UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

ESTRATEGIAS DE ARQUITECTURA DE SOLUCIÓN ESCALABLES CON APROVISIONAMIENTO DE INFRAESTRUCTURA AUTOMÁTICA (INFRAESTRUCTURE AS CODE - IAC)

AUTOR

FREDDY MAURICIO TACURI PAJUÑA

DIRECTOR:

CHRISTIAN MERCHÁN MILLÁN

QUITO – ECUADOR

2023

Autor:

Freddy Mauricio Tacuri Pajuña
Ingeniero informático
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Quito.
ftacurip@est.ups.edu.ec

Dirigido por:

Christian Merchán Millán
Ingeniero en computación
Magister en Sistemas de información
cmerchan@ups.edu.ec

TACURI PAJUÑA FREDDY MAURICIO
ESTRATEGIAS DE ARQUITECTURA DE SOLUCIÓN ESCALABLES CON APROVISIONAMIENTO DE INFRAESTRUCTURA AUTOMÁTICA (INFRAESTRUCTURE AS CODE - IAC)

DEDICATORIA

Se lo dedico primeramente a Dios, por darme la sabiduría necesaria para poder abarcar este nuevo reto, por darme la fuerza, energía y salud suficiente para llegar a la meta.

A mis padres por siempre brindarme ese apoyo moral, mis hermanos por ser la fuente de inspiración de los propósitos de mi vida, y mi novia Josselyn por acompañarme en las frías noches durante la realización de esta tesis.

En general a toda mi familia, muchas gracias por todo ese apoyo, se los dedico con mucho cariño y amor.

AGRADECIMIENTO

Primero le doy las gracias al Ing. Christian Merchán por toda su colaboración como director de tesis, por compartir su conocimiento y enseñanza en este proceso y haberlo llegado de la mejor manera.

Al mi equipo de trabajo los "DreamBuilders", especialmente a David y Gandhi primeramente por la oportunidad de prepararme, por la motivación y confianza de poder mejorar un poquito cada día.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

ESTRATEGIAS DE ARQUITECTURAS DE SOLUCIÓN ESCALABLE CON APROVISIONAMIENTO DE INFRAESTRUCTURA AUTOMÁTICA (INFRAESTRUCTURE AS CODE - IAC)

AUTOR:

FREDDY MAURICIO TACURI PAJUÑA

Resumen

Este documento tiene como objetivo principal identificar y analizar las diferentes estrategias de implementación de arquitecturas de software escalables, haciendo uso del aprovisionamiento automático de infraestructura, conocido como Infraestructure as Code (IAC), que busca garantizar la escalabilidad y la tolerancia a fallos en las aplicaciones empresariales.

En primer lugar, se realiza una investigación sobre la evolución de las arquitecturas de software en los últimos años, enfocándose en las estrategias utilizadas en aplicaciones empresariales. Se identificaron las principales limitaciones de la arquitectura tradicional monolítica y de la arquitectura orientada a servicios.

Posteriormente, se emplean herramientas de software como Terraform y Terratest para llevar a cabo una prueba de concepto. Estas herramientas permiten automatizar el proceso de aprovisionamiento de infraestructura en diferentes plataformas y entornos. Mediante esta prueba, se evalúa la viabilidad y efectividad de las estrategias de aprovisionamiento automático en términos de escalabilidad y tolerancia a fallos.

Los resultados de este estudio contribuirán a la comprensión de las arquitecturas de software escalables y sus beneficios en términos de operación eficiente y reducción de errores. Además, proporcionarán una base sólida para la implementación exitosa de estrategias de aprovisionamiento automático de infraestructura en aplicaciones empresariales.

Palabras clave:

Estrategias de arquitectura, arquitectura de software, tolerancia a fallos, aprovisionamiento automático de infraestructura, Infraestructure as Code (IAC), Terraform, Terratest, aplicaciones empresariales.

ABSTRACT

This document aims to identify and analyze different strategies for implementing scalable software architectures using Infrastructure as Code (IAC), an automated infrastructure provisioning approach. The primary objective is to ensure scalability and fault tolerance in enterprise applications.

Firstly, this research explores the evolution of software architectures in recent years, focusing on strategies employed in enterprise applications. It identifies the main limitations of traditional monolithic architecture and service-oriented architecture.

Subsequently, software tools such as Terraform and Terratest are utilized to conduct a proof of concept. These tools automate the infrastructure provisioning process across various platforms and environments. Through this proof of concept, the feasibility and effectiveness of automated provisioning strategies in terms of scalability and fault tolerance are evaluated.

The findings of this study will contribute to understanding scalable software architectures and their benefits in terms of efficient operation and error reduction. Furthermore, they will provide a solid foundation for the successful implementation of automated infrastructure provisioning strategies in enterprise applications.

Keywords:

Architecture strategies, software architecture, fault tolerance, automated infrastructure provisioning, Infrastructure as Code (IAC), Terraform, Terratest, enterprise applications.

1. Introducción

1.1. Antecedentes

Las aplicaciones empresariales modernas, en general, están conformadas por tres capas: presentación, servicios, y los repositorios de datos. Este tipo de aplicación se considera monolítica, por poseer un único ejecutable lógico, donde las pruebas, el desarrollo y el despliegue (un solo servidor de aplicación) son relativamente más sencillas; pero, a medida que la organización van creciendo, también crece el procesamiento de información, el número de clientes y posiblemente el número de incidencias en los servicios (Villamizar, 2018).

La escalabilidad de los sistemas fue creciendo a la par con la evolución de la arquitectura, donde empiezan a aparecer técnicas que permiten aprovechar todos los recursos de la organización. Algunas de estas técnicas de escalabilidad están relacionadas con la separación de la funcionalidad en componentes más pequeños (microservicios), réplicas de los servicios, y mejor administración de la data (Bolotin, 2019).

Los sistemas de información actualmente están construidos con una compleja composición de redes, servicios, aplicaciones e infraestructura, interconectados por grandes flujos de datos (Rong, 2022). Hoy en día el esfuerzo de despliegue de una aplicación se ha facilitado con las tecnologías actuales que permiten automatizar estos procesos, entre ellos el más popular es el uso de pipelines en CI/CD, siendo su propósito la canalización de un flujo de trabajo.

Con este fundamento, se trata de buscar la realización de una similitud sobre la forma de llevar una automatización, sacada de la implementación continua, donde los equipos de desarrollo más exitosos lo utilizan, con el objetivo de hacer lo mismo que realiza el CI/CD por las aplicaciones a la infraestructura, con herramientas que despliegen, con aprovisionamiento automático y escalable (Ahmad & Pahl, 2018).

La automatización de plataformas puede estar sujeto a distintas irregularidades, cómo ambigüedades de seguridad, degradación del rendimiento, accesos a recursos no permitidos, etc. generando resultados perjudiciales de varios tipos, cómo económicos, afectando a la privacidad de la información y también la seguridad interna de la organización (Rahman, 2018).

El autor (Patni & Tiwari, 2020), llevó a cabo las plantillas base entre Azure ARM de Microsoft, Pulumi y Terraform con el objetivo de explicar la infraestructura como código. Una vez consolidada la plantilla base, se indica como beneficia a la escalabilidad e implementación de varias instancias de la infraestructura, considerando el factor tiempo como una variable importante dentro de estos despliegues.

Algunas propuestas de estudios planteadas fueron: identificar todas características principales estructurales relacionadas con los defectos de infraestructura, especificar mediante la recopilación de datos cuáles factores estructurales están más propensos a fallos, e identificar las características que violen los objetivos de seguridad y privacidad de la información.

Entonces, bajo estos contextos se resalta como ha venido evolucionando los sistemas de información, en que nace la infraestructura como código y cómo se lo hace hoy en día, donde pensar en IaC es simplemente pensar automáticamente en virtualización y cloud. Entre los proveedores cloud más importantes donde se alojará la infraestructura tenemos los siguientes: Microsoft Azure, Amazon Web Services (AWS), Google Cloud Plataform (GCP), etc. Una plataforma que no es trivial para los clientes, donde los costos varían del uno al otro e igualmente la codificación dependerá de ello.

1.2. Determinación del problema

1.2.1. Descripción del problema

Las arquitecturas de software tradicionales tienden a tener ciertas limitaciones para adaptarse a las necesidades actuales de los sistemas empresariales, debido a que, comúnmente, estas aplicaciones están desarrolladas en miles de líneas de código, lógica de negocio embebido dentro de la misma vista, donde la aplicación de metodologías ágiles resulta muy compleja y también se presentan problemas con la integración y la entrega continuas.

1.2.2. Formulación del problema

La formulación permitirá identificar cual es el problema para investigar, pretendiendo resaltar el objetivo de estudio, donde se puede aplicar y finalmente se encontrará un marco teórico de investigación, planteado de la siguiente forma:

- ¿Cómo el diseño de una Arquitectura de solución escalable puede garantizar escalabilidad y tolerancia a fallas?
- ¿Cuáles serían las principales estrategias que se apegan a mejorar el escalamiento y las tolerancias a fallas?
- ¿Cómo la automatización de infraestructura minimizar el impacto de los errores humanos?
- ¿Cuáles fueron los principales retos dentro de la evolución de la arquitectura de software que permitió identificar los principales factores de escalabilidad?
- ¿Como mejora la arquitectura de software con el uso de patrones de diseño y principios de sistemas distribuidos?

1.2.3. Justificación del problema

Al plantear las diferentes estrategias de soluciones escalables se pretende utilizar un modelo de arquitectura de software que pueda ser aprovisionado de forma automática, aplicando buenas prácticas de diseño y los diferentes principios de sistemas distribuidos y escalables dentro de las diferentes plataformas o ambientes.

1.2.4. Delimitación del problema

El desarrollo del proyecto consiste en identificar las diferentes estrategias de arquitectura de soluciones para sistemas que sean viables y más utilizadas para BackEnd y FrontEnd, las cuales se enfocan principalmente en resolver la escalabilidad distribuida, por lo tanto, es importante hablar sobre cómo han evolucionado las arquitecturas de software en los últimos años, cuáles son las principales deficiencias de la arquitectura tradicional (monolítica), limitaciones sobre la arquitectura orientada a servicios (SOA) y las ventajas y limitaciones de la arquitectura orientada a microservicios y cómo contribuyó a la escalabilidad.

Una vez identificadas las estrategias, es necesario resaltar las principales deficiencias surgen en las arquitecturas tradicionales y orientadas a servicios, es por esta razón, la importancia de definir los principios de diseño, la escalabilidad y elasticidad, cuáles son las características fundamentales y ventajas de los microservicios, en la actualidad que representa la contenerización y cuál es el aporte en el escalamiento inmediato, los nuevos retos que se presentan y cómo la automatización como la infraestructura como código ayuda a asumirlos.

Finalmente, sobre las estrategias planteadas se realizará una prueba de concepto para demostrar el criterio esperado. Las herramientas empleadas para automatizar serán Terraform y Terratest seleccionando como entorno principal del proyecto una plataforma Cloud.

1.3. Justificación del problema

Las aplicaciones monolíticas se caracterizan por estar escritas en una única base de código, acopladas y con toda la ejecución en el mismo proceso, funcionan bien dentro de los escenarios clásicos (Al-Debagy & Martinek, 2018). En la actualidad, los despliegues de las aplicaciones con nuevas funcionalidades son mucho más frecuentemente y, sobre todo, estar en línea con la mayor disponibilidad posible (Gos & Zabierowski, 2020), donde, el estilo arquitectónico tradicional no cumple con las expectativas requeridas.

El escalamiento de este tipo de aplicaciones es más complejo debido a todo el trabajo relacionado con el despliegue de las diferentes instancias de ejecución (servidores de aplicación) de la aplicación monolítica (Amorim & Almeida, 2019).

Una de las técnicas de escalamiento para este tipo de aplicaciones consiste básicamente en descomponer la aplicación monolítica en pequeños servicios, independientes, atómicos, distribuidos y desacoplados, puesto que permite priorizar los recursos a los servicios más relevantes (Bahsoon & Emmerich, 2018).

Una característica importante de los servicios pequeños e independientes es la facilidad de desarrollo en paralelo por diferentes equipos, y no necesariamente dentro de las mismas tecnologías (Kuryazov & Khujamuratov, 2020), sino a lo contrario, usando las tecnologías que más se acoplen a la necesidad del servicio con el fin de cumplir su propósito.

El escalamiento de esos servicios individuales también se ejecuta de manera independiente, asignando los recursos necesarios en donde realmente necesita el negocio (Keqin, 2020).

Según el autor (Patni & Tiwari, 2020), la integración y distribución continua (CI/CD) ha traído mucho aporte y aprendizaje al momento de realizar un despliegue debido a que viene acompañando a la aplicación desde su construcción inicial, validando condiciones propias del negocio, como por ejemplo la cobertura del código, y su ejecución es simplemente correr un proceso preestablecido.

Por tal razón, el presente trabajo busca plantear varias estrategias de escalabilidad, considerando los cambios de requerimiento funcionales y no funcionales, que actualmente son demandamos sobre los sistemas de información, estos requerimientos ya no están relacionados, simplemente, sobre el desarrollo, sino también, en el ámbito de infraestructura, donde aparecen nuevos retos para llevar una estabilidad de mantenimiento dentro del despliegue. El uso de herramientas para automatizar el proceso del levantamiento, escalamiento y finalización de una infraestructura, por cualquier motivo, se considera uno de los mejores apoyos a los

equipos de TI, por ventajas como la reducción del tiempo, menos impacto de fallas por factor humano y un manejo de versiones de los distintos cambios realizados.

1.4. Objetivos

1.4.1. Objetivo general

Identificar las diferentes estrategias de implementación de arquitectura de software, utilizando la mínima operación y aprovisionamiento automático, para garantizar la escalabilidad y tolerancia a fallas.

1.4.2. Objetivos específicos

- Investigar sobre la evolución de la arquitectura de software de aplicaciones empresariales para identificar las diferentes estrategias utilizadas dentro de los últimos años.
- Identificar los principales problemas en la arquitectura de software tradicional (monolítica) y cuáles son las mayores limitantes en la arquitectura orientada a servicios.
- Utilizar herramientas de software como Terraform y Terratest para realizar una prueba de concepto sobre automatizar el aprovisionamiento de infraestructura en diferentes plataformas/ambientes.

2. Marco teórico referencial

Para desarrollar este capítulo es necesario considerar las definiciones previas más relevantes como marco conceptual que contenga la información requerida para entender los términos relacionados con la evolución de las arquitecturas de software. También es importante detallar los distintos patrones de diseño de sistemas y automatización de procesos.

Además, se analiza algunos trabajos realizados anteriormente, las discusiones y enfoques de los diferentes autores como base fundamental de investigación.

2.1. Marco conceptual/Estado del arte

2.1.1. Propósito de las empresas

En los últimos dos siglos, se ha experimentado tres grandes revoluciones. La primera revolución fue en el siglo XVIII, llamada la revolución industrial, y fue producto de un conjunto de avances tecnológicos, sociales y económicos. A mediados del siglo XX, la revolución informática deja un lado el conocimiento y la información y le abre paso a la producción en masa.

Hoy en día, se vive la tercera revolución, en donde el principal actor de esta revolución es el emprendedor, personaje que mantiene ciertas características comunes como la capacidad de generar oportunidades y convertirlas en realidad con diferentes tipos de recursos. Es así como en la actualidad existen organizaciones donde sus principales activos son el capital humano que tienen como reto cubrir una necesidad (Martínez Estrada, 2017).

Un ejemplo es Meta, una empresa fundad por Mark Zuckerberg el 4 de febrero de 2004, como un proyecto llamado “thefacebook.com” como una pequeña red de contactos social entre distintos estudiantes de Harvard. El capital inicial fue de solo, 15.000 dólares, actualmente la valuación de la empresa está rodeando los 192

billones de dólares. En conclusión, esto nos demuestra como los emprendedores parten de la ejecución de ideas atacando una necesidad con los recursos que se encuentran en el alcance.

2.1.2. Cultura organizacional

El concepto principal de cultura organización se va desarrollando a lo largo de los años setenta, una definición planteada por Robbins se puede expresar como un sistema de significados compartidos, que busca uno o varios objetivos en común, por parte de todos los que conforman la organización, mismos que caracterizan a una empresa de otra (Robbins, 2019). La cultura organizacional es un factor clave muy relevante para el manejo de las estrategias dentro de una empresa.

En la Figura 2.1, se muestran la comparación entre los valores de la cultura organizacional y cuál es la importancia de actuar y pensar para constituir un éxito estratégico. El eje vertical representa la importancia del movimiento hacia el éxito dentro de la estrategia y el eje horizontal, por otro lado, se enfoca en expresar la compatibilidad de las normas culturales por las distintas operaciones estratégicas.

Figura 2.1. Relación de la cultura organizacional

Fuente: (Davis, 2019)

El aporte de la cultura organizacional es mantener a las empresas dentro de un riesgo manejable, con maniobras eficientes, proyectando su resultando a tener buenos indicadores estratégicos.

Todas las empresas, proyectos, unidad (células) o personas utilizan indicadores de gestión (KPI's) como medida para validar el cumplimiento de los objetivos estratégicos propuestos (Roncancio, 2022).

Los KPI's pueden organizarse como los indicadores estratégicos que miden las variables directas de los recursos de la organización, como el total de ventas de un producto, los ingresos totales de la organización, total de utilidades producidas en ciertos periodos, etc. Pero también existen ciertos indicadores no comerciales que también sirven para la evolución del desempeño de las organizaciones, como la tasa de retención de clientes, la tasa de promoción de clientes (NPS), tasa de conversión de productos, cuota del mercado, etc.

2.1.3. Arquitecturas de software

La arquitectura de software es el núcleo del desarrollo y diseño de los sistemas informáticos, que no es solo el inicio del proyecto, sino también abarca al ciclo completo del sistema. Los beneficios de la arquitectura de software son lograr una integridad conceptual, una buena y efectiva base para el reúso de componentes y una comunicación efectiva (Martin, 2018).

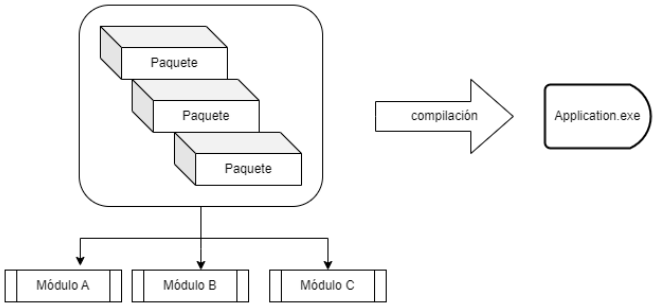
Una arquitectura de software permite tener una base sólida y escalable del proyecto, aumentando el rendimiento del sistema, con un buen manejo de costes, evitando duplicidad de código, teniendo un mantenimiento más eficaz, dentro de un producto con calidad y con mayor adaptabilidad. Otra ventaja transversal también es la facilidad de implementación de nuevos cambios, debido al conocimiento global del proyecto y del producto.

Generalmente, no es necesario crear nuevas arquitecturas de software, lo más común es adoptar una arquitectura conocida en función de las características, ventajas o inconvenientes de los casos en particular. Las arquitecturas tradicionales más importantes son:

2.1.3.1. Arquitectura monolítica

Este estilo arquitectónico se sustenta en crear una aplicación que contenta toda la funcionalidad para cumplir la función por la cual fue creada, sin contar con componentes o dependencias externas que sustenten su funcionalidad. Esto quiere decir que los componentes de la aplicación trabajan en conjunto, compartiendo recursos (Blancarte, 2019). Este tipo de arquitectura no fue planteada o planeada por alguien particular, básicamente parte desde el inicio de los sistemas que funcionaban tradicionalmente de esta forma.

