos de configuración pueden gestionarse mediante control de versiones, lo que facilita el registro y revisión de cada cambio en la infraestructura. Esto permite "revertir configuraciones a estados anteriores en caso de errores y auditar cambios en el sistema" (Buyya, Vecchiola y Selvi, 2013, p. 102).

3. **Automatización:** IaC automatiza el ciclo de vida de la infraestructura, desde su aprovisionamiento hasta las actualizaciones y eliminaciones. Esto reduce la intervención manual y minimiza la posibilidad de errores humanos, asegurando un proceso más ágil y fiable (Voorsluys et al., 2011).
4. **Uso de un enfoque declarativo sobre el imperativo:** El enfoque declarativo especifica el estado final deseado de la infraestructura, sin detallar los pasos exactos para llegar a ese estado. Esto permite que las herramientas de IaC determinen automáticamente cómo alcanzar esa configuración final. Este enfoque es preferido porque reduce la posibilidad de errores humanos y asegura que la infraestructura sea consistente en todos los entornos. (*Recommendations for Using Infrastructure as Code - Microsoft Azure Well-Architected Framework*, 2023)
5. **Modularización del código:** Dividir la infraestructura en módulos reutilizables es una práctica recomendada. Los módulos permiten encapsular configuraciones complejas y utilizarlas en diferentes proyectos o entornos, lo que mejora la eficiencia y reduce la duplicación de código. Por ejemplo, se pueden crear módulos para gestionar redes, almacenamiento o bases de datos, lo que facilita su mantenimiento y actualización.
6. **Separación por capas:** Dividir la infraestructura en capas según su nivel de complejidad o frecuencia de cambio es otra estrategia eficaz. Por ejemplo:
    - Una capa "low-touch" puede incluir recursos básicos como redes que cambian poco.
    - Una capa "medium-touch" podría abarcar servicios como bases de datos.
    - Una capa "high-touch" incluiría aplicaciones que requieren actualizaciones frecuentes.
7. **Seguridad integrada:** Es crucial priorizar la seguridad desde el inicio del desarrollo con IaC. Esto incluye:
    - Gestión segura de secretos.
    - Segregación adecuada de roles y permisos.
    - Implementación de políticas estrictas para evitar configuraciones inseguras. Integrar escaneos automáticos de seguridad en los *pipelines* CI/CD ayuda a identificar vulnerabilidades antes del despliegue.

8. **Documentación y estándares centralizados:** La documentación clara del código IaC y el uso de convenciones consistentes facilitan la colaboración entre equipos y aseguran que todos comprendan el propósito y funcionamiento del código. Además, establecer estándares centralizados dentro de la organización promueve la reutilización y consistencia en todos los proyectos.

Implementando estos principios, es posible aprovechar al máximo las ventajas del IaC, como la escalabilidad, eficiencia operativa y reducción del riesgo humano en los despliegues de infraestructuras en la nube.

Ventajas de IaC

IaC ofrece múltiples ventajas en la gestión de infraestructuras en la nube, mejorando tanto la eficiencia como la colaboración entre equipos:

- **Consistencia y reducción de errores humanos:** Con IaC, “cada despliegue se puede realizar de manera uniforme y confiable”, eliminando la necesidad de configuraciones manuales repetitivas y reduciendo errores en los procesos de infraestructura (Marinescu, 2017, p. 56).
- **Eficiencia y rapidez en el despliegue:** La automatización que proporciona IaC permite a las organizaciones realizar despliegues rápidamente y adaptarse a las demandas del negocio en tiempo real. Esto mejora significativamente la velocidad y confiabilidad del despliegue (Buyya et al., 2013).
- **Replicación de entornos:** IaC facilita la creación de entornos idénticos, lo cual permite realizar pruebas en condiciones similares a las de producción y reduce los problemas de compatibilidad entre entornos.
- **Control de cambios y colaboración:** Gracias al control de versiones, los cambios en la infraestructura pueden rastrearse y revisarse, promoviendo un trabajo en equipo más colaborativo y seguro. Como menciona Morris (2021), “IaC permite que los equipos trabajen en un mismo conjunto de configuraciones, asegurando una base común y actualizada para todos los miembros”.

Estas ventajas hacen de IaC una solución poderosa para la administración de infraestructuras en la nube, proporcionando consistencia, eficiencia y un marco de trabajo que facilita la colaboración y la trazabilidad de cambios.

1.3 Estrategias de despliegue

Las estrategias de despliegue son enfoques utilizados para implementar cambios en sistemas y aplicaciones, permitiendo una transición controlada y minimizando el riesgo de interrupciones o fallos. A continuación, se describen cuatro de las estrategias de despliegue más comunes: *Big Bang*, *Canary*, *Rolling Release* y *Blue-Green*. Cada una de estas estrategias ofrece distintos beneficios y desafíos, adecuados para diferentes necesidades y tipos de aplicaciones.

Big Bang

El despliegue *Big Bang* implica reemplazar todo el código existente de una sola vez. Es una estrategia adecuada para sistemas sin redundancia, donde se puede aceptar un tiempo de inactividad planificado. Aunque es simple de implementar y tiene bajos costos de infraestructura, tiene un alto riesgo, ya que cualquier fallo afecta a todos los usuarios, y la reversión puede ser compleja y lenta.

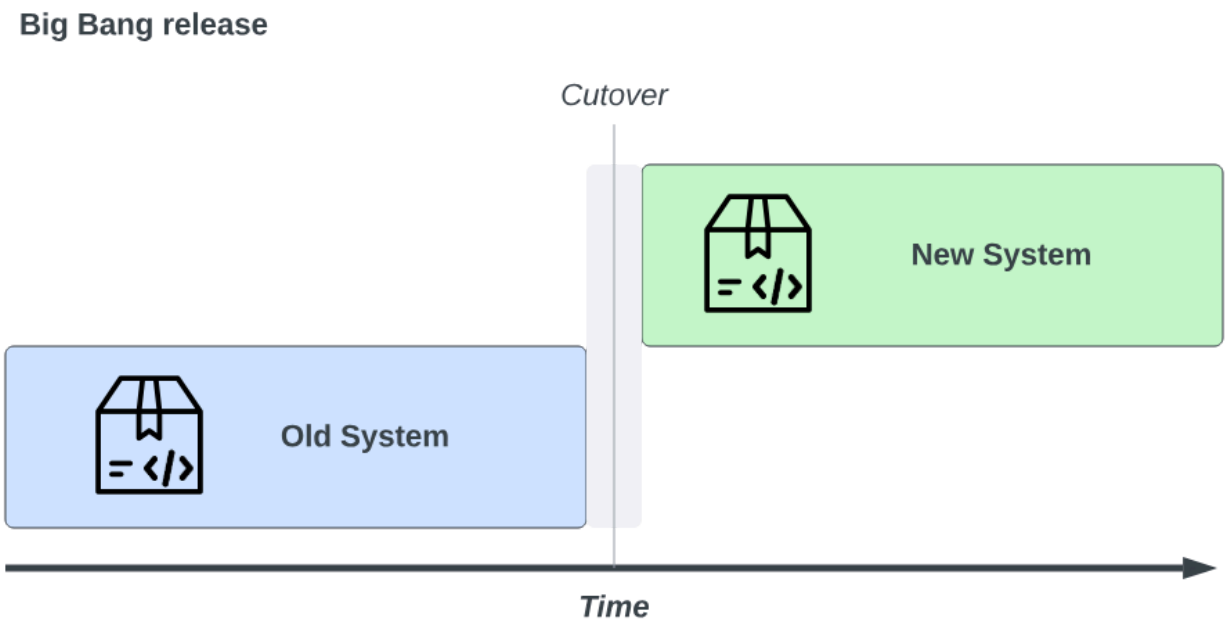

Figura 2 - Despliegue Big Bang (Sultantono, 2022)

Despliegue *Blue-Green*

En este modelo, el nuevo código se despliega en un entorno paralelo al actual (blue), llamado green, y el tráfico se redirige al nuevo entorno una vez que ha sido validado. Ofrece la capacidad de reversión inmediata y cero tiempo de inactividad, siendo ideal para aplicaciones sin estado. Sin embargo, su principal desventaja es el alto costo de infraestructura, ya que requiere duplicación de recursos durante el despliegue.

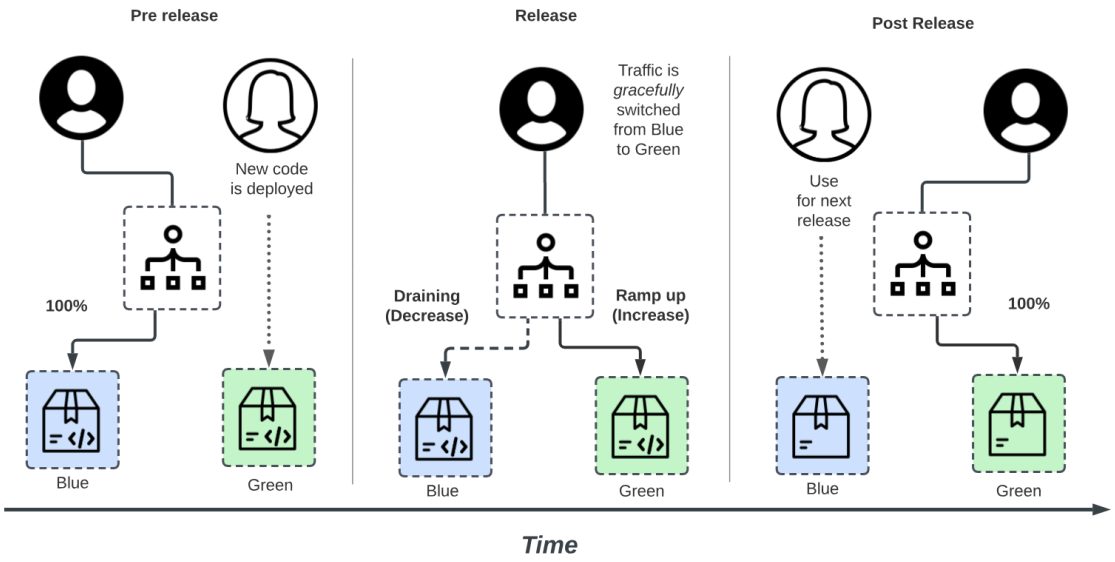

Figura 3: Despliegue Blue-Green (Sultantono, 2022)

Despliegue *Rolling Release*

El despliegue *Rolling Release* actualiza los recursos gradualmente, reemplazando las versiones antiguas con las nuevas sin detener el servicio. Es eficiente en términos de costos y adecuado para aplicaciones con estado que pueden ser fácilmente gestionadas en plataformas de contenedores o en la nube. Aunque permite mantener el servicio en funcionamiento, puede ser más lento y complicado de gestionar en comparación con otras estrategias

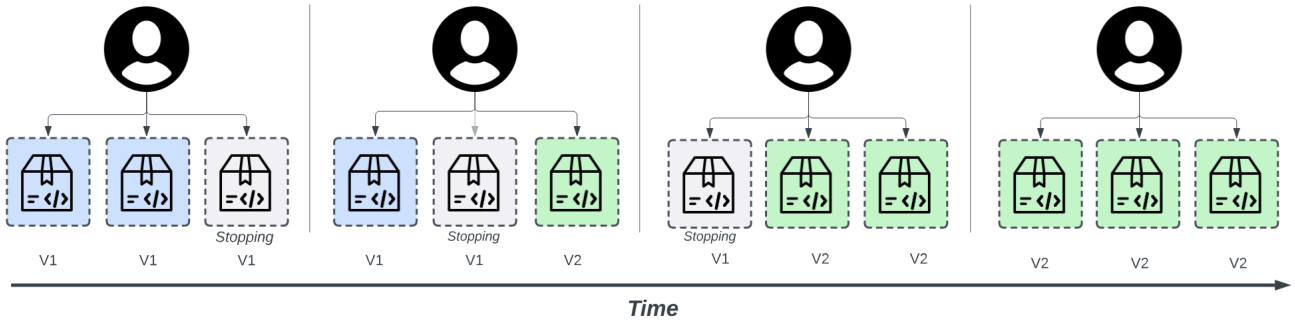

Figura 4: Despliegue Rolling (Sultantono, 2022)

Depliegue *Canary*

El despliegue Canary introduce el nuevo código a un pequeño subconjunto de usuarios o recursos para monitorear su rendimiento y, si es exitoso, se amplía gradualmente. Es ideal para validar la estabilidad de la nueva versión mientras se minimizan los riesgos. Aunque asegura cero tiempo de inactividad y pruebas en vivo, requiere un monitoreo continuo y puede ser más complejo de implementar que otros modelos.

