Universitat Oberta de Catalunya

uoc.edu

Protección de APIs REST

Eduardo San Juan Castellanos
Master Universitario en Ciberseguridad y Privacidad
Seguridad empresarial

Pau del Canto Rodrigo
Víctor García Font

06/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Protección de APIs REST*
Nombre del autor:	*Eduardo San Juan Castellanos*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2022
Titulación:	*Master Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Seguridad empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Seguridad* *REST* *Gateway*

Resumen del Trabajo (máximo 250 palabras):

Actualmente el uso de APIs REST por prácticamente todos los servicios y sistemas modernos es fundamental para proporcionar funcionalidades avanzadas, personalizadas y modernas a los usuarios, ya sea en sus dispositivos móviles o en cualquier página web de comercio, blog o entretenimiento, entre muchas otras.

Securizar estas APIs es fundamental para garantizar que los datos de los usuarios no se ven afectados, la confidencialidad de ciertos datos que podrían ser sensibles, así como para cumplir con el reglamento de protección de datos de la Unión Europea.

Este trabajo analiza las amenazas y vulnerabilidades más prevalentes en las APIs REST y los mecanismos de seguridad para darles protección. Un API Gateway es una solución de seguridad que permite gestionar, configurar y enrutar las peticiones entrantes hacia las APIs expuestas en los diferentes *backends* del entorno corporativo. Es por ello que el trabajo se completa con una parte práctica en la que se muestra un ejemplo de despliegue de una solución de API Gateway basada en el producto KONG, para proteger una API REST con los mecanismos analizados en la parte teórica.

Abstract (in English, 250 words or less):

Nowadays, the use of REST APIs by practically all modern services and systems is essential to provide advanced, personalised and modern functionalities to users, whether on their mobile devices or on any commercial, blog or entertainment website, among many others.

Securing these APIs is essential to ensure that users' data is not affected, the confidentiality of certain data that could be sensitive, as well as to comply with the European Union's data protection regulation.

This paper analyses the most prevalent threats and vulnerabilities in REST APIs and the security mechanisms to protect them. An API Gateway is a security solution that allows managing, configuring and routing incoming requests to the APIs exposed in the different backends of the corporate environment. That is why the work is completed with a practical part in which an example of the deployment of an API Gateway solution based on the KONG product is shown, to protect a REST API with the mechanisms analysed in the theoretical part.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Actualmente las APIs se pueden encontrar en cualquier servicio o producto digital. Prácticamente todas las aplicaciones de nuestro *Smartphone* hacen uso de APIs externas, ya sea para descargar contenido alojado en un servidor, para actualizar las noticias que se nos muestran, para enviar o recibir notificaciones o para subir contenido a redes sociales.
Por otra parte, cualquier página web que consultemos online también hace uso de múltiples API, así como herramientas corporativas o elementos de las Intranet corporativas de las diferentes empresas.

Hoy en día, cuando se habla de API, generalmente se tiende a interpretar esto como hablar de API REST. Sin embargo, una API (*Application Programming Interface*) no deja de ser un conjunto de funciones o subrutinas que un software expone en forma de bibliotecas para que otro software tercero haga uso de ellas. Esto da lugar a un procedimiento de invocaciones que pueden ser locales o remotas.

A mediados de la década de 1980 comienzan a aparecer los sistemas distribuidos. Los sistemas distribuidos proporcionan grandes ventajas en cuanto a la escalabilidad y reducción de latencia que puede tener un aplicativo, ya que no es necesario que el software de la aplicación se ejecute en el computador local del usuario, pudiendo aumentar o disminuir los recursos del servidor o servidores donde se ejecuta la aplicación basándose en la demanda.

Con la aparición de los sistemas distribuidos aparece el concepto de invocación remota. El objetivo de este concepto es la invocación y ejecución de una función de manera remota y transparente para el usuario, con lo que éste no es consciente de dónde se está invocando dicha función. La aparición de este concepto ha llevado a la definición de varios estándares a lo largo del tiempo, siendo hoy en día los estándares SOAP y REST los más utilizados, sobre todo el segundo.

Uno de los primeros estándares que surge es ONC RPC (*Open Networking Computing Remote Procedure Call*), desarrollado originalmente por SUN, y estandarizado como RFC 1050 [25]. Se basa en un sistema de codificación o serialización de parámetros en un lenguaje muy próximo al lenguaje de programación, denominado XDR, y una especificación de la interfaz de los procedimientos utilizando el lenguaje IDL (*Interface Description Language*), la cual permite que el cliente que invoque ese procedimiento tenga la información necesaria para que la invocación sea correcta.

Otro estándar que surge a raíz del nacimiento de los RPC es Java RMI (*Remote Method Invocation*). Este estándar se basa en el concepto de objeto distribuido. El servidor almacena y gestiona los objetos de un aplicativo, mientras que el cliente, mediante un fragmento de código compilado local, denominado *stub*

crea, modifica e invoca esos objetos de manera remota. Este estándar, sin embargo, es propio de Java, con lo que no es interoperable con otros lenguajes de programación. Por otro lado, no requiere un lenguaje de definición de interfaz, ya que se utiliza la propia interfaz de Java, además de que la interfaz no necesita ser compilada.

Sin embargo, tanto el estándar RPC como Java RMI presentan ciertos problemas:

- Requieren la apertura de puertos en el servidor, con lo que no son *firewall friendly*.
- RPC estaba fundamentalmente diseñado para trabajar con el lenguaje de programación C, aunque su idea original era que fuese interoperable e independiente del lenguaje de programación.
- Java RMI tenía problemas de interoperabilidad, ya que estaba pensado para que se implementase todo en Java.
- Surge la necesidad de que la invocación remota se realice sobre otros servicios y protocolos ya en uso, como la Web y el protocolo HTTP.

Más tarde, con la aparición y popularización de la Web comenzaron a aparecer aplicaciones distribuidas que necesitaban comunicarse mediante el protocolo HTTP. Uno de los primeros protocolos de invocación remota vía web fue XML-RPC. Este mecanismo se basa en la transmisión de los datos necesarios para la invocación de funciones serializados en XML y enviados mediante HTTP. La idea era la de disponer de un estándar sencillo e interoperable que permitiese hacer uso de los protocolos de comunicación ya existentes. Sin embargo, no contó con el apoyo de las grandes empresas tecnológicas, con lo que quedó en desuso.

A raíz del surgimiento de XML-RPC, surgió el estándar SOAP, de manera que se tuviese un método de invocación remota para servicios web pesados. Este estándar, a diferencia de XML-RPC sí contó con el apoyo de grandes empresas, como IBM y Microsoft, con lo que se popularizó enormemente en la década de 1990. Los principios son similares a XML-RPC, ya que utiliza XML para la serialización de los datos y HTTP para el transporte, aunque podría utilizarse otro protocolo. Se trata de una especificación más extensa que XML.RPC, ya que proporciona más opciones y soluciones a todos los problemas posibles. Sin embargo, presenta ciertos inconvenientes, como son la complejidad que tiene su especificación, la gran cantidad de especificaciones que tiene (*WS-I, WS-Addressing, WS-Security* y *WS-Transaction*, entre otros), la verbosidad de los mensajes y la popularización de JavaScript, lo que conlleva que XML se quiera sustituir por JSON como método de serialización de datos.

Por último, surgen los servicios REST (*Representational State Transfer*). El estilo de arquitectura REST está exclusivamente pensado para diseñar servicios web, con el objetivo de aprovechar al máximo las características de HTTP, como son los códigos de error y éxito para la gestión de los flujos. Con la adopción de JSON como mecanismo de serialización se reduce la verbosidad del estándar, siendo muy legible y entendible por la gran mayoría de desarrolladores. Estos servicios REST trabajan con recursos, como puede ser un producto de una tienda online, un usuario dentro de un proveedor de identidades o una ciudad

dentro del conjunto de municipios de un país. Cada recurso lleva asociado un identificador único. Estos recursos se pueden crear, modificar o borrar mediante los verbos HTTP GET, POST, PUT, DELETE o PATCH.

Actualmente, el método de invocación remota más extendido entre diferentes servicios es mediante el uso de APIs REST por su facilidad de uso y entendimiento. Este incremento en el uso de este tipo de comunicación permite desarrollar aplicaciones más complejas, con código más reusable y disminuyendo el acoplamiento entre los diferentes módulos de su código, así como permitiendo una mayor capacidad de interacción con el usuario y personalización del contenido. Sin embargo, este incremento de funcionalidades también se ha traducido en un incremento de los riesgos.

La facilidad de uso que proporcionan las APIs REST a los desarrolladores también hace que sean un blanco muy interesante de cara a actores maliciosos. Es por ello por lo que la securización de estas APIs ha tomado especial relevancia. A ello también se une la aplicación del RGPD [1] en la Unión Europea, ya que impone requisitos especiales a las empresas para la protección de los datos de los usuarios con grandes multas en caso de no cumplirlos.

El problema de la securización de las APIs se viene resolviendo mediante el desarrollo de código seguro, la implementación de elementos de seguridad como *Firewalls*, *WAF* o *VPN*, así como otros elementos como API Gateways, además del uso de protocolos de acceso y autorización como pueden ser OAuth2.0 [2] u OpenID Connect [3].

Este trabajo se centra en el análisis, desde el punto de vista de la gestión de la ciberseguridad, de las amenazas y las soluciones que se presentan en cuanto a la securización de las APIs REST. Para ello, se hará un estudio e implementación de un elemento denominado API Gateway, el cual se encargará de aplicar los mecanismos necesarios para garantizar la protección de los datos expuestos por estas APIs.

1.2. Objetivos del Trabajo

Los objetivos de este trabajo se pueden dividir en dos categorías.

Objetivos de investigación:

- Investigación sobre los riesgos principales que presenta el uso de APIs REST
- Investigación del listado Top 10 de seguridad en APIs de OWASP [5]
- Investigación y análisis sobre estándares y protocolos de protección de APIs REST
- Investigación sobre diferentes alternativas de API Gateway para la securización de las APIs

Objetivos de implantación:

- Instalación de un API Gateway que permita securizar las APIs

- Configuración de protocolos y métodos de securización en el API Gateway
- Implementación de un elemento que permita hacer la emulación de un servicio que expone ciertas APIs
- Puesta en marcha del producto escogido e implementación de batería de pruebas para comprobar la correcta securización de la API.

1.3. Enfoque y método seguido

Las diferentes estrategias para la realización del trabajo son las siguientes:

- Estudio de la securización de APIs REST desde el punto de vista del programador. Esta estrategia se basa en analizar y desarrollar diferentes métodos que garanticen la seguridad de las API REST a nivel de código.
- Estudio de la seguridad de APIs REST desde el punto de vista de un atacante. Se trata de buscar las vulnerabilidades y hacer ciertos ataques que permitan ejecutar un *exploit* para obtener información confidencial, modificar esa información o denegar el acceso a dicha información.
- Estudio de la seguridad de APIs REST desde el punto de vista de la gestión de ciberseguridad. Este enfoque trata de analizar e implementar ciertos mecanismos existentes hoy en día para securizar las APIs REST, librando al programador de esta tarea.

El enfoque escogido para el desarrollo de este trabajo es el tercero, por los siguientes motivos:

- Mayor experiencia en despliegue de herramientas de gestión de acceso y autorización.
- Conocimiento previo de protocolos y estándares de protección de APIs REST
- Mayor adaptación con las políticas que comúnmente se implementan en el ámbito empresarial.

La metodología que se seguirá consiste en realizar un análisis previo de las amenazas actuales de las APIs REST publicado por OWASP, realizar una comparativa de diferentes herramientas de tipo API Gateway, elegir la herramienta adecuada en base a criterios como coste, facilidad de implantación y características que dispone, instalar y configurar dicha herramienta y realizar una batería de pruebas para demostrar la correcta securización de una API REST.

1.4. Estado del Arte

Actualmente, existen diversos métodos, estándares, herramientas y protocolos que son usados para la protección de las APIs.

Dentro de los estándares más ampliamente utilizados se encuentran OAuth2.0 y OpenID Connect. Por una parte, el estándar OAuth2.0 proporciona un *framework* que "*permite a una aplicación de terceros obtener un acceso limitado a un servicio HTTP, ya sea en nombre del propietario de un recurso, orquestando una*

interacción de aprobación entre el propietario del recurso y el servicio HTTP, o permitiendo que la aplicación de terceros obtenga acceso en su propio nombre" [2], mientras que el estándar OpenID Connect "*permite a los clientes verificar la identidad del usuario final basándose en la autenticación realizada por un servidor de autorización, así como obtener información de perfil básica sobre el usuario final de forma interoperable y similar a REST*" [3].

Estos dos estándares son interoperables entre sí, ya que el estándar OpenID Connect está desarrollado basándose en el estándar de OAuth2.0.

Ilustración 1. Logo representativo del estándar OAuth

Ilustración 2. Logo representativo del estándar OpenID Connect

Por otra parte, existen diversos mecanismos para garantizar que la información, tanto de la autenticación como de la autorización, viaja cifrada y firmada entre el servidor de autorización y el cliente que desea consumir una API. Hoy en día, los estándares más utilizados para conseguir este propósito son:

- **JWT (JSON Web Token)** [5] para el intercambio de la información
- **JWE (JSON Web Encryption)** [6] para el cifrado de la información
- **JWS (JSON Web Signature)** [7] para la firma de la información
- **JWK (JSON Web Key)** [8] para la gestión de las claves de cifrado y firma
- **JWA (JSON Web Algorithms)** [9] para la gestión de los algoritmos de cifrado y firma

Ilustración 3. Componentes del conjunto JOSE (JSON Object Signing and Encryption)

En relación con el uso de herramientas de gestión de API y, de acuerdo con el cuadrante mágico de Gartner [10], se pueden destacar las siguientes herramientas:

- **Kong Gateway** [11]: herramienta *Open Source* que permite hacer una gestión integral de las APIs REST, con respuestas por debajo de los milisegundos, escalable y que permite desde la creación de reglas de acceso y reescritura hasta la definición de políticas de seguridad, todo ello basándose en módulos de múltiples fabricantes para añadir extensiones.

- **Tyk Gateway**[12]**:** herramienta disponible en formato *Open Source*, desarrollada para ser una herramienta nativa de Cloud, altamente escalable que permite gestionar el flujo completo de una API, desde la creación, la definición de políticas de acceso y securización y la publicación de esta API.

- **WSO2 API Manager** [13]: herramienta *Open Source* utilizada por un gran número de entidades y compañías para la gestión de las API. Permite definir políticas de seguridad y acceso a las API mediante archivos de configuración, cuenta con módulos de integración con herramientas de terceros para la gestión de los accesos y permite la importación de API en formatos estándar como CSV.

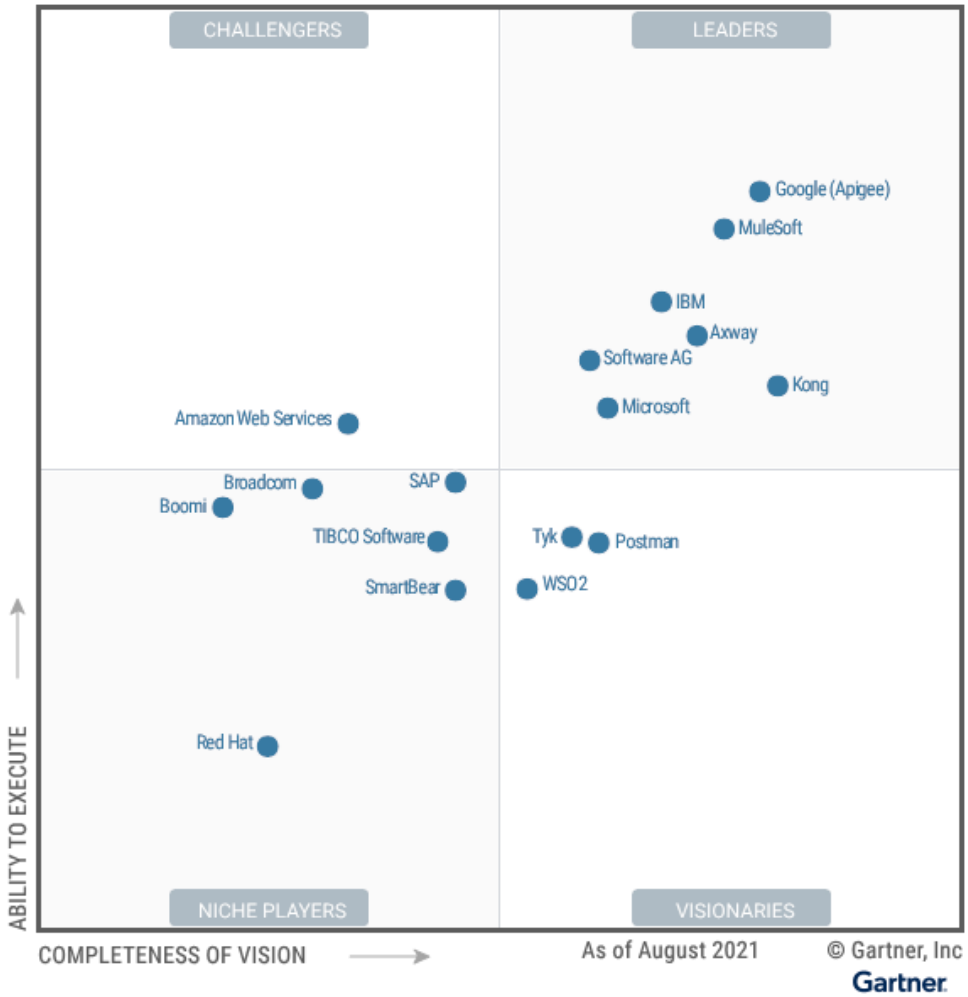


Ilustración 4. Cuadrante mágico de Gartner de 2021 sobre herramientas de API management

Merece la pena mencionar que se han obviado herramientas de código cerrado y de pago por no ser apropiadas para el desarrollo de este trabajo, como pueden ser Mulesoft, Google Apigee o IBM API Connect.

Ilustración 5. Logo representativo de la solución Kong

Ilustración 6. Logo representativo de la solución Tyk

Ilustración 7. Logo representativo de la solución WSO2

1.5. Planificación del Trabajo

La planificación de las tareas que se llevarán a cabo durante la ejecución de este trabajo se puede ver en las siguientes ilustraciones.