Figura 2.2. Compilación de una aplicación monolítica



Fuente: Propia

La Figura 2.2 nos indica claramente que un monolito podría estar conformado por un único componente lógico o varios módulos/librerías, sin embargo, al momento de compilar todos los componentes se empaqueta en una sola pieza funcional ejecutable.

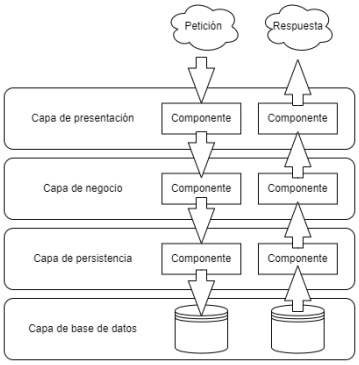
Con la llegada de internet, la posibilidad de consumir servicios externos a la aplicación permitió separar a la aplicación en unidades de software manejable, con alta cohesión y fácil de manejar. Pese a esta situación, las aplicaciones monolíticas sigues teniendo hoy en día gran participación.

2.1.3.2. Arquitectura en capas

La arquitectura distribuida en capas es una de las más comunes y utilizadas, por la simplicidad y el manejo de una estructura que se puede utilizar por defecto cuando no sabemos qué tipo de arquitectura utilizar para nuestro sistema.

La arquitectura de capas, como su nombre lo indica, consiste en dividir la funcionalidad en múltiples capas que tienen una responsabilidad o rol muy bien definido, como se muestra en la Figura 2.3, por ejemplo, una capa de manejo de la interfaz de usuario (presentación), una capa de manejo de lógica de negocio (servicio) y una capa de acceso a datos (DAO). Es importante resaltar que esta arquitectura no define cuantas capas usar, sino en como centralizar los componentes de la aplicación aplicando el principio "Separación Of Concerns" (separación de preocupaciones - SoC) (Blancarte Iturralde, 2019).

Figura 2.3. Arquitectura multicapas



Fuente: Propia

Cada una de las capas de esta arquitectura debe ser un componente separado e independiente, es decir, que se pueda desplegar de manera aislada. También es común colocar en otro servidor, siempre y cuando se mantengan en una comunicación de red.

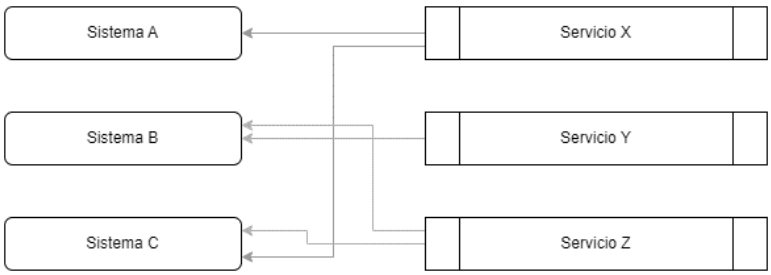
2.1.3.3. Arquitectura Orientada a Servicios (SOA)

Hace cerca de 20 años atrás, no existían tanta necesidad de integrar las distintas áreas de una organización, por lo que toda la información se centraba en un solo lugar. Sin embargo, con el transcurso del tiempo comenzaron a salir sistemas especializados en gestionar los distintos departamentos o áreas de una compañía de una forma más eficiente y sofisticada, como, por ejemplo, los sistemas ERP, CRM, CMS, de nómina, inventarios, contables, etc.

Uno de los problemas más importantes fue la incompatibilidad entre sistema, ya que aplicaciones desarrolladas en Visual Basic no eran compatibles con aplicaciones Java, o a su vez, no se podían comunicar con aplicaciones en C, esto provocó que los desarrolladores buscaran diferentes alternativas como por ejemplo intercambiar archivos planos o cargar una base de datos a sus aplicaciones, etc.

Con toda esta problemática, nace la Arquitectura Orientada a Servicios, la cual estimula a las organizaciones a crear servicios reutilizables para toda la organización, bajo estándares públicos que permita su consumo por cualquier tipo de aplicación en cualquier plataforma desarrollada. SOA no trajo simplemente el concepto de generar servicios, sino también, aplica un nuevo paradigma que dio paso a la revolución de la manera de producir software (Nossa Ortiz, 2019).

Figura 2.4. Consumo de servicios en SOA



Fuente: Propia

Así como se observa dentro de la en la Figura 2.4, las aplicaciones de la izquierda (A, B, C) brindan una cantidad de servicios disponibles que son consumidas por las aplicaciones del lado derecho (X, Y, Z), servicios que no necesariamente se encuentran en la misma red, sino en un sitio diferente.

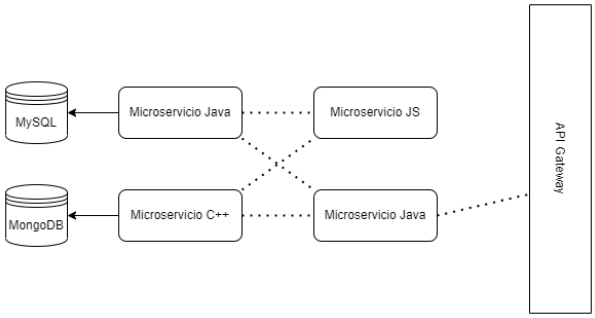
2.1.3.4. Arquitectura de microservicios

En los últimos años, la arquitectura orientada a microservicios se ha vuelto muy popular, debido a la importante tendencia que se le ha dado dentro de las conferencias y eventos de software. Consiste básicamente en desarrollar un pequeño componente de software, autosuficiente, que puedan evolucionar de forma independiente y que cumplan con una única responsabilidad.

Un microservicio se especializa en realizar una pequeña tarea y se enfoca totalmente en ello, se dice que la arquitectura de microservicios es altamente cohesiva, puesto que toda su funcionalidad está extremamente relacionada con resolver un único problema, es decir, una arquitectura orientada a microservicios es todo lo contrario a la arquitectura tradicional (monolíticas), debido a que se busca separar una aplicación compleja en muchos componentes de software pequeños con única responsabilidad (Blancarte Iturralde, 2019).

Una de las mayores ventajas de esta arquitectura, es que los microservicios(componentes) están totalmente encapsulados, lo que permite que puedan evolucionar basándose en su necesidad, además, cada microservicio puede ser desarrollado con la tecnología necesaria para cumplir su tarea asignada, incluso, una buena práctica recomienda el manejo de su propio repositorio de datos.

Figura 2.5. Arquitectura de microservicios



Fuente: Propia

En la Figura 2.5 podemos ver un objeto llamado API Gateway, es muy común utilizar este componente cuando hablamos de microservicios, debido a que sirve como puerta de entrada hacia otros microservicios y también apoya a controlar el acceso.

Además de esto, es importante resaltar que como cada uno de los componentes realiza una tarea, es poco probable que individualmente puedan completar una tarea de negocio, es por esto por lo que es usual ver que los microservicios se comuniquen entre ellos para delegar ciertas tareas, creando una red de comunicación entre ellos.

2.1.3.5. Diferencias entre arquitecturas

En la tabla 2.1 se puede observar cuales son las distintas ventajas y desventajas que se obtiene por manejar una u otra arquitectura, cabe resaltar que las arquitecturas han ido evolucionando a lo largo del tiempo, es por esta razón que algunas comparten ciertas características.

Tabla 2.1. Comparación entre arquitecturas

Características Arquitecturas	Beneficios	Inconvenientes
Monolítica	• Fácil de desarrollar • Autónomo • Pocos puntos de falla • Fácil de probar	• Escalado monolítico • Anclado al Stack tecnológico • Si falla, falla todo • Fácil de perderse
En capas	• Separación de responsabilidades • Fácil de desarrollar • Fácil de probar • Fácil de mantener	• Escalabilidad • Anclado al Stack tecnológico • Performance • Tolerancia a fallos
SOA	• Reduce el acoplamiento • Fácil de probar • Reutilización • Permite crear nuevos servicios en base a los existentes • Escalabilidad • Interoperable	• Latencia • Más puntos de fallo • Necesita un gobierno que ponga orden sobre el uso
Microservicios	• Alta escalabilidad • Agilidad • Facilidad de despliegue • Fácil de probar • Reusabilidad • Fácil de desarrollar • Interoperable	• Múltiples puntos de falla • Trazabilidad • Madurez en un equipo de desarrollo • Performance

Fuente: Propia

2.1.4. Cultura DevOps

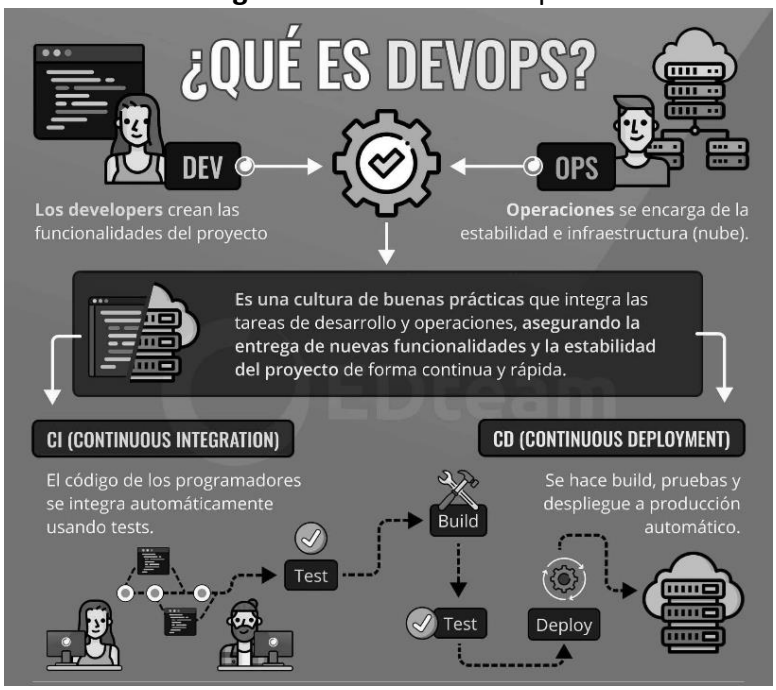
La palabra DevOps es un compuesto de desarrollo (Dev) y operaciones (Ops), este concepto está relacionado con el alto rendimiento de procesos, mejores productos, satisfacción de clientes, métricas y automatización. A partir de 2011, Amazon.com implementó un cambio en producción en 11.6 segundos sin impactar al usuario final, teniendo como resultados una mayor calidad, menor tiempo y costo (Richardson, 2018).

La cultura DevOps consiste en una colaboración más responsable y estrecha entre los profesionales de operaciones y los desarrolladores (equipo de desarrollo) con los productos que se realizan y mantienen, ayudando a las empresas a organizar eficientemente los procesos, recursos y las personas, dando así, un enfoque más unificado hacia el cliente. Es importante resaltar que DevOps no es una filosofía, enfoque, metodología ni herramienta como tal.

Los aspectos fundamentales de DevOps son manejar un correcto control de versiones de repositorios, integración continúa automatizando compilaciones y pruebas (pipelines), entrega continua del suministro del software, infraestructura como código mediante archivos de definición (texto) para montar, revertir o recrear entornos complejos, supervisión y registro mediante recopilación de métricas y aprendizaje validado para mejor los procesos en cada interacción de ciclo (Microsoft Azure, s.f.).

En la Figura 2.6 se observa un conjunto de procesos manejados dentro de DevOps, separadas por los distintos roles, empezando en el desarrollo, aplicando las buenas prácticas entre operaciones y desarrollo que permiten la automatización de los cambios realizados, integrando la automatización de la construcción, pruebas y el despliegue sin la intervención de ejecuciones manuales. En conclusión, la imagen claramente nos indica y resalta que DevOps no es una persona, si no es una cultura que promueve la colaboración entre los diferentes intervinientes.

Figura 2.6. Proceso DevOps



Fuente: (EDucation team , 2022)

Al final del día, cada empresa debe determinar cómo implementar su cultura DevOps adecuado para su producto, negocio o servicio; pero siempre teniendo claro sus objetivos y planes organizacionales y alineados con las necesidades de negocio. También es importante examinar el lado técnico de las empresas, como el conocimiento y la formación de los miembros del equipo, las características de la aplicación y tener muy claro los puntos críticos de mejora.

2.1.5. Rendimiento de entrega de software

La capacidad de utilizar este rendimiento como un diferenciador clave para las empresas que desarrollan software y entregas rápidas les permiten experimentar distintas formas de adopción y satisfacción de los clientes, manteniéndose con el cumplimiento de las exigencias del mercado actual. Un análisis realizado por (DORA DevOps Research & Assessment, 2018), muestra como cualquier equipo en cualquier industria que sea sujeto a un elevado grado de cumplimiento, tiene la suficiente capacidad de obtener un alto nivel de rendimiento de entrega de software.

Tabla 2.2. Aspectos del rendimiento del software

Rendimiento / Aspectos	Élite	Alto	Medio	Bajo
Frecuencia de implementación	Bajo demanda (Múltiples despliegues por día).	Entre una vez por hora y una vez por día.	Entre una vez por semana y una vez por mes.	Entre una vez por semana y una vez por mes.
Tiempo de entrega de cambios	Menos de una hora.	Entre un día y una semana.	Entre una semana y un mes.	Entre un mes y seis meses.
Tiempo de restablecer el servicio	Menos de una hora.	Menos de un día.	Menos de un día.	Entre una semana y un mes.
Porcentaje de cambios por falla	0-15%	0-15%	0-15%	46-60%

Fuente: (DORA DevOps Research & Assessment, 2018)

En la tabla 2.2 se observa los distintos aspectos característicos del rendimiento de la entrega de software, y la comparativa que se tiene entre los diferentes perfiles de rendimiento de las empresas, junto a la peculiaridad propia de cada una de ellas. A continuación, se procederá a indicar un breve análisis y una descripción de cada uno de los aspectos por rendimiento y estabilidad.

Rendimiento

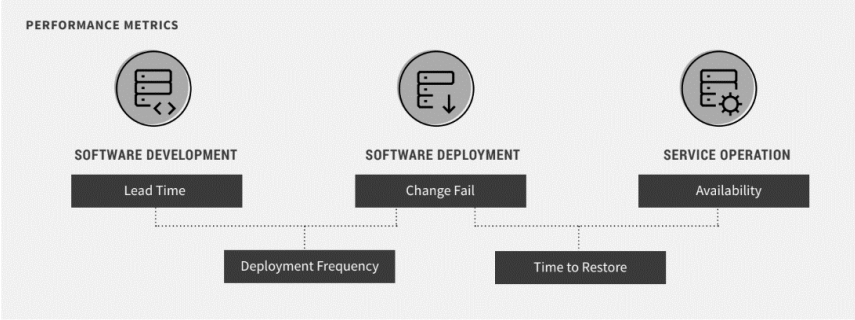
- **Frecuencia de implementación (Deployment frequency):** Los ejecutores tipo élite tienen la capacidad de realizar rutinariamente múltiples implementaciones en comparación a los otros niveles, obteniendo una implementación de nuevo código 46 veces con más frecuencia que los que manejan un bajo rendimiento.
- **Tiempo de entrega de cambios (Lead time for changes):** De la misma manera, los ejecutores élites están optimizando los plazos de entrega, desde el inicio de la codificación hasta la implementación exitosa en producción en menos de una hora, teniendo una velocidad de 2.555 veces más rápida que el grupo de bajo rendimiento.

Estabilidad

- **Tiempo de restablecer el servicio (Time to restore service):** Un ejecutor élite tiene la facilidad de restauración del servicio dentro de una hora, mientras que los grupos intermedios lo efectúan en una semana (168 horas) aproximadamente y un mes para los de bajo rendimiento (5040 horas). Según estos números se determina que las élites manejan una restauración del servicio con una velocidad, 2604 veces más en comparación con el grupo más bajo.
- **Porcentaje de cambios de fallos (Change failure rate):** Sacando la efectividad promedio entre los ejecutores élite y los ejecutores de bajo rendimiento se obtiene una media de 7.5% y 53% respectivamente, relacionando estos 2 valores tenemos una efectividad de siete veces un mejor resultado en fallos para el grupo élite.

Es importante resaltar una métrica adicional de alta relevancia para las organizaciones y es la "**disponibilidad**". En un alto nivel, la disponibilidad representa esa competencia de permitir a los usuarios el acceso autorizado a los datos siempre que se requiera dentro de un tiempo razonable. En el estudio realizado por (DORA DevOps Research & Assessment, 2018), se resalta que los grupos élites reputan una mayor probabilidad de tener una fuerte práctica de disponibilidad de 3.55 veces más en comparación con el nivel más bajo de rendimiento.

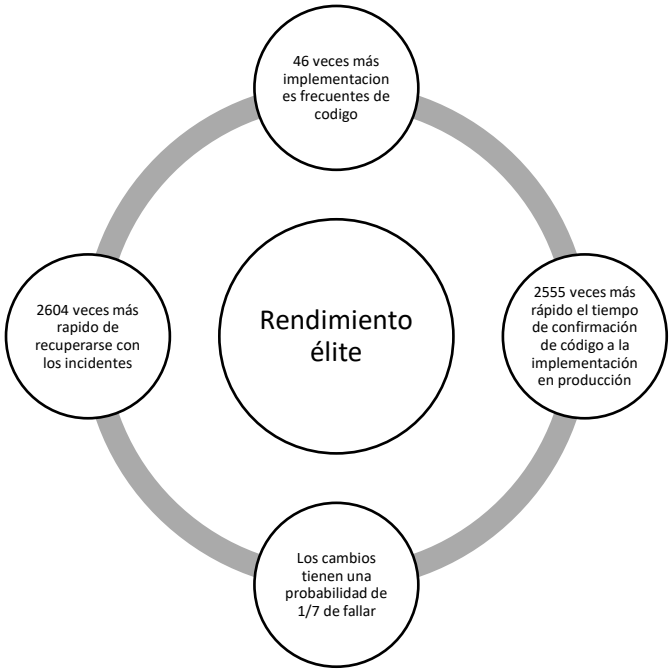
Figura 2.7. Métricas de rendimiento



Fuente: (DORA DevOps Research & Assessment, 2018)

Todas las métricas de desarrollo y entrega de software están relacionadas y dependen una de otra, tal como se aprecia en la Figura 2.7, y son directamente proporcional entre sí, porque si una empieza a descender, el desempeño organización empieza haberse afectado. En resumen, comparando los grupos élites con los intervinientes de bajo rendimiento, se llega al siguiente análisis (Figura 2.8).

Figura 2.8. Características de un rendimiento élite



Fuente: (DORA DevOps Research & Assessment, 2018)

2.1.6. Automatización de procesos

Al aprovechar al máximo la automatización de procesos y tareas repetitivas, las organizaciones mejoran la calidad del trabajo, eliminan reprocesos e inconsistencias al liberar a los trabajadores de ciertas actividades de poco valor (DORA DevOps Research & Assessment, 2018). Con procesos más automatizados, el personal técnico tiene la oportunidad de enfocar su tiempo a tareas de mayor innovación que agregue un valor más real a la organización.

Si bien es cierto, el trabajo manual es muy doloroso de manejar y las personas son conscientes de ello, pero una vez que el trabajo fue automatizado, esto ya no es un dolor y tienden a desaparecer de la atención de las personas.

