UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Framework de seguridad en entornos Edge-Cloud Computing

Trabajo Fin de Máster

Máster Universitario en Ciberseguridad y Ciberinteligencia

AUTOR/A: Reinosa Simón, Raúl

Tutor/a: Palau Salvador, Carlos Enrique

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Framework de seguridad en entornos Edge-Cloud Computing

TRABAJO FIN DE MÁSTER

Máster Universitario en Ciberseguridad y Ciberinteligencia

Autor: Raúl Reinosa Simón

Tutor: Carlos Enrique Palau Salvador

Curso 2021-2022

Dedicatoria

A mis abuelos maternos Julio y Angustias.
A mis abuelos paternos Apolonio y Enriqueta.

Gracias por todo.

Agradecimientos

Este TFM ha sido resultado no solo de mi esfuerzo, sino del trabajo en conjunto de las personas que han contribuido con su tiempo y dedicación para hacerlo posible.

En primer lugar agradecer a mi tutor Carlos Palau y a todos mis compañeros del grupo de investigación de Sistemas y Aplicaciones en Tiempo Real Distribuido (SATRD).

Agradecer a todos mis compañeros que han formado parte de mi etapa académica en el Máster Universitario de Ciberseguridad y Ciberinteligencia (MUCC).

Por último, pero no menos importante, agradecer a mi familia y mi pareja Sandra por apoyarme en todo el proceso de investigación, redacción y entrega de este TFM.

Resum

L'origen d'aquest Treball de Fi de Màster (d'ara endavant TFM) està motivat a l'estudi de seguretat perimetral i la veracitat de les dades (incertesa) sobre el paradigma emergent d'entorns *Edge-Cloud Computing* des del punt de vista conceptual com a pràctic. Quina evolució ha tingut des del seu origen, quines són les tècniques i estàndards utilitzats i sobre quin tipus de tecnologies subjacents recauen aquestes metodologies. Com a primera part del present TFM, es realitzarà una anàlisi prèvi relatiu a les principals propietats i models existents, a més dels avantatges i els inconvenients d'aquest tipus d'arquitectures, basades en l'*Edge-Cloud Computing*. Aquestes qüestions seran abordades des del punt de vista de l'eficiència, disponibilitat de les dades, seguretat de les dades i serveis dirigits per proveïdors al núvol. A continuació, després del anàlisi previ de les tecnologies més rellevants existents dins del paradigma *Edge-Cloud Computing*, s'identificaran dins del marc de la ciberseguretat les característiques relevants que cada infraestructura ha de complir i les recomendacions niades. Una vegada llistats els exercicis de seguretat que han de complir les infraestructures i serveis que proveeixen dades, es definiran solucions actuals basades en cadenes de blocs (Blockchain) que permetran portar un registre segur, descentralitzat, sincronitzat i distribuït de les dades que permetin assegurar el principi bàsic de veracitat de les dades. Posteriorment, recolzant-se en el conjunt de definicions i conceptes teòrics presentats a les fases inicials de la memòria, es realitzarà de manera pràctica un cas d'ús a través de la implementació i desplegament d'un entorn *Edge-Cloud Computing*. Aquest entorn estarà sustentat amb dades en temps real provinents de sensors prèviament instal·lats. Aquest cas comptarà amb diferents tecnologies basades en Internet Of Things (IoT) que permetran consolidar la teoria aplicada a l'operativa d'un servei eficient i lleuger. En matèria de seguretat, es crearà un framework de seguretat que permeti adaptar-se al marc legal i proporcionar un entorn fiable pel consumidor. A més, es realitzarà un model gràfic de representació de les dades al núvol que permetrà al client interactuar de manera visual, sense necessitat de tindre un coneixement previ a la tecnologia utilitzada. Per finalitzar, analitzant des del punt de vista del mercat el creixement que han tingut els serveis basats en *Edge-Cloud Computing*, s'analitzarà l'evolució del paradigma en matèria de seguretat aquests darrers anys comparant-la amb les solucions extretes d'aquest TFM. Així com altres vessants, donant lloc a noves interpretacions.

Paraules clau: IoT, ciberseguretat, edge, cloud, blockchain, veracitat de les dades

Resumen

El origen de este Trabajo de Fin de Máster (en adelante TFM) está motivado en el estudio de seguridad perimetral y la veracidad de los datos (incertidumbre) sobre el paradigma emergente de entornos *Edge-Cloud Computing* desde el punto de vista conceptual como práctico. Qué evolución ha tenido desde su origen, cuáles son las técnicas y estándares utilizados y sobre qué tipo de tecnologías subyacentes recaen estas metodologías. Como parte inicial del presente TFM, se realizará un análisis previo relativo a las principales propiedades y modelos existentes, además de las ventajas e inconvenientes de este tipo de arquitecturas, basadas en el *Edge-Cloud Computing*. Estas cuestiones serán abordadas desde el punto de vista de la eficiencia, disponibilidad de los datos, seguridad de los datos y servicios manejados por proveedores en la nube. A continuación, tras el previo análisis de las tecnologías más relevantes existentes dentro del paradigma *Edge-Cloud Computing*, se identificarán dentro del marco de ciberseguridad aquellas características relevantes que cada infraestructura debe cumplir y recomendaciones anidadas. Una vez listados los ejercicios de seguridad que deben cumplir las infraestructuras y servicios que provean datos, se definirán soluciones actuales basadas en cadenas de bloques (Blockchain) que permitirán llevar un registro seguro, descentralizado, sincronizado y distribuido de los datos que permitan asegurar el principio básico de veracidad de los datos. Posteriormente, apoyándose en el conjunto de definiciones y conceptos teóricos presentados en las fases iniciales de la memoria, se realizará de manera práctica un caso de uso a través de la implementación y despliegue de un entorno *Edge-Cloud Computing*. Este entorno estará sustentado con datos en tiempo real provenientes de sensores previamente instalados. Este caso de uso contará con diferentes tecnologías basadas en *Internet Of Things* (IoT) que permitirán afianzar la teoría aplicada a la operativa de un servicio eficiente y liviano. En materia de seguridad, se creará un framework de seguridad que permita adaptarse al marco legal y proporcionar un entorno fiable para el consumidor. Además, se realizará un modelo gráfico de representación de los datos en la nube que permitirá al cliente interactuar de manera visual, sin necesidad de obtener un conocimiento previo a la tecnología utilizada. Para finalizar, se analizará desde el punto de vista del mercado el auge que han tenido los servicios basados en *Edge-Cloud Computing*, se analizará la evolución del paradigma en materia de seguridad en estos últimos años comparándola con las soluciones extraídas de este TFM. Así como otras vertientes, dando lugar a nuevas interpretaciones.

Palabras clave: IoT, ciberseguridad, edge, cloud, blockchain, veracidad de los datos

Abstract

The origin of this Master's Thesis (hereinafter TFM) is motivated by the study of perimeter security and data veracity (uncertainty) on the emerging paradigm of *Edge-Cloud Computing* environments from a conceptual and practical point of view. How has it evolved since its origin, what are the techniques and standards used and on what kind of underlying technologies do these methodologies rely. As an initial part of this TFM, a preliminary analysis will be made of the main existing properties and models, as well as the advantages and disadvantages of this type of architectures, based on *Edge-Cloud Computing*. These issues will be addressed from the point of view of efficiency, data availability, data security and services managed by cloud providers. Then, after a previous analysis of the most relevant existing technologies within the *Edge-Cloud Computing* paradigm, the most relevant characteristics that each infrastructure has and its recommendations will be identified from a Cybersecurity point of view. Once the security exercises that the infrastructures and services that provide data must comply with have been listed, current solutions based on blockchains (Blockchain) will be defined that will make it possible to keep a secure, decentralised, synchronised and distributed record of the data that will ensure the basic principle of data veracity. Subsequently, based on the set of definitions and theoretical concepts presented in the initial phases of the report, a practical use case will be carried out through the implementation and deployment of an *Edge-Cloud Computing* environment. This environment will be supported with real-time data from previously installed sensors. This use case will have different technologies based on Internet Of Things (IoT) that will allow to strengthen the theory applied to the operation of an efficient and light service. In terms of security, a security framework will be created to adapt to the legal framework and provide a reliable environment for the consumer. In addition, a graphic model of data representation in the cloud will be created that will allow the customer to interact visually, without the need for prior knowledge of the technology used. Finally, the rise of services based on *Edge-Cloud Computing* will be analysed from the point of view of the market, and the evolution of the security paradigm in recent years will be analysed, comparing it with the solutions extracted from this TFM. As well as other aspects, giving rise to new interpretations.

Key words: IoT, cibersecurity, edge, cloud, blockchain, data veracity

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Este trabajo Fin de Máster (TFM) se encuentra dentro de la tecnología específica del *Edge-cloud Computing*. Se integran conceptos como el uso de plataformas de *Internet of Things* (en adelante *IoT* por sus siglas en inglés), sistemas distribuidos, *blockchain* y demás conceptos que son explicados y analizados a lo largo de la ejecución de este TFM.

En las secciones de este capítulo se detalla la motivación generada en el entorno informático que ha llevado a cabo la realización de este TFM, se definen los objetivos y subobjetivos del TFM, se detalla la metodología utilizada, la estructura de la memoria y el impacto esperado.

1.1 Motivación

Actualmente se está imponiendo el uso del paradigma emergente de entornos *Edge-Cloud Computing* [1]. El edge computing es un tipo de tecnología que se desarrolla en la ubicación física donde se generan los datos o cerca de ellos. Esto permite que los usuarios puedan obtener servicios más confiables y rápidos. La clave de su éxito es que las empresas pueden aprovechar esta tecnología para distribuir un conjunto común de recursos en gran cantidad de ubicaciones a partir de la flexibilidad que le aporta el cloud computing híbrido [2]. Esta tecnología incorpora cierto grado de organización, gestión y portabilidad entre dos o más entornos. Estos entornos en su mayoría se componen de un entorno virtual o servidores dedicados (*bare metal*) conectados al menos a una nube, ya sea pública o privada. Se determina ***Edge-Cloud computing*** a la estrategia de permite conseguir llevar un entorno uniforme hasta las ubicaciones físicas cercanas a los datos y distribuirlos a través de infraestructuras de nubes públicas o privadas.

En la actualidad, las tecnologías basadas en *Edge-Cloud Computing* se utilizan en varios sectores, como el de las telecomunicaciones, el transporte, la producción y los servicios públicos, entre otros.

Debido al éxito de la implementación de estos modelos en las tecnologías actuales, es necesario adaptar y abordar el paradigma de ciberseguridad que englobará varios conceptos y partes del proceso de procesamiento, envío y recepción de datos.

La motivación principal de este TFM es estudiar y aplicar técnicas de seguridad perimetral y veracidad de los datos (incertidumbre) sobre el paradigma emergente de entornos *Edge-Cloud Computing*, con el fin de implementar una solución sólida para garantizar todo lo posible el buen uso y distribución de estos entornos.

1.2 Objetivos

Los objetivos principales de este TFM son:

1. Conocer y exponer las características principales y el éxito que acompaña a las tecnologías basadas en el paradigma *Edge-Cloud Computing*.

2. Estudio y análisis de seguridad perimetral y veracidad de los datos (incertidumbre) sobre el paradigma emergente de entornos *Edge-Cloud Computing*.

3. Puesta en marcha de un caso práctico *Edge-Cloud Computing* basado en datos en tiempo real.

4. Análisis y conclusión acerca de las posibles nuevas implementaciones de seguridad sobre tecnologías subyacentes que utilizan dispositivos *IoT* (en adelante *IoT* por sus siglas en inglés).

Para el estudio y desarrollo de los objetivos principales, se han definido los siguientes subobjetivos:

- Análisis y estudio de los antecedentes y estado actual del paradigma emergente de entornos *Edge-Cloud Computing*.

- Implementación de *software* y despliegue de la solución propuesta

- Estudio del impacto en las prestaciones de estas tecnologías, a través de la ejecución de un caso práctico sobre un entorno real.

