

VERSIONAMIENTO EN ARQUITECTURA ORIENTADA A MICROSERVICIOS

A PREPRINT

 **Johann S. Bogotá**

Ingeniería de Sistemas
Escuela Colombiana de Ingeniería Julio Garavito
Colombia, Bogotá
johann.bogota@mail.escuelaing.edu.co

 **Johan S. Guerrero**

Ingeniería de Sistemas
Escuela Colombiana de Ingeniería Julio Garavito
Colombia, Bogotá
johan.guerrero@mail.escuelaing.edu.co

May 22, 2021

ABSTRACT

This paper analyzes an experiment attacking the versioning problem that is obtained when working with micro services architectures, where changing the contract or API can affect the operation of the application. To show the results, a template was developed in Cloud Formation that allowed us to create the Gateway API and migrate each of its clients to a new version.

Keywords API GateWay · Versioning · CloudFormation

1 Introducción

Cuando queremos trabajar con arquitecturas orientadas a microservicios, se genera una problemática común, ya que cuando se quiere cambiar el contrato del servicio o el API, este afecta a diferentes clientes o microservicios que la usan. Para resolver este problema se va a usar la práctica de versionamiento de servicios, en donde al cambiar algo del API se crea una versión nueva, la cual durante por un tiempo convive con la versión anterior. De esta manera Todos los clientes se irán migrando poco a poco hasta quedar solo la nueva versión.

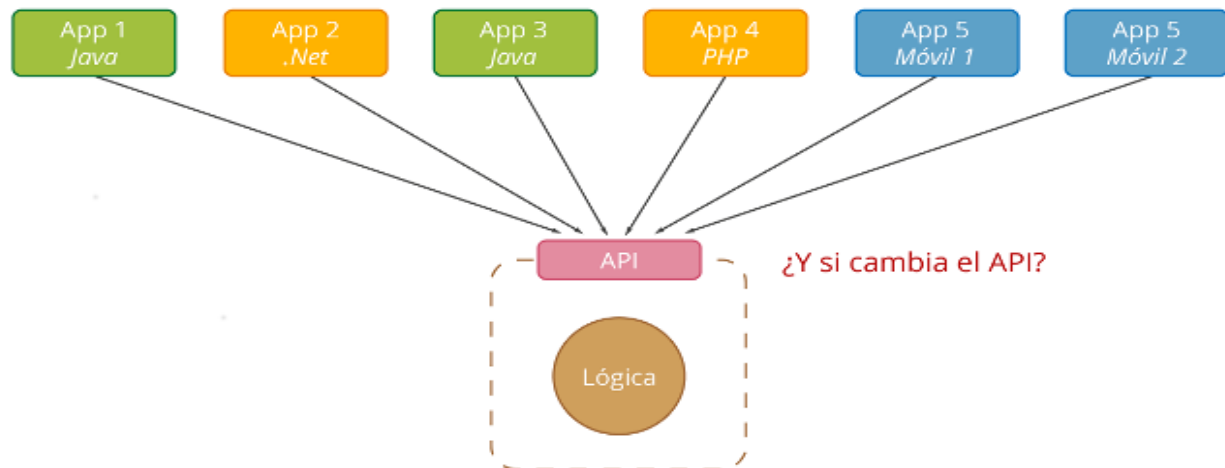
AWS nos proporciona un servicio llamado Amazon API Gateway, que nos facilita la creación, publicación, mantenimiento y monitoreo del API. También nos permite ejecutar varias versiones de la misma API simultáneamente, generando así beneficios en nuestra aplicación. A partir de este modelo, se hace uso de funciones lambda que proporcionan los recursos utilizados por los diferentes clientes y contendrán distintas versiones que serán usadas en los entornos creados en el API Gateway.

En este documento reflexionaremos sobre las ventajas del versionamiento, así como también llevaremos a cabo una implementación de una Rest API con ayuda de Cloud Formation, el cual nos ayudará a modelar los recursos necesarios para la creación y administración por medio de plantillas, se modelará diferentes API con ayuda de las etapas proporcionadas por el servicio de API Gateway, las cuales son referencias específicas de una implementación. Esto con el fin de medir las métricas de desempeño de la herramienta API Gateway y analizar sus tiempos de despliegue y migración con un gran tamaño de APIs simuladas.

