

FACULTAD DE INGENIERÍA

TESIS DE MAESTRÍA

DRLAnalizer: Un análisis sistemático de rate limiting en la nube

ALUMNO:

LIC. JUSTO VICTOR ANDREY

TUTOR DE TESIS:

MG. MARTIN ACOSTA

Buenos Aires - Argentina 2023

PÁGINA DE APROBACIÓN DEL COMITÉ

_____

Mg. Martin Acosta
Tutor de tesis

_____

Tutor de tesis

_____

JURADO INTERNO 1
Jurado Interno

_____

JURADO INTERNO 2
Jurado Interno

_____

JURADO EXTERNO
Jurado Externo

Resumen

Las empresas vienen creciendo y adoptando arquitecturas SOA y microservicios las cuales se hacen más fáciles de gestionar.

Los equipos internos tienen necesidades de consultar servicios externos e internos, como datos que no pertenecen a la empresa y datos de otros equipos. Es así como integraciones externas interactúan con las empresas a partir de las llamadas REST.

Los servicios creados por equipos necesitan de otros datos que puedan estar presentes en situaciones (servicios) críticos.

¿Cómo se puede identificar qué tal servicio es crítico?

Entendiendo el impacto del negocio o la criticidad de los datos almacenados ayudan a catalogar y definir las responsabilidades.

¿Cómo gobernar y proteger los activos y servicios más importantes de nuestra compañía y evitar el mal uso de la misma?

Esta es una tarea compleja dado que exige tener capas de seguridad como autenticación y autorización. Pero no solo estas cuestiones bastan ya que los consumidores pueden estar utilizando los servicios identificados como críticos de forma errónea o incluso en horarios de los cuales tenemos que proveer funcionalidades básicas a los clientes del negocio.

El trabajo de tesis en exposición se pondrá en evidencia cómo evaluar el uso de sistemas con limitaciones de tasa para proteger servicios y activos críticos, utilizando

como guía un caso de estudio hipotético del cual vamos a encontrarnos con distintos escenarios, simulando comportamientos que puedan ocurrir en el día a día.

Tabla de Contenidos

Introducción 11
- Definición del problema 11
- Antecedentes 12
- Justificación 14
- Alcance y Limitaciones 14
- Objetivos 15
  - Principal 15
  - Específicos 15

Fundamentos Teóricos 16
- Introducción del Capítulo 16
- Microservicios 17
- API Gateway 19
- Algoritmos de Limitación de Tasa 20
- Autenticación y Autorización 23
- Cloud Computing 24
- Definición de Tecnologías Base 26
  - API GATEWAY AWS 27
  - GOOGLE API GATEWAY 28
  - AZURE API MANAGEMENT 29
- API RESTful 30
- Denegación de Servicio 31
- Conclusión del Capítulo 32

Modelo 33
- Definición 33
- Motivación 35
- Conclusión del Capítulo 35

Caso de Estudio 36
- Introducción del Capítulo 36
- Metodología 36
- Escenario 37
  - Vista de componentes del Escenario 38
  - Arquitectura del Escenario 40
  - Arquitectura con componentes en la Nube 43
  - Infraestructura con componentes AWS 44
  - Infraestructura con componentes GCP 45
  - Infraestructura con componentes AZURE 46

Método de Comparación 46
Conclusión del Capítulo 49
Resultados 50
Experimentos 50
Comparación de Resultados 68
Conclusiones 71
Discusión General del Proyecto 73
Discusión 73
Conclusión Final 73
Trabajo Futuro 74
BIBLIOGRAFÍA 76
APÉNDICES 82
Script para infraestructura en AWS 82
Script para infraestructura en GCP 121
Script para infraestructura en AZURE 126
Script Escenario Marketing 132
Script Escenario Procesamiento de Pagos 134
Script Escenario Análisis de Riesgo 136
CURRICULUM VITAE 138

LISTA DE TABLAS

Tabla 1 - Comparación entre autenticación y autorización

Tabla 2 - Comparación entre IaaS, PaaS y SaaS

Tabla 3 - Evaluación de cada atributo arquitectónico.

Tabla 4 - Descripción de funcionalidades.

Tabla 5 - Peticiones de pruebas en AWS para el escenario de campañas de marketing.

Tabla 6 - Peticiones de pruebas en AWS para el escenario de procesamiento de pagos.

Tabla 7 - Peticiones de pruebas en AWS para el escenario de análisis de riesgo para ofrecer créditos.

Tabla 8 - Peticiones de pruebas en GCP para el escenario de campañas de marketing.

Tabla 9 - Peticiones de pruebas en GCP para el escenario de procesamiento de pagos.

Tabla 10 - Peticiones de pruebas en GCP para el escenario de análisis de riesgo para ofrecer créditos.

Tabla 11 - Peticiones de pruebas en AZURE para el escenario de campañas de marketing.

Tabla 12 - Peticiones de pruebas en AZURE para el escenario de procesamiento de pagos.

Tabla 13 - Peticiones de pruebas en AZURE para el escenario de análisis de riesgo para ofrecer créditos.

Tabla 14 - Comparación de escenarios

LISTA DE FIGURAS

Ilustración 1 - Funcionamiento general de un sistema distribuido de limitación de tasa.

Ilustración 2 - Ejemplo de API Gateway.

Ilustración 3 - Ejemplo de algoritmo de Fixed Window.

Ilustración 4 - Ejemplo de algoritmo de Sliding Window.

Ilustración 5 - Ejemplo de algoritmo de Token Bucket.

Ilustración 6 - CAPEX Vs OPEX.

Ilustración 7 - Ejemplo de ataque DDoS.

Ilustración 8 - Interacciones de Componentes.

Ilustración 9 - Interacciones externas.

Ilustración 10 - Protección de payments-service.

Ilustración 11 - Protección de sessions-service.

Ilustración 12 - Limitación del uso de accounts-service.

Ilustración 13 - Arquitectura de componentes AWS.

Ilustración 14 - Arquitectura de componentes GCP.

Ilustración 15 - Arquitectura de componentes Azure.

Ilustración 16 - Diagrama del método de comparación

Ilustración 17 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de campañas de marketing.

Ilustración 18 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de procesamiento de pagos.

Ilustración 19 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de análisis de riesgo para ofrecer créditos.

Ilustración 20 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de campañas de marketing.

Ilustración 21 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de procesamiento de pagos.

Ilustración 22 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de análisis de riesgo para ofrecer créditos.

Ilustración 23 - Gráfico de tiempos de respuestas y errores en AZURE para el escenario de campañas de marketing.

Ilustración 24 - Gráfico de tiempos de respuestas y errores en AZURE para el escenario de procesamiento de pagos.

Ilustración 25 - Gráfico de tiempos de respuestas y errores en AZURE para el escenario de análisis de riesgo para ofrecer créditos.

LISTA DE SIGLAS

2FA: Segundos Factores de Autenticación.
API: Interfaz de Programación de Aplicaciones.
AWS: Amazon Web Services
AZURE: Microsoft Azure
CORS: Intercambio de recursos de origen distintos
DDoS: Denegación de Servicio
DRL: Sistemas de Limitación de Tasa Distribuida
GCP: Google Cloud Platform
IaaS: Infraestructura como servicio
JSON: Notación de Objetos Javascript
JWT: Json Web Token
HTML: Lenguaje de marcado de hipertexto
HTTP: Protocolo de transferencia de hipertextos
HTTPS: Protocolo de transferencia de hipertextos seguro
PaaS: Plataforma como servicio
OpenAPI: Especificaciones de para una lenguaje de definición de interfaz por maquina para describir, producir consumir y visualizar servicios en la web
OWASP: Proyecto Abierto de Seguridad en Aplicaciones Web
SAAS: Software como Servicio
SAML: Lenguaje de marcado de aserción de seguridad
Serverless: Modelo de Computación sin servidor
SOA: Arquitectura Orientada a Servicios
URI: Identificador Uniforme de Recursos
URL: Localizador Uniforme de Recursos
XML: Lenguaje marco extensible

Introducción

Definición del problema

Soluciones de limitación de tasa distribuida son utilizadas para restringir el uso de un recurso. Esto es deseable para proteger este recurso de algunas circunstancias, que puede implicar mejoras en la seguridad, ahorros en la solución, resiliencia en casos de mal uso de recursos críticos o inclusive la protección contra implementaciones poco confiables.

Hoy en día este control es exigido para soluciones robustas, por la necesidad de escalar, exposición a hackers e incluso en arquitecturas modernas como la de los microservicios.

En soluciones que poseen una dirección en el uso de microservicios que por naturaleza tiene muchas ventajas, pero al mismo tiempo tienen como base dos fundamentos principales: diseñar sistemas que fallan y sistemas que necesitan el uso de red. La productividad que presenta tal tipo de arquitectura vislumbra los ojos de grandes empresas, por este motivo para crecer se necesita el uso de dicho abordaje.

Este contexto obliga que arquitectos de sistemas y desarrolladores tengan más conciencia que en cualquier momento, otra aplicación pueda estar consumiendo o utilizando sus servicios. Sin embargo, se entiende que los servicios pueden fallar, pero desde la perspectiva de negocio existe la necesidad de seguir suministrando servicios a sus clientes. Es en este momento que tenemos que identificar y proteger activos y

recursos para detener que ante una determinada situación la solución, en su totalidad, pare de operar.

Para este proyecto, se toma como referencia un escenario que tendrá una arquitectura basada en una Fintech ficticia utilizando microservicios y se evaluará 3 distintas herramientas de sistemas de limitación de tasa distribuida con el foco en proteger activos críticos contra el mal uso.

Para la evaluación, definiremos atributos de arquitectura claves para el enfoque y criterio de éxito para cada una de las herramientas estudiadas.

Antecedentes

En los últimos años, ha venido creciendo el uso de soluciones en la nube, haciendo más fácil la construcción y adopción de nuevas ideas y soluciones.

Sin embargo, con esta demanda nacieron diferentes soluciones y nuevos negocios dado que el costo inicial en TI es más bajo.

Según (Oliveira et al., 2012) y (Krafzig et al., 2005), ejemplifican que soluciones como SOA, así como (Lewis & Fowler, 2014) describieron como los microservicios ayudan a dividir responsabilidades entre aplicaciones y equipos. Entretanto, tal estrategia ayuda a acelerar el crecimiento del equipo de ingeniería y la escalabilidad de las aplicaciones, pero es importante proteger las aplicaciones y servicios críticos de acuerdo con (Dragoni et al., 2017), surgiendo la necesidad de tener mejores abordajes de gobernanza de TI para identificar qué aplicaciones pueden consumir y utilizar tales servicios y cuáles servicios deben tener métodos de contingencia o incluso restringir

el consumo abusivo para evitar degradación de características principales del producto, como autenticación y pagos.

Sistemas distribuidos de limitación de tasa (DRL) en los cuales (Raghavan et al., 2007) describen que tienen como foco restringir y limitar que consumidores abusen de los atributos arquitectónicos de aplicaciones críticas como fue mencionado por (Stanojevic & Shorten, 2009a). Por más que el uso de soluciones en la nube ayude a escalar, es clave entender hasta qué punto se debe pagar para que la infra escale, como fue explicado en (Stanojevic & Shorten, 2009b), o inclusive como priorizar las características principales del producto sobre campañas de marketing u otros rasgos que no están dentro de las métricas principales del negocio de una empresa. Además, ayudan a minimizar problemas que puedan ocurrir cuando la infra recibe picos bruscos de peticiones, así en (Yoo et al., 2020) ejemplifican cómo adaptarse a tales escenarios.

(Raghavan et al., 2007) también aclaran que es importante enfatizar que los sistemas distribuidos de limitación de tasa (DRL) tienen como foco:

- Ordenar el uso de consumidores a aplicaciones críticas.
- Restringir el abuso de consumidores.
- Priorizar el escalado de métricas principales del producto cuando sea necesario.
- Evitar el uso excesivo de infraestructura que puede ser generado por implementaciones defectuosas, el mal uso de la solución o incluso ataques cibernéticos.

Con este foco la prioridad es siempre mantener la salud del producto definida por métricas de negocio, evitando de esa forma la pérdida de activos y usuarios, así como ahorrando en infraestructura.

Justificación

Dado el incremento y complejidad de las nuevas arquitecturas y soluciones en la nube, existe la necesidad de tener herramientas para proteger los principales activos y servicios de una solución.

Esta necesidad de disminuir costos y optimizar recursos alocados en soluciones en la nube hace que exista la necesidad del uso de limitaciones de la tasa distribuida (Distributed Rate Limiting - DRL).

En esta tesis vamos a investigar soluciones existentes en el mercado y compararlas en un escenario utilizando microservicios en Amazon (AWS), Google Cloud (GCP) y Azure de acuerdo con (Cohen, 2021).

Alcance y Limitaciones

- Realizar un estudio preliminar de las tecnologías base de sistemas de limitación de tasa distribuida.
- Análisis de componentes arquitectónicos de los proveedores de soluciones en la nube: AWS API GATEWAY (AWS), CLOUD ENDPOINTS (GCP) y Azure API MANAGEMENT POLICIES.
- Presentación de un método de evaluación para la implementación de un prototipo funcional.

Objetivos

Principal

- Desarrollar un modelo para evaluar el correcto uso de sistemas distribuidos de limitación de tasa (DRL)

Específicos

- Analizar SaaS existentes para el uso de DRL
- Realizar estudios comparativos de las características de un DRL
- Definir un conjunto de criterios alineados a modelos arquitectónicos para evaluar el uso de un DRL.
- Definir métricas para criterio de evaluación
- Elaborar un modelo con los criterios seleccionados para evaluar un DRL.
- Desarrollar una aplicación prototipo para validar el modelo propuesto

Fundamentos Teóricos

Introducción del Capítulo

Datadome (Ago 2020) declara Rate limiting, o limitación de tasa, como una estrategia que limita el tráfico a un recurso. Sin embargo, en internet existe la necesidad del consumo de APIs y servicios es necesario poder protegerse ante el mal uso o implementaciones débiles que consumen tales servicios.

Hablamos qué solución en nube queremos y que escalen con la demanda sin impactar el usuario, pero tenemos que tener en cuenta que la necesidad de escalar puede tener distintos orígenes, como campañas de marketing, ataques cibernéticos, mal diseño de arquitectura, bugs introducidos en el sistema o incluso el crecimiento exponencial de un producto.

Es importante entender e identificar tales causas para que una solución en la nube escale de forma saludable a punto de utilizar los recursos de la mejor forma posible. En este trabajo vamos a poner el foco en cómo podemos proteger microservicios críticos de una solución para evitar el consumo abusivo, garantizando que la misma pueda proveer funcionalidades básicas, aunque sea para una población limitada.

(Yang et al., 2005), (Dickel et al., 2019) y (Kumar et al., 2020) describen y ejemplifican el uso de tal solución para protegerse de bots o ataques de denegación de servicio (DDoS), limitando el impacto dado por fallas en cascada o restringiendo el uso de un servicio de acuerdo a comportamientos anómalos, de esa forma se mejora la disponibilidad de soluciones.

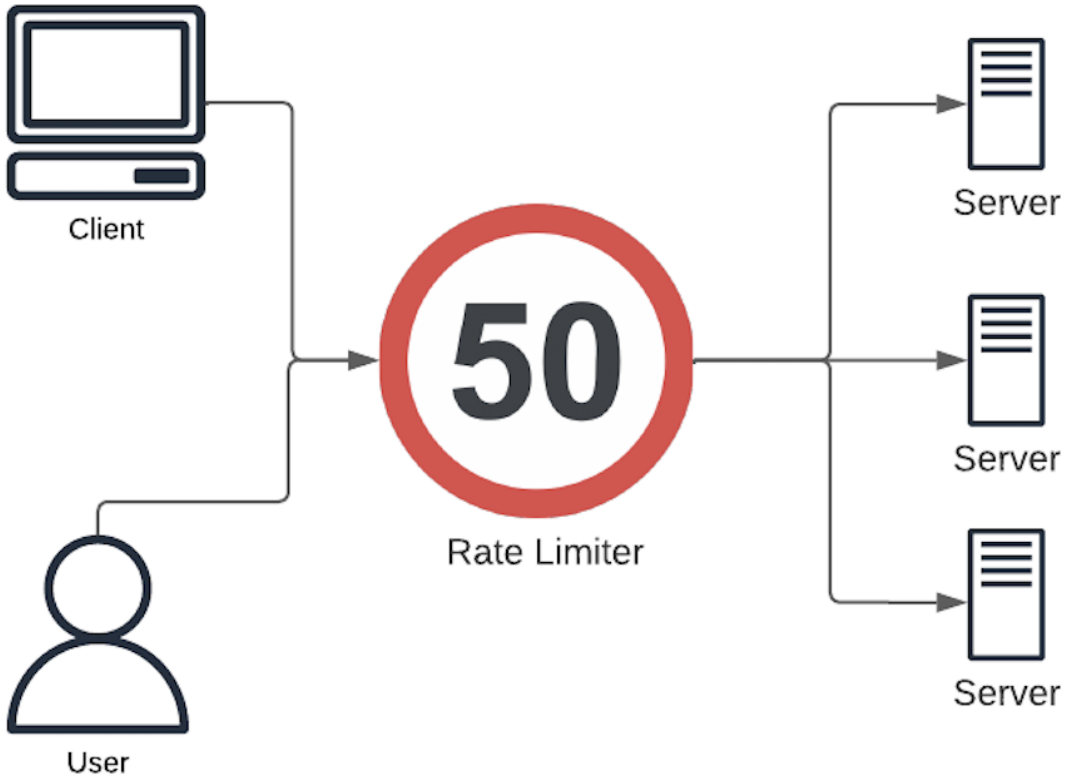


Ilustración 1 - Funcionamiento general de un sistema distribuido de limitación de tasa.