- Concluir y proponer nuevas vías de desarrollo y aplicación sobre estas tecnologías.

Por último, con este trabajo se trata de colaborar a la literatura y estudios actuales, tratando de analizar el paradigma actual del *Edge-Cloud Computing* para que cualquier potencial usuario tenga información que le pueda resultar de interés a la hora de tomar decisiones a hora de adaptar su entorno en el ámbito de diseño y seguridad, a través de sus propios requisitos.

1.3 Estructura de la memoria

Este trabajo está compuesto por seis capítulos. A continuación, se presenta el contenido de cada uno de ellos:

- **Primer Capítulo - Introducción.** Se expone el problema presentado para la realización del TFM y se explican los objetivos a alcanzar.
- **Segundo Capítulo - Antecedentes y estado del arte.** Se presentan los antecedentes relacionados con la temática a tratar, se describen las tecnologías utilizadas y se expone la literatura relacionada con el desarrollo del trabajo.
- **Tercer Capítulo - Metodología y desarrollo.** Se detalla la metodología utilizada durante el desarrollo del TFM.
- **Cuarto Capítulo - Entorno de pruebas y aplicaciones.** Se detalla el entorno *hardware*, *software* y las aplicaciones utilizadas para el desarrollo del TFM.
- **Quinto Capítulo - Implementación y despliegue de la solución propuesta.** Se presentan el caso práctico realizado, se detalla y se explican los resultados obtenidos.
- **Sexto Capítulo - Conclusiones y propuestas.** Se presentan las conclusiones obtenidas durante el desarrollo del TFM y los trabajos futuros.

1.4 Impacto esperado

A través de la idea presentada, se espera obtener unas conclusiones y resultados favorables que ayuden a mejorar la seguridad implementada en el paradigma de los entornos *Edge-Cloud Computing*. Este trabajo no está enfocado en la presentación de una solución innovadora que sea de inmediata aplicación en estos entornos, sino más bien detectar los problemas actuales de seguridad y plantear una solución coherente que ayude en el desarrollo de futuros proyectos de aplicación.

El impacto esperado de este TFM es poder servir de base en futuros proyectos relacionados con el paradigma de entornos *Edge-Cloud Computing* y la seguridad asociada. Con esto en mente, se habilita a este proyecto ser referenciado y utilizado en proyectos internos de la Universitat Politècnica de València, además de proyectos externos a esta si se diera el caso.

Parte de la problemática inicial y recursos utilizados provienen de proyectos asociados al departamento de Sistemas y Aplicaciones en Tiempo Real Distribuido (SATRD) [3] de la Universitat Politècnica de València. Con la realización y conclusiones de este proyecto se espera poder ayudar al desarrollo futuro de proyectos europeos como **ASSIST-IoT**, **DataPorts** o **aerOS**, entre otros.

Por último, gracias a la realización de este TFM, se espera poder anidar los conceptos de *blockchain* junto a las tecnologías *Edge-Cloud Computing* a través de la primera integración y despliegue viable en este campo. Esta solución permite generar impacto en el mercado de plataformas *IoT* generando seguridad y fiabilidad en su uso distribuido.

CAPÍTULO 2
Antecedentes y estado del arte

En las secciones de este capítulo se detallan los antecedentes relacionados a entornos *Edge Computing* junto al *Cloud híbrido*, se enumeran y analizan las diferentes técnicas y estándares utilizados, se describen las tecnologías subyacentes utilizadas y se detalla el estado del arte en la literatura actual. Por último se dicta una propuesta que servirá como referencia en siguientes secciones de este TFM.

2.1 Antecedentes

Para poder conocer en detalle la finalidad del Trabajo Fin de Máster (TFM) es necesario poner en contexto el proyecto, analizando el escenario actual donde se desarrolla, los detalles técnicos e introduciendo los principios de la tecnologías *Egde Computing* y *Cloud Computing*, junto a su combinación, dando parte al *Edge-Cloud Computing*. Por todo ello, es necesario explicar cómo y dónde surgió la idea y cómo han ido evolucionando desde su creación hasta la puesta en marcha en entornos empresariales.

2.1.1. Contexto

Desde la aparición de Internet en las últimas décadas del *S.XX* hasta hoy día, Internet ha cambiado el paradigma en el mundo de la computación de manera drástica. Los diferentes métodos, técnicas y procesos para procesar, almacenar y transmitir información son cada vez utilizados y es necesario adecuar el método utilizado a las necesidades, conocimientos y presupuesto de cada uso. Los métodos han ido variando desde la computación paralela, distribuida, *grid* (sin un nodo centralizado), etc. Durante el transcurso de este TFM nos centraremos en las técnicas de computación basadas en *cloud-computing* y *edge-computing*, que serán detalladas a continuación [4].

Centrándonos en los primeros modelos de computación utilizados encontramos el uso de servidores físicos [5]. Estos aparatos no son más que máquinas físicas integradas una red informática. Los servidores brindan servicios e información a los clientes dentro de un ámbito de producción. El principal problema que poseen es que la seguridad de los datos y la funcionalidad del servicio son dependientes de la propia empresa, añadiendo los elevados costos de mantener servidores físicos, refrigeración, flujo de aire continuo, etc. A través del paso del tiempo, estos entornos físicos están siendo migrados a entornos virtualizados e instalados en la nube, comúnmente conocidos como *Cloud Computing*.

La tecnología *Cloud Computing* posee diversas funcionalidades. Entre ellas, se encuentra la capacidad de poder albergar diferentes bases de datos y módulos de cómputo e interconexión. Estas características la hacen óptima para el paradigma *IoT*. A continua-

ción, se explican los diferentes modelos de procesado de datos y computación que se encuentran actualmente en el mercado, junto a su evolución y los beneficios y carencias que ofrece cada tecnología.

2.1.2. Cloud Computing

La tecnología *Cloud Computing* ofrece mayor seguridad, debido a que la información y servicios son controladas directamente por los proveedores del servicio. Además, es posible contratar sólo los servicios necesarios para el fin requerido, dando paso a un sistema más adaptado a cada entorno de producción.

Los sistemas de *Cloud Computing* se podrían haber convertido en la tecnología *estándar de facto* en entornos de computación complejos. El principal problema de está tecnología son sus limitaciones de ancho de banda para su funcionamiento. Desafortunadamente, no todos los clientes cuentan con gran ancho de banda para poder enviar y procesar grandes cantidades de datos. En el contexto de utilización de tecnologías basadas en *Internet Of Things* (IoT), se envían grandes volúmenes de información en pequeñas partes que deben ser procesadas adecuadamente para un correcto funcionamiento del sistema.

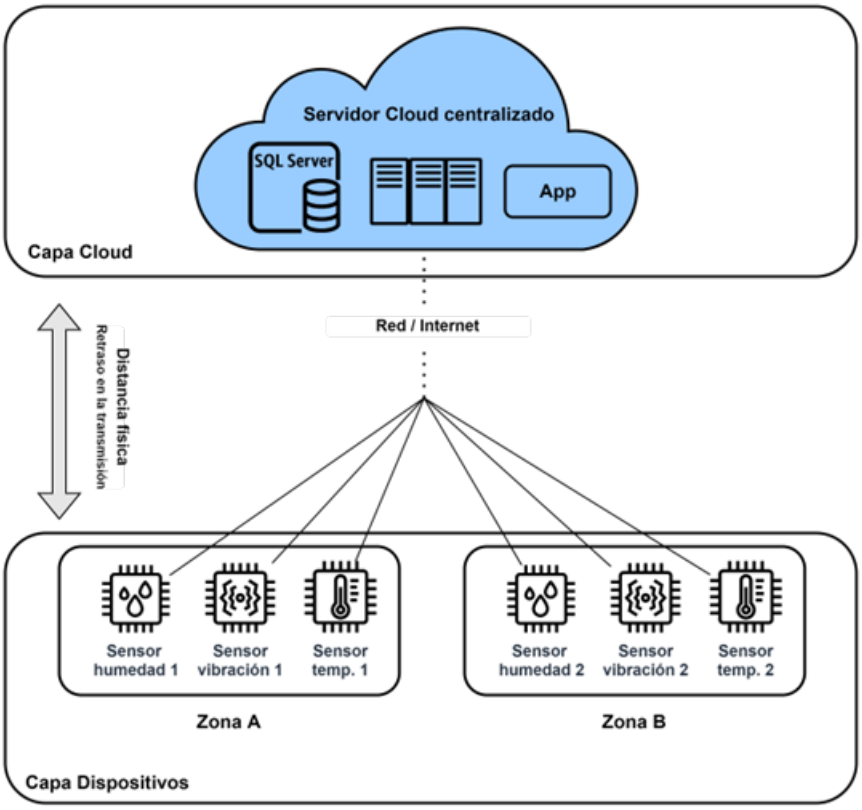

Figura 2.1: Arquitectura *Cloud Computing*

Según muestra la Figura 2.1, al existir demasiada distancia física entre los dispositivos *IoT* y el nodo de procesamiento, se genera un drástico retraso en la transmisión de los datos, lo que provoca una mala experiencia al usuario. Si combinamos los inconvenientes de congestión y retraso en la transmisión, creamos la necesidad de obtener una tecnología subyacente que resuelva estas carencias [6].

2.1.3. Edge Computing

Es aquí donde aparece la computación de borde (*Edge Computing*) como una solución a los problemas de congestión de las comunicaciones entre los clientes y los servicios en la nube. En esta arquitectura, se aplica una capa intermediaria entre las fuentes datos y las aplicaciones. Esta capa intermediaria estará lo más cercano a los usuarios o los datos de las aplicaciones, con el fin de poder agrupar el volumen de datos producido y mejorar la latencia de envío, consiguiendo así una mejora en la experiencia del usuario.

En la Figura 2.2 se muestra de manera gráfica como la distancia física entre los datos y el nodo de procesamiento es relativamente menor. Gracias a esta mejora, conseguimos obtener datos en tiempo real y sin retraso en las transmisiones.

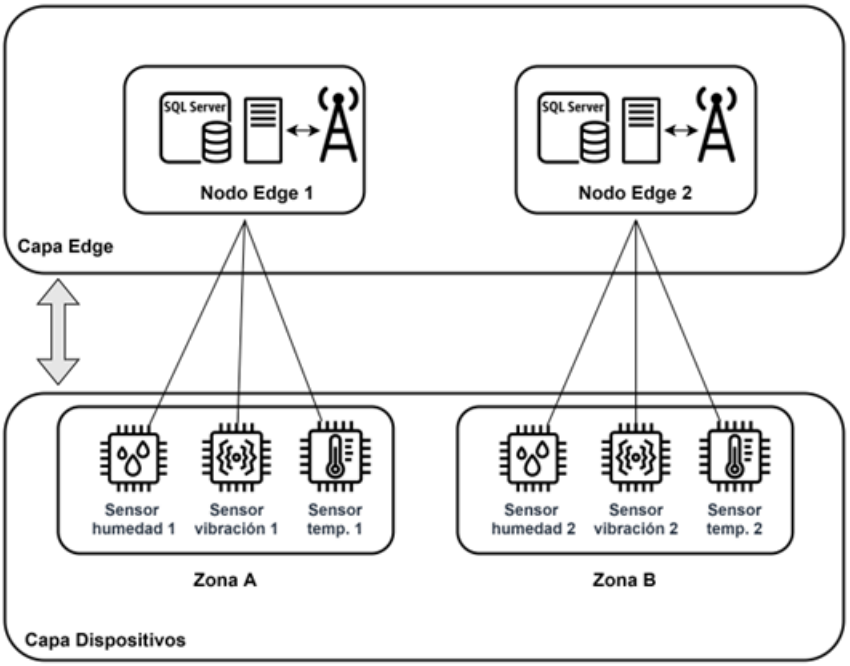

Figura 2.2: Arquitectura *Edge Computing*

Edge Computing no solo es considerada una tecnología dentro del ámbito de la computación, sino también es considerada un concepto arquitectónico. Esta tecnología genera un gran impacto en el medio ambiente, por diversas razones: en primer lugar, al tratar los datos en un entorno controlado antes de ser enviados por Internet, se reduce el número de peticiones que han de procesarse individualmente, generado un menor gasto computacional y consumiendo así menos recursos. En segundo lugar, el hecho de enviar datos procesados a los centros de datos en la nube, evitará tener que adaptar cada tecnología cloud dependiendo de los dispositivos utilizados, por lo que sufrirán menos transformaciones y podrá determinarse un sistema más homogéneo y adaptado a la necesidad de los recursos necesitados, sin malgastarlos [7].