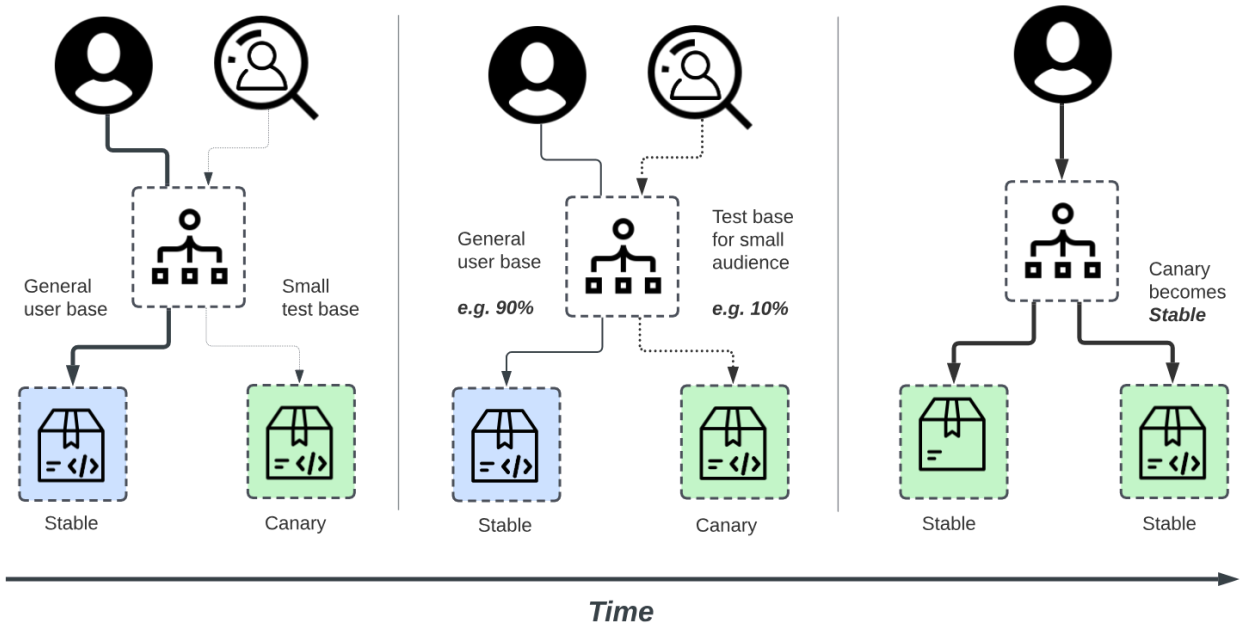

Figura 5: Despliegue Canary (Sultantono, 2022)

Después de analizar las características, ventajas y desventajas de los diferentes modelos de despliegue, se elige *Rolling Release* como la estrategia más adecuada para este proyecto. Esta decisión se basa principalmente en dos factores críticos: la necesidad de garantizar cero tiempo de inactividad y la optimización de costos de infraestructura.

Rolling Release permite realizar actualizaciones graduales mientras el sistema permanece en funcionamiento, asegurando que el servicio sea continuo y disponible para los usuarios en todo momento. Esto es crucial en aplicaciones modernas que no pueden permitirse interrupciones, como plataformas de comercio electrónico o servicios críticos en tiempo real.

A diferencia de estrategias como *Blue-Green*, que requieren duplicación de recursos y, por lo tanto, generan mayores costos, *Rolling Release* utiliza los recursos existentes para implementar las actualizaciones, lo que reduce significativamente los gastos operativos.

Además, esta estrategia permite minimizar el impacto de los errores, ya que los cambios se aplican progresivamente, reduciendo el riesgo de afectar a todos los usuarios en caso de fallos. Aunque no ofrece la reversión inmediata de modelos como *Blue-Green*, su enfoque gradual facilita identificar y corregir problemas en etapas tempranas del despliegue.

Por lo tanto, *Rolling Release* es la estrategia ideal para combinar eficiencia operativa, ahorro en infraestructura y la garantía de alta disponibilidad, alineándose perfectamente con los objetivos y requerimientos del proyecto.

1.4 Componentes de una infraestructura típica

En el contexto de aplicaciones modernas, una infraestructura robusta y escalable es esencial para garantizar una experiencia de usuario fluida y segura, especialmente en escenarios donde el volumen de tráfico y transacciones puede variar considerablemente. Contar con una infraestructura alojada en la nube permite que las plataformas respondan rápidamente a los cambios en la demanda, optimicen los tiempos de carga y mantengan altos estándares de seguridad y disponibilidad. Para lograr esto, se utiliza una arquitectura de múltiples capas que abarca desde la presentación de contenido hasta la administración segura de datos.

Descripción de la arquitectura

Una infraestructura en la nube incluye varios componentes clave que se integran para proporcionar un entorno seguro, escalable y de alto rendimiento. A continuación se detallan los elementos esenciales de esta arquitectura:

1. **Sistema de Nombres de Dominio (DNS, por sus siglas en inglés):** La solicitud del cliente comienza en el DNS, que traduce el nombre del dominio de la tienda en línea (por ejemplo, www.sistema.com) en una dirección IP que los navegadores utilizan para

encontrar el servidor de destino. Esto garantiza que el cliente sea redirigido a la infraestructura correcta de la tienda en la nube.

2. **Certificados digitales:** Los certificados digitales (como SSL/TLS) son esenciales para proteger la transmisión de datos entre el usuario y la aplicación. Estos garantizan que la información sensible, como las credenciales, se transmita de forma segura, encriptando la comunicación.

3. **Red de Distribución de Contenidos (CDN, por sus siglas en inglés):** un CDN almacena copias de los recursos estáticos de la aplicación (como imágenes, archivos CSS y JavaScript) en servidores distribuidos geográficamente. Esto reduce la latencia al servir contenido desde ubicaciones cercanas al usuario final, mejorando los tiempos de carga y optimizando la experiencia del cliente.

4. **Balanceador de carga:** Una vez que la solicitud del cliente llega al backend, un balanceador de carga distribuye el tráfico entre múltiples servidores de aplicación para evitar sobrecargas y asegurar que cada servidor funcione eficientemente. Esto mejora la disponibilidad y la capacidad de respuesta del sistema, permitiendo que los usuarios tengan una experiencia fluida sin interrupciones.

5. **Cluster de microservicios en contenedores:** En una arquitectura de microservicios, los componentes de la aplicación se ejecutan en contenedores dentro de un clúster. Este enfoque permite que cada servicio sea escalado y gestionado de manera independiente, mejorando la flexibilidad y la eficiencia del sistema. Plataformas como Kubernetes o *Elastic Container Services* (ECS) en *Amazon Web Services* (AWS) son comunes para orquestar y gestionar estos contenedores en entornos de producción.

6. **Bases de datos:** La infraestructura requiere bases de datos para almacenar información crítica, como datos de usuarios, detalles de productos y transacciones. Estas bases de datos deben estar diseñadas para garantizar alta disponibilidad y recuperación ante fallos.

7. **Uso de redes privadas y públicas:** La segmentación de la infraestructura en redes privadas y públicas permite gestionar mejor el acceso y la seguridad de los recursos. Las redes privadas alojan los datos y servicios críticos, mientras que las redes públicas manejan las conexiones de usuario. Esta separación ayuda a proteger la infraestructura de ataques externos y asegura que los datos sensibles permanezcan inaccesibles desde redes no autorizadas.

8. **Registro de logs de los servicios:** Los logs de los servicios proporcionan un registro detallado de las actividades y errores dentro de la infraestructura. Estos registros son fundamentales para el monitoreo, resolución de problemas y análisis de seguridad.

9. **Mecanismos para el control de acceso:** Los sistemas de control de acceso definen los permisos y políticas que regulan quién puede acceder a los diferentes recursos de la infraestructura. Estos mecanismos aseguran que solo los usuarios autorizados puedan gestionar o modificar los componentes de la infraestructura, protegiendo la plataforma de accesos no autorizados.

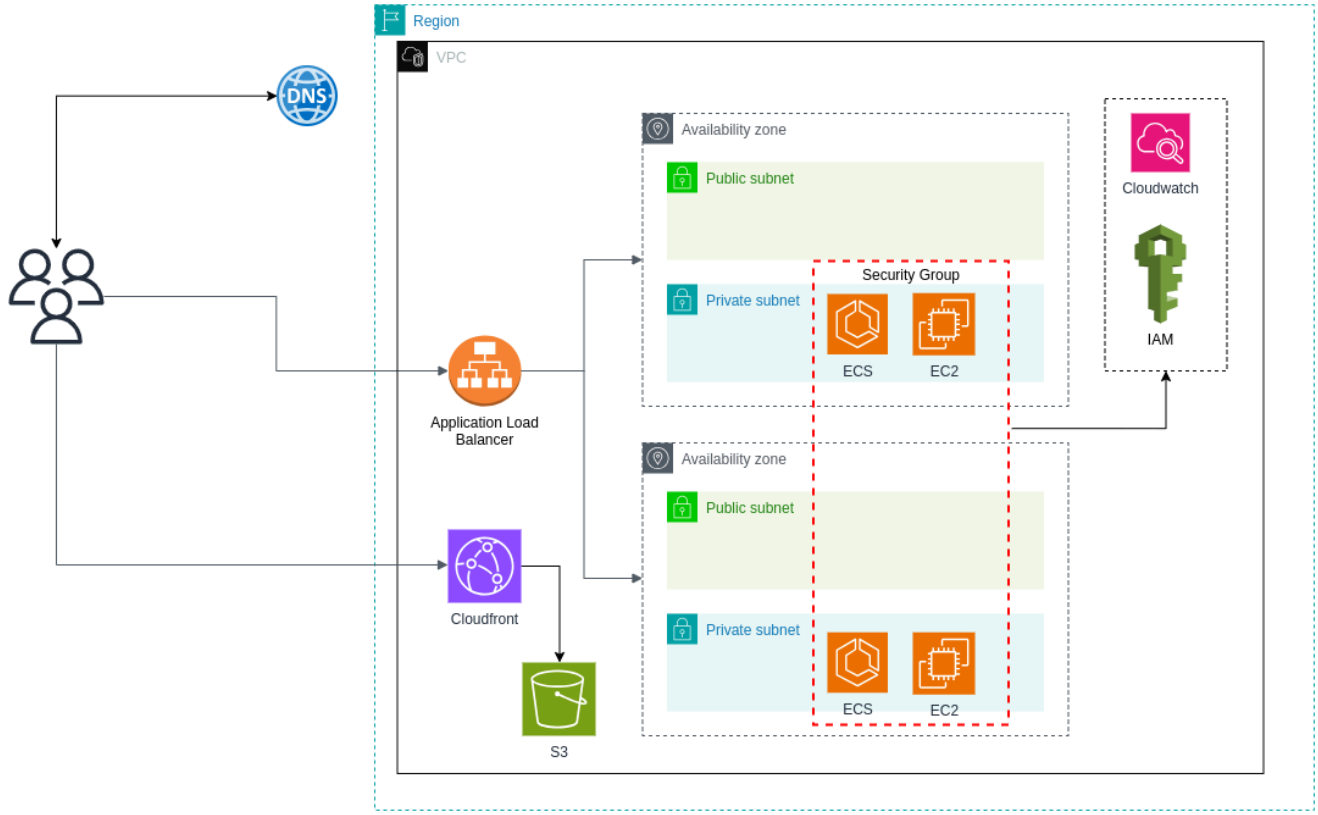

Figura 6: Arquitectura típica de una infraestructura de una aplicación moderna [Fuente: elaboración propia]

Esta arquitectura integra todos los componentes comunes para una aplicación robusta, escalable y segura. Cada componente cumple una función específica que contribuye a la experiencia general del usuario y a la estabilidad de la aplicación, permitiendo a las organizaciones responder a la demanda y garantizar la disponibilidad continua de su plataforma en la nube.

1.5 Sistemas homólogos para IaC

En el ámbito de la computación en la nube, las herramientas de IaC juegan un papel fundamental al permitir que las organizaciones gestionen y desplieguen sus infraestructuras de forma automatizada y repetible. Estas herramientas ofrecen un enfoque basado en archivos de configuración que permite definir y controlar los recursos de infraestructura mediante código, eliminando la necesidad de configuraciones manuales. Según Marinescu

(2017), "IaC permite a las organizaciones construir entornos complejos en la nube de manera eficiente y reproducible" (p. 230), asegurando que los recursos se mantengan consistentes y sean fácilmente replicables en distintos entornos.

Las herramientas de IaC han evolucionado para abarcar una variedad de necesidades, desde la gestión de servidores y redes hasta el aprovisionamiento de bases de datos y servicios específicos en la nube. Entre las herramientas más populares se encuentran Terraform, Pulumi, AWS CloudFormation, Ansible, Chef, Puppet y SaltStack cada una con características específicas y enfoques distintos en la gestión de infraestructura. Como señalan Buyya, Vecchiola y Selvi (2013), "la elección de una herramienta de IaC depende de factores como la flexibilidad, el soporte de múltiples proveedores y la capacidad de integración con sistemas de control de versiones" (p. 145).

El análisis se centrará en tres de las herramientas más destacadas en la gestión de infraestructura en la nube: Pulumi, Terraform y AWS CloudFormation.

Pulumi

Pulumi es una herramienta de Infraestructura como Código (IaC) que permite definir y gestionar recursos en la nube mediante lenguajes de programación de propósito general, como Python, TypeScript, Go y C#. Su enfoque distintivo permite a los desarrolladores utilizar lenguajes ya conocidos en lugar de aprender un lenguaje específico de IaC, como sucede con otras herramientas, lo cual facilita su adopción en equipos multidisciplinarios. Además, Pulumi es compatible con múltiples proveedores de nube, como AWS, Azure y Google Cloud, lo que permite la gestión de infraestructura en entornos multi-nube y reduce la dependencia de un solo proveedor. Esta capacidad de gestionar plataformas de nube de manera centralizada hace de Pulumi una opción versátil en entornos que requieren flexibilidad y un control detallado de la infraestructura (Alvaro & Satyanarayanan, 2019).