En este capítulo vamos a mencionar los trabajos relacionados para exponer los distintos problemas que un limitador de tasa resuelve, enseguida vamos a ejemplificar los algoritmos más utilizados con sus ventajas y desventajas y al final vamos a describir las tecnologías más conocidas en el mercado así con sus respectivos algoritmos.

Microservicios

Según (Lewis & Fowler, 2014) y (Thones, 2015), las arquitecturas con base en los microservicios han surgido hace casi 10 años y describen una forma de diseñar aplicaciones de software como un conjunto de servicios de implementación

independiente. Esta arquitectura es deseada por las grandes empresas por ser ventajoso en poder reforzar el foco en un dominio del problema, la colaboración y organización de equipos. Tales características ayudan en la expansión de la empresa y la especialización en el producto estimado. Desde el punto de vista técnico se hace más fácil ordenar y crear abstracciones para soportar la evolución del negocio, evitando así que la deuda técnica esté desparramada por todo el código de la plataforma.

Sin embargo, esta arquitectura estima crear servicios independientes, sabemos que en los problemas reales existe la necesidad de la comunicación e integración entre los microservicios.

De acuerdo con (Bogner et al., 2019), los microservicios poseen muchas ventajas con relación a la escalabilidad, flexibilidad y mantenibilidad, existen innumerables desafíos cómo se podría definir patrones de seguridad, mantener consistencia de datos, garantizar la resiliencia, disponibilidad, etc. (Doyle et al., 2010) detallando ejemplos de resiliencia utilizando las limitaciones de tasa (DRL) y esto se debe porque una arquitectura de microservicios tiene como base el uso de la red para poder comunicarse entre otros servicios y la misma es susceptible a fallas, por transitividad nuestros microservicios también son susceptibles a fallas como fue detallado por (Düllmann & van Hoorn, 2017).

Tal desafío exige que arquitectos y desarrolladores tengan que crear mecanismos de resiliencia o incluso optar por no escalar eternamente su infraestructura.

En conjunto con lo establecido surgen patrones como el uso de circuit breakers, arquitecturas en capas, gateways, proxy inversos, DRL, etc. Por eso vamos a enfocarnos en las limitaciones de tasa (DRL) para que evaluemos cómo utilizarlas y cómo pueden tornar una solución robusta.

API Gateway

API Gateway (Gerardus B., 2018) es un un patrón de diseño en lo cual exponemos API y microservicios de forma centralizada en la cual sabe redireccionar las peticiones al servicio responsable.

Con este patrón de diseño podemos simplificar los clientes porque puede hacer llamadas solamente a un único punto de entrada. Entretanto, cada servidor puede decidir como el cliente debe interactuar con sus recursos y cómo debe ser el modelo expuesto.

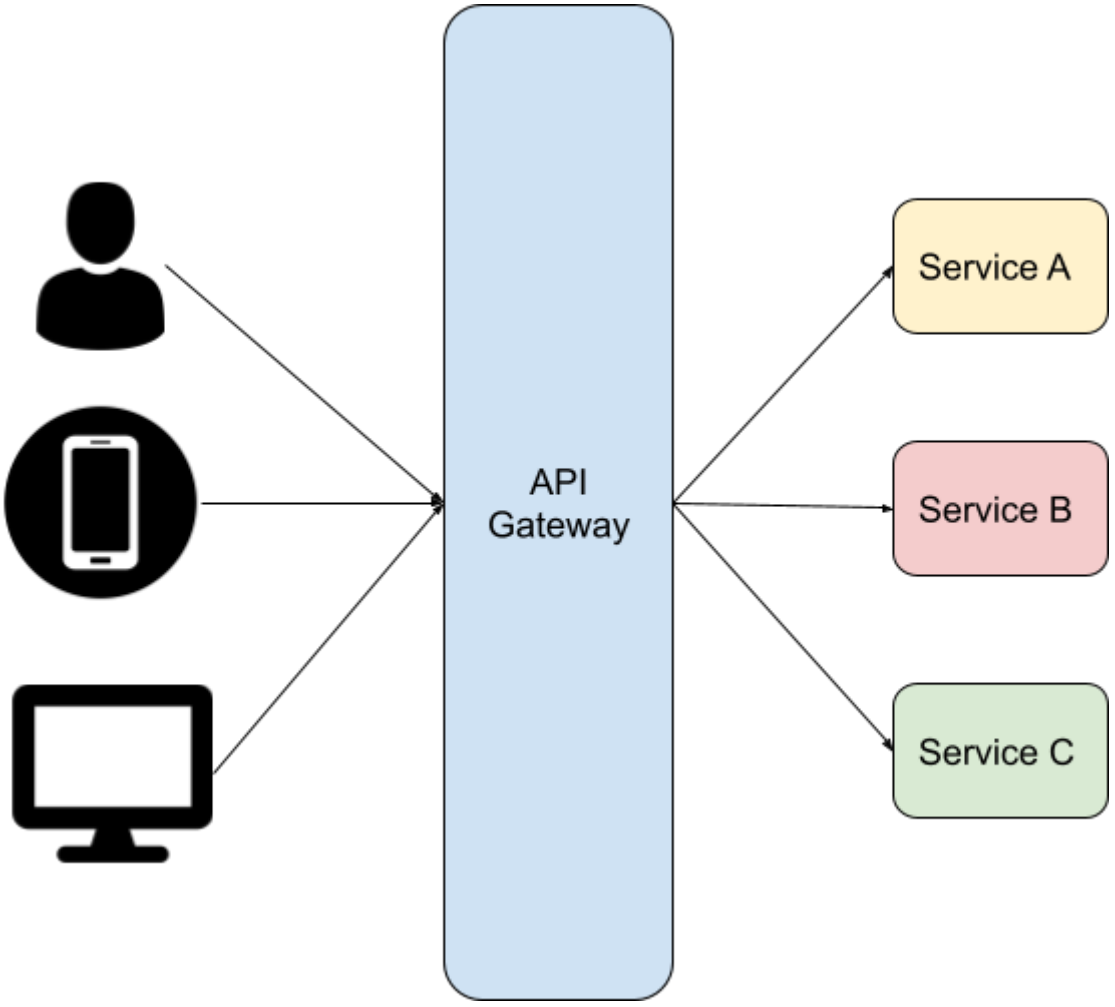


Ilustración 2 - Ejemplo de API Gateway

Como podemos ver en la ilustración anterior los clientes a la izquierda no necesitan conocer cada servicio para poder interactuar con la solución. Además de esto este patrón de diseño permite tener mecanismos de seguridad, como autenticación y rate limiting para proteger los servicios que están a la derecha.

Algoritmos de Limitación de Tasa

En (Rate-Limiting Strategies and Techniques | Cloud Architecture Center, 2019), se describen 3 tipos de estrategias más comunes de implementación de sistemas de

limitación de tasa y los vamos a estar describiendo en esta sección. Estas estrategias son importantes para entender las limitaciones de las herramientas estudiadas en esta tesis y las ventajas para controlar el uso excesivo de nuestros microservicios.

El primer y más simple algoritmo se llama Fixed Window, este algoritmo utiliza una ventana fija con una tasa fija para rastrear las solicitudes de un cliente, esto es posible con un contador (cronómetro, reloj) que se reinicia cuando la ventana termina. Uno de los algoritmos más básicos tiene como base ventanas de tiempo que determinan en una taza por minuto de peticiones, por ejemplo mil (1000) solicitudes por minuto esto pero esto permite picos de peticiones lo que puede sobrecargar los servicios.

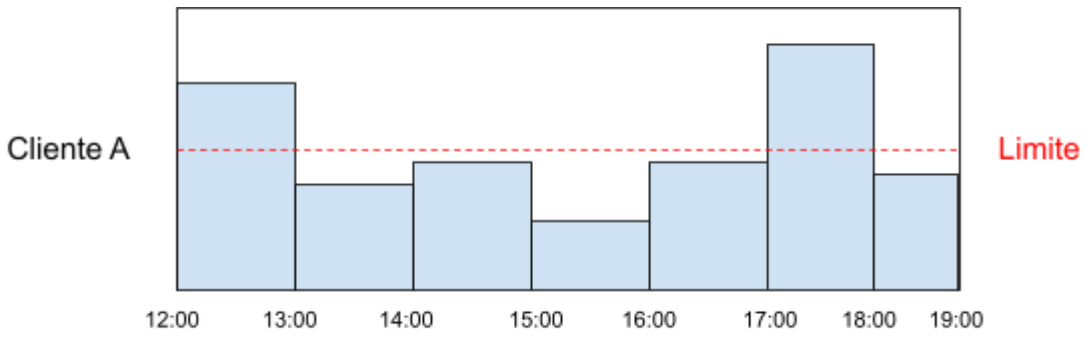


Ilustración 3 - Ejemplo de algoritmo de Fixed Window

Al mismo tiempo podemos encontrar implementaciones, como las descritas en (Henriksson & Bennhage, 2021), Andreas, y Svante Bennhage (2021) o en (Doyle & O'mahony, 2012) que utilizan solicitudes por segundo, con el fin de tener mayor control sobre la limitación del uso de tal servicio. Cuando el contador supera el límite acordado anteriormente, el Rate Limiting descarta las solicitudes siguientes.

Sin embargo este algoritmo es simple de implementar y mantener, su API puede sobrecargarse ya que el algoritmo no válida la salud de su servicio. Además, como

este algoritmo no se adapta a todo el tráfico recibido es necesario actualizar la configuración de la ventana en caso de que su servicio soporte más o menos solicitudes.

Teniendo en cuenta las desventajas mencionadas anteriormente , existe una evolución del Fixed Window, que su nombre es Sliding Window. Este algoritmo combina lo mejor del algoritmo anterior con un registro deslizante. Con esto tenemos un contador acumulativo para el periodo establecido, pero también se evalúa el periodo anterior para suavizar picos de tráfico entre las ventanas. Por lo tanto, con esta pequeña mejora y un poco más de procesamientos de puntos necesarios para evaluar cada solicitud este algoritmo es el ideal para procesar grandes cantidades de solicitudes sin impactar de forma negativa el performance de la solución.

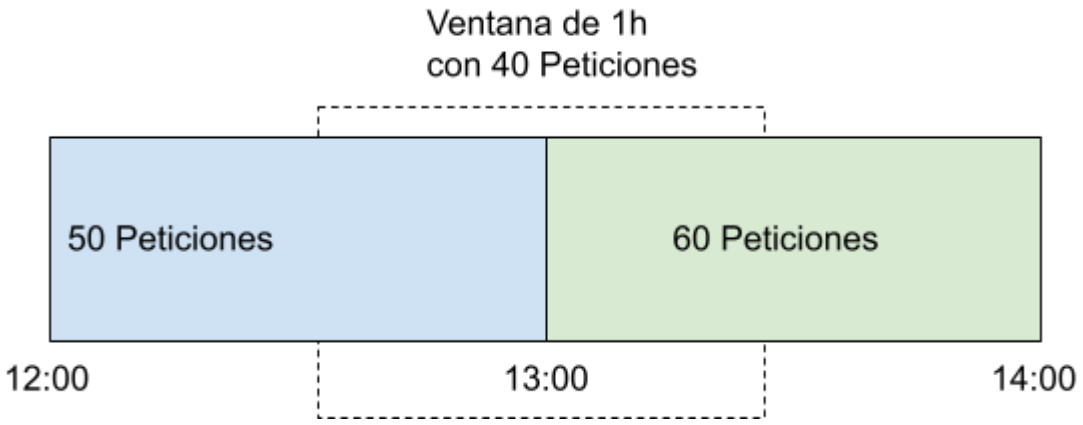


Ilustración 4 - Ejemplo de algoritmo de Sliding Window.

Por último, tenemos un algoritmo llamado Token bucket, este algoritmo tiene como foco una técnica en la cual una solicitud puede resultar en múltiples operaciones internas. Con esto cada cliente tiene un número de tokens con un presupuesto de uso continuo y acumulativo, cuando se hace una operación a un servicio se intenta retirar

un token para cumplir tal solicitud, cuando no hay más tokens, significa que el servicio ha llegado a su límite.

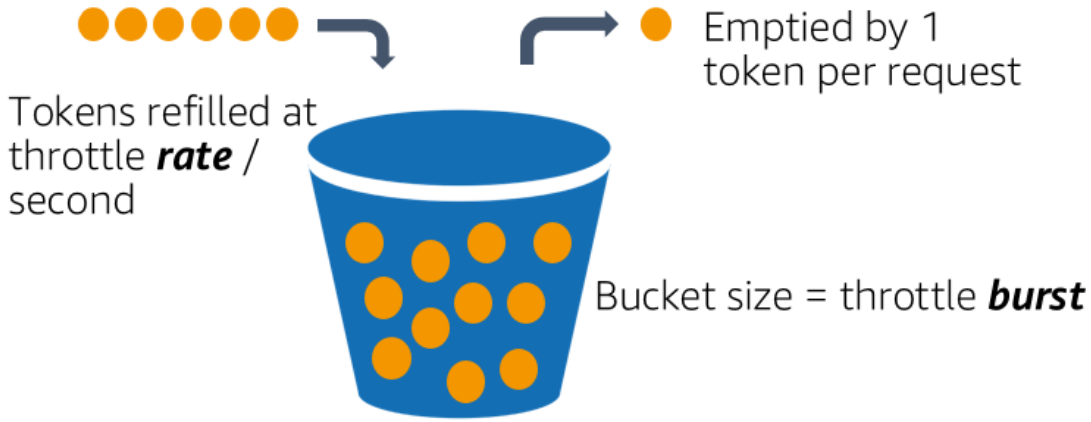


Ilustración 5 - Ejemplo de algoritmo de Token Bucket

Fuente: (Building well-architected serverless applications, 2021)

Los algoritmos descritos anteriormente son implementados de una forma que sean flexibles frente a pequeñas oscilaciones en el tráfico de modo que priorice el performance sobre la consistencia en los límites.

Autenticación y Autorización

En los días actuales la seguridad es una de las características más importantes en el desarrollo de soluciones en la internet. Para esto uno de los requisitos necesarios para cumplir tal característica es que se debe implementar la autenticación para que sea posible identificar un actor o usuario para interactuar con el sistema y además de esto se debe autorizar al uso del recurso. O sea, un actor debe probar que es quien dice ser

y solo puede hacer uno conjunto de acciones en las cuales cumple con su rol (Hwang, J, 2000).

Como descrito en (Geer, 1998) es importante preocuparse con la gestión de riesgos y esto puede ser cumplido con la autenticación y autorización.

Este atributo de arquitectura es clave para el uso de sistemas distribuidos de limitación de tasa, porque la limitación a un recurso se da de acuerdo con un actor.

Autenticación	Autorización
Permite identificar a un usuario de forma con que el mismo pruebe decir quien es	Permite definir se un usuario tiene acceso a un recurso
Es posible utilizar 2FA para comprobar la autenticidad del usuario	Usado para restringir acceso a datos sensibles
Necesario para garantizar seguridad de usuarios finales	Necesario para cumplir políticas de privacidad
En general ocurre antes de una autorización	En general ocurre después del proceso de autenticación exitosa

Tabla 1 - Comparación entre autenticación y autorización

Cloud Computing

Cloud Computing o computación en nube es la disponibilidad de servicios, infraestructura y recursos de computación de acuerdo con la necesidad de demanda o uso a través de internet (Google Cloud, 2023).

Este modelo que permite tener acceso a infraestructura y servicios bajo demanda se contrapone al modelo tradicional que tiene como metodología hacer una inversión inicial, llamada de CAPEX, comprando computadoras, hardware e infraestructura en el comienzo del proyecto. Pero con este nuevo modelo la área de TI puede cambiar a

una inversión de acuerdo con el uso, llamado de OPEX, esto trae una ventaja para probar nuevas ideas y negocios sin tener que hacer grandes inversiones iniciales.

