Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación y Gestión de Microservicios a través de un Api Gateway

Autor: Daniel Reynés Fernández

Tutor(a): Santiago Tapia Fernández

Madrid, Mayo 2023

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Creación y Gestión de Microservicios a través de un Api Gateway
Mes Año

Autor: Daniel Reynés Fernández

Tutor:
Santiago Tapia Fernández
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Agradecimientos

Antes de comenzar querría agradecer el apoyo y la ayuda que me han brindado ciertas personas a lo largo de mi trayectoria en la carrera, así como durante la realización de este TFG.

Primero agradecer a mi madre a mi padre a mi hermano y mi hermana quienes han tenido que aguantar mis momentos de nervios y de angustia.

También tengo que agradecer a Ana y Mónica mis amigas que me han acompañado toda la vida o al menos desde donde tengo recuerdos, siempre he podido contar con su confianza y ayuda y me han escuchado mis quejas y también me han ayudado a desconectar cuando me ha hecho falta. Además, cuando yo tenía dudas ellas me han mostrado su confianza y siempre han creído más en mí que yo mismo.

Querría agradecer también a mi amiga Marina, que me ha dado siempre apoyo. En particular cuando estaba agobiado por cumplir con plazos pensaba en cómo debería estar ella con muchas cosas por hacer que yo, y pensaba si ella va a poder yo que no lo tengo tan complicado también.

Por último y sin restarle importancia agradecer a la empresa por darme la oportunidad de hacer el TFG con ellos y a los compañeros por estar siempre dispuestos a ayudar.

Resumen

Este trabajo de fin de grado se ha desarrollado en base a la necesidad de gestionar de forma más adecuada los microservicios que tiene en funcionamiento una empresa.

Debido a la flexibilidad y velocidad de su implementación, los microservicios son cada vez más recurrentes a la hora de estructurar una aplicación. Sus beneficios le han permitido ganar cada vez más popularidad, aunque también dispone de algunos inconvenientes, como puede ser el manejo de un gran número de ellos o posibles vulnerabilidades de seguridad

En este TFG se ha llevado a cabo la implementación de un Api-Gateway como solución a estos problemas que genera el tener un amplio número de microservicios.

Con el propósito de conseguir la mejor solución se han buscado y analizado las distintas alternativas disponibles en el mercado para la implementación de un Api-Gateway. Por otro lado, se han creado algunos microservicios con el objetivo de entender mejor su funcionamiento y establecer una metodología acerca de cómo crearlos.

Además, se han aplicado funcionalidades a los servicios a través del Api-Gateway con el objetivo de mejorar su rendimiento seguridad y escalabilidad.

Abstract

This final degree project has been developed based on the need to manage the microservices that a company has in operation in a more appropriate way.

Due to the flexibility and speed of their implementation, microservices are becoming more and more common when structuring an application. Their benefits have allowed them to gain more and more popularity, although they also have some disadvantages, such as the handling of a large number of them or possible security vulnerabilities.

In this TFG we have carried out the implementation of an Api-Gateway as a solution to these problems generated by having many microservices.

In order to achieve the best solution, the different alternatives available on the market for the implementation of an Api-Gateway have been searched and analysed. On the other hand, some microservices have been created in order to better understand how they work and to establish a methodology on how to create them.

In addition, functionalities have been applied to the services through the Api-Gateway in order to improve their performance, security and scalability.

Tabla de contenidos

1 Introducción

En la actualidad, las aplicaciones y los servicios web han ganado una gran relevancia en el mundo empresarial y tecnológico. El aumento del uso de dispositivos móviles y la nube ha provocado que las aplicaciones y los servicios web se hayan convertido en la principal forma de ofrecer servicios y productos a los clientes.

Este aumento de uso de aplicaciones y la flexibilidad y velocidad de su implementación, ha hecho que los microservicios sean cada vez una opción más recurrente a la hora de estructurar una aplicación.

Los microservicios web [1] son una arquitectura de software que se basa en la división de una aplicación en múltiples componentes pequeños e independientes, cada uno de los cuales ofrece una funcionalidad concreta y se comunica con los demás a través de protocolos web como HTTP, REST o SOAP.

Cada microservicio se encarga de una tarea específica dentro de la aplicación y puede ser desarrollado y escalado de forma independiente de los demás. De esta forma, se consigue mayor flexibilidad, eficiencia y escalabilidad en el desarrollo y gestión de aplicación.

Además, los microservicios web permiten una mayor capacidad de prueba, debido a la posibilidad de probar cada microservicio de forma aislada, lo cual facilita la detección y corrección de errores de aplicación.

En este entorno, la arquitectura de microservicios se ha convertido en un enfoque popular para el desarrollo de aplicaciones distribuidas, por permitir a los desarrolladores trabajar de manera independiente y desacoplada en pequeños servicios reutilizables.

Sin embargo, cuando una aplicación depende de varios microservicios se crean múltiples puntos de entrada de tráfico de la aplicación, lo que dificulta la gestión y el control del tráfico.

Por todo esto, la gestión de las APIs (interfaces de programación) entre los servicios puede resultar complicada y requiere de una solución que facilite su gestión.

El API Gateway [2] [3] es una solución que actúa como un proxy inverso que recibe todas las solicitudes de la aplicación y las enruta a los microservicios correspondientes.

Un proxy inverso es un servidor que actúa como intermediario situándose en antes de los servidores web, recibe las solicitudes del cliente y se las envía a los servidores web. Se suelen utilizar para balancear la carga de los servicios web haciendo así más fluidas las aplicaciones y para aumentar la seguridad.

Al actuar en función de intermediario entre los clientes y los servicios de una aplicación, el API Gateway proporciona una capa de abstracción y un punto de entrada único para acceder a los servicios. Además, puede proporcionar funciones de seguridad, gestión de tráfico, caché y gestión de logs entre las funciones más destacadas.

Por otra parte, el API Gateway también puede proporcionar una serie de servicios adicionales, como en la autenticación y autorización de usuarios, la limitación de las llamadas a las APIs, la gestión de caché, la monitorización y análisis del tráfico, y la transformación de datos, lo que reduce la complejidad de la configuración y el mantenimiento de los microservicios.

En resumen, el API Gateway se ha convertido en una herramienta imprescindible para la gestión de APIs en cualquier aplicación. Su uso permite mejorar la seguridad, la escalabilidad y el rendimiento de la API, así como facilitar la integración con servicios de terceros y el control de acceso a la misma. Además, existen diferentes herramientas y tecnologías para implementar un API Gateway, lo que permite adaptarse a las necesidades y requerimientos de cada aplicación.

Nos centraremos en este trabajo en explicar el funcionamiento de un API Gateway, así como sus funcionalidades y características más relevantes. Se analizarán distintas alternativas que se ofrece en el mercado y se escogerá una de ellas para la implementación de uno propio. A través, de la implementación del API Gateway se desarrollará las distintas alternativas de configuración, así como la implementación de funcionalidades a microservicios como gestión de tráfico o autenticación.

1.1 Objetivos

El objetivo principal de este Trabajo de Fin de Grado (TFG) es explorar y comprender en profundidad el funcionamiento de un API Gateway, así como los conceptos y funcionalidades que se pueden aplicar a los microservicios a través de este. Para lograrlo, se establecen los siguientes objetivos específicos:

- Comprender el funcionamiento de un API Gateway
- Conocer las distintas plataformas existentes para la creación de uno
- Diseño y despliegue del Api Gateway a través de una plataforma escogida
- Establecer una metodología en la creación de los servicios
- Despliegue, configuración y uso de microservicios a través del Api Gateway
- Creación de flujos entre microservicios
- Análisis y explotación de funcionalidades que van más allá del enrutamiento
- Integración del uso de API Gateway en proyectos en producción

1.2 Motivación

Actualmente me encuentro trabajando y parte de este trabajo la realización de servicios web. La empresa en la que me encuentro está aumentando muy rápido el número de servicios web que usan por tanto ha nacido la necesidad de usar una herramienta que gestione estos servicios.

En el comienzo de este trabajo mi conocimiento sobre los servicios web, así como sobre la arquitectura de microservicios no era muy profunda, pero para las tareas que me eran solicitadas eran suficientes. Sin embargo, en cuanto a cómo gestionarlos y que es y cómo funciona un API Gateway mis nociones acerca de ello eran prácticamente nulas. Este es el principal motivo por el cuál las primeras tareas de este TFG han sido la documentación y formación.

Por último, uno de los motivos por lo que decidí centrarme en este tema fue porque me parece muy útil, ya que no solo se pueden hacer funciones de gestión como el tráfico o el acceso, sino que también permite aplicar funcionalidades como el guardado de cache lo que hacen más completos los servicios.

Además, el hecho de que sea un proyecto real en el sentido de que ciertos aspectos que se van a aprender y probar durante este TFG en un futuro cercano se aplicarán al proyecto de la empresa. Esto hace que la motivación sea mayor ya que hacer algo útil da valor al trabajo y genera más satisfacción.

2 Estado del arte

2.1 Métodos de comunicación entre microservicios

Las aplicaciones suelen comunicarse entre sí mediante protocolos específicos de que pueden ser síncronos, como HTTP o asíncronos como es el caso de AMQP. Las aplicaciones deben ser diseñadas acorde al protocolo que se deseaba usar, lo cual puede dificultar la integración.

2.1.1 Protocolo síncrono HTTP

El protocolo HTTP [4] (HyperText Transfer Protocol) se basa en estructuras de clientes y servidores. El cliente es quien inicia la comunicación realizando una petición. Cada petición individual se envía a un servidor, el cuál gestiona y responde. Entre la petición y respuesta pueden existir intermediarios denominados proxies. Un proxy es un programa intermediario que puede ser usado para facilitar acceso o añadir funcionalidades como transformar cabeceras o implementar cache. Esto hace que la ejecución del código del cliente solo continúa y fluida.

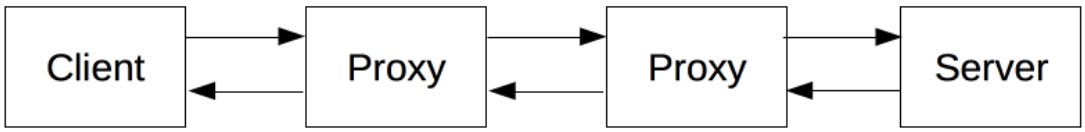

Ilustración 1 Representación protocolo HTTP

2.1.2 Protocolo asíncrono AMQP

AMQP (Advanced Message Queuing Protocol) [5] es un protocolo asíncrono compatible con diversos sistemas operativos, así como con la nube. Este se protocolo se caracteriza por la orientación a mensajes, el uso de encolamiento, el enrutamiento de los mensajes producidos, la exactitud en la entrega de los mensajes y la seguridad.

El funcionamiento se basa en el intercambio de mensajes a través de una red en la que intervienen productores, consumidores y colas de mensajes.

Los consumidores se conectan a las colas para extraer los mensajes generados por los productores. Entre el productor y la cola de mensajes debe existir uno más sistemas de enrutamientos denominados “exchange”.

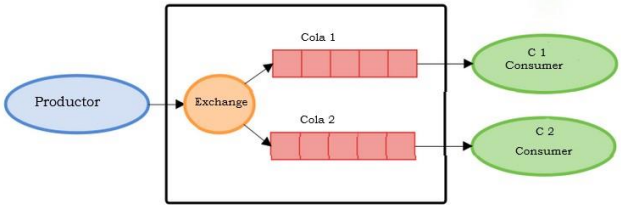

Ilustración 2 Representación protocolo AMQP

2.1.3 Comunicación de microservicios: API

Una API (Application Programming Interface) [6] es una tecnología usada en la conexión entre servicios web, permitiendo la transmisión de datos. En términos generales API se refiere a un conjunto de reglas, protocolos y herramientas que se utilizan para desarrollar software y la comunicación de aplicaciones.

Además, pueden ser utilizadas con el fin de integrar diferentes servicios para que trabajen juntos aumentando su eficiencia y permitiendo acceder a funcionalidades de unos y otros.

2.1.3.1 API SOAP

El API SOAP (Simple Object Access Protocol) [7] es una interfaz de programación de aplicaciones (API) basada en el protocolo SOAP. Este protocolo de comunicación está basado en XML y es utilizado para el intercambio de datos entre las aplicaciones.

Se utilizan mensajes estructurados en XML para la comunicación entre aplicaciones. Estos mensajes resultan complejos y además pueden contener información redundante lo que los hace más pesados aún.

Por otro lado, SOAP se compone de varias capas de abstracción, una capa para mensaje, transporte y aplicación. Si bien esta abstracción podría ser útil para hacerlo más seguro, lo que crea es una dificultad a la hora de comprender el protocolo. Además, tiene especificaciones para casa elemento del mensaje como pueden ser el cuerpo o las cabeceras.

El mapeo de los tipos de datos se requiere que sea de forma explícita esto aparte de aumentar la complejidad dificulta la compatibilidad entre aplicaciones.

Todo esto hace que la API SOAP poco flexible y con inconvenientes a la hora del mantenimiento y la escalabilidad de las aplicaciones. Sin embargo, fue muy popular en el pasado.

2.1.3.2 API REST

El API REST (Representational State Transfer) [7] se basa en la representación de los recursos como objetos que pueden ser manipulados mediante una serie de operaciones conocidas (GET, PUT, POST, DELETE). Por otra parte, está basado en el protocolo HTTP para la comunicación.

Las operaciones sobre los recursos se realizan mediante la manipulación de sus representaciones que pueden tener formato XML o JSON.