2.1.4. Edge-Cloud Computing

Combinando las tecnologías de *Cloud Computing* y *Edge Computing* se consigue procesar grandes volúmenes de datos mientras se conserva una buena experiencia de usuario. A esta tecnología se le denomina como *Edge-Cloud Computing*.

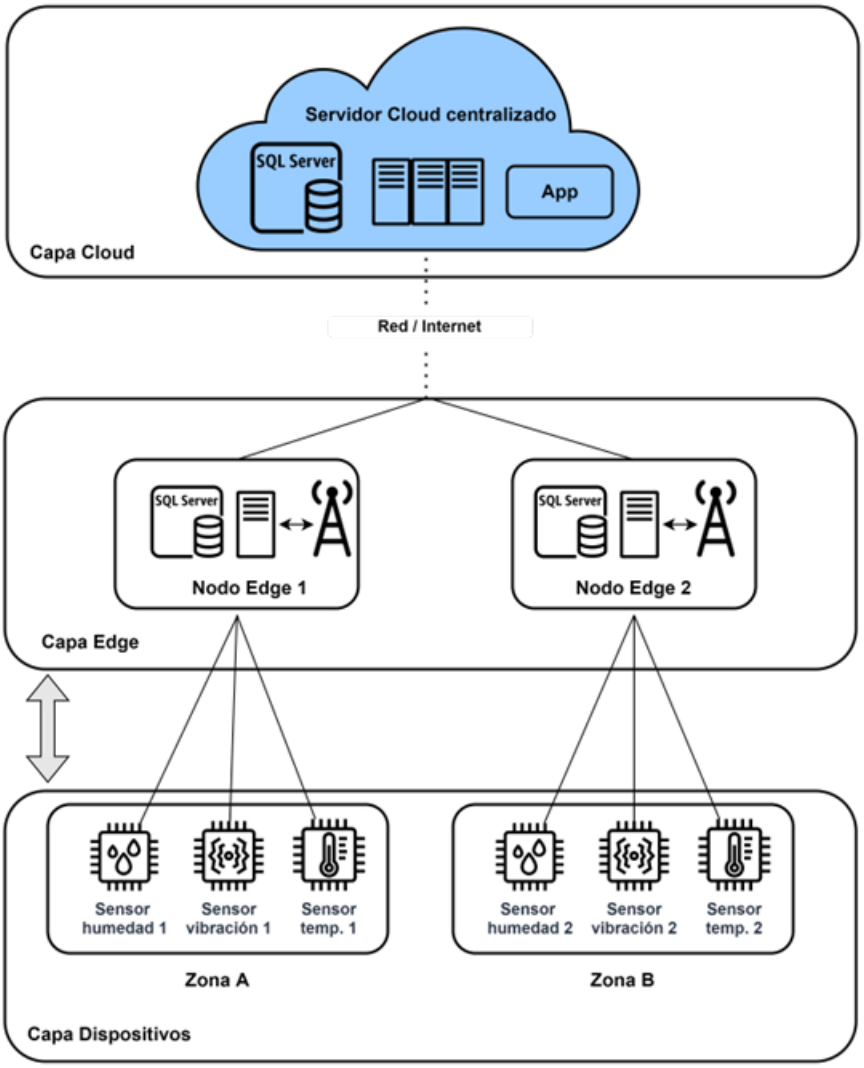

Figura 2.3: Arquitectura *Edge-Cloud Computing*

En la Figura 2.3 se representa la arquitectura final de la tecnología *Edge-Cloud Computing*. Gracias a esta tecnología se reduce la latencia y el número de peticiones a procesar por parte de la *Capa Cloud*, además de conservar las ventajas nombradas anteriormente que ofrecían los sistemas de computación basados en la nube, entre las que se encuentran la eficiencia de uso, disponibilidad de los datos y mejor adaptación según los requisitos necesarios a cada sistema.

Una vez que se soluciona la necesidad de un procesamiento ágil, se crea una necesidad adicional de seguridad. Al crear una nueva capa intermediaria entre los datos y el Cloud, será necesario habilitar un *framework* de seguridad que nos permita asegurar que los datos enviados desde los nodos de información coinciden con los que recibe el cliente final. Para que este paradigma sea seguro y funcional, debemos realizar un énfasis en la privacidad y la confianza de los datos enviados y recibidos. Por este motivo, mecanismos

de autenticación y verificación de las fuentes de datos son indispensables para asegurar la viabilidad de esta implementación.

2.1.5. Conclusiones

En la Tabla 2.1 se resumen las diferentes capas que forman la arquitectura *Edge-Cloud Computing*.

Capa	Descripción
Dispositivos	Los dispositivos *IoT* son los encargados de adquirir las fuentes de datos desde un entorno físico y enviarlos a su correspondiente nodo edge.
Edge	Se encuentran los nodos edge. Son directamente responsable del procesamiento de datos, enrutamiento y operaciones computacionales sencillas.
Cloud	Se encuentras las bases de datos y servidores. Se ocupa de el análisis de datos, las tareas de procesamiento e instrucción de los datos o la visualización y presentación de estos, además de otras diversas funciones.

Tabla 2.1: Capas de Edge-Cloud Computing

Como se ha mencionado en subsecciones anteriores, la tecnología *Edge-Cloud Computing* consigue reducir la latencia y la congestión en las redes de procesamiento, en este caso, la nube. Además, gracias a la incorporación de la tecnología *Cloud Computing* conseguimos un escenario centralizado que nos permite adaptar nuestro entorno a las necesidades arquitectónicas requeridas, optimizando así el uso de recursos y el coste total de producción.

Una de las principales desventajas que nos muestra este concepto de arquitectura es la aplicación de una nueva capa de gestión. Esta capa deberá cumplir con diversos estándares de ciberseguridad que puedan asegurar a los proveedores de servicio Cloud que los datos coleccionados por los diversos dispositivos *IoT* coinciden con los enviados desde el nodo edge correspondiente. Además, se deberá integrar mecanismos de seguridad que eviten que se pierda la disponibilidad del servicio.

Como conclusión a este apartado, disponemos de una tecnología enfocada a entornos relacionados con *IoT*, conexión vehicular, aplicaciones interactivas, etc. Con grandes capacidades de uso y despliegue. Al añadir un *framework* de seguridad que permita asegurar el buen funcionamiento y transporte de datos, está tecnología será una gran solución para procesar y distribuir la gran cantidad de operaciones computacionales que se generan hoy en día.

En las siguientes secciones de este capítulo hablaremos de diferentes entornos de aplicación de esta tecnología.

2.2 Descripción de las tecnologías

Este trabajo está destinado a analizar y representar desde el punto de vista de la seguridad las tecnologías basadas en el paradigma *Edge-Cloud Computing*. Por este motivo, se ha decidido trabajar y estudiar el entorno desde el uso de la plataforma *FIWARE*, debido a que es una de las plataformas más populares en entornos de producción de *IoT*. Ade-

más, como *framework* de seguridad se ha optado por *Blockchain*, que facilita el proceso de registro de transacciones y de seguimiento de datos.

Seguidamente, se describirán las principales características de estas tecnologías.

2.2.1. Plataforma FIWARE

FIWARE [8] es una plataforma *open-source*[1] íntegramente europea. Se utiliza para agilizar el desarrollo de soluciones inteligentes. Es un *software* preparado para el mundo empresarial. Posee elementos que intercalan la conexión entre el *IoT* con servicios de *Big Data* y gestión de la información.

Uno de los elementos esenciales en la arquitectura FIWARE es la propia gestión del contexto. Para poder desarrollar soluciones inteligentes será necesario obtener información de lo que ocurre en cada momento, como por ejemplo un ciudad. *FIWARE* proporciona un mecanismo que permite generar, recopilar, publicar y consumir información relacionada con el contexto de forma masiva, desde varias fuentes de datos y hacer un uso de ellas desde las aplicaciones, haciendo posible interactuar con lo que sucede.

Entre las distintas funcionalidades que ofrece esta plataforma destacamos las siguientes:

- **Uso inteligente de datos**: Proporciona una API[2] estándar para la gestión y uso de los datos, además de armonización de datos recopilados de diferentes fuentes de información y distintos formatos para interconectarlos de manera racional.

- **Soluciones y servicios inteligentes**: Los procesos son completamente automatizados, ofrece una fácil integración *Plug&Play* y es compatible con distintas tecnologías basadas en modelos *IoT*.

- **Adaptación a los diferentes mercados**: Esta tecnología está pensada desde sus comienzos para interactuar en diferentes entornos de aplicación, entre los que se encuentran las *Smart Cities*, agricultura, industria, energía, etc. Por lo que está preparada para recibir y procesar fuentes de datos desde distintos ámbitos.

En lo que respecta al desarrollo de este TFM, la plataforma FIWARE provee suficiente documentación y pruebas de concepto (*PoC* por sus siglas en inglés) para la creación y despliegue de un entorno de pruebas basado en el paradigma *Edge-Cloud Computing*. Esta plataforma será la encargada de operar entre los nodos *edge* y la plataforma *cloud*, consiguiendo así una capa de abstracción y seguridad de los datos, además de una gestión eficiente y granularidad de acceso a los datos.

En la Figura 2.4 se muestra la arquitectura principal de un entorno basado en la plataforma *FIWARE*. A continuación, se listan los principales componentes relacionándolos con las funcionalidades que presentan en los entornos *Edge-Cloud Computing* y en el enfoque cercano a este TFM.

[1] El software *open-source* es un código diseñado donde todos la comunidad pueden ver, modificar y distribuir el código de la forma que consideren conveniente.

[2] Interfaz de Programación de aplicaciones.