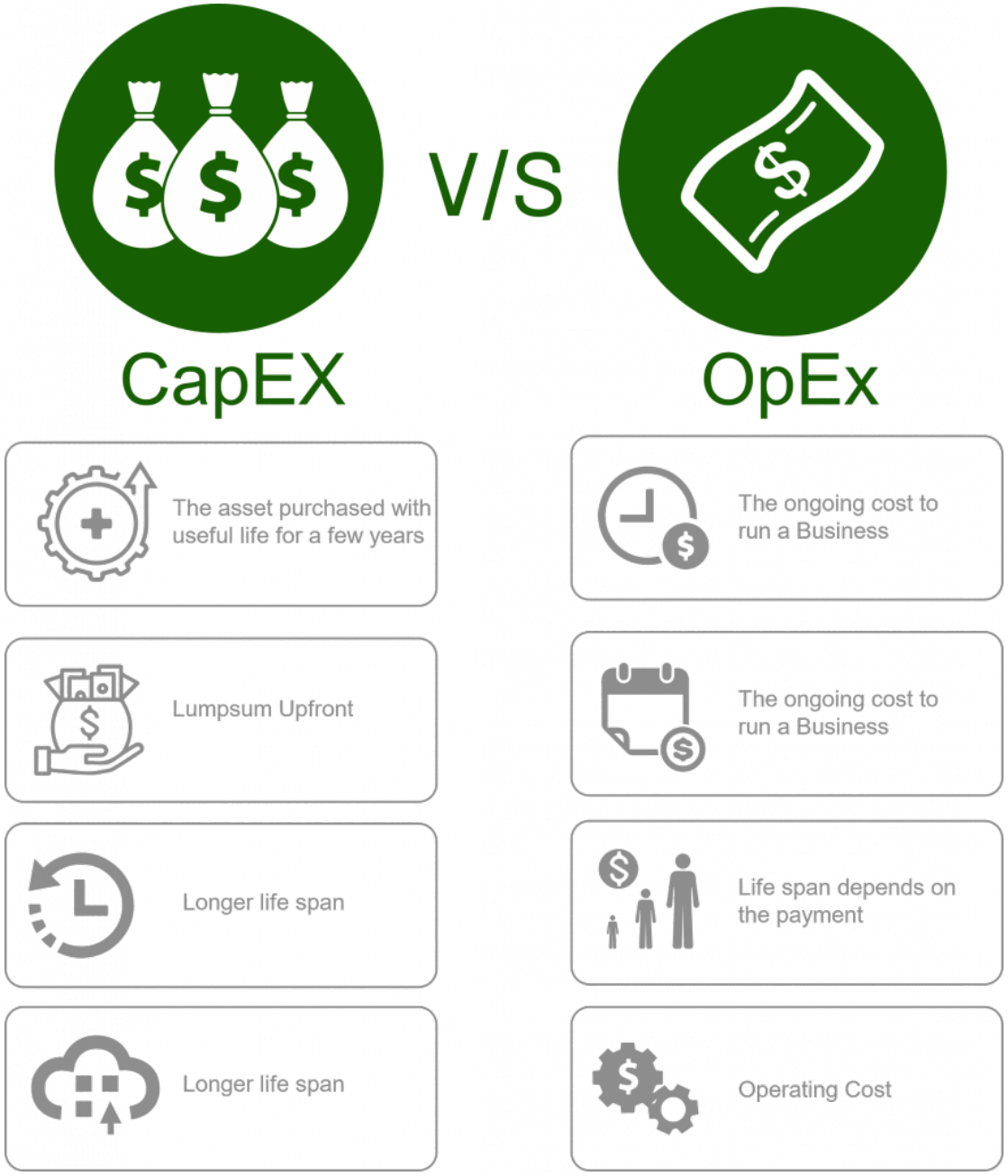


Ilustración 6 - CAPEX Vs OPEX

Fuente: Admin. (2021)

Según (Qian, L. 2009) la adopción de cloud computing trae otras ventajas como:

- Elasticidad: Es más fácil agregar o disminuir la cantidad de servidores necesarios para poder soportar el tráfico que es recibido en la solución o mismo utilizar más hardware cuando es necesario tener más computación.
- Seguridad: Los proveedores de nube brindan niveles de seguridad según el tipo de servicio o recurso que esté utilizando. Por esto es importante entender el modelo (shared responsibility)
- Costos: Es más fácil predecir los costos dado que es bajo demanda.
- Actualizaciones: El proveedor de nube se encarga de hacer actualizaciones de los servicios y de los sistemas operativos.
- Resiliencia: Tiene más herramientas ante fallos y utilizando DevOps es más fácil hacer la recuperación de servicios.
- Disponibilidad: Utilizando proveedores de nube podemos elegir distintas regiones para desplegar servicios.
- Flexibilidad: Desarrolladores pueden elegir distintas tecnologías y también es posible el uso de arquitecturas multi-cloud.

Por lo tanto, las empresas están eligiendo el uso de cloud computing por la velocidad en el desarrollo de nuevos productos trayendo una ventaja tecnológica.

Sin embargo este modelo de desarrollo trae muchas ventajas, el mismo exige tener profesionales de TI especializados en DevOps para poder proveer soluciones robustas. Por lo cual es importante entender que existen 3 tipos de servicios de computación en la nube, llamados IaaS, Paas y Saas.

Iaas o Infraestructura como servicio es definido como un servicio en la nube que disponibiliza infraestructura totalmente configurable en la nube. En este modelo el cliente debe elegir las características de un computador y entenderlas pero sin la responsabilidad de saber cómo están alojadas. Además el equipo de TI debe tener conocimiento previo de infraestructura para poder utilizarlo.

Ya en Paas o Plataforma como servicio, es un modelo en el cual se implementa y despliega soluciones de plataforma en la nube. Ejemplos como este tenemos kubernetes, Heroku, soluciones de dockerizacion. En general es una capa que fue implementada arriba de un IaaS y simplifica la complejidad de conocer como la infraestructura debe ser desarrollada y mantenida.

Y por fin Saas o Software como Servicio, es un modelo en el cual esta mas proximo de reglas de negocio y el desarrollador o el equipo de TI no tiene que saber cual es el hardware o infraestructura utilizado para poder ejecutar las funciones del software. Por tanto, el foco en sí es proveer características próximas a un negocio abstrayendo complejidades técnicas.

Abajo tenemos una comparación entre los modelos:

IaaS	PaaS	SaaS
Uso de hardware para la construcción de soluciones de TI	Uso de un entorno para la construcción de soluciones de TI	Uso de software para la construcción de productos o mejoras en procesos de empresas
Necesidad de conocimiento en Infraestructura	Abstrae detalles de infraestructura, pero trae libertad de customización caso necesario	No hay necesidad en conocer nada sobre infraestructura

Foco en mejorar la gestión y uso de infraestructura	Foco en abstracciones para la gestión y uso de infraestructura	Foco en el negocio
Flexibilidad en gestión de costos	Flexibilidad en gestión de costos	Facilidad en la gestión de costos
Necesidad de tener equipos de DevOps	Necesidad de tener equipos de DevOps	Sin necesidad de tener equipos de DevOps

Tabla 2 - Comparación entre IaaS, PaaS y SaaS

Como podemos ver los distintos modelos tienen siempre como base la computación en nube, pero agregando características o productos para que sea más fácil su uso.

Definición de Tecnologías Base

En este estudio hemos elegido 3 tecnologías base para la comparación de los experimentos. El criterio de las elecciones se dió primeramente por el uso de soluciones en la nube, que sean populares de acuerdo con (Top 10 Cloud Infrastructure Providers, 2022) y (Zhang, 2022). Además elegimos tecnologías que tienen mucha adopción, facilidad de uso en el mercado y buen soporte. Esto ocurre con SaaS y PaaS que son desarrollados por grandes empresas como Amazon, Google y Microsoft. Por esto elegimos API Gateway AWS, Google Cloud Endpoints y Azure API Management Policies. Estas herramientas son elegidas porque desarrollan el patrón de diseño llamado API Gateway, descrito en (Microservices Pattern: API Gateway Pattern, 2019). Sin embargo nuestro foco son los rate limiters, cada tecnología elegida utiliza un mecanismo de limitaciones de peticiones y autorización lo cual son construidos con DRL's (Distributed Rate Limiters) (Datadome, Ago 2020).

API GATEWAY AWS

Amazon API Gateway es un servicio, serverless, completamente administrado por AWS que facilita a los desarrolladores la creación, publicación, mantenimiento, monitoreo con integraciones a AWS Cloud Watch y protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica comercial o la funcionalidad de sus servicios de back-end. Así no importa como este sus servicios, si utilizan EC2, kubernetes o mismo AWS Lambdas es posible exponerlos en API Gateway de forma segura.

Con API Gateway, puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web.

Por ser completamente administrado por AWS, posee mucha resiliencia con la posibilidad de configurar reglas de restricciones a recursos o mismo limitar el número de llamadas a los mismos. En conjunto es posible configurar formas distintas de autorizaciones como el soporte a tokens JWT, SAML, ejecuciones a funciones lambda, etc.

Una ventaja a nombrar es que es posible generar SDKs a clientes para el uso, lo que facilita el desarrollo.

API Gateway gestiona todas las tareas relacionadas con la aceptación y el procesamiento de hasta cientos de miles de llamadas API simultáneas, incluida la gestión del tráfico, la compatibilidad con CORS, la autorización y el control de

acceso, la limitación, la supervisión y la gestión de versiones de API. API Gateway no tiene tarifas mínimas ni costos de inicio. El costo es manejado por la cantidad de llamadas que la API recibe y la cantidad de datos transferidos y, con el modelo de precios escalonados de API Gateway, puede reducir su costo a medida que escala el uso de la API.

GOOGLE API GATEWAY

Google API Gateway es el producto totalmente gerenciado por google serverless en lo cual es posible exponer servicios desplegados con Google Cloud Run, Google Cloud Functions y Google App Engine.

Este producto permite el acceso a través del protocolo HTTP a APIs REST de forma segura pudiendo configurarlo utilizando especificaciones del OpenAPI 2.0. También es posible configurar autenticación entre servicios, cuotas, límites de uso y soporta servicios desarrollados en distintos lenguajes desde que sean APIs REST.

Aunque sea, totalmente gerenciado por google, posee monitoreo, soporte contra DDoS y auditoría sobre cambios.

Es una herramienta de fácil uso, pudiendo agregar seguridad y monitoreo entre la capa pública y privada de soluciones modernas.

Además posee validaciones de autenticación para clientes en las cuales es posible configurar implementaciones distintas por esto es muy flexible. Es una herramienta completa principalmente para equipos de TI que quieren abstraerse de infraestructura y prever costos.

AZURE API MANAGEMENT

Azure posee un producto en lo cual nos ayuda a proteger y exponer nuestras APIs llamado de Azure API Management, tal producto tiene distintas características como autenticación, autorización, limitación de tasa distribuida, integraciones multi-cloud y on-premises.

Azure API Management es construido con un API Gateway, un management plane y un developer portal. Estos componentes son totalmente gerenciados por Azure y ayudan a configurar, monitorear y hostear su API Gateway.

Para esto vamos a describir cada componente:

- API Gateway: recibe las peticiones de los clientes y las redirecciona a cada servicio responsable por el recurso. Este componente implementa características como configuraciones de roteamento, seguridad, limitaciones de tasa, caching y observabilidad.
- Management Plane: posibilita el aprovisionamiento y configuraciones en el API Management. Utilizando especificaciones OpenAPI es posible definir esquemas de API y cada servicio responsable por el recurso, configurar cuotas, gestionar usuarios y políticas sobre sus recursos.
- Developer Portal: es un sitio web en el cual se encuentra la documentación de su API.

Cuando hablamos de policies o políticas de gestión, en Azure API Management, traen la capacidad al sistema de tener comportamientos distintos a través de configuraciones en el Management Plane. Las políticas se definen por ser un grupo de reglas que son

ejecutadas en cada petición o en cada respuesta de sus API. Es posible definir políticas sobre autenticación, autorización, estandarización en errores, soporte a OAuth, cache, adición de headers, logging.

Como mencionado anteriormente este producto posee limitadores de tasa (DRL), lo que se llama de API Management Policies, así podemos crear políticas de uso de nuestras API s de acuerdo con cada suscripción o cliente integrado a nuestro servicio. También es posible configurar reglas con IP 's evitando DDoS.

API RESTful

API RESTful (Richardson, L. 2008) es un patrón de diseño con base en el cliente servidor en lo cual se el servidor define una interfaz que define un modelo para que los clientes puedan comunicarse para el cambio de informaciones de forma segura en la internet.

La comunicación entre servidor y cliente es realizada a través del protocolo HTTP/HTTPS en lo cual se puede utilizar formatos JSON, XML, HTML, etc para definir el contrato y el mismo está disponible desde un Identificador Uniforme de Recursos (URI).

Por fin, para que una arquitectura cumpla con las condiciones de REST se debe tener en consideración:

- Los recursos deben estar disponibles en HTTP utilizando la arquitectura cliente-servidor
- Tener una interfaz uniforme de los componentes y en un formato

- Los recursos deben ser identificados en distintas URI's
- Tener una arquitectura en capas para que sea posible implementar otras responsabilidades, como seguridad, logs, rate limiters, etc..
- Poder utilizar datos en cache para optimizar algunas consultas.

Denegación de Servicio

Denegación de servicio, como explicado en (Giansanti, D., 2021) es uno de los ataques más comunes en 2016 y es un tipo de ciberataque en el cual el atacante tiene como objetivo hacer con que un dispositivo, sitio, servicio o recurso esté indisponible, interrumpiendo su funcionamiento normal para los usuarios reales.

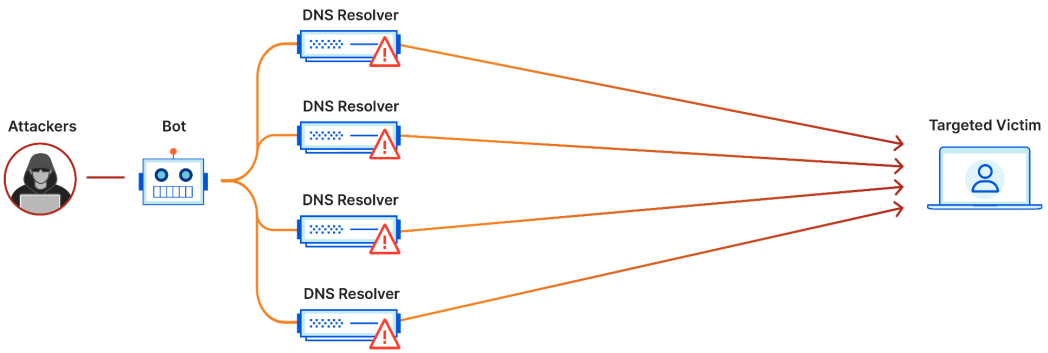


Ilustración 7 - Ejemplo de ataque DDoS

Fuente: (What is a distributed denial-of-service (ddos) attack. 2023)

En general este tipo de ataque intenta sobrecargar la infraestructura inundando con demasiado tráfico al punto de que el servicio no pueda escalar. Sin embargo como

podemos ver en la ilustración anterior el atacante utiliza bots para realizar el ataque, también es posible hacerlo con el uso de malware afectando maquinas de usuarios comunes para inundar un sistema en particular.

Entretanto tal ataque se tornó muy común en los últimos años, existen algunas abordajes para poder identificarlo, minimizando su impacto.

- Análisis de patrones en el tráfico a sus microservicios, identificando dispositivos sospechosos de acuerdo con su dirección de IP o un rango de IP's
- Anomalías en el tráfico a una URI en particular. Esto puede estar ocurriendo también para que el atacante intente obtener alguna información sensible.
- Picos de tráfico en horarios no comunes al horario de uso del producto. Por ejemplo, hay mucho tráfico de compras en un marketplace a las 2:00 am.

Conclusión del Capítulo

En este capítulo pudimos ver los proveedores de soluciones en la nube más relevantes y como ellos entregan productos en base el patrón de diseño Api Gateway que es muy importante para la construcción de las soluciones modernas y como el DRL es parte clave de tales soluciones.

Modelo

Definición

En este trabajo vamos a enfocarnos en un modelo que se asocia a atributos arquitectónicos con una solución que posee DRL.

Los atributos arquitectónicos elegidos son: **mantenibilidad, performance, escalabilidad, disponibilidad y seguridad**. Estos atributos van a direccionar la metodología de comparación de los resultados obtenidos en nuestro escenario utilizando cada tecnología base.

Con esto en mente vamos a realizar investigaciones sobre las tecnologías base con el objetivo de identificar cada atributo de la siguiente forma:

- **Mantenibilidad** tiene como objetivo evaluar la facilidad en el uso de la tecnología, como también cuán flexible, fácil de configurar y a adaptarse a distintos escenarios.
- El **performance** se va a medir de acuerdo con los tiempos de respuesta de los enlaces entre las distintas tecnologías base.
- En cuanto a la **escalabilidad** debemos demostrar que la solución se adapta de acuerdo con el tráfico.
- La **Disponibilidad** será medida de acuerdo con la cantidad de respuestas exitosas, incluso en situaciones en las cuales estamos forzando los límites que fueron configurados para cada cliente.

- La **Seguridad** se mide entendiendo como tal patrón de arquitectura puede ser combinado con otros protegiendo nuestros activos, por ejemplo la autenticación y autorización de los clientes.

Los atributos **Mantenibilidad**, **Escalabilidad, Disponibilidad, Performance** y **Seguridad** van a tener una escala de **Aprobado** y **No Aprobado**.

En seguida vamos a describir cómo utilizamos la escala propuesta y cómo vamos a adoptarlos en los escenarios de: **Campañas de marketing, Procesamiento de pagos online y Análisis de riesgo en batch.**

Atributos Arquitectónicos	**Aprobado**	**No Aprobado**
Mantenibilidad	Fácil de realizar cambios y soporte a distintos encabezados http	Posee un patrón pero no configurable
Performance	No genera picos en tiempos de respuesta	Genera picos no consistentes en tiempos de respuesta
Escalabilidad	Se ajusta conforme el tráfico	No se ajusta conforme el tráfico
Disponibilidad	Solo presenta errores HTTP 429	Suma errores desconocidos cuando tenemos mucho tráfico
Seguridad	Es posible identificarlo de acuerdo con cada cliente	No es posible identificar el cliente

Tabla 3 - Evaluación de cada atributo arquitectónico.

Se crearon simulaciones para cada escenario utilizando una herramienta llamada (Locust - a Modern Load Testing Framework, 2022) que torna posible la creación de scripts en Python para hacer pruebas de carga y generar reportes de cada tecnología base. Locust posee una interfaz que nos ayuda en la comparación ya que posee histogramas en tiempo real de peticiones, errores y tiempos de respuesta.

Motivación

El modelo debe ser utilizado con el fin de poder orientar arquitectos, desarrolladores y CTO's a dominar mejor el uso de los activos más importantes de una empresa, evitando (erradicando) la interrupción del negocio por errores humanos o por ataques cibernéticos.

En paralelo con el modelo debe ser en conjunto con una estrategia empresarial de tecnología, en la cual se elige herramientas y soluciones entendiendo las ventajas y desventajas para la toma de decisiones.

Conclusión del Capítulo

En este capítulo pudimos describir como nuestro modelo evalúa cada uno de los atributos arquitectónicos y el porqué de tales atributos sean importantes cuando hablamos de soluciones en las que se necesiten un DRL.

Caso de Estudio

Introducción del Capítulo

En este capítulo vamos a describir la metodología utilizada en el experimento,
por lo que vamos a especificar los casos de uso así como su arquitectura y como la arquitectura puede ser implementada con cada tecnología base.
Por fin vamos a detallar cada funcionalidad con la que queremos proteger con la limitación de tasa y cómo esto puede influenciar en la comparación en los resultados.

Metodología

Para la construcción del escenario vamos a utilizar microservicios en diferentes lenguajes, como python, java, kotlin y golang y para la configuración de infraestructura vamos a utilizar una herramienta de devops llamada Terraform que ayuda a orquestar y manejar recursos en la nube.