Nombre de tarea	Duración	Comienzo	Fin
Plan de trabajo	**9 días**	**mié 16/02/22**	**mar 01/03/22**
Contexto y justificación	1 día	mié 16/02/22	mié 16/02/22
Objetivos	1 día	jue 17/02/22	jue 17/02/22
Metodologías	1 día	vie 18/02/22	vie 18/02/22
Estado del arte	2 días	lun 21/02/22	mar 22/02/22
Definición de tareas	1 día	mié 23/02/22	mié 23/02/22
Análisis de riesgos	1 día	jue 24/02/22	jue 24/02/22
Planificación	1 día	vie 25/02/22	vie 25/02/22
Presupuesto	1 día	lun 28/02/22	lun 28/02/22
Entrega PEC1	0 días	mar 01/03/22	mar 01/03/22
Análisis	**20 días**	**mar 01/03/22**	**mar 29/03/22**
Análisis de amenazas	3 días	mar 01/03/22	jue 03/03/22
OWASP Top 10 API Security	3 días	vie 04/03/22	mar 08/03/22
Vectores de ataque	3 días	mié 09/03/22	vie 11/03/22
Descripción de métodos de protección	3 días	lun 14/03/22	mié 16/03/22
Análisis de soluciones	6 días	jue 17/03/22	jue 24/03/22
Redacción de PEC2	20 días	mar 01/03/22	lun 28/03/22
Entrega PEC2	0 días	mar 29/03/22	mar 29/03/22
Implementación	**20 días**	**mar 29/03/22**	**lun 25/04/22**
Configuración de máquina virtual	2 días	mar 29/03/22	mié 30/03/22
Instalación de API Gateway	2 días	jue 31/03/22	vie 01/04/22
Instalación de Mock API	2 días	lun 04/04/22	mar 05/04/22
Configuración de API Gateway	4 días	mié 06/04/22	lun 11/04/22
Ejecución de casos de prueba	6 días	mié 06/04/22	mié 13/04/22
Redacción de PEC3	18 días	jue 31/03/22	lun 25/04/22
Entrega PEC3	0 días	lun 25/04/22	lun 25/04/22
Memoria final	**25 días**	**mar 26/04/22**	**mar 31/05/22**
Conclusiones	2 días	mar 26/04/22	mié 27/04/22
Extensiones	3 días	jue 28/04/22	lun 02/05/22
Redacción de memoria final	18 días	mar 03/05/22	jue 26/05/22
Preparación de entregables	2 días	vie 27/05/22	lun 30/05/22
Memoria final	0 días	mar 31/05/22	mar 31/05/22
Video presentación	**5 días?**	**mar 31/05/22**	**lun 06/06/22**
Preparación de video	4 días	mar 31/05/22	vie 03/06/22
Entrega de vídeo	1 día?	lun 06/06/22	lun 06/06/22

Ilustración 8. Listado de tareas

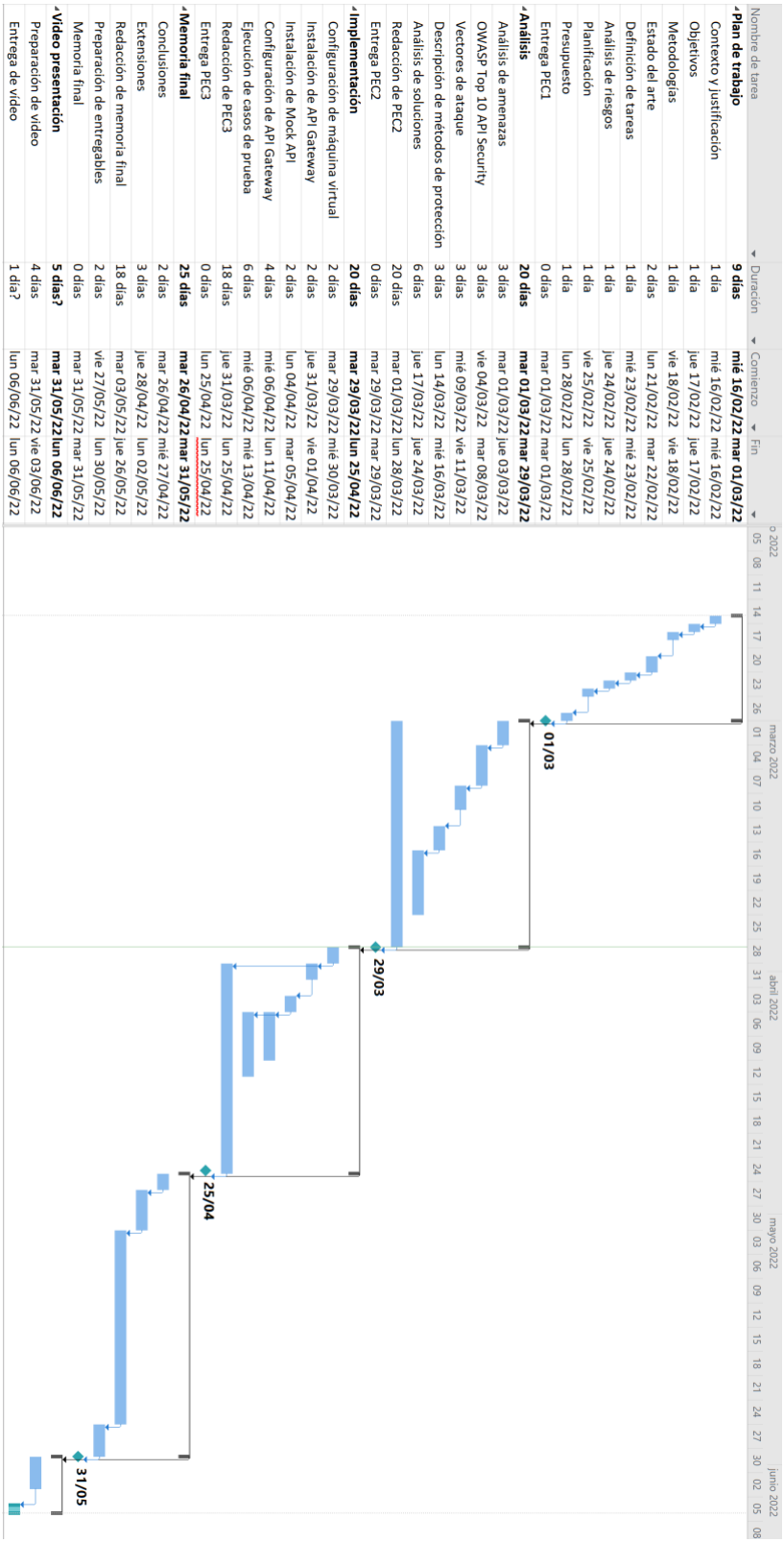


Ilustración 9. Listado de tareas con diagrama de Gantt

1.6. Recursos necesarios y presupuesto del proyecto

A continuación, se procede a enumerar los recursos necesarios para la ejecución del proyecto y el presupuesto estimado.

Recurso	Uso	Coste unitario	Coste total
Personal desarrollador	450 h	30 €/h	13.500 €
Ordenador	4 meses	29 €/mes	116 €
Conexión a Internet	4 meses	25 €/mes	100 €
Electricidad	4 meses	12 €/mes	48 €
		TOTAL	13.764€

Tabla 1. Recursos necesarios para la ejecución del proyecto

1.7. Análisis de riesgos

Preliminarmente se han identificado los riesgos que se enumeran a continuación. Se ha elaborado una valoración de la probabilidad y el impacto, en una escala de 1 a 5, de cada riesgo, así como ciertas acciones de mitigación.

R1 – Sobredimensión del alcance

Descripción	La definición del proyecto es demasiado ambiciosa como para terminarlo en tiempo y forma.
Probabilidad	2
Impacto	5
Mitigación	Simplificar las tareas y no dedicar más tiempo del planificado a cada tarea. Eliminar las partes superfluas o que no aporten información.

R2 – Implementación de plataforma escogida

Descripción	La plataforma escogida no se puede desplegar o implementar en el hardware disponible
Probabilidad	1
Impacto	4
Mitigación	Se tratará de implementar o desplegar la plataforma en un hardware o máquina virtual compatible no planeado para el desarrollo del proyecto.

R3 – Pérdida de información

Descripción	Se produce un borrado o pérdida de información accidental que conlleva la destrucción de la memoria y las implementaciones de la plataforma
Probabilidad	1
Impacto	5
Mitigación	Se realizarán copias de seguridad periódicas almacenadas en un dispositivo externo.

R4 – Desviación en la planificación

Descripción	Se produce una desviación o mala planificación de las tareas a desarrollar por no dimensionarlas de manera correcta.
Probabilidad	2
Impacto	3

Mitigación	Se mantendrá un seguimiento constante de la planificación y se corregirá en cuanto se detecte un posible desvío.

Tabla 2. Riesgos identificados

2. Análisis de amenazas

En este apartado se procede a analizar las amenazas que se presentan en implementaciones de APIs REST desde un punto de vista clásico, es decir, amenazas a la tríada CIA (*Confidentiality, Integrity* y *Availability*) así como desde el punto de vista del Top 10 de amenazas de seguridad en APIs REST publicado por OWASP [4].

2.1. Amenazas a la confidencialidad, disponibilidad e integridad.

La **confidencialidad** se define como la propiedad de un dato o recurso de mantenerse privado o secreto. Dicho de otro modo, la confidencialidad debe garantizar que sólo aquellos actores que están autorizados para acceder a un recurso pueden efectivamente hacerlo. A modo de ejemplo, el dueño de una cuenta bancaria y todos aquellos miembros a los que dicho dueño haya dado acceso a esa cuenta, podrán ver los movimientos y extractos que se han realizado.

La **disponibilidad** de un sistema, red, aplicación o información es la propiedad por la cual estos sistemas se mantienen a disposición de los agentes autorizados para su uso. Esto quiere decir que todos estos sistemas o recursos deben estar ejecutándose y que los actores autorizados pueden acceder a ellos en el momento en que lo requieran.

Por último, la **integridad** se refiere a la propiedad de un recurso de tener información correcta, autentica, confiable y fidedigna. Dicho de otro modo, la integridad asegura que la información almacenada o trasmitida en un sistema no ha sido manipulada por nadie que no esté autorizado a ello.

Estas propiedades se pueden ver amenazadas en una implementación de un sistema que exponga APIs REST. En el caso de la confidencialidad, si no se tienen los mecanismos de autenticación y autorización necesarios, cualquiera podría hacer una petición a una API y obtener información que podría perjudicar a una empresa o a un individuo.

Supongamos un sistema bancario compuesto por múltiples microservicios que exponen ciertas APIs para poder comunicarse entre ellos. Dentro de este sistema lo más seguro es que hubiese un microservicio que, a través de una API, permite consultar los últimos movimientos de una cuenta bancaria. Como es lógico, esta API debe contar con todas las medidas de protección necesarias para que sólo aquellos que están autorizados a ver los movimientos pueden verlos. De lo contrario, se estaría violando el principio de confidencialidad.

Siguiendo con el ejemplo del sistema bancario, si no se disponen de medios de protección que garanticen la disponibilidad de los datos, un atacante podría generar millones o miles de millones de peticiones contra el sistema bancario con el objetivo de saturarlo y dejarlo inutilizable. Esto generaría que parte de los usuarios o todos ellos no pudiesen utilizar sus cuentas bancarias, haciendo que,

entre otras cosas, no se pudiesen pagar recibos, cobrar nóminas o transferir dinero a familiares.

Por último, es sencillo llegar a la conclusión que, sin las medidas adecuadas, un actor malicioso podría modificar los datos del saldo de su cuenta bancaria para reflejar una cantidad de dinero mayor que la que realmente tiene, o para disminuir la cantidad de dinero que un individuo o compañía podría tener en su cuenta bancaria, lo que sería una violación de la integridad de los datos que el banco almacena.

2.2. Análisis de API Security Top 10 de OWASP

OWASP (*Open Web Application Security Project*) [14] es una fundación sin ánimo de lucro que trabaja para mejorar la seguridad del software. Se trata de una de las fundaciones de referencia en cuanto a seguridad en tecnologías y software se refiere. Proporciona herramientas, recursos, guías y formaciones, entre otras cosas, para que la comunidad tecnológica implemente medidas de seguridad en sus sistemas que aborden los problemas actuales relacionados con la ciberseguridad.

Una de las guías de referencia de OWASP es el API Security Top 10, donde recopila las 10 amenazas y riesgos de seguridad más comunes que se presentan en implementaciones de APIs REST. La última versión data de 2019, y se puede encontrar en [4].

Este ranking es el siguiente:

1. **API1: Broken Object Level Authorization (Autorización a nivel de objeto rota)**

   Se trata de la amenaza que más comúnmente se ha explotado, generando un gran impacto en los sistemas atacados. La autorización a nivel de objeto es un mecanismo de control de acceso que normalmente está implementado a nivel de código, y su función es validar que un usuario sólo puede acceder a objetos para los que debería tener acceso.
   Un acceso no autorizado podría resultar en una filtración de datos a terceros no autorizados, pérdida de datos o manipulación de datos. Por otra parte, un acceso no autorizado a los objetos manejados por una API puede llevar a un control total de la cuenta dueña de esos datos. [15]

2. **API2: Broken User Authentication (Autenticación del usuario rota)**

   La autenticación en las APIs es un mecanismo normalmente complejo y confuso para los desarrolladores, lo que puede llevar a ideas equivocadas o implementaciones erróneas. Además, los mecanismos de autenticación son un objetivo fácil y atractivo para los atacantes, ya que normalmente está expuesto a todo el mundo dentro de una red. Esto hace que el componente de autenticación sea potencialmente vulnerable a muchos *exploits*.

Un atacante puede obtener el control sobre otras cuentas de usuario de un sistema, leer sus datos personales o realizar acciones sensibles en su nombre. [16]

3. **API3: Excessive Data Exposure (Exposición excesiva de datos)**

   La exposición o envío excesivo de datos por parte de las APIs puede llevar a compromisos en la confidencialidad de los datos, ya que con un simple escaneo o *sniffing* del tráfico en la repuesta de una API se puede lograr obtener datos sensibles que comprometan al sistema o los usuarios finales. [17]

4. **API4: Lack of Resource & Rate Limiting (Falta de recursos y limitación de peticiones)**

   Se trata de una amenaza muy sencilla de explotar, ya que no requiere autenticación ni conocimientos técnicos muy complejos. Esta amenaza puede llevar a situaciones de denegación de servicio (DoS), haciendo que la API no sea capaz de responder o que quede en un estado de indisponibilidad. [18]

5. **API5: Broken Function Level Authorization (Autorización a nivel de función rota)**

   Esta amenaza es similar a la descrita en el punto 1 (*Broken Object Level Authorization*), con la diferencia de que, en este caso, la amenaza está dirigida a las propias funciones implementadas y no a los objetos. Por lo general, las comprobaciones de autorización de una función o recurso se gestionan, normalmente, a través de configuraciones y, a veces, a nivel de código. Estas tareas pueden llegar a ser confusas, ya que muchas aplicaciones modernas pueden contener muchos tipos de roles o grupos y jerarquías de usuarios complejas.
   Esta amenaza puede llevar a un atacante no autorizado a tener acceso a cierta funcionalidad restringida. Por lo general, las funciones administrativas suelen ser el principal objetivo. [19]

6. **API6: Mass Assignment (Asignaciones masivas)**

   Los *frameworks* de desarrollo de software modernos animan a los desarrolladores a usar funciones que automáticamente vinculan los datos que recibe una función de parte del cliente en variables internas u objetos. Un atacante puede aprovechar esta metodología para actualizar o sobrescribir propiedades de objetos sensibles que el desarrollador nunca tuvo intención de exponer públicamente.
   Por lo general, la explotación de esta amenaza requiere de un entendimiento de la lógica de negocio, las relaciones de los objetos y la estructura de la API. Entre los posibles impactos de la explotación de esta amenaza se encuentran el escalado de privilegios, la manipulación de datos y la evasión de mecanismos de seguridad adicional que pueda implementar la red o sistemas. [20]

7. **API7: Security Misconfiguration (Configuraciones erróneas de seguridad)**

   Los errores en las configuraciones de seguridad o la falta de estas pueden ocurrir en cualquier capa dentro de una API, desde la capa de red hasta la capa de aplicación. Un atacante habitualmente intenta buscar código o aplicativos no actualizados o con brechas de seguridad conocidas no parcheadas, *endpoints* comunes y públicamente expuestos o ficheros y directorios no protegidos, de manera que pueda obtener un acceso no autorizado al sistema o conocimiento acerca de los sistemas que se encuentran por debajo de la API.
   Estos errores o falta de configuración de las opciones de seguridad de los sistemas o *frameworks* puede llevar a exponer datos sensibles de los usuarios o de los sistemas, pudiendo permitir a un atacante obtener un control completo sobre el servidor. [21]

8. **API8: Injection (Inyección)**

   Los defectos en los sistemas de almacenamiento de datos que dan lugar a ataques de inyección son muy comunes, encontrándose en sistemas de bases de datos SQL, NoSQL, sistemas LDAP, comandos de sistema operativo y parseadores XML, entre otros.
   Un ataque de inyección puede ser crítico para la empresa o institución en el que se produce, ya que puede llevar a filtración de datos sensibles, control total del sistema, pérdida de datos o incluso a una denegación de servicio. [22]

9. **API9: Improper Assets Management (Gestión de activos inapropiada)**

   La falta de documentación de los sistemas o las APIs desplegadas en un entorno, o el hecho de que esta sea imprecisa, puede llevar a que un sistema pueda ser comprometido sin mayor esfuerzo. Una documentación desactualizada hace que sea más difícil encontrar y/o parchear vulnerabilidades conocidas, tanto en servidores como en otro tipo de sistemas.
   Es muy común encontrarse hosts que publican APIs innecesariamente expuestos, lo que conlleva un riesgo para la seguridad corporativa. Además, si un atacante se encuentra con versiones de APIs antiguas y sin parchear, fácilmente podría comprometer el sistema sin tener que luchar contra mecanismos de seguridad avanzados. [23]

10. **API10: Insufficient Logging & Monitoring (Monitorización y obtención de logs insuficientes)**

    Una correcta monitorización y registro de logs de las acciones, accesos y, en general, toda la actividad que ocurre en una API, tanto a nivel de red como a nivel de aplicación, es fundamental para tener visibilidad de posibles ataques que se puedan estar llevando a cabo o para analizar

ataques que hayan ocurrido en el pasado, de manera que se puedan aplicar medidas correctivas para que no vuelvan a ocurrir.
En el supuesto de que una API no se monitorice o no esté registrando ningún log un atacante podría realizar acciones maliciosas pasando completamente desapercibido. [24]

Analizando las amenazas dispuestas en el ranking anterior, se procede a elaborar una relación entre cada amenaza y las propiedades básicas dentro de la ciberseguridad (tríada CIA) que se ven directamente comprometidas.

Amenaza	**Confidencialidad**	**Disponibilidad**	**Integridad**
API1: Broken Object Level Authorization	**X**		**X**
API2: Broken User Authentication	**X**		**X**
API3: Excessive Data Exposure	**X**		
API4: Lack of Resource & Rate Limiting		**X**	
API5: Broken Function Level Authorization	**X**		**X**
API6: Mass Assignment		**X**	**X**
API7: Security Misconfiguration	**X**	**X**	**X**
API8: Injection		**X**	**X**
API9: Improper Assets Management	**X**	**X**	**X**
API10: Insufficient Logging & Monitoring	**X**	**X**	**X**

Tabla 3. Relación entre amenazas a las API y propiedades de la seguridad afectadas

3. Vectores de ataque

Como se ha visto en el capítulo anterior, las amenazas que presentan hoy en día las APIs REST son múltiples y muy variadas, lo que da lugar a que los vectores de ataque también lo sean.

Se puede definir un vector de ataque como un camino o medio por el que un atacante puede acceder a un sistema con la intención de enviar y desplegar una carga maliciosa o *payload* que le permita tener acceso, control o inhabilitar dicho sistema. Los vectores de ataque permiten a los actores malignos explotar las vulnerabilidades del sistema.

Los vectores de ataque más comunes son los siguientes:

- **Manipulación del ID del objeto enviado en una petición:** consiste en modificar el identificador único que representa a un objeto dentro del servicio REST, ya sea de manera manual o automatizada, con el objetivo de evadir los controles de autorización propuestos por el desarrollador de la API. Esto puede llevar a problemas de confidencialidad y de integridad de los datos almacenados.
  A modo de ejemplo, si se expone la API */users/{id}*, donde *{id}* corresponde con el identificador único de cada usuario, y en la cual se puede obtener y modificar los datos de usuarios, se podría automatizar mediante un script la obtención de todos los usuarios presentes en el sistema, simplemente enviando un identificador diferente en cada petición.

- **Ataques de fuerza bruta:** se trata de la utilización de diccionarios de usuarios y contraseñas para la autenticación contra una API utilizando herramientas automáticas. En caso de que el ataque sea satisfactorio, tanto la confidencialidad como la integridad se pueden ver afectadas e, incluso, la disponibilidad, ya que un número elevado de peticiones malignas puede hacer que el sistema se quede sin recursos para procesar las peticiones legítimas.

- **Ataques Man In The Middle (MITM):** este tipo de ataques consisten en interceptar las comunicaciones entre un cliente y un servidor que expone una API, de manera que se tenga acceso a todas las peticiones y comunicaciones que se llevan a cabo entre estos dos actores. Un ataque de estas características, en caso de que las comunicaciones entre cliente y servidor no estén cifradas, puede resultar en robo de credenciales, lectura y filtración de datos sensibles, entre otros. Esto supone un claro problema de confidencialidad de los datos.

- **Ataques de denegación de servicio (DoS) o denegación de servicio distribuido (DDoS):** con el envío masivo de peticiones a *endpoints* de una API públicamente expuestos, o con el envío de *payloads* extremadamente grandes, se puede llegar a un escenario de denegación de servicio (si se realiza desde un solo cliente) o denegación de servicio

distribuido (si se realiza desde una *botnet*). Esto puede dar lugar a que el sistema no sea capaz de responder a peticiones que le lleguen al *endpoint* atacado o, incluso, a cualquier *endpoint*, dando lugar a un fallo en la disponibilidad de la API.

- **Modificación de método HTTP enviado:** si no se implementan los controles adecuados, un atacante podría modificar un verbo GET enviado en una llamada a una API por un PUT o DELETE y, sin tener autorización, modificar o eliminar el recurso o dato, dando lugar a fallos en la integridad de los datos.