Pulumi es ideal para entornos DevOps que buscan mejorar la colaboración entre desarrollo e infraestructura, además de garantizar la trazabilidad y la coherencia en entornos de producción y desarrollo. Este enfoque en IaC ofrece un control granular sobre los recursos,

adaptándose a las necesidades de proyectos que requieren una infraestructura escalable y flexible (Yevick, 2020).

AWS CloudFormation

AWS CloudFormation es la herramienta nativa de IaC para AWS y permite a los usuarios definir y gestionar infraestructura en la nube mediante plantillas en YAML o JSON. Este enfoque permite organizar los recursos en *stacks* o conjuntos de infraestructura gestionados como una sola unidad, lo que facilita la administración de configuraciones complejas en AWS. CloudFormation proporciona un control detallado sobre los recursos de AWS y permite que los usuarios definan dependencias y conexiones entre ellos, proporcionando una estructura ordenada para entornos en la nube de gran escala y complejidad (Duffy, 2020).

En despliegues de infraestructura, AWS CloudFormation destaca por su integración directa con todos los servicios de AWS, permitiendo a las organizaciones aprovechar actualizaciones automáticas cuando se lanzan nuevos servicios o funcionalidades. Es ideal para organizaciones que operan exclusivamente en AWS y desean replicar entornos de desarrollo y producción con consistencia. CloudFormation es especialmente útil en aplicaciones que requieren arquitecturas complejas, como las basadas en microservicios, donde varios servicios interconectados deben implementarse y actualizarse de forma coordinada. Esta herramienta también permite un control de versiones de la infraestructura, brindando trazabilidad en los cambios y facilitando la colaboración en entornos de desarrollo continuo y equipos DevOps (Duan, 2019).

Terraform

Terraform, desarrollado por HashiCorp, es una herramienta de IaC ampliamente reconocida por su enfoque multi-nube y su independencia de proveedor, lo que permite gestionar infraestructura en diversas plataformas de nube como AWS, Google Cloud y Azure mediante una única configuración. Utiliza su propio lenguaje de configuración, HCL (*HashiCorp Configuration Language*), diseñado específicamente para describir la infraestructura de forma declarativa y modular.

En despliegues de infraestructura, Terraform permite gestionar grandes arquitecturas de manera eficiente gracias a su sistema de módulos reutilizables y su gestión de estado, lo que asegura la sincronización entre el código y los recursos reales en la nube. Terraform es ideal para proyectos que requieren una infraestructura escalable y flexible en entornos dinámicos, permitiendo a los equipos DevOps implementar cambios, integrar pruebas automatizadas y colaborar en un entorno de control centralizado (White, 2021).

A continuación se comparan Terraform, Pulumi y AWS CloudFormation, destacando sus características, ventajas y desventajas:

Tabla 2: Comparación de herramientas analizadas (Briggs, 2022)

Herramienta	**Terraform**	**Pulumi**	**CloudFormation**
Organización	HashiCorp	Pulumi	Amazon Web Services
Código abierto	Sí	Sí	No
Precio	Gratis hasta 5 usuarios en Terraform Cloud, sin costo para instalaciones *on-premise*	Gratis para 1 usuario	Gratis
Lenguaje	HCL	Varios	YAML / JSON
Ventajas	Líder del mercado; Multi-nube; Código escalable	Lenguajes de programación familiares; Multi-nube	Integración con AWS
Desventajas	HCL puede ser limitante	Comunidad más pequeña; Documentación insuficiente	Principalmente solo soporta AWS
Usabilidad	Se debe aprender HCL, pero tiene una barrera de entrada baja	Herramientas de programación familiares	YAML/JSON familiar, pero se vuelve difícil de leer y mantener
Gestión de estado	Almacenado localmente (gratis) o en la nube	Almacenado en la nube	Almacenado en la cuenta de AWS

Terraform destaca sobre Pulumi y AWS CloudFormation principalmente por su independencia de proveedor, lo que permite gestionar infraestructura en múltiples nubes (AWS, Azure,

Google Cloud, entre otras) con una única configuración, evitando el vendor lock-in. Su lenguaje HCL, aunque específico, es sencillo de aprender y facilita la creación de infraestructuras modulares y escalables. A diferencia de AWS CloudFormation, que está limitado a la infraestructura de Amazon Web Services, Terraform proporciona una solución multi-nube más madura y con una comunidad activa y extensa, lo que facilita el acceso a módulos preconstruidos y soporte comunitario. Además, su robusta gestión de estado, tanto local como en la nube asegura un control eficiente de los cambios y una colaboración fluida en equipo, superando en flexibilidad y soporte a Pulumi y CloudFormation en entornos heterogéneos.

1.6 Instrumentos y tecnologías de implementación

El desarrollo de una infraestructura para una aplicación moderna requiere un conjunto de herramientas y tecnologías que permitan gestionar, desplegar y mantener los recursos de forma automatizada y eficiente. En este proyecto, se han seleccionado tecnologías específicas para implementar IaC, realizar la integración y despliegue continuo (CI/CD, por sus siglas en inglés), gestionar contenedores y sus imágenes de forma efectiva. La combinación de estas herramientas proporciona un entorno integrado que mejora la consistencia, trazabilidad y colaboración en el desarrollo y mantenimiento de la infraestructura.

1.6.1 Terraform

Terraform es una herramienta utilizada en la IaC desarrollada por HashiCorp que permite a las organizaciones definir, aprovisionar y gestionar infraestructura en múltiples plataformas de nube a través de archivos de configuración. Diseñada para ser independiente del proveedor, Terraform se ha convertido en una opción preferida en entornos multi-nube, facilitando el despliegue y mantenimiento de infraestructuras complejas. Su propósito principal es simplificar y automatizar el ciclo de vida de la infraestructura, proporcionando un enfoque declarativo en el cual el usuario especifica el estado deseado de los recursos, y

Terraform se encarga de crearlos, actualizarlos o eliminarlos según sea necesario (Morris, 2021).

Desde su lanzamiento, Terraform ha ganado popularidad gracias a su capacidad para evitar el *vendor lock-in*, permitiendo que las organizaciones gestionen recursos en múltiples plataformas como AWS, Google Cloud, Microsoft Azure, por mencionar algunos. Su enfoque modular y reutilizable facilita la creación de infraestructuras consistentes y escalables, mejorando la eficiencia operativa y minimizando los errores humanos (Marinescu, 2017).

Lenguaje de configuración (HCL)

Terraform utiliza su propio lenguaje de configuración, llamado HCL (HashiCorp Configuration Language). Este lenguaje declarativo fue diseñado específicamente para definir infraestructura de manera simple y legible, lo que permite a los usuarios describir la configuración deseada de los recursos en un formato intuitivo. A diferencia de los lenguajes de propósito general, HCL está optimizado para IaC, lo que facilita la creación y mantenimiento de infraestructura sin requerir conocimientos avanzados de programación (Buyya, Vecchiola y Selvi, 2013).

HCL permite a los usuarios definir recursos, variables, módulos y dependencias de forma clara y concisa, lo cual es fundamental en entornos de infraestructura compleja. Además, el uso de HCL mejora la transparencia y trazabilidad de la infraestructura, facilitando la colaboración entre equipos y permitiendo auditar los cambios realizados en la configuración. Esta simplicidad y claridad en la definición de recursos han contribuido al éxito de Terraform en organizaciones de todo tipo.

Componentes básicos de Terraform

Terraform se basa en varios componentes fundamentales que permiten definir, gestionar y desplegar infraestructura de manera eficiente y escalable. Estos componentes forman la estructura que facilita el uso de Terraform para administrar recursos en múltiples plataformas de nube. En la tabla 3 se describen los principales componentes básicos de Terraform:

Tabla 3: Resumen de los componentes básicos de Terraform [Fuente: elaboración propia]

Componente	Descripción	Funcionalidad	Beneficios
Módulos	Agrupaciones de configuraciones que encapsulan recursos específicos para facilitar su reutilización.	Definen uno o varios recursos como redes, servidores o bases de datos, y pueden instanciarse en diferentes partes de la infraestructura.	- Reutilización de configuraciones - Estandarización de la infraestructura - Escalabilidad al actualizar módulos en múltiples entornos
Proveedores	Plugins que permiten a Terraform interactuar con diversas plataformas de nube y servicios de infraestructura.	Contienen conjuntos de recursos específicos que Terraform puede gestionar, como instancias de cómputo, bases de datos y redes.	- Multi-nube: Gestión desde una única herramienta - Flexibilidad al soportar diversos servicios y plataformas
Variables y Outputs	- Variables: Definen valores dinámicos para parametrizar configuraciones. - Outputs: Proporcionan información específica de los recursos creados.	- Variables: Hacen las configuraciones adaptables y reutilizables cambiando solo sus valores. - Outputs: Facilitan el acceso a datos importantes para otros módulos o uso externo.	- Flexibilidad para adaptar configuraciones a diferentes entornos - Reutilización mediante parametrización - Acceso sencillo a información clave de recursos
Fichero de Estado	Archivo que almacena la representación actual de la infraestructura gestionada por Terraform.	Rastrea todos los recursos creados, modificados o eliminados para mantener sincronizado el código con la infraestructura real.	- Consistencia al proporcionar una "fuente de verdad" - Facilita la colaboración en equipo - Seguridad al evitar conflictos y pérdida de datos

Ciclo de vida de IaC con Terraform

Terraform facilita la gestión de la infraestructura mediante un ciclo de vida de comandos que permite planificar, aplicar y, en caso necesario, desmantelar o destruir los recursos. Este ciclo de vida ayuda a los administradores y desarrolladores a gestionar la infraestructura de manera controlada y segura, con visibilidad de los cambios y la capacidad de revertir configuraciones si es necesario. En la Figura 7 se muestra un diagrama de flujo que representa el proceso de planificación de Terraform.

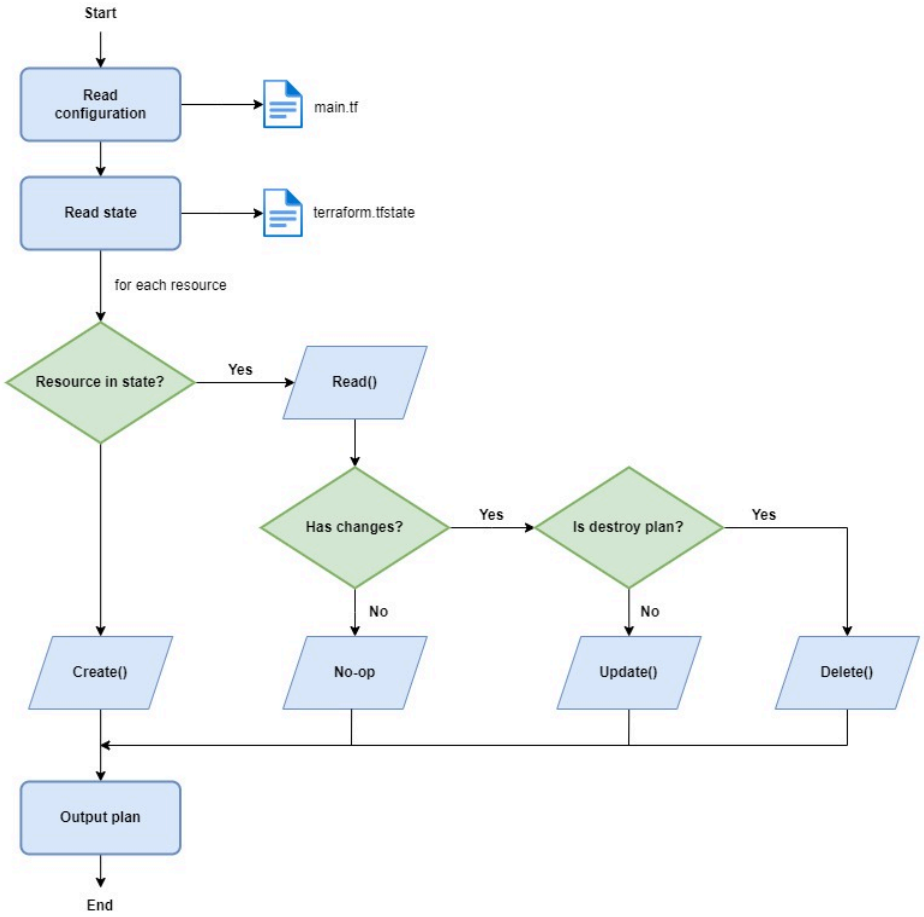

Figura 7: Diagrama de flujo del proceso de planificación. [Tomado de: Huỳnh, 2021]

1. Planificación (terraform plan)

El comando *terraform plan* es el primer paso en el ciclo de vida de Terraform y se utiliza para prever los cambios que se aplicarán a la infraestructura antes de ejecutarlos. Este comando genera un "plan de ejecución", una descripción detallada de los recursos que se crearán, actualizarán o eliminarán en función de la configuración actual y el estado de la infraestructura. Al ejecutar terraform plan, los usuarios pueden revisar estos cambios en un entorno seguro, lo cual permite identificar y corregir posibles errores o inconsistencias antes de realizar modificaciones reales en la infraestructura.