Cómo fue nombrado en las tecnologías base vamos a utilizar dos tecnologías de código abierto en las cuales vamos a estar utilizando EC2 con auto escalados en AWS y herramientas SAAS de AWS (API Gateway) y Google (Google Cloud Endpoints). Terraform nos va a facilitar el intercambio entre las soluciones de infraestructura porque posee integraciones directas con los proveedores de la nube.

Escenario

Vamos a trabajar en un escenario hipotético de una fintech que posee 6 microservicios principales que vamos a describir a continuación.

La fintech posee servicios básicos de datos del usuario, de acceso a la plataforma como acceso y autorización de transacciones. Además de esto tenemos que realizar operaciones de pagos, así como almacenar el historial de cada movimiento y poder analizar los riesgos en las operaciones.

Así mismo tenemos users-service que maneja los datos de los usuarios, sumado a esto tenemos el sessions-service que maneja las sesiones de los usuarios y las obtiene a partir del uso de factores de autenticación. Para simplificar la solución no vamos a detallar servicios que manejan la validación de los factores de autenticación, pero es importante entender que también hay un desafío que merece su respectiva atención cuando hablamos de grandes empresas que deben proveer seguridad a sus usuarios.

Una vez que tenemos los datos de los usuarios es necesario mantener una relación entre el usuario y sus cuentas virtuales. Tenemos un servicio, llamado accounts-service, que almacena e identifica de forma única a las cuentas. En seguida tenemos payments-service, responsable por gestionar la lógica de pagos y movements-service que almacena cada operación. Por fin, pero no menos importante tenemos un servicio, risks-service, que analiza los riesgos del usuario asincrónicamente en batches y analiza el riesgo de cada operación de forma transaccional.

Dado el escenario descrito anteriormente, vamos a destacar algunas funcionalidades claves y cómo estos microservicios interactúan entre sí.

Vista de componentes del Escenario

Vamos a mostrar dos vistas, la primera es una pantalla general de los microservicios presentes en la plataforma y sus interacciones básicas. Y en la segunda tenemos una vista externa para ejemplificar qué actores pueden estar utilizando nuestra plataforma.

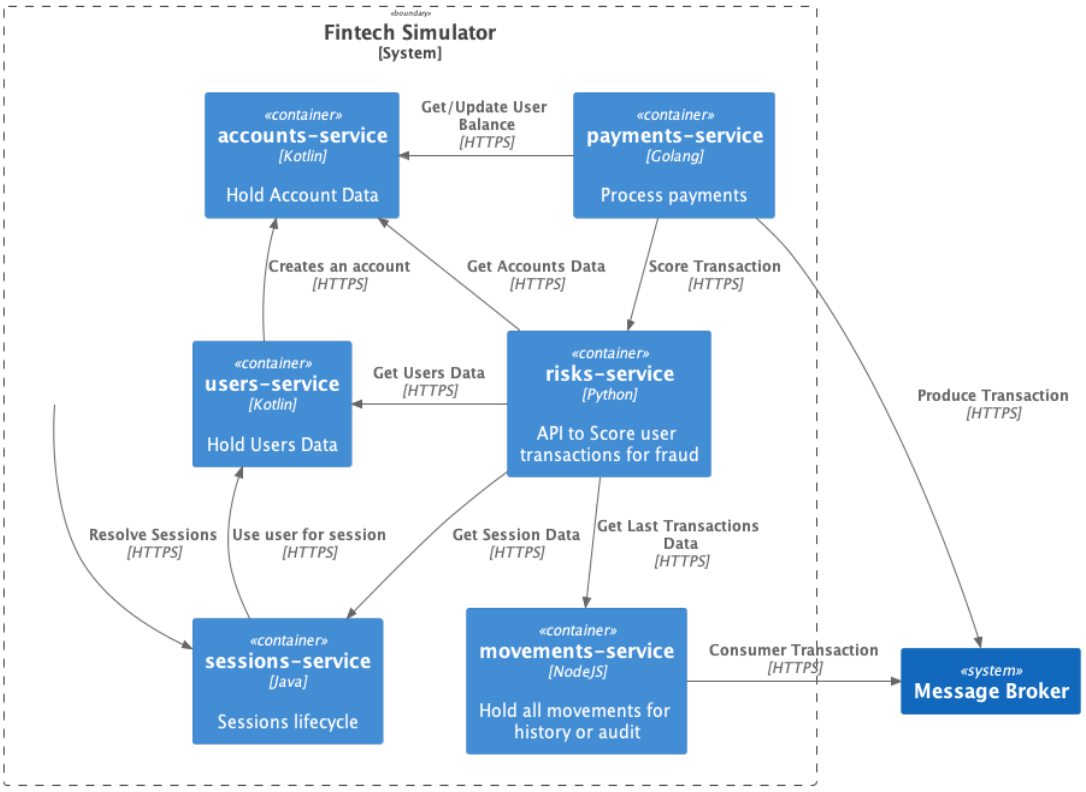


Ilustración 8 - Interacciones de Componentes.

En la ilustración 2 podemos notar distintas interacciones y que algunos componentes son más críticos que otros. Por ejemplo, risks-service necesita hablar con todos los otros servicios para poder nutrir su análisis de riesgo con datos y poder evaluar la operación o al usuario. Por otro lado tenemos servicios como de pagos que sería la

parte más importante del negocio, pero que ningún otro servicio depende de este para proveer una funcionalidad. Por fin tenemos servicios que reciben consultas de distintas funcionalidades como accounts-service o sessions-service, entonces este tipo de servicio puede o no hacer parte del camino crítico de la plataforma.

Cuando creamos una solución es importante identificar estos caminos críticos para poder identificar puntos de mejora. En este proyecto tenemos el foco en los sistemas que hacen parte del camino crítico, como payments-service, sessions-service y accounts-service.

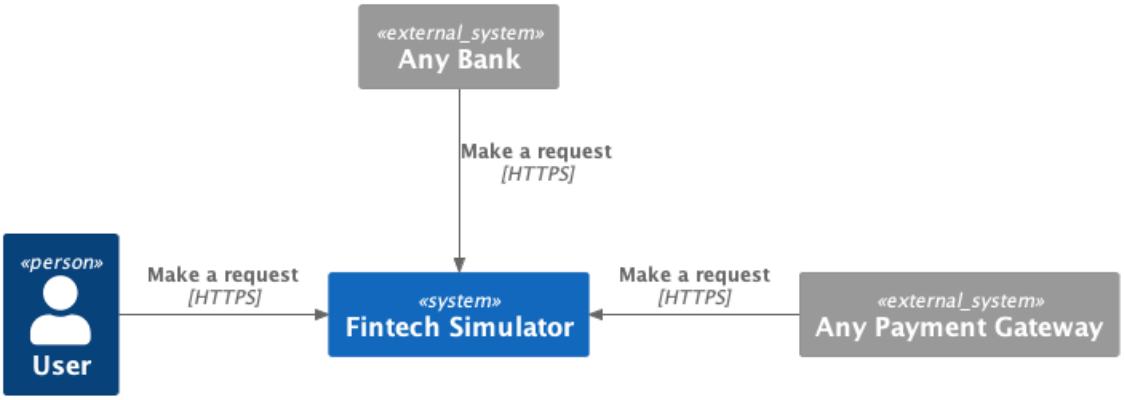


Ilustración 9 - Interacciones externas.

La ilustración 3 nos ayuda a entender el grado de exposición de este tipo de soluciones y esto nos ayuda a proponer otro tipo de uso de sistemas de limitación de tasa, sobretodo a payments-service que tiene la capacidad de recibir llamadas de cualquier banco para efectuar transacciones entre los mismos.

Arquitectura del Escenario

En la arquitectura general vamos a proponer el uso de DRL en la comunicación entre algunos componentes a modo de proteger los microservicios más críticos de la solución

Primeramente vamos a proteger el payments-service porque es un servicio expuesto a la red pública, que puede recibir tráfico de usuarios, de integraciones tanto de otros bancos, como de otros servicios financieros y por esta razón puede estar sujeto a ataques contra usuarios malintencionados.

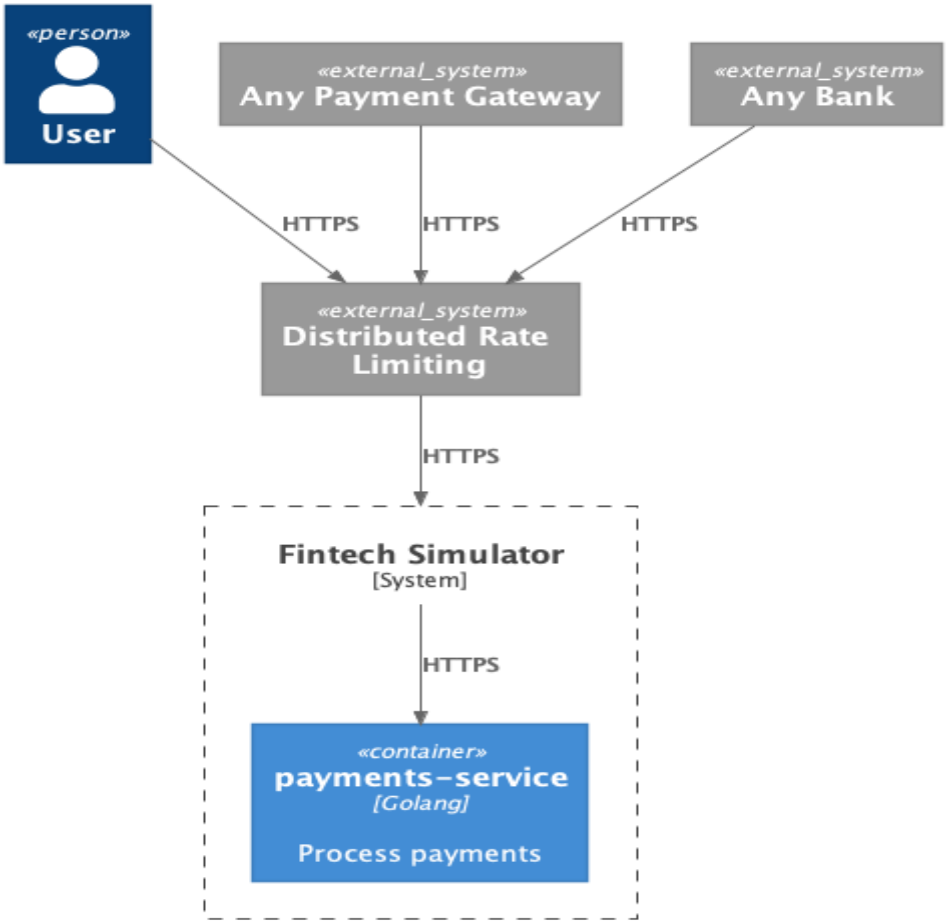


Ilustración 10 - Protección de payments-service.

Es importante aclarar que cada usuario interactúa con esta API a través de una sesión que debe ser otorgada anteriormente por el sessions-service. También las integraciones deben interactuar con esta API con el uso del protocolo OAuth 2.0.

Sumado a esto tenemos el sessions-service que provee sesiones a los usuarios y esto brinda una potencial puerta para atacantes que realizan fuerza bruta para enumerar cuentas, phishing e ingeniería social.

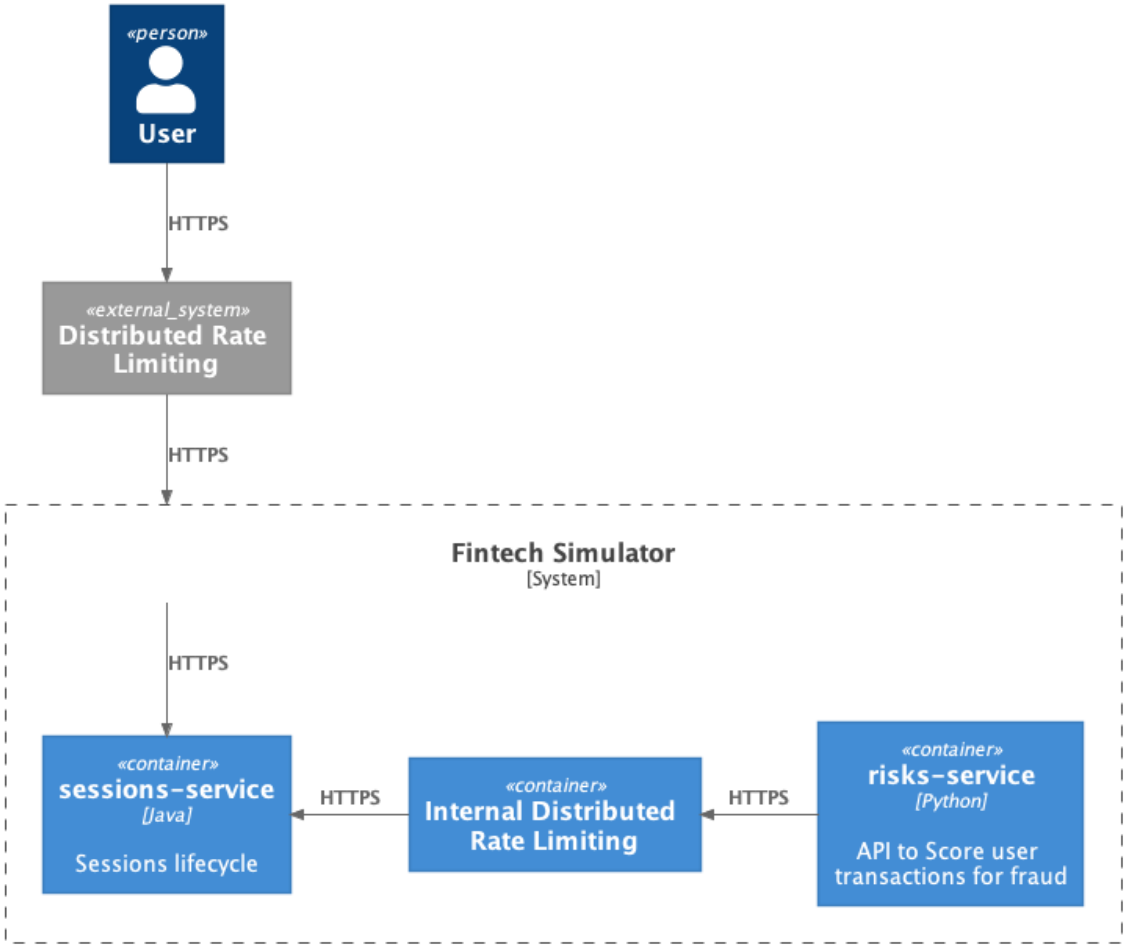


Ilustración 11 - Protección de sessions-service.

Además necesitamos de este servicio para analizar el riesgo de las sesiones presentadas, como por ejemplo, si la sesión existe en un dispositivo vulnerable, que se

presenta en una ubicación no habitual al usuario o incluso si se utilizó un segundo factor débil en casos de ataques masivos de ingeniería social.

Sin embargo este servicio de base debe ser protegido para garantizar que los usuarios puedan utilizar la plataforma, también tenemos que protegerlo de servicios internos por el mal uso de la funcionalidad de las sesiones.

Por fin el accounts-service también es un servicio crítico porque maneja el dinero actual del usuario. Por esto, algunas URLs no deben ser utilizadas desde servicios externos y también debemos limitar su uso interno.

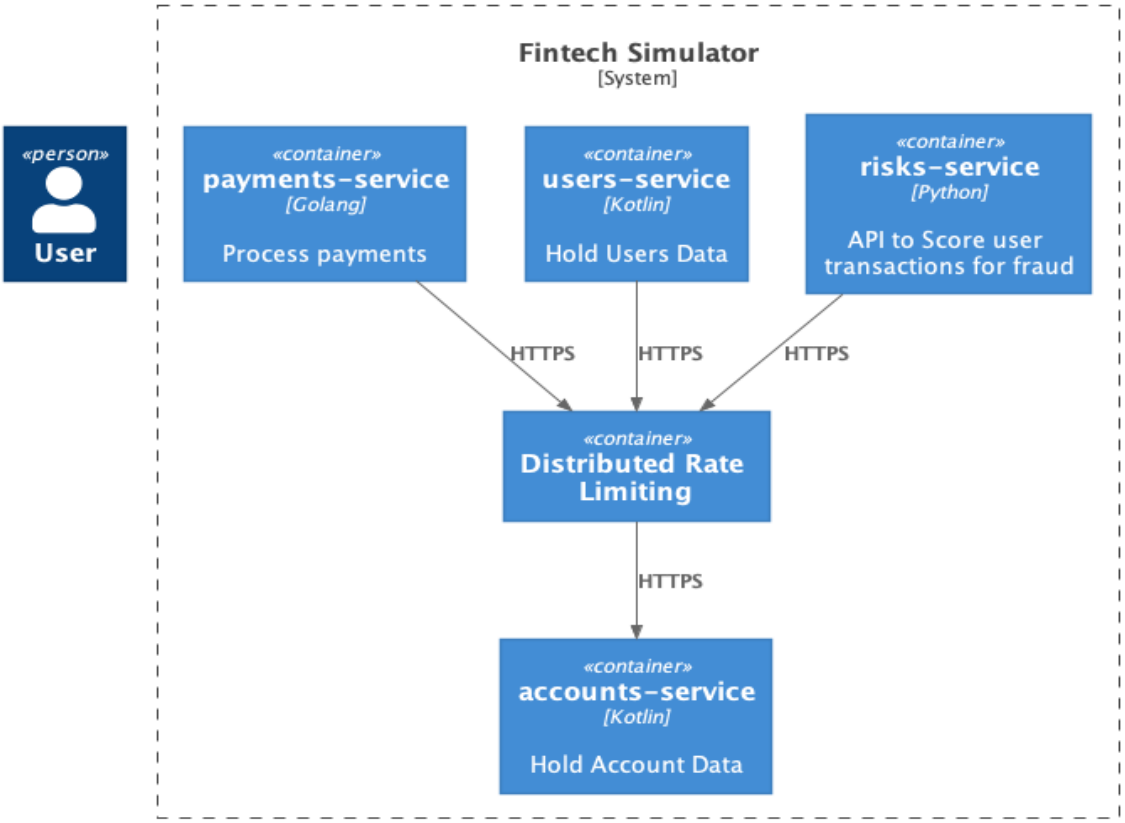


Ilustración 12 - Limitación del uso de accounts-service.