- **Ataques de inyección:** por lo general, las APIs REST se encuentran una capa por encima de la capa de datos, lo que quiere decir que se apoyan en sistemas de almacenamiento de datos para leer o modificar recursos. Estos sistemas pueden ser bases de datos SQL, NoSQL, sistemas LDAP o el propio sistema operativo del servidor. Un atacante, a través un escáner de vulnerabilidades o analizando el comportamiento o respuestas de las APIs podría llegar a conocer qué sistema de almacenamiento de datos se encuentra por debajo y, por consiguiente, intentar atacarlo mediante ataques de inyección.
  Los ataques de inyección en una API consisten en enviar fragmentos de código ejecutable (consultas SQL/LDAP o comandos de sistema operativo) mediante los parámetros de la URL llamada o en el cuerpo del mensaje, de manera que el sistema lo interprete como un comando a ejecutar. Esto puede resultar en un control completo del sistema por parte del atacante, en filtración de datos sensibles o en la inhabilitación del consumo de la API.

- **Abuso de datos enviados:** en peticiones de tipo POST o PUT en las que se envían datos en el cuerpo del mensaje HTTP, un atacante podría incluir valores de atributos que el desarrollador no pretende que se modifiquen y que, sin la protección adecuada, podrían llegar a modificarlos. Esto afecta directamente a la integridad de los datos que expone la API REST.

- **Robo o exposición de credenciales de acceso:** en multitud de ocasiones las contraseñas de administración de servicios o de APIs se almacenan en texto claro en archivos de texto, de configuración o en repositorios compartidos por el equipo de mantenimiento de estos servicios. Se podría producir un ataque de ingeniería social que permita acceder a estas claves si no se cuenta con la formación del personal adecuada, lo que conllevaría un control total de la información y sistemas que soportan una API REST.

4. Métodos de protección

Los métodos de protección que existen hoy en día se basan en la aplicación de buenas prácticas, utilización de protocolos estandarizados, gestión documental, desarrollo de código seguro e implementación de elementos de seguridad a nivel de red y aplicación, entre otros.
Como se verá más adelante, muchos de los métodos de protección de APIs disponibles se pueden implementar en un elemento común denominado API Gateway, el cual, por lo general, actúa a modo de proxy inverso entre el cliente y el *backend* que ejecuta la API.

4.1. Mecanismos de cifrado de comunicaciones

Con el objetivo de evitar ataques de tipo *Man in the Middle*, en los cuales un actor malicioso se posiciona entre el cliente y el servidor que expone las APIs, de manera que pueda interceptar y leer las comunicaciones, siempre se debe habilitar el protocolo navegación cifrada HTTPS o, en su defecto, el protocolo TLS. Estos protocolos garantizan que las comunicaciones entre un cliente y un servidor viajan cifradas de extremo a extremo, con lo cual un atacante que intercepte la comunicación nunca podrá ser capaz de leer su contenido.

4.2. Mecanismos de autenticación y autorización

La resolución al problema de autenticación en los servicios que exponen APIs REST ha llevado a la definición de múltiples protocolos, estándares y buenas prácticas, de los cuales prevalecen los siguientes hoy en día.

4.2.1. Autenticación básica de HTTP

Debido a que el estándar REST está pensado para exprimir al máximo las características del protocolo HTTP, la autenticación básica HTTP [26] ha sido uno de los primeros métodos de autenticación implementados para proteger el acceso a las APIs REST. Este método se basa en el envío de una cabecera especial en cada petición HTTP que se realiza contra la API. Esta cabecera se llama *Authorization*, y el valor que contiene está formado por la palabra *Basic* seguido del resultado de aplicar la función base64 a la concatenación del conjunto *usuario:password* con el que se desea autenticar.

Como ejemplo, suponiendo que se dispone de una API cuyas credenciales de acceso están formadas por un nombre de usuario *eduardo* y una contraseña *tfmuoc22*, la cabecera que se envía es la siguiente:

Authorization: Basic ZWR1YXJkbzp0Zm11b2MyMg==

La cadena *ZWR1YXJkbzp0Zm11b2MyMg==* es el resultado de aplicar la función base64 a la cadena *eduardo:tfmuoc22.*

Sin embargo, este método presenta una gran debilidad, y es que la función base64 es reversible, con lo que si un atacante consigue interceptar alguna

petición enviada a la API REST podría obtener de manera trivial las credenciales que se usan.

4.2.2. Autenticación HTTP Digest

Este método de autenticación surgió para paliar los problemas de seguridad surgidos debido a la reversibilidad de la función base64 utilizada en la autenticación básica de HTTP. Se basa en la utilización de la función hash MD5 para el envío de las credenciales de la cuenta que se quiere autenticar. El servidor puede estar configurado para exigir ciertas directivas para la aplicación del hash MD5, así como para exigir una calidad de protección o *quality of protection (qop) [26]*.

De forma básica, el cliente forma la credencial *response* de la siguiente manera:

HA1 = MD5(username:realm:password)
HA2 = MD5(metodo:digestURI)
response = MD5(HA1:nonce:HA2)

Hay que tener en cuenta que el *nonce* es un valor aleatorio generado por el servidor que envía en la respuesta de la petición de acceso a la API, cuando la cuenta no está autenticada. Suponiendo que se desea acceder a una API cuya URI relativa es */contracts/all*, utilizando el método GET, con una cuenta cuyo nombre de usuario es *eduardo*, su contraseña es *tfmuoc22* y se encuentra en el dominio o realm *alumnos@uoc.edu*, y sabiendo que el servidor ha enviado al cliente un *nonce* con valor *12345*, el resultado se formaría de la siguiente manera:

HA1 = MD5(eduardo:alumnos@uoc.edu:tfmuoc22)
HA2 = MD5(GET:/contracys/all)
response = MD5(HA1:12345:HA2)

Con lo cual, el valor de la cabecera *Authorization* que se enviaría sería el siguiente:

Authorization: Digest username="eduardo",
realm="alumnos@uoc.edu",
nonce="12345",
uri="/contracys/all",
response="3c82c02f8100810b167150792f2e5d23"

Como se puede ver, en ningún momento el cliente envía en texto plano o con una función reversible la contraseña de la cuenta. Para validar las credenciales, el servidor simplemente tiene que computar el valor del *response* en su lado y compararlo con el que ha recibido del cliente.

A pesar de que, en su día, este método se consideraba seguro, con el aumento exponencial de la potencia de cómputo de los ordenadores, se llegó a un punto en el que la función hash MD5 ya no se consideraba segura, ya que está

demostrado que se puede revertir en cuestión de segundos, obteniendo la cadena original que da lugar al hash.

4.2.3. Autenticación mediante API Key

Este método consiste en la creación y expedición de claves alfanuméricas para que los desarrolladores puedan hacer uso de esta credencial para autenticarse contra la API. El mecanismo de autenticación es muy similar al mecanismo de autenticación básica de HTTP. La diferencia está en que, en este caso, no se envía ninguna contraseña en las cabeceras HTTP, sino que se envía directamente esta API Key.
Los métodos de envío de la API Key difieren de un servicio a otro, ya que no está estandarizado. Pueden enviarse en una cabecera HTTP personalizada, en la cabecera *Authorization*, como parámetro de la URL o en el cuerpo del mensaje. La ventaja que tiene sobre el uso de la autenticación básica HTTP es la posibilidad de no revelar las credenciales de ninguna cuenta y la gestión de esta API Key por parte del proveedor del servicio o administrador, permitiendo revocar cualquier clave que se detecte que haya podido ser robada.

Sin embargo, presenta los mismos problemas que la autenticación básica HTTP, ya que un atacante que intercepte una petición que contenga una API Key podría hacer uso de ella de manera no autorizada.

4.2.4. Autenticación mediante protocolo OAuth2.0

El protocolo OAuth2.0 [2] para aportar una solución que permitiese a un cliente o una aplicación tercera acceder a los datos o recursos de un individuo en nombre de dicho individuo, con el consentimiento previo de este, sin que esta aplicación tercera conociese ni manejase las credenciales de dicho individuo. A pesar de ser un protocolo diseñado para la autorización y no para la autenticación, en escenarios en los que una aplicación *backend* desea consumir una API se suele utilizar como protocolo de autenticación y autorización.

En este protocolo existen cuatro roles fundamentales:

- **Dueño del recurso o *resource owner*:** se trata del usuario o cliente que tiene ciertos recursos o datos en un servidor externo a los cuales una aplicación tercera quiere acceder.

- **Servidor de recursos o *resource server***: se trata del servidor o API que contiene o expone los recursos o datos pertenecientes al *resource owner*.

- **Cliente:** se trata de una aplicación que desea acceder a los recursos de un usuario en su nombre y con su consentimiento. El cliente normalmente dispone de unas credenciales compuestas por un *client_id* y un *client_secret* para autenticarse contra el servidor de autorización, aunque estas credenciales pueden variar.

- **Servidor de autorización o *authorization server***: se trata de un servido que actúa como proveedor OAuth2.0, el cual se encarga de autenticar al

usuario y, en base a los permisos que el cliente solicite, expedir una credencial o *token*, lo cual permite al cliente consumir un recurso o API en nombre del usuario gracias a dicho *token*.

Existen varios flujos o *grant types* definidos en el protocolo para solventar diferentes casuísticas. Sin embargo, en la actualidad solamente dos de ellos están recomendados, ya que el resto presentan ciertos problemas a nivel de seguridad que no cumplen las buenas prácticas y que pueden llevar a un ataque. Estos dos flujos son los siguientes:

- ***Authorization code*:** este flujo se ejecuta en los casos en los que el dueño del recurso es un usuario que interactúa con la aplicación o cliente mediante una interfaz gráfica, generalmente a través del navegador web. El flujo que se sigue es el siguiente:

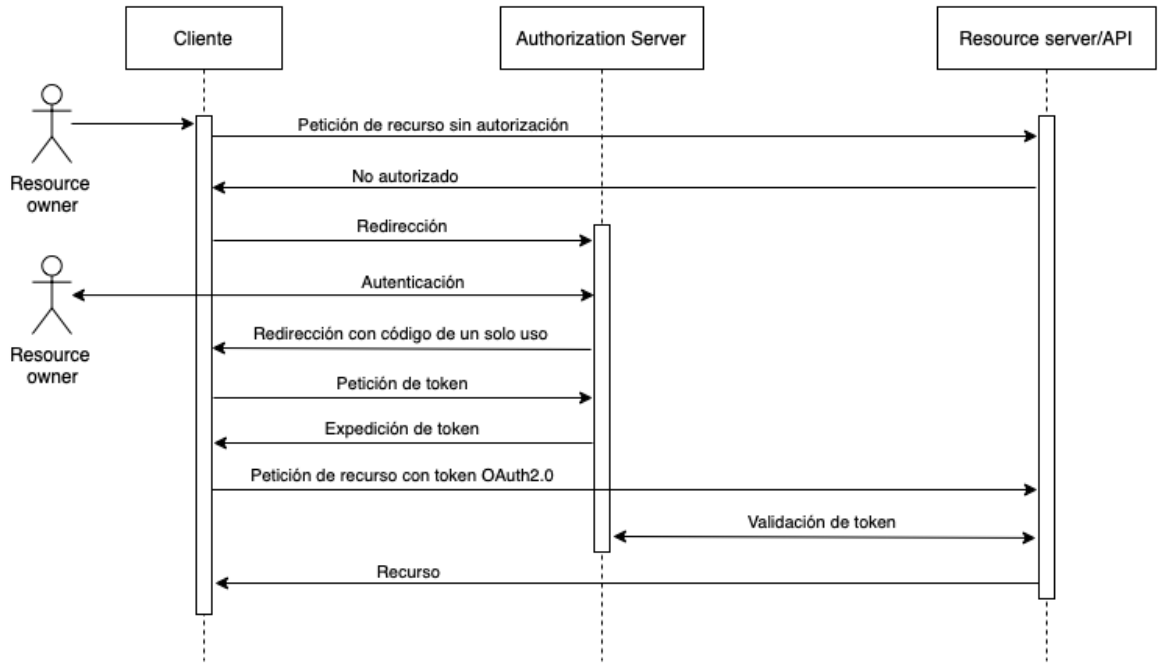


Ilustración 10. Flujo OAuth2.0 Authorization Code

Como se observa en la ilustración anterior, el cliente en ningún momento gestiona o ve las credenciales del usuario, ya que estas son manejadas exclusivamente por el servidor de autorización.

- ***Client credentials*:** este flujo se ejecuta en los casos en los que el cliente y el dueño del recurso son el mismo sistema. Generalmente este flujo se usa en arquitecturas de microservicios en los que múltiples servicios de *backend* consumen una o varias APIs de otros servicios de *backend*. En este caso, el flujo es ligeramente más simple que el anterior.

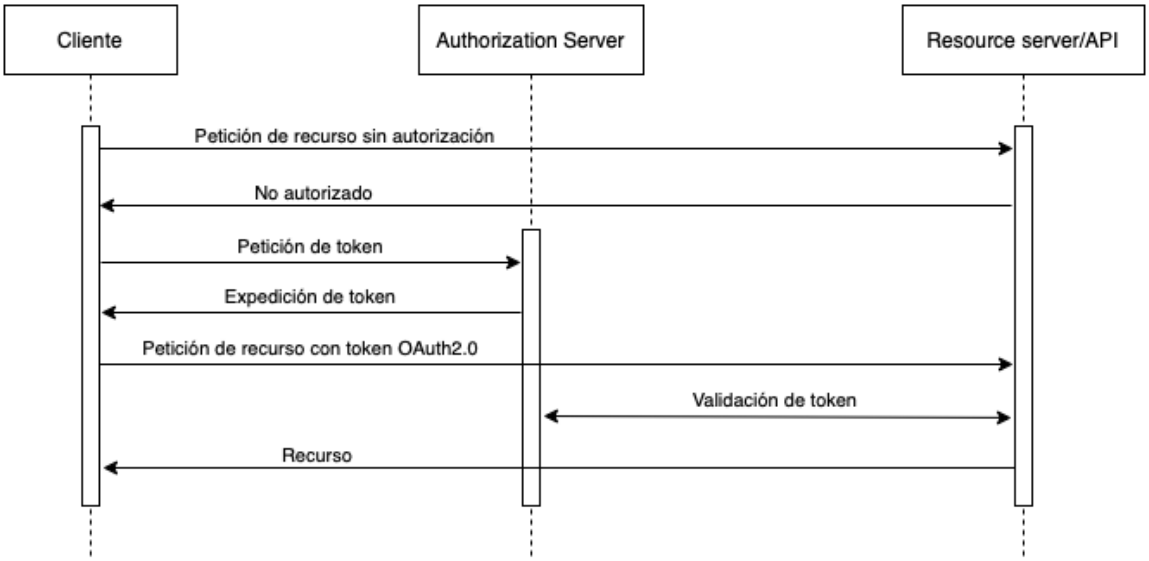


Ilustración 11. Flujo OAuth2.0 Client Credentials

En este caso, como se ve, no existe un dueño del recurso explícito, sino que es el propio cliente. La autenticación se realiza en el paso de petición de *token* enviando las credenciales que tenga asignadas el cliente en el servidor de autorización.

El protocolo OAuth2.0 define un concepto denominado *scope* que actúa a modo de permiso o privilegio. A modo de ejemplo, suponiendo que el servidor de recursos permite tanto la lectura como la escritura de nuevos datos, podría existir el *scope* llamado *read* y el *scope* llamado *write*. En el flujo *authorization code*, en el momento en el que el cliente redirige al usuario hacia el servidor de autorización, también le envía a este servidor qué *scopes* o privilegios está solicitando. Una vez que el usuario se autentica, el servidor de autorización solicita el consentimiento al usuario de la cesión de estos permisos al cliente.

Independientemente de los flujos que se sigan a la hora de utilizar el protocolo OAuth2.0, el *token* que se obtiene siempre se transmite de la misma forma. Para ello se hace uso de la cabecera HTTP *Authorization*, cuyo valor consiste en la palabra *Bearer* seguido del *token* expedido por el servidor de autorización. Un ejemplo de este envío de token podría ser el siguiente:

Authorization: Bearer 2737cc91-3c2b-426d-a32e-5e45fb6b8c89

4.3. Mecanismos de prevención de ataques de inyección

Existen ciertos mecanismos y buenas prácticas que permiten a los desarrolladores y a los gestores de ciberseguridad de una compañía prevenir los ataques de inyección de código.

A nivel de desarrollo, la práctica más recomendada es hacer uso de lo que se denomina *prepared statements* o consultas preparadas. Estas consultas permiten separar la información que introduce el usuario del código dinámico que se genera para realizar la consulta o ejecución del comando. Una consulta

preparada permite al desarrollador escribir el comando o *query* que desea ejecutar incluyendo marcadores de posición que se sustituirán por los datos introducidos por el usuario. Esto hace que, en caso de que se introduzca código malicioso como parámetro, el sistema que almacena los datos no ejecute ese código, sino que lo interprete como un valor más de los parámetros que necesita la consulta.

Desde el punto de vista de la gestión de ciberseguridad, estos mecanismos se pueden implementar en un sistema que se denomina *Web Application Firewall*. Este sistema, como su nombre indica, es un *firewall* que protege a las aplicaciones web de la mayoría de los ataques conocidos, ya sea previniendo ataques de inyección de código, ataques de tipo XSS (*Cross-Site Scripting*) o evitando la exposición de datos sensibles, entre otros. Este sistema WAF es posible desplegarlo como un módulo dentro de la mayoría de API Gateways.

4.4. Mecanismos de control de flujo

Estos mecanismos permiten implementar un control del flujo de peticiones entrantes con el objetivo de disminuir el riesgo de agotamiento de recursos o de ataques de denegación de servicio. Por lo general, estos mecanismos se implementan en elementos que se encuentran por delante del servidor de APIs en la topología de red, ya sean servidores proxy inversos o API Gateways, permitiendo liberar a los servidores de las APIs de la ejecución de estas tareas.

Los servicios de control de flujo o *throttling* almacenan registros de las direcciones IP de origen, el número de peticiones desde estos orígenes, la ventana de tiempo en las que se han realizado estas peticiones e incluso la ubicación del origen y, en base a umbrales definidos por el responsable de seguridad, permiten rechazar estas peticiones o enviar respuestas HTTP con código 429 *Too Many Requests*. Con esto se consigue evitar ataques de denegación de servicio que dejarían inhabilitado el servidor que expone la API REST.

Al igual que los WAF mencionados en el apartado anterior, estos mecanismos de control de flujo se pueden implementar en un API Gateway, ya sea de manera configurable con las opciones por defecto del sistema o instalando un módulo adicional, si fuese necesario.

5. Análisis de soluciones

Tal y como se ha visto en el capítulo anterior, la mayoría de los mecanismos de protección de APIs REST se pueden implementar en un elemento denominado API Gateway. Este elemento empezó a popularizarse con el surgimiento de las arquitecturas de microservicios, ya que surgió la necesidad de centralizar todos los *endpoints* que expone cada microservicio en un único lugar, de manera que cambios que se pudiesen producir en un microservicio no llevasen a la redefinición de código de los clientes de dicho microservicio. Además, al tener centralizado el acceso a estas APIs de múltiples servicios, se pueden aplicar políticas de seguridad comunes para todos los servicios en un solo elemento. A modo de resumen, un API Gateway es un proxy inverso que permite gestionar, configurar y enrutar las peticiones entrantes hacia las APIs expuestas en los diferentes *backends*.

La arquitectura típica de microservicios en la que se incluye un API Gateway es la siguiente:

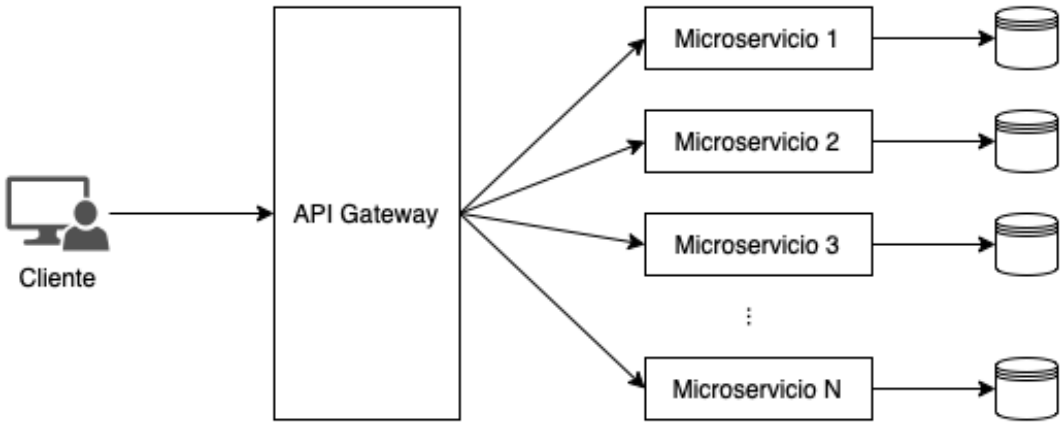


Ilustración 12. Arquitectura de microservicios con API Gateway

5.1. Análisis de Kong Gateway

Kong Gateway es un API Gateway ligero, flexible y con unos tiempos de respuesta muy bajos, desarrollado para ser *cloud-native*. Está pensado para ejecutarse delante de cualquier API REST y su funcionalidad por defecto se puede extender gracias a la instalación de módulos o plugins. Está diseñado para ejecutarse en arquitecturas descentralizadas, incluyendo arquitecturas de nube híbridas y despliegues en varias nubes. Se trata de uno de los líderes del mercado en cuanto a API Gateway (véase ilustración 4).