La principal característica de la API REST que la diferencia de otras es que no la comunicación no tiene estado, lo que significa que cada solicitud debe incluir toda la información necesaria para procesarla. Los clientes y servidores si tienen su estado propio pero el servidor no almacena datos del cliente como pueden ser sus credenciales solo se verían modificaciones que hace el cliente a través de una petición.

Otra característica importante es el modo de identificación de recursos. Cada recurso es identificado de manera única mediante una URI (Uniform Resource Identifier).

La representación de los recursos suele ser en JSON o XML. Siendo más usado JSON por ser un recurso más ligero.

Por último, he de destacar otro principio el cual es que cada mensaje intercambiado entre cliente y servidor debe tener la información necesaria para que el receptor pueda entenderlo y procesarlo correctamente.

El enfoque de la arquitectura rest es muy común debido a su flexibilidad simplicidad y escalabilidad en el diseño de sistemas distribuidos.

2.2 Tecnologías usadas

A continuación, se va a hablar de las tecnologías que han sido requeridas para la realización de este trabajo de fin de grado. Algunas de ellas ya eran conocidas previamente, sin embargo, otras no y he tenido que familiarizarme con ellas y aprender sus características y funcionalidades.

2.2.1 Microservicios

En este apartado se mencionan aquellas herramientas usadas específicamente para el desarrollo de los microservicios web. Han sido utilizadas en conjunto ya que se complementan entre sí ayudando al desarrollo.

2.2.1.1 Java

El lenguaje de programación escogido para la realización de los servicios ha sido java. En esta elección, no había más opciones en la empresa en la cual se ha realizado este trabajo ya tenían servicios en java y para seguir la misma línea se ha usado el mismo lenguaje.

Java [8] es un lenguaje de programación caracterizado por ser orientado a objetos. Esto permite la reutilización de entidades (objetos) entre proyectos esto es uno de los motivos de su gran popularización. Otro motivo podría ser la independencia de la plataforma esto significa que los programas escritos en java pueden ejecutarse de la misma manera en otro tipo de hardware. Otro aspecto relevante de este lenguaje es como evita el problema de fugas de memoria. Esto lo hace gestionando el ciclo de vida de los objetos, cuando no quedan referencia a uno este es eliminado.

2.2.1.2 Eclipse

Como plataforma para el desarrollo se ha usado Eclipse IDE, esta elección no tiene mayor motivo que ser la herramienta que se ha usado con más regularidad durante la carrera y al ser una con la que ya se estaba familiarizado. Además, no se han encontrado encontrando funcionalidades de otras plataformas lo suficientemente atractivas y diferentes que supusiesen una ventaja respecto a Eclipse.

Eclipse IDE (Integrated Development Environment) [9] es una plataforma de desarrollo, creada con el objetivo de poder ser ampliada mediante el uso de plug-ins. En cuanto al lenguaje, es un IDE genérico y no fue concebido para usarse con ninguno en concreto. Sin embargo, es comúnmente usado por usuarios de java.

Algunas características que hacen atractivo Eclipse son la gestión de proyectos, la depuración de código que incluye el cual es potente fácil e intuitivo, y la extensa colección de plug-ins estos ya sean creados por terceros o por Eclipse añaden funcionalidades útiles para el programador.

2.2.1.3 SpringBoot

SpringBoot [10] es un framework desarrollado para el trabajo con el lenguaje de programación Java. Es importante mencionar, que se trata de un software de código abierto y gratuito. Es comúnmente usado en el desarrollo de microservicios por su arquitectura modular y por la facilidad de integración otras herramientas.

Otra característica que facilita la creación de microservicios es la posibilidad de crear microservicios como aplicaciones independientes, lo que ayuda a la escalabilidad y por otro lado.

Además, SpringBoot tiene a su disposición una amplia gama de librerías y herramientas como la integración con bases de datos o la seguridad. También tiene una amplia gama de opciones de configuración para poder adaptarse mejor a cada caso.

Estas características hacen de esta herramienta una gran opción para el desarrollo de microservicios por su arquitectura modular, su capacidad de integración con otras herramientas y sus diversas opciones de configuración.

2.2.1.4 Maven

Maven [11] es una herramienta de software creada para la gestión y construcción de proyectos software Java. Se usa como ayuda para los programadores automatizando la construcción de aplicaciones y gestionando las dependencias eficientemente.

El archivo que define el proyecto Maven es el pom.xml en este se define la configuración del proyecto, las dependencias y plugins usados con el fin de compilar correctamente el proyecto.

Otro aspecto relevante es la capacidad de integrarse con otras herramientas. En este caso se ha usado junto con SpringBoot. Al combinar estas dos herramientas en un mismo proyecto se ve facilitado el desarrollo y despliegue de los microservicios.

En resumen, Maven y SpringBoot ofrecen una solución completa en el desarrollo de microservicios. Maven se encargaría de la gestión de dependencias y construcción del proyecto mientras SpringBoot del despliegue de estos.

2.2.2 WildFly

WildFly [12] es un servidor de aplicaciones Java de código abierto y gratuito desarrollado por Red Hat, Inc. Se utiliza para crear, implementar y hospedar aplicaciones javas. Además, es compatible con cualquier sistema operativo siempre y cuando este cuente con java.

En este proyecto WildFly se ha usado para poder desplegar varios microservicios al mismo tiempo.

2.2.3 Postman

Postman [13] es una herramienta usada en el desarrollo de aplicaciones basadas en http para crear probar y documentar APIs. La interfaz es amigable al usuario permitiendo crear colecciones para organizar las solicitudes. Los desarrolladores pueden crear solicitudes http a un api, visualizar la respuesta en distintos formatos como JSON o XML y guardar estas respuestas como ejemplo.

Una característica importante para la selección de esta herramienta en el ambiente laboral es la capacidad de compartir colecciones con otros miembros lo que facilita el trabajo en equipo.

A pesar de su funcionalidad principal es la de probar las APIs a través de solicitudes HTTP, Postman cuenta con otras funciones. Postman cuenta con la herramienta “flows” que permite encadenar solicitudes, manejar datos y crear

flujos de trabajo. Otra herramienta es “Mock server” en ella puedes simular el funcionamiento de una API real, además puedes hacer peticiones al mock server.

2.2.4 Kong

Kong [14] es una herramienta que nos permite crear nuestro propio API Gateway. Está escrito en el lenguaje Lua y creado sobre Nginx. Es, además, de código abierto y ofrece una versión gratuita bastante completa.

Kong ofrece una amplia variedad de herramientas que podemos añadir para aplicar funcionalidades como gestión de tráfico o autenticación sobre las distintas Apis de un proyecto.

Por otra parte, Kong ofrece distintas opciones de instalación como puede ser a través de contenedores y Docker o en un sistema operativo Debian o Ubuntu entre algunas opciones.

3 Desarrollo

3.1 Análisis alternativas implementación Api Gateway

En este capítulo se tratarán de explicar las distintas alternativas para la construcción del Api Gateway que se han considerado, ventajas que tienen frente a otras y los motivos de la decisión de usar Kong.

Una de las opciones podría ser crear un Api Gateway desde cero, para ello se deberían definir bien los requisitos, así como las funcionalidades que debería tener (autenticación, gestión de caché, etc.), y los lenguajes y frameworks a usar para su desarrollo. Sin embargo, se consideró que crearlo desde cero no era una alternativa viable debido a la complejidad, el tiempo requerido y los recursos necesarios. También es importante mencionar que la escalabilidad y seguridad sería menor que implementando una de las opciones que están disponibles en el mercado.

La alternativa de construir el API Gateway desde cero se descartó debido a la complejidad que supone y a que a largo plazo es menos escalable se vuelve más complejo y menos seguro. También se tuvo en cuenta que este proyecto tiene como propósito acercarse lo máximo a un proyecto real a llevar a cabo por una empresa de dimensiones pequeñas-mediana.

Siendo descartada esta opción queda buscar alternativas en el mercado para la implementación del API Gateway. Las principales características que se tuvieron en cuenta para buscar opciones fueron que sean de escalables, que exista cierta facilidad en la aplicación de funcionalidades y que tuviesen licencia para un uso de prueba o gratuito.

Teniendo en cuenta estas restricciones durante la búsqueda y análisis de alternativas sobresalen algunas dignas de mención que son las que se van a explicar y desarrollar en los siguientes apartados que son WSO2 Api Gateway, APIsix y Kong.

3.1.1 WSO2 API Gateway

WSO2 [15] es una compañía de middleware que entre sus productos se encuentra una solución a la implementación de API Gateway que es WSO2 API Manager. Es una plataforma, de código abierto, de gestión de APIs y cuenta con

las características necesarias para la implementación de un API Gateway como son el control de tráfico, seguridad y autenticación.

Una de sus características más relevantes es su flexibilidad y capacidad de personalización al usuario. Permite la comunicación con distintos tipos de API como pueden ser Rest, Soap o GraphQL. Esto es un gran punto para poder ser usado en proyectos multiplataforma.

Además, permite la protección de las API con distintas opciones de seguridad como pueden ser el control de acceso, la encriptación o la autenticación. También permite la creación de grupos de usuario para la configuración de seguridad y así crear medidas de seguridad personalizadas.

Por otra parte, destaca la capacidad de integración con otros productos WSO2 como WSO2 Identity server y WSO2 Enterprise Integrator lo que añade aún más funcionalidades para integrar lo que permite crear una plataforma coherente y completa.

Otras funcionalidades relevantes de WSO2 API Manager son el monitoreo en tiempo real o la gestión de ciclo de vida de API que permite a los usuarios gestionar cuando se publican y cuando se retiran las APIs. Además, cuenta con un portal para desarrolladores que permite a los desarrolladores gestionar las APIs y realizar pruebas.

En resumen, tiene todas las herramientas necesarias para la gestión y creación de un API Gateway, desde la creación y la publicación hasta la monitorización y el análisis. La plataforma además es altamente personalizable y escalable, lo cual la hace perfecta para proyectos de diferente complejidad y tamaño.

Por otra parte, WSO2 cuenta con una licencia “trial” para estudiantes y otros campos que permiten el uso de la herramienta sin embargo esta tiene una duración en tiempo limitada a 90 días.

3.1.2 APISIX

APIsix [16] es una plataforma, de código abierto, de gestión de un API Gateway creada por Tengine aunque ahora es mantenida por Apache Fundation.

APIsix está desarrollado con el fin de poder garantizar una proporcionar una alta disponibilidad, seguridad y escalabilidad. Para ello ofrece una gran variedad de funcionalidades.

Estas funcionalidades son el enrutamiento y transformación de API que ofrece la posibilidad de gestionar distintas API de manera centralizada y la transformación de solicitudes y respuestas. El control de acceso el cual se lleva a cabo a través de la gestión de usuarios de contraseñas o la gestión de tokens de autenticación. También puede limitar acceso por grupos de usuarios. Y el monitoreo y análisis pudiendo recoger métricas para analizar el rendimiento, registrar logs y detectar errores.

Por otra parte, APIsix ofrece una arquitectura basada en plugins que permite a los usuarios agregar funcionalidades según su conveniencia además dispone de plugins desarrollados por APIsix, así como por otros usuarios. APIsix también permite una configuración dinámica sin tener que reiniciar el API Gateway para aplicar cambios.

APIsix es una gran alternativa a la hora de a creación de un API Gatway porque además de ser una plataforma de código abierto es totalmente gratuita. Además, cuenta con APIsix Dashboard que es la implementación de una interfaz de usuario que facilita la gestión del API Gateway siendo esta vista muy intuitiva y fácil de usar. Sin embargo, a pesar de sus grandes características, APIsix aún cuenta con una documentación limitada y una comunidad de usuarios creciente pero pequeña.

3.1.3 Kong

Kong [14] es una plataforma de código abierto de la empresa Kong Inc que proporciona las herramientas para la creación y gestión adecuada de un API Gateway.

Kong está basado en el servidor proxy Nginx, Se caracteriza por ser altamente escalable y adecuado para proyectos y organizaciones de distintos tamaños. Cuenta al igual que APIsix con herramientas para el enrutamiento y transformación de APIs, control de acceso y monitoreo y análisis.

Kong tiene una arquitectura modular basada en plugins los cuales pueden ser desarrollados por la propia empresa Kong Inc o por usuarios externos. Además, permite el desarrollo e integración de tus propios plugins. Esto proporciona una gran capacidad de personalización y una mayor flexibilidad en la implementación del API Gateway.

Además, Kong ofrece soluciones para gestionar el tráfico a través de balanceo de carga a través de múltiples instancias de API. También permite la integración con servicios de la nube como pueden ser Kubernetes, Docker o AWS Lambda. Esto facilita la implementación y escalado de APIs en la nube.

Por otra parte, Kong cuenta con un portal de desarrollador que proporciona una interfaz donde los desarrolladores pueden gestionar el API Gateway. Esto lo hace más sencillo de usar ya que de otro modo toda la gestión debería hacerse por comandos a través de consola.

En resumen, Kong es una opción muy completa para el desarrollo de un API Gateway, cuenta con una versión gratuita, sin embargo, está limitada a un único workspace y no da acceso a todos los plugins ni funcionalidades.

3.1.4 Comparación y elección de herramienta

Haciendo un análisis de las distintas opciones para la implementación del API Gateway se optó por usar Kong. La elección de esta herramienta se basa en distintos factores.

Por un lado, WSO2 API Gateway unido al uso de otras herramientas de wso2 como WSO2 Esterprise Integrator ofrece más funcionalidades que Kong y una en particular que puede resultar muy interesante. Esta funcionalidad es la creación de flujos entre microservicios. Sin embargo, el sistema de Kong basado en plugins lo hace más personalizable además de que wso2 tan solo permite 90 días de prueba lo que sería un tiempo algo ajustado en la realización de este proyecto.