Como se puede ver los servicios payments-service, users-service, risks,service dependen de accounts-service para proveer algunas funcionalidades descritas en Arquitectura con componentes de la Nube.

Por la naturaleza del negocio, queremos que la funcionalidad del uso de dinero sea la menos afectada en casos extremos, como ataques cibernéticos, sobrecarga del sistema o incluso, defectos de implementación.

Sin embargo este escenario necesita experimentos en los cuales vamos a generar estímulos en la solución propuesta y comparar resultados entre cada experimento. Con fines de simular lo más parecido al mundo real, cada experimento va a representar una funcionalidad del sistema.

Arquitectura con componentes en la Nube

Abajo describiremos cada funcionalidad y cómo se impacta en los componentes del sistema.

Funcionalidad	**Descripción**	**Componentes necesarios**
Campañas de marketing	Ofrecer promociones a usuarios	movements-service
Procesamiento de Pagos	Efectuar transacciones	payments-service, sessions-service
Análisis de riesgo de crédito	Analizar usuarios potenciales para ofrecer créditos	payments-service, sessions-service, accounts-service, users-services

Tabla 4 - Descripción de funcionalidades.

Infraestructura con componentes AWS

Para proveer tal arquitectura necesitamos de componentes en los que podamos desplegar microservicios y exponerlos en la internet. Para esto vamos a utilizar AWS Fargate y AWS API Gateway.

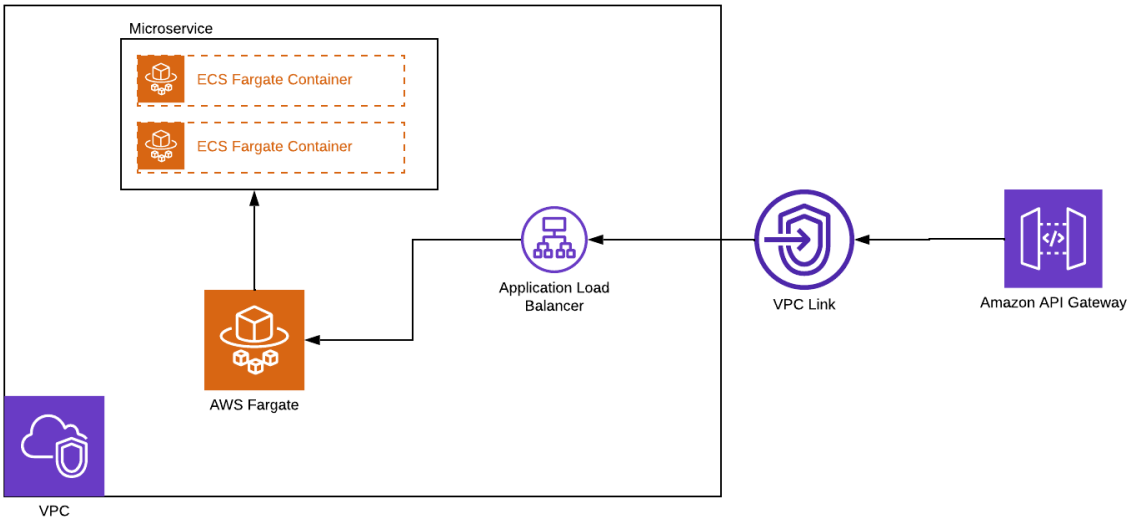


Ilustración 13 - Arquitectura de componentes AWS.

AWS Fargate nos ayuda a aprovisionar cada instancia de los microservicios y en conjunto con el ALB podemos dividir el tráfico entre las instancias, así proveyendo escalabilidad a nuestra solución. La VPC nos ayuda a que solo exista tráfico autorizado desde la API Gateway hasta nuestros servicios, por fin el AWS API Gateway expone nuestros enlaces para el internet.

Es importante aclarar que AWS API Gateway posee una característica llamada planos de uso con claves, descrita en (Creating and Using Usage Plans with API Keys - Amazon API Gateway, 2023) en la cual es posible limitar el uso de enlaces de acuerdo

con cada cliente, proveyendo palabras secretas (tokens), esto en conjunto de autorizadores, en (Use API Gateway Lambda Authorizers - Amazon API Gateway, 2023), que simplifican nuestra autenticación y autorización. Es recomendable utilizar estas dos características en conjunto dado que el DRL no sirve para la autenticación.

Infraestructura con componentes GCP

En Google Cloud es necesario utilizar un producto llamado Google Cloud API Gateway (API Gateway, 2023) para exponer nuestros enlaces, este producto posee una opción de crear cuotas configurando las especificaciones de OPEN API utilizando la opción x-google-management en la que referencia la configuración de secretos que pueden ser utilizados por los distintos microservicios y clientes externos de nuestro escenario. Además la API Gateway va a estar conectada a Google Cloud Functions (Cloud Functions, 2023) que es un servicio Serverless que gestiona la infraestructura de nuestros microservicios así como el escalado.

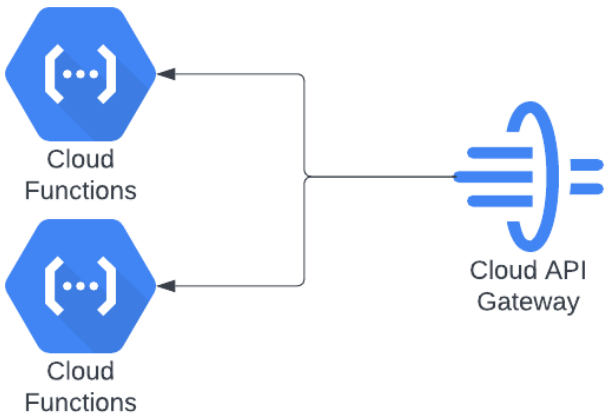


Ilustración 14 - Arquitectura de componentes GCP.

Infraestructura con componentes AZURE

En la API Management de Azure (API Management – Manage APIs | Microsoft Azure, 2023) podemos exponer nuestro micro servicios de forma segura y totalmente aprovisionada por Azure, sumado a que el producto tiene una forma de gestionar y limitar el uso de los recursos con la API Management Policies (dlepow, 2023) en el cual podemos configurar políticas de limitación de tasa para distintos consumidores y aplicaciones.

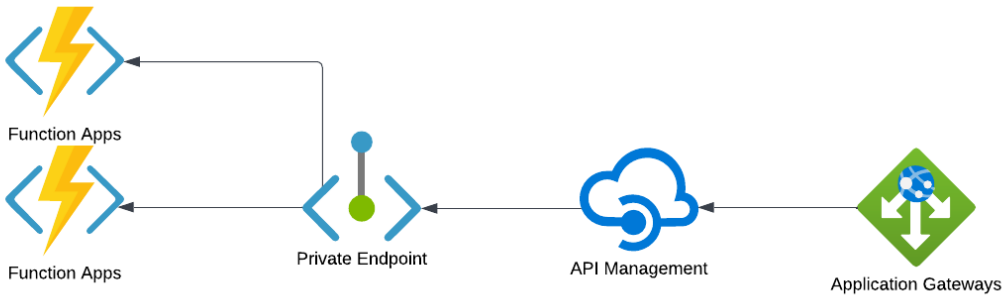


Ilustración 15 - Arquitectura de componentes Azure.

Como podemos ver en la imagen anterior también vamos a utilizar Azure Functions (ggailey777, 2023) Apps en la cual Azure coordina el escalado y la gestión de la infraestructura de nuestros micro servicios.

Método de Comparación

El método de comparación se da en cada escenario descrito en la sección anterior utilizando una tecnología base. Sin embargo vamos a estar estresando al sistema para

entender el comportamiento generado en cada escenario, podemos describir esto en la figura abajo.

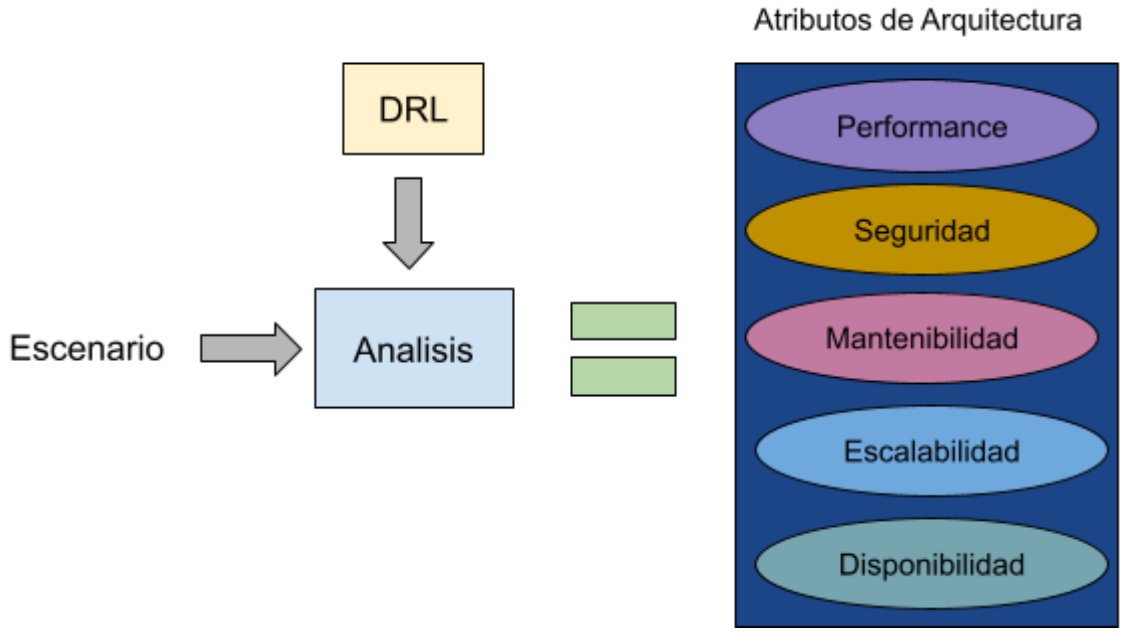


Ilustración 16 - Diagrama del método de comparación.

En este trabajo el análisis de cada tecnología de base nos ayuda a identificar la mejor propuesta con base en cada escenario que se propone.

Para cada atributo necesitamos definir cómo medirlo para que en el próximo capítulo sea posible realizar las comparaciones.

El performance vamos a medirlo en milisegundos sobre el promedio de cada llamada así como los tiempos máximos y percentiles 99. Esto nos ayuda a entender la estabilidad de la solución y como esta puede variar en casos extremos.

En seguida la seguridad es muy importante ya que el principal foco de un sistema de limitación de tasa es mejorar la seguridad y con esto debemos apoyarnos en buenas prácticas de desarrollo de microservicios como es descrito en (OWASP API Security -

Top 10 | OWASP, 2019). Una de las prácticas principales se recomienda que en el diseño de la arquitectura se tenga en cuenta que algunos recursos deben tener límites de uso, porque en escenarios reales no deberían escalar sin controles o de forma infinita de acuerdo con (OWASP API Security Top 10, 2022). Tal práctica no solo ayuda a controlar los costos, sino también a evitar que hackers hagan el uso indebido de la solución, como poder enumerar usuarios, invalidar servicios de análisis de riesgo, DDoS o incluso atacar a empresas para robar datos de los usuarios. Por esto su evaluación se mide por que tan fácil sea hacer que la herramienta cumpla con los requerimientos de OWASP como rompimiento de autorización a nivel de recursos, rompimiento de la autenticación del usuario, exposición excesiva de datos, rompimiento de autorización a nivel de funciones, registros de aplicac

iones insuficientes y monitoreo en conjunto con nuestro principal foco que es el cómo gestionar y evitar Lack of Resources & Rate Limiting .

En el ciclo de desarrollo de una solución se entiende que es deseado que esta quede mucho tiempo de forma productiva y que se pueda hacer los cambios necesarios. Por esto la mantenibilidad es clave para definir una arquitectura empresarial y que los conocimientos de herramientas sean compartidos entre equipos. Por lo cual vamos a evaluar este atributo por tres características, la comunidad, la documentación y si la tecnología base es amigable al momento de hacer cambios.

Sin embargo, no podemos olvidar que la escalabilidad es una de las principales razones del porque las empresas eligen una solución en la nube. Su evaluación se da en escala de forma vertical la respectiva solución y si en grandes cambios hay

necesidad de una posible migración que puede producir bajas generales en la solución.

Por fin tenemos que hablar de disponibilidad, que sería básicamente que hacer cuando una de las tecnologías base elegidas empieza a fallar, como definir un plan de contingencia o incluso si esto es algo que se pueda delegar (tercerizar) al servicio.

Conclusión del Capítulo

En este Capítulo describimos los escenarios y cómo será aplicada la arquitectura de acuerdo con cada tecnología base. El escenario será aplicado a nuestro modelo al que vamos a poder hacer comparaciones en el próximo capítulo.

Resultados

Experimentos

Los experimentos fueron realizados de acuerdo con los escenarios planteados en la sección de Escenario.

Para esta cuestión, utilizando Locust, haremos pruebas de carga en 3 distintos enlaces http:

- POST /accounts/{accountId}/payments (script llamado scenario_receive_payments)
- POST /accounts/{accountId}/payments/{paymentId}/risks (script llamado scenario_batch_risk_analisis)
- GET /users/{userId}/accounts (script llamado scenario_marketing_campaigns)

En cada endpoint vamos a tener los siguientes experimentos:

- Aproximadamente 100 RPS (Peticiones por segundo) por aproximadamente 20 minutos

Antes que nada, analizaremos la solución propuesta utilizando herramientas de AWS.

Script: scenario_marketing_campaigns.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	marketing-campaign	163322	0	566	502	1154	2	94.6	0.0
	Aggregated	**163322**	**0**	**566**	**502**	**1154**	**2**	**94.6**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	marketing-campaign	570	580	590	600	610	610	640	1200
	Aggregated	**570**	**580**	**590**	**600**	**610**	**610**	**640**	**1200**

Tabla 5 - Peticiones de pruebas en AWS para el escenario de campañas de marketing.



Ilustración 17 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de campañas de marketing.

Script: scenario_receive_payments.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	receive-payments	147777	10289	1665	404	3337	0	85.8	6.0
	Aggregated	**147777**	**10289**	**1665**	**404**	**3337**	**0**	**85.8**	**6.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	receive-payments	1700	1900	2200	2400	2700	2800	2900	3300
	Aggregated	**1700**	**1900**	**2200**	**2400**	**2700**	**2800**	**2900**	**3300**

Failures Statistics

Method	Name	Error	Occurrences
POST	receive-payments	CatchResponseError(502)	4332
POST	receive-payments	CatchResponseError(429)	5957

Tabla 6 - Peticiones de pruebas en AWS para el escenario de procesamiento de pagos.



Ilustración 18 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de procesamiento de pagos.

Script: scenario_batch_risk_analisis.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	risk-analisis	152494	18333	1267	504	2318	0	88.7	10.7
	Aggregated	**152494**	**18333**	**1267**	**504**	**2318**	**0**	**88.7**	**10.7**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	risk-analisis	1300	1400	1600	1700	1900	1900	2000	2300
	Aggregated	**1300**	**1400**	**1600**	**1700**	**1900**	**1900**	**2000**	**2300**

Failures Statistics

Method	Name	Error	Occurrences
POST	risk-analisis	CatchResponseError(206)	18333

Tabla 7 - Peticiones de pruebas en AWS para el escenario de análisis de riesgo para ofrecer créditos.



Ilustración 19 - Gráfico de tiempos de respuestas y errores en AWS para el escenario de análisis de riesgo para ofrecer créditos.

En segundo lugar, analizamos la solución propuesta utilizando herramientas de GCP.

Script: scenario_marketing_campaigns.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	marketing-campaign	137854	0	563	501	1146	2	94.7	0.0
	Aggregated	**137854**	**0**	**563**	**501**	**1146**	**2**	**94.7**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	marketing-campaign	570	570	580	600	600	610	620	1100
	Aggregated	**570**	**570**	**580**	**600**	**600**	**610**	**620**	**1100**

Tabla 8 - Peticiones de pruebas en GCP para el escenario de campañas de marketing.



Ilustración 20 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de campañas de marketing.

Script: scenario_receive_payments.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	receive-payments	120635	8399	2010	502	3646	0	83.2	5.8
	Aggregated	**120635**	**8399**	**2010**	**502**	**3646**	**0**	**83.2**	**5.8**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	receive-payments	2000	2300	2600	2900	3200	3400	3500	3600
	Aggregated	**2000**	**2300**	**2600**	**2900**	**3200**	**3400**	**3500**	**3600**

Failures Statistics

Method	Name	Error	Occurrences
POST	receive-payments	CatchResponseError(502)	3589
POST	receive-payments	CatchResponseError(429)	4810

Tabla 9 - Peticiones de pruebas en GCP para el escenario de procesamiento de pagos.



Ilustración 21 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de procesamiento de pagos.

Script: scenario_batch_risk_analisis.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	risk-analisis	125280	14838	1614	703	2673	0	86.1	10.2
	Aggregated	**125280**	**14838**	**1614**	**703**	**2673**	**0**	**86.1**	**10.2**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	risk-analisis	1600	1800	2000	2200	2300	2400	2500	2700
	Aggregated	**1600**	**1800**	**2000**	**2200**	**2300**	**2400**	**2500**	**2700**

Failures Statistics

Method	Name	Error	Occurrences
POST	risk-analisis	CatchResponseError(206)	14838

Tabla 10 - Peticiones de pruebas en GCP para el escenario de análisis de riesgo para ofrecer créditos.