En la Tabla 2.3, se observan los estudios realizados por Dora y se muestran los diferentes porcentajes de actividades manuales que se manejan dentro de los diferentes niveles de las organizaciones.

Tabla 2.3. Porcentajes de trabajo manual

Rendimiento / Trabajo manual	Élite	Alto	Medio	Bajo
Gestión de configuración	5%	10%	30%	30%
Pruebas	10%	20%	50%	30%
Despliegues	5%	10%	30%	30%
Aprobación de cambios	10%	30%	75%	40%

Fuente: (DORA DevOps Research & Assessment, 2018)

En cuanto a pruebas se trata, es importante resaltar que en el desempeño de nivel medio se ejecutan más tareas manuales que los de bajo rendimiento, esta variación se da debido a que las organizaciones de desempeño medio generan un poco más de desarrollo que las de nivel bajo, por tanto, su nivel de tareas también se ve reflejado en más procesos, algo similar sucede con la aprobación de cambios.

Otro lugar para medir el valor y calidad del trabajo es en como pasan su tiempo los diferentes equipos, es decir, ¿Son capaces de distribuir su tiempo y energía en nuevas características?, o la mayor parte de su tiempo pasan corrigiendo problemas y respondiendo a los defectos ocasionados a los clientes.

En la Tabla 2.4 se visualiza ciertas categorías relevantes que los diferentes grupos planifican su tiempo en cada ciclo o periodo, entre ellas están, la categoría de proactividad, en donde se mide el porcentaje necesario de atención requerido para implementar nuevos cambios, que pueden ser de funcionalidad o infraestructura.

La siguiente categoría se enfoca al trabajo reactivo no planificado o reelaboraciones que abarca los incidentes que pueden salir en una mala implementación enfocada a la calidad. Luego está la categoría de trabajos identificados con los defectos del

usuario final y finalmente se da un espacio a la categoría de trabajo de atención al cliente (research).

Tabla 2.4. Tiempo empleado por categoría

Rendimiento / Trabajo manual	Élite	Alto	Medio	Bajo
NUEVO TRABAJO	*50%*	*50%*	*40%*	*30%*
Trabajo no planificado	19.5%	20%	20%	20%
Corrección de problemas de seguridad	5%	5%	5%	10%
Trabajos en defectos del usuario final	10%	10%	10%	20%
Trabajo de atención al cliente	5%	10%	10%	15%

Fuente: (DORA DevOps Research & Assessment, 2018)

2.1.7. Modelos de servicios en la nube

La infraestructura en la nube está compuesta por componentes de hardware y software indispensables para construir una computación en la nube (Cloud Computing). Los diferentes elementos de esta infraestructura son los recursos de red, el almacenamiento y el procesamiento, así también los distintos recursos virtualizados (Vmware, s.f.).

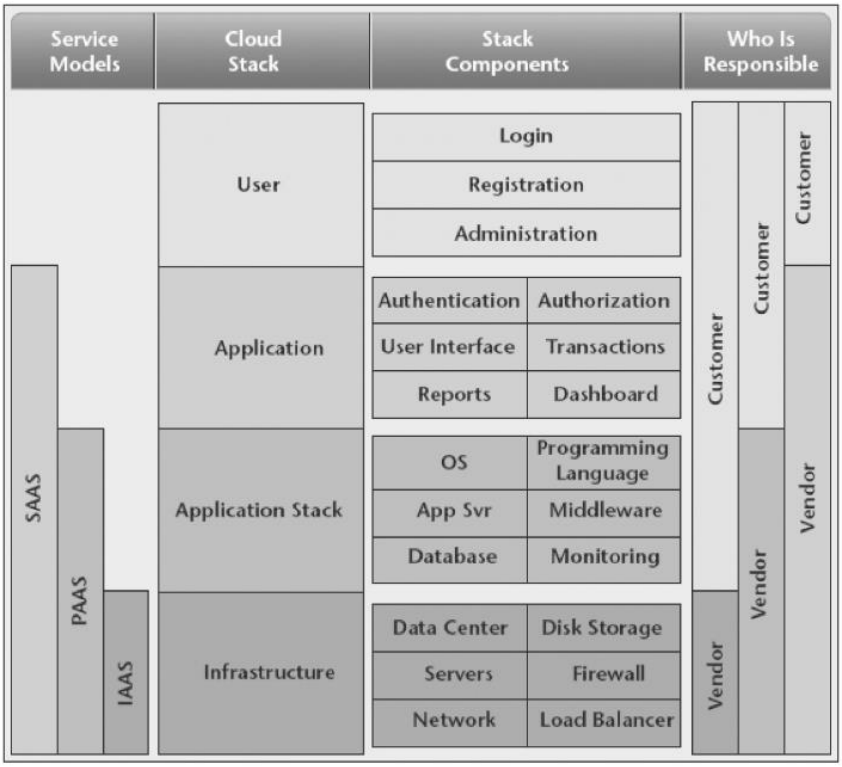
Una de las ventajas importantes de utilizar una infraestructura en la nube es la forma eficiente de gestionarlo, en comparación a la infraestructura tradicional física, en donde generalmente se requiere adquirir y ensamblar los distintos servidores para las aplicaciones y servicios (Kavis, 2014). En la actualidad, los equipos DevOps usando la infraestructura en la nube implementan infraestructura a través de manejo de código. Los tres modelos de servicios que existen en la nube son:

- Infraestructura como servicio (IaaS)
- Plataforma como servicio (PaaS)
- Software como servicio (SaaS)

Cada modelo provee un distinto nivel de abstracción y automatización de tareas que reduce el nivel de esfuerzo y brinda una mayor agilidad al consumidor para implementar un sistema e infraestructura (Olivares & Ramírez, 2019).

En la Figura 2.9 se muestra el Stack Cloud, en la parte interna está el centro de datos clásico, que maneja algo de virtualización, pero no tiene ninguna de las características Cloud. También podemos ver los responsables de cada una de estas interfaces y las distintas capas entre los modelos de servicio de las distintas pilas (servicios, nube y componentes).

Figura 2.9. Stack Cloud



Fuente: (Kavis, 2014)

Infraestructura como servicio (IaaS)

Con la Infraestructura como servicio (IaaS) muchas de las tareas relacionadas con el mantenimiento y administración del centro de datos físicos se abstraen en una colección de servicios disponibles para la ejecución y automatización desde un repositorio de código y/o consola de administración (Le & Gia, 2018).

Atrás quedan los largos periodos de adquisición de componentes de hardware, desempaquetar e instalar el hardware, la infraestructura con IaaS funciona bajo solicitud y en un corto tiempo está operativo. En resumen, mediante IaaS se puede poner a disposición grandes cantidades de centro de datos virtuales para que los clientes de los servicios pongan su atención en crear y administrar aplicaciones.

Una de las innovaciones claves dentro de la cultura DevOps es el manejo de la infraestructura como código (IaC), en donde este paradigma sirve para reproducir o realizar cambios dentro de los entornos de forma automatizada y permite tener un correcto control de versiones en lugar de tener una configuración de infraestructura manual. Esta forma de trabajar es ideal para el manejo natural de infraestructura en la nube, porque los recursos pueden ser aprovisionados y configurados a través de las API (Patni & Tiwari, 2020).

Herramientas como Terraform permiten simplificar el aprovisionamiento de la nube, mediante una definición declarativa y controlada por versiones, facultando tener un entorno de pruebas y producción más rápido, homogéneo y confiable.

Plataforma como servicio (PaaS)

El siguiente nivel en el Stack es la plataforma como servicio, que se asienta y abstrae una gran parte de las funciones de la infraestructura como servicio y permite proporcionar funcionalidades como un servicio más robusto. Los clientes de este servicio tampoco tienen control del software de nivel inferior.

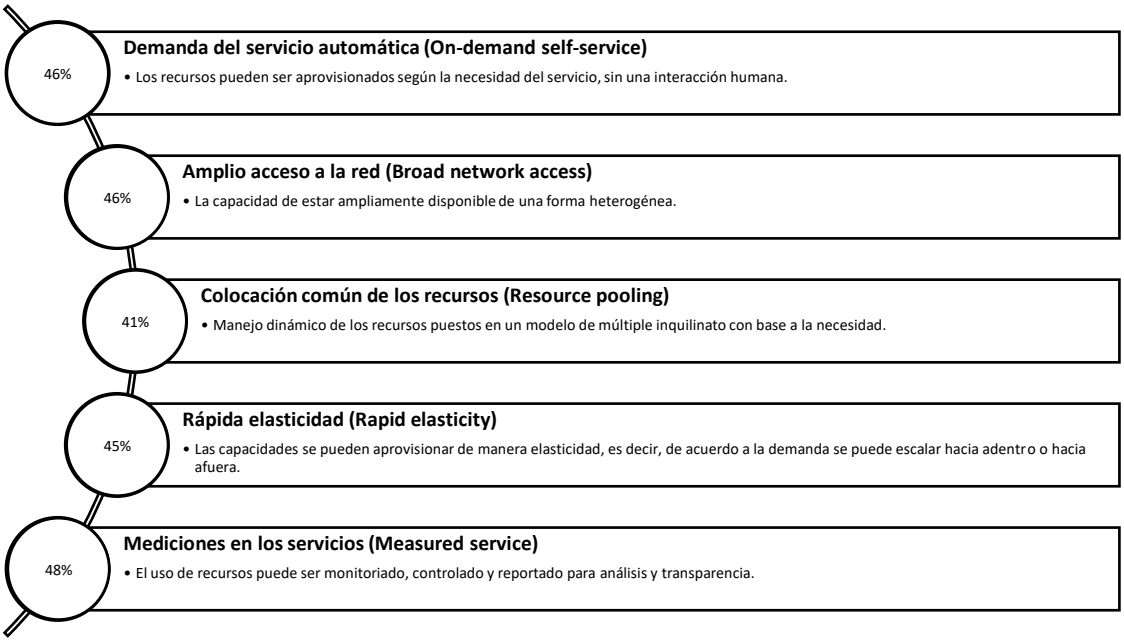
El cliente que utiliza estos servicios no administra ni gestiona los recursos de la plataforma subyacente como la conectividad, el servidor, el sistema operativo o el almacenamiento, sin embargo, si tiene el control sobre las aplicaciones alojadas y posibles configuraciones del entorno de despliegue de la aplicación (Kavis, 2014).

Software como servicio (PaaS)

En la capa superior se tiene el Software manejado como un servicio, que es un aplicativo completo entregado al consumidor del servicio, donde este solo tiene que parametrizar la configuración de la aplicación y administrar sus usuarios (Le & Gia, 2018). El propietario del software controla la lógica e infraestructura de la aplicación relacionada a las implementaciones contratadas. Algunos SaaS están relacionado al manejo de los clientes (CRM), gestión de recursos empresariales (ERP) y otros servicios comerciales comunes.

Existen cinco características esenciales que provee la computación en la nube, a la hora de definir realmente de que significa adoptar una plataforma Cloud. En la Figura 2.9 se detalla cuáles son estas características y el porcentaje de mejoras aplicando un entorno en la nube.

Figura 2.10. Características de computación en la nube



Fuente: (DORA DevOps Research & Assessment, 2018)

3. Desarrollo del proyecto

Para el desarrollo de este trabajo se plantearon dos estrategias de arquitecturas, la primera enfocándose en un escenario de un sistema altamente transaccional y la segunda en un entorno de analítica de datos. En este capítulo se resaltaron las diferentes arquitecturas de acuerdo con los distintos niveles, considerando también los componentes y tecnologías necesarias.

3.1. Estrategia para sistemas altamente transaccionales

Los sistemas transaccionales son aquellos sistemas diseñados específicamente para el manejo de información, es decir, desde la gestión de recolección, modificación, almacenamiento y recuperación que generan por todas las transacciones de la organización o negocio, incluyendo confiabilidad, rendimiento y consistencia de los datos (Novoa & Maldonado, 2018). Partiendo de esta definición, los sistemas altamente transaccionales manejan un gran volumen de información por segundo (TPS), automatizando las tareas operativas de la organización y generalmente sé enfocada a las áreas de recursos humanos, ventas, marketing, administración y finanzas.

A continuación, se procede a presentar un caso de uso detallado con los componentes necesarios, como las definiciones requeridas y los involucrados para poder estructurar la arquitectura y validar la correcta implementación.

3.1.1. Caso de uso

Tabla 3.1. Caso de uso transaccional

Nombre del caso de uso: Implementación de una arquitectura para un sistema altamente transaccional	**ID:** 1.0
Módulo: Transacciones de una institución financiera	
Actores: Equipo de desarrollo, arquitecto de soluciones, ingeniero de infraestructura y soluciones tecnológicas.	
Interesados: Cliente externo, cliente interno, especialista de negocio, Product manager.	
Descripción: El escenario analizado fue un sistema financiero, donde, en general, se tienen las actividades de depósito, transferencias, pagos, retiros de agencias o ATM, créditos, etc. Produciendo una gran cantidad de transacciones de clientes internos y externos.	
Pasos realizados:	**Información de los pasos:**
1.- Diseño de arquitectura lógica	Información de todos los componentes macro que posee la arquitectura planteada.
2.- Diseño de arquitectura tecnológica	Se detallan las tecnologías utilizadas dentro de la arquitectura lógica.
3.- Diseño de arquitectura física	Consiste en el diseño real de los componentes implementados en AWS.
Precondiciones: Conocer el modelo de negocio de las transacciones.	
Postcondiciones: Validar las condiciones del negocio enfocada a las transacciones.	
Suposiciones: La implementación de la arquitectura aborda todos los elementos principales del negocio de una institución financiera altamente transaccional.	
Garantía del éxito: El diseño de la arquitectura física debe contener los elementos necesarios para aplicar los distintos principios de escalabilidad.	
Prioridad: Alta	
Riesgo: Alto	

Fuente: Propia

3.1.2. Descripción de los actores principales

Dentro de un sistema financiero hay varios participantes dentro del ciclo de vida de un sistema transaccional, algunos permiten la evolución del software, nuevas definiciones de tecnología y otros colocan las reglas de negocio.

En la figura 3.2, se observar cómo los diferentes actores tienen algunas responsabilidades dentro del sistema financiero, buscando resolver las problemáticas modernas.

Tabla 3.2. Actores de un sistema transaccional

Actor	Descripción
Cliente externo	Está relacionado con los usuarios que hacen uso de la funcionalidad de los sistemas transaccionales, pero que no forman parte de la organización.
Cliente interno	Es todo usuario responsable del uso del sistema, pero con la característica que está dentro de la organización.
Equipo de desarrollo	Son un grupo de personas que manejan la parte técnica del producto o proyecto, y se encargan de materializar los modelos de arquitectura de software.
Arquitecto de soluciones	Tiene la responsabilidad de evaluar las necesidades de negocio de una organización, y juega un rol fundamental en cualquier organización que requiere alinear sus objetivos con el negocio.
Ingenieros de infraestructura y plataformas tecnológicas	Gestiona todo lo relacionado con servicios de infraestructura, realizando el soporte a usuarios y servicios informáticos, que operan dentro de las plataformas TI.
Product Manager o director de producto	Es la persona encargada de identificar las necesidades del negocio y satisfacerlas a través del desarrollo de un proyecto, también es un puente entre los diferentes miembros de equipo como negocio, experiencia de usuario y tecnología.

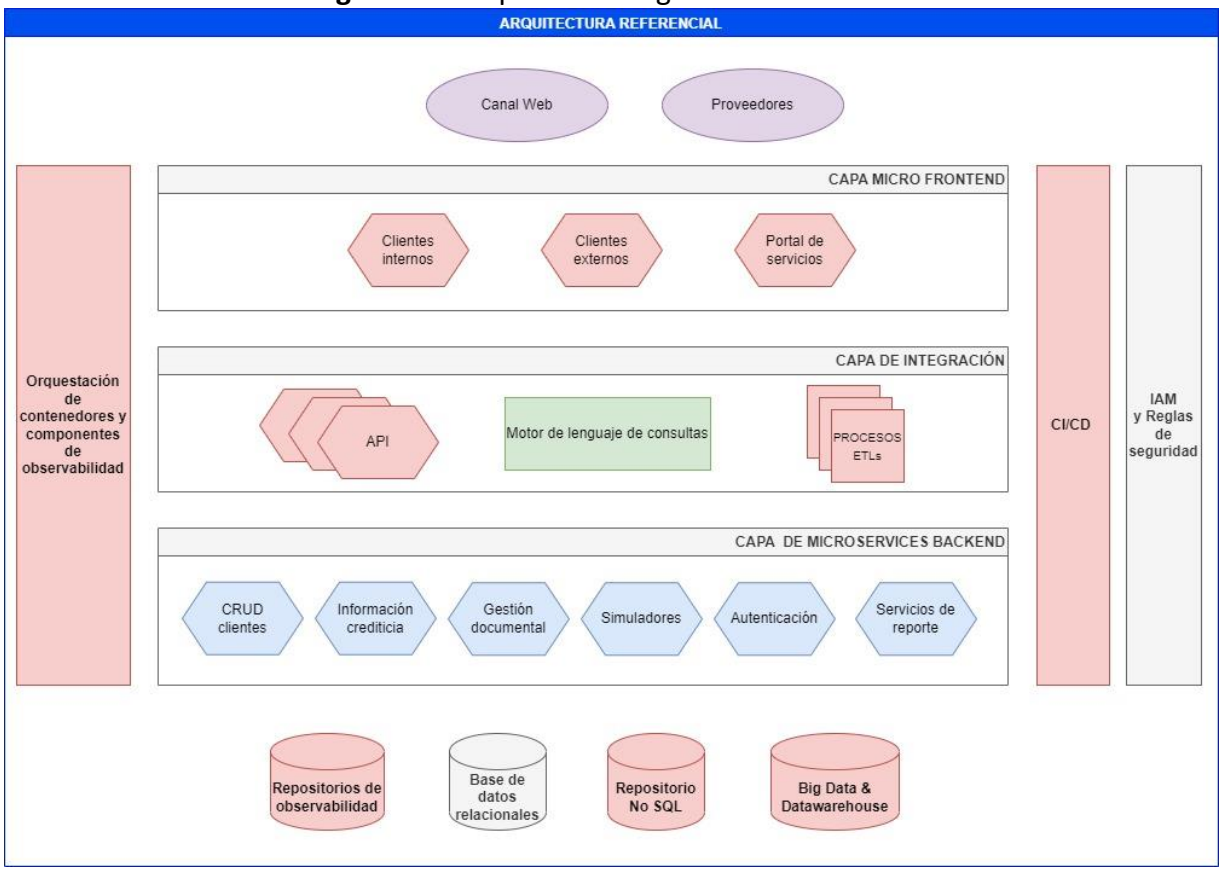
Fuente: Propia

3.1.3. Arquitectura de alta disponibilidad

3.1.3.1. Arquitectura lógica

En la Figura 3.1, se observan los diferentes grupos que componen una arquitectura lógica, como los canales de acceso, por ejemplo, directamente desde un portal web, sin embargo, es importante resaltar que puede haber otros medios que no necesariamente utilizan un Frontend, si no solamente consumen servicios, algunos componentes transversales, componentes de orquestación, reglas de entorno Cloud, repositorios y los distintos componentes de software que manejan las aplicaciones.

Figura 3.1. Arquitectura lógica transaccional



Fuente: Propia

Con este nivel de arquitectura se pretende consolidar los niveles de capas, agrupar los elementos modelados y plantear las vistas estructurales de manera abstracta, para posteriormente ser llevada a una arquitectura física que consista en determinar que componentes tendrán asignados las respectivas tareas, manteniendo una gestión centralizada entre los distintos módulos.