Por otro lado, Kong ofrece capacidad de balanceo de carga una característica que no recoge WSO2 y que se considera bastante útil si se espera un tráfico elevado un API.

En cuanto a APIsix tiene a su favor que es una herramienta totalmente gratuita algo que las otras dos herramientas no sin embargo aún está en etapa de desarrollo y en algunos aspectos falta documentación.

Aunque APIsix se considera una opción igual de válida que Kong sin embargo la documentación es más escasa, así como el número de usuarios. Esto lo hace menos atractivo debido a la mayor dificultad a la hora de encontrar soluciones a problemas durante la implementación.

En cuanto a la instalación, no hay ninguna diferencia entre las tres alternativas. Todas tienen las opciones de ser instaladas en local directamente en un sistema operativo como podría ser Ubuntu, instalarse como un contenedor usando Docker lo que permite una implementación fácil y rápida o como imágenes de una máquina virtual lo que permite una implementación en diferentes entornos y sistemas operativos.

Estos aspectos son los que han hecho que la elección para la implementación del API Gateway sea a través de Kong.

	Kong	WSO2	APISIX
Gratuita	✓	✗	✓
Altamente personalizable	✓	✓	✓
Gran comunidad documentación y foros de ayuda	✓	✓	✗
Permite creación de flujos entre servicios	✗	✓	✗
Arquitectura basada en plugins	✓	✗	✓
Escalable	✓	✓	✓

3.2 Instalación y configuración Kong

Una vez elegida Kong como la herramienta a usar para la implementación del API Gateway, el siguiente paso es instalar y configurar la herramienta.

3.2.1 Instalación

En cuanto a la instalación Kong nos proporciona varias alternativas por un lado se pude hacer a través de contenedores con Docker o Kubernetes entre otras posibilidades, mientras que también da la posibilidad de instalarlo directamente en un sistema operativo como pueden ser Debian o Ubuntu.

En nuestro caso se decidió optar por la instalación en un sistema operativo, concretamente en un sistema CentOS debido a que a futuro en la empresa se quiere instalar Kong en una maquina con este sistema operativo, por lo tanto, se ha considerado el más adecuado.

Para realizar la instalación se ha seguido las instrucciones de la documentación que proporciona Kong. Se han seguido las instrucciones para el sistema Red Hat Enterprise Linux (RHEL) ya que es el más similar a CentOS.

Los unos prerrequisitos definidos por la documentación para la instalación de Kong son tener un sistema operativo acceso root o equivalente y en el caso de querer usar las funcionalidades exclusivas de la versión Enterprise para lo cual es necesario un archivo denominado `license.json`.

El primer paso es descargar el paquete de instalación de Kong. Esto se puede hacer a través del siguiente comando.

- `curl -Lo kong-enterprise-edition-3.2.2.1.rpm $(rpm --eval "https://download.konghq.com/gateway-3.x-rhel-%{rhel}/Packages/k/kong-enterprise-edition-3.2.2.1.rhel%{rhel}.amd64.rpm")`

Una vez, se ha descargado el paquete podemos proceder a la instalación, para ello hay dos opciones la primera es usando rpm con el siguiente comando.

- `rpm -iv kong-3.2.2.rpm`

Sin embargo, si se usa esta opción se deben instalar las dependencias de Kong de forma independiente vía microdnf. Este es el motivo por el cuál en vez de usar rpm se ha usado la otra opción. La cuál es a través del uso de yum y el comando es el siguiente.

- `sudo yum install kong-enterprise-edition-3.2.2.1.rpm`

En este momento ya tendríamos instalado el software ahora deberíamos modificar el archivo denominado `kong.conf` generado tras la instalación donde podemos configurar la herramienta acorde a nuestras preferencias y necesidades.

3.2.2 Configuración

Una vez realizada la instalación es momento de establecer la configuración que vaya acorde a nuestro caso. Para ello se debe modificar el fichero kong.conf que es en el cuál se define la configuración.

En este apartado se van a explicar los aspectos más relevantes e importantes de la configuración. En total hay una gran cantidad de opciones de configuración; sin embargo, muchas de ellas son exclusivas de la versión Enterprise de Kong por lo cual no podemos hacer uso de ellas.

Por defecto Kong utiliza una configuración predeterminada la cual viene definida en un archivo llamado `Kong.conf.default` el cual se usó como base para crear nuestro archivo `kong.conf` que será el reemplazo del otro.

El primer aspecto que se definió de la configuración fue la dirección donde se generaran los archivos que definen los logs como `error.log` y `access.log`. Para ello se definió una ruta para guardarlos en una carpeta que se denominó logs.

Una de las opciones que es la configuración acerca de NGINX. Hay que definir las direcciones y puertos en los que el servidor va a estar definido. En cuanto a los puertos se ha optado por la configuración recomendada por Kong por ejemplo usar el 8001 para el administrador. Esta decisión se tomó con el fin de a futuro poder seguir usando la documentación de Kong y que los puertos a los que hace referencia coincidiesen con los nuestros.

Una de las opciones de configuración que puede ser relevante son los plugins. Se debe definir qué tipo de plugins serán aceptados por defecto solo están permitidos los distribuidos de forma oficial por Kong. Se decidió dejar esta opción ya que no se tenía pensado usar plugins externos y siempre se puede modificar la opción si es necesario más adelante.

El aspecto de configuración quizás más importante es el de almacenamiento de datos de Kong. Se permite la posibilidad de usar un archivo yml como sistema de almacenamiento o por otro lado usar una base de datos. Debido a la cantidad de datos y servicios que se quieren usar se ha optado por usar una base de datos. En Kong se recoge la posibilidad de usar dos tipos de bases de datos, que son Cassandra o Postgre.

La base de datos que se ha decidido usar ha sido Postgre ya que acorde a la documentación de Kong, Cassandra no es recomendable y en versiones posteriores de Kong dejará de estar soportadas.

Para poder usar una base de datos de postgre lo primero ha sido instalar postgre para lo cual se ha usado el siguiente comando.

- `sudo apt-get -y install postgresql-12`

Una vez instalado se creó un usuario y contraseña para que sean usados por Kong.

- `CREATE USER kong WITH PASSWORD 'super_secret';`

Luego se ha creado una base de datos propia para el usuario creado previamente.

- `CREATE DATABASE kong OWNER kong;`

El usuario, la contraseña y la base de datos creados deben añadirse en el archivo de configuración para que puedan ser usados.

El último parámetro de configuración importante es habilitar el Kong Manager. Un portal web que nos permitirá usar las funcionalidades de Kong con mucha más facilidad y tener una perspectiva más visual de la herramienta.

3.2.3 Comprobación y ejecución de Kong

Una vez se ha instalado y configurado la herramienta es el momento de iniciarla y comprobar que esté funcionando.

Para lanzar la herramienta se usa el siguiente comando.

- `kong start -c {PATH_TO_KONG.CONF_FILE}`

Si todo ha ido como debería de poder leerse un mensaje en el que diga “Kong started”, aún así hay otro modo de verificar que la instalación ha sido correcta ejecutando el siguiente comando.

- `curl -i http://localhost:8001`

En este caso si todo va bien debería recibirse “status code 200”. En el caso de recibir otro código que se identifique como error debería revisarse el archivo de configuración por si algo no estuviese bien definido y volver a ejecutar Kong.

3.3 Creación microservicios

Los microservicios se han creado en java con la ayuda de Maven y SpringBoot. La creación de distintos microservicios puede resultar monótona y algo repetitiva puesto que hay pasos que se repiten o clases que van a ser casi idénticas independientemente de la lógica del servicio un ejemplo de eso es la clase encargada de lanzar el servicio.

Además, hemos seguido el modelo cliente-servidor para el desarrollo de nuestra arquitectura por ello aparte de crear un servidor hay que crear un cliente y ambos deben coincidir en varios aspectos como los tipos devueltos o nombres de métodos para que la comunicación entre ambos se produzca sin ningún tipo de error.

Antes de proceder a la creación de los microservicios ya se conocía una técnica de crear los clientes de forma automática a partir de su definición en un yml un archivo jar y un archivo config.json donde se definían los nombres del cliente y de los paquetes.

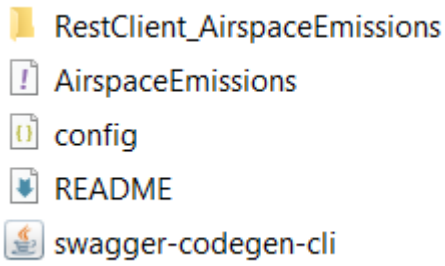

Ilustración 3 Archivos del cliente

En la imagen anterior podemos ver los archivos usados para generar el cliente, así como la carpeta que lo contiene una vez se ha ejecutado el comando que lo crea.

El archivo config tiene la estructura que se muestra en la ilustración 4 donde podemos ver que define nombres de paquetes y la versión del cliente.

```
{
    "invokerPackage": "RestClient_AirspaceEmissions",
    "modelPackage": "RestClient_AirspaceEmissions.model",
    "apiPackage": "RestClient_AirspaceEmissions.api",
    "name": "RestClient_AirspaceEmissions",
    "artifactId": "RestClient_AirspaceEmissions",
    "groupId": "api.crida",
    "artifactVersion": "1.0"
}
```

Ilustración 4 Estructura archivo config

Por otro lado, el archivo denominado en este caso como *RestClient_AirspaceEmissions* es el yml con la definición del cliente. La estructura del yml podríamos dividirla en varias secciones.

En el comienzo del yml se indica la versión de Swagger u OpenApi que se van a usar, el nombre y descripción del proyecto y el tipo de seguridad que se quiere usar.

```
1  openapi: 3.0.0
2  tags:
3    - name: AirspaceEmissions
4      description: Data about emissions group by particle
5  security:
6  - APIKeyHeader: []
```

Ilustración 5 Primeras líneas del fichero yml

En el yml se definen las operaciones, así como los endpoint para acceder a ellas los parámetros que se usan para la llamada y que se devuelve en caso de que todo vaya bien o en caso de error.

```yaml
paths:
  /airspace/emissions/{particle}/{dateFrom}/{dateTo}:
    get:
      description: Returns Emissions per particle
      operationId: getAirspaceEmissions
      responses:
        '200':
          content:
           application/json:
            schema:
             type: array
             items:
               type: object
               additionalProperties:
                 type: object
          description: successful operation
        '500':
          description: Internal server error
        '404':
          description: AirspaceEmissions not found
```

Ilustración 6 Definición operación

En la ilustración 6 podemos ver como se define el path o endpoint de la operación su descripción e id así como que devuelve en distintos casos. En este caso en el path podemos ver entre llaves los parámetros que se envían como path params. En la siguiente ilustración podemos ver como se definen todos los parámetros.

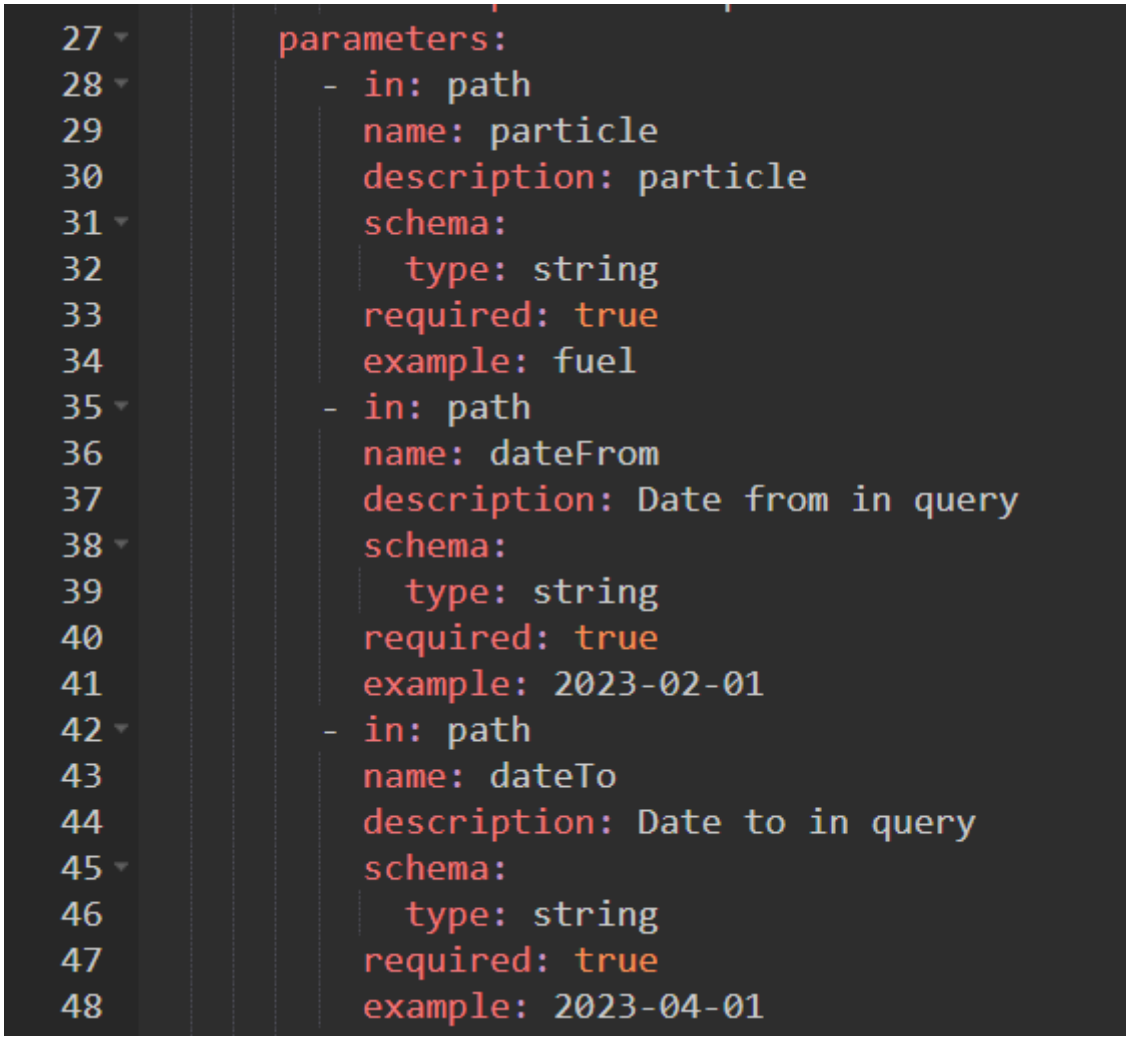

Ilustración 7 Definición de parámetros de la operación

Por último, el archivo README contiene información acerca del cliente, explicación de que son cada archivo, como generar el cliente y como instalarlo como una librería Maven en el repositorio local.