2 Problemática

Cuando contamos con diferentes clientes consumiendo una API que está siendo reutilizada, tenemos una arquitectura distribuida. Pero surge la duda de, ¿qué pasa si algo del API cambia? Una posible solución sería sincronizar todos los clientes con nuestra nueva API, sin embargo, esto puede ser muy riesgoso ya que tocaría idear un plan de contingencia por si algo falla, además si un punto falla se tendría que revertir todo o es posible que alguno de los

clientes no pueda migrarse en ese momento, por lo tanto, es aquí donde sale a resaltar el versionamiento de servicios.



3 Marco teórico

3.1 Microservicios

El paradigma de microservicio proporciona a los equipos de desarrollo un enfoque más descentralizado para construir software. Los microservicios permiten que cada servicio sea aislado, reconstruido, red desplegado y administrado de forma independiente. Por ejemplo, si un programa no genera informes correctamente, puede ser más fácil rastrear el problema a ese servicio específico. Ese servicio específico podría luego probarse, reiniciarse, parchearse y volver a implementarse según sea necesario, independientemente de otros servicios.

Los microservicios facilitan la prueba y el despliegue de cambios. Debido a que cada uno está separado de los otros, se mejora el aislamiento de fallas. Si hay un problema en el software, el servicio problemático puede ser aislado, remediado, probado y redistribuido sin la necesidad de realizar una prueba de regresión de toda la aplicación como ocurre con las arquitecturas de aplicaciones monolíticas tradicionales. La arquitectura de microservicios mejora la agilidad empresarial con un desarrollo e implementación de software más rápido en comparación con la arquitectura de software monolítica.

Sin embargo, los microservicios no son libres de gestión. Con la misma cantidad de servicios que un producto de software monolítico, la administración requerida en una arquitectura de microservicios podría ser más compleja ya que cada servicio está separado uno del otro. Esto puede llevar a dificultades para manejar todas las partes de un todo. Por ejemplo, se necesita una cuidadosa supervisión y administración para rastrear la disponibilidad y el rendimiento de todos los servicios de componentes que operan dentro de una aplicación de microservicio.

3.2 API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la publicación, el mantenimiento, el monitoreo, la protección y la utilización de las API a cualquier escala. Se trata de un servicio de pago por uso que se ocupa de las tareas más arduas relacionadas con la ejecución segura y de confianza de las API a escala.

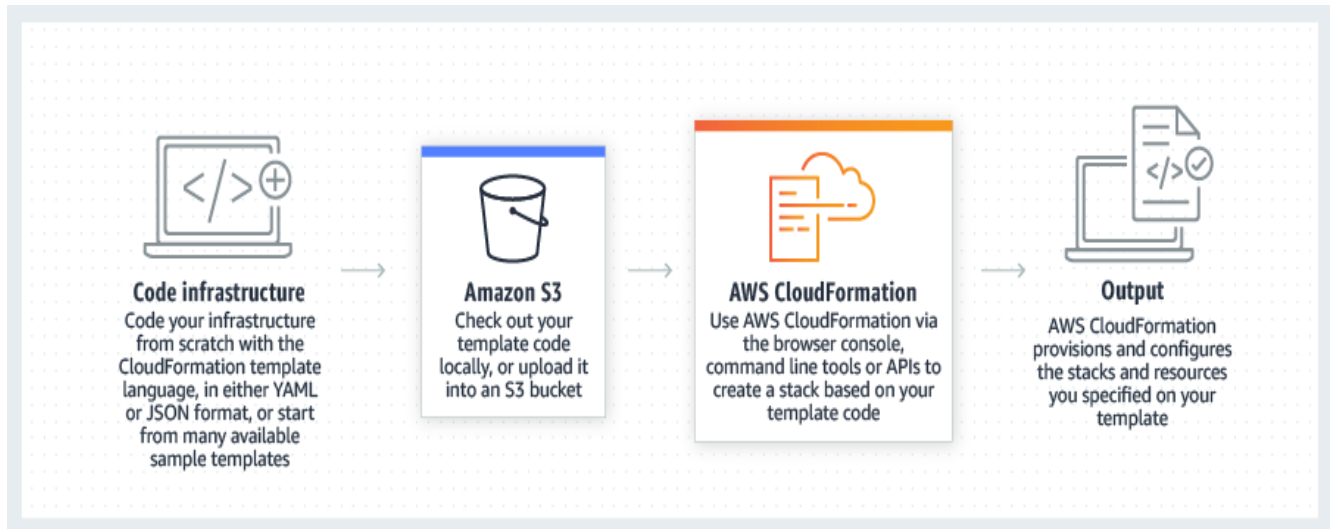
Beneficios

- Desarrollo de API eficiente
- Rendimiento a cualquier escala
- Ahorro de costos a escala