En la Tabla 3.3, se detallaron estos componentes de la arquitectura lógica con la respectiva descripción y la capa que le corresponde, resaltando los aspectos más relevantes utilizados dentro de la Figura 3.1.

Tabla 3.3. Componentes de una arquitectura lógica transaccional

Capa	Componente	Descripción
Microfrontend	Presentación	El microfrontend es una arquitectura en donde la aplicación web se divide en distintos componentes o funciones individuales, que son implementados de manera independiente con el fin de tener la misma flexibilidad y velocidad que los microservicios en el BackEnd.
Contenedores	Orquestador y componentes de observabilidad	Utilizando el concepto clave de la contenerización y contenedor del sistema de transporte hacia el entorno IT, podemos utilizar los contenedores para empaquetar código de distintos lenguajes y administrarlos en herramientas de contenerización como Docker y Kubernetes, permitiendo tener la observabilidad de cada uno de estos componentes de software.
Integración	API	Este componente maneja la traducción agnóstica de las tecnologías desarrolladas para el negocio, permitiendo al producto tener su propia dependencia.
BackEnd	Microservicios	Los microservicios se utilizan para el manejo distribuido, flexible y altamente escalable de la funcionalidad del negocio, permitiendo su despliegue y adaptación de manera independiente.
Componentes de arquitectura	Base de datos	Para el manejo de los datos generados dentro de las transacciones es importante tener un repositorio que se ajuste a la necesidad de procesamiento de la información, entre estos tipos podemos mencionar las bases de datos SQL, No SQL, distribuidas, de grafos, cache, etc.
Componentes de arquitectura	Sistemas externos	La comunicación de los sistemas altamente transaccionales no se desarrolla simplemente con los sistemas internos, sobre todo si estamos analizando los sistemas financieros, existen transacciones que permiten a los usuarios completar actividades requeridas por el negocio como obtener información del registro civil, información de entes de control como el SRI, etc.
Componentes de seguridad	IAM y reglas de seguridad	El sistema de administración de accesos e identidades (IAM) contiene un marco de reglas de negocio para la gobernar las identidades electrónicas de una manera automatizada. Es importante tener este control para el correcto manejo de la autenticación, autorización y autoría por usuario.

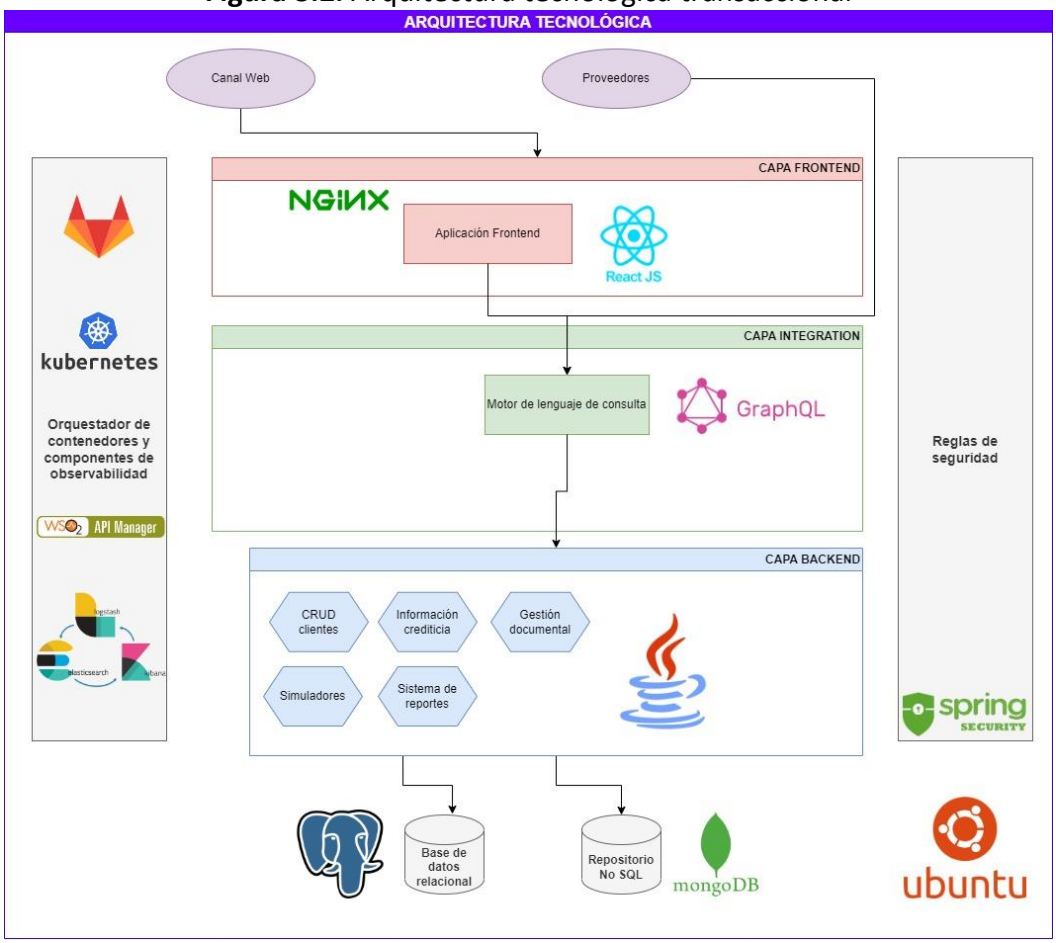
Fuente: Propia

3.1.3.2. Arquitectura tecnológica

La arquitectura tecnológica de componentes proporciona información de las distintas plataformas, tecnologías, lenguajes y framework utilizados dentro de un contexto de implementación de componentes. Esta arquitectura también traduce la complejidad del mundo real en elementos TI más identificable manteniendo una eficiencia y la agilidad.

En la Figura 3.2, se observa la arquitectura tecnológica, que, partiendo de una arquitectura de la sección 3.1.3.1 (arquitectura lógica), se colocaron un nombre y tecnología a los componentes anteriormente mencionados de una forma superficial.

Figura 3.2. Arquitectura tecnológica transaccional



Fuente: Propia

Para las tecnologías que a continuación se detallan, se analizaron principalmente la flexibilidad del lenguaje o componente, basándose en la mejor consideración de adaptabilidad para un sistema altamente transaccional.

Como interacción directa entre el cliente y los distintos servicios (incluyendo los microfrontends), tenemos un servidor web de código abierto de nombre **NGINX**, colocado justamente en la capa frontEnd como proxy inverso de acceso interno. Es necesario resaltar que NGINX supera en pruebas de rendimiento, bajo consumo de memoria a otros populares servidores web, es por esta razón que varias empresas grandes de alto perfil como IBM, Google, Apple, Intel, Cisco, Facebook, entre otras lo utilizan.

Para los microfrontend se utilizó **ReactJS**, que es una librería de JavaScript de código abierto que se enfoca en una fácil creación de componentes dinámicos y reutilizables en las distintas interfaces de usuario. Otra característica importante de ReactJS es que su paradigma de desarrollo está basado en una programación orientada a componentes, que permite manejar las funcionalidades de manera modular. Hoy en día, grandes empresas de primer nivel como Uber, PayPal, Airbnb, Netflix, etc. usan esta librería dentro de su desarrollo.

Como administrador de API, se utilizó **WSO2 API MANAGER**, porque combina la facilidad de la gestión de las API con el análisis, el monitoreo y uso de estas, aplicando el patrón de diseño de Backend For Frontend a los servicios para los diferentes formatos de datos que pueden ser XML, JSON, REST y SOAP. También permite establecer políticas de utilización como rate limit, restringir los orígenes y controlar el acceso por medio de Key Manager. Es importante mencionar el aporte que tiene WSO2 dentro del desarrollo y las pruebas, porque permite realizar Mock-API de las API sin ningún riesgo, obteniendo un feedback de los usuarios de prueba desde el primer momento.

Para el manejo de microservicios, la tecnología que mejor se adapta con su principio de “Escribir una vez, ejecutar en todas partes”, es **JAVA**, desde el punto de vista empresarial, aporta un amplio marco de trabajo de código abierto, permitiendo

crear software más mantenible y manejable. Java es multiplataforma, es decir, su compilación no genera código nativo, permitiendo ejecutar su código en la máquina virtual de Java (JVM) de forma portable. Java tiene el soporte de empresas mundialmente implementadas como Facebook, Oracle, Twitter, y muchas más, que resaltan su solidez y fácil mantenimiento.

También se escogió **KUBERNETES**, como suite de contenedores, por el entorno administrativo centrado que tiene, la forma de orquestar las redes, almacenamiento y la infraestructura de cómputo como tal, permitiéndonos obtener toda la parte simple de las plataformas como un servicio (PaaS) y conservando la flexibilidad de la infraestructura como un servicio (IaaS) dentro de los distintos ámbitos. Kubernetes abarca todo el ecosistema de herramientas y componentes que permiten un sencillo escalamiento, el despliegue y administración de aplicaciones. Es importante resaltar que Kubernetes no es un PaaS tradicional, porque no opera en el ámbito de hardware, sino nivel de contenedor.

Existen algunos repositorios de datos que se ajustan mejor a la necesidad de la información, entre ellos se eligió una base de datos relacional, un motor de **PostgreSQL**, por su alta disponibilidad, fácil manejo, sostenibilidad de altos volúmenes de datos, y al ser de código abierto, da la libertad en el desarrollo para que existan grandes colaboradores profesionales que ha permitido el crecimiento en múltiples plataformas. Para el uso de bases de datos no relacional, se consideró importantemente el uso de **MongoDB**, debido a que a medida que las aplicaciones creces o evolucionan se implementan las mejores prácticas de diseño de esquema bajo demanda para obtener la mayor eficiencia.

Hoy en día las herramientas de CI/CD proporciona todo el apoyo necesario para el levantamiento de las aplicaciones, por tal razón, se utilizó **GitLab CI/CD** como herramienta para el uso de las metodologías de integración, entrega e implementación continua abarcando desde el desarrollo, las pruebas y la publicación del software sin la necesidad de un agente externo o factor humano.

Una de las características de GitLab es la forma que permite realizar múltiples funcionalidades en un solo lugar, así como unir peticiones, escribir código y verificar el cumplimiento de funcionalidades específicas como validaciones y seguridad de despliegue.

Y finalmente, como componente de observabilidad se escogió un **framework ELK (Elasticsearch, Logstash y Kibana),** que proviene de las siglas de los tres proyectos compuestos, Elasticsearch como motor de búsqueda y analítica, Logstash como procesador de datos de lado del servidor que transformarlos la información y envía a Elasticsearch, y Kibana que es la interfaz gráfica de visualización de los datos por parte del usuario.

Los principales beneficios de usar EKL, son la detección de incidencias en tiempo real, eliminar la falta de consistencia de información, generar un formato estándar de tiempo y descentralizar todo el sistema información de los logs. Es necesario resaltar que cada uno de estos componentes de EKL se pueden utilizar de forma integrada o separada.

3.1.3.3. Arquitectura física

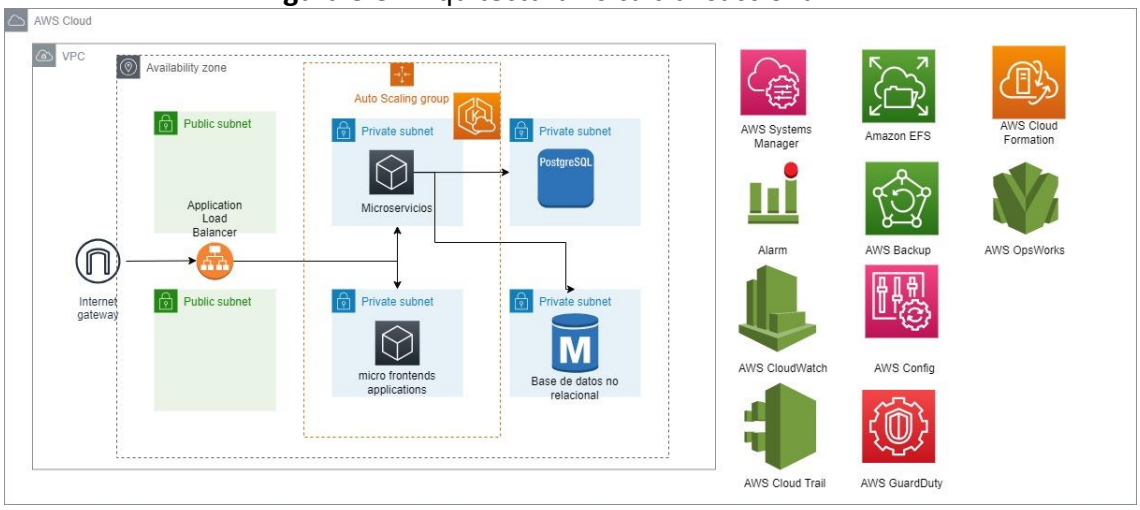
Una vez detallado el nivel tecnológico de los componentes a utilizar, se procedió a realizar el levantamiento de una arquitectura física optimizada para el cómputo con artefactos excelentemente operacionales utilizados dentro de AWS (Amazon Web Service), confiando en la escalabilidad, la estabilidad, la robustez y la seguridad para las implementaciones de los workloads con características elásticas y un gran número dinámico de peticiones.

Los desafíos se presentan de acuerdo con la naturaleza del negocio, al ser una arquitectura altamente transaccional, el número de transacciones es agresivo, por este motivo es importante el correcto análisis de la escalabilidad y elasticidad dentro del diseño de la arquitectura.

El diagrama de arquitectura física consistente y confiable, muy eficiente y optimizado en costos, con herramientas para evaluar constantemente la carga de

trabajo, el manejo de registro de cambios e identificación de problemas de alto impacto, así como se puede observar en la Figurar 3.3.

Figura 3.3. Arquitectura física transaccional



Fuente: Propia

Visibilidad de la arquitectura

Se debe comenzar con añadir indicadores a los servicios, con el fin de agrupar la infraestructura en grupos virtuales, para poder identificar como una única carga de trabajo.

Una vez colocado los indicadores se debe usar *AWS Systems Manager* para la instalación de los agentes, permitiendo tener el estado de la operación de los recursos con un agente de monitoreo. Este agente de monitoreo es *Amazon CloudWatch Logs* que obtiene los registros generados como logs, para procesarlos dentro de un tablero de monitoreo de servicio.

El manejo de tableros de información ayuda a tener una mejor visibilidad de los servicios y también alertar de excesos de recursos como por ejemplo el almacenamiento máximo disponible, finalmente se utiliza *AWS CloudTrail* para el manejo de la trazabilidad de los cambios efectuados a nuestra arquitectura.

Alta disponibilidad

Con la visibilidad de los servicios, podemos entender cómo está el entorno de los servicios de nuestra arquitectura durante la distinta carga aplicada, sin embargo, si

solo usamos una única zona de disponibilidad, es posible que no sea la suficiente y no tengamos una funcionalidad adecuada.

Entonces se debe usar múltiples zonas de disponibilidad y configurar las subredes públicas y privadas dentro de ellas, esto significa la ejecución de las instancias en diferentes localidades de una región de AWS, con una conectividad optima a baja latencia en otra zona de disponibilidad de la misma región.

Para este proceso de distribución y balanceo se utiliza *Elastic Load Balancers* en las subredes publicas permitiéndonos crecer de manera horizontal a nuestras instancias. La seguridad incrementa al ejecutar instancias dentro de subredes privadas.

Amazon EKS (Amazon Elastic Kubernetes Service) es un servicio de computación en la nube proporcionado AWS, que facilita la ejecución de aplicaciones en Kubernetes como orquestador de contenedores de código abierto. Kubernetes es ampliamente utilizado para el despliegue, la gestionar y el escalamiento de aplicaciones en contenedores de manera simple y eficiente.

Amazon EKS simplifica esta tarea al proporcionar una plataforma completamente administrada para la ejecución de clústeres de Kubernetes. También se tiene la posibilidad que los usuarios puedan lanzar y administrar fácilmente sus clústeres sin la preocupación de la infraestructura latente, las actualizaciones de Kubernetes o la escalabilidad.

Servicios administrativos

Con los servicios administrativos aumenta la confiabilidad y mejora la facilidad de uso de asistencia, debido a que se le puede delegar a AWS la tarea de administración de servicios desplegados y darle una buena atención a la operación de las aplicaciones.

Con *AWS Backup* se puede automatizar y centralizar el manejo de procesos de protección de información, debido al efectivo y completo servicio que nos ofrece, simplificando la protección de datos a gran escala.

Con *Amazon Elastic File System (EFS)* el cual es un servicio de almacenamiento de datos elástico, que como principales características tiene el efecto de compartir datos sin la necesidad de un aprovisionamiento o la administración del almacenamiento por nuestra parte.

Ítems de seguridad

La seguridad es el principal punto para cuidar, por tal razón, bloqueamos el tráfico no deseado hacia la subred privada virtual, mediante listas de control de accesos. Procederemos a bloquear puertos indeseados con los Firewalls tipo Stateful llamados grupos de seguridad, estos a su vez protegen la arquitectura con respecto a la red.

Para proteger nuestra línea base de configuración, *AWS Config* nos provee el servicio que evalúa las configuraciones de movimientos y recursos con un constante monitoreo, supervisando el cumplimiento y comprobando la aplicación de la gobernanza corporativa.

Para la detección de intrusiones se recomienda el uso de *Amazon GuardDuty*, que monitorea constantemente la actividad maliciosa, protegiéndonos nuestra información, cuentas y cargas de trabajo. Este servicio utiliza técnicas de machine learning como aprendizaje continuo para la detección de anomalías y amenazas.

Y, por último, aprovechamos la utilización de *Elastic Load Balancer* y *AWS WAF* para protegernos de contra ataques comunes hacia nuestras aplicaciones, o de bots que afecten nuestra seguridad, disponibilidad o un excesivo consumo de recursos. *AWS Shield* también nos cuida de los ataques de denegación de servicio de manera continua.

Elasticidad y escalabilidad

El principal identificador dentro de uso de Cloud es la escalabilidad, que no es más que esa capacidad de adaptarse a una mayor demanda cuando es requerido, y la elasticidad definida como la suficiencia de adaptación de cambios repentinos bajo la demanda.

Con *AWS CloudFormation* nos ayuda a crear recursos de manera sencilla, basado en IaC (Infraestructura como código), partiendo de una plantilla de texto, con los indicadores de servicio a utilizar y los recursos necesarios para utilizar junto a sus dependencias.

Y, por último, con *AWS OpsWorks* que es una herramienta enfocada en automatizar los servidores y su configuración usando el código generado, también como detalle adicional con la utilización de *AWS OpsWorks* se puede realizar la implementación de instancias EC2 mediante el Chef y Puppet.

3.2. Estrategia para sistemas de analítica de datos

La analítica de datos se enfoca directamente al análisis de todos los datos en tiempo real, como históricos, estructurados y no estructurados, y cualitativos, para generar patrones o identificar un conocimiento importante para la toma de decisiones organizaciones, y en otros escenarios se realiza la automatización de decisiones cuando se permite conectar con la inteligencia de negocios de la organización.

La analítica también permite a las organizaciones ejecutar una transformación digital a su empresa, y mejora la cultura, convirtiéndola en una organización innovadora y con una buena visión en la toma de decisiones.