El comando usado para generar los clientes es el siguiente:

- `java -jar swagger-codegen-cli.jar generate -i RestClient_AirspaceEmissions.yml -l java -o RestClient_AirspaceEmissions -c config.json`

- swagger-codegen-cli.jar es el nombre del jar generador

- RestClient_AirspaceEmissions.yml es el nombre con la definición del cliente

- RestClient_AirspaceEmissions es la carpeta donde queremos que se genere el cliente

- config.json archivo con parámetros de configuración como el artifact id o nombre de los paquetes.

Esto agilizaba la creación de los clientes, pero el problema de tener que estar muy pendiente de no cometer errores en nombres de parámetros o tipos para la buena conexión con el servidor seguía existiendo.

Partiendo de este conocimiento de poder crear el cliente con la ayuda de un jar se buscó soluciones similares para la generación de los servicios. Se encontró una solución a esto y es que OpenApi [17] ofrece una gama de generadores tanto de servicios como de clientes para distintos lenguajes. En nuestro caso escogimos el de java con SpringBoot que es el lenguaje y framework que veníamos usando previamente.

La generación del servicio con este jar es similar a la del cliente necesita de la definición de este en un yml y un archivo config.json donde se indican nombres de paquetes y del servicio; además permite añadir otras opciones de configuración como que librería usar para las fechas.

La ventaja principal de usar el jar para generar la base del servicio es que se puede hacer usando el mismo yml que se usa para generar el cliente lo que facilita evitar errores de comunicación entre ellos.

A la hora de generar el servicio se ejecuta el siguiente comando:

- `java -jar openapi-generator-cli-6.3.0.jar generate -g spring -i AirspaceEmissions.yml -c config.json -o AirspaceEmissions`

Después, de la generación del servidor debemos irnos al IDE (entorno de desarrollo integrado) que estemos usando en nuestro caso Eclipse e importar el proyecto como un proyecto Maven ya existente. Una vez importado veremos algo

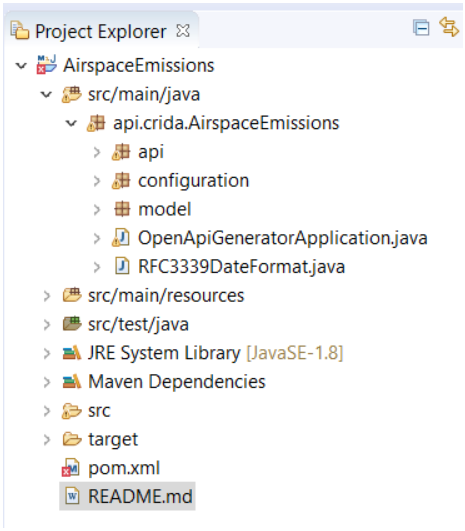

Ilustración 8 Estructura de paquetes del servidor

como lo siguiente.

En el paquete denominado api podremos encontrar la clase controler la cual es la encargada de recibir las peticiones y enviar la respuesta. En configuration encontramos clases que ayudan en aspectos de documentación y configuración. Por otro lado en model encontramos las clases que definen los objetos que va a usar el servicio.

Los cambios necesarios para hacer funcionar el servicio se llevarian a cabo en la clase controller donde se debe hacer la conversión de tipos necesarios y una llamada a otra clase que es la que debe implementar la logica del programa. Idealmente esta clase debe estar en un paquete a parte que debemos crear. Se ha seguido el patrón de llamarlo service ya que es donde se realizaran las operaciones y llamadas a la base de datos. El controller por tanto debe gestionar las llamadas y respuetas para que se adecuen a los tipos establecidos.

Otro aspecto relevante son los archivos de recursos datasource.propperties y application.propperties. En ellos se definen credenciales de la base de datos y propiedades del programa como el puerto a usar.

Con esto el servicio estaría creado y para poder ejecutarlo debemos hacer uso de la clase *OpenApiGeneratorApplication* que es la que lanza el servicio usando SpringBoot. Para probarlo una vez desplegado en local hemos usado Postman para hacer las peticiones.

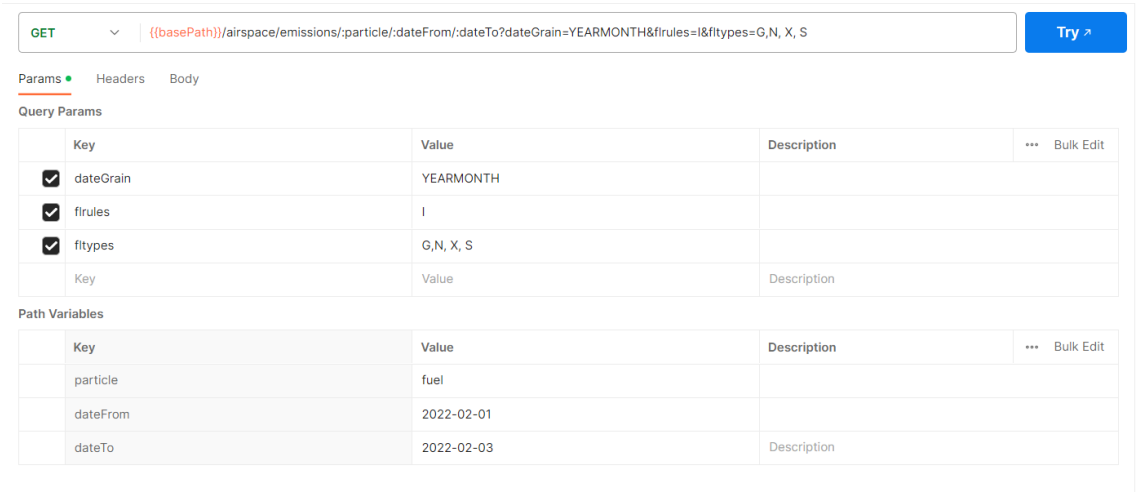

Ilustración 9 Ejemplo petición de Postman

3.3.1.1 Modelo Cliente-Servidor

El modelo cliente-servidor en el contexto de los microservicios es una arquitectura de software utilizada para construir sistemas escalables y distribuidos.

En este modelo, el cliente es responsable de realizar una solicitud a un microservicio en particular para obtener algún tipo de información o realizar una acción. El cliente es quien inicia peticiones o solicitudes al microservicio y esto se hace normalmente a través de una interfaz gráfica con la que interactúa el usuario final. Además, el cliente espera y recibe las respuestas del servidor. Esto es lo que lo convierte en una parte activa de la comunicación.

El servidor, que es el microservicio, recibe la solicitud del cliente y la procesa. Cuando se completa la solicitud, el microservicio envía una respuesta al cliente, que puede ser en forma de datos, el resultado de una operación o un código de estado para indicar si la solicitud finalizó correctamente o tuvo lugar algún error durante el proceso.

Es importante tener en cuenta que los microservicios se comunican entre sí a través de interfaces y protocolos estándar bien definidos. Esto significa que cada microservicio puede implementarse en diferentes lenguajes de programación y ejecutarse en diferentes servidores, siempre que sigan las reglas de comunicación establecidas. Esta flexibilidad permite que cada microservicio se adapte y crezca de forma independiente sin afectar a todo el sistema.

En definitiva, el modelo cliente-servidor en el contexto de los microservicios se basa en la comunicación entre el cliente y el microservicio a través de solicitudes y respuestas. Este enfoque modular y descentralizado permite crear sistemas flexibles y escalables en los que cada microservicio admite una funcionalidad específica y se comunica con otros microservicios para lograr la funcionalidad completa del sistema.

En nuestro caso hay que añadir un componente más que es la vista que es el componente que interactúa con el usuario final recibiendo las peticiones de este y a través del cliente se las redirigirá al servicio.

3.3.1.2 Servicio Mock

En el uso de microservicios Postman es una herramienta muy solicitada y no solo por ser muy útil a la hora de hacer pruebas y guardar peticiones de ejemplo de cómo funcionan los servicios sino también por otras funcionalidades que ofrece.

Una de estas funcionalidades es la posibilidad de crear servicios mock. Estos son servicios que se pueden publicar y tienen su propia url para acceder a ellos. Esta funcionalidad es muy útil porque permite crear versiones de prueba donde indicas la respuesta y así servir de respaldo temporal por ejemplo durante el desarrollo de un servicio.

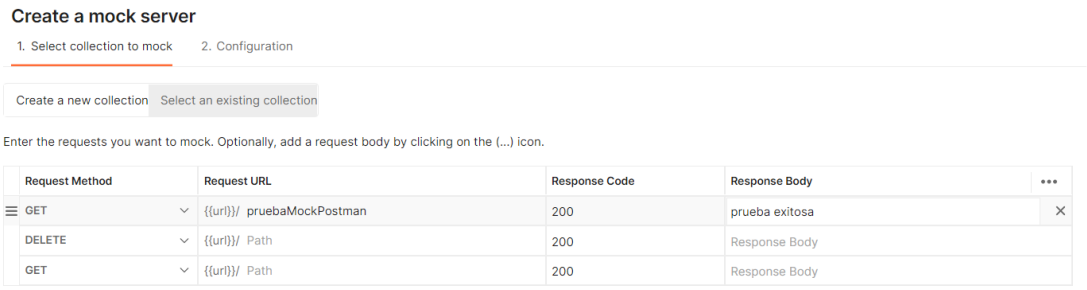

Ilustración 10 Ejemplo servicio mock Postman

Además, se puede asociar una colección al servicio mock y según se modifican los ejemplos de las peticiones variará la respuesta a devolver por el server.

3.4 Conexión microservicios con Kong

En este apartado se va a explicar cómo se debe usar Kong para que sea el encargado de recibir las peticiones y redirigirlas a los servicios.

Previo a crear los servicios en Kong es necesario que los servicios estén desplegados para que pueda existir conexión. Para ello se ha usado un servidor wildfly y se han desplegado los servicios como war. Se ha tomado esta decisión porque facilita la gestión de puertos en comparación con desplegarlos en jar.

Al estar en el servidor wildfly el puerto para todos los servicios es en el que está alojado el servidor por lo tonto solo se usa un puerto en cambio si desplegásemos los servicios como jar deberíamos tener un puerto asignado por cada servicio. No obstante, en algunos momentos para hacer pruebas y desplegar un servicio momentáneamente se ha hecho a través de su jar.

Desde Kong Manager se puede crear el servicio indicando una serie de parámetros. Los más relevantes son el host y path para apuntar a donde este desplegado el servicio, así como el protocolo. Adicionalmente se pueden añadir configuraciones como establecer tiempos máximo de conexión de lectura y/o de escritura.

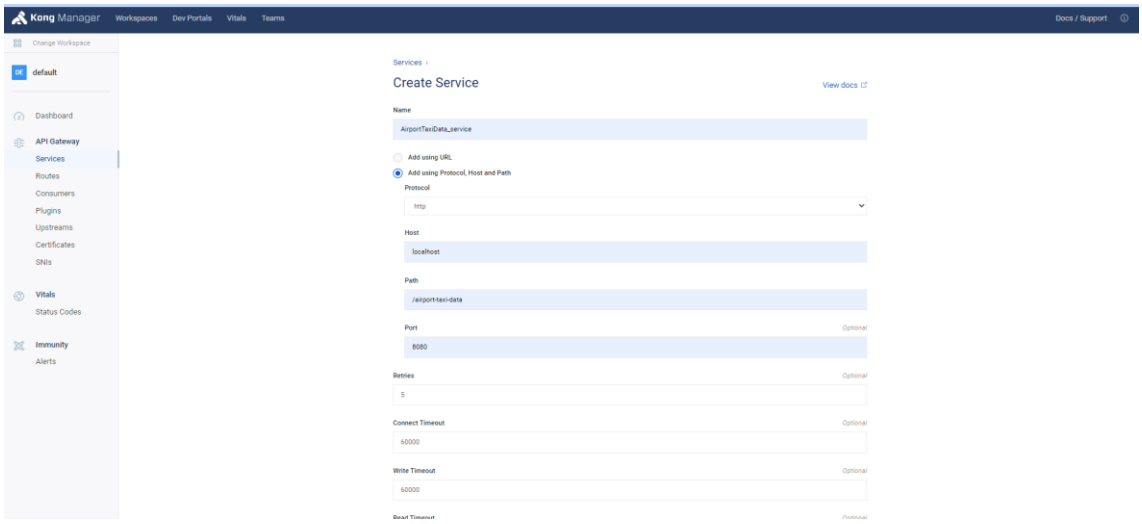

Ilustración 11 Ejemplo de cómo crear un servicio desde Kong Manager

Otra opción para crear el servicio es en vez de usando la interfaz que proporciona Kong hacerlo desde consola o incluso usando postman con una petición similar al siguiente.

- ```
  curl -i -X POST http://localhost:8001/services\
  --data name= AirportTaxiData_route \
  --data url='http://localhost:8080'
  ```

Se utiliza localhost porque al momento de hacer la petición estamos en la misma máquina, pero podría sustituirse por la dirección en la que está alojado el Kong otro aspecto es el puerto que el 8001 es el designado para funciones de administración de Kong.