La planificación es una característica crucial en entornos de producción, ya que ayuda a mitigar errores potenciales y permite al equipo anticipar el impacto de cualquier cambio en los recursos existentes. Según Marinescu (2017), "el uso de un plan de ejecución en

Terraform permite evaluar el alcance de los cambios y asegurar que solo los recursos necesarios serán modificados" (p. 289).

2. Aplicar los cambios (terraform apply)

El comando *terraform apply* es el proceso de ejecución de la configuración de infraestructura. Una vez que los cambios han sido revisados y aprobados en el plan de ejecución, *terraform apply* se utiliza para aplicar esos cambios en la infraestructura real. Este comando lee la configuración especificada en los archivos de Terraform y crea, modifica o elimina los recursos en el proveedor de nube de acuerdo con el plan.

Durante la aplicación, Terraform sincroniza el estado de la infraestructura actual con la configuración deseada, asegurando que los cambios se realicen de manera ordenada y sin conflictos. La ejecución de *terraform apply* es idempotente, lo que significa que los recursos no se duplicarán si el comando se vuelve a ejecutar sin cambios en la configuración. Esto garantiza consistencia y evita la creación accidental de recursos redundantes. Morris (2021) señala que "la idempotencia de Terraform asegura que el estado final de la infraestructura refleje exactamente lo especificado en la configuración, sin importar cuántas veces se aplique" (p. 104).

3. Destrucción (terraform destroy)

El comando *terraform destroy* permite desmantelar toda la infraestructura creada por Terraform. Esta operación elimina todos los recursos definidos en la configuración, devolviendo la infraestructura a un estado limpio sin dejar recursos residuales. Antes de proceder, Terraform solicita una confirmación para ejecutar terraform destroy, ya que este comando desmantela por completo la infraestructura administrada.

La capacidad de destruir la infraestructura de forma ordenada es especialmente útil en entornos de prueba y desarrollo, donde se crean entornos temporales que deben eliminarse una vez completado el trabajo. Según Buyya, Vecchiola y Selvi (2013), "la capacidad de

eliminar la infraestructura con un solo comando en IaC permite una administración eficiente de los entornos temporales y facilita el ahorro de recursos en la nube" (p. 295).

1.6.2 Gitlab

GitLab es una plataforma integral que combina control de versiones con herramientas de integración y despliegue continuo (CI/CD, por sus siglas en inglés), lo que la convierte en una solución ideal para proyectos de IaC. En este caso, GitLab desempeña un papel esencial en la gestión de versiones de la configuración de infraestructura, además de automatizar el ciclo de vida de despliegue de la infraestructura y de las aplicaciones mediante *pipelines* de CI/CD.

GitLab permite realizar un seguimiento de todos los cambios realizados en los archivos de configuración de Terraform, asegurando que el equipo pueda revertir a versiones anteriores cuando sea necesario, y proporcionando trazabilidad en los cambios de la infraestructura. Además, GitLab facilita la ejecución de *pipelines* automatizados que gestionan el ciclo de vida de Terraform, desde la planificación (terraform plan) y aplicación (terraform apply) de la infraestructura hasta la actualización controlada de los recursos. Al utilizar GitLab CI/CD, se garantiza que cada cambio en el código de IaC sea probado y aprobado antes de ser implementado, reduciendo errores y manteniendo la consistencia en los entornos de desarrollo y producción.

Otra funcionalidad clave de GitLab en este proyecto es la gestión de contenedores Docker. A través de GitLab CI/CD, se construyen y despliegan imágenes de Docker, que luego se almacenan en un repositorio o registro de Docker privado. Esto asegura la disponibilidad y accesibilidad de las imágenes para todos los entornos de forma segura, facilitando la estandarización y coherencia de los contenedores en cada etapa del despliegue. GitLab permite integrar la administración de IaC con el desarrollo y despliegue de aplicaciones, asegurando una infraestructura estable y fácilmente reproducible en cada actualización.

1.6.3 Docker

Docker es una plataforma de contenerización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portátiles, facilitando la ejecución consistente en cualquier entorno. En este caso, Docker se utiliza para crear y desplegar contenedores que encapsulan aplicaciones y servicios, asegurando que se ejecuten de manera idéntica en entornos de desarrollo, pruebas y producción. Docker ayuda a minimizar problemas de compatibilidad, ya que cada contenedor incluye todas las librerías, configuraciones y archivos necesarios para que la aplicación funcione, eliminando así las diferencias entre los entornos locales y los de producción (Turnbull, 2018).

Para almacenar y gestionar las imágenes de Docker, se utiliza un Docker Registry centralizado, donde las imágenes se alojan y versionan. Este registro actúa como un repositorio desde el cual los contenedores pueden ser desplegados fácilmente en cualquier entorno. En combinación con GitLab CI/CD, se construyen, prueban y despliegan automáticamente nuevas versiones de las imágenes, permitiendo una integración y despliegue continuo de las aplicaciones. Este enfoque facilita la actualización de servicios en producción de manera controlada y sin interrupciones, asegurando que la infraestructura esté siempre alineada con las últimas versiones de la aplicación.

Conclusiones parciales

En el presente capítulo se ha establecido una base teórica sólida sobre los principios de la computación en la nube y la IaC, analizando los beneficios que estas tecnologías ofrecen en términos de escalabilidad, automatización y eficiencia. La computación en la nube permite desplegar y gestionar recursos de forma flexible y adaptable, optimizando costos y mejorando la disponibilidad. A su vez, la IaC asegura que las configuraciones de infraestructura se puedan gestionar y replicar de forma automatizada y controlada, lo cual es crucial para entornos de despliegue rápido y continuo.

Entre las herramientas de IaC analizadas, Terraform ha sido seleccionada como la opción a utilizar debido a su enfoque multi-nube, con una comunidad activa y extensa y facilita la

portabilidad y gestión de recursos en diferentes plataformas como AWS, Google Cloud y Azure. Su capacidad para definir infraestructura mediante el lenguaje HCL y su sistema de módulos reutilizables permite estandarizar configuraciones y simplificar la administración en entornos complejos. Terraform ofrece soporte para una amplia variedad de recursos y escenarios, lo cual facilita la implementación de soluciones complejas y mejora la colaboración en equipos de desarrollo e infraestructura.

Además, se introdujo el ecosistema de tecnologías complementarias, como GitLab y Docker, que facilitan la integración y el despliegue continuo (CI/CD), así como la contenerización de aplicaciones para asegurar la consistencia entre entornos. GitLab se emplea para controlar versiones y ejecutar *pipelines* que automatizan la gestión de IaC y el despliegue de imágenes en un registro de Docker centralizado. Docker permite empaquetar y desplegar aplicaciones en contenedores que aseguran la portabilidad y estandarización de entornos.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Este capítulo presenta la propuesta de solución para implementar una infraestructura en la nube mediante IaC con Terraform. La propuesta se estructura en varios pasos clave que abarcan el diseño de la arquitectura, la elección de los componentes, y las configuraciones necesarias para asegurar un entorno escalable, seguro y adaptable.

2.1 Descripción de la propuesta

La propuesta para el despliegue de una infraestructura en la nube se basa en el uso de Terraform como herramienta IaC y GitLab para la gestión de CI/CD. La estrategia se centra en la configuración modular y parametrizable de componentes en AWS, lo cual permitirá una infraestructura escalable, reutilizable y controlada que se adapte tanto a entornos de desarrollo como de producción. Esta infraestructura se definirá de manera que el despliegue y mantenimiento sean gestionados de forma automatizada, segura y con la menor intervención manual posible.

La propuesta incluye la configuración detallada utilizando Terraform de los recursos necesarios para el funcionamiento de la infraestructura en la nube. Se utilizarán los proveedores de AWS y GitLab para gestionar los recursos necesarios como DNS, balanceadores de aplicaciones, certificados digitales y componentes de red. Además, se configurará un clúster de *Elastic Container Service* (ECS, por sus siglás en inglés) para alojar servicios en contenedores Docker. El código de Terraform será modular y parametrizable, permitiendo su reutilización en diferentes proyectos al modificar variables específicas del proyecto, como el nombre de dominio y el entorno (desarrollo o producción), lo cual se gestionará a través de variables definidas en GitLab. En la figura 8 se puede apreciar la estrategia para el despliegue de una infraestructura mediante Terraform utilizando IaC.

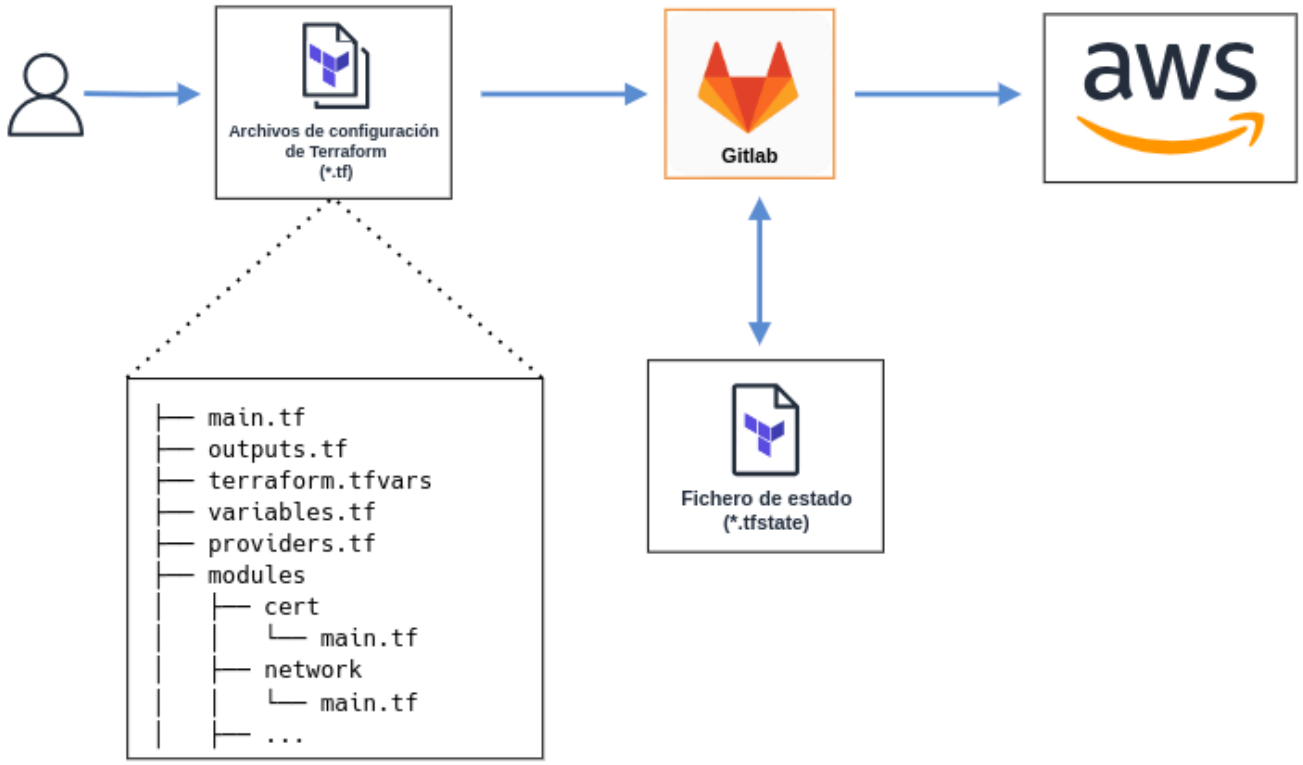

Figura 8: Estrategia para el despliegue de infraestructura utilizando Terraform [Fuente: elaboración propia]

A continuación se describen los componentes principales y la interrelación entre estos.

Archivos de configuración de Terraform

Los archivos de configuración de Terraform son fundamentales para definir y gestionar la infraestructura de manera automatizada mediante IaC. A continuación, se describen los archivos clave que conforman esta configuración y cómo están organizados en función de gestionar módulos reutilizables:

1. **Archivos de configuración**
    - **main.tf:** Define los recursos y la infraestructura principal que se desplegará, como redes, instancias, y configuraciones específicas de cada entorno.

- **variables.tf:** Declara las variables necesarias para parametrizar la configuración de la infraestructura. Estas variables permiten personalizar ciertos aspectos del despliegue, como los nombres de recursos, tamaños, o configuraciones específicas de cada entorno.
- **providers.tf:** Este archivo define los proveedores de nube y otros servicios que Terraform necesita para gestionar los recursos. En este caso, algunos de los proveedores utilizados son AWS y GitLab, permitiendo que Terraform interactúe tanto con la infraestructura en la nube (AWS) como con GitLab.
- **outputs.tf:** Especifica las salidas o *outputs* de Terraform, que exponen información relevante sobre los recursos creados, como direcciones IP o nombres de instancias. Estas salidas son útiles para integrar la infraestructura con otros sistemas o *scripts*.
- **terraform.tfvars:** Contiene los valores específicos para las variables declaradas en variables.tf, proporcionando los detalles concretos para cada entorno (por ejemplo, producción o desarrollo). Esto permite reutilizar el mismo código de infraestructura con diferentes valores.
- **modules**: La estructura incluye una carpeta llamada *modules* que contiene módulos reutilizables, los cuales encapsulan configuraciones comunes que pueden utilizarse en distintos entornos. Cada módulo tiene su propio *main.tf*, en el que se define un conjunto de recursos específicos, como redes, balanceadores de carga, o servidores de bases de datos. Estos módulos permiten mantener el código de la infraestructura modular y reutilizable, facilitando la administración y escalabilidad.