Kong Gateway se distribuye en dos modos:

- **Open Source:** contiene la funcionalidad básica del API Gateway además de módulos desarrollados por la comunidad en formato *open source*. Incluye una API de administración para gestionar la configuración del API Gateway.

- **Closed source:** este modo se divide a su vez en tres submodos, gratuito, plus y enterprise e incluye funcionalidad adicional, como un gestor del API Gateway en formato de interfaz gráfica de usuario, plugins desarrollados por el equipo de Kong, métricas o permisos de tipo RBAC.

En la siguiente ilustración se puede ver la arquitectura y funcionalidad que incluye cada modo.

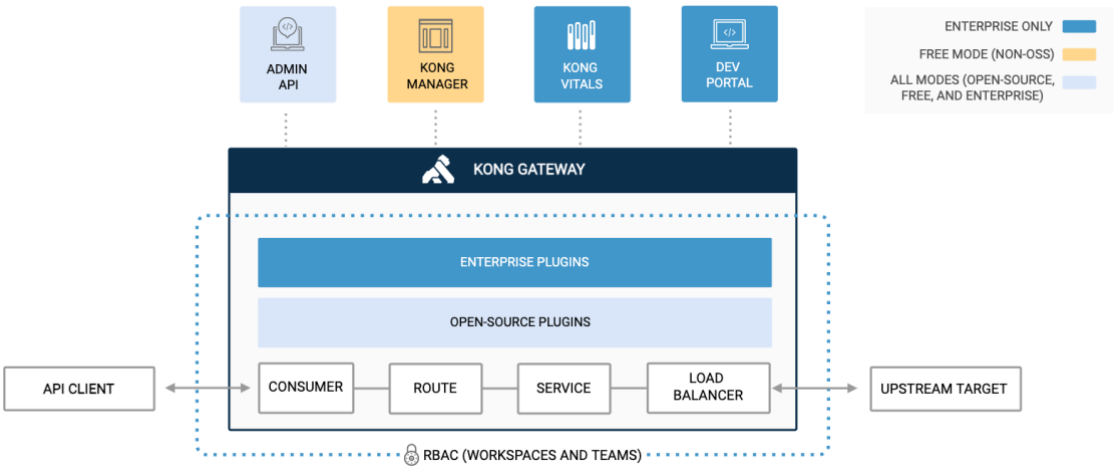


Ilustración 13. Arquitectura de Kong

Al ser un producto desarrollado para ser nativo en la nube, es escalable, elástico y resiliente. Esto quiere decir que permite añadir más nodos o eliminarlos en función de los recursos requerides y disponibles de manera automática y que, en caso de error en un nodo, el resto de los nodos del clúster pueden soportar la carga sin depender del nodo caído.

5.1.1. Conceptos y características de Kong

Kong define los siguientes conceptos y características:

- **Service:** un objeto *Service* es el identificador que utiliza Kong Gateway para invocar las APIs y microservicios que gestiona. Es una entidad que representa la API expuesta o el microservicio. El atributo principal de un servicio es la URL donde el servicio escucha por peticiones.

- **Routes:** las rutas especifican cómo las peticiones entrantes se envían a los *Services*. Un solo *Service* puede tener varias rutas.

- **Consumers:** los consumidores representan usuarios finales de las APIs. Un objeto *Consumer* permite controlar quién puede acceder a las APIs, además de permitir hacer un reporte del tráfico utilizando plugins de *logging*.

- **Kong Manager:** es la herramienta visual que se utiliza para monitorizar y gestionar el Kong Gateway. Se trata de una aplicación web, sólo disponible en la capa gratuita del modo de código cerrado.

- **Admin API:** se trata de una API REST interna cuyo propósito es el de permitir la administración de Kong Gateway. Los comandos API se pueden ejecutar en cualquier nodo del clúster y la configuración se aplicará en todos los nodos de manera consistente.

- **Plugins:** proporcionan un sistema modular para modificar y controlar el Kong Gateway y sus capacidades. Entre el catálogo de plugins se incluyen plugins de control de acceso, de cacheo, de limitación de peticiones y de *logging*, entre otros.

- **Load balancing:** Kong Gateway proporciona dos métodos para el balanceo de carga: directamente basado en DNS o utilizando un balanceo de anillo. Con este último método la adición o eliminación de servicios de *backend* es gestionada por Kong Gateway, haciendo que no sean necesarias actualizaciones en el servidor DNS.

- **Proxy caching:** Kong Gateway proporciona un rendimiento muy alto con tiempos de respuesta muy bajos gracias al almacenamiento en caché de las respuestas de los servicios que se exponen por detrás. Esta caché se puede configurar en base a los métodos de la petición, a los códigos de respuesta, tipo de contenido de la petición, *Consumer* y *backend*. La cache se puede almacenar por un periodo de tiempo configurable. Cuando este tiempo expira, Kong Gateway redirige la petición al *backend* y vuelve a cachear la respuesta, respondiendo con esta caché hasta la siguiente expiración del tiempo. Esta caché se puede almacenar en memoria o en una base de datos Redis.

5.1.2. Modos de despliegue

Kong Gateway proporciona soporte para diferentes modos de despliegue y sistemas operativos. Se puede desplegar tanto contenerizado en Docker, como en Kubernetes o en modo servicio del sistema en los siguientes sistemas operativos:

- Amazon Linux
- CentOS
- macOS
- Debian
- RHEL
- Ubuntu

Por otra parte, también proporciona la capacidad de hacer un despliegue con una base de datos para almacenar las configuraciones de Kong Gateway o sin base de datos, en cuyo caso las configuraciones se almacenan en memoria.

5.1.3. Configuraciones en Kong Gateway

Las configuraciones y definición de las APIs, por norma general, se pueden realizar o bien mediante la API de administración proporcionada por Kong o

mediante la herramienta Kong Manager, aunque existen ciertas configuraciones que se pueden definir en un fichero en formato YAML. Las configuraciones mediante la API de administración se realizan ejecutando peticiones de tipo POST a dicha API. Estas configuraciones pueden almacenarse en una base de datos Redis, en caso de un despliegue con base de datos, o en memoria, si no se desea desplegar una base de datos.

5.2. Análisis de Tyk Gateway

El producto de gestión de APIs de Tyk está dividido en los siguientes servicios o aplicaciones:

- **Tyk Gateway:** se trata del API Gateway de Tyk, el cual es un producto *open source* que permite la gestión de las APIs expuestas por los *backends*.

- **Tyk Pump:** se trata de un purgador de datos analíticos, en formato *open source*, que mueve los datos generados por los nodos de Tyk Gateway a cualquier repositorio o servicio de analítica de datos. Su principal uso es el de mostrar los datos analíticos en el *Tyk Dashboard*.

- **Tyk Dashboard:** es la interfaz gráfica de usuario y la plataforma de análisis de datos de Tyk. Proporciona una interfaz de gestión de fácil uso para gestionar las instalaciones de Tyk además de para realizar un análisis claro y granularizado de los datos de Tyk.

- **Tyk Developer Portal:** es un pequeño portal de gestión de contenidos estáticos que habilita a los desarrolladores a exponer de manera gráfica las APIs que gestiona Tyk, haciendo que los desarrolladores externos a Tyk puedan registrar y conocer la información de cada API gestionada por Tyk.

- **MCDB:** el sistema *Multi Data Centre Bridge* permite una gestión centralizada de múltiples e independientes clústeres de Tyk, así como una transición sin fisuras de APIs entre diferentes entornos, zonas de disponibilidad o nodos segmentados.

- **Tyk Identity Broker:** es un portal en formato microservicio que proporciona un puente entre diferentes sistemas de gestión de identidades y los sistemas Tyk instalados.

Los productos Tyk Dashboard, Tyk Developer Portal y MCDB son productos de código cerrado que requieren una licencia para su uso. Por el contrario, los productos Tyk Gateway, Tyk Pump y Tyk Identity Broker son de código abierto y no requieren licencia para su uso.

Se ha de mencionar que de todos los productos que proporciona Tyk se analizará únicamente Tyk Gateway, por ser el producto correspondiente al API Gateway.

5.2.1. Características de Tyk Gateway

Tyk Gateway es una API Gateway de código abierto que da soporte a la gestión de APIs usando los protocolos REST, SOAP, GraphQL, TCP y gRPC. Este producto se suministra sin ningún bloqueo de funcionalidades, permitiendo a la organización controlar quién accede a las APIs, cuándo y cómo. No cuenta con ninguna dependencia de terceros, a excepción de la base de datos Redis, para permitir almacenamiento de *tokens* y limitación de peticiones distribuidas.

La arquitectura funcional de Tyk Gateway se puede ver en la siguiente ilustración:

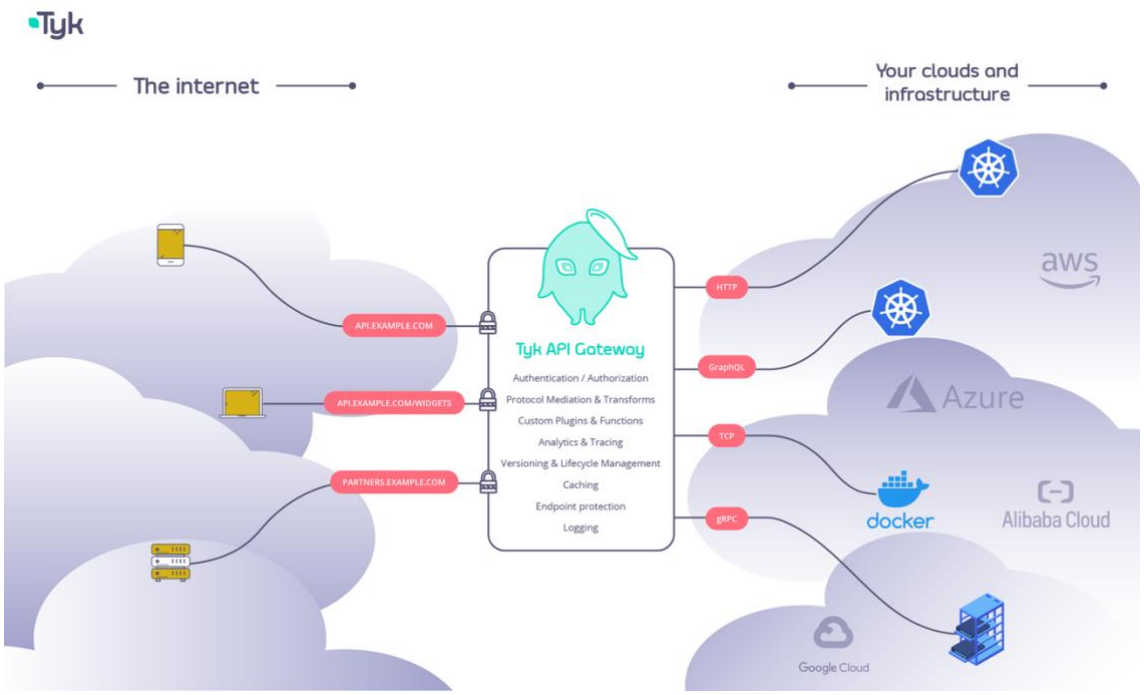


Ilustración 14. Arquitectura funcional de Tyk Gateway

Las características más destacadas de Tyk Gateway son las siguientes:

- Soporta los protocolos de autenticación y autorización más usados hoy en día, como son OAuth2.0, OpenID Connect, autenticación básica HTTP y autenticación con certificado, entre otros.

- Es altamente escalable, tanto horizontal como verticalmente, permitiendo una latencia muy baja y la gestión de miles de peticiones con pocos recursos.

- Permite la extensibilidad de sus funcionalidades mediante el uso de plugins, escritos en cualquier lenguaje de programación, desde Python hasta JavaScript o Go, o cualquier lenguaje que soporte el protocolo gRPC.

- Permite proteger las APIs expuestas con cuotas y limitación de peticiones por cada consumidor.

- Proporciona mecanismos de transformación tanto de peticiones como de respuestas, así como transformación entre protocolos, por ejemplo, entre SOAP y GraphQL.

- Permite gestionar de manera granular los controles de acceso, ya sea por versión de la API o en base a la operación solicitada.

- Gestiona y mantiene listados de *endpoints* permitidos, bloqueados o ignorados por consumidor, para hacer cumplir las medidas de seguridad organizacionales

- Registra datos de uso detallados acerca de quién está utilizando cada API.

- Las configuraciones pueden ser modificadas dinámicamente y el servicio reiniciado sin afectar ninguna petición activa.

5.2.2. Configuraciones en Tyk Gateway

Las configuraciones y definiciones de las API que gestiona Tyk Gateway se almacenan en objetos con formato JSON en una base de datos MongoDB, en caso de utilizar Tyk Dashboard o, en caso de utilizar la API de administración, en un fichero JSON en el directorio */apps* del Gateway.

5.2.3. Modos de despliegue

Tyk Gateway puede ser desplegado en formato contenerizado, utilizando kubernetes o como servicio de sistema operativo. Los sistemas operativos soportados son:

- CentOS
- RHEL
- Debian
- Ubuntu

En la siguiente ilustración se puede ver la arquitectura de despliegue de Tyk Gateway:

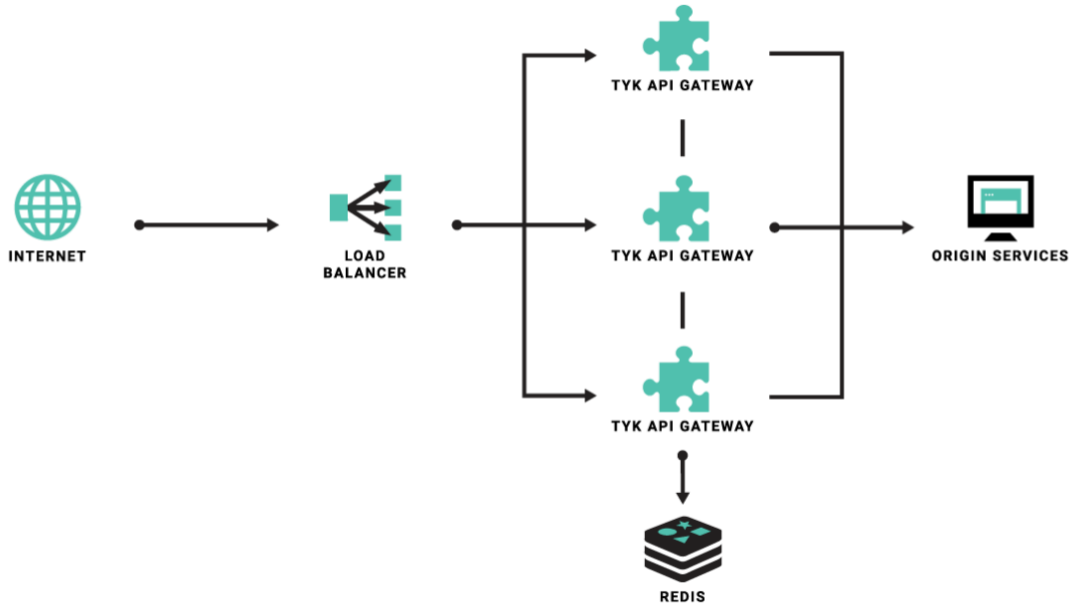


Ilustración 15. Arquitectura de despliegue de Tyk Gateway

5.3. Análisis de WSO2 API Manager

WSO2 API Manager es una plataforma de código abierto que permite construir, integrar y exponer APIs REST, ya sea en la nube, en un servidor propio o en una arquitectura híbrida.

Soporta el diseño, la publicación, la gestión del ciclo de vida, la securización y la visualización de estadísticas de las APIs REST que expongan los *backends*, así como la conexión entre diferentes APIs.

5.3.1. Características y componentes de WSO2 API Manager

En la siguiente ilustración se puede ver la arquitectura de componentes de WSO2 API Manager:

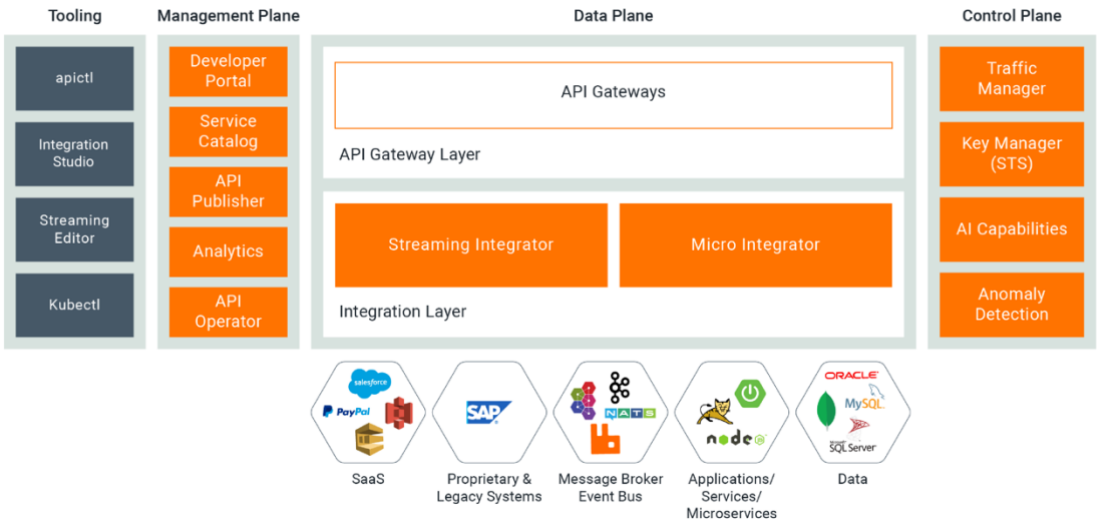


Ilustración 16. Arquitectura de componentes de WSO2 API Manager

Como se puede ver, cuenta con múltiples componentes, de los cuales destacan los siguientes:

- **Management Plane:** se trata del componente donde se crean y gestionan todas las APIs expuestas en WSO2 API Manager. Está compuesto por varios portales (*Publisher*, *Developer Portal* y *Service Catalog*) que permiten a los usuarios o gestores crear y gestionar las APIs, implementar políticas de limitación de peticiones, monitorizar las APIs, monetizar el uso de estas, etc.

    - **API Publisher:** es una interfaz gráfica de usuario de última generación que permite desarrollar, documentar, securizar, testear y versionar las APIs con facilidad. También es el lugar donde se publican, monetizan y aplican limitaciones de ratio de peticiones a las APIs.
    - **API Developer Portal:** es una interfaz web de última generación que permite a los publicadores de APIs publicitar y albergar sus APIs mientras permiten a los consumidores hacer un auto-registro, descubrimiento, evaluación, suscripción y consumo de las APIs de manera segura y sencilla.

- **Data Plane:** es el lugar donde las APIs creadas en el *Management Plane* son expuestas a los consumidores públicos. Es el elemento que actúa a modo de proxy para las llamadas a las APIs, y proporciona características adicionales como seguridad reforzada, limitación de peticiones, procesador de mensajes y transformación de protocolos de transporte, entre otras. Está compuesto por los siguientes elementos:

    - **API Gateway:** en este punto se pueden definir qué métodos de autenticación se desean utilizar, qué tipo de datos se van a analizar, qué contenido va a recibir y enviar cada API y, en caso de ser necesario, qué transformaciones se deben realizar a los

mensajes. En despliegues en la nube permite un escalado en base a la demanda de requisitos.
    - **Micro Integrator:** se trata de un motor de mensajería que actúa a modo de bus integrador. Permite la integración de diferentes microservicios en entornos descentralizados y en la nube. Realiza tareas de enrutado de peticiones y transformación de mensajes.
    - **Streaming Integrator:** este elemento es un procesador de flujo de datos que integra estos flujos de datos entre diferentes elementos y ejecuta acciones en base al flujo. Sus tareas son las de capturar, analizar, procesar, entender y actuar sobre flujos de datos y eventos en tiempo real. Entre las funciones con las que cuenta está la transformación de mensajes, el enriquecimiento de respuestas, la correlación y agregación de eventos y la limpieza de estos flujos.