Si el servicio se ha podido crear correcto se recibirá un mensaje de éxito y un código de operación 200 que significa creado.

Una vez tenemos el servicio el siguiente paso es crear una ruta. La ruta define la dirección en la cual Kong esta pendiente de recibir peticiones las cuales redirigirá al servicio correspondiente en cada caso.

Para crear la ruta al igual que el servicio tenemos las mismas dos opciones. En cuanto a usar el Kong Manager debemos rellenar los campos correspondientes al servicio que hace referencia, nombre de la ruta, Path y tipo operación u operaciones (GET, POST, DELETE …).

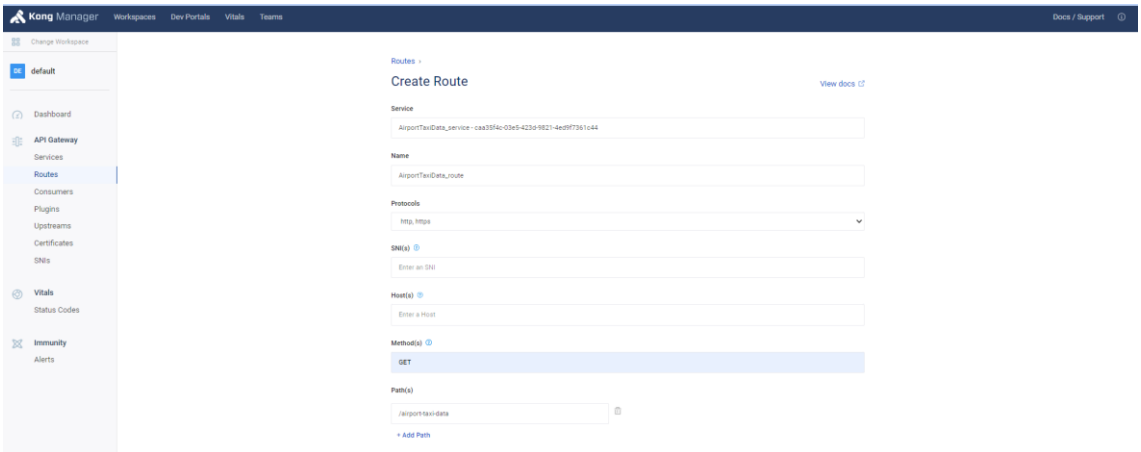

Ilustración 12 Ejemplo creación de ruta con Kong Manager

La petición para crear la ruta es muy similar a la de creación del servicio en este caso en el path debemos incluir el nombre del servicio al que vamos a asociar la ruta.
  ```

- ```
  curl -i -X POST
  http://localhost:8001/services/AirportTaxiData_service /routes\
  --data 'paths[]=/airport-taxi-data' \
  --data name=AirportTaxiData_route
  -- data 'methods[]=GET'
  ```

En este caso igual que en el anterior la petición en caso de haber sido realizada con éxito nos enviará una respuesta de código 201.

Una vez hemos completado la creación del servicio y la ruta para comprobar su funcionamiento bastaría con enviar una petición al servicio, pero en vez de hacerla a la dirección donde esta desplegado el servicio debemos hacerlo a través de Kong.

En el caso de que ambos estén en local algo que puede ocurrir debemos diferenciarlo apuntando al puerto en el que se encuentra Kong.

- http://localhost:8080/airport-taxi-data/available-date

  Esta petición seria directa al servicio ya que está apuntando al puerto en el que esta desplegado.

- http://192.168.25.46:8000/airport-taxi-data/available-date

  Esta petición seria a Kong ya que está apuntando al puerto en el que esta desplegado Kong que recibirá la petición y la redirigirá al servicio.

Para hacer la prueba se ha utilizado Postman porque así podemos guardar las peticiones y respuestas comprobar el código de respuesta y los headers. Además, en los casos en los cuáles las peticiones tienen parámetros, la gestión de estos es mucho más sencilla desde la aplicación. Sin embargo, se podría hacer directamente copiando la url en un navegador.
  ```

3.5 Implementación funcionalidades

En este apartado se procederá a hablar acerca de las funcionalidades que se han implementado en los servicios a través de plugins u otras opciones que ofrece Kong como Api Gateway.

En primer lugar, es necesario destacar la importancia de las funcionalidades ya que pueden ser clave a la hora de proporcionar ciertas características que facilitan la gestión y pueden ayudar a mejorar el rendimiento de los servicios en una organización.

Además, Kong específicamente tiene la ventaja frente a otros Api Gateway que permite aplicar las funcionalidades en caliente. Es decir, no es necesario reiniciarlo para que las funcionalidades se apliquen.

3.5.1 Balanceo

Es común que el tráfico hacia un servicio pueda ser mayor que el de otros e incluso en algunos casos ser muy alto. Para esto Kong ofrece la posibilidad de gestionar el equilibrio de carga distribuyendo las peticiones entre varias instancias del mismo servicio.

Kong Gateway se encarga de equilibrar la carga en ambos servidores, de modo que, si uno de ellos no está disponible, detecte automáticamente el problema y dirija todo el tráfico al servidor que funciona.

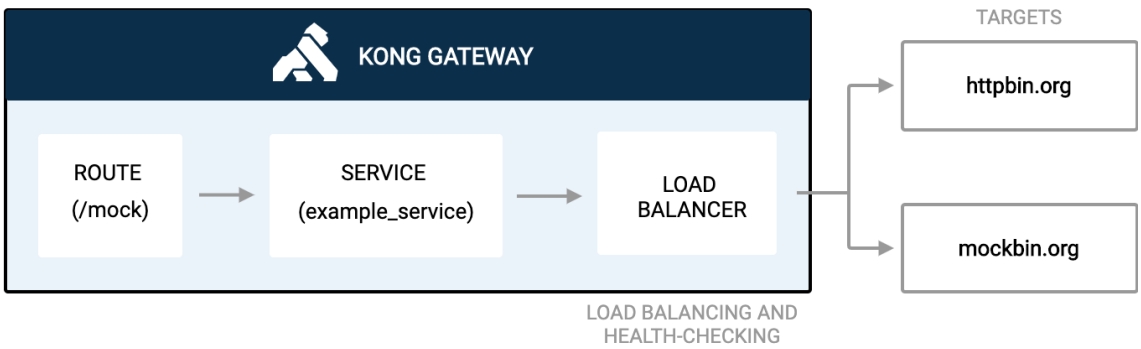

Ilustración 13 Esquema del balanceo [18]

En la ilustración 13 podemos ver un esquema de cómo funciona el balanceo. En este caso es un ejemplo sacado de la documentación de Kong donde httbin.org y mockbin.org son instancias de este backend ejecutándose en diferentes sistemas de host.

En nuestro caso se ha decidido por desplegar dos instancias del mismo servicio a través de un jar y cada instancia en un puerto diferente. De esta forma podemos acceder a ambos a través de localhost y diferenciarlos por el puerto.

En primer lugar, debemos modificar el servicio en Kong para que, en vez de apuntar al servicio desplegado directamente, apunte al upstream que es el responsable de balancear las cargas de tráfico entre las instancias.

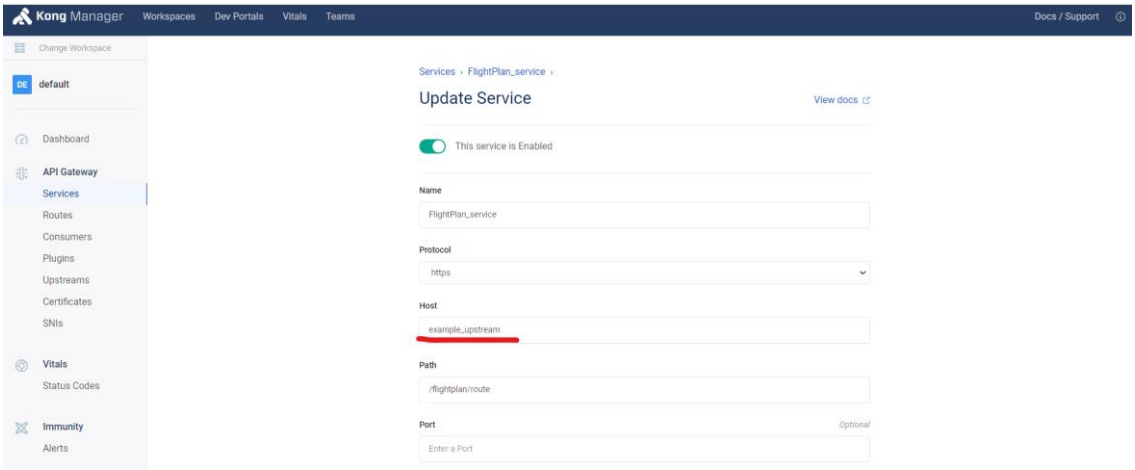

Ilustración 14 Modificación del servicio para el balanceo

Es importante remarcar que no solo se debe cambiar el host del servicio al upstream, el otro punto importante es asegurarse de no definir un puerto. Si definimos un puerto estaríamos forzando a que el upstream utilizase la instancia que se encuentre en ese puerto y eso impediría el equilibrio de carga.

Por "upstream" se entiende las aplicaciones de servicio situadas detrás de Kong Gateway a las que se reenvían las peticiones del cliente. En Kong Gateway, un upstream representa un host virtual y se puede utilizar para comprobar el estado, interrumpir el circuito y equilibrar la carga de las solicitudes entrantes en varios servicios backned de destino.

Un upstream también incluye un comprobador de salud, que es capaz de habilitar y deshabilitar objetivos basándose en su capacidad o incapacidad para resolver peticiones. La configuración del comprobador de estado se almacena en el objeto upstream y se aplica a todos sus objetivos.

Una vez hemos cambiado el host deberíamos configurar el upstream. Para ello se a hecho a través de Kong Manager por resultar de mayor facilidad que hacerlo por peticiones de tipo curl.

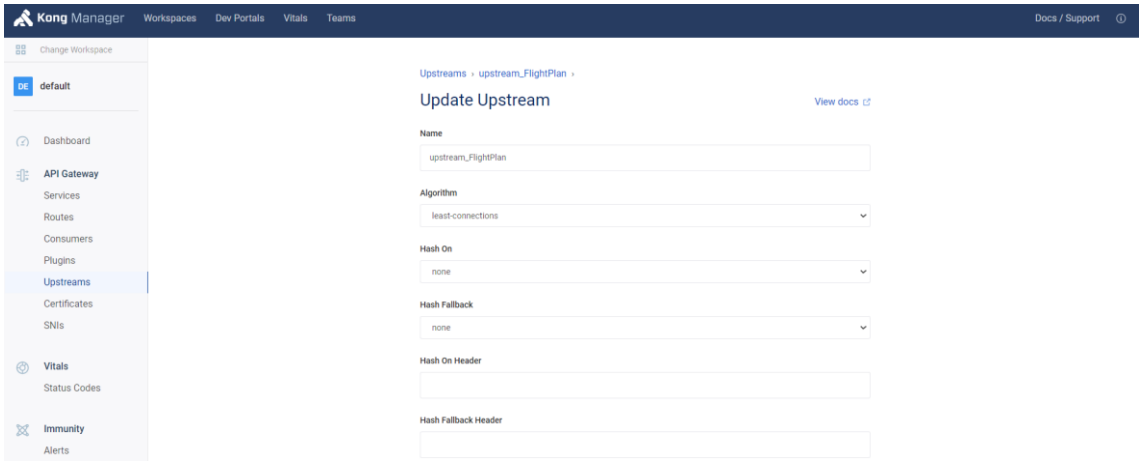

Ilustración 15 Creación upstream

En la creación del upstream debemos elegir el algoritmo de balanceo de carga que queramos. Las opciones son round-robin, lest-connections y conistent-hashing.

El algoritmo round-robin es el más usado y el seleccionado por defecto por Kong. Este algoritmo se caracteriza por ser fácil de entender e implementar. Su funcionamiento consiste en para balancear la carga alternar los servidores. Es decir, si tienes dos servidores esperando solicitudes cuando lleguen peticiones la primera irá a un servidor y la segunda, al contrario.

Sin embargo, este algoritmo tiene un claro inconveniente. En el caso en que los servidores no tuviesen las mismas especificaciones como capacidad de RAM o CPU, el algoritmo no tendría en cuenta eso y el servidor con menos capacidad podría verse colapsado.

Por otro lado, el algoritmo least-connection no tiene la desventaja que tiene el anterior. Este algoritmo tiene en cuenta el número de conexiones actuales a cada servidor y cuando llega una petición se le asignará al servidor con menor número de conexiones en ese momento.

Por último, encontramos el consisten-hashing este algoritmo consiste en que a cada servidor se le asignan varios valores hash basándose en el ID del servicio, y cada petición se asigna a un servidor en función del valor del hash. Esto hace que una petición que se realiza dos veces vaya al mismo servidor. Esto es una gran ventaja si se está guardando cache de las peticiones porque se optimizaría el espacio de guardado de cache porque no tendría que guardarse la misma petición en los distintos servidores debido a que si aumenta el número de

servidores una gran parte de peticiones seguirían perteneciendo al mismo servidor que antes.