Esta organización de archivos permite una infraestructura modular, fácil de gestionar y adaptable a múltiples entornos, asegurando que el despliegue en producción y desarrollo mantenga consistencia y minimice errores.

El fichero de estado en Terraform

El fichero de estado (*.tfstate) es un archivo fundamental en Terraform, ya que contiene la información actualizada sobre el estado de la infraestructura. Este fichero permite a Terraform realizar un seguimiento de los recursos que ha creado, modificado o eliminado, y es crucial para ejecutar operaciones de manera idempotente. Al conocer el estado actual de la infraestructura, Terraform puede determinar los cambios necesarios para que la configuración coincida con el estado deseado.

En un flujo de trabajo con GitLab y Terraform, es importante gestionar adecuadamente el fichero de estado para evitar conflictos y asegurar que todos los cambios en la infraestructura sean consistentes. La relación entre GitLab y el fichero de estado incluye los siguientes aspectos:

1. **Almacenamiento remoto del fichero de estado:**
    - Cuando se busca trabajar de forma colaborativa, es preferible almacenar el fichero de estado en un backend remoto, como Terraform Cloud, AWS S3 o GitLab CI/CD. Esto permite que el fichero de estado esté accesible de forma centralizada y segura para todos los miembros del equipo, evitando inconsistencias que podrían surgir si el fichero se almacenará localmente.
    - GitLab se integra con estos backends para gestionar el estado remoto, asegurando que el *pipeline* CI/CD siempre tenga acceso a la última versión del fichero de estado antes de realizar cualquier cambio.
2. **Bloqueo del fichero de estado:**
    - Cuando varios *pipelines* en GitLab intentan acceder simultáneamente al fichero de estado, pueden surgir conflictos que provoquen corrupción o inconsistencias. Para evitar esto, el backend de almacenamiento remoto permite bloquear el fichero de estado mientras un *pipeline* de GitLab lo está utilizando.
    - Este bloqueo asegura que solo un proceso pueda modificar el fichero de estado a la vez, evitando problemas como sobrescritura o pérdida de datos, y garantizando que los cambios se apliquen de forma secuencial y controlada.
3. **Automatización de la gestión del estado mediante CI/CD:**

- En el pipeline de GitLab, las etapas de ***plan*** y ***apply*** de Terraform dependen del fichero de estado para conocer el estado actual de los recursos y aplicar solo los cambios necesarios. GitLab CI/CD se encarga de ejecutar estas etapas en un entorno aislado, donde Terraform obtiene el fichero de estado desde el backend remoto, realiza los cambios y luego actualiza el fichero de estado con el nuevo estado de la infraestructura.
- Mantener actualizado el fichero de estado garantiza que sucesivas actualizaciones trabaje siempre con la versión más reciente del estado de la infraestructura.

2.2 Proceso de despliegue de la infraestructura

El proceso de despliegue de la infraestructura define los pasos necesarios para crear y configurar la infraestructura en la nube de manera automatizada y controlada, utilizando Terraform y GitLab CI/CD. Este proceso asegura que todos los recursos se desplieguen de forma ordenada, rastreable y con mínima intervención manual, manteniendo la consistencia entre los entornos de desarrollo y producción. En la Figura 9 se puede observar las etapas para el despliegue.

Figura 9: Etapas para el despliegue de la infraestructura [Fuente: elaboración propia]

A continuación se detallan los pasos para desplegar la infraestructura desde cero:

Paso 1: Configuración inicial

- Clonar el repositorio: Se comienza clonando el repositorio Git que contiene los archivos de configuración de Terraform para la infraestructura. Esto asegura que la última versión del código esté disponible en el entorno de trabajo.

```
$ git clone git@gitlab.com:dofleini/devops/terraform/ecommerce.git
Cloning into 'ecommerce'...
remote: Enumerating objects: 4589, done.
remote: Counting objects: 100% (412/412), done.
remote: Compressing objects: 100% (409/409), done.
remote: Total 4589 (delta 254), reused 0 (delta 0), pack-reused 4177 (from 1)
Receiving objects: 100% (4589/4589), 667.04 KiB | 473.00 KiB/s, done.
Resolving deltas: 100% (2624/2624), done.
```

- Definir variables de configuración: Configurar el archivo de variables (*terraform.tfvars*) con los valores específicos para el entorno en el que se desplegará la infraestructura (desarrollo o producción). Esto incluye parámetros como el nombre del dominio, los identificadores del VPC, subredes, y credenciales necesarias para los proveedores.

```
# terraform.tfvars

# AWS
AccessKey = "XMNCNXCYYJWJCSAVJLCZ"
SecretKey = "FAfduMSzZ+pVmAegmAkAxpqHNaJPcMTsyiJQjwbZ"

# Gitlab
gitlab_token = "glpat-yqrEHUoYADaJxTvJabEr"

# Cloudflare
cloudflare_api_token = "WStfYcJmjJYqWkmvaVJewAVkrjqzTPiFvhYvPMom"

domain = "tienda.com"
environment = "production"
```

Paso 2: Inicialización de Terraform

3. Inicializar Terraform (*terraform init*):
    - Ejecutar *terraform init* en la raíz del directorio de configuración. Este comando prepara el entorno de Terraform, descargando los proveedores y módulos necesarios definidos en *providers.tf* y en los módulos externos especificados.
    - *terraform init* también configura el backend de almacenamiento del fichero de estado permitiendo que el estado de la infraestructura se almacene de forma remota para facilitar la colaboración y mantener la consistencia.

```
$  export GITLAB_ACCESS_TOKEN=glpat-yqrEHUoYADaJxTvJabEr
$ export TF_STATE_NAME=production
$ terraform init \
    -backend-config="address=https://gitlab.com/api/v4/projects/95397549/terraform/state/$TF_STATE_NAME" \
    -backend-config="lock_address=https://gitlab.com/api/v4/projects/95397549/terraform/state/$TF_STATE_NAME/lock" \
    -backend-config="unlock_address=https://gitlab.com/api/v4/projects/95397549/terraform/state/$TF_STATE_NAME/lock" \
    -backend-config="username=leandro" \
    -backend-config="password=$GITLAB_ACCESS_TOKEN" \
    -backend-config="lock_method=POST" \
    -backend-config="unlock_method=DELETE" \
    -backend-config="retry_wait_min=5"
```

Paso 3: Planificación de cambios

4. Validación de la configuración (*terraform validate*):
    - Ejecutar *terraform validate* para verificar que la configuración de los archivos .tf no contiene errores de sintaxis o de estructura. Esto asegura que la infraestructura esté definida correctamente antes de intentar aplicar cambios.

```
$ terraform validate
Success! The configuration is valid.
```

5. Revisión del plan de ejecución (*terraform plan*):
    - Ejecutar *terraform plan* para generar un plan de ejecución que detalla las acciones que Terraform tomará para crear o actualizar la infraestructura. Este comando permite revisar los recursos que se crearán, modificarán o eliminarán sin realizar cambios en el entorno real.
    - *terraform plan* muestra una lista de todos los cambios previstos, lo cual es útil para asegurarse de que los cambios sean los deseados, especialmente antes de desplegar en entornos críticos como producción. Este plan puede guardarse para aplicar los mismos cambios posteriormente sin necesidad de recalcular.

```
...
# module.network[0].aws_vpc.vpc[0] will be created
+ resource "aws_vpc" "vpc" {
    + arn                                  = (known after apply)
    + cidr_block                           = "10.1.0.0/22"
    + default_network_acl_id               = (known after apply)
    + default_route_table_id               = (known after apply)
    + default_security_group_id            = (known after apply)
    + dhcp_options_id                      = (known after apply)
    + enable_classiclink                   = (known after apply)
    + enable_classiclink_dns_support       = (known after apply)
    + enable_dns_hostnames                 = true
    + enable_dns_support                   = true
    + enable_network_address_usage_metrics = (known after apply)
    + id                                   = (known after apply)
    + instance_tenancy                     = "default"
    + ipv6_association_id                  = (known after apply)
    + ipv6_cidr_block                      = (known after apply)
    + ipv6_cidr_block_network_border_group = (known after apply)
    + main_route_table_id                  = (known after apply)
    + owner_id                             = (known after apply)
    + tags                                 = {
        + "Name"    = "vpc-mercatoria"
        + "project" = "mercatoria"
    }
    + tags_all                             = {
        + "Name"        = "vpc-mercatoria"
        + "automation"  = "terraform"
        + "environment" = "production"
        + "product"     = "mercatoria"
    }
}

Plan: 288 to add, 0 to change, 0 to destroy.
Changes to Outputs:
  + aws_eip_id                          = (known after apply)
  + internetgateway_id                  = (known after apply)
  + lb_production                       = "mercatoria-alb"
  + natgateway_id                       = (known after apply)
  + public1                             = "us-east-1a"
  + public2                             = "us-east-1b"
  + vpc_id                              = (known after apply)
```

Paso 4: Despliegue de la Infraestructura

6. Aplicar cambios (terraform apply):
    - Ejecutar *terraform apply* para aplicar los cambios especificados en el plan de ejecución. Este comando crea, modifica o elimina los recursos necesarios para que el entorno de la infraestructura coincida con la configuración definida en los archivos .tf.
    - Durante la ejecución de *terraform apply*, se solicitará una confirmación para proceder, permitiendo una última revisión antes de aplicar los cambios

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

...
module.network[0].aws_vpc.vpc[0]: Creating...
module.network[0].aws_vpc.vpc[0]: Still creating... [10s elapsed]
module.network[0].aws_vpc.vpc[0]: Creation complete after 12s [id=vpc-gjf99iscr4c4dusdi]
...

Apply complete! Resources: 288 added, 0 changed, 0 destroyed.

Outputs:
aws_eip_id = "eipalloc-3i2o7qjbhis4qj9y7"
internetgateway_id = "igw-7CA775222CE92435"
lb_production = "mercatoria-alb"
natgateway_id = "nat-9F4C9CE455F9EC22"
public1 = "us-east-1a"
public2 = "us-east-1b"
vpc_id = "vpc-gjf99iscr4c4dusdi"
```

Paso 5: Gestión del estado y control de versiones

7. Almacenamiento y bloqueo del fichero de estado:
    - GitLab CI/CD utiliza el backend remoto para acceder al fichero de estado durante las ejecuciones automáticas de *terraform plan* y *terraform apply*, asegurando que siempre se trabaje con el estado más reciente.
8. Control de versiones:
    - Realizar un *commit* en GitLab con cualquier cambio o ajuste en los archivos de configuración de Terraform, manteniendo así un registro de los cambios aplicados. GitLab proporciona un control de versiones completo, permitiendo revisar el historial de cambios y revertir modificaciones si es necesario.

El proceso de despliegue de la infraestructura con Terraform, integrado en GitLab CI/CD, permite gestionar los recursos de manera automatizada, controlada y eficiente. La ejecución secuencial de *terraform init*, *terraform validate*, *terraform plan* y *terraform apply* asegura que todos los cambios sean revisados y aprobados antes de ser implementados, proporcionando un entorno seguro y consistente. Esta metodología facilita la colaboración entre equipos y la

escalabilidad de la infraestructura en diferentes entornos, garantizando una administración de IaC efectiva y confiable.

2.3 Ciclo de vida de los servicios

El ciclo de vida de los servicios comienza una vez que la infraestructura ha sido desplegada exitosamente y el clúster ECS en AWS se encuentra operativo. En este punto, los servicios requieren las imágenes de Docker necesarias para su ejecución. Estas imágenes se gestionan mediante *pipelines* de GitLab CI/CD, que automatizan todo el flujo desde la construcción de las imágenes hasta su despliegue en el clúster ECS. Las configuraciones relacionadas, como el repositorio de imágenes y las acciones a ejecutar después de subir una imagen, ya han sido definidas como variables de entorno en Gitlab durante el despliegue de la infraestructura.

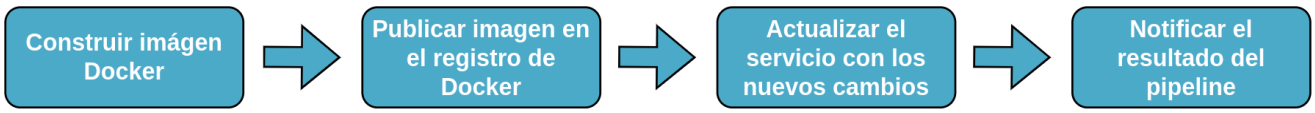

Figura 10: Flujo de trabajo de un pipeline en GitLab para actualizar los servicios [Fuente: elaboración propia]

En la Figura 10 se muestra el flujo de trabajo de un *pipeline* en GitLab para actualizar los servicios. El flujo se inicia cada vez que se realiza un *commit* en ramas específicas del repositorio Git, como *dev* o *main*. El *pipeline* se configura para ser ejecutado únicamente cuando se hace *commit* en ramas específicas:

- Rama *dev*: Orientada al entorno de desarrollo, genera imágenes etiquetadas para pruebas y validaciones en este entorno.
- Rama *main*: Dirigida al entorno de producción, crea imágenes etiquetadas como prod para su despliegue en producción.

Esta configuración garantiza que solo los cambios aprobados y versionados en estas ramas avancen hacia la construcción y despliegue de servicios, proporcionando control y

trazabilidad. Con cada *commit se* activa automáticamente el *pipeline* en GitLab CI/CD, según las configuraciones definidas en el archivo *.gitlab-ci.yml*.