- Monitoreo fácil
- Controles de seguridad flexibles
- Opciones de API RESTful

3.3 AWS CloudFormation

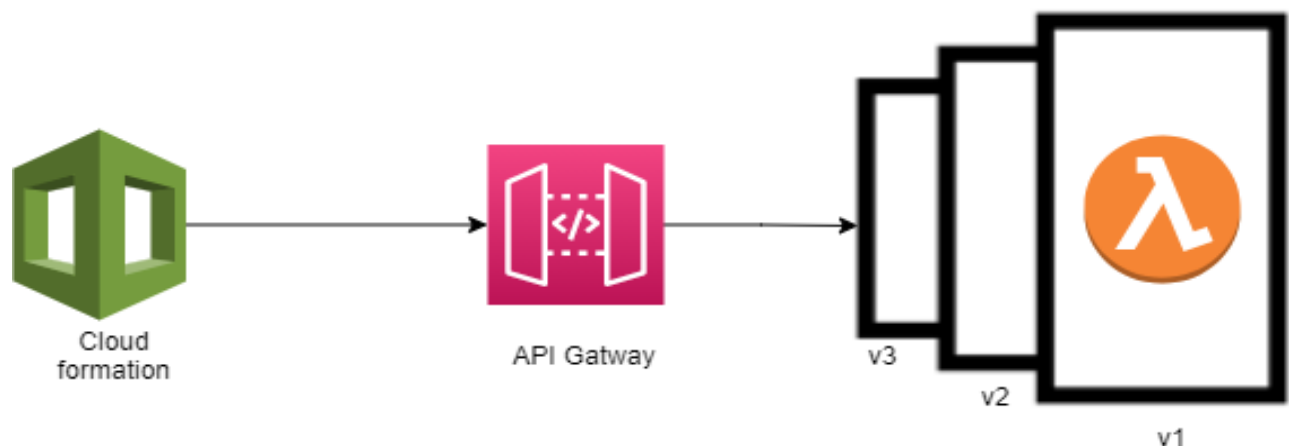
Esta herramienta, nos permite administrar recursos relacionados con AWS y terceros, tratándola como una infraestructura como código. Con ayuda de plantillas se describe los recursos y dependencias para lanzar y configurar una pila tantas veces como sea necesario. La plantilla ya sea bajo formatos correspondientes de YAML o JSON son procesados y subidos a un bucket de Amazon S3, para posteriormente ser utilizada para la creación de una pila basado en el template. Y este a su vez ejecutara todos los comandos para la creación de los recursos especificados.



4 Implementación

4.1 Arquitectura

Se realizará una arquitectura basada en microservicios, con ayuda del servicio de AWS CloudFormation, se generara el despliegue y migración del recurso API Rest de Api Gateway a la nueva versión con el fin de crear el numero máximo de etapas permitidas por una cuenta AWS educate, que cumplan el rol de diferentes APIs clientes que apuntaran a una función lambda.



4.2 Desarrollo

Para generar el template indicando la creación del API Gateway y cada una de sus etapas, se utilizó un script con un algoritmo implementado en lenguaje python, capaz de escribir en un archivo Json los parámetros necesarios de acuerdo a un número de etapas indicados por el usuario, posteriormente, se realizó el despliegue del template en cloudformation, donde se generaban distintos datos como el tiempo del ciclo de vida de cada uno de los recursos.

Una vez creada el API Gateway procedemos a realizar pruebas con la aplicación Postman verificando la versión que contiene el endpoint de un stage en específico y procedemos a actualizar nuestro template, en este caso ajustando los recursos necesarios para migrar de una versión a otra. Para llevar a cabo esto, se apoyó de otro script que genera diferentes despliegues con una marca de tiempo única requerida para hacer la migración y todos los cambios que apuntan hacia la nueva versión lambda.