Una vez estudiadas las posibilidades se optó por el lest-connection porque, aunque se podría optar por round-robin ya que los servidores van a tener las mismas especificaciones, las peticiones no son homogéneamente complejas por lo que unas llevarán más tiempo que otras. Por otra parte, se ha descartado el consisten-hashing ya que es más complejo y su uso está pensado para sistemas qué van a escalar en el número de servidores lo cual no es nuestro caso.

Una vez decidido el algoritmo de control de carga y creado el upstream debemos definir los objetivos o servidores a los que va a redirigir el upstream. En nuestro caso como se mencionó anteriormente van a ser dos instancias del mismo servicio desplegados en jar en distintos puertos.

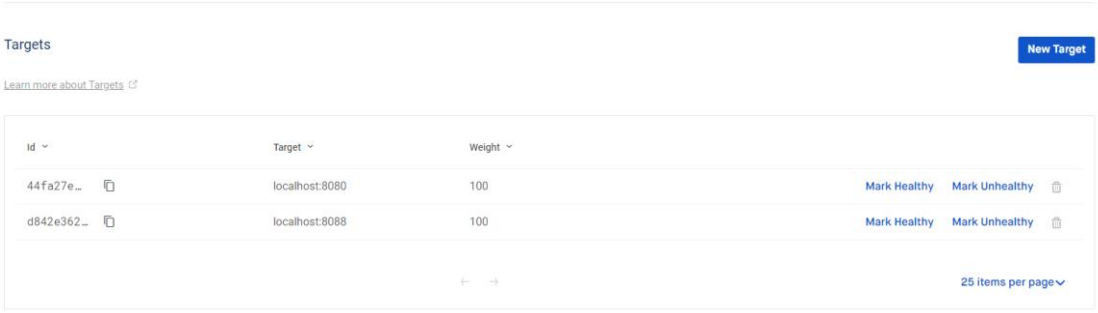

Ilustración 16 Definición de servidores destino del upstream

Se debe definir donde están alojados los servicios en este caso localhost:8080 y localhost:8088. El número de conexiones máximo que acepta asignado en la variable denominada weight en este caso se han puesto cien, aunque no se espera que llegue nunca a esa cantidad.

3.5.2 Flujos entre microservicios

Los flujos entre microservicios era una funcionalidad que se quería implementar con el objetivo de poder agilizar las consultas. El objetivo que se buscaba era que de la salida de un servicio se pudiesen usar los valores como parámetros de entrada de otro.

Esto es posible a través del Api-Gateway de WSO2 donde se puede implementar una funcionalidad denominada como chaining-service[19] la cual es a grandes rasgos lo que se ha explicado antes que buscábamos conseguir con los flujos.

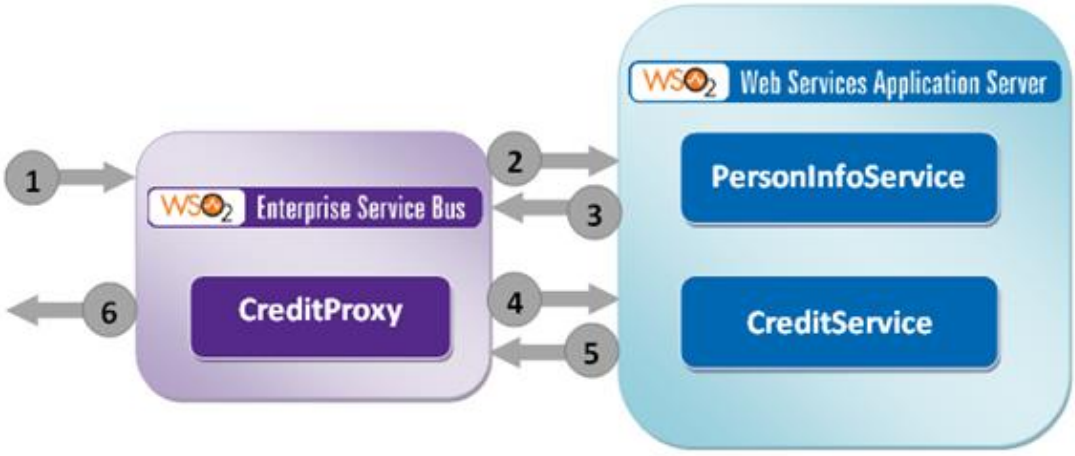

Ilustración 17 Chaining services WSO2 [20]

En la ilustración 17 podemos ver un ejemplo de cómo funcionaría el flujo entre servicios de WSO2.

- **CreditProxy** Este servicio está en WSO2 ESB. Recibe un ID y una cantidad de crédito y hace peticiones a los otros servicios.

- **PersonInfoService** Este servicio devuelve el nombre y dirección asociado a un ID. Es el primer servicio llamado por CreditProxy.

- **CreditService** Este servicio es el último llamado por CreditProxy con la información recibida de PersonInfoService

Lamentablemente, aunque se tuvo como opción WSO2 no fue la elección para llevar a cabo nuestro Api-Gateway. La razón fue que solo ofrecía de forma gratuita una versión de prueba de 90 días lo que era tiempo algo justo para la realización del TFG. Además, no entraba esta funcionalidad ya que sería necesario el WSO2 Enterprise Service Bus el cual requiere de una suscripción

Sabiendo que existía la posibilidad de encadenar servicios, aunque Kong no la ofreciese se han buscado alternativas para cumplir con el objetivo. Una opción sería por cada encadenación de servicios que se quisiese hacer crear un nuevo servicio en java que se encargase de hacer llamadas a estos y gestionar las respuestas y llamadas. Esto fue descartado porque es una solución que para

un flujo entre servicios puede servir, pero si se quieren crear varios la complejidad aumentaría y no aporta grandes ventajas respecto a llamar individualmente a cada uno.

Finalmente, se encontró una alternativa que nos resultó satisfactoria y la cual a futuro puede mejorar. Esta solución es a través de Postman el cual ofrece una funcionalidad aún en desarrollo denominada “Flows” que permite encadenar peticiones.

Flows es una interfaz visual de desarrollo de aplicaciones centrada en las API. Le proporciona un lienzo infinito para organizar y conectar APIs.

Se nutre de la red de API públicas Postman y de sus colecciones. Flows proporciona capacidades de manipulación de datos para tender puentes entre sus API y visualización de datos para mostrar el resultado final de sus flujos.

En nuestro caso se ha realizado un Flow que consiste en una primera petición que devuelve todos los aeropuertos nacionales. Con cada aeropuerto se hace una segunda petición a otro servicio que calcula las aproximaciones frustradas de los vuelos de cada aeropuerto.

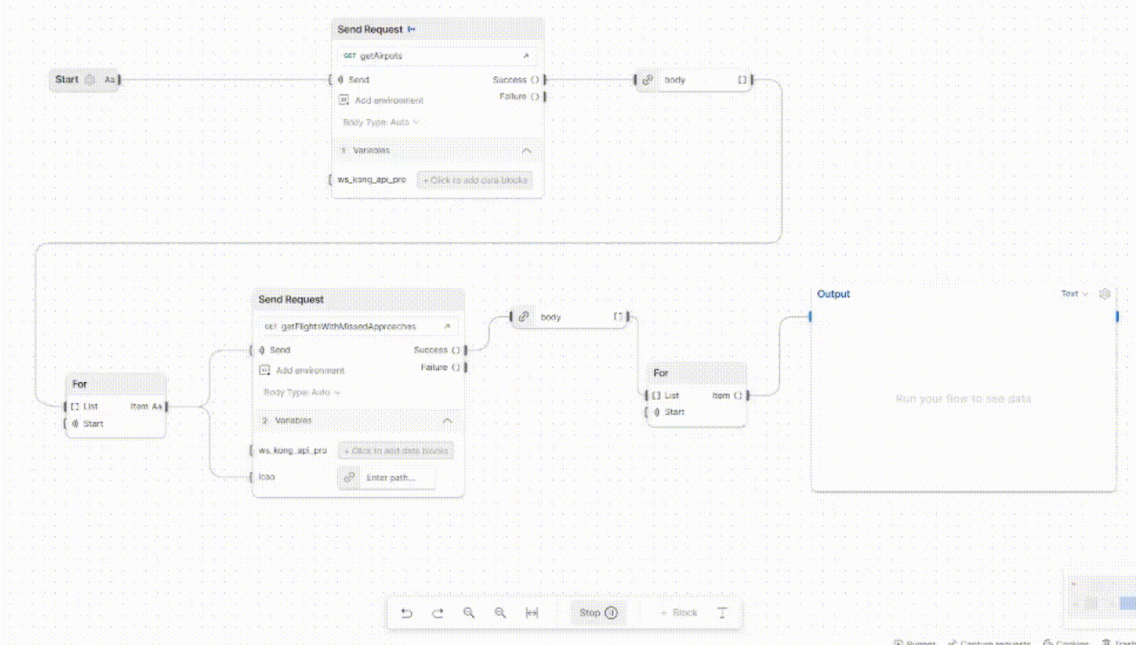

Ilustración 18 Gif Muestra flow de Postman

La idea es a futuro poder desplegar el Flow para poder acceder a él de forma remota. Sin embargo, esa función aún está en fase beta y de momento no

funciona correctamente ya que no devuelve el output sino un mensaje de que está funcionando el Flow.

Ilustración 19 Llamada al flow desplegado

3.5.3 Autenticación

En primer lugar, la seguridad es uno de los elementos más importantes a la hora de gestionar microservicios. Kong ofrece una serie de plugins que ayudan a proteger los servicios y dar acceso solo a aquellos usuarios autorizados.

Aunque la seguridad, como se ha mencionado anteriormente, es uno de los elementos más importantes, no se ha implementado a través del Kong hasta el final. Esto es debido a que la seguridad la definíamos a través de servicios a partir de una funcionalidad de SpringBoot que habilitaba la posibilidad de proteger los servicios a través de un Api-Key obligando a los usuarios a autenticarse antes de hacer una petición.

Kong nos ofrece varias alternativas para gestionar la autenticación de nuestros servicios. Nosotros nos hemos centrado en el Api-Key que ya estaba implementado en los servicios y en autenticación con tokens JWT porque se considera una mejor opción que usar autenticación por Api-Key.

3.5.3.1 Api-Key

Un Api-Key es una clave que restringe el acceso a una API. Es un conjunto de caracteres representados en hexadecimal.

Para poder usar la autenticación por Api-Key en nuestros servicios es necesario especificarlo en el yml que usamos para generar el cliente y el servicio.

```
security:
- APIKeyHeader: []
```

Ilustración 21 Definir seguridad por Api-Key

```
components:
  securitySchemes:
    APIKeyHeader:
      type: apiKey
      name: X-API-Key
      in: header
```

Ilustración 20 Definir seguridad por Api-Key en yml

En el caso de establecer la autenticación por Kong, este ofrece un plugin denominado key-auth que podemos usar para establecer este tipo de autenticación.

A la hora de definir el plugin de Kong tenemos varias opciones. En primer lugar, si queremos definir el plugin de forma global que afecte a todos los servicios o se puede especificar para un servicio o ruta en concreto.

Por otro lado, podemos definir la forma de pasar la key hemos seleccionado vía header y por query. El último campo relevante es el Config.Key Names en este definimos los nombres de las keys que autorizan a este servicio.

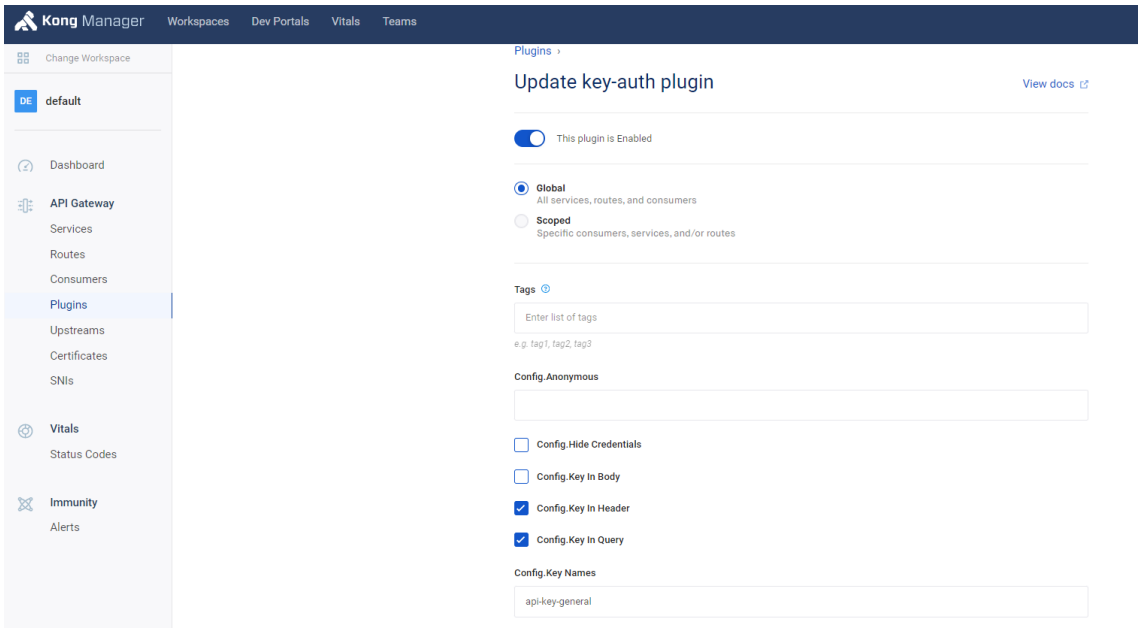

Ilustración 22 Definir plugin key-auth

A parte de definir el plugin es necesario crear un consumer para asociarlo con un api-key. El consumer es el elemento de Kong que contiene las claves de autenticación.

Una vez tenemos creado el consumer es el momento de añadir las credenciales del api-key al consumer.