```
image: registry.gitlab.com/gitlab-org/cloud-deploy/aws-base:latest

stages:
  - build
  - deploy
  - notify

.default-job: &default-job
  rules:
    - if: '$CI_COMMIT_REF_NAME == "dev" || $CI_COMMIT_REF_NAME == "main"'
  script:
    - echo "Using branch: $CI_COMMIT_REF_NAME"

build:
  stage: build
  extends: .default-job
  script:
    - echo "Building Docker image..."
    - docker build -t $ECR_REPO:$CI_COMMIT_REF_NAME .

deploy:
  stage: deploy
  extends: .default-job
  environment:
    name: $CI_COMMIT_REF_NAME
  script:
    - echo "Authenticating with AWS ECR..."
    - aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $ECR_REPO
    - echo "Pushing Docker image to ECR..."
    - docker push $ECR_REPO:$CI_COMMIT_REF_NAME
    - echo "Updating ECS service..."
    - aws ecs update-service --cluster my-cluster --service $CI_COMMIT_REF_NAME --force-new-deployment

notify:
  stage: notify
  extends: .default-job
  script:
    - if [ "$CI_JOB_STATUS" == "success" ]; then curl -X POST -H 'Content-Type: application/json' --data
'{"text": "Pipeline finalizado con éxito para '$CI_COMMIT_REF_NAME'."}'
https://api.telegram.org/bot$TELEGRAM_TOKEN/sendMessage?chat_id=<CHAT_ID>; else curl -X POST -H 'Content-Type:
application/json' --data '{"text": "Pipeline fallido para '$CI_COMMIT_REF_NAME'."}'
https://api.telegram.org/bot$TELEGRAM_TOKEN/sendMessage?chat_id=<CHAT_ID>; fi
```

Construir imágen *Docker*

La tarea inicia la construcción de la imágen de Docker a partir del fichero *Dockerfile* definido en el repositorio del proyecto. Este archivo describe las instrucciones para empaquetar el código fuente, las dependencias y las configuraciones necesarias para ejecutar el servicio.

```
# Usar una imagen base de Node.js ligera
FROM node:alpine

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /usr/src/app

# Configurar registro
RUN npm config set registry "http://nexus.dofleinisoftware.com/repository/npm/"

# Copiar el archivo package.json e instalar dependencias
COPY package*.json ./
RUN yarn install --prod

# Copiar el resto de los archivos de la aplicación
COPY . .

# Exponer el puerto en el que se ejecutará la aplicación
EXPOSE 8080

# Comando por defecto para ejecutar la aplicación
CMD [ "node","dist/main" ]
```

Publicar imagen en el registro de Docker

La imagen de Docker se sube a un registro de contenedores, como *AWS Elastic Container Registry (ECR)*. Este proceso está sujeto a restricciones adicionales para garantizar la seguridad y el control de acceso.

En este caso, se utiliza un *Identity Provider* (IdP) de AWS para GitLab, configurado con roles específicos que limitan los permisos de interacción con AWS, como subir imágenes al registro o realizar otras operaciones relacionadas. En lugar de credenciales estáticas, el IdP permite asumir roles temporales que otorgan permisos mínimos necesarios, fortaleciendo la seguridad y asegurando una mejor gestión de accesos.

Actualizar el servicio con los nuevos cambios

En el proceso de despliegue, cada nueva imagen de Docker creada desde de GitLab CI/CD sobrescribe la existente utilizando un esquema de etiquetas o *tags* para identificar el entorno. Las imágenes generadas desde la rama *dev* reciben el *tag* correspondiente a dicho entorno, mientras que las provenientes de la rama *main* son etiquetadas como *prod*. Este enfoque asegura que las imágenes estén claramente identificadas y asociadas al entorno correspondiente.

Además, el registro *ECR* retiene las imágenes sin *tag* por un período de dos días, lo que permite realizar análisis temporales o revertir a versiones recientes si es necesario. Este sistema de gestión evita el almacenamiento innecesario de imágenes obsoletas, manteniendo el registro limpio y optimizado.

Una vez que la imagen es subida al registro con el tag correspondiente, el *pipeline* fuerza la actualización del servicio en ECS, asegurando que las nuevas réplicas del servicio utilicen la imagen actualizada. Este proceso emplea la estrategia de *Rolling Release*, previamente explicada en el marco teórico, para garantizar una actualización gradual que minimice riesgos y mantenga la disponibilidad del servicio durante la transición. De esta forma los cambios se logran implementar de manera controlada, asegurando que la infraestructura siga operativa mientras las nuevas versiones se despliegan de forma incremental.

Notificar el resultado de la tarea

Finalmente, se envían notificaciones a un canal predefinido de Telegram con el resultado de la ejecución, indicando si el despliegue fue exitoso o si ocurrieron errores. Telegram es una aplicación de mensajería instantánea basada en la nube.

Conclusiones parciales

El presente capítulo ha detallado la propuesta de solución para el despliegue de infraestructura en la nube, utilizando principios de IaC y herramientas de automatización como Terraform y GitLab CI/CD. A través de este enfoque, se logra una infraestructura modular, escalable y controlada, que permite la gestión automatizada y reproducible de recursos en múltiples entornos de trabajo (desarrollo y producción). Los objetivos específicos definidos en la propuesta se cumplen mediante la implementación de un proceso de despliegue sistemático y colaborativo que facilita la escalabilidad, consistencia, trazabilidad y seguridad.

El uso de Terraform como herramienta central de la IaC ha permitido definir una infraestructura que puede desplegarse y gestionarse de manera estandarizada, eliminando la

necesidad de configuraciones manuales y reduciendo errores humanos. La creación de módulos específicos proporciona un alto nivel de reutilización y flexibilidad, permitiendo que esta solución pueda adaptarse y escalar según las necesidades del entorno. Además, la integración con GitLab CI/CD y el uso de tareas o trabajos automáticos ha permitido un flujo de trabajo ordenado y seguro, en el cual cada cambio en la infraestructura es probado y revisado antes de ser implementado en producción.

Uno de los aspectos críticos abordados en esta propuesta es la gestión del fichero de estado en un backend remoto, lo cual asegura la integridad y consistencia de la infraestructura al permitir el bloqueo del estado para evitar modificaciones simultáneas. La automatización del ciclo de vida de IaC mediante las etapas de *validate*, *plan* y *apply* en GitLab CI/CD proporciona un nivel adicional de control y trazabilidad, asegurando que solo los cambios aprobados se apliquen en el entorno de producción.

El proceso de despliegue descrito en este capítulo ofrece una solución completa para la gestión de infraestructura en la nube, que mejora la colaboración entre equipos, asegura la consistencia en los entornos, y garantiza un despliegue controlado y seguro. Esta propuesta sienta las bases para una administración de infraestructura eficiente, adaptable y alineada con las necesidades del negocio, proporcionando una estructura que facilita la expansión y actualización de los recursos a medida que la organización crece.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN PROPUESTA

En este capítulo se llevará a cabo la implementación y prueba de la solución propuesta para demostrar que cumple con los objetivos definidos en términos de escalabilidad, disponibilidad, trazabilidad y automatización. Este proceso incluye la implementación práctica de la infraestructura y los servicios, pruebas específicas para evaluar su comportamiento, y el análisis de resultados que respaldan la viabilidad de la solución en entornos reales.

3.1 Implementación de la solución

Despliegue con Terraform

El proceso de implementación con Terraform sigue una serie de pasos bien definidos que aseguran la creación, gestión y configuración de la infraestructura de manera eficiente y reproducible. A continuación, se describe el flujo general del despliegue:

- Inicialización de Terraform (terraform init)
- Creación del plan de despliegue (terraform plan)
- Ejecución del despliegue (terraform apply)
- Verificación del estado (terraform show / terraform state list)
- Gestión continua (terraform destroy o actualizaciones)
- Integración de GitLab para CI/CD

GitLab se utiliza para integrar la gestión de versiones del código y la automatización del flujo de trabajo a través de pipelines o trabajas para el CI/CD. Este enfoque garantiza que las configuraciones de infraestructura y las implementaciones sean reproducibles y controladas:

1. **Repositorio de código:**
   El código de infraestructura, incluyendo configuraciones de Terraform y módulos personalizados, se almacena en un repositorio GitLab. Esto facilita la colaboración y mantiene un historial detallado de cambios.
2. ***Pipeline* CI/CD:**
   Se configura un *pipeline* en GitLab para automatizar las tareas de validación, despliegue y pruebas de infraestructura.

Un ejemplo de configuración de *pipeline* para Terraform:

```
stages:
  - validate
  - plan
  - apply


validate:
  stage: validate
  script:
    - terraform init
    - terraform validate
  tags:
    - terraform


plan:
  stage: plan
  script:
    - terraform plan -
out=tfplan
  artifacts:
    paths:
      - tfplan
  tags:
    - terraform


apply:
  stage: apply
  script:
    - terraform apply tfplan
  when: manual
  tags:
    - terraform
```

Este *pipeline* automatiza la validación de configuraciones, genera un plan de despliegue y permite la aplicación manual del plan para asegurar control y supervisión en entornos de producción.

3. **Control de estado y seguridad**

Los pipelines aseguran que solo se implementen cambios validados y aprobados. Además, GitLab CI/CD permite configurar permisos y roles para garantizar que las actualizaciones sean realizadas por personal autorizado.

Configuraciones específicas

A continuación, se presentan ejemplos representativos del uso de código Terraform para configurar los principales componentes de la infraestructura:

1. **Módulo de red:** El propósito del módulo de red es establecer la base de conectividad necesaria para la infraestructura en la nube mediante la creación de una Red Virtual

Privada (VPC) y sus respectivas subredes públicas. Este módulo permite gestionar y organizar los recursos de red de manera eficiente, asegurando la escalabilidad, flexibilidad y trazabilidad de las configuraciones.

```
# main.tf
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = "${var.project_name}-vpc"
  }
}

resource "aws_subnet" "public" {
  count                   = length(var.public_subnets_cidr)
  vpc_id                  = aws_vpc.main.id
  cidr_block              =
var.public_subnets_cidr[count.index]
  map_public_ip_on_launch = true
  tags = {
    Name = "${var.project_name}-public-subnet-${count.index}"
  }
}
```

2. **Balanceadores de carga:** El propósito de este módulo es configurar un balanceador de carga para distribuir el tráfico de red de manera eficiente entre múltiples instancias de aplicación, garantizando alta disponibilidad, escalabilidad y resiliencia.

```
resource "aws_lb" "application_lb" {
  name               = "app-load-balancer"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.lb_sg.id]
  subnets            = var.public_subnets
}
```

3. **Módulo de instancia para MongoDB:** Este módulo configura una instancia o servidor virtual en AWS para ejecutar un servidor MongoDB. Su propósito es desplegar automáticamente una base de datos funcional dentro de la infraestructura, utilizando Terraform para la creación de la instancia y configurándose con los comandos necesarios para instalar y ejecutar MongoDB.

```
resource "aws_instance" "mongo_instance" {
  ami           = var.ami
  instance_type = var.instance_type
  subnet_id     = var.subnet_id
  security_groups = [var.security_group]
  key_name      = var.key_name


  user_data = <<-EOF
              #!/bin/bash
              sudo apt-get update -y
              sudo apt-get install -y mongodb
              sudo systemctl start mongodb
              sudo systemctl enable mongodb
            EOF


  tags = {
    Name = "${var.project_name}-mongodb-instance"
  }
}
```

Este módulo despliega instancias más robustas en producción y más económicas en desarrollo, adaptándose a las necesidades de cada entorno.

```
module "mongo" {
  source       = "./modules/ec2/mongo"
  instance_type = var.environment == "production" ? "t3.large" : "t3.micro"
  ...
}
```

Este enfoque combina la potencia de Terraform para gestionar infraestructura y GitLab para automatizar los flujos de trabajo, asegurando que el despliegue sea eficiente, trazable y

fácilmente escalable. Esta integración también garantiza que los procesos de implementación sigan las mejores prácticas y estándares de la industria.

4. Módulo de certificados

Este módulo, diseñado para gestionar certificados SSL y realizar validaciones en el DNS de Cloudflare, es un ejemplo práctico que demuestra cómo los procesos automatizados pueden reducir errores manuales, pero que también pueden ser susceptibles a configuraciones incorrectas si no se gestionan adecuadamente.

Este módulo configura un certificado SSL mediante un proveedor de nube (por ejemplo, AWS Certificate Manager) y realiza automáticamente el proceso de validación DNS en Cloudflare. Este proceso incluye:

1. Creación del certificado SSL.
2. Generación automática de registros DNS en Cloudflare (como CNAME o TXT) para validar el dominio.
3. Validación automática por parte del proveedor del certificado.