- **Control Plane:** en este punto es donde se realizan las validaciones de seguridad de las APIs, la generación de API keys y donde se toman las decisiones de limitación de ratio de peticiones. Está compuesto por los siguientes elementos:

    - **Key Manager:** se trata del proveedor de identidades para WSO2 API Manager. Actúa como servicio de token seguro o *Secure Token Service (STS).* Soporta los protocolos OAuth2.0, autenticación básica de HTTP, Mutual TLS y autenticación basada en API Key. Permite, a su vez, la integración con proveedores de identidad externos para la autenticación mediante OAuth2.0.
    - **Traffic Manager:** ayuda a los usuarios y gestores de las APIs a regular el tráfico de las APIs y permitir que estén siempre disponibles con diferentes niveles de servicio, así como a prevenir ataques de denegación de servicio. Actúa en tiempo real basándose en direcciones IP de origen, cabeceras HTTP, campos del *token* JWT o parámetros de la URL.

5.3.2. Configuraciones en WSO2 API Manager

La definición de APIs y configuraciones se pueden realizar mediante las interfaces gráficas de usuario disponibles en el *Management Plane* o a través de una herramienta de línea de comandos denominada *apictl*. Por defecto, estas definiciones y configuraciones se almacenan en una base de datos H2.

5.3.3. Modos de despliegue

WSO2 soporta la instalación y despliegue contenerizado en Docker, a través de kubernetes o similar y como servicio tanto de sistemas operativos basados en Linux/UNIX como de Windows. Los sistemas operativos que soporta son:

- Windows 2016
- Windows Server 2019
- Ubuntu 18.04
- RHEL 7.0
- CentOS 7.4 y 7.5

6. Solución escogida

En base a las características que proporcionan, se puede ver que las tres soluciones analizadas proporcionan prácticamente los mismos mecanismos de autenticación y autorización basados en estándares, funciones como limitación de ratio de peticiones, transformación de mensajería y funciones como la escalabilidad.

6.1. Ventajas e inconvenientes de Kong Gateway

Entre las ventajas se pueden enumerar las siguientes:

- Posición líder según la consultora Gartner (véase ilustración 4)
- Facilidad en el despliegue y configuración
- Extensibilidad gracias a los plugins y módulos adicionales
- Cacheado de peticiones para dar unos tiempos de respuesta muy bajos
- Soporte para múltiples sistemas operativos
- Diseño *cloud-native*
- Posibilidad de despliegue sin necesidad de base de datos

Los principales inconvenientes son:

- Ciertas funciones sólo están disponibles tras el pago de una licencia, como la limitación de la ratio de peticiones avanzada o ciertos plugins que pudiesen ser útiles
- Soporte sólo para APIs de tipo REST de manera nativa
- Falta de una interfaz gráfica de usuario para gestionar y monitorizar las APIs en su versión *open source*

6.2. Ventajas e inconvenientes de Tyk Gateway

En cuanto al producto Tyk Gateway, las principales ventajas son:

- Facilidad de uso y de despliegue
- Extensibilidad por medio de plugins
- Integración con herramientas propias de análisis de datos y gestión de identidades
- Soporte para APIs de tipo REST, SOAP, gRPC y GraphQL de manera nativa
- Diseño *cloud-native*
- Soporte para múltiples lenguajes de programación para desarrollo de módulos

Los inconvenientes son:

- Necesita una base de datos para su funcionamiento
- Características como *Single Sign On*, gestión de permisos *RBAC* o el uso de *Tyk Manager* requieren el pago de una licencia

6.3. Ventajas e inconvenientes de WSO2 API Manager

Por último, las ventajas de WSO2 API Manager son:

- Software completo de código abierto, todas las características están disponibles sin pagos
- Mayor flexibilidad en cuanto a configuraciones
- Mayor número de funcionalidades proporcionadas de caja
- Soporte para APIs REST y GraphQL

En cuanto a los inconvenientes:

- Mayor complejidad en la definición de configuraciones
- No diseñado para ser *cloud-native*
- Menores tiempos de respuesta que sus competidores
- La extensibilidad no se basa en módulos o plugins. Requiere desarrollo en el código fuente propio del aplicativo
- No incluye balanceador de carga integrado

Los dos candidatos más idóneos son Kong y Tyk, ya que proporcionan mayores facilidades para la instalación y despliegue al contar con menos dependencias de terceros, ya que no requieren disponer de un balanceador de carga externo, además de contar con una mejor posición en el mercado. Entre estos dos candidatos existen muy pocas diferencias que inclinen la balanza hacia un lado o hacia otro en el contexto de este trabajo. Si se necesitase flexibilidad en la elección de lenguaje de programación para el desarrollo de módulos se escogería Tyk. Por otro lado, si se quiere garantizar unos tiempos de respuesta muy bajos y una comunidad de desarrolladores más amplía, se elegiría Kong. Dado que estos son aspectos que están fuera del ámbito del proyecto, se decide utilizar **Kong Gateway** por el simple hecho de que se encuentra en una posición de líder en el cuadrante mágico de Gartner [10].

7. Implementación de la solución escogida

7.1. Instalación y configuración de máquina virtual

En el desarrollo de este trabajo se ha contado con varias tecnologías base que sirven de apoyo para la implementación de los sistemas y elementos que permiten obtener un resultado práctico de la securización de APIs REST. La primera tecnología base ha sido el software Parallels Desktop, el cual permite virtualizar diferentes sistemas operativos utilizando como *host* el sistema operativo Mac OS.

Se ha decidido utilizar el sistema operativo Parrot OS [28] como sistema operativo base por su facilidad de uso, la cantidad de herramientas preinstaladas del ámbito de la ciberseguridad y la mayor comodidad del alumno en el uso de este sistema operativo. La instalación tanto del sistema operativo como del virtualizador Parallels queda fuera del ámbito de este trabajo, por ser triviales e independientes del resultado de este.

Por otra parte, para la ejecución de los sistemas relacionados con el API Gateway se ha decidido utiliza Docker [29], una tecnología basada en contenedores que permite la creación y virtualización de aplicaciones completas, incluyendo todas sus dependencias, compartiendo el *kernel* de Linux del sistema operativo en el que está instalado.
Docker permite disponer de un aislamiento e independencia de los procesos que se ejecutan dentro de cada contenedor, de manera que se puedan llegar a emular varios servidores dentro de una misma máquina. Este sistema se basa en la utilización de imágenes precompiladas que contienen las instrucciones necesarias y suficientes para la instalación y despliegue de un servicio y todas sus dependencias. También permite la definición de redes para interconectar los contenedores, mapeo de puertos para poder hacer uso de múltiples servicios que nativamente escuchan en el mismo puerto sin que surjan conflictos, así como volúmenes que permitan persistir datos de aplicaciones *stateful*.

Para la instalación de Docker se ha seguido la guía oficial proporcionada por el fabricante [30], teniendo en cuenta que el sistema operativo utilizado está basado en Debian. Los pasos más relevantes se describen a continuación:

- Obtención de la clave GPG del repositorio de paquetes de Docker y adición de la URL del repositorio de Docker al gestor de paquetes del sistema operativo.

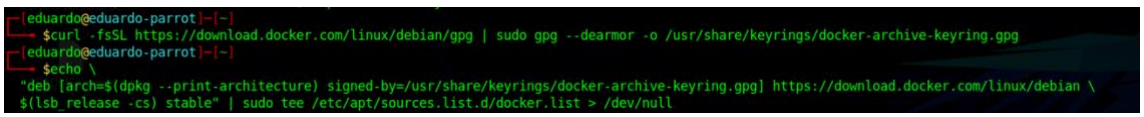


Ilustración 17. Instalación de Docker I

- Actualización del gestor de paquetes del sistema operativo e instalación de los paquetes necesarios *docker-ce*, *docker-ce-cli* y *containerd.io*.

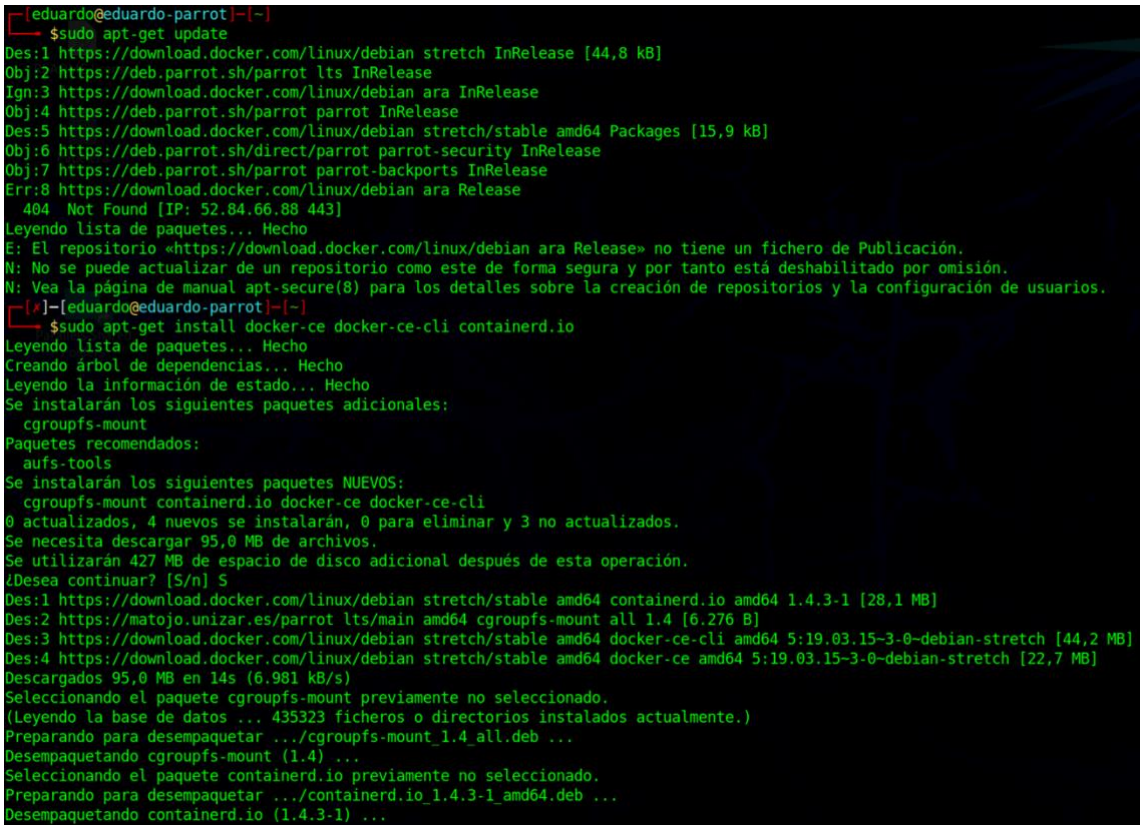


Ilustración 18. Instalación de Docker II

- Configuración del usuario local en el grupo preconfigurado *docker* para evitar utilizar el comando *sudo* en cada ejecución de Docker.

Ilustración 19. Instalación de Docker III

7.2. Diagrama de diseño

La implementación de esta solución consta de cuatro elementos fundamentales:

- Kong Gateway
- Base de datos para almacenamiento de configuraciones
- API REST
- Cliente

El diagrama de diseño se puede ver en la siguiente figura.

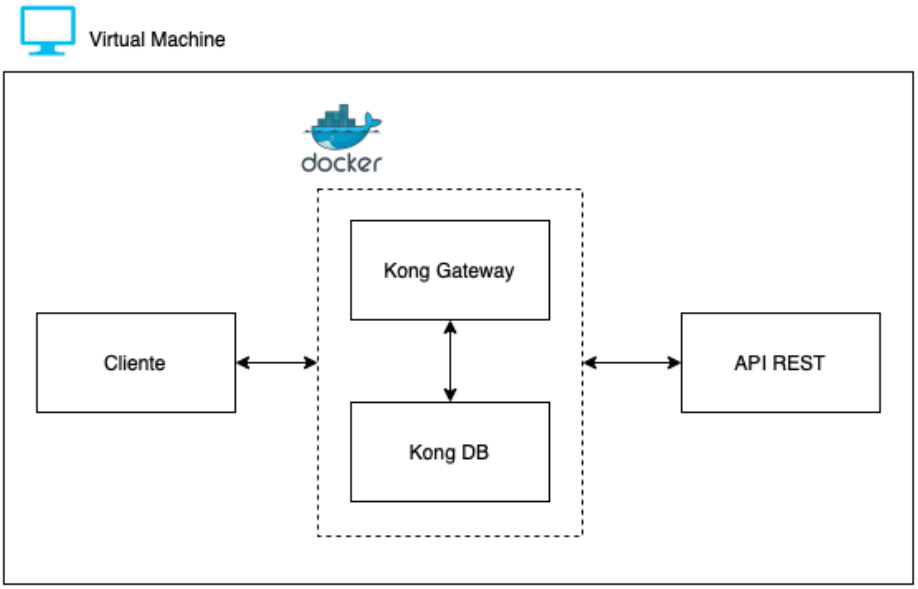


Ilustración 20. Diagrama de diseño

Como se puede ver en el diagrama, tanto el servicio de Kong Gateway como el de Kong DB se ejecutan sobre Docker, mientras que el API REST y el cliente que consuma este API se ejecutan sobre el propio sistema operativo.

7.3. Instalación del API Gateway Kong

Kong proporciona varias imágenes Docker listas para su uso y ejecución, de manera que no sea necesaria la instalación de ninguna dependencia ni la propia configuración del sistema operativo. Proporciona una guía detallada sobre la instalación y uso de su plataforma utilizando Docker como sistema base [31]. Kong proporciona dos modos de implementación, uno utilizando una base de datos para almacenar las configuraciones y otro sin hacer uso de base de datos, en cuyo caso las configuraciones se almacenan en archivos en formato YAML. En este caso se ha decidido hacer uso de la implementación con base de datos, ya que proporciona mayor funcionalidad y permite realizar cambios en caliente sin necesidad de tener que reiniciar el servicio.

7.3.1. Creación de red en Docker y ejecución de base de datos.

El primer paso para la instalación de Kong Gateway ha sido la creación de una red Docker para permitir la comunicación entre el contenedor del API Gateway y el contenedor de la base de datos. Se le ha dado el nombre de *kong*-net a dicha red. Para ello se ha ejecutado el comando que se ve en la siguiente ilustración.

```
[eduardo@eduardo-parrot]-[~]
    $ docker network create kong-net

b91afc6c28f090e988610c5b4dd10b3362e5d230499439cf8302e4b5d8093546
```

Ilustración 21. Creación de red Docker

A continuación, se ha ejecuta el comando *docker run* para el despliegue de la base de datos. Se ha utilizado la base de datos PostgreSQL en su versión 9.6 por ser la recomendada por el fabricante del API Gateway en su guía de

instalación. Los parámetros y variables de entorno utilizados han sido los siguientes:

- **--name:** identifica el nombre del contenedor. En este caso se ha decidido llamarlo kong-database.
- **--network:** identifica la red docker a la que se conectará este contenedor. En este caso se utilizará la red anteriormente creada *kong-net*.
- **-p:** con este parámetro se indica la exposición de puertos del contenedor a puertos del host. Es necesaria la definición de este parámetro si se desea acceder al servicio del contenedor desde el exterior del servicio Docker. En el caso de este trabajo se ha mapeado el puerto 5432 del contenedor al puerto 5432 del sistema operativo.
- **Variable de entorno POSTGRES_USER:** define el nombre de usuario que se creará y utilizará para la conexión a la base de datos por parte del API Gateway.
- **Variable de entorno POSTGRES_DB:** define el nombre de la base de datos que se utilizará para el almacenamiento de las configuraciones.
- **Variable de entorno POSTGRES_PASSWORD:** define la contraseña del usuario utilizado por el API Gateway para la conexión a la base de datos.

En la siguiente ilustración se puede ver la ejecución y resultado del comando.

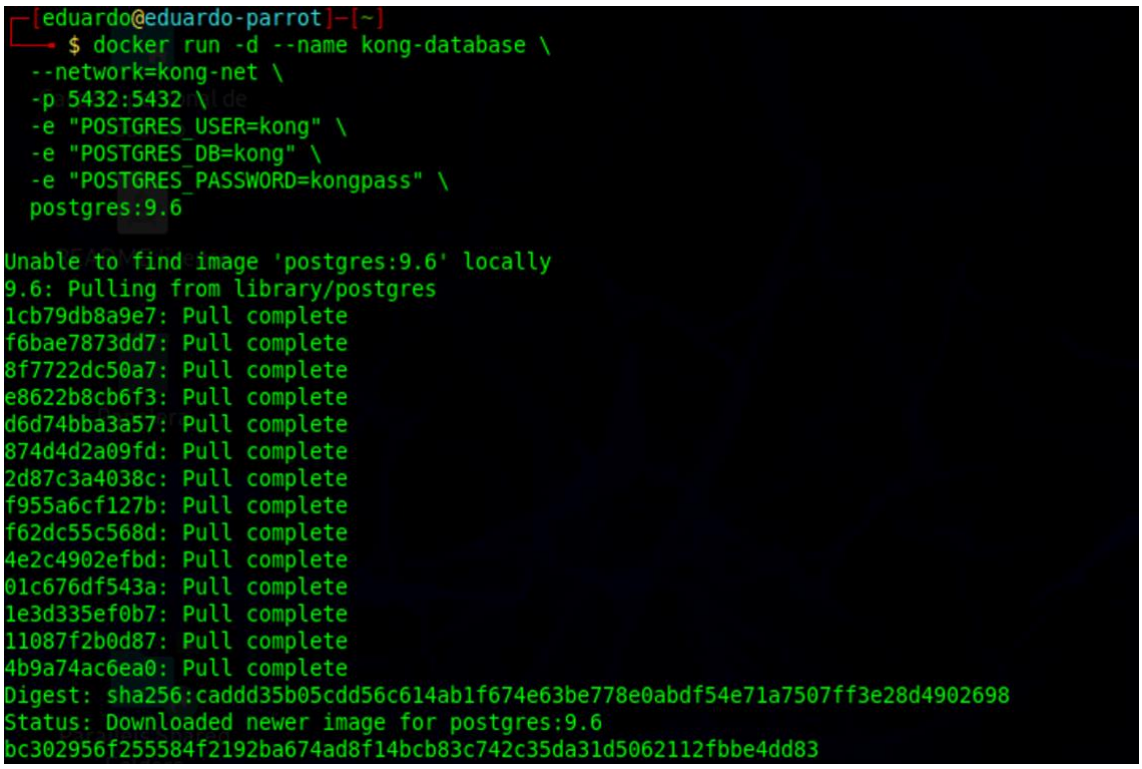


Ilustración 22. Despliegue de la base de datos

7.3.2. Preparación de la base de datos

Antes de realizar el despliegue del API Gateway es necesario hacer una preconfiguración de la base de datos. Esta preconfiguración realiza, entre otras cosas, la carga de esquemas y servicios o plugins predefinidos por el API

Gateway. Para ello, se ha ejecutado el comando que se puede ver en la siguiente ilustración.