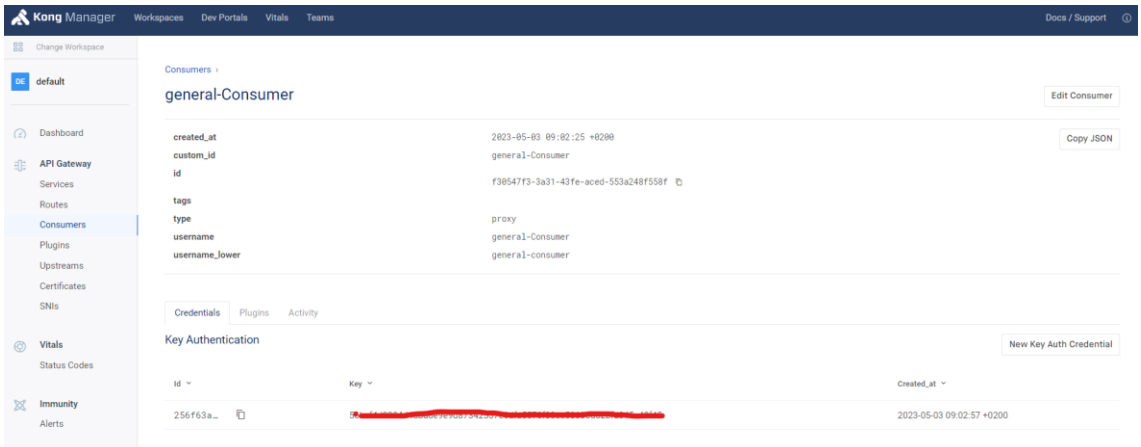

Ilustración 23 Consumer y credenciales key-auth

En general, las claves de API no se consideran seguras, ya que suelen ser accesibles para los clientes, lo que facilita que alguien pueda robarlas. Una vez que se roba la clave, no tiene vencimiento, por lo que se puede usar de forma indefinida, a menos que el propietario del proyecto revoque o vuelva a generar la clave. Por esto era necesario buscar otra opción a la de key-auth se ha optado por los JWT que son tokens que se envían cifrados y tienen un tiempo de

expiración. Además, los tokens no tienen por qué ser conocidos por el usuario lo que soluciona los inconvenientes del Api-Key.

3.5.3.2 JWT

JSON Web Token (JWT) [21] es un estándar abierto (RFC 7519) que define una forma compacta y autocontenida de transmitir información de forma segura entre las partes como un objeto JSON. Esta información puede ser verificada y fiable porque está firmada digitalmente. Los JWT pueden firmarse utilizando un secreto (con el algoritmo HMAC) o un par de claves pública/privada utilizando RSA o ECDSA.

En Kong se ofrece un plugin para implementar este tipo de autenticación. Para configurarlo debemos, al igual que con key-auth, seleccionar si queremos que vaya asociado a un único servicio o ruta o de forma global.

Se puede definir el tiempo de expiración haciendo que Kong rechace los tokens que tienen más de ese tiempo de vida. También se define el nombre con el que se identifica la clave por defecto es iss.

Después de configurar el plugin se deben crear las credenciales de jwt en el consumer. Se optó por dejar en blanco los parámetros de key y secret que se usaran para que los genere de forma automática Kong.

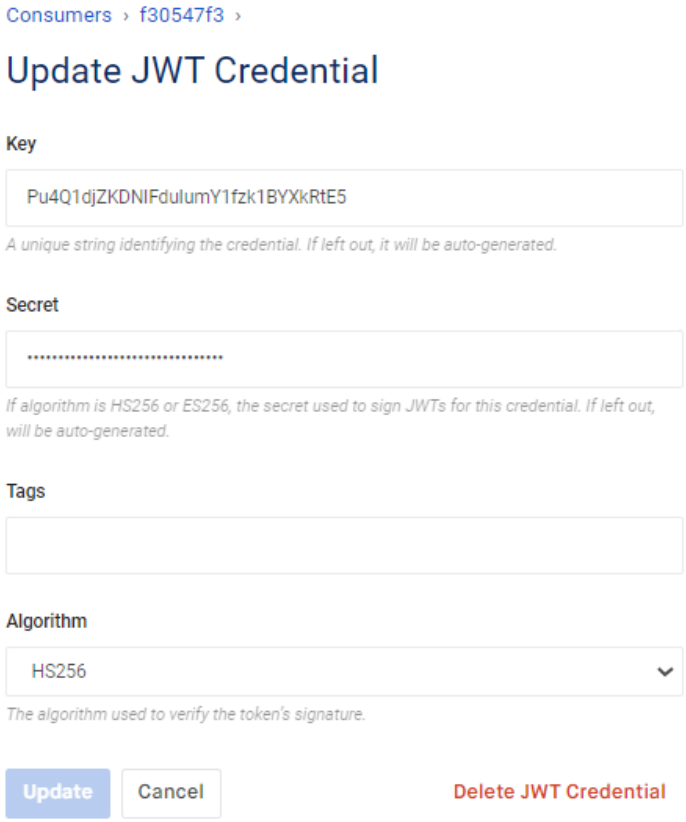

Ilustración 24 Credenciales JWT para un consumer

Para poder conocer el secret si lo hemos generado de forma automática a través de Kong debemos hacer una petición a Kong a través del puerto de administración para conocerlo.

- `curl -X GET http://localhost:8001/consumers/{consumer}/jwt`

Conociendo el secret y el id podemos generar un jwt para acceder al servicio. En la página de jwt [22] podemos generar el nuestro para hacer pruebas a través de Postman. Es importante en campo exp definir el tiempo de expiración como

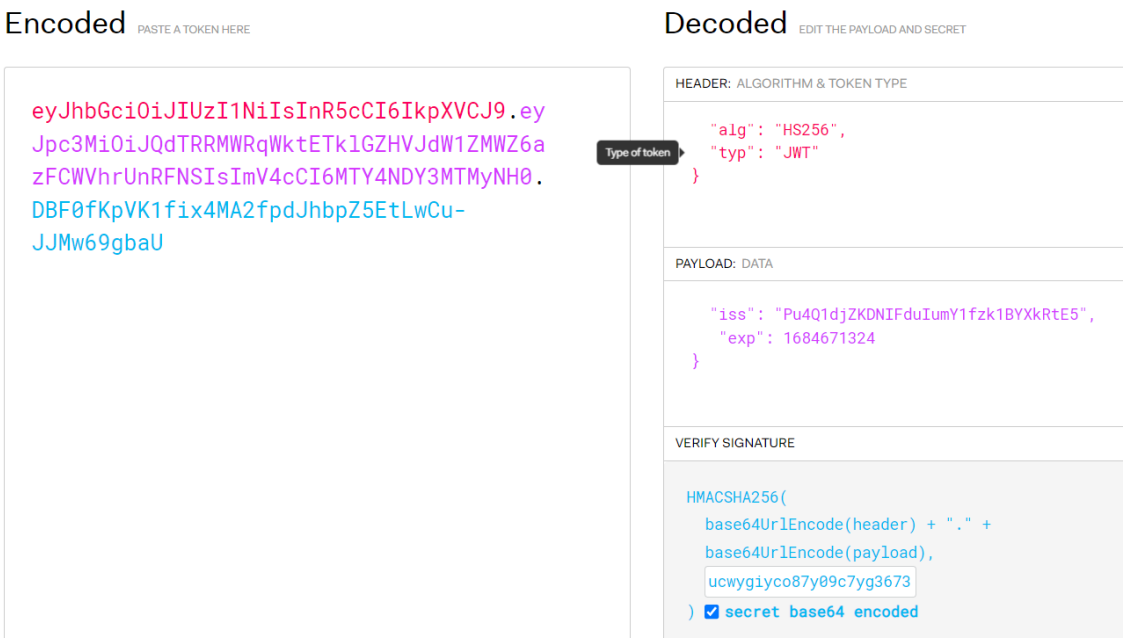

Ilustración 25 Generar un jwt token

la hora en milisegundos en la que caducará el token.

Eso puede servir para hacer pruebas, pero de cara al uso en una aplicación no es posible generar el token de este modo. Una opción sería generar un token externamente y que la aplicación lo almacenase para las llamadas a los servicios sin embargo esto sería demasiado similar al api-key.

La opción por la cual se decidió fue usar una de las librerías[23] que ofrece jwt para la generación de los tokens desde java. Uno de los problemas a abordar con los estos tokens es la caducidad del mismo. Las opciones que teníamos eran controlar si ha expirado a través de gestionar la excepción que salta al intentar autorizarse con un token caducado. Sin embargo, este método no fue considerado el óptimo ya que el código de error para un token expirado y uno invalido es el mismo. Por esto se decidió por guardar el tiempo de duración del token y comprobar antes de cada petición si ese tiempo había transcurrido.

Para la creación del token a través de la librería y comprobar si se había pasado el tiempo de duración del token se creó un método el cual debe ser llamado antes de cada petición y genera un token nuevo cuando sea necesario.

```
private static void isTokenExpired() {
    if (dateExp == null || new Date(System.currentTimeMillis()).after(dateExp)) {
        Mac mac;
        try {
            mac = Mac.getInstance("HmacSHA256");
            SecretKeySpec secretKeySpec = new SecretKeySpec(PROPERTIES.getProperty("secret").
                    getBytes("UTF-8"),"HmacSHA256");
            mac.init(secretKeySpec);
            dateExp = new Date(System.currentTimeMillis() + 300000);
            System.out.print(dateExp.toString());
            Claims claims = Jwts.claims().setExpiration(dateExp).setIssuer(PROPERTIES.getProperty("iss"));
            String jwt = Jwts.builder().setClaims(claims).signWith(secretKeySpec, SignatureAlgorithm.HS256).compact();
            API_airportTaxi.getApiClient().setAccessToken(jwt);
        } catch (InvalidKeyException | NoSuchAlgorithmException | UnsupportedEncodingException e) {
            e.printStackTrace();

    }
```

Ilustración 26 Método gestión tokens

3.5.4 Cache

La cache puede ser esencial en algunos contextos en los que se requiere una respuesta rápida. Además, ayuda al rendimiento y escalabilidad de los servicios. Sin embargo, no en todos los servicios puede ser lo mejor ya que algunos lo que se busca es que la respuesta se recalcule ya que puede cambiar porque esta varie con el tiempo.

En nuestro caso se han querido cachear unos servicios que la duración de las consultas es demasiado larga. En concreto se han decidido cachear ciertas consultas que sacan datos de un mes. Esto es así porque el uso que se da a estos servicios es desde una aplicación que debe mostrar unas gráficas y debe recibir los datos de manera rápida.

Kong ofrece funcionalidades de caché, vamos a usar el plugin proxy-cache ya que el resto están reservados para la versión Enterprise o Plus de Kong que requieren una suscripción. Este plugin permite almacenar en caché las respuestas de los microservicios. Esto significa que, en lugar de procesar cada solicitud desde cero y obtener la respuesta directamente del microservicio, Kong puede almacenar temporalmente las respuestas en la caché y servirlas directamente a las solicitudes futuras que sean idénticas o similares.

La cache de Kong tiene un espacio limitado de 128 MB, aunque se pude aumentar modificando el archivo de configuración. Esta limitación ha hecho que se seleccionen los servicios a cachear y más concretamente las consultas.

Debido a querer solo cachear ciertas consultas se ha optado por configurar el plugin para que se cache cuando se usa un consumer concreto. Para ello se ha

creado un consumer con unas credenciales distintas que solo deben ser usadas con las consultas que se quieren cachear.

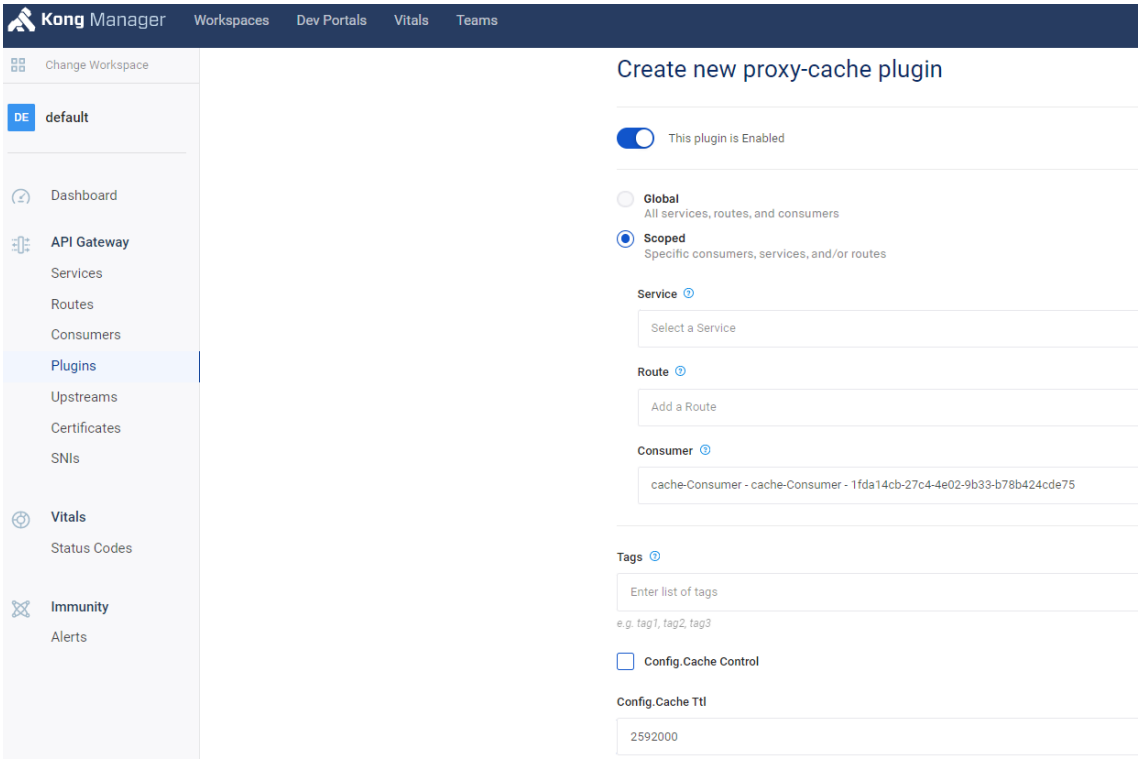

Ilustración 27 Configuración plugin de cache

Las opciones de configuración interesantes son cache ttl que define el tiempo de vida de la cache y storage ttl que define el tiempo que permanece almacenada la memoria en cache. Ambos parámetros se definen por un valor en milisegundos en este caso se ha puesto el valor un mes en segundos.