Se realizaron 6 pruebas donde se modificaba la cantidad de etapas y se repite el mismo ejercicio dicho anteriormente, se divide en dos el experimento, el primero extrayendo datos de tiempo de la creación del API Gateway y cada uno de los stages, y el segundo extrayendo datos de la actualización y despliegue de los mismos. En la siguiente sección procedemos a realizar un análisis de resultados de las tablas y graficas obtenidas.

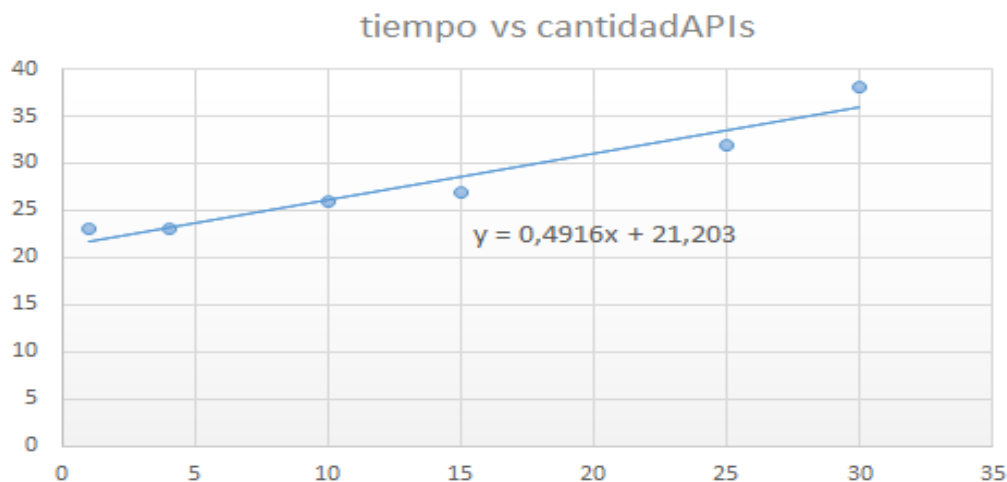
4.3 Análisis de resultados

Una vez obtenido todos los datos de cada una de las pruebas, se obtuvo distintos tiempos que con ayuda de una regresión lineal se pudo encontrar una línea de tendencia. La cual da una idea del comportamiento de AWS API Gateway ante la creación y actualización de muchos recursos.

Se realizó un análisis de las métricas de disponibilidad y tiempo que toma cloud formation en la creación de distintas cantidades de recursos. Aumentando en cada prueba la cantidad de etapas que se desean desplegar. Dando como resultados la siguiente información.

CANTIDAD DE ETAPAS	tiempo de inicio	tiempo en terminar	Dif	Horas	Minutos	Segundos	Time(s)
1	17/05/2021 23:09:02	17/05/2021 23:09:25	0,000266204	0	0	23	23
4	17/05/2021 21:37:17	17/05/2021 21:37:40	0,000266204	0	0	23	23
10	17/05/2021 21:11:10	17/05/2021 21:11:36	0,000300926	0	0	26	26
15	18/05/2021 21:34:03	18/05/2021 21:34:30	0,0003125	0	0	27	27
25	17/05/2021 21:26:33	17/05/2021 21:27:05	0,00037037	0	0	32	32
30	17/05/2021 21:41:00	17/05/2021 21:41:38	0,000439815	0	0	38	38

Con estos datos y con ayuda de la regresión lineal podemos observar el comportamiento promedio con respecto al tiempo y cantidad de etapas que puede tomar cuando se aumenta cada una de estas. Dando como resultado una ecuación lineal, como vemos en la siguiente imagen.