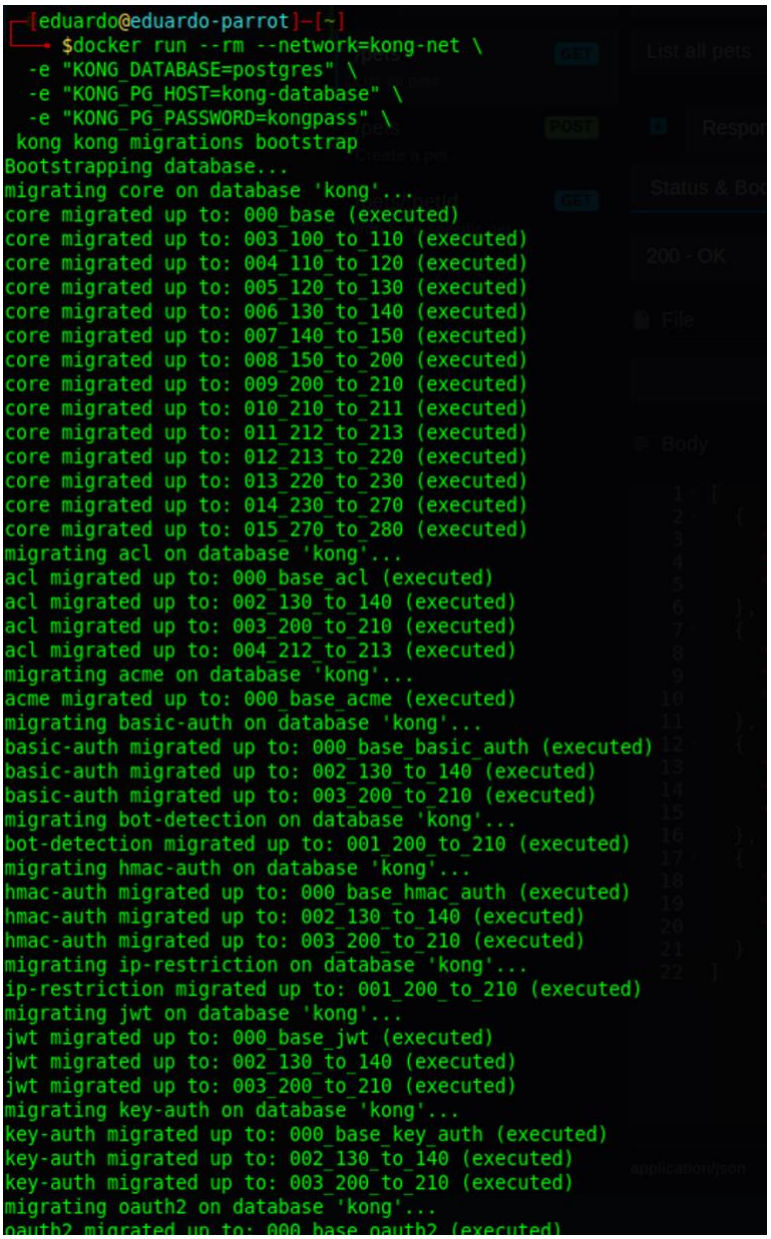


Ilustración 23. Preparación de la base de datos

Los parámetros y variables de entorno más relevantes del anterior comando son:

- **Variable de entorno KONG_DATABASE:** define el tipo de base de datos utilizada para almacenar las configuraciones. En este caso, ya que la base de datos escogida es PostgreSQL, el valor se ha definido como *postgres*.
- **Variable de entorno KONG_PG_HOST:** define el nombre del contenedor con la base de datos PostgreSQL que está conectado a la red *kong-net*.
- **Variable de entorno KONG_PG_PASSWORD:** define la contraseña con la que se debe conectar a la base de datos, haciendo uso del usuario por defecto *kong*.
- **"kong migrations bootstrap":** el primer argumento *kong* identifica la imagen Docker que se utilizará. Los siguientes parámetros *kong*

migrations bootstrap identifican el comando que se ejecutará al desplegar la imagen *kong*, el cual permite lanzar la preparación de la base de datos.

7.3.3. Despliegue del API Gateway

Por último, con el comando que se puede ver en la siguiente ilustración se realiza el despliegue y ejecución del API Gateway.

```
[eduardo@eduardo-parrot]-[~]
 $docker run -d --name kong-gateway \
--network=kong-net \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=kong-database" \
-e "KONG_PG_USER=kong" \
-e "KONG_PG_PASSWORD=kongpass" \
-e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
-e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \
-e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_LISTEN=0.0.0.0:8001, 0.0.0.0:8444 ssl" \
-p 8000:8000 \
-p 8443:8443 \
-p 127.0.0.1:8001:8001 \
-p 127.0.0.1:8444:8444 \
kong

ade6738d92c24a883c7714dbf5b196611e4d0883c41576dc76be94e5e4520ec9
```

Ilustración 24. Despliegue de API Gateway

Los parámetros y variables de entorno más relevantes del comando y que no se han explicado anteriormente son los siguientes:

- **Variables de entorno *_LOG**: todas las variables de entorno con el patrón *_LOG permiten definir las rutas a las que se redirigirán los logs de accesos y errores generados por el API Gateway. En este caso, todos los logs se redirigen a la salida estándar, la cual es salida por consola.
- **Variable de entorno KONG_ADMIN_LISTEN**: define las interfaces y puertos por los que escucha el API de administración del API Gateway, definiendo además si se utiliza protocolo SSL para el establecimiento de la conexión.

Una vez desplegados tanto la base de datos como el API Gateway, se comprueba con el comando *docker ps* la correcta ejecución de ambos servicios.

```
[eduardo@eduardo-parrot]-[~]
 $docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
        NAMES
ade6738d92c2        kong                "/docker-entrypoint.…"   About a minute ago  Up About a minute (healthy)
44/tcp   kong-gateway
121512cd4958        postgres:9.6        "docker-entrypoint.s…"   2 minutes ago       Up 2 minutes
        kong-database
```

Ilustración 25. Comprobación de ejecución de Servicios

Por último, se prueba a acceder al *endpoint* llamado *services* proporcionado por el API Gateway para comprobar que se obtiene respuesta.

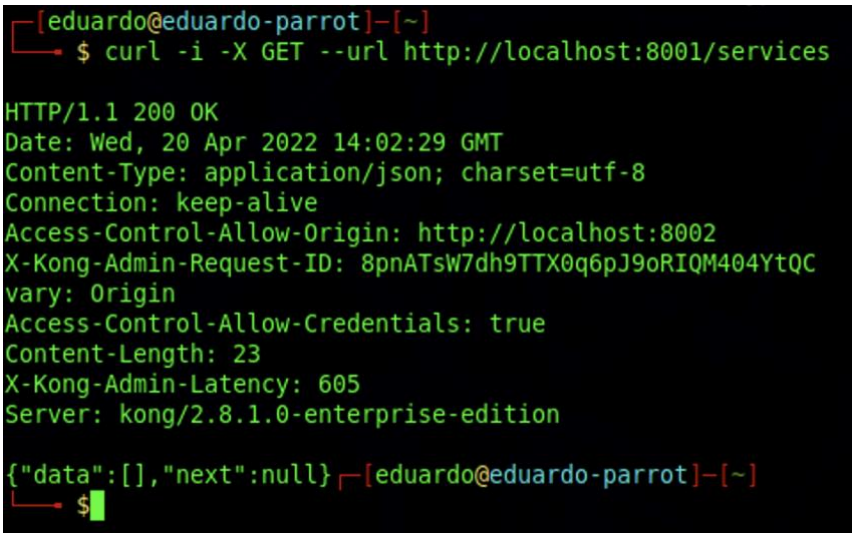


Ilustración 26. Comprobación de escucha del API Gateway

7.4. Instalación de servicio de Mock API

Para poder emular una API REST es necesario hacer uso de algún servicio que permita la definición e implementación de APIs REST sin lógica, es decir, que devuelvan datos falsos, hardcodeados o aleatorios. Esto se conoce normalmente como un *mock*, o imitación. Existen múltiples alternativas que proporcionan este servicio, desde alternativas Cloud hasta alternativas desplegadas en el propio *host* del desarrollador. En este caso se hará uso de una solución ampliamente utilizada entre desarrolladores de APIs REST, la cual es Mockoon [32]. Se trata de una solución de código abierto que permite el despliegue de servidores locales que emulan un servicio de API REST, dentro del cual se pueden definir los esquemas de las APIs desarrolladas, respuestas esperadas, lógica para devolución de códigos de error, etc. Esto permite a los desarrolladores proporcionar el esquema básico de las APIs que se están desarrollando con los que otros sistemas se pueden integrar sin la necesidad de que la API esté completamente desarrollada, agilizando los procesos de integración entre dichos sistemas.

Para la instalación de Mockoon se ha utilizado el gestor de paquetes *snap* [33], ya que es la manera más sencilla de instalarlo. Simplemente ejecutando el siguiente comando se consigue instalar Mockoon:

```
sudo snap install mockoon
```

Mockoon permite la importación de definiciones de APIs en un formato estándar llamado OpenAPI [34], el cual mediante el intercambio de ficheros en formato JSON permite compartir la especificación completa de cualquier API, facilitando las tareas de integración entre los diferentes equipos desarrolladores. OpenAPI proporciona ejemplos de especificación, concretamente un ejemplo de una API

para la gestión de una tienda de mascotas [35]. Se ha utilizado dicho ejemplo para realizar las pruebas de este trabajo.

En las siguientes ilustraciones se pueden ver los endpoints definidos en Mockoon gracias al ejemplo de la tienda de mascotas.

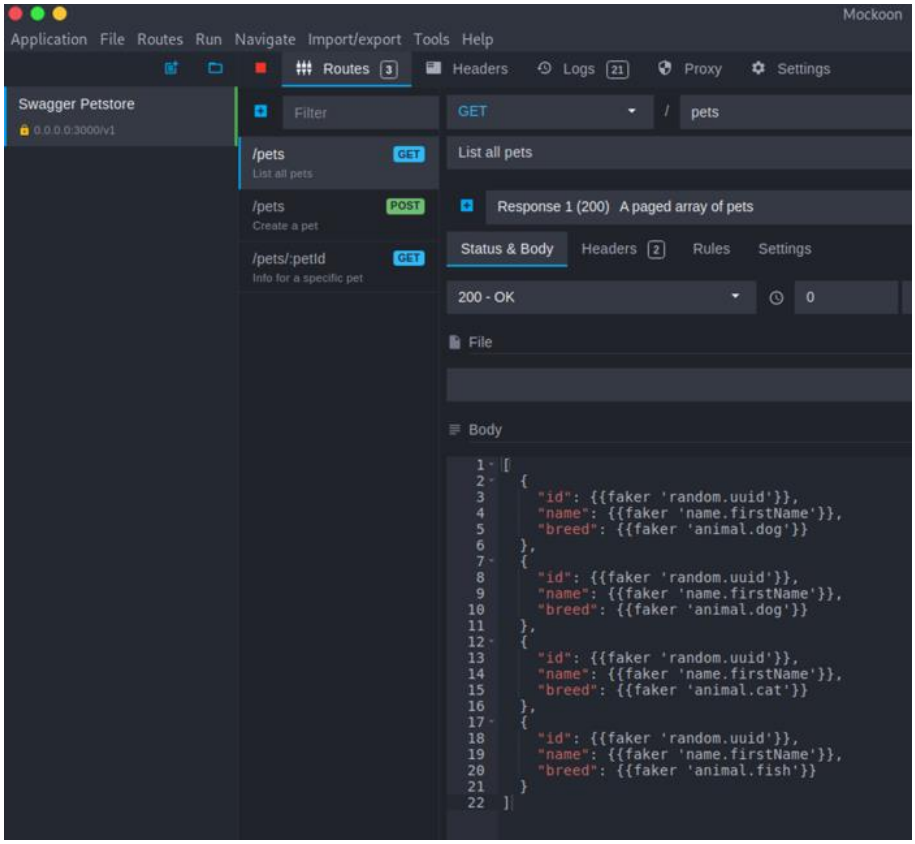


Ilustración 27. Endpoints de Mockoon I

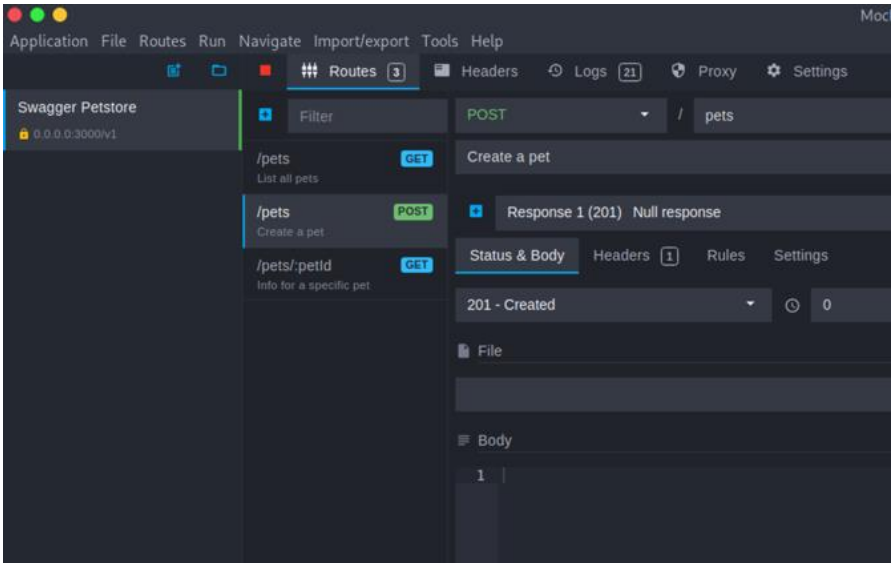


Ilustración 28. Endpoints de Mockoon II

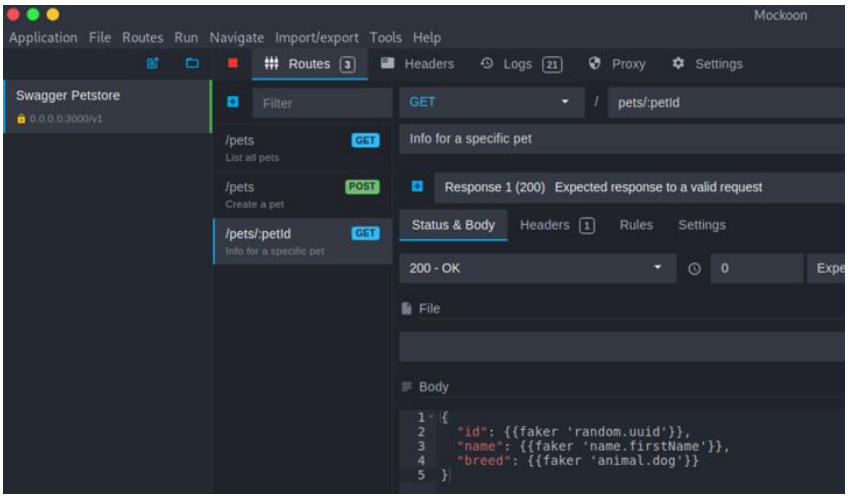


Ilustración 29. Endpoints de Mockoon III

Como se puede ver, este servicio *mock* cuenta con tres *endpoints*. El primero de ellos es el *endpoint* ***GET /pets***, en el cual se emula una consulta a la tienda de mascotas para devolver el listado completo de animales presentes en la tienda. El segundo *endpoint* es ***POST /pets***, gracias al cual se permite la creación de una mascota en la tienda. Por último, el tercer *endpoint* es ***GET /pets/:petid***, donde *petid* es el identificador de una mascota. Este endpoint devuelve información específica sobre la mascota consultada.

Es importante mencionar que estos endpoints se sirven utilizando el protocolo HTTPS, ya que es un requerimiento del protocolo de autenticación OAuth2.0, el cual se usará más adelante. Para ello, Mockoon proporciona una opción de caja para habilitar la escucha por SSL. Al habilitar esta opción se genera un certificado auto firmado que se expone en el *socket* en el que escucha Mockoon.

En la siguiente ilustración se puede ver el ejemplo de llamada a uno de los *endpoints* expuestos.

Ilustración 30. Ejemplo de llamada sin API Gateway

7.5. Configuración del API Gateway

Tal y como se ha mencionado en la sección 5.1.1 Conceptos y características de Kong, este API Gateway cuenta con cuatro elementos fundamentales para la gestión de APIs REST: servicios, rutas, consumidores y plugins. Los servicios permiten tener una definición y representación de un microservicio o API REST dentro de Kong; las rutas, por su parte, permiten definir el *endpoint* que se deberá invocar para poder hacer peticiones a un servicio; los consumidores permiten tener un identificador de cada cliente que vaya a realizar llamadas a un servicio; los plugins permiten añadir capas de funcionalidad o seguridad extras a cada servicio, ruta o de manera global.

Debido a que se está haciendo uso de una base de datos para el almacenamiento de las configuraciones, todos los cambios se realizarán mediante peticiones REST a la API de administración.

7.5.1. Definición del servicio

La primera tarea que se debe realizar es la definición del servicio, es decir, crear un identificador y una representación del servicio que expone la API de la tienda de mascotas.

A continuación, se puede ver la creación de dicho servicio.

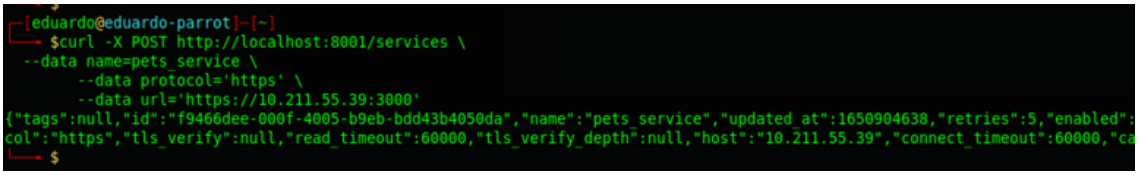


Ilustración 31. Creación de servicio en API Gateway

Con este comando se define un servicio llamado *pets_service*, cuyo protocolo de comunicación es *https* y que está escuchando en la URL *https://10.211.55.39:3000*, siendo esta IP la IP asignada a la máquina virtual en la red LAN.

7.5.2. Definición de la ruta del servicio

Para que el API Gateway pueda exponer el servicio recientemente creado, es necesario definir una ruta, de manera que el API Gateway sea capaz de enviar cada petición entrante a cada microservicio dependiendo del *endpoint* que se invoque.

En la siguiente ilustración se puede ver el resultado de la ejecución del comando.

Ilustración 32. Creación de ruta en API Gateway

Con el comando anterior se define una ruta para el servicio *pets_service* cuyo nombre es *pets*, definiendo el *path* en el que se expondrá el servicio como */mockpets* y permitiendo sólo los métodos *GET* y *POST*.

Tras haber definido tanto el servicio como la ruta, es posible hacer una prueba funcional para validar que las peticiones se están redirigiendo correctamente al servicio Mockoon.

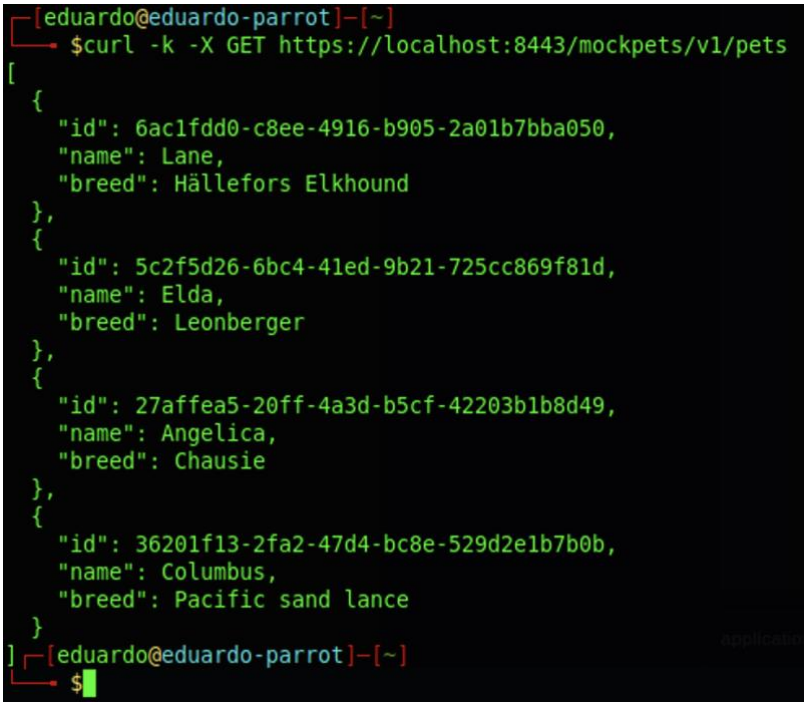


Ilustración 33. Test funcional básico I

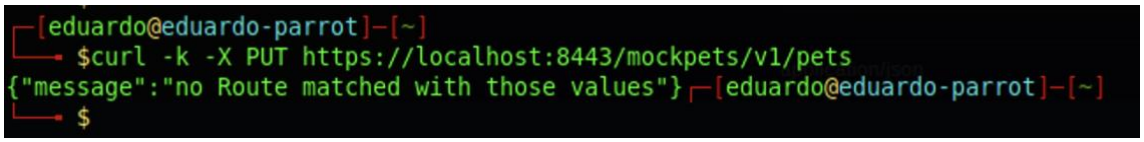


Ilustración 34. Test funcional básico II

Como se puede ver en las anteriores ilustraciones, la petición se redirige correctamente a la API REST, comprobando además que el API Gateway no redirige la petición si el método no es uno de los registrados para la ruta.