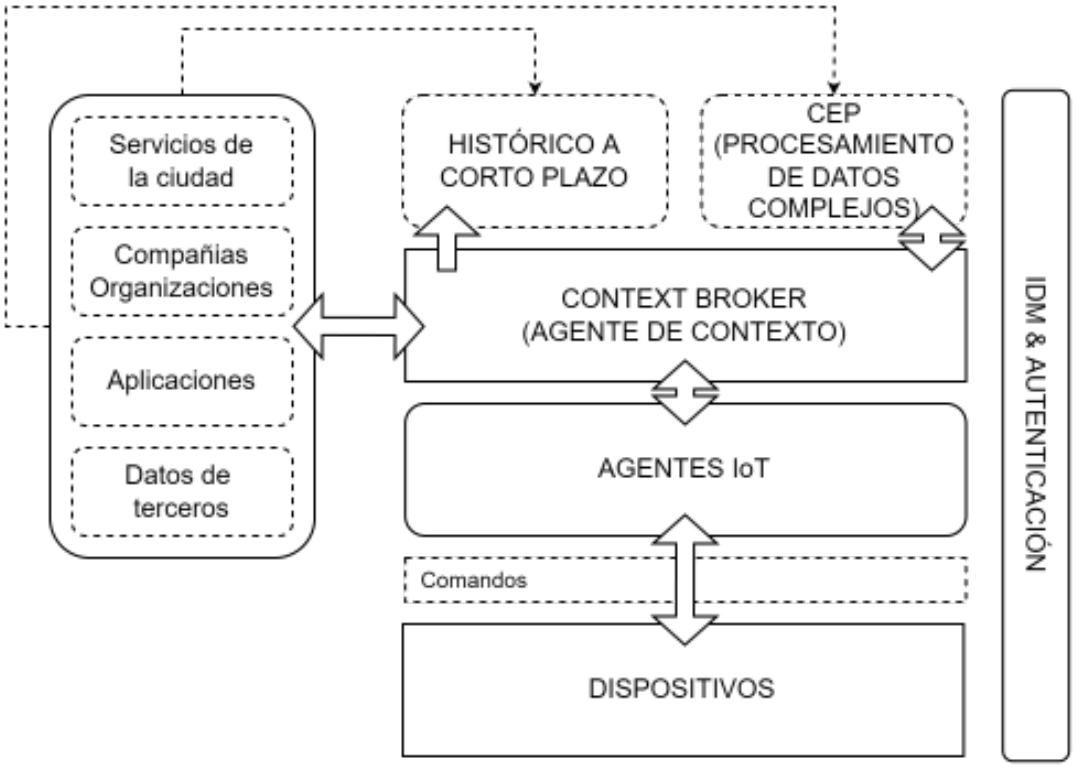

Figura 2.4: Arquitectura plataforma *FIWARE*

- **Dispositivos**: Dispositivos *IoT*. Son los responsables de capturar y enviar los datos con su correspondiente protocolo. Entre los más utilizados se encuentran: CoaP, JSON o UltraLight sobre *HTTP/MQTT*, UPC-UA, Sigfox o LoraWAN [9].
- **Agentes IoT**: Captura los datos enviados por los dispositivos *IoT* y los procesa. Una vez posee todos los datos debidamente procesados los envía a la *API* del *Context Broker*.
- **Context Broker**: Actúa como *API FIWARE NGSI v2*, que es una *API Restful* simple que permite realizar actualización, consultas o suscripciones a los cambios de información que ocurren dentro del contexto.
- **Base de datos**: Usualmente se utiliza la base de datos *NoSQL* MongoDB. Almacena y actualiza los datos recibidos por el *Context Broker*.

Además esta tecnología cuenta con diferentes funcionalidades y tecnologías subyacentes que permiten agregar autenticación segura y privada basada en *OAuth2*. Estas tecnologías son *Keyrock Identity Manager* [10] y *Wilma* [11]. Esta última ofrece soporte para funciones proxy dentro de los esquemas de autenticación basados en *OAuth2*.

Como hemos nombrado en la sección anterior, el acto de implementar una capa extra de gestión como es la tecnología derivada de la plataforma *FIWARE*, conlleva una falla de seguridad de los datos. Esta falla se conoce como veracidad de los datos (o incertidumbre de los datos). Es necesario comprobar que los datos enviados por los sensores (capa de dispositivos) coincide con los datos recibidos en la capa de procesamiento y análisis (capa *cloud*).

Por este motivo, en la implementación de la capa *edge* será necesario implementar un mecanismo de seguridad adicional, que nos asegure que la información no ha sido modificada ni manipulada en el proceso de gestión y percepción de estos. Se ha decidido implementar la tecnología *Blockchain*, que se explicará en la siguiente subsección.

2.2.2. Blockchain

En el mercado actual, donde el volumen de información que se transmite y recibe es enorme, será necesario que el intercambio de información entre productores y consumidores sea transparente y seguro para ambas partes.

Desde el punto de vista tecnológico, el *Blockchain* es una base de datos distribuida que tiene como objetivo el registro de bloques de información y entrelazarlos para poder facilitar la verificación y recuperación de información que no ha sido modificada [12].

Explicado de manera sencilla, la tecnología *Blockchain* se puede definir como una base de datos distribuida donde sólo se puede añadir contenido, pero no puede ser modificado o eliminado. Las principales que poseen son las siguientes:

- **Inmutabilidad**: Los datos añadidos a la cadena *blockchain* no pueden ser eliminados o modificados.
- **Transparencia**: Cualquier usuario con acceso puede observar el contenido de este en cualquier momento. Se podrá verificar el estado de las transacciones y que tipo de operaciones se han utilizado.
- **Eficiencia**: Al ser un modelo descentralizado, permite llevar a cabo operaciones de manera eficiente y sin necesidad de agregar un agente intermediario, lo que conlleva ser más económico.
- **Trazabilidad**: Es posible observar las transacciones en cualquier momento del proceso. Gracias a esta propiedad se evitan posibles usos fraudulentos de los datos.
- **Seguridad**: A través de diferentes técnicas criptográficas se asegura la veracidad de los datos y su origen.

Gracias a las características nombradas anteriormente, esta tecnología opta a ser una gran opción para implementar seguridad de la información en nuestra plataforma *FIWARE* y con ello, poder asegurar la veracidad de los datos a la hora de implementar una plataforma intermediaria entre el proveedor de los datos y el cliente.

A continuación y para detallar el funcionamiento interno de esta tecnología, se explicará de manera resumida, el proceso de almacenamiento y procesamiento de bloques que utilizan las tecnologías basadas en *Blockchain* [13].

1. Un nodo de la cadena genera una nueva transacción en la cadena de bloques (*Blockchain*) y este lo firma con su clave privada.

2. Esta transacción es propagada por otros nodos, que validan la transacción dependiendo de un criterio determinado. Normalmente, se requieren de varios nodos para confirmar las transacciones.

3. Cuando la transacción es validada por todos los nodos participantes, esta transacción se incluye en un bloque y es propagada por la red. Una vez esto ocurre se considera que la transacción ha podido ser confirmada con éxito.

4. El bloque que ha sido generado a partir de un conjunto de transacciones validadas se convierte en un bloque adyacente a la cadena de bloques que pertenece a la *blockchain*. En este momento, la transacción recibe su segunda confirmación y el bloque la primera.

5. Existen diversas tecnologías basadas en confirmaciones de las cadenas de bloques. Por ejemplo, la tecnología *Bitcoin* considera que una transacción ha llegado a su confirmación final cuando se ha confirmado 6 veces.

Este proceso de funcionamiento del *Blockchain* puede ser aplicado a diferentes ámbitos. En nuestro caso, requeriremos que los datos que son enviados por los nodos *edge*, encargados de recibir y procesar los datos recibidos por los sensores, implementen esta tecnología en cada una de sus actualizaciones. La Figura 2.5 muestra el proceso comentado anteriormente adecuado a nuestro entorno de trabajo.

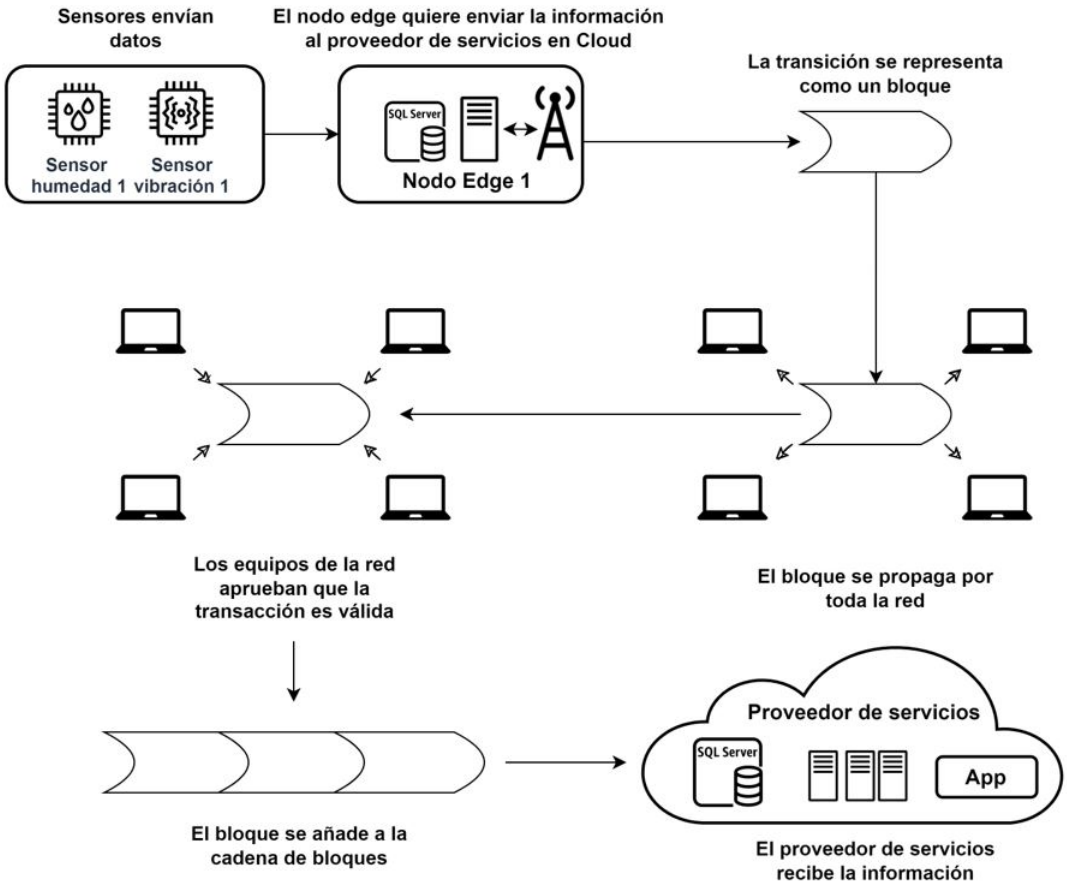

Figura 2.5: Ejemplo de funcionamiento de transacción en *Blockchain*

2.3 Estado del arte

Para la realización de este TFM se ha revisado la literatura y los documentos más actualizados respecto al uso de las tecnologías *Edge-Cloud Computing*, sus principales diferencias sobre otras tecnologías semejantes en el mercado y, para finalizar, su relevancia en entornos empresariales y mercantiles. A continuación, en esta sección se presentas los trabajos de investigación y publicaciones realizadas recientemente en el contexto de este TFM. Este estado del arte nos permitirá resaltar las carencias que existen en el ámbito de la seguridad y justificar claramente la necesidad de realizar este trabajo de implementación de un *framework* de seguridad en entornos *Edge-Cloud Computing*.

Para comenzar, en [15] se presenta como durante mucho tiempo los diferentes tipos de computación eran ejecutados y distribuidos completamente en servidores físicos. Con el paso del tiempo, las tecnologías *Cloud Computing* y *Edge Computing* se popularización y ganaron mucha atención ya que su éxito radicaba principalmente en la virtualización de los servicios y la optimización de uso de recursos.

En [16] se describe el uso de tecnologías *Cloud Computing* en entornos empresariales son populares y muy útiles para muchos escenarios de uso. El problema reside en que existen escenarios donde el envío de datos es masivo, por lo que la red experimenta un aumento de latencia, lo que causa una peor experiencia para el usuario. La importancia de mantener un uso óptimo en todo el proceso de procesamiento y entrega de datos deriva en la implementación de la tecnología *Edge Computing*, que procesa los datos inmediatamente más cercano a los dispositivos de *Internet Of Things*, que son responsables de la recolección de información.

Otro punto a mencionar en la implementación de estas tecnologías es la seguridad de los datos en el proceso de transmisión de los datos hasta el cliente final. Este problema se agrava cuando utilizamos la tecnología *Edge-Cloud Computing*. Esto es debido a que añadimos un componente más a la infraestructura general, por lo que los datos ya no son recibidos directamente por el proveedor de servicios, sino que son procesados en estructuras más cercanas a los datos. Esta tecnología no sería viable desde el punto de vista de veracidad de los datos si no se implementa un mecanismo de seguridad que pueda asegurar que los datos obtenidos y enviados por los dispositivos IoT son los recibidos por el cliente final. Por este motivo será necesario la creación de un *framework* de seguridad que solucione esta problemática.

Los autores de [17] y [18] coinciden en que la tecnología *blockchain* sería viable para solucionar la problemática existente. Esto es posible gracias a la red de bloques descentralizada que ofrece, pero presenta problemas y limitaciones a la hora de realizar la integración. Por este motivo, la tecnología *blockchain* en la actualidad está enfocada a otros ámbitos y se encuentra en sus principios de desarrollo para este tipo de tecnologías.