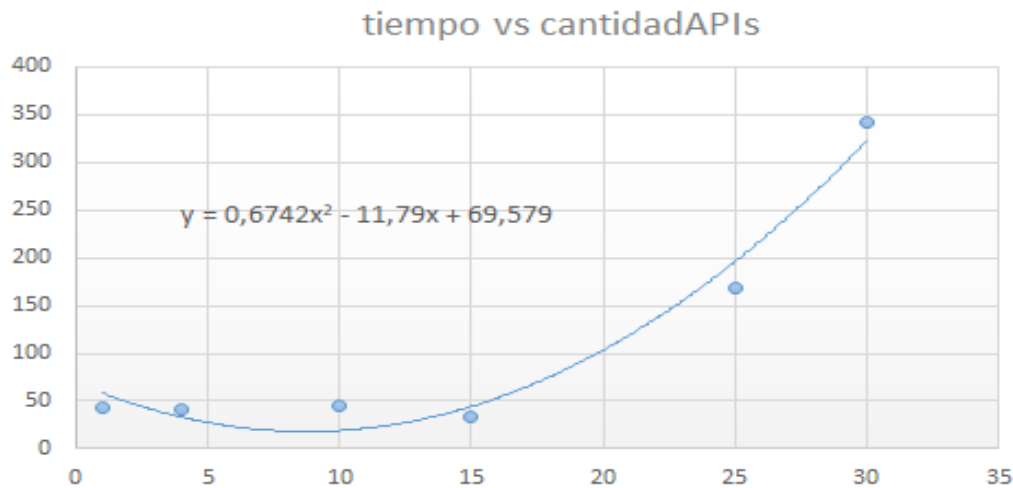


Finalmente podemos observar el aumento del tiempo con respecto a la cantidad de etapas a crear, por lo que la creación de 100 etapas siguiendo la ecuación nos costaría un tiempo de 70 segundos aproximadamente, dando como resultado 1 min y 10 segundos que toma CloudFormation para desplegar esta cantidad de etapas en ApiGateway.

El segundo análisis se toma respecto a la métrica de rendimiento y tiempo de migración de versión de una función lambda en cada una de las etapas creadas en Api GateWay. Dando como resultados la siguiente información.

CANTIDAD DE ETAPAS	tiempo de inicio	tiempo en terminar	Dif	Horas	Minutos	Segundos	Time(s)
1	17/05/2021 23:09:44	17/05/2021 23:10:27	0,000497685	0	0	43	43
4	17/05/2021 21:42:36	17/05/2021 21:43:17	0,000474537	0	0	41	41
10	18/05/2021 21:12:10	18/05/2021 21:12:56	0,000532407	0	0	46	46
15	18/05/2021 21:35:56	18/05/2021 21:36:30	0,000393519	0	0	34	34
25	19/05/2021 21:27:35	19/05/2021 21:30:24	0,001956019	0	2	49	169
30	20/05/2021 21:42:45	20/05/2021 21:48:26	0,003946759	0	5	41	341

De la misma manera que la gráfica anterior, se realizó una regresión lineal con los nuevos datos, en este caso se obtuvo como resultado una ecuación exponencial, que se evidencia en la figura siguiente, que da a entender cómo es su comportamiento en el tiempo al actualizar muchas más etapas, por lo tanto si tomamos el mismo valor usado en el análisis anterior, tener 100 etapas y actualizar cada una de ellas, nos costaría 6799 segundos, es decir 1 hora y 53 minutos.



5 Conclusión

- Con el experimento realizado se llegó a la hipótesis que el tiempo que dura la migración de un servicio es potencialmente exponencial con respecto al tiempo de creación, esto debido a que API Gateway genera una alta disponibilidad, es decir, que el servicio sigue funcionando mientras se realiza la actualización de un recurso a una nueva versión.
- Gracias a herramientas de AWS como CloudFormation se puede manejar la infraestructura como código lo que permite una fácil gestión y mantenimiento de la información en todo su ciclo de vida.
- Por medio de AWS API Gateway y CloudFormation se pudo observar el ciclo de vida de cada uno de los stages, donde AWS lo hace de manera paralela, por lo tanto, al hacer muchas operaciones de creación, actualización y eliminación al mismo tiempo se necesita un gran manejo de la concurrencia.

6 Referencias

Implementación de una API de REST en Amazon API Gateway - Amazon API Gateway https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/how-to-deploy-api.html

Petersen, S. (2015). AWS API Gateway, Device Farm Are Wins for Developers. EWeek, 1.

Tutorial: Creación de una API CRUD con Lambda y DynamoDB, Available: <https://docs.aws.amazon.com/eses/apigateway/latest/developerguide/http-api-dynamo-db.html>

Curso completo de infraestructura como código usando AWS CloudFormation, aprende desde lo básico hasta lo avanzado: <https://www.udemy.com/course/cloudformation/>

Documentación oficial AWS Cloud Formation. https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/Welcome.html