7.5.3. Creación de consumidor

Para poder asignar credenciales a un cliente o para hacer efectivas las políticas de seguridad de los servicios, es necesario crear un consumidor del servicio. Para la creación de un consumidor basta con definir un nombre de usuario para eso consumidor o un identificador único arbitrario.

En la siguiente imagen se puede ver cómo se ha realizado la creación del consumidor.

Ilustración 35. Creación de consumidor

7.6. Configuraciones de seguridad

En los siguientes subapartados se procede a describir las configuraciones de seguridad que se han llevado a cabo.

7.6.1. Gestión de certificados

Como se ha mencionado, Kong Gateway expone por defecto el servicio HTTPS en el puerto 8443 haciendo uso de un certificado auto firmado y auto generado. En este caso se procede a la creación de un certificado auto firmado, ya que no se dispone de una autoridad certificadora para la firma de validación del certificado.

El primer paso consiste en la generación de la clave privada de firma del certificado. Este paso genera un archivo *kong.key* que contiene la clave privada de firma en formato PEM.

Ilustración 36. Creación de clave privada de firma

A continuación, se crea la petición de firma del certificado.

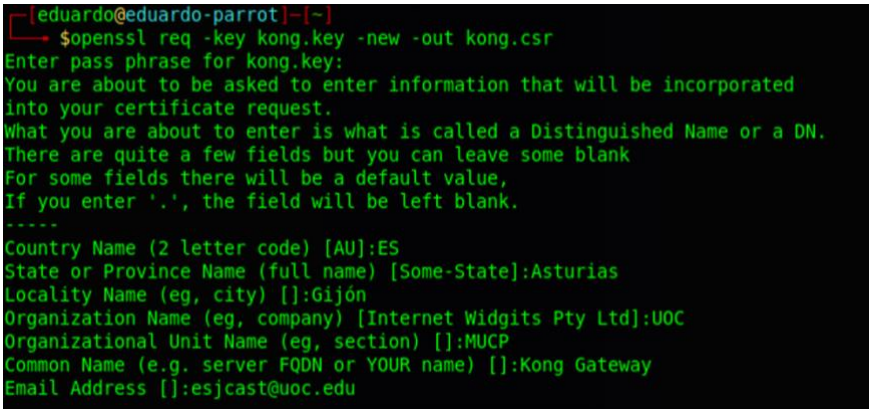
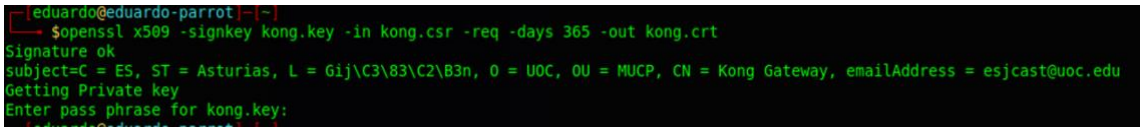


Ilustración 37. Petición de firma de certificado

Esta petición de firma genera un archivo *kong.csr*, el cual se debería enviar a una autoridad certificadora para obtener un certificado públicamente confiable. En este caso, como se ha mencionado, se auto-firma esta petición, como se puede ver en la siguiente ilustración.



Este comando genera un archivo *kong.crt*, el cual representa el certificado de firma generado en formato PEM. Una vez que se tienen tanto el certificado como la clave, se puede proceder al cambio del certificado con el que escucha Kong Gateway. En las siguientes ilustraciones se puede ver cómo se ha configurado el certificado para la interfaz *localhost*.

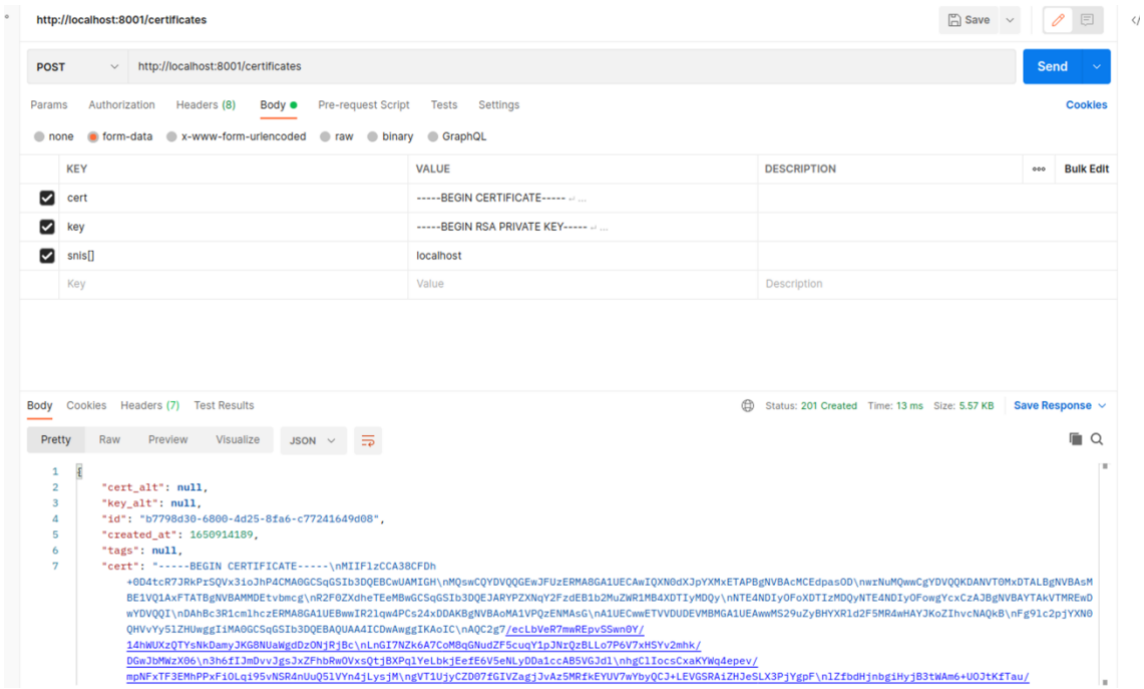


Ilustración 38. Configuración de certificado HTTPS

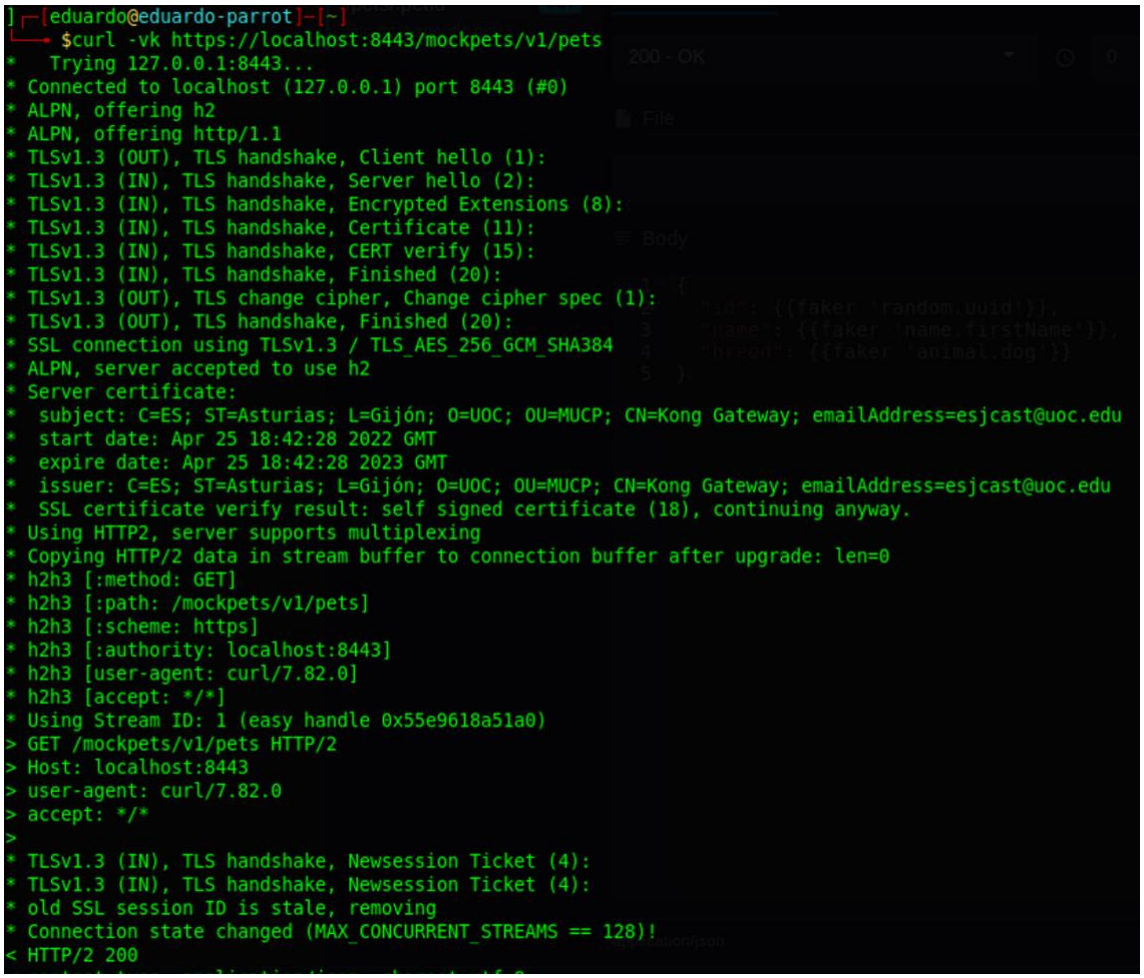


Ilustración 39. Testeo de configuración de certificado

7.6.2. Limitación de la ratio de peticiones

Kong Gateway dispone, por defecto, de un plugin básico para limitar el número de peticiones que se pueden enviar a un servicio REST. Este plugin permite limitar el número de peticiones que se reciben por segundo, minuto, hora, día e incluso mes. También proporciona opciones para limitar las peticiones por IP, consumidor, servicio, credencial utilizada, cabecera o *path*.

En el caso de este trabajo se ha limitado la ratio de peticiones a 5 por minuto, de manera que se disponga de suficiente tiempo para realizar la prueba funcional correspondiente.

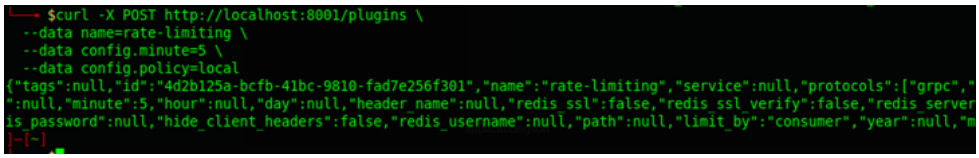


Ilustración 40. Configuración de limitación de peticiones

A continuación, se puede ver cómo, tras haber realizado más de 5 peticiones al minuto, el API Gateway rechaza las siguientes peticiones.

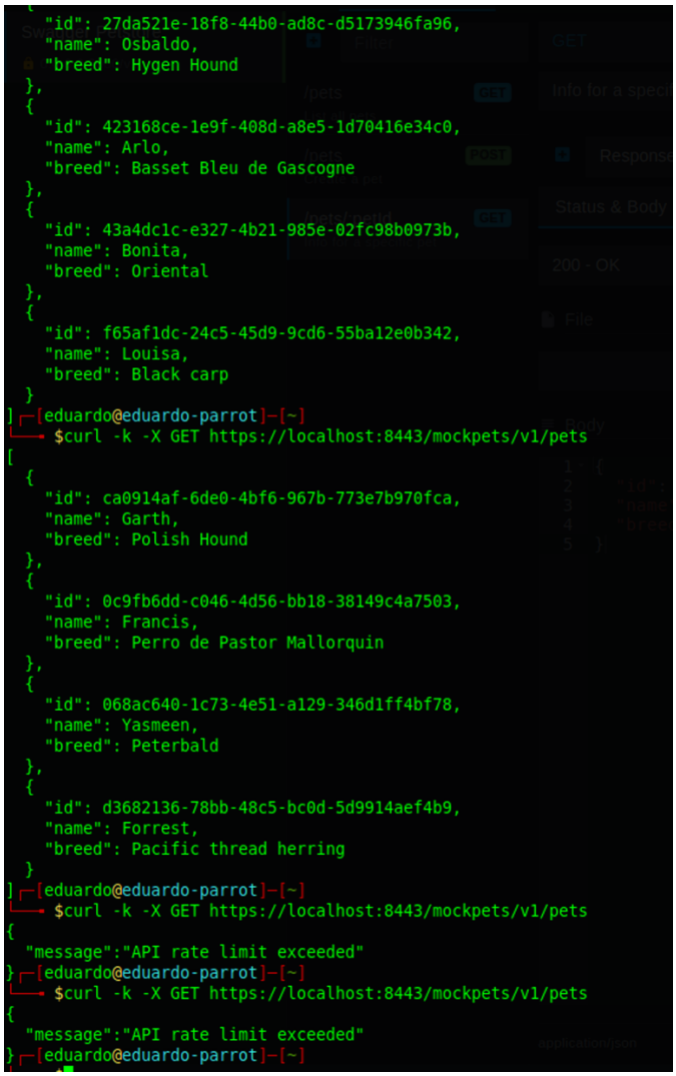


Ilustración 41. Prueba limitación de ratio de peticiones

7.6.3. Activación de autenticación básica

Tal y como se ha comentado anteriormente, uno de los métodos más comunes de autenticación es el uso de la tecnología de autenticación básica por HTTP. Gracias a este estándar, es posible securizar el acceso a una API REST mediante una credencial consistente en un nombre de usuario y una contraseña asociada. Kong Gateway incluye por defecto varios módulos de autenticación, entre los que se encuentran el módulo de autenticación básica.

Para activar este módulo, se ha ejecutado el siguiente comando.

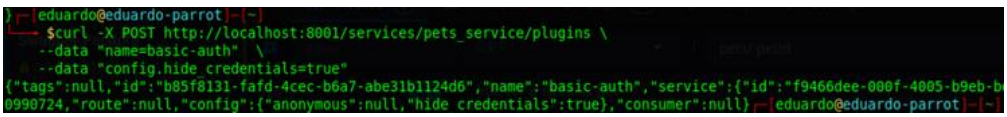


Ilustración 42. Activación de autenticación básica

El parámetro *config.hide_credentials* permite ocultar o no las credenciales al servicio que expone la API REST. En este caso, dado que el servicio Mockoon no tiene activa la autenticación, se ha configurado con valor *true*.

El siguiente paso consiste en la generación de una credencial para un cierto consumidor. A modo de ejemplo, esta credencial consta de un nombre de usuario *Aladdin* y una contraseña *OpenSesame*.

Ilustración 43. Creación de credencial básica

Por último, se comprueba que, enviando esta credencial tal y como se especifica en el estándar *HTTP Basic Authentication*, se consigue obtener respuesta por parte del API Gateway.

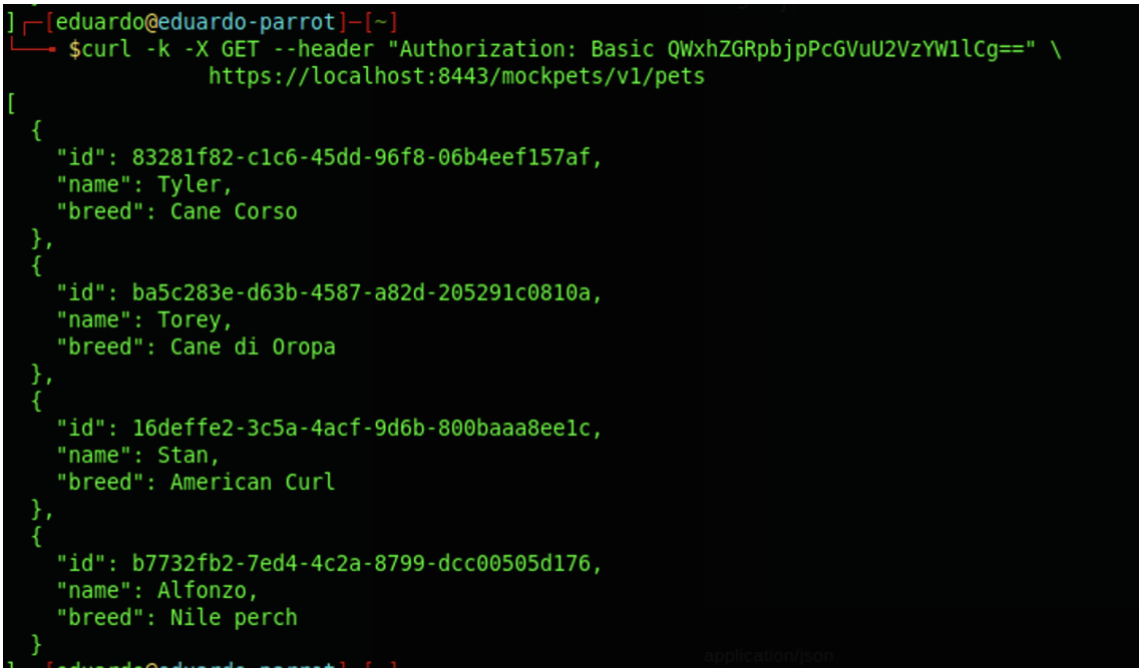


Ilustración 44. Prueba de autenticación básica

7.6.4. Activación de autenticación mediante API Key

Otro método de autenticación para la protección de APIs REST muy utilizado hoy en día es la autenticación mediante una API Key. Las ventajas del uso de esta modalidad de autenticación se han explicado en el apartado 4.2.3.

Para activar este método de autenticación lo primero es habilitar el plugin en Kong Gateway.

Ilustración 45. Activación de autenticación mediante API Key

Los parámetros *config.key_in_body*, *config.key_in_header* y *config.key_in_query* permiten configurar en qué lugar de la petición HTTP se

permite el envío de la API Key. En este caso, se ha configurado para que se permita como parámetro y como cabecera, pero no en el cuerpo del mensaje. Por otro lado, el parámetro *config.key_names* permite definir una lista de nombres que identificarán a la API Key en la recepción de la petición. Por último, el parámetro *config.run_on_preflights* permite forzar la ejecución de la autenticación con API Key en peticiones de tipo *OPTIONS*, usadas en su mayoría por los navegadores como método de seguridad.

El siguiente paso consiste en generar una API Key. Es posible definir una clave arbitrariamente escogida o delegar la generación de esta clave al API Gateway. En este caso se ha optado por la segunda opción.

Ilustración 46. Generación de API Key

Como se puede ver, en la respuesta de la petición se ha devuelto un campo *key* que contiene la API Key generada para el consumidor *test-consumer*.

Por último, en las siguientes ilustraciones se puede ver una petición realizada sin API Key y otras dos enviando la API Key como parámetro y en las cabeceras HTTP, respectivamente.