A continuación, se mencionan algunos escenarios de caso de uso para la analítica de datos, y se tomará un pseudo ejercicio para realizar la implementación teoría de este caso de uso:

- Detección de fraudes
- Análisis de operaciones
- Realización de estudios de mercado
- Gestión de riesgos
- Detección de anomalías
- Análisis de estudios de mercado

3.2.1. Caso de uso

Tabla 3.4. Caso de uso analítico

Nombre del caso de uso: Detección de fraudes	**ID:** 2.0
Módulo: Creación de cuentas de usuarios	
Actores: Equipo de riesgos, arquitecto de soluciones, ingeniero de datos.	
Interesados: Institución financiera de grandes datos.	
Descripción: El escenario para implementar se orienta en el estudio de fraudes en la creación de múltiples cuentas para lavado de dinero.	
Pasos realizados:	**Información de los pasos:**
1.- Diseño de arquitectura lógica	Información de todos los componentes macro que posee la arquitectura planteada.
2.- Diseño de arquitectura tecnológica	Se detallan las tecnologías utilizadas dentro de la arquitectura lógica.
3.- Diseño de arquitectura física	Consiste en el diseño real de los componentes implementados en AWS.
Precondiciones: Conocer el modelo de información de creación de cuentas.	
Postcondiciones: Validación de modelos generados con las herramientas seleccionadas.	
Suposiciones: La ejecución de esta estratégica nos permitirá identificar la creación fraudulenta de cuentas que tienen como objetivo la colocación de dinero en el mercado actual.	
Garantía del éxito: El modelo de arquitectura debe permitir tomar decisiones con los resultados generados.	
Prioridad: Alta	
Riesgo: Alto	

Fuente: Propia

3.2.2. Descripción de los actores de analítica

En la Tabla 3.5. identificamos brevemente los actores que se involucran como implementadores y validadores de un sistema analítico, es importante resaltar que los principales actores que se centran en esta tabla son de roles de negocio, y personas enfocadas a la toma de decisiones organizacionales, en conjunto con un equipo técnico que realiza la implementación y validación.

Tabla 3.5. Actores de un sistema de analítica de datos

Actor	Descripción
Arquitecto de soluciones	Tiene la responsabilidad de evaluar las necesidades de negocio de una organización, y juega un rol fundamental en cualquier organización que requiere alinear sus objetivos con el negocio.
Ingenieros de datos	Son las personas responsables de definir, administrar, organizar el almacenamiento de la información, para que posteriormente pueda ser procesado u organizado en base a la necesidad de los interesados.
Equipo de análisis de riesgos	Este grupo de personas identifican, examinan y validan las posibles amenazas que afecten los activos o el capital de una organización. Existen varios grupos de análisis de riesgos entre ellos están los de mercado, de operación, los crediticios y los de reglamento organizacional.

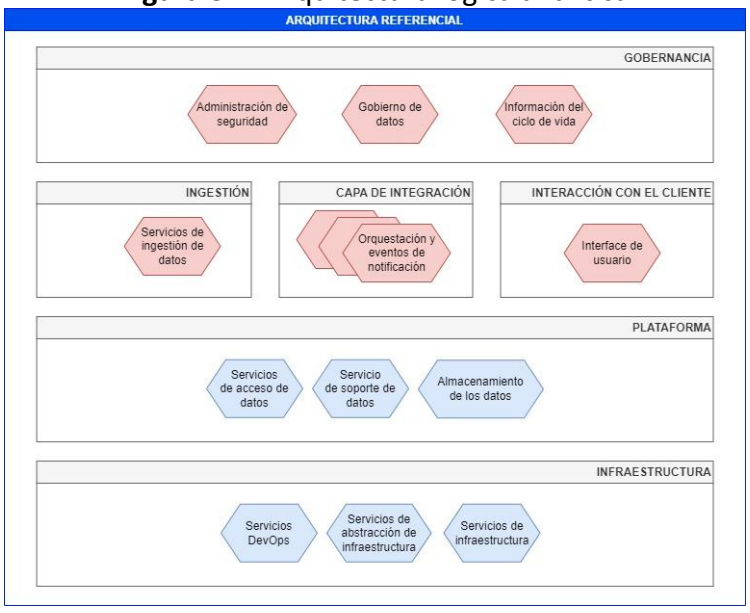
Fuente: Propia

3.2.3. Diagramas de una arquitectura analítica

3.2.3.1. Arquitectura lógica

La arquitectura lógica nos permite visualizar de manera macro, todos los componentes principales utilizados, como se observa en la Figura 3.4, resaltando las principales capas o pilares que proporcionan un servicio independiente, que se integran unos con otros para obtener toda la funcionalidad completa.

Figura 3.4. Arquitectura lógica analítica



Fuente: Propia

Tabla 3.6. Capas de la arquitectura lógica

Capa	Descripción
Gobernanza	Esta capa tiene como objetivo almacenar y mantener la seguridad, las políticas del ciclo de vida de la información y la gobernanza de los datos. También incluyen los procesos y políticas para implementar las prioridades de negocio y el orden de los datos.
Infraestructura	Servicios de gestión y usos para el correcto funcionamiento continuo de la plataforma. Estos son servicios de DevOps para la administración de las entregas, los recursos y el abastecimiento de los clústeres.
Plataforma	Son una combinación de los componentes y productos que proporciona el alojamiento y ejecución de los servicios básicos, facilitando las tareas de CRUD de datos, ejecución de lógica de negocio, etc.
Ingestión	Esta es la capa configurable y automatizada para permitir la carga de datos recolectados de los clientes internos relevantes y externos a la organización.
Aplicaciones	Contiene todas las aplicaciones específicas de la empresa para el procesamiento y cálculo de los datos adecuados sobre las plataformas que permitan precisar una gran cantidad de información, con un correcto manejo y administración de los recursos y los datos.
Componentes de arquitectura	Esta capa es la vista del usuario para la interacción, búsquedas de registros, análisis estadísticos, minería de datos, etc.

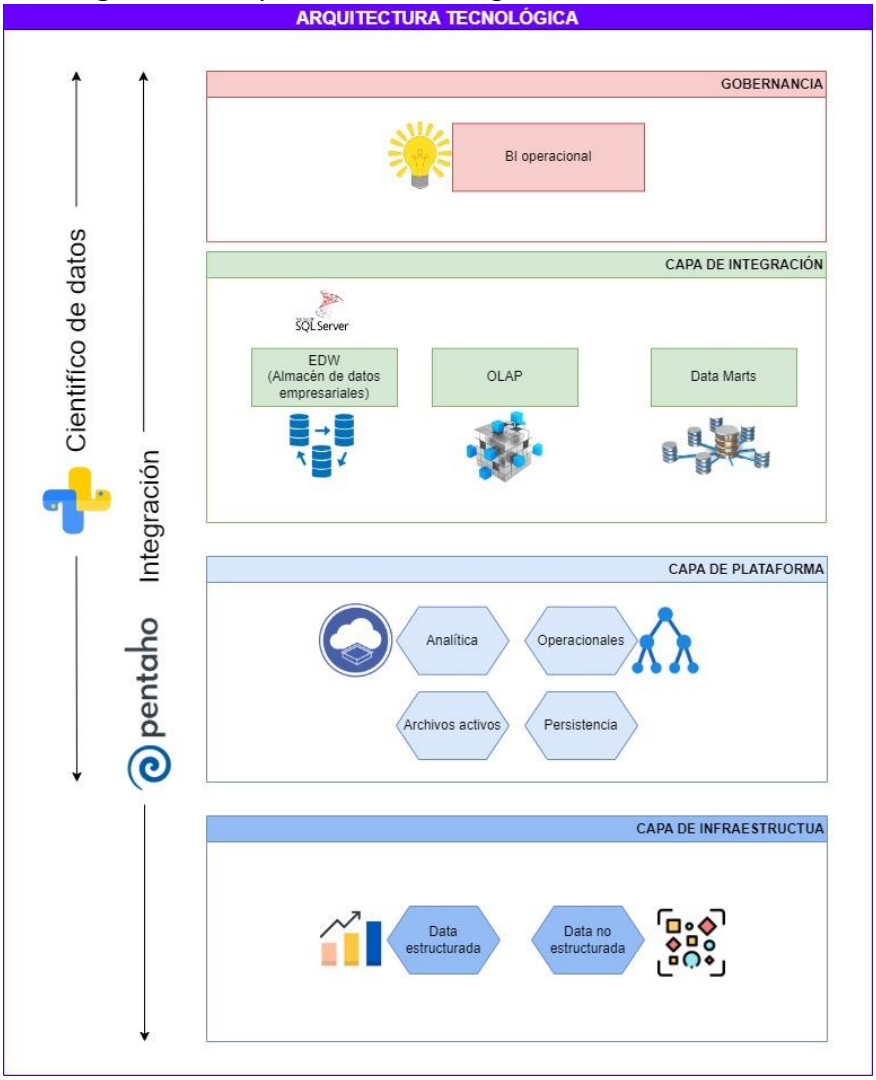
Fuente: Propia

3.2.3.2. Arquitectura tecnológica

En la actualidad, las tecnologías de análisis de información nos proporcionan una estructuración analítica moderna de datos, que no precisamente sustituyen los sistemas tradicionales. En la Figura 3.5, se plantea el reto de como diseñar una arquitectura sostenible de analítica empresarial que se incorpore al manejo de Big Data.

Entre las ventajas de las arquitecturas modernas de datos, tenemos que nos permite juntar la velocidad del procesamiento con el análisis y el procesamiento de la data, desde los grandes repositorios de información, como los DataWarehouse, para todas las áreas de las organizaciones.

Figura 3.5. Arquitectura tecnológica de análisis de datos



Fuente: Propia

Persistencia de datos

En esta capa, por medio de las distintas herramientas como Apache Sqoop o Apache Flume, se recogen constantemente los datos de los diferentes orígenes internos relevantes y externos de la organización. A este repositorio donde se almacenan el contenido de la información hasta que sea utilizada se le conoce como Data Lake.

Esta información puede ser de tipo estructurados, semiestructurados y no estructurados. En esta capa se procesa los datos que seguirán a la siguiente capa, porque no necesariamente toda esta información puede ser utilizada, sin embargo, la data que secundaria se mantendrá dentro de esta capa para futuros casos de uso y aplicación de esta.

Tratamiento de los datos

Los analistas y científicos de datos mantienen acceso a las distintas capas de persistencias, donde se realizan los análisis exploratorios, experimentaciones y de descubrimiento de las nuevas ideas de negocio. Estas personas también son encargados de dar la forma y catalogar los datos para que serán utilizados en capas superiores de la arquitectura de manera automática para más usuarios de negocio.

Esta información escalada a las capas superiores, principalmente se consumen por los niveles más estructurales de la organización, donde el ciclo de toma de decisiones tiene ejecución desde periodos diarios hasta periodos personalizados, se recomienda el paso de esta información al DataWarehouse organizacional para la exploración más profunda mediante tecnologías BI.

Integración entre capas

El proceso de integración de la información cumple el siguiente ciclo de vida: se consumen los datos desde los diferentes orígenes, se organiza la información en las capas de categorización de datos, se aplica una transformación, se procesan realizando las mejoras y ajustes y finalmente se carga dentro del DataWarehouse Empresarial. Independientemente de la personalización de las integraciones, es

importante elegir herramientas que permitan automatizar los procesos y auditar los flujos de ejecución de estos.

Tener los mecanismos de procesamiento, la gestión de los contenidos, la administración y normalización de la información, y la seguridad, nos permite tener el mayor éxito en la implementación de estrategias de análisis de una gran cantidad de datos.

Power BI es una poderosa herramienta de análisis y procesamiento de datos que permite visualizar esta información. Su popularidad ha crecido significativamente debido a sus ventajas y beneficios para profesionales y grandes organizaciones en el campo de la analítica de datos. A continuación, se presentan los principales puntos favorables para utilizar Power BI como herramienta de análisis de datos en tu tesis:

1. **Facilidad de uso y aprendizaje:** Power BI cuenta con una interfaz sencilla y amigable de fácil aprendizaje para usuarios de distintos niveles de habilidad técnica. No se requiere una amplia experticia en programación o análisis de datos para comenzar a utilizar la herramienta.
2. **Conexión con múltiples fuentes de datos:** mediante Power BI se puede establecer conexiones y consolidar información de diversas fuentes, como bases de datos locales, hojas de cálculo, servicios presentados en la nube, entre otros. Esto posibilita la integración de los distintos orígenes de datos entre sistemas y fuentes para un análisis integral.
3. **Visualizaciones interactivas y atractivas:** Una de las fortalezas de Power BI radica en su capacidad para crear visualizaciones interactivas y dinámicas. Gráficas, tablas y dashboard posibilitan una comprensión más rápida y profunda de los datos, facilitando la toma de decisiones informadas.
4. **Actualizaciones automáticas de datos:** Power BI puede programarse para actualizar automáticamente los informes y dashboard con la última información disponible. Esto garantiza que los usuarios siempre tengan datos actualizados y relevantes para sus análisis.

5. **Capacidad para compartir y colaborar:** Power BI ofrece opciones para distribuir informes y dashboard con otros usuarios y equipos, tanto dentro como fuera de la organización. La colaboración se vuelve más fluida y efectiva, lo que promueve la colaboración en equipo y la toma de decisiones basadas en datos.
6. **Integración con otras herramientas de Microsoft:** Power BI tiene una integración inherente con otras herramientas de Microsoft, como Excel y Azure. Esto permite un flujo de trabajo más eficiente y la posibilidad de aprovechar recursos y datos existentes en otros sistemas de la empresa.
7. **Seguridad y cumplimiento:** Power BI proporciona funciones de seguridad avanzadas para resguardar los datos y controlar el acceso a información confidencial, cumpliendo con los estándares de seguridad y privacidad, lo que es esencial para proteger la confidencialidad de los datos empresariales.
8. **Escalabilidad y flexibilidad:** Power BI se adapta a diferentes tamaños de empresas y necesidades, desde pequeños startups hasta grandes corporaciones. Su escalabilidad y flexibilidad hacen que sea una opción adecuada para empresas en crecimiento y en constante cambio.

3.2.3.3. Arquitectura física

La aplicación de una arquitectura de analítica moderna viene del principio de implementación de un enfoque distintivo de visión de negocio o riesgo para ser analizado. No es cuestión de simplemente implementar una Data Lake a un almacenamiento masivo de datos, sino de manejar un mecanismo de integración de una Data Lake, un almacenamiento de datos y los correspondientes almacenes creados por los científicos de datos con un fin particular, para conseguir como objetivo una gobernabilidad de la información y una fácil transferencia de datos.

El crecimiento de la información es exponencial, existen organizaciones que su flujo de trabajo genera desde terabytes hasta exabyte en algunos casos. La utilización de arquitecturas tradicionales para esta gran cantidad de información no soporta la inmensa capacidad del manejo de información, primeramente, por el costo que esto genera, y segundo por no ser lo suficientemente escalables para mantener un ritmo sostenible.

Un ejemplo de recolección de información puede ser el registro periódico de cuentas de ahorros por clientes desde varios orígenes, se puede apreciar en la Figura 3.6, como estos datos se centran en una Data Lake para ser tratados basándonos en la necesidad del negocio o en toma de decisiones de las distintas áreas. Esta información no es necesariamente data tratada, se requiere aplicaciones de procesos para categorizar el respectivo uso de esta.

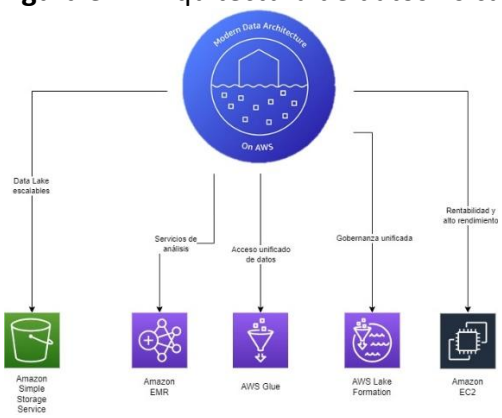
Figura 3.6. Fuentes de Data Lake

Fuente: (Services, 2023)

Las organizaciones agrupan la información en los diferentes silos, para posteriormente almacenarlos en un solo lugar y realizar actividades de análisis y/o machine learning. Por eso es importante el uso de una arquitectura moderna para la facilitación de los datos entre las distintas Data Lakes.

A continuación, en la Figura 3.7, se observa las herramientas que se requiere dentro de una arquitectura de analítica de datos.

Figura 3.7. Arquitectura de datos física

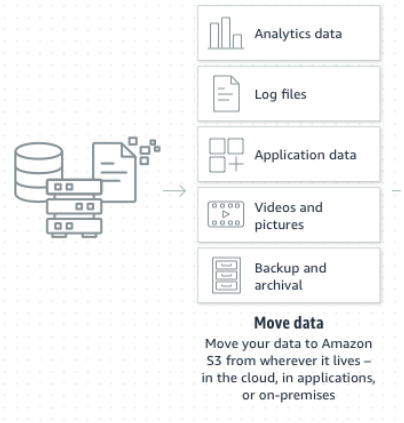


Fuente: Propia

Data Lake escalables

Por lo general, la administración y configuración de lagos de datos (Data Lake) implica un gran esfuerzo y una gran demanda de tiempo, *Amazon S3* es la mejor opción para almacenar estos datos, por su sostenibilidad de 11 nueves, y una disponibilidad de 99.99%; tiene una gran capacidad de seguridad, auditoria, escalabilidad y flexibilidad con sus cinco niveles de almacenamiento como se aprecia en la Figura 3.8, sin mencionar el menor costo entre los aprovisionamientos cloud líderes del mercado ya que va desde menos de un dólar por TB al mes.

Figura 3.8. Niveles de almacenamiento Amazon S3



Fuente: (Services, 2023)

Todos los tipos de cliente puede guardar su información para cualquier caso de uso, cómo: Data Lakes, aplicaciones móviles y aplicaciones nativas. Gracias a las distintas clases de almacenamiento se puede optimizar costos, configurar los niveles de acceso y organizar los datos cumpliendo con las mayores expectativas de los clientes empresariales.

Servicios de análisis para fines particulares

Aws proporciona una gran cantidad de herramientas diseñadas para el manejo de servicios de analíticos específicos, personalizados y optimizados enfocados a diferentes usos y aplicaciones.

Amazon EMR, es la herramienta de procesamiento macro de datos líder en el mercado Cloud, por su técnica de manejo de datos de grandes escalas de manera

interactiva y con aprendizaje automático, ejecutando un marco de código abierto de Presto, Apache Spark y Apache Hive.

Los casos de uso más utilizados para Amazon EMR son:

- Ejecución de procesamientos de grandes cantidades de información y análisis hipotéticos.
- La extracción de una variable de datos de varios orígenes con el fin de procesarla y ponerle a disposición de usuarios y aplicaciones (canalización).
- Análisis de eventos de tipo streaming en tiempo real, con el propósito de establecer una centralización de información del mismo tipo, pero de larga duración.
- Aceleración de la adopción de técnicas de machine learning con ingeniería de datos.

Acceso unificado de datos

A medida que la información y los Data Lake crecen, es necesario reorganizarlos con mayor frecuencia, teniendo que trasladarlos de un almacén a otro. Aws nos permite simplificar este traslado y la replicación entre almacenes de datos por todo los Data Lakes, con *Aws Glue* tenemos una gran capacidad de integración de datos, facilitándonos el análisis y la exploración entre las distintas fuentes de información.