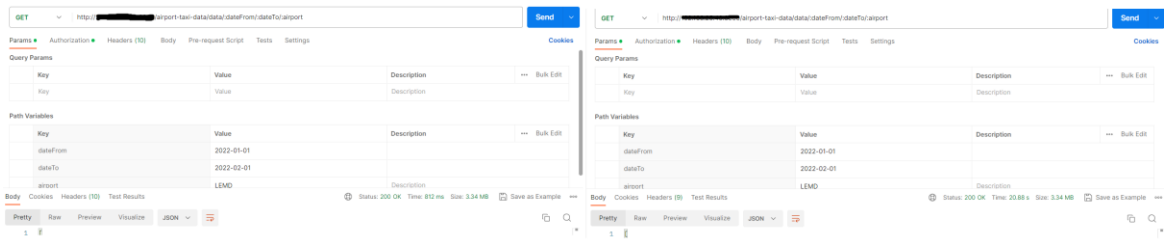

Ilustración 28 Comparativa petición con cache y sin cache

En la ilustración anterior podemos ver que la petición cacheada(derecha) dura menos de un segundo mientras que la otra requiere 20 segundos. Por lo cual el tiempo ganado con la cache es muy significativo.

Una vez con la cache configurada nos centramos en una forma de poder renovar cache sin tener que esperar a que esta caducase tras el mes que está puesto como tiempo de guardado.

Para borrar cache Kong ofrece un endpoint a través del cual especificando el id de la cache se puede borrar. Este id se devuelve como header al hacer una consulta y se llama X-Cache-key.

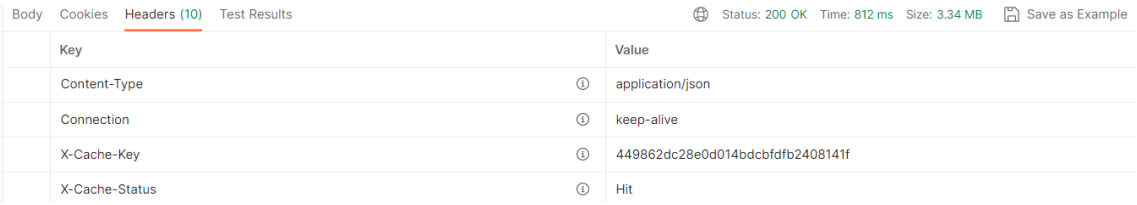

Ilustración 29 Headers devueltos en una consulta cacheada

Para poder borrar la cache se ha creado un Flow de Postman el cual hace una llamada a la consulta de la cual se quiere borrar de cache, se recoge su X-Cache-key y se pasa a la petición de borrado.

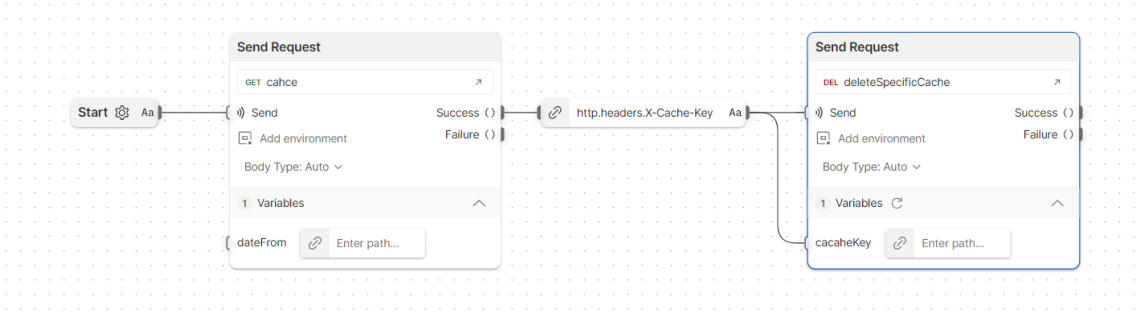

Ilustración 30 Flow borrar cache

Partiendo de este Flow se ha creado otro para renovar la cache que consiste en hacer lo mismo que el anterior y al final añadir la consulta para que quede cacheada de nuevo.

4 Análisis de resultados

En este apartado se van a presentar una vista general de los resultados y se va a hacer un análisis de estos.

En primer lugar, se ha conseguido desplegar la herramienta de construcción del Api Gateway que elegimos, Kong. Esto era el paso básico para poder seguir desarrollando el proyecto y cumplir el resto de los objetivos.

En cuanto a la creación de servicios se ha podido crear una metodología basada en usar un yml como definición del mismo la cual permite que todos los servicios creados con este archivo y el jar proveniente de OpenApi tengan la misma estructura. Esto es muy útil para cuando se tienen muchos servicios y en ámbitos donde hay varios desarrolladores ya que en el caso de tener que modificar alguno todos estén familiarizados con la estructura y funcionamiento de todos los servicios ya que deben ser iguales.

Centrándonos en las funcionalidades, el balanceo funciona como se esperaba repartiendo la carga de trabajo entre las instancias del servicio que se encuentren desplegadas.

Por otra parte, la autenticación mediante api-key resulta mucho más sencilla tanto de configurar como luego usarla en las llamadas a los servicios; sin embargo, se considera mucho mejor opción el uso de JWT ya que el usuario no conoce el token y además estos expiran no como la key del key-auth.

En cuanto a la cache no se ha logrado aplicar a los servicios que la requerían sin embargo el método para renovar cache antes de que esta expire no es el óptimo. Podría haberse conseguido una mejor solución si se pudiese forzar a refrescar a la cache a través de headers en la petición.

Por último, la aplicación de flujos de los microservicios se ha conseguido llevar a cabo con un buen resultado que permite agilizar las peticiones desde Postman y que tienen margen de mejora ya que la herramienta de creación de flujos de Postman está en pleno desarrollo.

5 Conclusiones

En este apartado se van a presentar una vista general de los resultados y conclusiones basándonos en los objetivos propuestos para el proyecto.

En primer lugar, resaltar el aprendizaje durante de la realización del proyecto ya que se partía de una base muy pequeña de conocimiento en la cual no siquiera se tenía conocimiento concreto acerca de que es un Api-Gateway y su funcionamiento.

Sin embargo, a pesar de contar con una base tan pobre se ha logrado construir un Api-Gateway completamente funcional partiendo desde el inicio de este y viendo distintas alternativas. Es necesario destacar la importancia del software libre ya que gracias a él se ha podido realizar este proyecto y de las versiones de gratuitas de software de pago como en el caso de Kong.

En cuanto a la elección de Api-Gateway se considera una desventaja el hecho de no haber podido probar WSO2 porque tiene funcionalidades como la encadenación de servicios muy interesantes.

En cuanto a la creación de los microservicios se considera que se ha encontrado una solución que consideramos la óptima ya que logra agilizar su creación y proporciona consistencia entre el servidor y el cliente.

Si nos fijamos en la aplicación de funcionalidades se considera que se ha conseguido aplicar las más útiles e interesantes, aunque en el caso del flujo entre microservicios no se ha conseguido tal y como se planteaba en un principio, pero el resultado final ha sido satisfactorio.

Concretamente la configuración del balanceo de carga no resulta muy útil en este proyecto ya que no se espera un gran número de llamadas a los servicios; no obstante, el conocer esta posibilidad y como llevarla es muy interesante y puede ser útil en proyectos de mayor envergadura.

Un aspecto interesante a futuro sería en el caso de la cache poder definir nuestro propio plugin usando lua para así poder tener configuraciones más específicas a nuestro proyecto. Un objetivo podría ser el de conseguir un mejor método de renovar cache o poder especificar en la configuración del plugin que consultas cachear. Así como una vez la cache estuviese funcionando poder tener un endpoint que retorne las consultas que se encuentran almacenadas en ese momento.

En un futuro sería interesante ver cómo evoluciona Apisix que en los últimos meses si ha ampliado su documentación que la escasez de esta fue uno de los motivos para su rechazo. Aún se encuentra en etapas tempranas de desarrollo, está en la versión 0.4.0 pero quizás en un futuro sea una mejor alternativa a Kong principalmente por tener todas sus funcionalidades gratis.

En conclusión, se han conseguido alcanzar los objetivos y los resultados han sido satisfactorios a pesar de que en algunos casos se han tenido que optar por soluciones alternativas a las que se tenía en mente al comienzo del proyecto.

6 Análisis de Impacto

En este capítulo se expone un análisis potencial del impacto de los resultados obtenidos por el desarrollo del presente Trabajo de Fin de Grado. Podemos diferenciar el impacto en el ámbito personal, en el empresarial y haciendo referencia a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

En primer lugar, a nivel personal ha consolidado conocimientos adquiridos durante la carrera, como la arquitectura de microservicios del mismo modo que se han ampliado al descubrir nuevas tecnologías y formas de programación. Por otra parte, también ha reforzado ciertas cualidades, como una mejor organización o el aprendizaje autodidacta de nuevas tecnologías que, seguramente, me serán útiles en un futuro.

Por otra parte, a nivel de la empresa esto supone un punto de partida para la construcción de un Api-Gateway. Concretamente en la empresa en la que me encuentro ahora mismo se está empezando a implementar funcionalidades en base a lo que se ha ido haciendo en este TFG.

En cuanto a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 este proyecto está relacionado con aspectos de las tecnologías de la información. Por esto se puede relacionar con el objetivo 9(Industria innovación e infraestructuras) que tiene como una de sus metas “Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo”. En este contexto este proyecto puede ser útil ya que en él todas las herramientas utilizadas son de software libre y son herramientas muy utilizadas en escenarios de I+D+I.

7 Bibliografía

[1] *IBM Documentation*. (s. f.). https://www.ibm.com/docs/es/was/9.0.5?topic=services-web

[2] *¿Qué es API Gateway?* (s. f.). TIBCO Software. https://www.tibco.com/es/reference-center/what-is-an-api-gateway

[3] *Amazon API Gateway | API Management | Amazon Web Services*. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/api-gateway/

[4] *Tecnología para desarrolladores web | MDN*. (s. f.). https://developer.mozilla.org/es/docs/Web

[5] *Spelluru. (2022, 7 diciembre). Guía del protocolo AMQP 1.0 en Azure Service Bus y Event Hubs - Azure Service Bus. Microsoft Learn.* https://learn.microsoft.com/es-es/azure/service-bus-messaging/service-bus-amqp-protocol-guide

[6] *¿Qué es una API? - Explicación de interfaz de programación de aplicaciones - AWS*. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/api/

[7] *API Features Individualizing of Web Services: REST and SOAP. (2019). International journal of innovative technology and exploring engineering*, *8*(9S), 664-671. https://doi.org/10.35940/ijitee.i1107.0789s19

[8] *Software Java*. (s. f.). Oracle España. https://www.oracle.com/es/java/

[9] Nov, M. M. (s. f.). *About the Eclipse Foundation | The Eclipse Foundation.* https://www.eclipse.org/

[10] *Deploying Spring Boot Applications*. (2014, 7 marzo). Deploying Spring Boot Applications. https://spring.io/blog/2014/03/07/deploying-spring-boot-applications

[11] Porter, B. (s. f.). *Maven – Welcome to Apache Maven.* https://maven.apache.org/

[12] *WildFly Documentation*. (s. f.). https://docs.wildfly.org/14/

[13] *Postman*. (s. f.). https://www.postman.com/

[14] Kong Inc. (2023, 17 abril). *The Cloud Native API Management Platform | Kong Inc.* https://konghq.com/

[15] *The #1 open-source platform for building and managing APIs with ease*. (s. f.). https://wso2.com/es/api-manager/

[16] *Apache APISIX® - Cloud-Native API Gateway*. (2022, 9 agosto). https://apisix.apache.org/

[17] OpenAPI Generator. (s. f.). https://openapi-generator.tech/

[18] Imagen de la documentación de Kong https://docs.konghq.com/assets/images/docs/getting-started-guide/upstream-targets.png

[19] *Service Chaining Example - Enterprise Service Bus 4.9.0 - WSO2 Documentation*. (s. f.). https://docs.wso2.com/display/ESB490/Service+Chaining+Example

[20] Imagen de la documentación de WSO2 https://docs.wso2.com/download/attachments/38472701/327.png?version=1&modificationDate=1401835014000&api=v2

[21] auth0.com. (s. f.-b). JWT.IO - JSON Web Tokens Introduction. JSON Web Tokens - jwt.io. https://jwt.io/introduction

[22] auth0.com. (s. f.-a). JWT.IO. JSON Web Tokens - jwt.io. https://jwt.io/#debugger-io

[23] auth0.com. (s. f.). JWT.IO - JSON Web Tokens Libraries. JSON Web Tokens - jwt.io. https://jwt.io/libraries?language=Java

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue May 30 19:52:07 CEST 2023
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)