Ilustración 22 - Gráfico de tiempos de respuestas y errores en GCP para el escenario de análisis de riesgo para ofrecer créditos.

Por último analizaremos la solución propuesta utilizando herramientas de AZURE.

Script: scenario_marketing_campaigns.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	marketing-campaign	118034	0	563	502	754	2	94.6	0.0
	Aggregated	**118034**	**0**	**563**	**502**	**754**	**2**	**94.6**	**0.0**

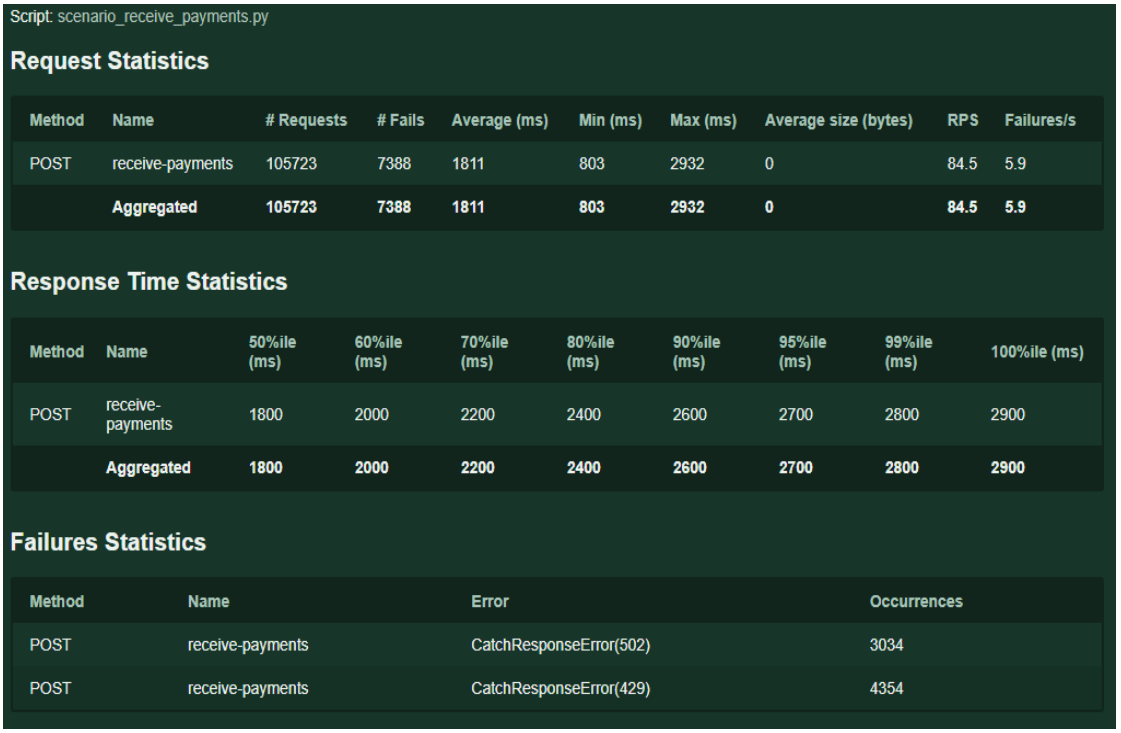
Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	marketing-campaign	570	570	580	600	600	610	620	750
	Aggregated	**570**	**570**	**580**	**600**	**600**	**610**	**620**	**750**

Tabla 11 - Peticiones de pruebas en AZURE para el escenario de campañas de marketing.



Ilustración 23 - Gráfico de tiempos de respuestas y errores en AZURE para el escenario de campañas de marketing.



Script: scenario_receive_payments.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	receive-payments	105723	7388	1811	803	2932	0	84.5	5.9
	Aggregated	**105723**	**7388**	**1811**	**803**	**2932**	**0**	**84.5**	**5.9**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	receive-payments	1800	2000	2200	2400	2600	2700	2800	2900
	Aggregated	**1800**	**2000**	**2200**	**2400**	**2600**	**2700**	**2800**	**2900**

Failures Statistics

Method	Name	Error	Occurrences
POST	receive-payments	CatchResponseError(502)	3034
POST	receive-payments	CatchResponseError(429)	4354

Tabla 12 - Peticiones de pruebas en AZURE para el escenario de procesamiento de pagos.



Ilustración 24 - Gráfico de tiempos de respuestas y de errores en AZURE para el escenario de procesamiento de pagos.

Script: scenario_batch_risk_analisis.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	risk-analisis	110737	13133	1312	802	2086	0	88.3	10.5
	Aggregated	**110737**	**13133**	**1312**	**802**	**2086**	**0**	**88.3**	**10.5**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	risk-analisis	1300	1400	1500	1600	1700	1800	1800	2100
	Aggregated	**1300**	**1400**	**1500**	**1600**	**1700**	**1800**	**1800**	**2100**

Failures Statistics

Method	Name	Error	Occurrences
POST	risk-analisis	CatchResponseError(206)	13133

Tabla 13 - Peticiones de pruebas en AZURE para el escenario de análisis de riesgos para ofrecer créditos.



Ilustración 25 - Gráfico de tiempos de respuestas y errores en AZURE para el escenario de análisis de riesgos para ofrecer créditos.

Comparación de Resultados

Es importante resaltar que los escenarios de procesamiento de pagos y campañas de marketing poseen errores independientes del proveedor de la nube que estamos utilizando. Esto se da porque los sistemas distribuidos de limitación de tasa están configurados para soportar un tráfico menor de lo que fue probado, entonces estos errores son dados por HTTP 429, que representan el uso excesivo del recurso a través de un actor. También podemos ver que incluso con tales errores la plataforma sigue funcionando en algunas condiciones.

Podemos hacer la siguiente pregunta, ¿por qué el escenario de análisis de crédito no posee errores ya que este utiliza datos de todos los otros microservicios? Esto se da porque el análisis para ofrecer créditos no es crítica, entonces es posible hacerlo con menos datos o incluso que algunas llamadas a otros servicios fallen, lo que va a generar menos ofertas, pero no impacta de modo

general el funcionamiento de tal servicio. Por otro lado, las campañas de marketing, por más que no sean una característica crítica no pueden funcionar sin datos básicos del usuario.

Cuando hablamos de seguridad, las 3 opciones deben ser utilizadas en conjunto con un mecanismo de autenticación.

Al hablar de performance, AWS posee una pequeña ventaja en comparación con los otros dos, aunque no existan muchas diferencias. Además, podemos ver que GCP tiene una excepción en tiempos máximos de respuesta. Pero vale la pena aclarar que el

performance depende de muchas otras variables como el internet, el tráfico, las políticas de DRL y la autenticación.

Por fin, como todas las soluciones de API Gateway son gestionadas por el proveedor, esto ya nos garantiza alta disponibilidad y escalabilidad. Sin embargo, debemos tener cuidado con la arquitectura interna, como base de datos y cuellos de botella que pueden ocurrir, no por el uso de un DRL pero sí por características particulares de la solución.

En resumen podemos ver todas las consideraciones en la tabla de abajo:

Escenario	Performance	Seguridad	Mantenibilidad	Escalabilidad	Disponibilidad
Marketing en AWS	No Aprobado	Aprobado	No Aprobado	Aprobado	Aprobado
Procesamiento de Pagos en AWS	Aprobado	Aprobado	No Aprobado	Aprobado	Aprobado
Análisis de riesgo en AWS	Aprobado	Aprobado	No Aprobado	Aprobado	Aprobado
Marketing en GCP	No Aprobado	Aprobado	Aprobado	Aprobado	Aprobado
Procesamiento de Pagos en GCP	Aprobado	Aprobado	Aprobado	Aprobado	Aprobado
Análisis de riesgo en GCP	Aprobado	Aprobado	Aprobado	Aprobado	Aprobado
Marketing en AZURE	No Aprobado	Aprobado	Aprobado	Aprobado	Aprobado
Procesamiento de Pagos en AZURE	Aprobado	Aprobado	Aprobado	Aprobado	Aprobado
Análisis de riesgo en AZURE	Aprobado	Aprobado	Aprobado	Aprobado	Aprobado

Tabla 14 - Comparación de escenarios.

Conclusiones

Los principales proveedores de la nube poseen soluciones muy consistentes para este tipo de problema, sin embargo, al el momento de tomar una decisión en que la tecnología a ser utilizada es necesario discutir algunos elementos teniéndolos en consideración:

- Costo
- Performance
- Cuál es la estrategia de la arquitectura empresarial
- Migración a otras tecnologías

En general AWS tiene un costo más elevado, no obstante para algunas regiones del globo es considerada la más popular, sin embargo en las pruebas vemos una ventaja en el uso de AWS, debemos evaluar otras características del problema, como región en que se hospedará la infraestructura, costo, redundancia, como disminuir errores y potencializar la disponibilidad sin perder componentes del producto. Estas características están fuera del alcance de este trabajo. Por eso aclaramos que la decisión de elegir una solución o otra, va a depender considerablemente de cómo la empresa desea enfatizar un atributo de arquitectura determinado y el mismo debe ser discutido entre arquitectos y CTO's.

Por último, pero no menos importante, en conjunto de la estrategia, se debe considerar como la construcción de una solución depende de un proveedor. Muchas veces se eligen herramientas de código abierto para facilitar cambios futuros o se elige utilizar un proveedor para ahorrar tiempo, complejidad de desarrollo y despliegue. Así mismo

hay que tener en mente como pueden ocurrir futuros cambios, costos o migraciones es algo que siempre debemos tener en cuenta cuando estamos analizando el uso de tales herramientas, porque a veces está también la necesidad de cambiar los clientes . Esto se da porque cada proveedor tiene una implementación del uso de DRL diferente , por ejemplo AWS exige un encabezamiento x-api-key, en GCP podemos utilizar un query string key para el envío del token que represente al cliente y con AZURE es posible configurarlo con el header que se desee.

Discusión General del Proyecto

Discusión

Es importante enfatizar que la utilización del DRL ayuda a gestionar el uso de recursos, así como un mecanismo de protección, pero debe ser utilizado en conjunto con políticas de autenticación y autorización.

Así también, el uso de tal tecnología puede ayudar a las empresas a tener más conciencia sobre la exposición y el consumo de sus respectivos recursos y servicios críticos.

Conclusión Final

Los proveedores de la nube poseen excelentes soluciones de sistemas distribuidos de limitación de tasa, lo que se suma al momento de elegir cuál solución va ser la adecuada para utilizarse y cuál será la estrategia de la empresa con relación a la performance, costos y agilidad en la implementación.

También vale la pena aclarar que migrar de una solución a otra no es tan simple ya que por la falta de estándar en el uso de tokens, puede dificultar mantener los contratos de API's con los consumidores.

Pero es importante destacar que en algunas situaciones económicas tal abordaje debe ser una preocupación no solo cuando hablamos de mejorar la seguridad de las plataformas, sino también en cómo gestionar los riesgos, gestionar el uso indebido de

los recursos que por consecuencia pueden generar impactos negativos en gastos con infraestructura.

Por último, el trabajo ejemplifica de forma pragmática el cómo hacer la evaluación de una arquitectura, teniendo en cuenta las preocupaciones abordadas en esta tesis y con esto poder diseñar y mejorar las aplicaciones, visando mejor seguridad, performance, mantenibilidad y crecimiento saludable evitando gastos innecesarios.

Trabajo Futuro

Todavía hay mucho que investigar en este medio, además cuando tenemos escenarios de empresas que poseen modelos híbridos, o sea, que utilizan más de un proveedor de la nube, o incluso la combinación de proveedor de la nube e instalaciones físicas, y tal arquitectura hace que sea más difícil adoptar tal tecnología de modo que esté centralizada y sea fácil de gestionar.

Imaginemos a este tipo de arquitectura y que la empresa adopte un tipo de DRL para lo que es on-premises y otro para los proveedores de la nube. Esto haría que los costos de manutención sean muy elevados, dado que la necesidad de equipos y los desarrolladores tienen que aprender más de una tecnología para poder garantizar el buen uso de sus servicios. Así como para los directores y para C-level sería difícil tener un panorama general del uso de tales herramientas.

De esta forma se cuestiona, si vale la pena invertir en soluciones privadas o de código abierto que puedan tener tales características para cada tipo de empresa. Al mismo

tiempo invertir en una arquitectura que soporte un modelo híbrido, de forma centralizada, con alta disponibilidad, escalable, segura y auditable.

BIBLIOGRAFÍA

Admin. (2021). Zero Cost Cloud Migration – Thinking Beyond CapEx vs. OpEx. Suneratech. https://www.suneratech.com/blog/zero-cost-cloud-migration-thinking-beyond-capex-vs-opex/

Amazon API Gateway. (2019). Amazon Web Services, Inc. https://aws.amazon.com/api-gateway/

API Gateway. (2023). Google Cloud. https://cloud.google.com/api-gateway

API Management – Manage APIs | Microsoft Azure. (2023). Azure.microsoft.com. https://azure.microsoft.com/en-us/products/api-management/

Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019). Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). https://doi.org/10.1109/icsa-c.2019.00041.

Building well-architected serverless applications: Regulating inbound request rates – part 1 | Amazon Web Services. (2021, July 27). Amazon Web Services. https://aws.amazon.com/pt/blogs/compute/building-well-architected-serverless-applications-regulating-inbound-request-rates-part-1/

Cloud Functions. (2023). Google Cloud. https://cloud.google.com/functions

Cohen, J. (2021, December 29). 4 Companies Control 67% of the World's Cloud Infrastructure. PCMAG. https://www.pcmag.com/news/four-companies-control-67-of-the-worlds-cloud-infrastructure

Creating and using usage plans with API keys - Amazon API Gateway. (2023). Docs.aws.amazon.com. Retrieved January 22, 2023, from https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-usage-plans.html

Datadome, Ago 2020. What is API Rate Limiting and How to Implement It. Disponible en: https://datadome.co/bot-management-protection/what-is-api-rate-limiting/ (Nov. 2022).

Dickel, H., Podolskiy, V., & Gerndt, M. (2019, November 1). Evaluation of Autoscaling Metrics for (stateful) IoT Gateways. IEEE Xplore. https://doi.org/10.1109/SOCA.2019.00011

dlepow. (2023). Azure API Management policy reference. Learn.microsoft.com. Retrieved January 23, 2023, from https://learn.microsoft.com/en-us/azure/api-management/api-management-policies

Doyle, J., Shorten, R., & O'Mahony, D. (2010). An experimental evaluation of distributed rate limiting for cloud computing applications. Proceedings of the

6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. https://doi.org/10.1145/1872007.1872042

Doyle, J., & O'mahony, D. (2012). Load Balancing and Rate Limiting Based Algorithms for Improving Cloud Computing Performance. https://scss.tcd.ie/publications/theses/phd/TCD-SCSS-PHD-2012-14.pdf

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. Present and Ulterior Software Engineering, 195–216. https://doi.org/10.1007/978-3-319-67425-4_12

Düllmann, T. F., & van Hoorn, A. (2017). Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. https://doi.org/10.1145/3053600.3053627

Geer, D. (1998). Risk Management is Where the Money is. Risk Management Is Where the Money Is. https://cseweb.ucsd.edu/~goguen/courses/275f00/geer.html

Gerardus B. (2018). API Gateways Second Edition. 5starcooks.

ggailey777. (2023). Azure Functions Overview. Learn.microsoft.com. https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview

Giansanti, D. (2021, January 11). Cybersecurity and the Digital-Health: The Challenge of This Millennium. Healthcare, 9(1), 62. https://doi.org/10.3390/healthcare9010062

Henriksson, A., & Bennhage, S. (2021). Trading performance for precision in a CRDT-based rate-limiting system. Odr.chalmers.se. https://odr.chalmers.se/handle/20.500.12380/304211

Hwang, J. J., Wu, K. C., & Liu, D. R. (2000, March). Access control with role attribute certificates. Computer Standards & Interfaces, 22(1), 43–53. https://doi.org/10.1016/s0920-5489(99)00029-x

Krafzig, D., Banke, K., & Slama, D. (2005). Enterprise SOA: Service-oriented Architecture Best Practices. In Google Books. Prentice Hall Professional. https://books.google.com.br/books?hl=pt-BR&lr=&id=R7oGhITYUuUC&oi=fnd&pg=PR17&dq=Krafzig

Kumar, S., & Vasthimal, D. K. (2020). Cardinality Based Rate Limiting System for Time-Series Data. Lecture Notes in Computer Science, 250–260. https://doi.org/10.1007/978-3-030-59635-4_18

Lewis, J., & Fowler, M. (2014). Microservices. Martinfowler.com. https://martinfowler.com/articles/microservices.html

Locust - A modern load testing framework. (2022). Locust.io. https://locust.io/

Microservices Pattern: API gateway pattern. (2019). Microservices.io. https://microservices.io/patterns/apigateway.html

OWASP API Security Top 10. (2022, May 12). GitHub. https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa4-lack-of-resources-and-rate-limiting.md

Oliveira, S. B. de, Balloni, A. J., Oliveira, F. N. B. de, & Toda, F. A. (2012). Information and Service-Oriented Architecture & Web Services: Enabling Integration and Organizational Agility. Procedia Technology, 5, 141–151. https://doi.org/10.1016/j.protcy.2012.09.016

OWASP API Security - Top 10 | OWASP. (2019). Owasp.org. https://owasp.org/www-project-api-security/

Qian, L., Luo, Z., Yujian, D., & Guo, L. (2009). Cloud Computing: An Overview. In *Lecture Notes in Computer Science* (pp. 626–631). Springer Science+Business Media. https://doi.org/10.1007/978-3-642-10665-1_63

¿Qué es cloud computing? | Google Cloud. (2023). Google Cloud. https://cloud.google.com/learn/what-is-cloud-computing?hl=es

Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., & Snoeren, A. C. (2007). Cloud control with distributed rate limiting. ACM SIGCOMM Computer Communication Review, 37(4), 337–348. https://doi.org/10.1145/1282427.1282419

Rate-limiting strategies and techniques | Cloud Architecture Center. (2019). Google Cloud. https://cloud.google.com/architecture/rate-limiting-strategies-techniques

Richardson, L., & Ruby, S. (2008). RESTful web services. " O'Reilly Media, Inc.".