Y en esta parte resaltamos la cuestión ¿Por qué utilizar Aws Glue?, debido a que la primera parte de un proyecto de análisis de datos es necesaria la preparación de la información para la obtención de resultados concisos y de calidad, Aws Glue es un servicio de integración de información que facilita la conexión con más de 70 orígenes de datos, siendo administrados por un catálogo centralizado de datos, además mantiene los controles sobre los ETL para crear, validar, ejecutar y canalizar la información. Es mayormente utilizado en los siguientes escenarios:

- Eliminación de la administración de una infraestructura BI, debido al aprovisionamiento automático y el consolidado de las integraciones de datos.

- Análisis oportuno y temprano de variables dentro de grandes cantidades de información.
- Exploración y análisis de datos de manera intuitiva, gracias al entorno de desarrollo integrado que tiene la herramienta, o cualquier procesador de texto de nuestra elección.
- Multiprocesamiento de los marcos de información, como lotes, micro lotes y streaming.

Gobernanza unificada

Entre las características más importantes para la gestión de la arquitectura de analítica de datos es el manejo de autorizaciones de clientes, y la administración y auditoria de los datos.

Ahora, pensando en esta aplicación de gobernanza con respecto a toda la organización, en donde existen varios almacenes complejos de datos, y múltiples fuentes de orígenes, esto puede ser complicado. *AWS Lake Formation* nos ayuda a realizar esta administración de todas las políticas de seguridad, gobernanza y auditoria con un control uniforme para todos los datos compartidos dentro de la empresa o externos.

Los casos de uso de implementación de AWS Lake Formation son fundamentados en las siguientes características:

- Implementar Data Lakes rápidamente desde fuentes de datos, descubrirlos y manejarlos desde un catálogo de datos centralizados.
- Escalamiento de permisos con mayor facilidad, incluyendo permisos más descriptivos entre celdas, filas, etiquetas, etc.
- Implementaciones de mallas de datos, para simplificar el reparto de datos dentro de la organización.

Alto rendimiento y rentabilidad

Es importante resaltar el principal compromiso de AWS con respecto a la mejora del rendimiento con el menor costo posible, innovando constantemente los servicios y sus características enfocadas a los clientes.

Amazon Elastic Compute Cloud (EC2) ofrece un acceso a la infraestructura con una alta confiabilidad y una escalabilidad bajo demanda en cuestión de minutos, manteniendo un 99.99% de disponibilidad de SLA. Amazon EC2 también nos ofrece una gran posibilidad de elegir recursos, pudiendo ser procesadores acordes al negocio o utilización, manejo de la red y sistema operativo.

El escenario de implementación para efecto de este caso de uso es una entrega de mayor oferta de los servicios desplegados dentro de un entorno Cloud, siendo potenciado con una red de hasta 400 Gbps y los almacenamientos creados como medida de optimización de costos.

4. Resultados y discusión

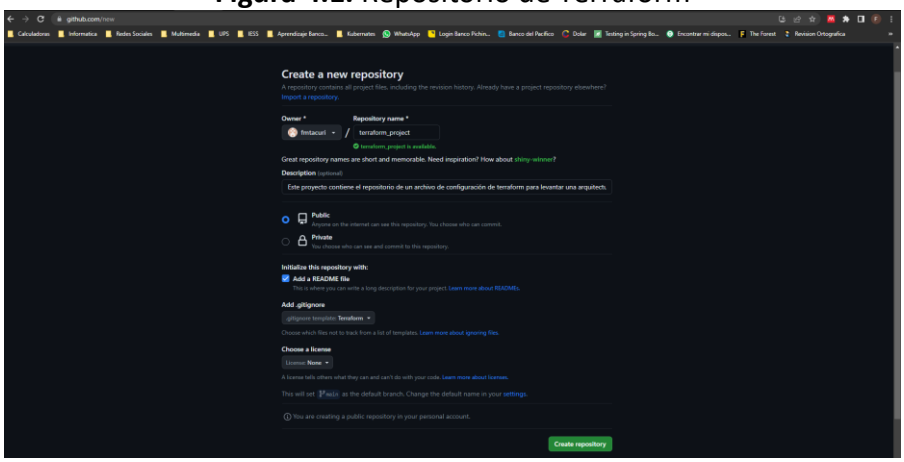
4.1. Implementación

Luego de realizar la investigación y el planteamiento de las dos diferentes estrategias de arquitecturas (transaccional y la arquitectura de analítica de datos), podemos tomar de ejemplo cualquiera de las dos arquitecturas para interpretar los resultados de la realización de alguna de ellas, demostrando la facilidad del aprovisionamiento y levantamiento de los distintos recursos de una infraestructura Cloud.

Para la ejecución de la prueba de concepto se seleccionó la arquitectura altamente transaccional, con el fin de mostrar los resultados sobre la creación de los recursos y componentes identificados dentro de la arquitectura física (Figura 3.3).

Se creó un repositorio dedicado a la codificación declarativa del lenguaje de configuración HashiCorp Configuration Language (HCL) que permite crear, cambiar y eliminar cualquier versión relacionada hacia el diseño de la infraestructura, como se muestra en la Figura 4.1. Este repositorio se encuentra declarado de manera pública para cualquier validación y ejemplo de implementación: https://github.com/fmtacuri/terraform-component-project, con nombre de rama "main".

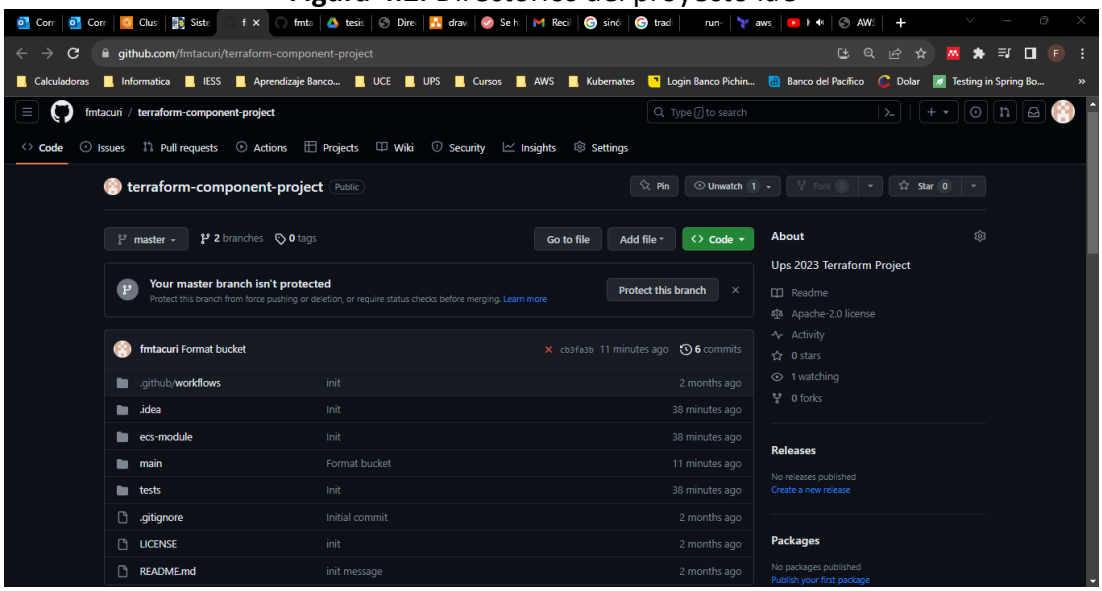
Figura 4.1. Repositorio de Terraform



Fuente: Propia

En la Figura 4.2, se puede apreciar los 2 directorios del repositorio, el primero contiene toda la estructura de la infraestructura detallada como un código fuente sobre los distintos recursos que se necesitan implementar **(ecs-modules),** como por ejemplo el levantamiento del Clúster de ECS de las aplicaciones. La extensión de este archivo es .tf, para la codificación de la infraestructura.

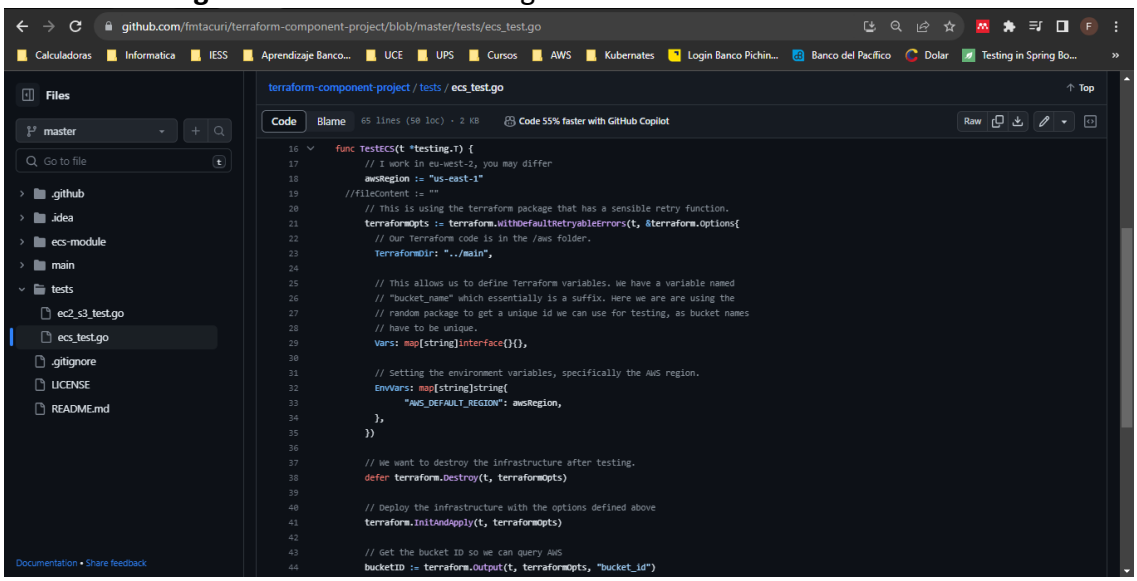
Figura 4.2. Directorios del proyecto IaC



Fuente: Propia

A nivel de desarrollo, las pruebas unitarias y pruebas de integración son lo que nos asegura que la codificación tiene la mayor calidad y confiabilidad posible. En la Figura 4.3, se observa algunas de las afirmaciones (assert's) creadas para comprobar la implementación E2E (pruebas de extremo a extremo) de la infraestructura creados con el script anterior, este archivo declarado con una extensión .go, se ejecuta con Terratest permitiendo validar los componentes y recursos de la infraestructura que se encuentran desplegados con las características de la parametrización.

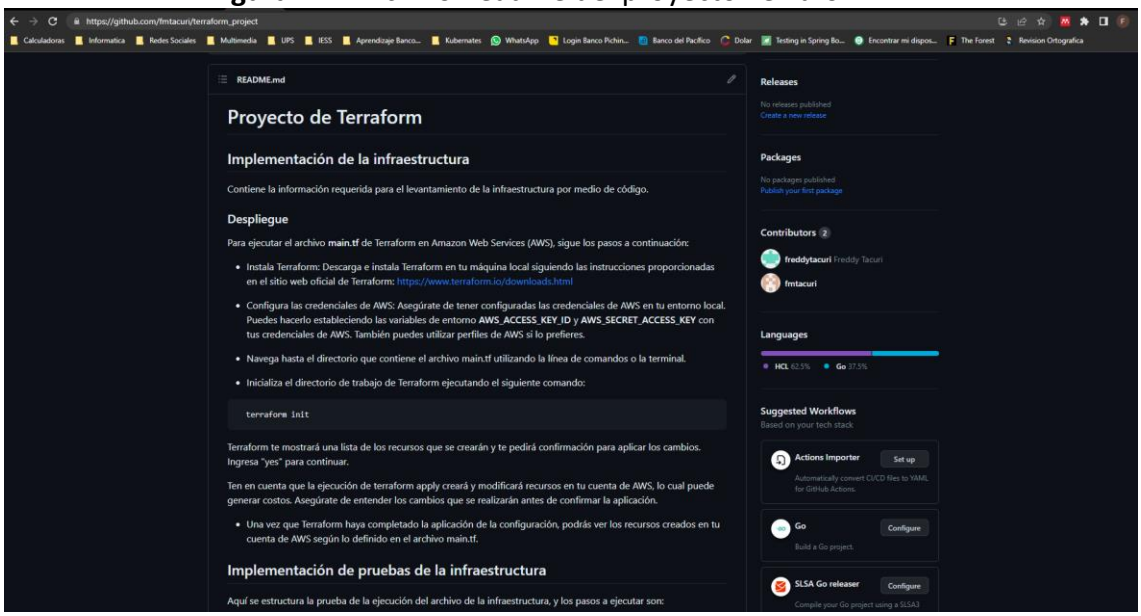
Figura 4.3. Pruebas de integridad sobre la infraestructura



Fuente: Propia

Cómo se observa dentro de la Figura 4.4, este repositorio cuenta con una breve descripción del levantamiento de los módulos de la infraestructura y los test en formato Markdown creado con el nombre de README.

Figura 4.4. Archivo Readme del proyecto Terraform



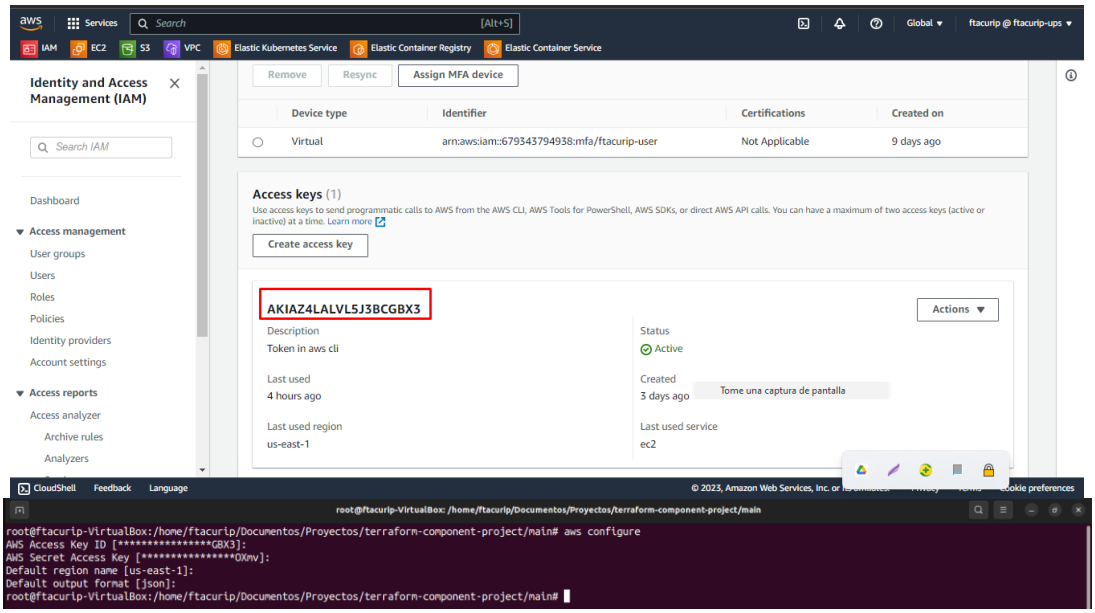
Fuente: Propia

4.2. Resultados

En esta sección, se detalla la ejecución de los escenarios de prueba diseñados para evaluar la eficacia de la estrategia seleccionada en una arquitectura escalable con aprovisionamiento de infraestructura automática (Infrastructure as Code - IaC). Estos casos de prueba se han diseñado para analizar diversos aspectos de la implementación y escalabilidad de la solución.

En primer lugar, se configuran las credenciales generadas dentro de AWS utilizando el comando *aws configure* para el usuario de prueba, como se muestra en la Figura 4.5. Es importante destacar que este usuario se creó previamente con el propósito de no utilizar la cuenta de usuario raíz (root) durante la ejecución de la infraestructura, siguiendo así una buena práctica de seguridad.

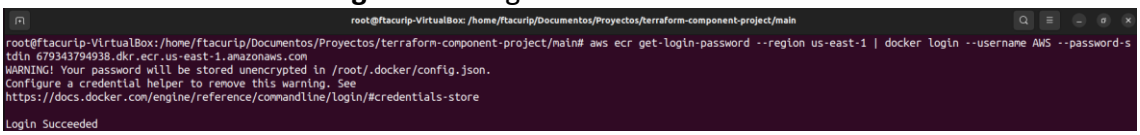
Figura 4.5. Consola de AWS Cli



Fuente: Propia

Como en nuestro script disponemos una aplicación web que puede versionar en conjunto con la infraestructura, realizamos el login por medio de Docker para determinar el Push hacia el repositorio centralizado de imágenes de AWS (ECR – Elastic Container Registry) como se muestra en la Figura 4.6.

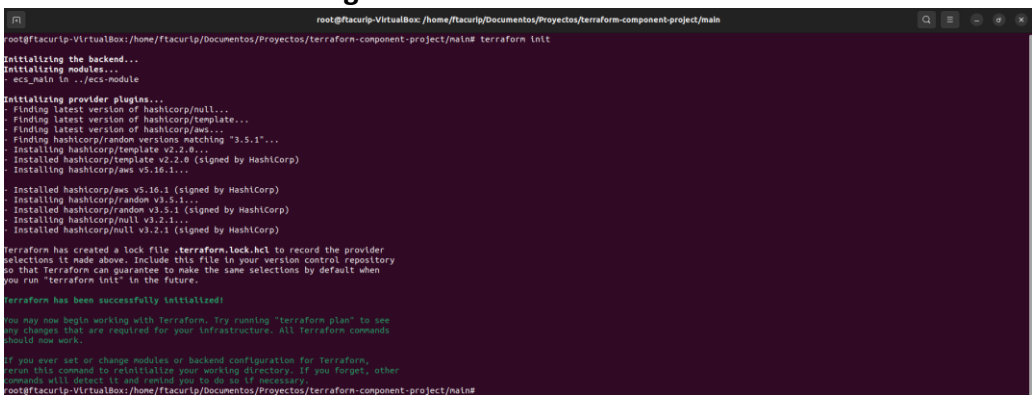
Figura 4.6. Login en Amazon ECR



Fuente: Propia

Una vez colocadas las credenciales y configuraciones necesarias procedemos a iniciar el proceso de Terraform (Figura 4.7) con el comando *"terraform init"*, con el objetivo de descargar/actualizar los providers requeridos para la correcta ejecución de la infraestructura.

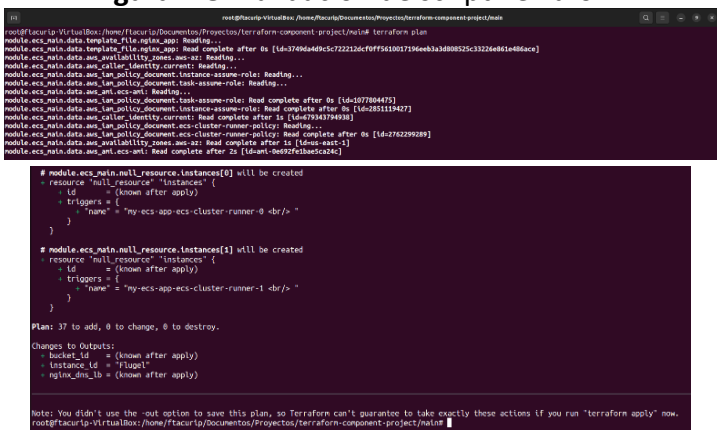
Figura 4.7. Inicio de Terraform



Fuente: Propia

Luego, se realiza una validación adicional mediante el uso de "*terraform plan*", que genera una vista previa de los recursos definidos en el script. Esta vista previa permite evaluar la viabilidad de la ejecución antes de llevar a cabo la acción apply en la infraestructura en la nube. Esta secuencia de pasos se muestra en detalle en la Figura 4.8.