Según [19], en la actualidad existe poca literatura sobre la integración de la solución basada en la tecnología *blockchain* en entornos *Edge-Cloud Computing* y aplicaciones relacionadas con un ámbito real de producción. Debido a la necesidad de implementación de un *framework* [3] de seguridad en estos entornos, existen soluciones emergentes creados por *PYMES* que generan entornos *Edge-Cloud Computing* junto a *blockchain* en entornos y casos de uso específicos.

[3] Un framework marco de trabajo que posee una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software.

2.4 Crítica al estado del arte y propuesta

Tal y como se ha descrito anteriormente, existen soluciones para entornos y casos de uso específicos. Analizando las soluciones *open-source* del mercado, destaca la utilización de la plataforma *FIWARE* como entorno *Edge-Cloud Computing*.

Una de las ventajas que nos proporciona la plataforma *FIWARE* es la integración de mecanismos de seguridad basados en autenticación y autorización, además de manejar diversas fuentes de datos entrantes de manera gestionada y organizada.

Como propuesta de creación de un *framework* de seguridad que cubra la necesidad de veracidad de los datos en entornos *Edge-Cloud Computing* se ha optado por utilizar la solución tecnología denominada *CanisMajor*. Es un adaptador de cadena de bloques (*blockchain*) que admite la persistencia y la verificación de transacciones de entidades en cadenas de bloques. Este *framework* está enfocado en la utilización bajo la plataforma *FIWARE*.

Una vez habiendo obtenido y estudiado las tecnologías base para la realización de un caso práctico, se realizará un caso de uso enfocado en la implementación de un *framework* de seguridad en entornos *Edge-Cloud Computing*.

Por último, una vez desarrollado e implementado el caso de uso, se obtendrán conclusiones de viabilidad y el posible futuro que puede tener estas tecnologías en materia de ciberseguridad.

CAPÍTULO 3
Metodología y desarrollo

Este capítulo está dedicado a describir la metodología que se ha realizado para el desarrollo ágil del TFM.

3.1 Método de trabajo

Para el desarrollo de este TFM se ha hecho uso de una metodología ágil basada en *Scrum* [14]. *Scrum* es una metodología basada en un modelo de proceso empírico[1], por este motivo, se definen objetivos simples y se van adaptando conforme el proyecto avanza. La metodología se ha dividido en tres partes, preparación, seguimiento y validación de resultados, se detallan a continuación.

3.1.1. Preparación

Al comienzo del TFM se definieron los entornos de pruebas y las tareas iniciales del trabajo. Respecto a los entornos de pruebas, se determinó seguir una metodología basada en la experimentación sobre entornos reales, que se detallarán en el Capítulo 4. Las tareas consistían en:

- Estudio de las tecnologías *Edge-Cloud Computing*.
- Especificación y análisis de la plataforma *FIWARE* y el uso integrado de la tecnología *blockchain*.
- Familiarización con el entorno de pruebas.
- Instalación y configuración de las tecnologías a utilizar.
- Despliegue de la infraestructura en caso práctico.
- Uso de la plataforma por parte del cliente final.

De este modo, a lo largo del TFM se han ido redefiniendo y adaptando las distintas tareas y, consiguiendo así, los objetivos marcados de forma incremental en base al conocimiento adquirido gracias a los avances realizados.

[1] Que está basado en la experiencia y en la observación de los hechos.

3.1.2. Seguimiento

Para el seguimiento del TFM, se planificaron reuniones quincenales con el director del TFM. Durante estas reuniones, se mostraban los avances realizados a la dirección, se evaluaba el estado del trabajo y se definían las tareas a completar de cara a la próxima reunión.

3.1.3. Validación

Durante las reuniones de seguimiento con el director del TFM se mostraba la ejecución de los experimentos realizados, se valoraba la viabilidad de los experimentos y se definía si las pruebas realizadas eran válidas o se habían identificado errores en la ejecución.

CAPÍTULO 4
Entorno de pruebas y aplicaciones

Este capítulo analiza el soporte *hardware* y *software* en el que se realizó el TFM. Además, se explican las diferentes aplicaciones utilizadas para la realización de los diversos experimentos.

4.1 Entorno hardware

Para la realización de los diferentes experimentos expuestos en el Capítulo 3, será necesario un entorno *hardware* que permita evaluar los experimentos y cumpla los requisitos para poder obtener conclusiones eficientes.

Según la Tabla 2.1 existen diferentes tipos de capas que conforman la arquitectura *Edge-Cloud Computing*. Por este motivo, se ha subdivido en tablas los componentes *hardware* utilizados para cada tarea en el experimento a realizar.

4.1.1. Capa de dispositivos

La Tabla 4.1 muestra los múltiples sensores utilizados para la realización del caso de uso. Los sensores utilizados constituyen la capa de dispositivos dentro de la arquitectura *Edge-Cloud Computing*. Cada sensor se sitúa físicamente en una posición estratégica dentro de la zona del Puerto de Valencia [20], localizado en Valencia, España.

A continuación, se listan las características de cada dispositivo perteneciente a la capa de dispositivos:

- **PEOPLE COUNTER**: Dispositivo encargado de contabilizar el paso de personas en una determinada área.
- **TRAFFIC COUNTER**: Dispositivo encargado de contabilizar el número de vehículos que atraviesan una determinada área.
- **INDOOR ENVIRONMENT SENSOR**: Dispositivo multisensor que determina los valores actuales de un entorno cerrado.
- **OUTDOOR ENVIRONMENT SENSOR**: Dispositivo multisensor que determina distintos valores en entornos abiertos.

Nombre sensor	Modelo	Localización (latitud, longitud)
PEOPLE COUNTER 1	Parametric Radar People Counter with LoRaWAN for Outdoor Applications PCR2-EU868-OD	39.4627, -0.3223
PEOPLE COUNTER 2	Parametric Radar People Counter with LoRaWAN for Outdoor Applications PCR2-EU868-OD	39.4614, -0.3302
PEOPLE COUNTER 3	Parametric Radar People Counter with LoRaWAN for Outdoor Applications PCR2-EU868-OD	39.4602, -0.33219
TRAFFIC COUNTER 1	Parametric TCR-LS LoRaWAN	39.4610, -0.3255
TRAFFIC COUNTER 2	Parametric TCR-LS LoRaWAN	39.4627, -0.3223
TRAFFIC COUNTER 3	Parametric TCR-LS LoRaWAN	39.4625, -0.3242
TRAFFIC COUNTER 4	Parametric TCR-LS LoRaWAN	39.4565, -0.3301
TRAFFIC COUNTER 5	Parametric TCR-LS LoRaWAN	39.4556, -0.3278
TRAFFIC COUNTER 6	Parametric TCR-LS LoRaWAN	39.4594, -0.3326
TRAFFIC COUNTER 7	Parametric TCR-LS LoRaWAN	39.4563, -0.3304
TRAFFIC COUNTER 8	Parametric TCR-LS LoRaWAN	39.4556, -0.3300
TRAFFIC COUNTER 9	Parametric TCR-LS LoRaWAN	39.4557, -0.3289
TRAFFIC COUNTER 10	Parametric TCR-LS LoRaWAN	39.4657, -0.3278
INDOOR ENVIRONMENT SENSOR 1	enLink Air Wireless Air Quality Sensor	39.4611, -0.3310
OUTDOOR ENVIROMENT SENSOR 1	Libelium-Gill-EX-Machina Smart Weather Forecast 4G Solution Kit	39.4619, -0.3253

Tabla 4.1: Hardware capa de dispositivos

4.1.2. Capa edge

En la capa edge de nuestra infraestructura *hardware* se posicionan los dispositivos que se encuentran inmediatamente cercanos a los datos. Estos dispositivos serán los encargados de actuar como intermediarios entre los sensores y el cloud. La Tabla 4.2 muestra los dispositivos utilizados en el caso de uso.

A continuación, se listan las características de cada dispositivo perteneciente a la capa *edge*:

- **LoRaWAN Starter Kit**: Conjunto de sensores y pasarela para demostrar las capacidades de la tecnología LoRaWAN.
- **Pico clúster (5 y 10)**: Un conjunto de dispositivos Raspberry Pi para servir como clúster.
- **LoRaWAN outdoor *gateway***: *Gateway* exterior para recibir datos de sensores de exterior y transmitirlo por la red.
- **LoRaWAN indoor *gateway***: *Gateway* interior para recibir datos de sensores de interior y transmitirlo por la red.

- **Raspberry Pi 4**: Dispositivo Raspberry encargado de recibir y transmitir información entre arquitecturas.
- **Servidor**: Servidor de datos y almacenamiento.

Dispositivo	Modelo	Unidades
LoRaWAN Starter kit MultiTech Conduit IoT Starter Kit.	MTCDT-L4E1-247A-STARTERKIT-868	1
Pico 10 cluster	Pico 10H cluster RPI4	2
Pico 5 cluster	Pico 5 cluster RPI 4	1
LoRaWAN outdoor gateway	MTCDTIP-L4E1-267A-868	2
LoRaWAN indoor gateway	Multitech Conduit MTCDT-L4E1-247A-868-EU-GB	1
Raspberry Pi 4	Raspberry Pi 4	1
Servidor	Proxmox	1

Tabla 4.2: Hardware capa edge

Entorno de desarrollo

Como entorno de desarrollo de la plataforma *FIWARE* se ha empleado una máquina virtual ubicada dentro del servidor de datos con las características descritas en la Tabla 4.3.

En la fase de desarrollo se han creado y depurado el código necesario para ejecutar eficientemente la plataforma *FIWARE* y la tecnología *blockchain*.

Entorno de desarrollo			
Sistema Operativo	**Memoria RAM**	**Procesador**	**Almacenamiento**
Ubuntu 20.04	8,00 GB	Intel Xeon Gold 6230R	60,00 GB

Tabla 4.3: Hardware entorno desarrollo

4.1.3. Capa cloud

En la capa cloud se encuentra el módulo de procesamiento de datos y representación de estos. Aunque disponemos de una nube privada, no se utiliza un entorno *hardware* específico proporcionado especialmente para la realización del TFM.

4.2 Entorno software

Tal y cómo se ha descrito en el Capítulo 3, es necesario agregar el *software* necesario para que los dispositivos *hardware* realicen su función correctamente y sea posible la implementación del caso de uso. En nuestro caso será necesario hacer un estudio del entorno que utilizan los dispositivos en capa de dispositivos para intercomunicarse con los dispositivos en el nodo *edge*, además del *software* utilizado por el nodo *edge* en la etapa de procesamiento y envío de datos a la capa de *cloud*.

Análogamente a la sección anterior, se estudiarán el entorno *software* divididos por los diferentes tipos de capas que componen el entorno *Edge-Cloud Computing*.

4.2.1. Capa de dispositivos

El *software* que utilizan los dispositivos de sensorización del entorno será el propio protocolo de comunicación que utilizarán para transmitir los datos a los dispositivos alojados en la capa *edge*.

LoRaWAN

El protocolo de comunicación que utilizan es *LoRaWAN* [21]. *LoRaWAN* es una tecnología para intercomunicar pequeños dispositivos electrónicos que son empleados en *IoT*. *LoRaWAN* se encarga de gestionar la comunicación y *LoRa* es la propia tecnología de la comunicación en sí. Utilizan comunicación de baja frecuencia y diferentes velocidades de transmisión.

LoRaWAN es una especificación de las redes *LPWAN* (*Low Power Wide Area Network*). Se encarga de unir distintos dispositivos, gestionando los canales y los parámetros locales (ancho de banda, canal, cifrado, etc.).

4.2.2. Capa edge

En la capa *edge*, es importante que los datos recibidos por la capa de dispositivos sean procesados correctamente, por este motivo el protocolo de entrada de datos (*LoRaWAN*), muy eficiente para el envío de datos en crudo (*raw data*), no es eficiente. Por este motivo, debido a que disponemos un dispositivo *hardware* que actúa como *gateway* entre los datos y nuestra infraestructura, podemos transformar y adaptar estos datos a un formato adaptado y mayormente eficiente.