Stanojevic, R., & Shorten, R. (2009, July 1). Generalized distributed rate limiting. IEEE Xplore. https://doi.org/10.1109/IWQoS.2009.5201389

Stanojevic, R., & Shorten, R. (2009, June 1). Load Balancing vs. Distributed Rate Limiting: An Unifying Framework for Cloud Control. IEEE Xplore. https://doi.org/10.1109/ICC.2009.5199141

Thones, J. (2015). Microservices. IEEE Software, 32(1), 116–116. https://doi.org/10.1109/ms.2015.11

Top 10 cloud infrastructure providers. (2022, April 5). Datacentremagazine.com. https://datacentremagazine.com/it/top-10-cloud-infrastructure-providers

Use API Gateway Lambda authorizers - Amazon API Gateway. (2023). Docs.aws.amazon.com. https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html

What is a distributed denial-of-service (ddos) attack? - cloudflare. (2023). Retrieved May 7, 2023, from https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

Yang, X., Wetherall, D., & Anderson, T. (2005). A DoS-limiting network architecture. ACM SIGCOMM Computer Communication Review, 35(4), 241–252. https://doi.org/10.1145/1090191.1080120

Yoo, Y., Yang, G., Kang, M., & Yoo, C. (2020). Adaptive Control Channel Traffic Shaping for Virtualized SDN in Clouds. 2020 IEEE 13th International Conference on Cloud Computing (CLOUD). https://doi.org/10.1109/cloud49709.2020.00013

Zhang, M. (2022, June 28). Top 10 Cloud Service Providers Globally in 2022. Dgtl Infra. https://dgtlinfra.com/top-10-cloud-service-providers-2022/

APÉNDICES

Script para infraestructura en AWS

```
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = "~> 4.16"
  }
 }

 backend "s3" {
  bucket = "fintech-simulator"
  key    = "terraform/state/aws"
  region = "us-east-1"
 }
```

```
  required_version = ">= 1.2.0"
}

provider "aws" {
  region  = "${var.region}"
}

module "payments_service_module" {
    source = "./microservices"
    service_name = "payments"
    vpc_id = aws_vpc.aws-vpc.id
    subnets_id = aws_subnet.private.*.id
    public_subnets_id = aws_subnet.public.*.id
    environment_definitions = jsonencode(
      [
        {
          name="PORT"
          value="8080"
        },
        {
          name="API_GATEWAY_KEY"
          value="Vz1WEaHhcU9j0nC7nqGSI84JFe3EhxxiaVVpi5Qm"
        },
        {
          name="RISK_URL"
          value="https://spt90yhwv6.execute-api.us-west-2.amazonaws.com/production/"
        },
```

```
      {
       name="ACCOUNTS_URL"
       value="https://spt90yhwv6.execute-api.us-west-2.amazonaws.com/production/"
      }
    ]
  )
}

module "accounts_service_module" {
  source = "./microservices"
  service_name = "accounts"
  vpc_id = aws_vpc.aws-vpc.id
  subnets_id = aws_subnet.private.*.id
  public_subnets_id = aws_subnet.public.*.id
}

module "movements_service_module" {
  source = "./microservices"
  service_name = "movements"
  vpc_id = aws_vpc.aws-vpc.id
  subnets_id = aws_subnet.private.*.id
  public_subnets_id = aws_subnet.public.*.id
}

module "users_service_module" {
  source = "./microservices"
  service_name = "users"
  vpc_id = aws_vpc.aws-vpc.id
```

```
    subnets_id = aws_subnet.private.*.id
    public_subnets_id = aws_subnet.public.*.id
    environment_definitions = jsonencode(
      [
        {
          name="API_GATEWAY_KEY"
          value="VALUE"
        },
        {
          name="ACCOUNTS_URL"
          value="DNS"
        },
        {
          name="SESSION_SECRET"
          value="ABC"
        },
        {
          name="SALT_SECRET"
          value="2"
        }
      ]
    )
}

module "sessions_service_module" {
    source = "./microservices"
    service_name = "sessions"
    vpc_id = aws_vpc.aws-vpc.id
```

```
    subnets_id = aws_subnet.private.*.id
    public_subnets_id = aws_subnet.public.*.id
    environment_definitions = jsonencode(
      [
        {
          name="API_GATEWAY_KEY"
          value="VALUE"
        },
        {
          name="USERS_URL"
          value="<DNS>"
        }
      ]
    )
}

module "risks_service_module" {
    source = "./microservices"
    service_name = "risks"
    vpc_id = aws_vpc.aws-vpc.id
    subnets_id = aws_subnet.private.*.id
    public_subnets_id = aws_subnet.public.*.id
    container_port = "5000"
    environment_definitions = jsonencode(
      [
        {
          name="API_GATEWAY_KEY"
          value="VALUE"
```

```
        },
        {
         name="MOVEMENTS_URL"
         value="<DNS>"
        },
        {
         name="ACCOUNTS_URL"
         value="<DNS>"
        },
        {
         name="SESSIONS_URL"
         value="<DNS>"
        },
        {
         name="USERS_URL"
         value="<DNS>"
        }
      ]
    )
}
resource "aws_nat_gateway" "main" {
  count         = length(aws_subnet.public.*.id)
  allocation_id = element(aws_eip.nat.*.id, count.index)
  subnet_id     = element(aws_subnet.public.*.id, count.index)
  depends_on    = [aws_internet_gateway.aws-igw]
}

resource "aws_eip" "nat" {
```

```
  count = length(aws_subnet.public.*.id)
  vpc = true
}

resource "aws_route_table" "nat-gateway" {
  count                  = length(aws_nat_gateway.main)
  depends_on = [
    aws_nat_gateway.main
  ]

  vpc_id = aws_vpc.aws-vpc.id
}

resource "aws_route" "nat-gateway" {
  count                  = length(aws_nat_gateway.main)
  route_table_id         = element(aws_route_table.nat-gateway.*.id, count.index)
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = element(aws_nat_gateway.main.*.id, count.index)
}

resource "aws_route_table_association" "nat-gateway" {
  depends_on = [
    aws_route_table.nat-gateway
  ]

  count          = length(var.private_subnets)
  subnet_id      = element(aws_subnet.private.*.id, count.index)
  route_table_id = element(aws_route_table.nat-gateway.*.id, count.index)
```

```
}
resource "aws_vpc" "aws-vpc" {
  cidr_block = var.cidr
  enable_dns_hostnames = true
  enable_dns_support   = true
  tags = {
    Name        = "fintech-simulator-vpc"
    Environment = "production"
  }
}

resource "aws_internet_gateway" "aws-igw" {
  vpc_id = aws_vpc.aws-vpc.id
  tags = {
    Name        = "fintech-simulator-igw"
    Environment = "production"
  }

}

resource "aws_subnet" "private" {
  vpc_id            = aws_vpc.aws-vpc.id
  count             = length(var.private_subnets)
  cidr_block        = element(var.private_subnets, count.index)
  availability_zone = element(var.availability_zones, count.index)

  tags = {
    Name        = "fintech-simulator-private-subnet-${count.index + 1}"
```

```
    Environment = "production"
  }
}

resource "aws_subnet" "public" {
  vpc_id                  = aws_vpc.aws-vpc.id
  cidr_block              = element(var.public_subnets, count.index)
  availability_zone       = element(var.availability_zones, count.index)
  count                   = length(var.public_subnets)
  map_public_ip_on_launch = true

  tags = {
    Name        = "fintech-simulator-public-subnet-${count.index + 1}"
    Environment = "production"
  }
}

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.aws-vpc.id

  tags = {
    Name        = "fintech-simulator-routing-table-public"
    Environment = "production"
  }
}

resource "aws_route" "public" {
  route_table_id         = aws_route_table.public.id
```

```
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.aws-igw.id
}

resource "aws_route_table_association" "public" {
  count          = length(var.public_subnets)
  subnet_id      = element(aws_subnet.public.*.id, count.index)
  route_table_id = aws_route_table.public.id
}
variable "region" {
  type    = string
  default = "us-west-2"
}

variable "instance_type" {
  type    = string
  default = "t4g.nano"
}

variable "cidr" {
  description = "The CIDR block for the VPC."
  default     = "10.0.0.0/16"
}

variable "public_subnets" {
  description = "List of public subnets"
  default     = ["10.0.100.0/24", "10.0.101.0/24"]
}
```

```
variable "private_subnets" {
  description = "List of private subnets"
  default     = ["10.0.0.0/24", "10.0.1.0/24"]
}

variable "availability_zones" {
  default = ["us-west-2a", "us-west-2b"]
}
resource "aws_api_gateway_rest_api" "fintech_simulator" {
  body = jsonencode({
    openapi = "3.0.1"
    info = {
      title   = "example"
      version = "1.0"
    }
    securitySchemes = {
      ApiKeyAuth ={
        type = "apiKey"
        name = "x-api-key"
        in = "header"
      }
    }
    paths = {
      "/users" = {
        post = {
          security = {ApiKeyAuth= []}
          x-amazon-apigateway-api-key-source = "HEADER"
```

```
    x-amazon-apigateway-integration = {
      httpMethod           = "POST"
      payloadFormatVersion = "1.0"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["users"]
      type                 = "HTTP_PROXY"
      uri                  = var.lb_dns["users"]
    }
  }
}

"/users/{userId}" = {
  get = {
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "GET"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["users"]
      uri                  = var.lb_dns["users"]
    }
  }
}

"/users/login" = {
  post = {
```

```
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "POST"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["sessions"]
      uri                  = var.lb_dns["sessions"]
    }
  }

  get = {
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "GET"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["sessions"]
      uri                  = var.lb_dns["sessions"]
    }
  }
}

"/users/{userId}/movements" = {
  get = {
```

```
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "GET"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType     = "VPC_LINK"
      connectionId       = var.vpc_link["movements"]
      uri                = var.lb_dns["movements"]
    }
  }
}

"/users/{userId}/accounts" = {
  get = {
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "GET"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType     = "VPC_LINK"
      connectionId       = var.vpc_link["accounts"]
      uri                = var.lb_dns["accounts"]
    }
  }

  post = {
```

```
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "POST"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["accounts"]
      uri                  = var.lb_dns["accounts"]
    }
  }
}

"/accounts/{accountId}" = {
  get = {
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "GET"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType       = "VPC_LINK"
      connectionId         = var.vpc_link["accounts"]
      uri                  = var.lb_dns["accounts"]
    }
  }

  post = {
```

```
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "POST"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType     = "VPC_LINK"
      connectionId       = var.vpc_link["accounts"]
      uri                = var.lb_dns["accounts"]
    }
  }
}

"/accounts/{accountId}/payments" = {
  post = {
    security = {ApiKeyAuth= []}
    x-amazon-apigateway-api-key-source = "HEADER"
    x-amazon-apigateway-integration = {
      httpMethod           = "POST"
      payloadFormatVersion = "1.0"
      type                 = "HTTP_PROXY"
      connectionType     = "VPC_LINK"
      connectionId       = var.vpc_link["payments"]
      uri                = var.lb_dns["payments"]
    }
  }
}
```

```
    "/risks" = {
      post = {
        security = {ApiKeyAuth= []}
        x-amazon-apigateway-api-key-source = "HEADER"
        x-amazon-apigateway-integration = {
          httpMethod           = "POST"
          payloadFormatVersion = "1.0"
          type                 = "HTTP_PROXY"
          connectionType       = "VPC_LINK"
          connectionId         = var.vpc_link["risks"]
          uri                  = var.lb_dns["risks"]
        }
      }
    }
  })

  name = "fintech-simulator-public-api"

  endpoint_configuration {
    types = ["REGIONAL"]
  }
}

resource "aws_api_gateway_deployment" "fintech_simulator" {
  rest_api_id = aws_api_gateway_rest_api.fintech_simulator.id

  triggers = {
```

```
                                                    redeployment =
sha1(jsonencode(aws_api_gateway_rest_api.fintech_simulator.body))
  }

  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_api_gateway_stage" "production" {
  deployment_id = aws_api_gateway_deployment.fintech_simulator.id
  rest_api_id   = aws_api_gateway_rest_api.fintech_simulator.id
  stage_name    = "production"
}
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  backend "s3" {
    bucket = "fintech-simulator"
    key    = "terraform/state/aws-api-gateway"
    region = "us-east-1"
  }
```

```
  required_version = ">= 1.2.0"
}

provider "aws" {
  region  = "${var.region}"
}
resource "aws_api_gateway_usage_plan" "fintech_simulator_external_payments" {
  name         = "payments-external-integrations-plan"
  description  = "Payments External Quota"
  product_code = "PAYMENTS_EXTERNAL"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 10000
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_payments_to_risk" {
```

```
  name         = "payments-risk-integrations-plan"
  description  = "Payments To Risk Quota"
  product_code = "PAYMENTS_RISK"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 10000
    period = "WEEK"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource  "aws_api_gateway_usage_plan"  "fintech_simulator_payments_to_accounts"
{
  name         = "payments-accounts-integrations-plan"
  description  = "Payments To Accounts Quota"
  product_code = "PAYMENTS_ACCOUNTS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
```

```
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 10000
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_risk_to_users" {
  name         = "risk-users-integrations-plan"
  description  = "Risk To users Quota"
  product_code = "RISK_USERS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 10000
    period = "DAY"
  }
```

```
  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_risk_to_movements" {
  name         = "risk-movements-integrations-plan"
  description  = "Risk to movements Quota"
  product_code = "RISK_MOVEMENTS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 10000
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}
```

```
resource "aws_api_gateway_usage_plan" "fintech_simulator_risk_to_sessions" {
  name         = "risk-sessions-integrations-plan"
  description  = "Risk to sessions Quota"
  product_code = "RISK_SESSIONS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 20
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_risk_to_accounts" {
  name         = "risk-accounts-integrations-plan"
  description  = "Risk to accounts Quota"
  product_code = "RISK_ACCOUNTS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
```

```
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 20
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_users_to_accounts" {
  name         = "users-accounts-integrations-plan"
  description  = "Users to acounts Quota"
  product_code = "USERS_ACCOUNTS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 20
    period = "DAY"
  }
```

```
  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}

resource "aws_api_gateway_usage_plan" "fintech_simulator_sessions_to_users" {
  name         = "sessions-users-integrations-plan"
  description  = "Sessions to users Quota"
  product_code = "SESSIONS_USERS"

  api_stages {
    api_id = aws_api_gateway_rest_api.fintech_simulator.id
    stage  = aws_api_gateway_stage.production.stage_name
  }

  quota_settings {
    limit  = 20
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 5
    rate_limit  = 100
  }
}
variable "region" {
```

```
  type    = string
  default = "us-west-2"
}

variable "instance_type" {
  type    = string
  default = "t4g.nano"
}

variable "cidr" {
  description = "The CIDR block for the VPC."
  default     = "10.0.0.0/16"
}

variable "public_subnets" {
  description = "List of public subnets"
  default     = ["10.0.100.0/24", "10.0.101.0/24"]
}

variable "vpc_link" {
  type = map
  default     = {
    "sessions" = "g0vsjt"
    "accounts" = "qluk28"
    "risks" = "qpd9i3"
    "payments" = "hdltsi"
    "users" = "fgzypi"
    "movements" = "gvvkz3"
```

```
  }
}

variable "lb_dns" {
  type = map
  default = {
    "sessions" = "http://sessions-lb-9a6971fbfb96cf77.elb.us-west-2.amazonaws.com"
    "accounts" = "http://accounts-lb-087dec338ac80f93.elb.us-west-2.amazonaws.com"
    "risks" = "http://risks-lb-f171ef7f099bd186.elb.us-west-2.amazonaws.com"
                                                    "payments"          =
"http://payments-lb-a0f0c58a82c6dc40.elb.us-west-2.amazonaws.com"
    "users" = "http://users-lb-f9b1be416a2ee5d9.elb.us-west-2.amazonaws.com"
                                                  "movements"           =
"http://movements-lb-ee66bfe4dee08f55.elb.us-west-2.amazonaws.com"
  }
}

variable "private_subnets" {
  description = "List of private subnets"
  default     = ["10.0.0.0/24", "10.0.1.0/24"]
}

variable "availability_zones" {
  default = ["us-west-2a", "us-west-2b"]
}

locals {
  namespace      = "fintech-simulator-production"
```

```
}
resource "aws_cloudwatch_log_group" "logs" {
  name              = "/fargate/service/${var.service_name}"
  retention_in_days = 7
}
resource "aws_ecr_repository" "service_ecr" {
  name                 = "${var.service_name}"
  image_tag_mutability = "MUTABLE"

  tags = {
    project = "fintech"
  }
}
resource "aws_ecs_task_definition" "task" {
  family                   = "${var.service_name}-task" # Naming our first task
  container_definitions    = <<DEFINITION
  [
    {
      "name": "${var.service_name}-container",
      "image": "${aws_ecr_repository.service_ecr.repository_url}:latest",
      "essential": true,
      "portMappings": [
        {
          "containerPort": ${var.container_port},
          "hostPort": ${var.container_port},
          "protocol": "tcp"
        }
      ],
```

```
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-group": "/fargate/service/${var.service_name}",
        "awslogs-region": "us-west-2",
        "awslogs-create-group": "true",
        "awslogs-stream-prefix": "ecs"
      }
    },
    "environment": ${var.environment_definitions},
    "memory": 512,
    "cpu": 256
  }
 ]
 DEFINITION
 requires_compatibilities = ["FARGATE"] # Stating that we are using ECS Fargate
  network_mode             = "awsvpc"    # Using awsvpc as our network mode as this is
required for Fargate
 memory                   = 512         # Specifying the memory our container requires
 cpu                      = 256         # Specifying the CPU our container requires
 execution_role_arn       = "${aws_iam_role.ecsTaskExecutionRole.arn}"
 task_role_arn            = "${aws_iam_role.fargate_role.arn}"
}


resource "aws_iam_role" "ecsTaskExecutionRole" {
 name               = "${var.service_name}-ecsTaskExecutionRole"
                                              assume_role_policy                    =
"${data.aws_iam_policy_document.assume_role_policy.json}"
```

```
}

data "aws_iam_policy_document" "assume_role_policy" {
  statement {
    actions = ["sts:AssumeRole"]

    principals {
      type        = "Service"
      identifiers = ["ecs-tasks.amazonaws.com"]
    }
  }
}

resource "aws_iam_role_policy_attachment" "ecsTaskExecutionRole_policy" {
  role       = "${aws_iam_role.ecsTaskExecutionRole.name}"
                                                    policy_arn                    =
"arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}

resource "aws_iam_role" "ecs_task_role" {
  name = "${var.service_name}-task"

  assume_role_policy = <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Action": "sts:AssumeRole",
```

```
    "Principal": {
      "Service": "ecs-tasks.amazonaws.com"
    },
    "Effect": "Allow",
    "Sid": ""
   }
 ]
}
EOF
}
#assume role is the assume role

resource "aws_iam_role" "fargate_role" {
  name               = "${var.service_name}-fargate-role"
  assume_role_policy = <<EOF
{
   "Version": "2008-10-17",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {
            "Service": "ecs-tasks.amazonaws.com"

         },
         "Action": "sts:AssumeRole"
      }
```

```
    ]
}
EOF
}

resource "aws_iam_policy" "policy" {
  name        = "${var.service_name}-policy-role"
  path        = "/"

  policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "ecr:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
```

```
            "logs:CreateLogGroup"
          ],
          "Resource": "*"
        }
    ]
}
EOF
}

resource "aws_iam_role_policy_attachment" "fargate_attachment" {
  role       = aws_iam_role.fargate_role.name
  policy_arn = aws_iam_policy.policy.arn
}

resource "aws_ecr_repository_policy" "service-repo-policy" {
  repository = aws_ecr_repository.service_ecr.name
  policy     = <<EOF
  {
    "Version": "2008-10-17",
    "Statement": [
      {
        "Sid": "adds full ecr access to the demo repository",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
          "ecr:BatchCheckLayerAvailability",
          "ecr:BatchGetImage",
          "ecr:CompleteLayerUpload",
```

```
          "ecr:GetDownloadUrlForLayer",
          "ecr:GetLifecyclePolicy",
          "ecr:InitiateLayerUpload",
          "ecr:PutImage",
          "ecr:UploadLayerPart",
          "logs:CreateLogGroup"
        ]
      }
    ]
  }
  EOF
}
resource "aws_lb" "nlb" {
  name               = "${var.service_name}-lb"
  internal           = true
  load_balancer_type = "network"
  subnets            = var.subnets_id
  enable_deletion_protection = false
  enable_cross_zone_load_balancing = true

  tags = {
    Name        = "${var.service_name}-lb"
    Environment = "production"
  }
}
resource "aws_lb_target_group" "nlb_tg" {
  depends_on  = [
    aws_lb.nlb
```

```
  ]
  name        = "${var.service_name}-tg"
  port        = var.lb_port
  protocol    = "TCP"
  vpc_id      = var.vpc_id
  target_type = "ip"
}

# Redirect all traffic from the NLB to the target group
resource "aws_lb_listener" "nlb_listener" {
  load_balancer_arn = aws_lb.nlb.id
  port              = var.lb_port
  protocol    = "TCP"
  default_action {
    target_group_arn = aws_lb_target_group.nlb_tg.id
    type             = "forward"
  }
}

resource "aws_api_gateway_vpc_link" "main" {
  name        = "${local.namespace}-${var.service_name}"
   description = "allows public API Gateway for ${local.namespace} to talk to private
NLB"
  target_arns = [aws_lb.nlb.arn]
}
resource "aws_security_group" "service_security_group" {
  name = "${var.service_name}-private-sg"
  vpc_id = var.vpc_id
```

```
  ingress {
    from_port       = 0
    to_port         = 0
    protocol        = "-1"
    security_groups = [aws_security_group.load_balancer_security_group.id]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name        = "${var.service_name}-service-sg"
    Environment = "production"
  }
}

resource "aws_security_group" "load_balancer_security_group" {
  vpc_id = var.vpc_id

  ingress {
    from_port        = var.container_port
    to_port          = var.container_port
```

```
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  tags = {
    Name        = "${var.service_name}-sg"
    Environment = "production"
  }
}


resource "aws_security_group_rule" "task_ingress_rule" {
   description              = "Only allow connections from SG ${var.service_name} on
port ${var.container_port}"
  type                     = "ingress"
  from_port                = var.container_port
  to_port                  = var.container_port
  protocol                 = "tcp"
  source_security_group_id = aws_security_group.load_balancer_security_group.id


  security_group_id = aws_security_group.service_security_group.id
```

```
}

resource "aws_security_group_rule" "nlb_egress_rule" {
  description              = "Only allow connections from SG ${var.service_name}"
  type                     = "ingress"
  from_port                = var.container_port
  to_port                  = var.container_port
  protocol                 = "tcp"
  source_security_group_id = aws_security_group.service_security_group.id

  security_group_id = aws_security_group.load_balancer_security_group.id
}
resource "aws_ecs_cluster" "cluster" {
  name = "${var.service_name}-cluster" # Naming the cluster
}

resource "aws_ecs_service" "service" {
  name            = "${var.service_name}-service"                     # Naming our first service
  cluster         = "${aws_ecs_cluster.cluster.id}"             # Referencing our created Cluster
  task_definition = "${aws_ecs_task_definition.task.arn}" # Referencing the task our service will spin up
  launch_type     = "FARGATE"
  desired_count   = 1 # Setting the number of containers we want deployed to 1

  network_configuration {
    subnets         = var.subnets_id
    assign_public_ip = false
```

```
    security_groups = [
      aws_security_group.service_security_group.id,
      aws_security_group.load_balancer_security_group.id
    ]
  }

  load_balancer {
    target_group_arn = aws_lb_target_group.nlb_tg.arn
    container_name   = "${var.service_name}-container"
    container_port   = var.container_port
  }

}
variable "service_name" {
  type    = string
  default = ""
}

variable "vpc_id" {
  type    = string
  default = ""
}

variable "subnets_id" {
  default = ""
}

variable "public_subnets_id" {
```

```
  default = ""
}

variable "environment_definitions" {
  default = "[]"
}

variable "container_port" {
  default = "8080"
}

variable "lb_port" {
  default = "80"
}

locals {
  namespace      = "fintech-simulator-production"
}
```