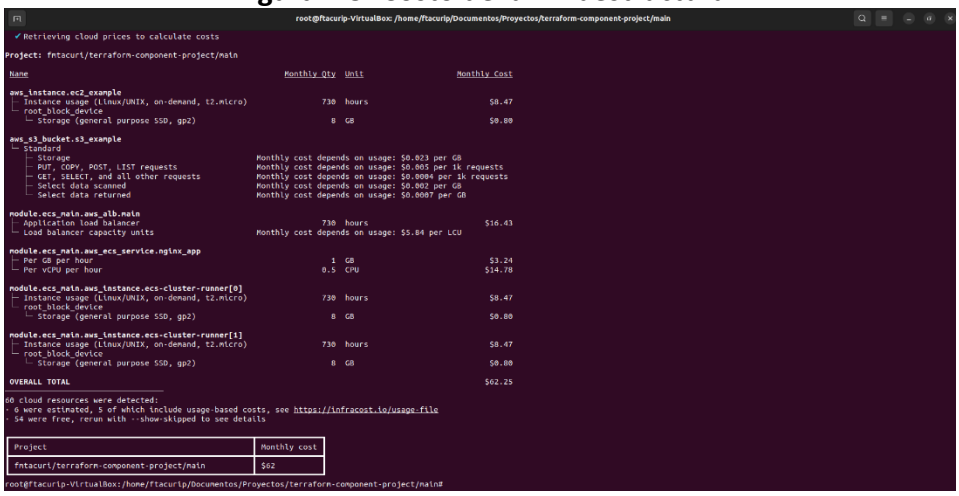
Figura 4.8. Validación de script Terraform



Fuente: Propia

Se disponen de herramientas esenciales para estimar el presupuesto inicial de la implementación de una plataforma basada en Terraform para la Infraestructura como Código (IaC). Un ejemplo notable de esta herramienta es Infracost, que, mediante la ejecución del comando "infracost breakdown --path .", proporciona una estimación mensual de los costos asociados con la implementación de nuestra infraestructura. Este proceso se ilustra en la Figura 4.9. En el escenario particular que estamos considerando, el costo proyectado se estima en $62.

Figura 4.9. Costo de la infraestructura

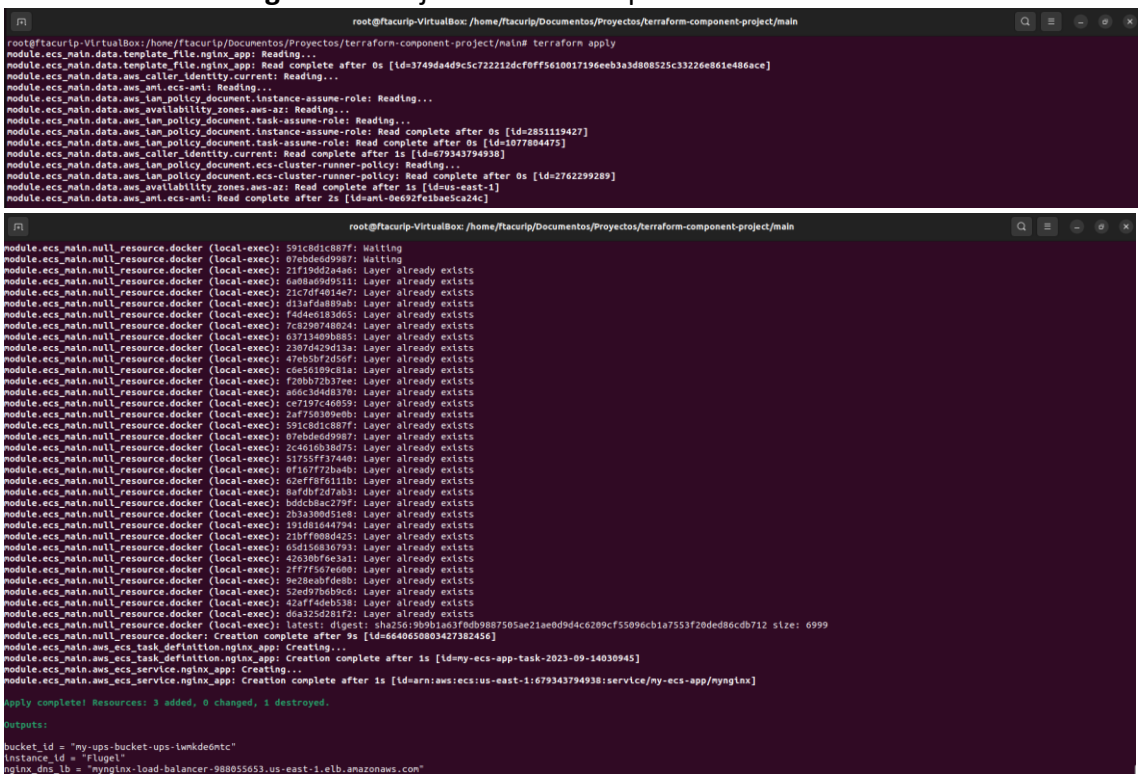


Fuente: Propia

Identificamos la creación de 37 recursos, que incluyen subredes, componentes de ECS (Amazon Elastic Container Service), instancias EC2 (Amazon Elastic Compute Cloud), almacenamiento en S3 (Amazon Simple Storage Service) y el balanceador de carga ELB (Elastic Load Balancer) destinados a nuestra aplicación de prueba.

A continuación, avanzamos en el proceso ejecutando el comando *"terraform apply"*. Este paso es crucial para poner en marcha nuestra infraestructura definida por código. Una vez que Terraform ha generado y presentado el plan de ejecución implícito en este comando, conforme a lo que se ilustra en la Figura 4.10, procedemos a confirmar la ejecución. Esto se lleva a cabo respondiendo con un *"Yes"* en el consentimiento de ejecución, lo que da luz verde al despliegue de los recursos y componentes previamente definidos en nuestro archivo de configuración de Terraform.

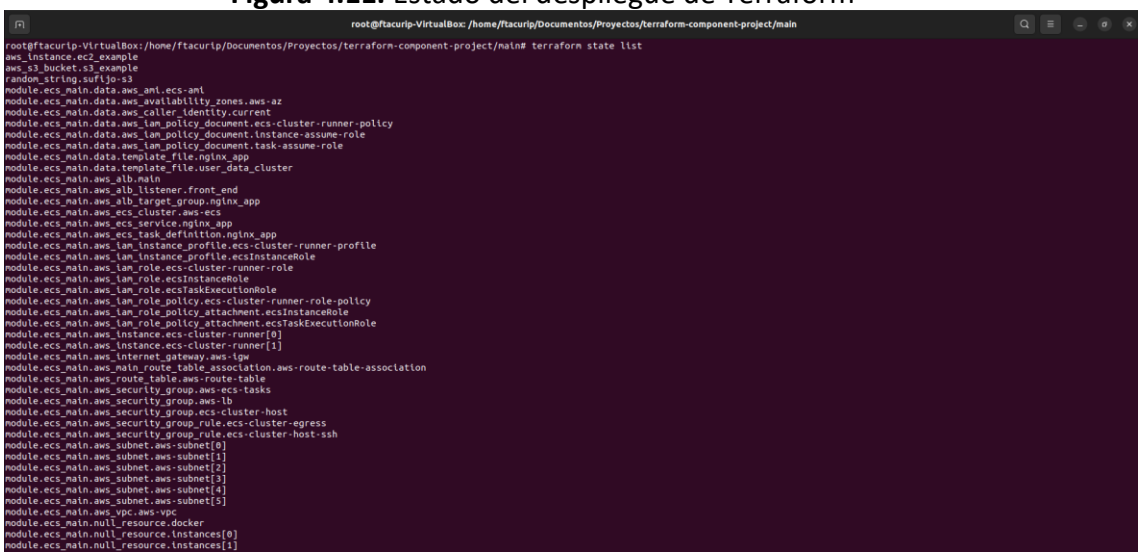
Figura 4.10. Ejecución del script IaC Terraform



Fuente: Propia

Validamos nuestro *state* que es el registro de Terraform para ver los recursos desplegados en la ejecución con el comando *"terraform state list"*, como nos indica la Figura 4.11.

Figura 4.11. Estado del despliegue de Terraform



Fuente: Propia

Una vez que hemos parametrizado y validado la infraestructura como código, avanzamos hacia la realización de los escenarios de prueba planificados. Es fundamental recordar que la dirección de nuestro balanceador de carga, crucial en este proceso, ya que se encuentra en la ubicación: *"http://mynginx-load-balancer-988055653.us-east-1.elb.amazonaws.com".* Por defecto desplegamos el balanceador con una instancia como se observa en la Figura 4.12.

Figura 4.12. Aplicación con las instancias levantadas

Fuente: Propia

4.2.1. Escenario de prueba 1: Tiempo de aprovisionamiento

1. **Propósito:** Evaluar el tiempo necesario para aprovisionar una nueva instancia de infraestructura utilizando IAC.
2. **Procedimiento:**

    - Iniciar el proceso de aprovisionamiento de una nueva instancia de infraestructura empleando IAC.
    - Registrar el tiempo transcurrido desde el inicio del proceso hasta que la instancia de infraestructura esté completamente aprovisionada y lista para su uso.
    - Repetir el proceso de aprovisionamiento varias veces para obtener un promedio de tiempo de aprovisionamiento.
    - Registrar cualquier problema o error que ocurra durante el proceso de aprovisionamiento.

3. **Resultados:**

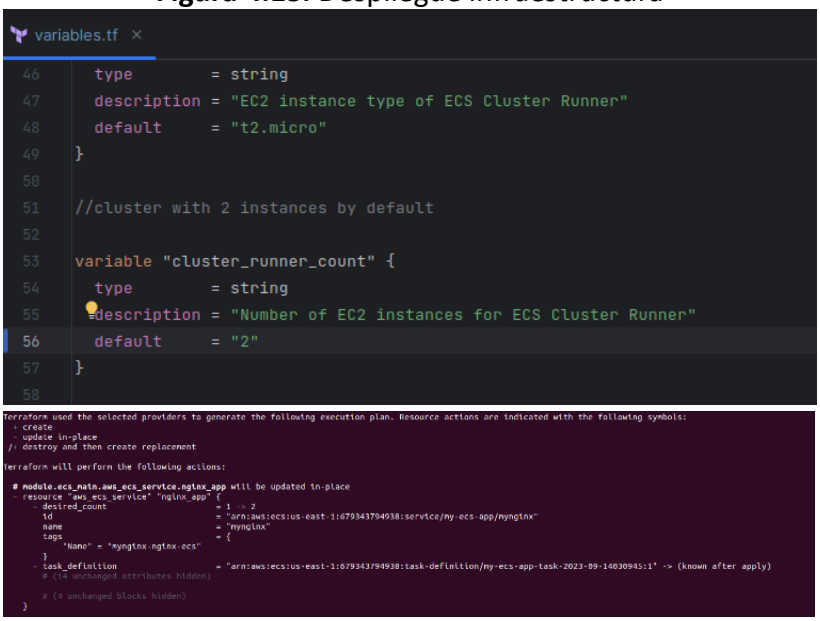
El tiempo promedio de aprovisionamiento de una nueva instancia utilizando IAC fue de 2 minutos y 30 segundos. Se observó una mejora representativa comparada con los métodos tradicionales de aprovisionamiento manual, que requerían aproximadamente 30 minutos.

El tiempo necesario para aprovisionar una nueva instancia de infraestructura usando IAC fue satisfactorio y se mantuvo dentro de los límites esperados. La solución de IAC demostró ser eficiente en la creación y configuración de nuevas instancias de infraestructura, lo que permite una rápida adaptación y escalabilidad del entorno. La capacidad de aprovisionamiento rápido y confiable proporcionada por IAC agiliza el despliegue de nuevas instancias de infraestructura.

En la etapa práctica, realizamos la implementación de la infraestructura mediante la ejecución del comando *"terraform apply"*. Con este comando, logramos desplegar el recurso encargado de gestionar las instancias de ECS, que se ajusta

según la variable de entorno que especifica la cantidad de instancias, tal como se muestra en detalle en la Figura 4.13.

Figura 4.13. Despliegue infraestructura



Fuente: Propia

4.2.2. Escenario de prueba 2: Tolerancia a fallos

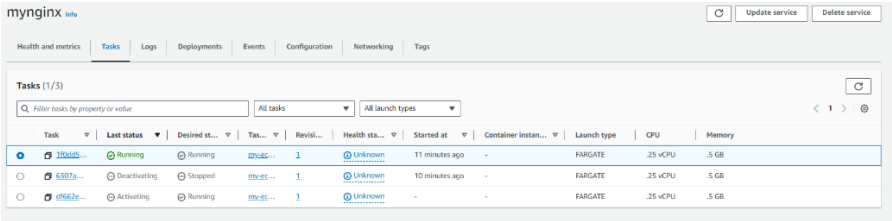
1. **Propósito:** Evaluar la capacidad de la solución para manejar fallos en los componentes de la infraestructura.
2. **Procedimiento:**

    - Configurar la solución con una infraestructura que incluya diferentes componentes, como servidores, bases de datos y/o servicios, en este caso práctico utilizamos los servicios de las instancias desplegadas.
    - Simular un fallo en uno de los componentes de la infraestructura, como un recurso de servicio.
    - Observar y registrar el comportamiento de la solución cuando ocurre el fallo.
    - Verificar si la solución es capaz de detectar automáticamente el fallo y tomar medidas para manejarlo, como redireccionar el tráfico a componentes alternativos o iniciar la recuperación del componente fallido.
    - Evaluar la estabilidad y continuidad del servicio proporcionado por la solución durante y después del fallo.

3. **Resultados:**

Durante la simulación del fallo en un componente de la infraestructura como se puede observar en la Figura 4.14, la solución demostró una capacidad efectiva para manejar el fallo y mantener la disponibilidad del servicio, además, se detectó automáticamente el fallo en el componente y tomó medidas para minimizar el impacto en el servicio levantando otra replica para ajustarse a la configuración anterior.

Figura 4.14. Simulación de falla de un componente



Fuente: Propia

Se observó una transparencia en la transición del tráfico hacia componentes alternativos sin interrupción significativa del servicio, poniendo en marcha los procesos de recuperación del componente fallido de manera automática y eficiente al tener varias instancias de la aplicación como se aprecia en la Figura 4.15.

Figura 4.15. Instancias activas de la aplicación

Fuente: Propia

Después de la recuperación, la solución volvió a un estado estable y continuó operando de manera normal, brindando el servicio esperado a los usuarios.

La detección automática de fallos y las acciones de recuperación implementadas garantizaron la disponibilidad y continuidad del servicio. La solución fue capaz de redirigir el tráfico a componentes alternativos sin interrupciones significativas y realizar la recuperación de los componentes fallidos de manera automática y eficiente. Estos resultados refuerzan la confiabilidad y robustez de la solución, así como su capacidad para mantener la estabilidad del sistema, incluso en situaciones de fallos inesperados. La tolerancia a fallos de la solución proporciona a los usuarios una experiencia de servicio fluida y minimiza los impactos negativos en la disponibilidad y el rendimiento de los recursos.

4.2.3. Escenario de prueba 3: Escalabilidad horizontal

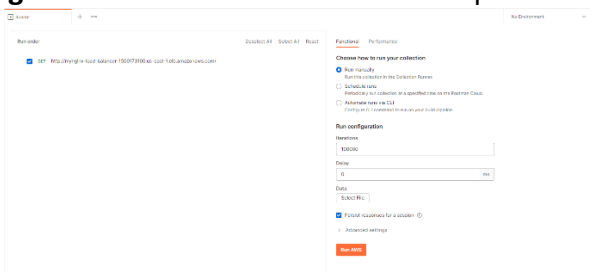
1. **Propósito**: Evaluar la capacidad de la solución para escalar horizontalmente al aumentar la carga de trabajo.
2. **Procedimiento**:

    - Configurar la solución con una infraestructura inicial que incluya un número determinado de instancias de aplicación.
    - Generar una carga de trabajo que simule un aumento significativo en la demanda del sistema.
    - Medir y registrar el rendimiento de la solución, incluyendo la capacidad del procesamiento, los tiempos de respuestas y la utilización de los recursos, con la carga de trabajo inicial.
    - Incrementar gradualmente la carga de trabajo, aumentando la cantidad de solicitudes concurrentes o la cantidad de datos procesados.
    - Verificar si la solución es capaz de escalar horizontalmente, es decir, si es capaz de agregar automáticamente nuevas instancias de aplicación para manejar la carga de trabajo adicional.
    - Evaluar la estabilidad y continuidad del servicio proporcionado por la solución durante el aumento de carga de trabajo.

3. **Resultados**:

Con la carga de trabajo inicial, la solución mostró un rendimiento satisfactorio, con tiempos de respuesta aceptables y una utilización de recursos eficiente. A medida que se incrementó la carga de trabajo realizando una petición automatizada desde postman con 100000 peticiones a la aplicación como se muestra en la Figura 4.16, la solución demostró la capacidad de escalar horizontalmente, agregando nuevas instancias de aplicación según fuera necesario.

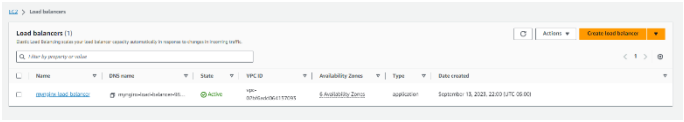
Figura 4.16. Prueba automatizada de peticiones



Fuente: Propia

La escalabilidad horizontal permitió a la solución adaptarse de manera dinámica a la carga de trabajo adicional, manteniendo un rendimiento óptimo y evitando la degradación del servicio. La solución logró distribuir eficientemente por medio de un balanceador implementado en la Figura 4.17, mostrando la carga entre las instancias, evitando cuellos de botella y maximizando la capacidad de procesamiento.

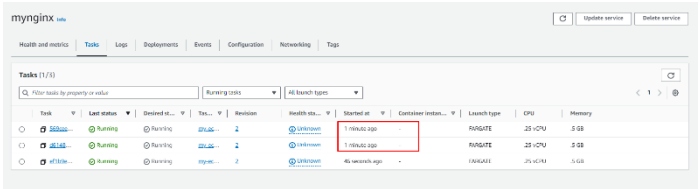
Figura 4.17. Balanceador de la aplicación



Fuente: Propia

La capacidad de agregar automáticamente nuevas instancias de aplicación permitió a la solución adaptarse dinámicamente a las demandas cambiantes y mantener un rendimiento óptimo. La escalabilidad horizontal permitió una distribución homogénea de la carga ejecutada de trabajo como se observa dentro de la Figura 4.18, y evitó la saturación de los recursos, asegurando tiempos de respuesta aceptables y una alta capacidad de procesamiento. La escalabilidad horizontal proporciona una solución robusta y confiable que puede crecer según sea necesario para mantener un servicio estable y de calidad en presencia de aumentos en la carga de trabajo.

Figura 4.18. Instancias levantadas acorde a la demanda



Fuente: Propia

4.2.4. Escenario de prueba 4: Escalabilidad vertical

1. Propósito: Evaluar la capacidad de la solución para escalar verticalmente al aumentar los recursos de una aplicación.

2. Procedimiento:

- Configurar la solución con una instancia de aplicación en un único clúster.
- Incrementar los recursos del servidor (por ejemplo, aumentar la cantidad de CPU o memoria).
- Medir el rendimiento de la solución y la capacidad para manejar una carga de trabajo mayor.