MQTT

El protocolo *MQTT* [22] es liviano y eficiente, permite comunicaciones bidireccionales entre el dispositivo y la nube y es completamente escalable. Esta tecnología está diseñada como un protocolo de mensajería estándar para el *IoT*. Su característica principal es el transporte de mensajería publicación/suscripción ideal para interconectar dispositivos con un ancho de banda mínimo.

Otra de las características principales que nos brinda este protocolo es la posibilidad de implementación de un *Broker MQTT*. Este broker permite la suscripción en tiempo real de clientes a los tópicos (canales) de comunicación para obtener los datos en tiempo real de los dispositivos de sensorización.

VPN

Una vez podemos obtener los datos a través del protocolo *MQTT*, será necesario interconectar de manera eficiente los servicios de comunicación y procesamiento de los datos a través de la red. Aunque existan diversos métodos para transmitir esta información a través de la red, se ha optado por conectar un dispositivo que actúa como *gateway* entre ambas redes, la red interna de obtención de datos y la red de procesamiento de los datos e implementación del *framework* de seguridad.

Una VPN o red privada virtual es uno de los métodos más sencillos de proteger el tráfico a través de la red. Al conectarse a un servidor interno dentro de la plataforma que ofrezca VPN, el propio tráfico fluye a través de un túnel encriptado en el que se necesitarán credenciales para poder acceder. Se realiza estableciendo una conexión virtual punto a punto mediante cifrado criptográfico. Cumple las medidas básicas de seguridad:

- Autenticación.
- Integridad de los datos.
- Confidencialidad de los datos.
- No repudio.
- Control de acceso.
- Auditoría y registro de las actividades.
- Calidad de servicio.

HTTPS

Dado que la tecnología *FIWARE* posee una *API Restful* existen dos posibles variantes para transmitir los datos desde el nodo emisor hasta la propia plataforma. Por una parte, es posible enviar los datos utilizando el propio protocolo *MQTT* utilizado anteriormente, el problema principal de esta opción es que sería necesario agregar un nuevo componente a nuestra plataforma que fuera capaz de transformar el protocolo de transmisión de *MQTT* a *HTTP*.

Aprovechando que poseemos un entorno controlado con el que poder transformar los datos libremente, se ha optado por transformar el protocolo de envío de *MQTT* a *HTTPS* desde el propio nodo emisor. Además, esta opción proporciona una capa de seguridad extra en el envío de los datos a través de la red interna, previniendo de si algún atacante lograra interceptar la red y pudiera interceptar las comunicaciones entre componentes, los datos viajarían cifrados a través del algoritmo de cifrado y protocolo de transporte *TLS*[1].

Por último, este protocolo de envío de datos también será utilizado para enviar los datos cifrados a través de internet desde la plataforma *FIWARE* hacia la capa *cloud*, donde se representarán los datos obtenidos y se comprobará la veracidad de estos.

4.2.3. Capa cloud

Análogamente a la subsección anterior, se utiliza el protocolo *HTTPS* para recibir datos al entorno *cloud*. Además, se utilizan *tokens* de autenticación en la plataforma para asegurar la veracidad del cliente o sistema que adquiere las fuentes de datos.

[1]Permite y garantiza el intercambio de datos en un entorno securizado y privado entre dos entes, el usuario y el servidor a través del uso de algoritmos criptográficos basados en clave pública/privada, utiliza comunicación TCP/IP y posee mecanismos de de integridad, vigencia y emisión de certificados públicos y privados

4.3 Aplicaciones

Gracias al conocimiento del entorno *hardware* y *software* que disponemos para la realización de este proyecto, se ha definido las aplicaciones que serán directamente responsables en la aplicación de nuestro caso de uso y la posible creación de un *«Framework de seguridad en entornos Edge-Cloud Computing»*. Análogamente a las subsecciones anteriores, se dividirán las aplicaciones utilizadas en capa *edge* y cada *cloud* (no se utilizará la capa de dispositivos puesto que no poseen aplicaciones subyacentes a las utilizadas directamente por el *software* preinstalado en los dispositivos de sensorización).

4.3.1. Capa edge

Docker

Docker es una plataforma *software* de código abierto [23] para poder crear, implementar y administrar contenedores que contienen aplicaciones virtualizadas en un sistema operativo común. La característica principal de encapsulación de *software* permite la creación de imágenes (contenedores que contienen aplicaciones, bibliotecas, archivos de configuración, dependencias y otros servicios) que aplican las dependencias necesarias para poder ejecutar código independientemente del sistema operativo que utilice el *host*.

Existen otras tecnologías que cumplen las mismas necesidades de encapsulación y ejecución de código como 'Singularity' [24], en este caso se ha optado por esta tecnología por su facilidad de instalación e implementación, además de la integración que dispone junto a otras aplicaciones utilizadas en el desarrollo de este trabajo.

Kubernetes clúster

Un clúster de kubernetes es un conjunto de máquinas que pueden ser de uno o varios nodos de trabajo que permite la ejecución controlada de aplicaciones en contenedores [25]. Cada clúster cuenta con un plano de control al menos, que es encargado de mantener el estado deseado de las máquinas y de controlar las aplicaciones que se ejecutan, los recursos *hardware* que pueden utilizar, el tiempo de ejecución y las imágenes de contenedores (Docker en este caso de uso) que se utilizan, entre otras configuraciones.

Es posible desplegar un clúster de kubernetes por cada servidor físico que dispongamos, gracias a esta característica, podemos definir varias rutas de envío de los datos, obteniendo una mejora de rendimiento y obteniendo la posibilidad de utilizar redundancia para poder disponer de un mecanismo de funcionamiento secundario en caso de un clúster deje de funcionar.

ChirpStack AS

ChirpStack es un *stack* LoRaWAN de código abierto [26]. Ofrece todos los servicios de *backend* LoRaWAN. Aporta cifrado y validación de la integridad de los mensajes enviados a través del protocolo LoRaWAN, realiza las funciones de puerta de enlace entre los datos LoRaWAN y *MQTT* y cuenta con una interfaz gráfica de configuración.

La implementación de esta aplicación ha sido posible gracias a su fácil integración en entornos clusterizados, en este caso en *Pico-cluster* (clúster funcional ensamblado por módulos Raspberry PI).

Node-RED

Node-RED es una herramienta de programación visual que permite la integración de intercomunicación entre nodos y procesos. Para el desarrollo de nuestro entorno de trabajo, se ha optado por utilizar esta tecnología como *gateway* entre las conexiones recibidas a través del protocolo *MQTT* e inmediatamente reenviar esta información a través del protocolo seguro *HTTPS*.

Orion Context Broker

Entrando dentro de la tecnología *FIWARE* nos encontramos con el nodo central de la infraestructura. *Orion Context Broker* (OCB) [27] es el componente encargado de administrar el ciclo de vida de la información. A través del OCB es posible registrar diferentes entidades de contexto y ser administradas a través de actualizaciones y consultas. Además, es posible suscribirse a esta información de contexto para que cuando se produzca alguna modificación automáticamente se reciba una notificación (modelo publicador/-suscriptor).

OCB utiliza el modelo *NGSI* caracterizado por la definición de entidades y atributos. Las entidades son la representación visual de todos los objetos físicos en el mundo, mientras que los atributos es la información disponible sobre las entidades físicas, expresadas de manera virtual.

La Figura 4.1 muestra de forma genera el funcionamiento del OCB. Se considera un intermediario entre los productores (dispositivos) y los consumidores (aplicaciones). Como hemos mencionado en el subapartado anterior, realizamos consultas a través de *Node-RED* a través del protocolo *HTTPS*, esto se debe a que la forma principal de interactuar con el OCB es a través de estas llamadas, que incluirán el código correspondiente de creación, actualización o eliminación de atributos.

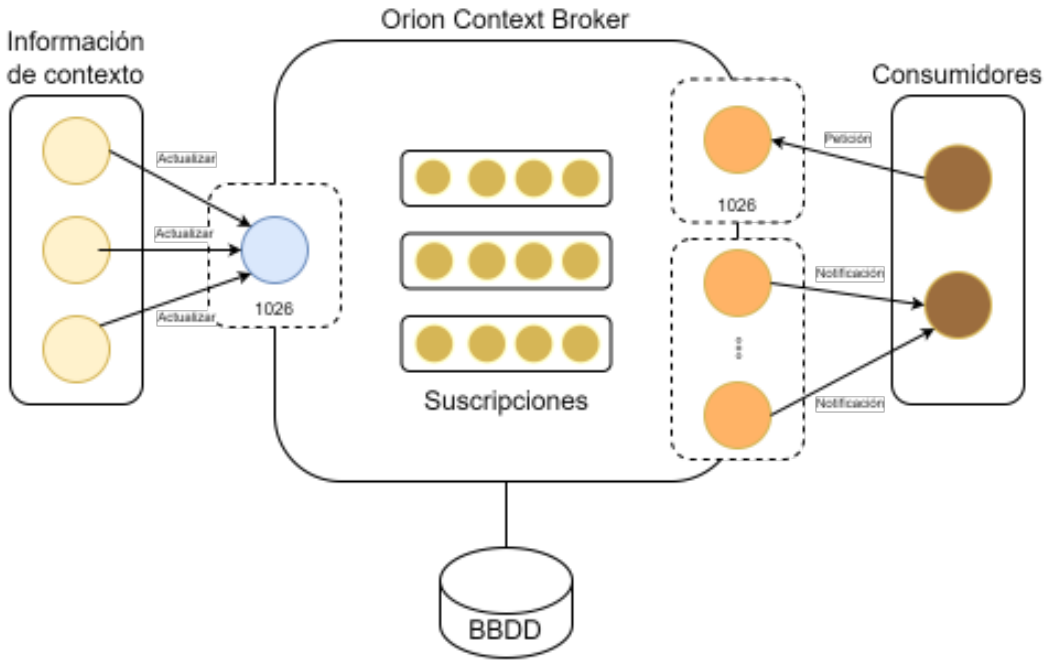

Figura 4.1: Funcionamiento principal de *Orion Context Broker*

Keyrock Identity Manager

Keyrock [10] es el componente *FIWARE* responsable de gestión de identidad. El uso de *keyrock* dentro de la plataforma *FIWARE* permite agregar seguridad de autorización y autenticación basada en *OAuth2*[2] a sus servicios y aplicaciones.

Wilma Pep Proxy for Orion

Un *proxy PEP* se encuentra en el medio a un recurso seguro y es un punto final que se encuentra en una ubicación pública conocida. En otras palabras, este recurso se utiliza como un puente entre el origen (un cliente o un dispositivo) y el destino de la solicitud (nodo *edge*). Permite la conexión a un servicio o recurso de forma indirecta.

FIWARE Wilma [11] es una implementación simplificada de un *proxy PEP* para funcionar junto a *FIWARE Keyrock*. Cada vez que un usuario desea obtener acceso a un recurso del *proxy PEP*, deberá describir los atributos que desea leer o escribir, además de las credenciales de acceso. Gracias a este proceso, el *software* podrá aceptar o denegar la petición según el nivel de acceso de usuario, credenciales correctas o permisos de lectura/escritura sobre el recurso enviado.

Canis Major for Orion

Uno de los principales objetivos de este Trabajo Fin de Máster es introducir un *Framework* de seguridad basado en la tecnología *Blockchain* dentro de un entorno *FIWARE*. Canis Major [28] es un adaptador de *blockchain* que permite la persistencia y verificación de transacciones de entidades *NGSI*. Además, es la **primera solución** viable de agregación de seguridad de incertidumbre de los datos enfocada a plataformas *Edge-Cloud Computing*.

La Figura 4.2 muestra la visión general de actuación de *Canis Major* dentro de la tecnología *FIWARE*. Para poder conservar las transacciones dentro de una cadena de bloques, el cliente deberá enviar información sobre sus transacciones (actualización/creación/eliminación de atributos) a *Canis Major*. La solicitud deberá incluir información relativa a la billetera (*Wallet* en inglés) que será la que se utilizará para firmar esta transacción.