```
data "cloudflare_zone" "cloudflare_name" {
  name = var.domain
}

resource "aws_acm_certificate" "validation" {
  count                     = (var.cert == true && var.cert_condition == true) ? 1 : 0
  domain_name               = local.domain
  subject_alternative_names = ["*.${local.domain}"]
  validation_method         = "DNS"

  lifecycle {
    create_before_destroy = true
  }
}

resource "cloudflare_record" "cloudflare_name" {
  count   = (var.cert == true && var.record_condition == true) ? 1 : 0
  zone_id = data.cloudflare_zone.cloudflare_name.id
  type    = tolist(aws_acm_certificate.validation[0].domain_validation_options)[0].resource_record_type
  value   = tolist(aws_acm_certificate.validation[0].domain_validation_options)[0].resource_record_value
  ttl     = 60
  name    = <<EOF
${var.third_level_domain != false ?
  "${split(".", tolist(aws_acm_certificate.validation[0].domain_validation_options)
[0].resource_record_name)[0]}.${var.third_level_domain}" :
  tolist(aws_acm_certificate.validation[0].domain_validation_options)[0].resource_record_name}
EOF
}
```

Al crear manualmente registros como CNAME o TXT, es común equivocarse en los valores de *Name* o *Value*, lo que puede provocar que la validación del certificado falle.

3.2 Pruebas realizadas y resultados

Pruebas de Seguridad

Objetivo: Evaluar la capacidad de la infraestructura y los flujos de trabajo asociados para protegerse contra accesos no autorizados y garantizar la integridad de los recursos, incluyendo el registro de imágenes Docker en AWS.

Método:

1. **Escaneo de puertos:** El escaneo de puertos es una técnica utilizada para identificar los puntos de entrada en un sistema que están abiertos y disponibles para conexiones externas. En este trabajo se utilizó Nmap (nmap.org) a través de la plataforma hostedscan.com para confirmar que solo los necesarios estuvieran accesibles.
2. **Cifrado de datos:** Se empleó SSL Labs (ssllabs.com) para analizar la seguridad de las conexiones HTTPS y TLS en los servicios desplegados. Esto permitió evaluar la configuración de certificados SSL/TLS, identificar vulnerabilidades y calificar el nivel de seguridad de las conexiones.
3. **Pruebas en el flujo de CI/CD de GitLab:**
    - Se intentó subir una imagen Docker al registro del proyecto desde una rama no autorizada para confirmar que las políticas de seguridad y los permisos configurados en GitLab bloquearan el acceso.
    - Se probó acceder al fichero de estado remoto de Terraform utilizando un usuario con permisos de desarrollador para verificar que dichos accesos estuvieran restringidos únicamente a usuarios con rol de *maintainer* o superiores, siguiendo las configuraciones definidas en GitLab.

Resultados:

- **Escaneo de puertos:** Se identificaron puertos adicionales a los necesarios, correspondientes a Cloudflare, que actúa como proxy inverso. Estos puertos abiertos son requeridos para el funcionamiento de la red anycast de Cloudflare (Cloudflare Developers).

Scan Summary

Nmap 7.70 was initiated at Tue Nov 19 05:47:05 2024 with these arguments:
nmap -v -oX=- --host-timeout=8h -Pn -T4 -sT --webxml --max-retries=1 --open -p0-65535 mercatoria.store

Verbosity: 1; Debug level 0

Nmap done at Tue Nov 19 05:48:33 2024; 1 IP address (1 host up) scanned in 88.08 seconds

172.67.186.238 / mercatoria.store

Address

- 172.67.186.238 (ipv4)

Hostnames

- mercatoria.store (user)

Ports

The 65523 ports scanned but not shown below are in state: **filtered**

- 65523 ports replied with: **no-responses**

Port		State (toggle closed [0] | filtered [0])	Service	Reason	Product	Version	Extra info
80	tcp	open	http	syn-ack			
443	tcp	open	https	syn-ack			
2052	tcp	open	clearvisn	syn-ack			
2053	tcp	open	knetd	syn-ack			
2082	tcp	open	infowave	syn-ack			
2083	tcp	open	radsec	syn-ack			
2086	tcp	open	gnunet	syn-ack			
2087	tcp	open	eli	syn-ack			
2095	tcp	open	nbx-ser	syn-ack			
2096	tcp	open	nbx-dir	syn-ack			
8080	tcp	open	http-proxy	syn-ack			
8443	tcp	open	https-alt	syn-ack			
8880	tcp	open	cddbp-alt	syn-ack			

Figura 11: Reporte de escaneo de puertos utilizando Nmap (hostescan.com)

- **Cifrado de datos:** La herramienta utilizada, SSL Labs, realiza un análisis exhaustivo de la configuración de certificados SSL/TLS en un servidor web para evaluar su seguridad y cumplimiento con las mejores prácticas. Las pruebas que realiza incluyen:
  - **Validación del Certificado:** Verifica si el certificado fue emitido por una Autoridad de Certificación (CA) confiable.
  - **Compatibilidad de Protocolos:** Comprueba si el servidor soporta únicamente protocolos seguros como TLS 1.2 y TLS 1.3.
  - **Compatibilidad de Cifrado:** Evalúa si las suites de cifrado implementan algoritmos modernos y seguros, como AES-GCM.

- **Configuraciones del Protocolo:** Revisa si Perfect Forward Secrecy (PFS) está habilitado para proteger la confidencialidad de las sesiones anteriores.
- **Configuraciones del Servidor:** Analiza si el encabezado HTTP Strict Transport Security (HSTS) está configurado para prevenir ataques como downgrade o MITM.
- **Rendimiento:** Mide la eficiencia del handshake SSL/TLS inicial.

En base a los criterios anteriores asigna una calificación global al servidor, desde **A+** (excelente) hasta **F** (fallos críticos), basada en los resultados de todas las pruebas.

Como podemos ver a continuación durante las pruebas realizadas se obtuvo una calificación **A** en SSL Labs, confirmando que los certificados están correctamente configurados, con cifrados robustos y sin vulnerabilidades críticas.

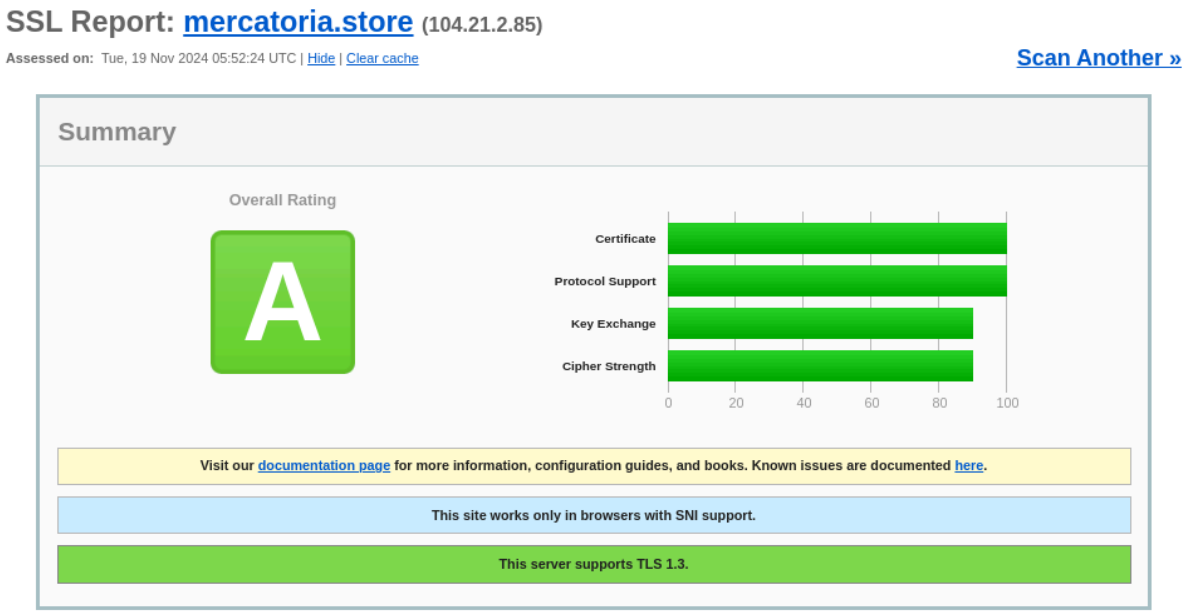

Figura 12: Reporte de la revisión de los certificados SSL/TLS

- **En el flujo de CI/CD de GitLab:** se realizó un intento de subir una imagen Docker al registro del proyecto desde una rama no autorizada. Este proceso falló al intentar interactuar con la nube, lo que confirmó que las configuraciones establecidas en el

flujo de CI/CD bloquearon correctamente las acciones provenientes de ramas sin los permisos adecuados.

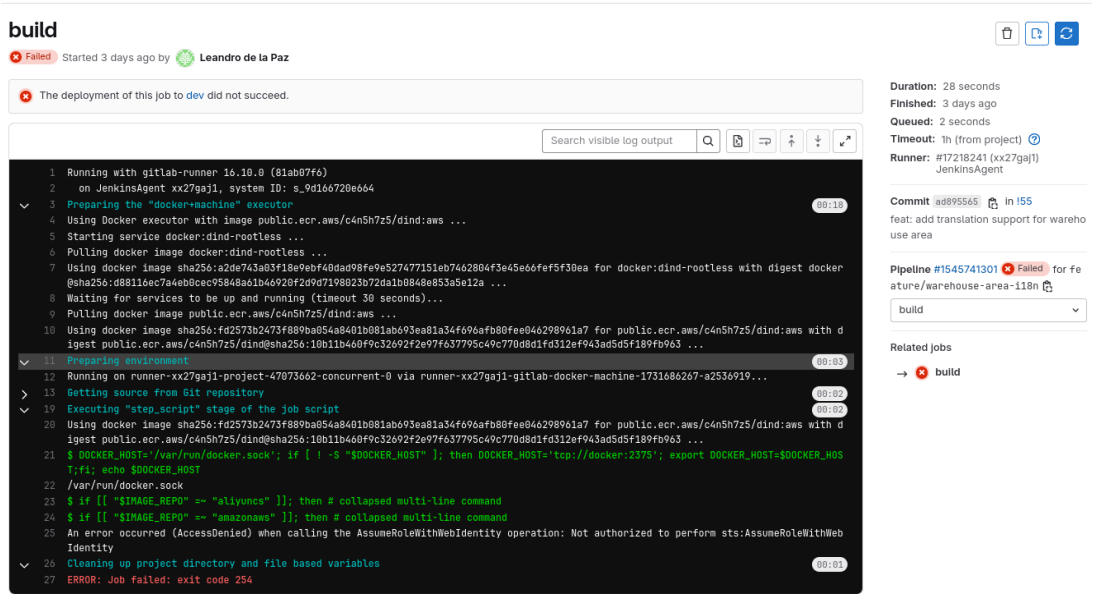

Figura 13: Error al intentar autenticarse desde una rama no autorizada [Fuente: elaboración propia]

3.3 Análisis de resultados del despliegue

En este apartado, se presenta un análisis detallado de los tiempos registrados para las operaciones de creación, modificación y destrucción de recursos, obtenidos directamente de los logs generados por los flujos de CI/CD en GitLab. Para realizar este análisis, se recopilaron datos de cuatro proyectos diferentes, lo que permitió establecer una base comparativa representativa del desempeño del proceso automatizado de despliegue.

Se midieron los tiempos correspondientes a cada tipo de operación:

- **Creación de recursos:** Abarca el aprovisionamiento inicial de la infraestructura, como instancias o redes.
- **Modificación de recursos:** Incluye ajustes a configuraciones existentes, como actualizaciones o cambios en parámetros.
- **Destrucción de recursos:** Considera la eliminación segura y controlada de los componentes desplegados.

Las operaciones de **creación**, con una mediana de **7.42 segundos**, permiten aprovisionar recursos en cuestión de segundos, mientras que manualmente estos mismos procesos podrían tardar desde minutos hasta horas dependiendo de la complejidad del recurso y el entorno. Incluso en los casos atípicos donde los tiempos máximos alcanzaron **782 segundos**, la automatización sigue representando un ahorro considerable en comparación con los procesos manuales.

Tabla 4: Análisis de la duración del despliegue según el tipo de operación [Fuente: elaboración propia]

Operación	cantidad	mediana	std	max
creación	259	7.42	53.42	782
destrucción	66	7.5	56.08	456
modificación	125	13.67	49.35	228

De manera similar, las **modificaciones**, que corresponden a las operaciones de mantenimiento de la infraestructura, presentan una mediana de **13.67 segundos**, mientras que las destrucciones muestran un tiempo promedio de **7.5 segundos**. Estos tiempos de ejecución son significativamente más rápidos en comparación con los métodos manuales tradicionales. En particular, las modificaciones aseguran una alta consistencia y rapidez, minimizando los riesgos de errores humanos al realizar ajustes en recursos ya existentes.

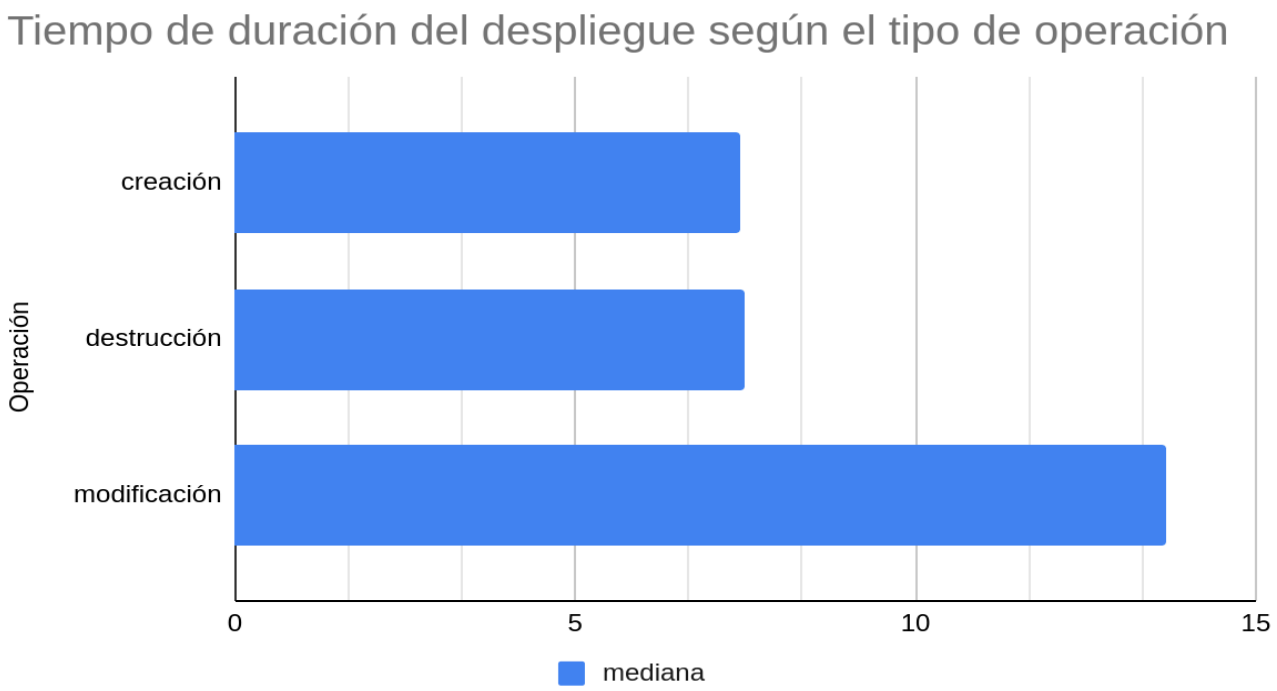

Figura 14: Tiempo de duración del despliegue según el tipo de operación [Fuente: elaboración propia]

3.4 Criterio de expertos

En el proceso de validación de la solución propuesta, se empleó el método de criterio de expertos como una herramienta fundamental para evaluar su efectividad, relevancia y aplicabilidad en escenarios reales. Este enfoque permitió obtener retroalimentación cualitativa de profesionales con experiencia en el área de Infraestructura como Código (IaC), gestión en la nube y DevOps, complementando los resultados obtenidos mediante pruebas experimentales. En la Tabla 5 se muestra los datos de los expertos consultados así como sus años de experiencia:

Tabla 5: Expertos consultados sobre la propuesta [Fuente: Elaboración propia]

No.	**Experto**	**Años de experiencia**
1.	Joelsy Porven	21
2.	Luis Frómeta	14
3.	Ernesto Yero	9

A los expertos se les formularon las siguientes preguntas para evaluar la solución:

1. **Reducción de tiempos:** ¿En qué medida considera que el enfoque propuesto reduce el tiempo necesario para tareas de mantenimiento o actualizaciones en comparación con un enfoque tradicional?