Script para infraestructura en GCP

```
resource "google_api_gateway_api" "api_gw" {
  api_id = "fintech-simulator"
}

resource "google_api_gateway_api_config" "api_gw" {
  provider = google-beta
  api = google_api_gateway_api.api_gw.api_id
```

```
  api_config_id = "config"

  openapi_documents {
    document {
      path = "spec.yaml"
      contents = filebase64("../docs/specs/swagger.yaml")
    }
  }
  lifecycle {
    create_before_destroy = true
  }
}

resource "google_api_gateway_gateway" "api_gw" {
  provider = google-beta
  api_config = google_api_gateway_api_config.api_gw.id
  gateway_id = "api-gw"
}
provider "google" {
  project = var.project
  region  = var.region
}

terraform {
  backend "gcs" {
    bucket = "fint-sim-gcp"
    prefix = "terraform/state"
  }
```

```
  required_providers {
    google      = ">= 4.32.0"
  }
}

module "payments_service_module" {
    source = "./microservices"
    service_name = "payments"
    service_runtime = "go118"
    environment_variables = {
      PORT = "8080",
      RISK_URL = "",
      ACCOUNTS_URL = ""
    }
}

module "accounts_service_module" {
    source = "./microservices"
    service_name = "accounts"
    service_runtime = "java17"
}

module "movements_service_module" {
    source = "./microservices"
    service_name = "movements"
    service_runtime = "nodejs16"
}
```

```
module "users_service_module" {
    source = "./microservices"
    service_name = "users"
    service_runtime = "java17"
    environment_variables = {
      PORT = "8080",
      RISK_URL = "",
      ACCOUNTS_URL = "",
      SESSION_SECRET = "",
      SALT_SECRET = ""
    }
}

module "sessions_service_module" {
    source = "./microservices"
    service_name = "sessions"
    service_runtime = "java17"
    environment_variables = {
      PORT = "8080",
      USERS_URL = ""
    }
}

module "risks_service_module" {
    source = "./microservices"
    service_name = "risks"
    service_runtime = "java17"
```

```
    environment_variables = {
      PORT = "8080",
      MOVEMENTS_URL = "DNS",
      ACCOUNTS_URL = "DNS",
      SESSIONS_URL = "DNS",
      USERS_URL = "DNS",
    }
    #container_port = "5000"
}
variable "project" {
        default = "fintech-simulator"
}
variable "region" {
        default = "us-central1" # Choose a region
}
resource "google_cloudfunctions_function" "service" {
  provider              = google-beta
  name                  = var.service_name
  description           = var.service_name
  runtime               = var.service_runtime
  available_memory_mb   = var.memory
  trigger_http          = true
  docker_repository     = "<artifact registry path>" //This is wrong
  docker_registry       = "<artifact registry path>" //This is wrong
  environment_variables = var.environment_variables
}
output "function_url" {
        value = google_cloudfunctions_function.service.https_trigger_url
```

```
}
variable "service_name" {
  type    = string
}

variable "service_runtime" {
  type    = string
  default = "java17"
}

variable "memory" {
  type    =  string
  default = "256"
}

variable "environment_variables" {
  type    = map
  default = {}
}
```

Script para infraestructura en AZURE

```
resource "azurerm_application_gateway" "app_gateway" {
  name                = var.application_gateway_name
  resource_group_name = var.resource_group_name
  location            = var.region
```

```
  tags = var.tags

  # app gateway SKU
  dynamic "sku"  {
    for_each = var.sku
      content{
          name     = sku.value.name
          tier     = sku.value.tier
      }
  }

  # following settings configures firewall policy with application gateway
  firewall_policy_id = var.firewall_policy_id ==""?null : var.firewall_policy_id

  # app gateway standard WAF setting, applicable if firewall policy Id not configured
  dynamic "waf_configuration"  {
    for_each =  var.waf_configuration #var.firewall_policy_id == ""  ?: []
      content{
        #  name                  = waf_configuration.value.name
          enabled               = lookup(waf_configuration.value,"enabled",false)
                                                    file_upload_limit_mb                    =
lookup(waf_configuration.value,"file_upload_limit_mb",30)
                                                firewall_mode                               =
lookup(waf_configuration.value,"firewall_mode","Detection")
                                                            max_request_body_size_kb        =
lookup(waf_configuration.value,"max_request_body_size_kb",128)
                                                request_body_check                          =
lookup(waf_configuration.value,"request_body_check",true)
                                                rule_set_type                               =
lookup(waf_configuration.value,"rule_set_type","OWASP")
```

```
            rule_set_version          = lookup(waf_configuration.value,"rule_set_version",
"3.1")
      }
  }

  dynamic "identity"  {
      for_each = var.requires_identity==true?[1]:[]
      content{
        identity_ids = var.user_managed_identity
         type        = "UserAssigned"
      }
  }
provider "azurerm" {
  features {}
}

terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.0.0"
    }
  }
}

module "payments_service_module" {
    source = "./microservices"
    name = "payments"
```

```
}

module "accounts_service_module" {
    source = "./microservices"
    name = "accounts"
}

module "movements_service_module" {
    source = "./microservices"
    name = "movements"
}

module "users_service_module" {
    source = "./microservices"
    name = "users"
}

module "sessions_service_module" {
    source = "./microservices"
    name = "sessions"
}

module "risks_service_module" {
    source = "./microservices"
    name = "risks"
}
# Public Ip
resource "azurerm_public_ip" "appgw_public_ip" {
```

```
  name                = var.appgw_public_ip_name
  resource_group_name = var.resource_group_name
  location            = var.region

  allocation_method   = var.appgw_public_ip_allocation_method
  sku                 = var.appgw_public_ip_sku
}
variable region {
    default = "West Europe"
}

variable "application_gateway_name" {
    type = string
    default = "fintech-simulator-gtw"
}

variable "resource_group_name" {
    type = string
    default = "fintech-simulator-network-rg"
}

variable "user_managed_identity" {
   type          = list
}

resource "azurerm_service_plan" "service" {
  name                = var.name
  resource_group_name = azurerm_resource_group.example.name
```

```
  location            = azurerm_resource_group.example.location
  os_type             = "Linux"
  sku_name            = "Y1"
}

resource "azurerm_linux_function_app" "service" {
  name                = "example-linux-function-app"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location

  storage_account_name       = azurerm_storage_account.example.name
                                                    storage_account_access_key                    =
azurerm_storage_account.example.primary_access_key
  service_plan_id            = azurerm_service_plan.example.id

  site_config {}
}
resource "azurerm_resource_group" "service" {
  name     = var.group_name
  location = var.region
}
resource "azurerm_storage_account" "service" {
  name                = "${var.name}-storage"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

```
variable "group_name" {
   default = "backend-pool"
}

variable "region" {
   default = "West Europe"
}

variable "name" {
   default = ""
}
```

Script Escenario Marketing

```
import os
import sys
import random

from locust import HttpUser, User, task, between

sys.path.append(os.getcwd())

BASE_URI = os.getenvb(b'BASE_URI', "http://localhost:8001")


class MarketingCampaign(HttpUser):
  """
```

```
Class defining marketing campaign event that checks user account
"""

host = BASE_URI
wait_time = between(5, 15)

__MAX_USER_ID = 500000000

""" Contains task definition for executing the request to the server """

@task
def marketing_campaigns(self):
  rand_index = random.randint(0, self.__MAX_USER_ID)

  headers = {
    'Content-Type': 'application/json',
  }
  with self.client.get(
      url=self.user_account_path(rand_index),
      headers=headers,
      name="marketing-campaign",
      catch_response=True
  ) as response:
    if response.status_code < 500:
      response.success()
    else:
      response.failure()
```

```
  def user_account_path(self, user_id) -> str:
    return f"/users/{user_id}/accounts"
```

Script Escenario Procesamiento de Pagos

```
import os
import sys
import random

from locust import HttpUser, User, task, between

sys.path.append(os.getcwd())

BASE_URI = os.getenvb(b'BASE_URI', "http://localhost:8003")
BASIC_AUTH = os.getenvb(b'BASIC_AUTH', "")


class ReceivePayments(HttpUser):
  """
  Class defining payments workflow for fintech
  """

  host = BASE_URI
  wait_time = between(5, 15)

  __MAX_ACCOUNT_ID = 500000000
  __MAX_USER_ID = 500000000
```

```
__MAX_AMOUNT = 1000000

""" Contains task definition for executing the request to the server """

@task
def receive_payments(self):
  rand_account_id = random.randint(0, self.__MAX_ACCOUNT_ID)
  rand_user_id = random.randint(0, self.__MAX_USER_ID)
  rand_amount = random.randint(0, self.__MAX_AMOUNT)
  rand_receiver_account_id = random.randint(0, self.__MAX_ACCOUNT_ID)

  headers = {
    'Content-Type': 'application/json',
    'Authorization': f"{BASIC_AUTH}"
  }
  with self.client.post(
      url=self.payments_path(rand_account_id),
      json={
        "type": "WITHDRAW",
        "correlation": {
          "account_number": rand_receiver_account_id
        },
        "user_id": rand_user_id,
        "operation": {
          "amount": rand_amount,
          "currency": "USD",
          "type": 1
        }
```

```
        },
        headers=headers,
        name="external-payments-integration",
        catch_response=True
    ) as response:
      if response.status_code < 500:
        response.success()
      else:
        response.failure()

  def payments_path(self, account_id) -> str:
    return f"/accounts/{account_id}/payments"
```

Script Escenario Análisis de Riesgo

```
import os
import sys
import random

from locust import HttpUser, User, task, between

sys.path.append(os.getcwd())

BASE_URI = os.getenvb(b'BASE_URI',
                      "http://localhost:8004")
BASIC_AUTH = os.getenvb(b'BASIC_AUTH', "fruta")
API_KEY = os.getenvb(b'X_API_KEY', "XXX")
```

```
class BatchRiskAnalisis(HttpUser):
  """
  Class defining batch risk analisis for your fintech.
  """

  host = BASE_URI
  wait_time = between(5, 15)

  __MAX_ACCOUNT_ID = 500000000
  __MAX_AMOUNT = 1000000

  """ Contains task definition for executing the request to the server """

  @task
  def risk_analisis(self):
    rand_account_id = random.randint(0, self.__MAX_ACCOUNT_ID)
    rand_payment_id = random.randint(0, self.__MAX_ACCOUNT_ID)
    rand_amount = random.randint(0, self.__MAX_AMOUNT)

    headers = {
      'Content-Type': 'application/json',
      'X-FINTECH-AUTH': f"{BASIC_AUTH}",
      'x-api-key': API_KEY
    }
    with self.client.post(
        url=self.risk_analisis_path(),
```

```
        json={
          "type": "WITHDRAW",
          "amount": rand_amount,
          "account_id": rand_account_id,
          "user_id": rand_account_id,
          "payment_id": rand_payment_id
        },
        headers=headers,
        name="risk-analisis",
        catch_response=True
    ) as response:
      if response.status_code < 500:
        response.success()
      else:
        response.failure()


  def risk_analisis_path(self) -> str:
    return f"/risks"
```

CURRICULUM VITAE