4.3.2. Capa cloud

Google Sheets

Una vez tenemos desplegada la infraestructura por parte del servidor, recolectando los datos y listos para ser enviados a cualquier cliente de manera segura y a tiempo real, se ha desarrollado una aplicación automática que reflejará los datos en tiempo real al usuario final y será completamente automática y transparente al usuario.

A través de una biblioteca de *python* denominada *pygsheets* [29] conseguimos acceso a hojas de cálculo de Google a través de *Google Sheets API v4*. Gracias a esta característica, podemos introducir las credenciales de acceso a los datos a nuestra plataforma que contiene los datos relativos a los dispositivos de sensorización y exportarlas a una hoja de *Google Sheets*.

[2] OAuth2 es un framework de autorización, que permite a las aplicaciones obtener acceso (limitado) a las cuentas de usuario de determinados servicios

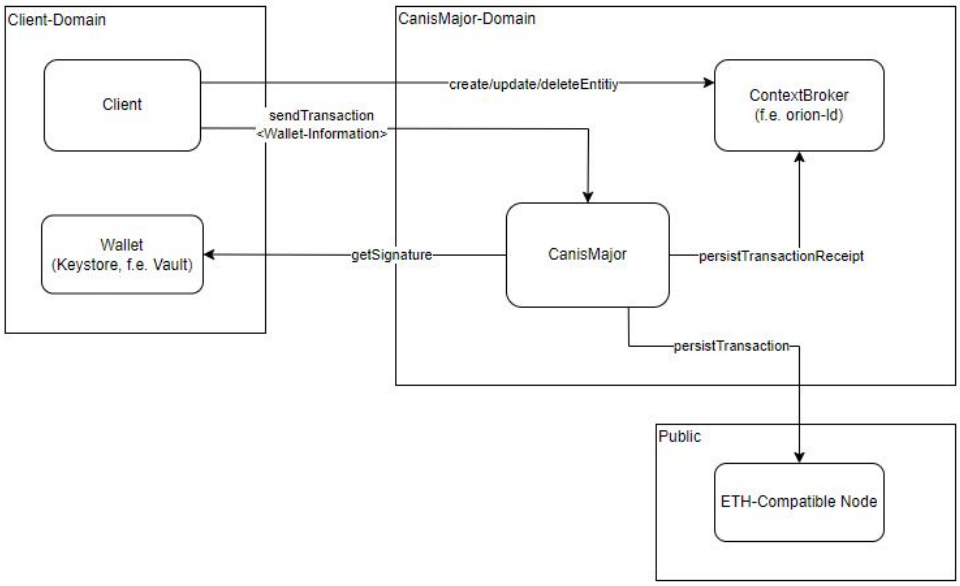

Figura 4.2: Visión principal de *Canis Major* para *FIWARE*

Geosheets

Una vez disponemos de los datos en tiempo real dentro de una hoja de *Google Sheets*, podremos visualizar estos datos de manera gráfica y fácil de entender para el usuario.

Geosheets [30] es un complemento para *Google Sheets* que permite visualizar distintos datos a través de una jerarquía de datos expuestos en cada hoja de *Google Sheets*. Es posible visualizar los datos en tiempo real y se actualiza cada vez que los datos con modificados en nuestra base de datos.

La Figura 4.3 muestra un ejemplo de visualización final de los datos recibidos por el usuario en tiempo real, incluyendo latitud, longitud y nombre del dispositivo, junto a los atributos correspondientes.

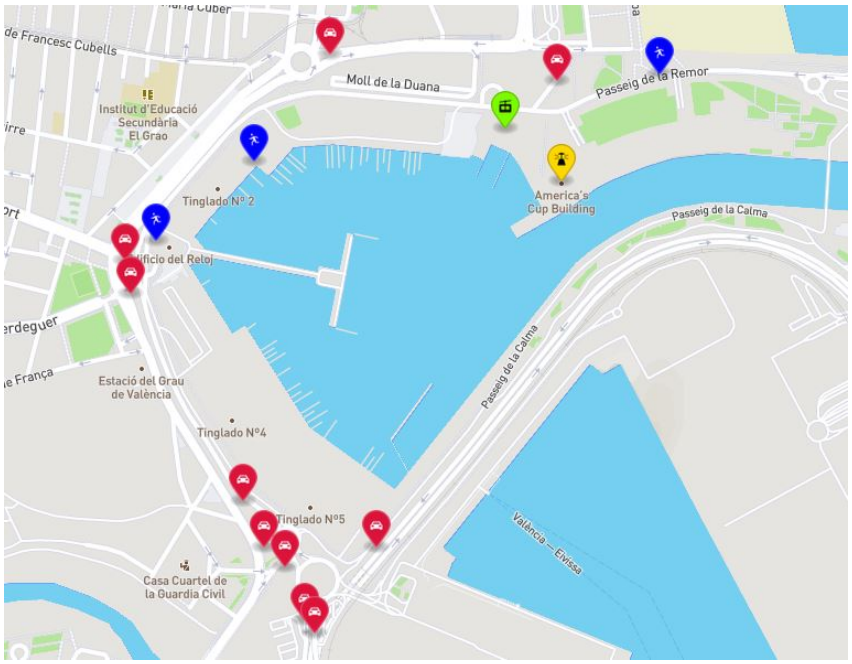

Figura 4.3: Ejemplo visualización de dispositivos en Geosheets

4.4 Infraestructura general

Una vez conocemos tanto el entorno *hardware*, entorno *software* y las aplicaciones involucradas en la realización de este TFM, la Figura 4.4 representa de manera gráfica la infraestructura general utilizado en el desarrollo e implementación del caso de uso asignado a este proyecto.

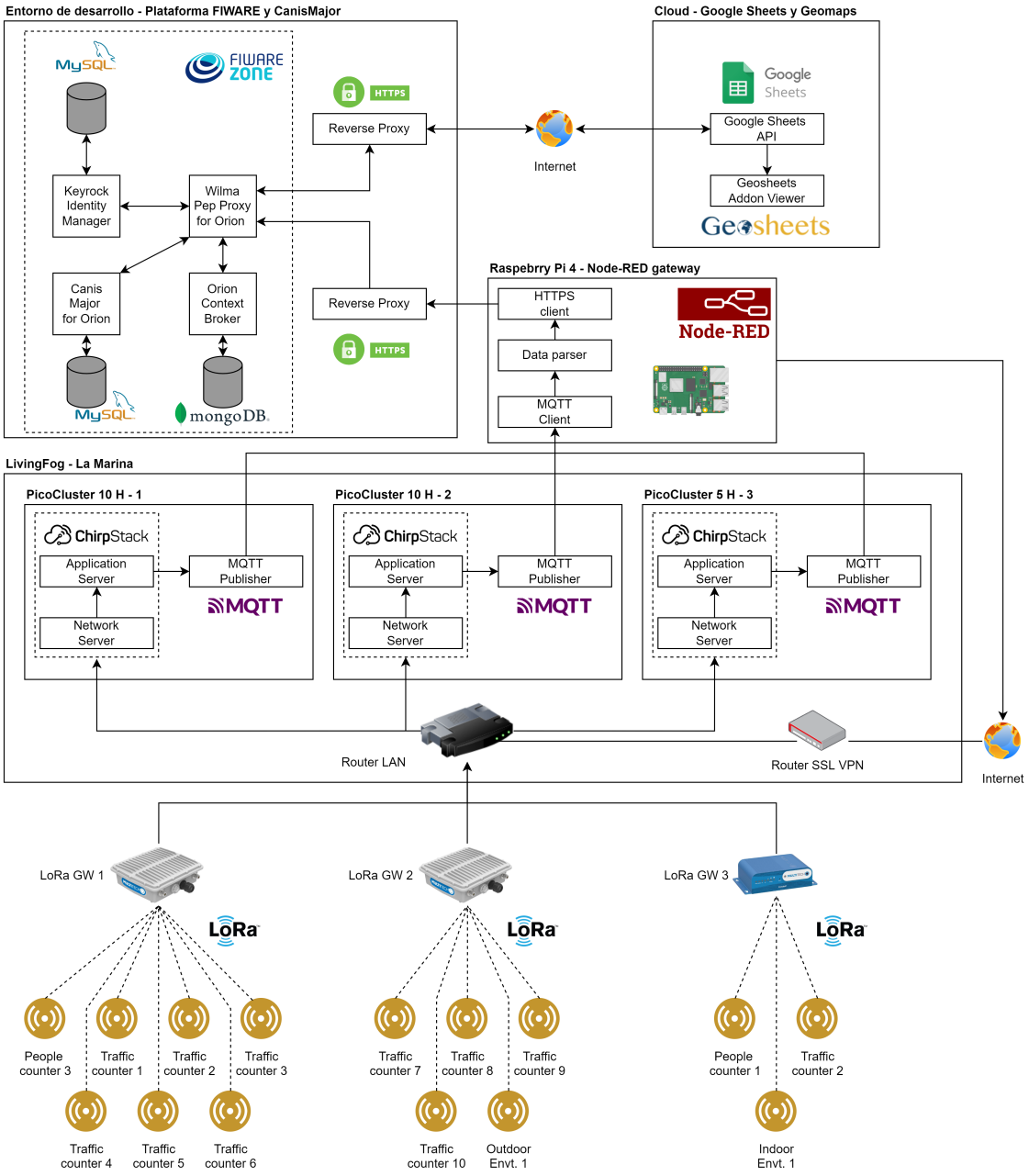

Figura 4.4: Infraestructura general

CAPÍTULO 5
Implementación y despliegue de la solución propuesta

En este capítulo se van a detallar los procesos de instalación y configuración del entorno de desarrollo tanto de la parte del proveedor de servicios como del cliente. Además, se explica el proceso de creación de las imágenes *docker*, usabilidad y pruebas de despliegue. También se han realizado distintas pruebas por la parte de un cliente final en relación a obtener el historial de transacciones realizados y guardados en la cadena de *blockchain* para poder terminar así, basándonos en los resultados obtenidos, de las prestaciones que nos ofrecen los entornos *Edge-Cloud Computing* junto al *framework* de seguridad implementado y así poder contribuir a futuros proyectos.

5.1 Instalación y configuración

A continuación, se explica y detalla el proceso de instalación y configuración de todos los entornos que han sido participes en la realización de este proyecto:

5.1.1. LivingFog - La Marina

El primer entorno a instalar y configurar es el entorno prestado por el proyecto **LivingFog**, realizado por el proyecto europeo **FogGuru** [34]. Este permite a los usuarios la posibilidad de conectar dispositivos *IoT* utilizando el protocolo inalámbrico de larga distancia *LoRa* a grupos locales de *Raspberry PI* donde es posible procesar los datos.

La Figura 5.1 muestra la infraestructura que se ha seguido para realizar el proceso de instalación, desde la instalación de los sensores de datos hasta el envío de los datos a través del protocolo de comunicación *MQTT*. A continuación, se describe paso a paso el proceso realizado para asegurar el funcionamiento interno de esta infraestructura.

Configuración servidores de datos - PicoCluster 1, 2 y 3

1. Preparación de cada dispositivo Raspberry PI para crear el clúster.
2. Creación de un clúster de Kubernetes en cada PicoCluster.
3. Instalación del *software* del *gateway* - **Chirpstack**.
4. Configuración del dispositivo **Chirpstack** para conectarse al clúster.

5. Habilitar la conexión entre el clúster, gateway *LoRa* y los dispositivos de sensorización mediante la interfaz de usuario (GUI) del servidor de aplicaciones **Chirpstack**.