2. **Minimización de errores humanos:** ¿Qué tan efectiva cree que es la Infraestructura como Código para disminuir los errores humanos en comparación con un enfoque manual?
3. **Escalabilidad y adaptabilidad:** ¿Qué tan adecuada considera la solución para entornos que requieren escalabilidad rápida y frecuentes cambios en la infraestructura?
4. **Consistencia entre entornos:** ¿Qué tan confiable evalúa el uso de módulos reutilizables de Terraform para garantizar consistencia entre entornos (desarrollo, pruebas, producción)?
5. **Trazabilidad de los cambios:** ¿Qué tan efectiva es la solución para rastrear y documentar los cambios realizados en la infraestructura?

En la tabla 6 se presentan el resultado de las encuesta aplicada al grupo de expertos:

Tabla 6: Resultado de la encuesta aplicada al grupo de expertos [Fuente: elaboración propia]

Pregunta	Muy Efectiva/Conf.	Efectiva/Conf.	Moderada/ Poco Adecuada	Inadecuada/No Conf.
Reducción de errores humanos	66.7%	33.3%	0.0%	0.0%
Uso de módulos reutilizables	66.7%	33.3%	0.0%	0.0%
Reducción de tiempos	66.7%	33.3%	0.0%	0.0%
Adecuación para entornos dinámicos	66.7%	33.3%	0.0%	0.0%
Rastreo y documentación de cambios	66.7%	33.3%	0.0%	0.0%

El análisis de los resultados obtenidos a través del criterio de expertos destaca la efectividad de la solución propuesta en diversos aspectos clave de la gestión de infraestructura en la nube. Un 66.7% de los expertos evaluó la Infraestructura como Código (IaC) como muy

efectiva para reducir los errores humanos, mientras que el 33.3% restante la calificó como efectiva. Esto confirma que la automatización de procesos complejos mediante IaC elimina inconsistencias y minimiza las fallas asociadas a la gestión manual. Además, el uso de módulos reutilizables de Terraform fue considerado muy confiable por el 66.7% y confiable por el 33.3%, lo que evidencia la capacidad de esta herramienta para mantener configuraciones uniformes y consistentes en distintos entornos como desarrollo, pruebas y producción.

Por otro lado, la reducción de tiempos de despliegue y mantenimiento también fue valorada positivamente, con un 66.7% de los expertos considerando la solución como significativamente eficiente en comparación con enfoques tradicionales. Asimismo, la adecuación de la solución para entornos que demandan cambios frecuentes fue destacada como muy adecuada por la mayoría de los expertos, reforzando la flexibilidad y adaptabilidad del enfoque. Finalmente, la trazabilidad y documentación de cambios fueron evaluadas como muy efectivas, lo que resalta la importancia de integrar herramientas como GitLab para rastrear y gestionar las modificaciones de forma eficiente. Estos resultados respaldan la relevancia de la solución y su aplicabilidad en contextos reales, consolidando su efectividad en términos de automatización, confiabilidad y escalabilidad.

Evaluación de los objetivos

Los objetivos iniciales de la implementación se enfocaron en diseñar una infraestructura en la nube que fuera ágil, reproducible, trazable y escalable. A través del uso de Terraform, se logró establecer una base sólida para la gestión de infraestructura, automatizando tareas críticas como el despliegue de redes, servidores y servicios auxiliares. Esto permitió una gestión más eficiente y consistente de los recursos, con configuraciones reutilizables y parametrizadas que facilitaron la adaptación a diferentes entornos.

En términos de seguridad las pruebas de seguridad validaron que la infraestructura estaba protegida contra accesos no autorizados y que los flujos de trabajo en GitLab aseguraron la integridad de las operaciones, como la restricción del acceso al fichero de estado y el

bloqueo de acciones desde ramas no autorizadas. La integración de Terraform con herramientas complementarias como GitLab garantizan flujos CI/CD seguros y efectivos. En conjunto, garantiza la robustez, escalabilidad y seguridad requeridas por la infraestructura.

Conclusiones parciales

Este capítulo permitió validar la infraestructura propuesta mediante la ejecución de pruebas enfocadas en aspectos clave como la seguridad, la automatización y la consistencia operativa. Se destacaron resultados significativos, como la eficacia de Terraform en la gestión rápida y reproducible de recursos, y la integración con GitLab para garantizar flujos CI/CD seguros. Las pruebas realizadas confirmaron que la solución diseñada es capaz de gestionar operaciones críticas, reducir errores manuales y adaptarse a diferentes escenarios de despliegue con altos niveles de trazabilidad.

Los hallazgos de este capítulo refuerzan la importancia de adoptar Infraestructura como Código (IaC) en entornos dinámicos y la relevancia de herramientas complementarias para garantizar la seguridad y eficiencia en la gestión de recursos en la nube. Además, la implementación modular y parametrizable evidenció su capacidad de adaptabilidad, alineándose con los objetivos generales del trabajo. No obstante, se identificaron áreas de mejora, como la necesidad de cifrar el fichero de estado y utilizar imágenes Docker inmutables, lo que sugiere ajustes específicos para optimizar la solución en futuros despliegues.

CONCLUSIONES GENERALES

La estrategia propuesta cumplió con los objetivos de escalabilidad, automatización y trazabilidad, demostrando la eficacia de Infraestructura como Código en la gestión moderna de infraestructuras en la nube. Terraform resultó ser una herramienta clave para reducir la complejidad operativa, automatizando procesos que de forma manual habrían sido más lentos y susceptibles a errores, lo que refuerza su relevancia en entornos dinámicos.

La integración de GitLab en los flujos CI/CD fortaleció la trazabilidad y seguridad de las operaciones, garantizando la protección de los recursos mediante restricciones y controles de acceso. Además, el diseño modular y parametrizable de la solución permitió una alta capacidad de adaptación, haciendo que la infraestructura fuera aplicable a diversos escenarios con facilidad.

Los resultados obtenidos respaldan los conceptos teóricos de IaC y computación en la nube, reafirmando su importancia en la gestión eficiente y confiable de recursos tecnológicos. La combinación de estas herramientas y principios proporciona una base sólida para seguir explorando e implementando soluciones innovadoras en este campo.

RECOMENDACIONES

Aunque se demuestra la efectividad de la IaC, se identifican algunas áreas de mejora en la solución propuesta. Ellas son:

- **Implementar el cifrado del fichero de estado de Terraform:** Implementar el cifrado del fichero de estado de Terraform tanto en reposo como en tránsito.
- **Adoptar imágenes Docker inmutables:** Considerando la necesidad de garantizar la consistencia entre entornos, se sugiere configurar imágenes Docker con etiquetas únicas o hashes específicos. Esta práctica evita la sobrescritura de imágenes y asegura que cada versión sea fija y confiable.
- **Desarrollar guías operativas y capacitación:** Elaborar documentación detallada para los módulos de Terraform y los flujos CI/CD, acompañada de programas de capacitación para el equipo técnico. Esto facilitará el mantenimiento y la adopción por parte de nuevos usuarios.

BIBLIOGRAFÍA

- Carrasco, E., González, M., & Hernández, J. (2022). *Cloud Computing for e-Commerce: Scaling and Managing Demand with Cloud Infrastructure*. New York: TechPress.
- Marinescu, D. C. (2017). Cloud Computing: Theory and Practice. Elsevier.
- Voorsluys, W., Broberg, J., & Buyya, R. (2011). Cloud Computing: Principles and Paradigms. Wiley.
- Morris, J. (2021). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). *A View of Cloud Computing*. Communications of the ACM, 53(4), 50-58.
- Buyya, R., Vecchiola, C., & Selvi, S. T. (2013). *Mastering Cloud Computing: Foundations and Applications Programming*. Morgan Kaufmann.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). *A View of Cloud Computing*. Communications of the ACM, 53(4), 50-58.
- Briggs, L. (2022, August 26). Choosing an IaC tool. *Lee Briggs Blog*. Recuperado de https://leebriggs.co.uk/blog/2022/08/26/choosing-an-iac-tool
- Viblo. (2022). *Terraform Series – Bài 2: Life Cycle của một Resource trong Terraform*. Viblo.asia. Recuperado de https://viblo.asia/p/terraform-series-bai-2-life-cycle-cua-mot-resource-trong-terraform-RnB5pOMDlPG
- Alvaro, P., & Satyanarayanan, M. (2019). *Foundations of Cloud Computing: Infrastructure and Tools*. MIT Press.
- Yevick, D. (2020). *Programming the Cloud: Using Infrastructure as Code for Multi-Cloud Deployments*. CRC Press.
- Duffy, J. (2020). *Cloud Engineering on AWS: Concepts, Technologies, and Solutions*. Packt Publishing.
- Duan, Q. (2019). *Cloud Computing and Services Science*. Springer.
- Al-Khouri, A. (2020). *Cloud Computing and Beyond: Architecture, Protocols and Applications*. Springer.
- White, R. (2021). *HashiCorp Infrastructure Automation Certification Guide: Managing IaC Using Terraform*. Packt Publishing.
- Turnbull, J. (2018). *The Docker Book: Containerization is the new virtualization*. O'Reilly Media.
- F5 Networks. (s.f.). ¿Qué es la infraestructura como código (IaC)? F5. Recuperado el 13 de octubre de 2024, Recuperado de https://www.f5.com/es_es/glossary/infrastructure-as-code-iac

- Microsoft. (s.f.). Recommendations for using infrastructure as code - Microsoft Azure Well-Architected Framework. Microsoft Learn. Recuperado el 13 de octubre de 2024, de Recuperado de https://learn.microsoft.com/en-us/azure/well-architected/operational-excellence/infrastructure-as-code-design