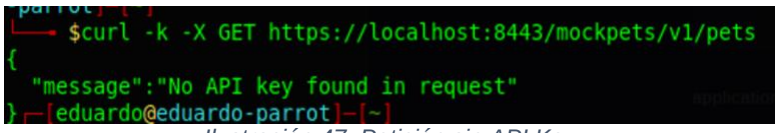


Ilustración 47. Petición sin API Key

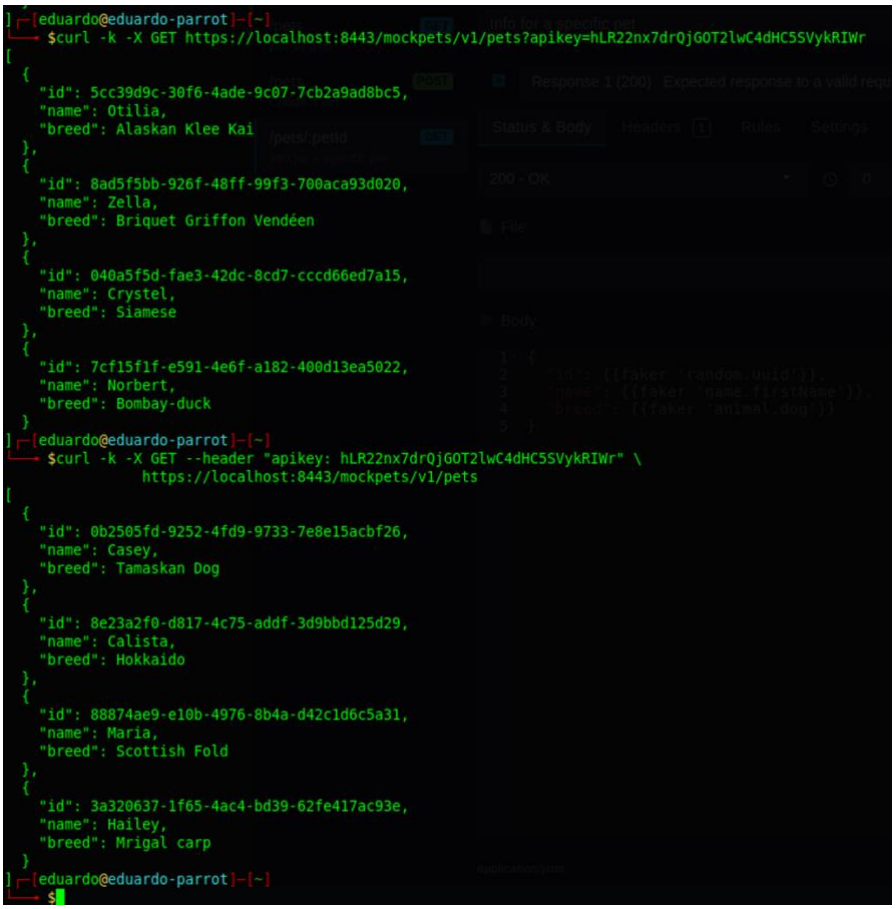


Ilustración 48. Petición con API Key

7.6.5. Activación de autenticación OAuth2.0

El estándar OAuth2.0 para la protección de las APIs REST es, sin ninguna duda, el más popular en este momento. Ofrece muchas ventajas y beneficios de seguridad que otros métodos de autenticación no ofrecen, tal y como se ha explicado en el apartado 4.2.4.

Kong Gateway proporciona un plugin de manera predeterminada que transforma el API Gateway en un servidor de autorización OAuth2.0, permitiendo expedir tokens de acceso, validarlos, revocarlos y renovarlos.

En la siguiente ilustración se puede ver la activación de dicho plugin.

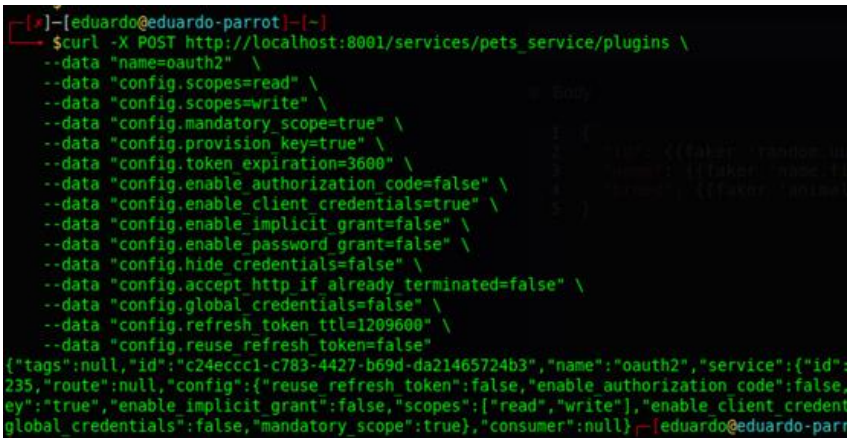


Ilustración 49. Activación de plugin OAuth2.0

Los parámetros más relevantes definidos son los siguientes:

- config.scopes: con este parámetro se definen los *scopes* o permisos válidos en el servidor de autorización. Estos *scopes* permiten autorizar las acciones que se realizan en la API REST. En este caso se ha definido un *scope read* que permitiría leer o hacer peticiones *GET* a la API y un *scope write* que permitiría escribir o hacer peticiones *POST* o *PUT* a la API.

- config.mandatory_scope: define si el envío del parámetro *scope* en la petición para la obtención de un token de acceso es obligatorio o no.

- config.token_expiration: define la validez de los tokens de acceso que se generen.

- config.enable_*: define qué *grant types* o flujos se habilitarán. En este caso, dado que las peticiones son *machine-to-machine* sólo se ha habilitado el flujo *client_credentials*.

- config.refresh_token_ttl: define la expiración o tiempo de vida del token de refresco que se expide junto con el token de acceso.

- config.reuse_refresh_token: define si el token de refresco puede ser usado más de una vez o no.

El siguiente paso consiste en la creación de un cliente OAuth2.0 asociado a un consumidor. En este caso se ha creado un cliente llamado *Pets Consumer*, y se ha delegado la generación del *client_id* y *client_secret* al API Gateway, aunque existe la posibilidad de definirlo de manera arbitraria.

Ilustración 50. Creación de cliente OAuth2.0

Con el cliente recién creado es posible hacer peticiones para la obtención de tokens de acceso. En la siguiente ilustración se puede ver un ejemplo de petición.

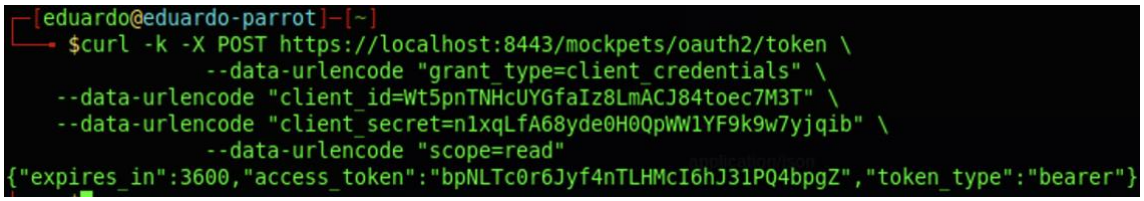


Ilustración 51. Obtención de token de acceso

En caso de que se realice alguna petición a la API REST sin enviar ningún token de acceso, el API Gateway responde con un mensaje de error indicando que el token de acceso no está presente.

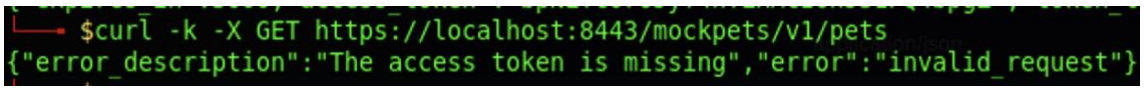


Ilustración 52. Petición sin token de acceso

Sin embargo, enviando el token de acceso en la cabecera *Authorization: Bearer* se obtiene respuesta satisfactoria.

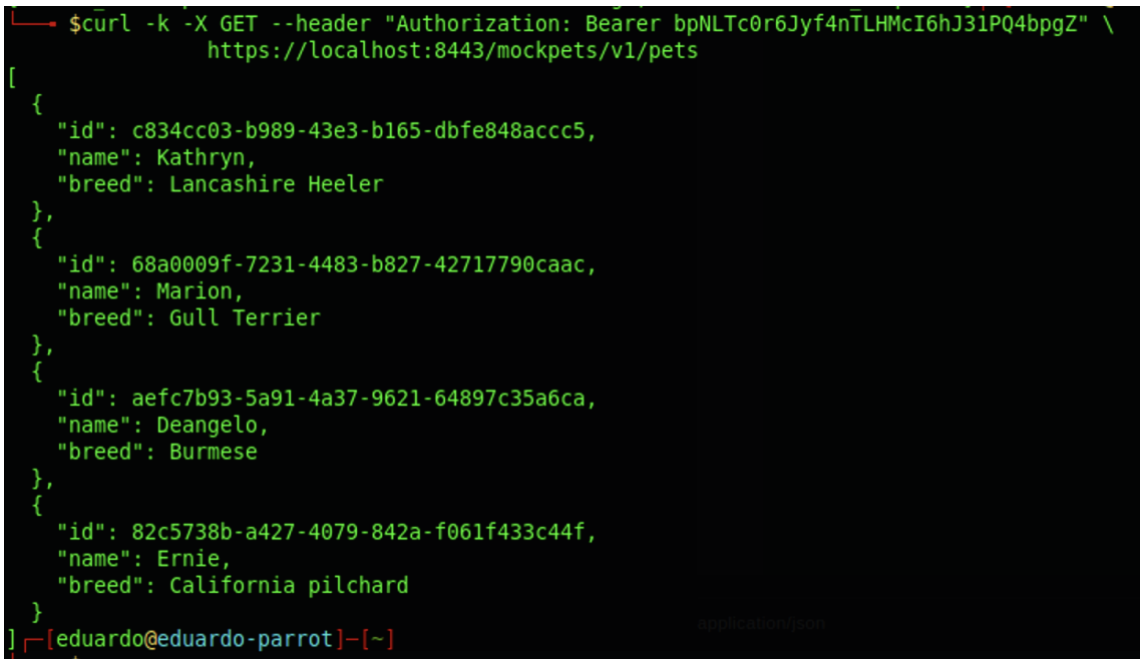


Ilustración 53. Petición con token de acceso

7.6.6. Configuración de prevención de ataques de inyección

Kong Gateway no cuenta con ningún módulo o plugin predeterminado para proteger la API REST contra ataques de inyección. Sin embargo, existen plugins de terceros que permiten realizar estas funciones de securización, como son *Signal Science* [36] o *Wallarm* [37].

Estos dos plugins permiten disponer de una protección altamente eficaz, ya que cubren el top 10 de riesgos en APIs REST de OWASP. Se trata de soluciones en modo SaaS, con lo que solamente es necesario la instalación de un agente o del propio plugin desarrollado por ellos. Se basan en el análisis activo y en tiempo real de las peticiones recibidas por el API Gateway, el cual reenvía estas

peticiones a los servidores de estas empresas para obtener una respuesta analizada.

Desafortunadamente no ha sido posible la implementación de dichos *plugins*, ya que al estar basados en un formato de software de tipo SaaS requieren una licencia de pago para su uso y no disponen de versión gratuita.

8. Conclusiones

Con el aumento del uso de APIs REST han surgido nuevos escenarios y posibilidades, pero a su vez más riesgos en cuanto a la seguridad. Proteger estas APIs REST es fundamental para garantizar el acceso seguro a datos normalmente confidenciales, la no manipulación o integridad y para garantizar que siempre estén disponibles para su acceso.

El uso de un API Manager que permita definir e implementar políticas de seguridad es fundamental hoy en día en grandes entornos corporativos, ya que facilita enormemente la gestión de estas APIs, así como la definición centralizada de todas sus políticas. Por otra parte, libera al desarrollador de la responsabilidad de tener que estar al día con nuevos protocolos de seguridad, así como de saber cómo implementarlos, aunque no por ello estas tareas se deben dejar de lado por parte del desarrollador. Por último, permite no solo incluir aspectos de seguridad, sino funcionalidades adicionales reutilizables que, de otra manera, se deberían implementar en cada uno de los microservicios que se encuentren en el *backend*.

Como todos los sistemas, es fundamental saber elegir adecuadamente el API Manager que se desea implementar, en base a la modularidad que proporcione, herramientas adicionales como *Dashboards* de gestión, extensibilidad y precio. En este trabajo se ha escogido Kong, pero no por ello resulta la mejor opción en cualquier situación y, como se ha visto, es posible que, a pesar de utilizar un API Manager libre de pago, se requiera el pago por el uso de ciertos módulos.

Los objetivos planeados en el inicio del trabajo se han cumplido satisfactoriamente, a excepción de la implementación del módulo que proporciona un *WAF* en el API Manager. El motivo fundamental ha sido el requerimiento del pago de una licencia por su uso, o el desarrollo de un módulo *ad hoc*, lo cual no estaba planeado inicialmente y requeriría de su propio estudio fuera del ámbito de este trabajo.

En cuanto al seguimiento de la planificación, esta se ha cumplido sin ningún contratiempo, disponiendo de suficiente tiempo para evaluar varias de las opciones disponibles en el mercado, así como para realizar el estudio de las amenazas más comunes hoy en día. La metodología seguida ha sido adecuada, al no ser un trabajo de desarrollo de software. En ese caso, quizá una metodología *Agile* o en cascada hubiese sido mejor.

Por último, cabe destacar las siguientes líneas de trabajo:

- **Módulo para implementar un WAF:** una línea de trabajo futuro interesante sería, o bien el desarrollo de un módulo personalizado que permita implementar métodos de protección contra el Top 10 de amenazas a las APIs REST de OWASP, o bien el pago de la licencia de uno ya desarrollado.

- **Integración con un gestor de accesos:** normalmente, en entornos con un nivel de seguridad muy alto, la gestión de accesos mediante los protocolos OAuth2.0 u OpenID Connect se delega en un sistema externo denominado sistema de gestión de accesos. En este caso, el API Manager podría actuar a modo de cliente de este gestor de accesos y, en lugar de expedir tokens de acceso, se dedicaría a realizar la validación y permitir el consumo de la API o no.

- **Implementación de métodos de autenticación adicionales:** existen múltiples métodos de autenticación hoy en día, tales como mTLS o autenticación mediante HMAC. La implementación de estos métodos podría ser una línea de trabajo interesante.

9. Glosario

- **Firewall:** Parte de un sistema o una red informáticos que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas

- **WAF:** Tipo de firewall que supervisa, filtra o bloquea el tráfico HTTP hacia y desde una aplicación web. Se diferencia de un firewall normal en que puede filtrar el contenido de aplicaciones web específicas, mientras que un firewall de red protege el tráfico entre los servidores.

- **Autenticación:** Acto o proceso de confirmar que algo (o alguien) es quien dice ser.

- **Autorización:** Proceso por el cual una red de datos o sistema autoriza al usuario identificado a acceder a determinados recursos de la red o de otros sistemas.

- **Inyección:** Método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos, sistema operativo u otro sistema que aloje datos.

- **DoS/DDoS:** Ataque de denegación de Servicio (DoS). Se trata de un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. En caso de que el ataque se realice de manera distribuida se denomina ataque de denegación de servicio distribuido (DDoS).

- **Phising:** Término informático que distingue a un conjunto de técnicas que persiguen el engaño a una víctima ganándose su confianza haciéndose pasar por una persona, empresa o servicio de confianza para manipularla y hacer que realice acciones que no debería realizar.

- **TLS:** Protocolo criptográfico que proporcionan comunicaciones seguras por una red, comúnmente Internet, mediante el cifrado de las comunicaciones entre los pares que se comunican.

- **mTLS:** es una extensión de la capa de seguridad de transporte (TLS), que requiere que tanto el servidor como el cliente verifiquen sus certificados digitales, de manera que se puedan autenticar sin repudio.

10. Bibliografía

1. UNIÓN EUROPEA. Reglamento 2016/679 General de Protección de Datos, 27 de abril de 2016, 88 páginas.
2. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 6749: The OAuth 2.0 Authorization Framework* [en línea]. Editado por D. Hart. Octubre 2012 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc6749
3. *OpenID Connect* [en línea] [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://openid.net/connect
4. *OWASP API Security Project* [en línea] [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://owasp.org/www-project-api-security/
5. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 7519: JSON Web Token (JWT)* [en línea]. Editado por M. Jones, J. Bradley y N. Sakimura. Mayo 2015 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7519
6. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 7516: JSON Web Encryption (JWE)* [en línea]. Editado por M. Jones, J. Hildebran. Mayo 2015 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7516
7. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 7515: JSON Web Signature (JWS)* [en línea]. Editado por M. Jones, J. Bradley y N. Sakimura. Mayo 2015 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7515
8. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 7517: JSON Web Key (JWK)* [en línea]. Editado por M. Jones. Mayo 2015 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7517
9. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 7518: JSON Web Algorithms (JWA)* [en línea]. Editado por M. Jones. Mayo 2015 [fecha de consulta: 28 de febrero de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7518
10. *GARTNER. Magic Quadrant for Full Life Cycle API Management* [en línea]. Editado por Shameen Pillai y Kimihiko Lijima. 28 de septiembre de 2021 [fecha de consulta: 01 de marzo de 2022]. Disponible en: https://www.gartner.com/doc/reprints?id=1-27KEXUK1&ct=211001&st=sb
11. *KONG*. [en línea] [fecha de consulta: 01 de marzo de 2022]. Disponible en: https://konghq.com
12. *TYK*. [en línea] [fecha de consulta: 01 de marzo de 2022]. Disponible en: https://tyk.io/open-source/
13. *WSO2 API Manager*. [en línea] [fecha de consulta: 01 de marzo de 2022]. Disponible en: https://wso2.com/api-manager/
14. *OWASP* [en línea] [fecha de consulta: 07 de marzo de 2022]. Disponible en: https://owasp.org
15. *GITHUB. OWASP API1:2019 Broken Object Level Authorization* [en línea] [fecha de consulta: 08 de marzo de 2022]. Disponible en:

https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa1-broken-object-level-authorization.md
16. *GITHUB. OWASP API2:2019 Broken User Authentication* [en línea] [fecha de consulta: 08 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa2-broken-user-authentication.md
17. *GITHUB. OWASP API3:2019 Excessve Data Exposure* [en línea] [fecha de consulta: 08 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa3-excessive-data-exposure.md
18. *GITHUB. OWASP API4:2019 Lack of Resources & Rate Limiting* [en línea] [fecha de consulta: 08 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa4-lack-of-resources-and-rate-limiting.md
19. *GITHUB. OWASP API5:2019 Broken Function Level Authorization* [en línea] [fecha de consulta: 08 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa5-broken-function-level-authorization.md
20. *GITHUB. OWASP API6:2019 Mass Assignment* [en línea] [fecha de consulta: 09 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa6-mass-assignment.md
21. *GITHUB. OWASP API7:2019 Security Misconfiguration* [en línea] [fecha de consulta: 09 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa7-security-misconfiguration.md
22. *GITHUB. OWASP API8:2019 Injection* [en línea] [fecha de consulta: 09 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa8-injection.md
23. *GITHUB. OWASP API9:2019 Improper Assets Management* [en línea] [fecha de consulta: 09 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa9-improper-assets-management.md
24. *GITHUB. OWASP API10:2019 Insufficient Logging & Monitoring* [en línea] [fecha de consulta: 09 de marzo de 2022]. Disponible en: https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xaa-insufficient-logging-monitoring.md
25. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 1050: Remote Procedure Call Protocol Specification (RPC)* [en línea]. Editado por Sun Microsystems, Inc. Abril 1988 [fecha de consulta: 16 de marzo de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc1050
26. *INTERNET ENGINEERING TASK FORCE (IETF). RFC 2617: HTTP Authentication: Basic and Digest Access Authentication* [en línea]. Editado por J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen y L. Stewart. Junio 1999 [fecha de consulta: 18 de marzo de 2022]. Disponible en: https://datatracker.ietf.org/doc/html/rfc2617
27. *PARALLELS. Parallels Desktop [software]. 17.1.2. 2022.*
28. *PARROT SECURITY. ParrotOS [software]. 5.0. 2022.*
29. *DOCKER, Inc. Docker [software]. 20.10.14. 2022.*

30. *DOCKER, Inc. Install Docker Engine on Debian* [en línea]. [fecha de consulta: 02 de abril de 2022]. Disponible en: https://docs.docker.com/engine/install/debian/
31. *KONG DOCS. Install Kong Gateway on Docker* [en línea]. [fecha de consulta: 05 de abril de 2022]. Disponible en: https://docs.konghq.com/gateway/latest/install-and-run/docker/
32. *MOCKOON. Mockoon [software]. 1.18.1. 2022.*
33. *CANONICAL GROUP LIMITED. Snap [software]. 2.55.3. 2022.*
34. *OPENAPI INITIATIVE. OpenAPI Specification v3.1.0* [en línea] [fecha de consulta: 15 de abril de 2022]. Disponible en: https://spec.openapis.org/oas/v3.1.0
35. *GITHUB: OpenAPI Petstore example* [en línea] [fecha de consulta: 15 de abril de 2022]. Disponible en: https://github.com/OAI/OpenAPI-Specification/blob/main/examples/v3.0/petstore.json
36. *KONG PLUGIN HUB: Signal Science* [en línea] [fecha de consulta: 20 de abril de 2022]. Disponible en: https://docs.konghq.com/hub/signal_sciences/signal-sciences/
37. *KONG PLUGIN HUB: Wallarm* [en línea] [fecha de consulta: 20 de abril de 2022]. Disponible en: https://docs.konghq.com/hub/wallarm/wallarm/