3. Resultados:

La solución demostró una destacada capacidad para escalar verticalmente al incrementar los recursos iniciales del servidor de aplicación, que se puede apreciar en la Figura 4.19, detallado en el script de IaC.

Figura 4.19. Recursos iniciales de la aplicación

```
# module.ecs_main.aws_ecs_task_definition.nginx_app will be created
+ resource "aws_ecs_task_definition" "nginx_app" {
    + arn                    = (known after apply)
    + arn_without_revision   = (known after apply)
    + container_definitions  = jsonencode(
        [
          + {
              + cpu          = 256
              + image        = "679343794938.dkr.ecr.us-east-1.amazonaws.com/tesis-ups:latest"
              + memory       = 512
              + name         = "mynginx"
              + portMappings = [
                  + {
                      + containerPort = 80
                      + hostPort      = 80
                    },
                ]
            },
        ]
    )
```

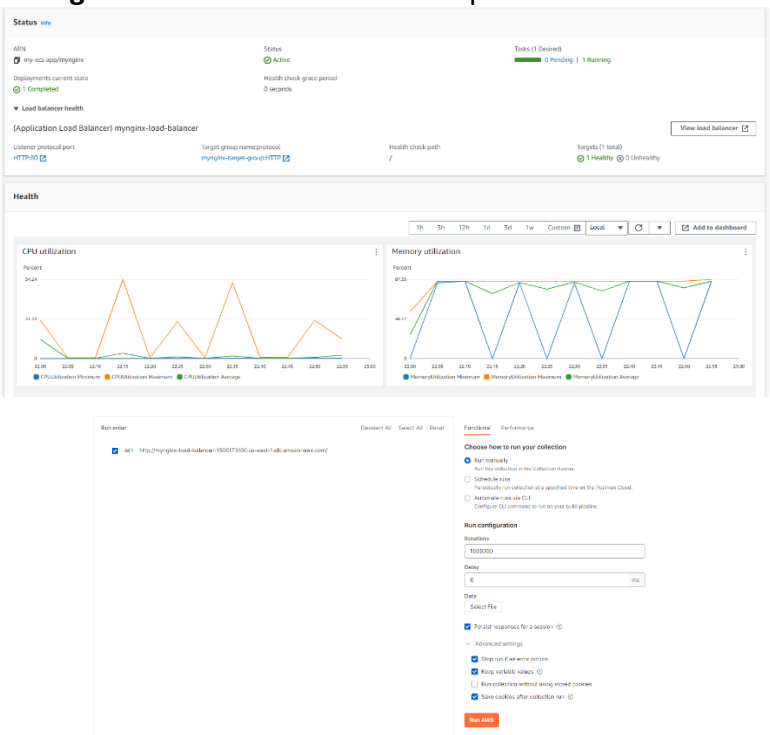
Fuente: Propia

El sistema logró utilizar eficientemente los recursos adicionales asignados, lo que resultó en un rendimiento más rápido y una mayor capacidad para atender la demanda del usuario.

La escalabilidad vertical resultó exitosa, lo que nos permitió ajustar eficazmente la infraestructura original cuando el nivel de procesamiento superó el 80% de utilización de memoria, como se muestra en la Figura 4.20. Esto se tradujo en una

adaptación eficiente de acuerdo con las cambiantes necesidades del sistema y las demandas del entorno de ejecución al solicitar un millón de peticiones.

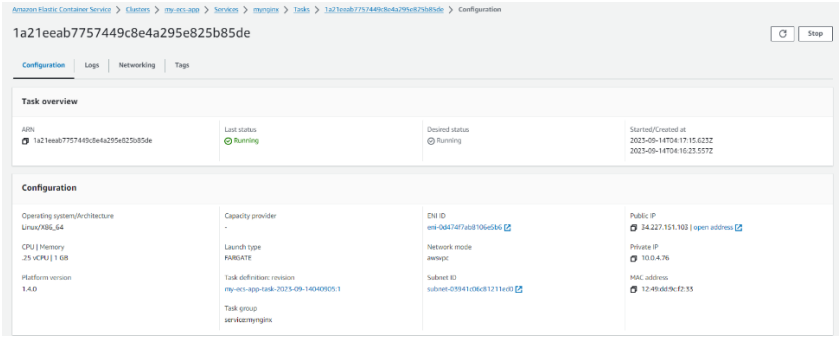
Figura 4.20. Rendimiento de la aplicación con estrés



Fuente: Propia

La capacidad de ajustarse de manera ágil a un incremento en la carga de trabajo al asignar recursos adicionales de memoria ha demostrado su valía en entornos dinámicos, elevando la capacidad a 1GB según lo ilustra la Figura 4.21. Esta aptitud de escalabilidad vertical proporciona una solución sumamente flexible, capaz de amoldarse a las cambiantes demandas del sistema, lo que resulta en un mejor rendimiento y una experiencia de usuario mejorada.

Figura 4.21. Aumento de memoria de aplicación a 1GB



Fuente: Propia

4.2.5. Escenario de prueba 5: Reversibilidad

1. Propósito: Evaluar la capacidad de revertir los cambios en la infraestructura realizados a través de IAC.

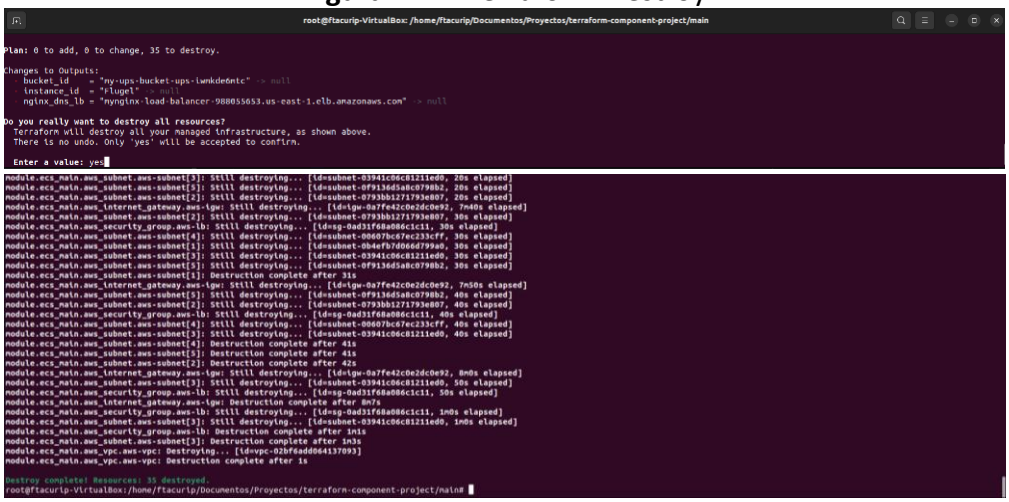
2. Procedimiento:

- Ejecutar cambios en la infraestructura utilizando IAC, como agregar o modificar componentes.
- Revertir los cambios a un estado anterior usando IAC.
- Verificar si la infraestructura vuelve al estado previo de manera correcta.

3. Resultados:

Las modificaciones realizadas en la infraestructura mediante la Infraestructura como Código (IAC) se llevaron a cabo de manera exitosa, garantizando que la infraestructura se configurara conforme a las especificaciones establecidas en el script. Durante la fase de prueba de reversión, logramos deshacer las modificaciones en la infraestructura de manera efectiva mediante el uso de IAC y el comando "terraform destroy", como se muestra en la Figura 4.22. Cabe destacar que esta ejecución resultará en la eliminación de los 35 recursos que inicialmente se implementaron.

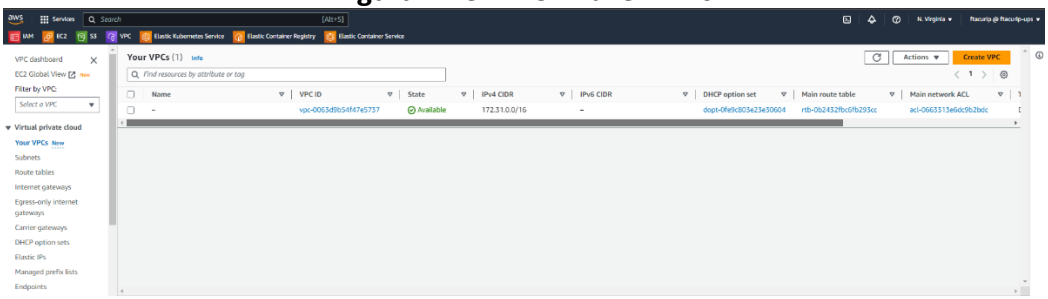
Figura 4.22. Terraform Destroy



Fuente: Propia

La infraestructura se restauró con precisión a su estado anterior, como se puede constatar en el panel de VPC de la consola de AWS utilizado en este ejemplo (Figura 4.23). En dicho panel, queda patente que no existe ninguna configuración a menos que se haya desplegado previamente una VPC. Durante el proceso de reversión, se aseguró que no quedaran rastros de las modificaciones previamente realizadas. Además, se verificó que no se produjeran errores ni conflictos, garantizando así un funcionamiento óptimo de la infraestructura.

Figura 4.23. VPC final en Aws



Fuente: Propia

La reversibilidad de las modificaciones en la infraestructura a través de la Infraestructura como Código (IAC) se ha revelado como una característica sólida y fiable. La capacidad de revertir los cambios de manera precisa y sin complicaciones brinda un mayor nivel de control y seguridad en la administración de la infraestructura. Con el uso de IAC, se puede confiar en la efectividad de la reversión de cambios, lo que proporciona flexibilidad y agilidad en la adaptación y el mantenimiento de la infraestructura. Esta funcionalidad adquiere un valor destacado en entornos dinámicos donde los requisitos y las configuraciones pueden cambiar con frecuencia.

4.2. Discusiones

- Tener un ambiente Cloud, claramente resalta los múltiples beneficios para el despliegue de arquitecturas, aquí se puede cuestionar lo siguiente ¿Cómo obtener el máximo beneficio de las nuevas características de los servicios que AWS proporciona?
- Dentro de las implementaciones de arquitecturas es importante resaltar el gran compromiso que tiene AWS hacia el objetivo final de los usuarios, que a través del "trabajo a la inversa" (Working Backwards) primero se determina el objetivo del negocio y con esa finalidad se procede a diseñar e implementar la mejor arquitectura posible, esto quiere decir evitar el uso de una tecnología por el mismo hecho de que exista, sino más bien enfocarse en crear una satisfacción integra, optimizada, moderada y sobre todo segura para las exigencias de los clientes.
- A medida que aparecen nuevas tecnologías basadas en la analítica de los datos, incorporando aplicaciones con IA, la visión, tolerancia y flexibilidad del manejo de los datos se vuelve una necesidad dentro de las organizaciones que al final ya no solo resulta ser útil, sino una necesidad dentro de la cultura empresarial. Aunque las arquitecturas orientas a análisis de datos parecen ser un desafío, pueden ayudarnos a desarrollar que nuestros sistemas coexistan en un ambiente basado en datos, permitiéndonos integrarnos a nuevas aplicaciones como por ejemplo la inteligencia de negocios.
- La ejecución de Terraform y Terratest dentro de una plataforma Cloud permite tener esa automatización de la infraestructura fácilmente repetible, reduciendo los errores y mejorando el tiempo de entrega e implementación. Este despliegue puede ser constante y reproducible, permitiendo tener una infraestructura muy consistente en cada ejecución.
- El control de versiones y seguimiento de cambios es posible cuando se versionan distintas modificaciones al código de IaC, aprovechando los sistemas de controles de versiones como Git. La validación continua de un flujo de trabajo se puede implementar con CI/CD para establecer un proceso

automatizado de constante validación de infraestructura sobre las actualizaciones y modificaciones, minimizando el impacto de los sistemas en producción.

5. Conclusiones

1. La implementación de estrategias de arquitectura de software utilizando operaciones mínimas y aprovisionamiento automático, como se logra con Terraform y Terratest, es fundamental para garantizar la escalabilidad y la tolerancia a fallos en las aplicaciones empresariales.
2. La investigación que involucra la evolución de la arquitectura de software de aplicaciones empresariales revela un cambio significativo en los últimos años, donde se ha pasado de arquitecturas monolíticas a arquitecturas orientadas a servicios. Estas últimas ofrecen mayor flexibilidad y capacidad de escalamiento, pero también presentan desafíos específicos.
3. La arquitectura de software tradicional (monolítica) presenta problemas como la falta de modularidad, la dificultad para escalar de manera eficiente y la falta de tolerancia a fallos. Por otro lado, la arquitectura orientada a servicios puede tener limitantes en términos de complejidad de implementación, coordinación de servicios y rendimiento.
4. La utilización de herramientas como Terraform y Terratest proporciona una solución práctica y eficiente para automatizar el aprovisionamiento de infraestructura en diferentes plataformas y entornos. Estas herramientas permiten describir la infraestructura como código, lo que simplifica la gestión y garantiza la reproducibilidad de los entornos.
5. La prueba de concepto realizada con Terraform y Terratest demostró la viabilidad de la automatización del aprovisionamiento de infraestructura en diferentes plataformas y ambientes. Esta automatización reduce la posibilidad de errores humanos, acelera los tiempos de implementación y brinda mayor confiabilidad a través de pruebas automatizadas.
6. La implementación de estrategias de arquitectura de software utilizando operaciones mínimas y aprovisionamiento automático con Terraform y Terratest contribuye a la escalabilidad y la tolerancia a fallos, al tiempo que simplifica la gestión de la infraestructura y acelera los procesos de implementación y entrega.

En general, estas conclusiones resaltan la importancia de adoptar enfoques modernos de arquitectura de software y utilizar herramientas como Terraform y Terratest para lograr una infraestructura automatizada, escalable y tolerante a fallos en aplicaciones empresariales. Estas soluciones permiten enfrentar los desafíos de la arquitectura orientada a servicios y brindan beneficios favorables en términos de eficacia, eficiencia y confiabilidad.

6. GLOSARIO

IAC: Proviene de las siglas en ingles Infraestructure As Code (Infraestructura como código), permite la gestión de la infraestructura por medio de código en lugar de realizarlo de forma manual.

Pipeline CI/CD: es un conjunto de buenas prácticas incorporados que permiten la automatización de manera continua y controla el ciclo de vida de los procesos.

CI/CD: de las siglas en inglés Continuous Integration y Continuous Delivery (Integración y entrega continua), conceptos clave dentro de la automatización de procesos.

Terraform: un programa para administrar IaC.

GCP: Google Cloud Plataform.

AWS: Amazon Web Service.

Azure ARM: Azure Resource Manager es el servicio de implementación y administración para Azure que a través de su capa de administración permite la gestión de recursos.

Stack tecnológico: es una lista de todas las tecnologías o servicios utilizados en la construcción y ejecución de una aplicación.

API: proviene de las siglas en español Interfaz de programación de aplicaciones que es el conjunto de funciones y procedimientos que pueden ser utilizados por otro software.

SOA: de las siglas en español Arquitectura Orientada a Servicios, es un tipo de patrón de diseño de software que permite la reutilización de sus elementos mediante la implementación de interfaces de servicios.

DAO: de las siglas en inglés Data Access Object, que es un componente de software que facilita el manejo de los datos almacenados.

SoC: proviene de las siglas en español separación de Preocupaciones, que se enfoca en la única responsabilidad de las capas de una arquitectura de software.

PaaS: proviene de las siglas en inglés de Plataform As A Service, como plataforma como un servicio, que provee una implementación y desarrollo completo en la nube.

BFF: de las siglas en inglés Backend For FrontEnd, es un patrón de diseño de software que permite estructuras a los clientes con necesidades específicas.

Referencias

Ahmad, A., & Pahl, C. (2018). *Customisable Transformation-Driven Evolution for Service Architectures.* doi:10.1109/CSMR.2018.56

Al-Debagy, O., & Martinek, P. (2018). *A Comparative Review of Microservices and Monolithic Architectures.* doi:10.1109/CINTI.2018.8928192

Amorim, S., & Almeida, E. (2019). *Scalability of Ecosystem Architectures.* doi:10.1109/WICSA.2019.36

Bahsoon, R., & Emmerich, W. (2018). *An Economics-Driven Approach for Valuing Scalability in Distributed Architectures.* doi:10.1109/WICSA.2018.45

Blancarte Iturralde, O. J. (2019). *Introducción a la arquitectura de software.*

Blancarte, O. (2019). *Introducción a los patrones de diseño.*

Bolotin, G. (2019). *Space-cube: a flexible computer architecture based on stacked modules.* California. doi:10.1109/MCMC.2019.510763

Davis, S. (2019). *Cultura corporativa y Estrategia, dos piezas que deben ir juntas.*

DORA DevOps Research & Assessment. (2018). *State of DevOps.*

EDucation team . (Noviembre de 2022). *Introducción a DevOps*. Obtenido de https://ed.team/

Gos, K., & Zabierowski, W. (2020). *The Comparison of Microservice and Monolithic Architecture.* doi:10.1109/MEMSTECH49584.2020.9109514

Kavis, M. (2014). *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS).* Wiley.

Keqin, L. (2020). *Average-case scalability analysis of parallel computations on k-ary d-cubes.* doi:10.1109/IPDPS.2020.1015520

Kuryazov, D., & Khujamuratov, B. (2020). *Towards Decomposing Monolithic Applications into Microservices.* doi:10.1109/AICT50176.2020.9368571

Le, D., & Gia, N. (2018). *Cloud Computing and Virtualization.* Wiley.

Martin, R. (2018). *Arquitectura limpia.* Anaya Multimedia.

Martínez Estrada, P. (2017). *MI PRIMER STARTUP.* San Luis Potosí, México, México: Laboratorio Emprendedor SAPI de CV.

Microsoft Azure. (s.f.). *Tutorial de DevOps*. Obtenido de https://azure.microsoft.com/es-es/solutions/devops/tutorial/#building

Nossa Ortiz, L. (2019). *Buenas prácticas en Arquitectura Orientada a Servicios (SOA).*

Novoa, A., & Maldonado, P. (2018). *SISTEMAS DE INFORMACIÓN GERENCIAL TIPO TRANSACCIONAL PARA PYMES.* Ágora: Universidad Militar Nueva Granada. Obtenido de https://ojs.tdea.edu.co/index.php/agora/article/view/242

Olivares, G., & Ramírez, C. (2019). *DevOps y seguridad cloud.* Editorial UOC.

Patni, J., & Tiwari, D. (2020). *Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM).* Shillong. doi:10.1109/ComPE49325.2020.9200030

Rahman, A. (2018). *Characteristics of Defective Infrastructure as Code Scripts in DevOps.*

Richardson, C. (2018). *Microservices patterns.*

Robbins, D. (2019). *Fundamentos de Administración.* México: Pearson.

Roncancio, G. (2022). *Pensemos*. Obtenido de https://gestion.pensemos.com/indicadores-de-gestion-tipos-y-ejemplos

Rong, C. (2022). *OpenIaC: open infrastructure as code - the network is my computer* (Vol. XI).

Services, A. W. (2023). *Amazon Web Services*. Obtenido de https://aws.amazon.com/es/big-data/datalakes-and-analytics/modern-data-architecture/?nc=sn&loc=2

Villamizar, M. (2018). *Scaling the Colombian Data Cube Using a Distributed Architecture.* doi:10.1109/IGARSS.2018.8517888

Vmware, L. (s.f.). *Vmware*. Obtenido de https://www.vmware.com/latam/topics/glossary/content/cloud-computing-infrastructure.html