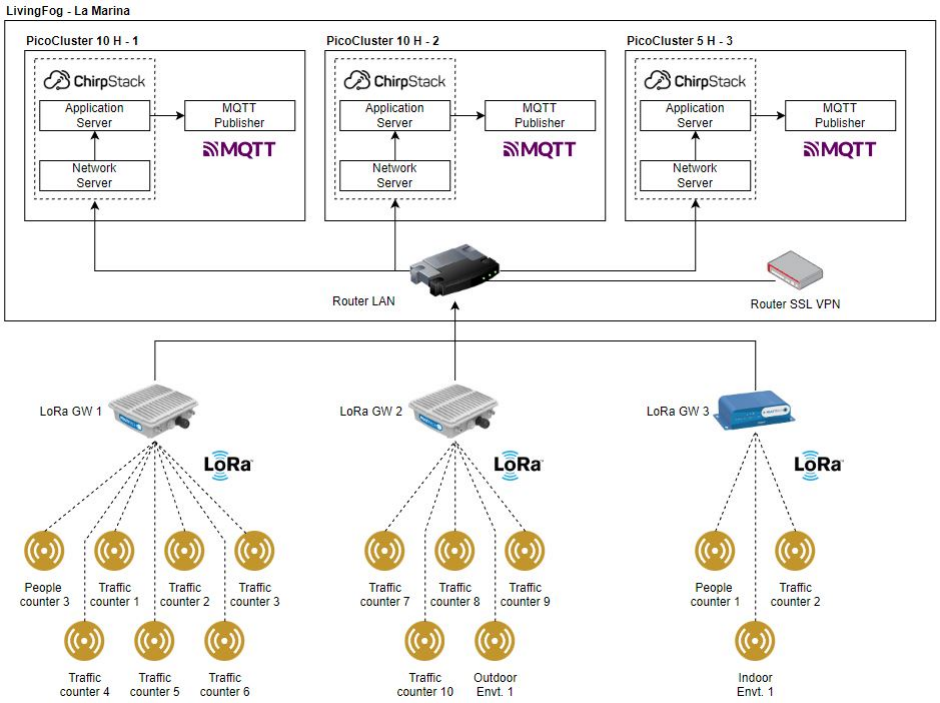

Figura 5.1: Infraestructura general - Entorno LivingFog - La Marina

Para el desarrollo de este TFM, se ha dispuesto de 3 *PicoClusters* ensamblados y configurados, por lo que consta como instalación y configuración refiere a partir del paso 3º, instalando el *software* relativo al *gateway* utilizado hasta la configuración final de los dispositivos de sensorización y conexión entre todo el entorno.

En primer lugar, accedemos a la interfaz gráfica que nos proporciona el *software* **Chirpstack** con el usuario y contraseña predefinidos. Una vez iniciado sesión, se visualizará la pantalla definida en la Figura 5.2.

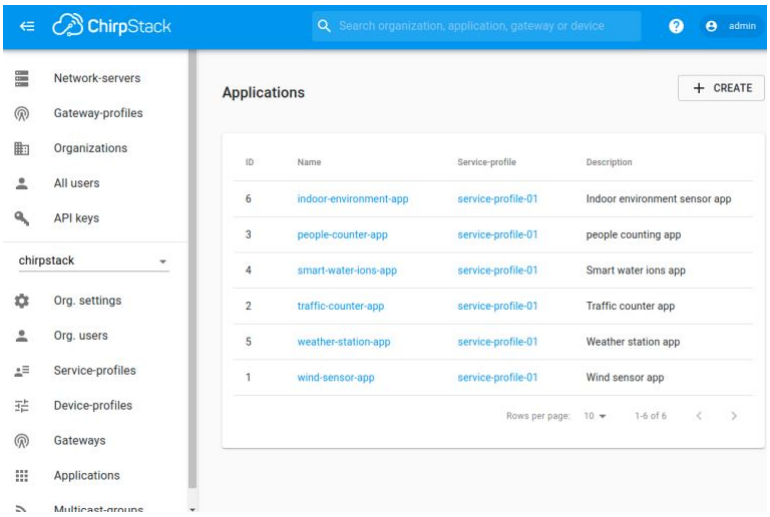

Figura 5.2: Pantalla principal interfaz gráfica ChirpStack

El proceso de agregación de nuevos sensores será el siguiente:

1. Crear un perfil de dispositivo, en la pantalla 'Perfiles de dispositivos', visualizado en la Figura 5.3.
2. Completar los campos en la pestaña 'General' y guardar los cambios.
3. Crear una aplicación en la pestaña 'Aplicaciones' y completar los datos requeridos, visualizado en la Figura 5.4.
4. Una vez creada la aplicación, se añaden los dispositivos relacionados, visualizado en la Figura 5.5.
5. Por último, se repite el proceso de creación de aplicaciones y dispositivos para cada tipo de dispositivos disponibles, estos se podrán visualizar nuevamente en la Figura 5.2.

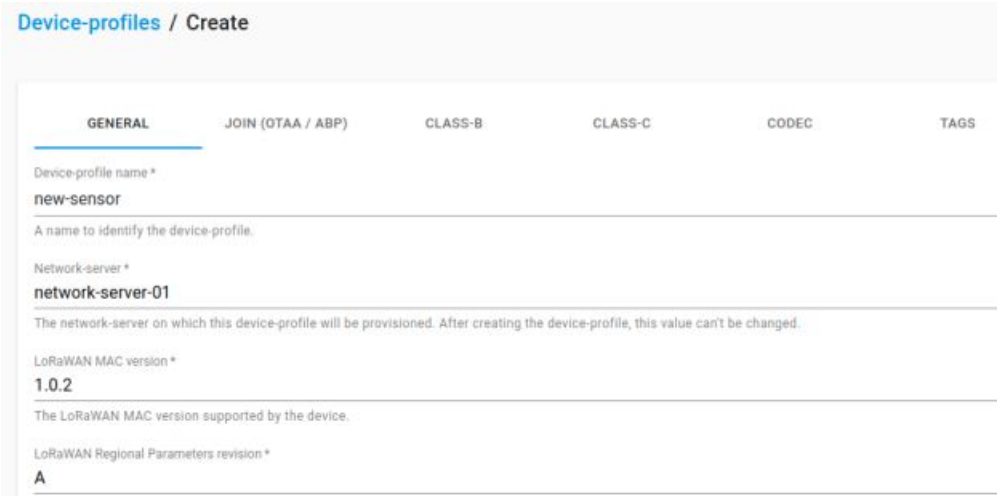

Figura 5.3: ChirpStack - Perfiles de dispositivos

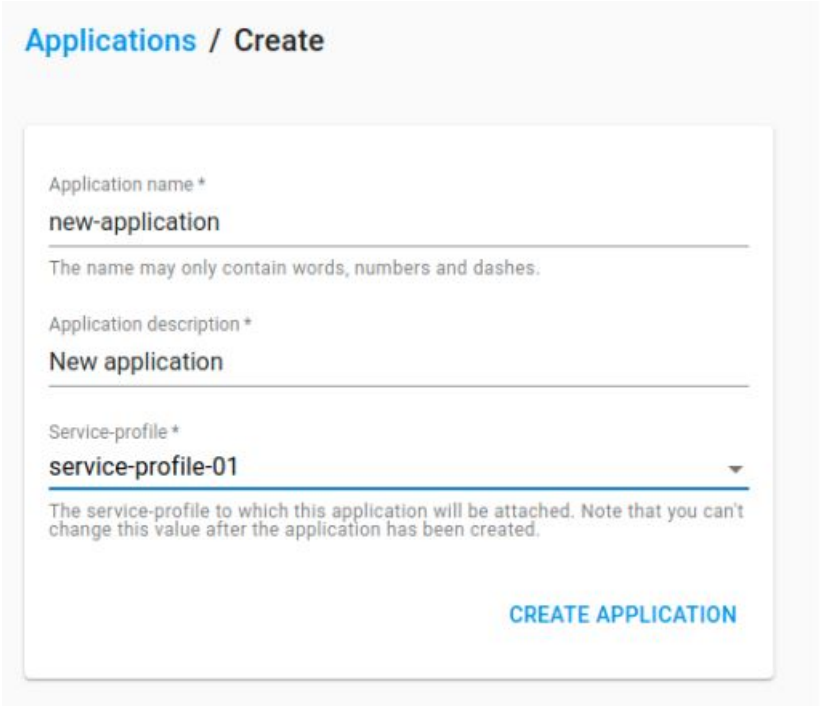

Figura 5.4: ChirpStack - Creación de aplicaciones

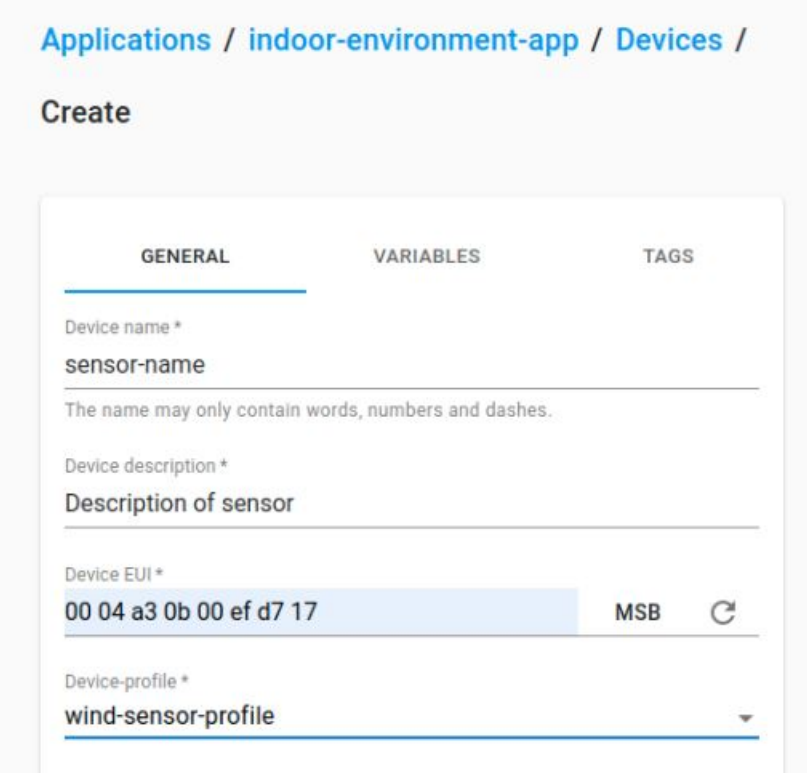

Figura 5.5: ChirpStack - Añadir dispositivos a las aplicaciones

Una vez hemos configurado los servidores de datos, estarán listos para publicar los datos recibidos mediante el protocolo de comunicación *MQTT*. A continuación, configuraremos los *gateways LoRa* para terminar la configuración inicial y poder intercomunicar toda la infraestructura *Living-Fog*.

Configuración gateway LoRa

Los *gateway LoRa* realizan la mediación entre los dispositivos de sensorización y el *software Chirpstack*. Al no contar con una interfaz gráfica de configuración, se debe configurar a través de un *shell* remoto, los pasos son los siguientes:

1. Iniciar sesión en el *gateway LoRa* a través de *ssh*.
2. Abrir el archivo de configuración principal 'chirpstack-gateway-bridge.toml'.
3. Actualizar puerto UDP seleccionado para reenviar la información de paquetes *LoRa*.

```
udp_bind = "0.0.0.0:1782"
```

4. Editar la información sobre el servidor *MQTT*

```
# Generic MQTT authentication.
[integration.mqtt.auth.generic]
# MQTT server (e.g. scheme://host:port where scheme is tcp, ssl
    or ws)
server="tcp://192.168.9.10:1883"
# Connect with the given username
username="USERNAME"
# Connect with the given password
password="PASSWORD
```

5. Guardar el archivo.

6. Reiniciar el servicio *gateway LoRa*.

```
sudo /etc/init.d/chirpstack-gateway-bridge restart
```

Una vez hemos realizado todos los pasos requeridos, tendremos instalado y configurado el entorno *LivingFog* que nos proporcionará a nuestro *Node-RED Gateway* los datos de los dispositivos de sensorización a través del protocolo de comunicación *MQTT*.

5.1.2. Raspberry Pi 4 - Node-Red Gateway

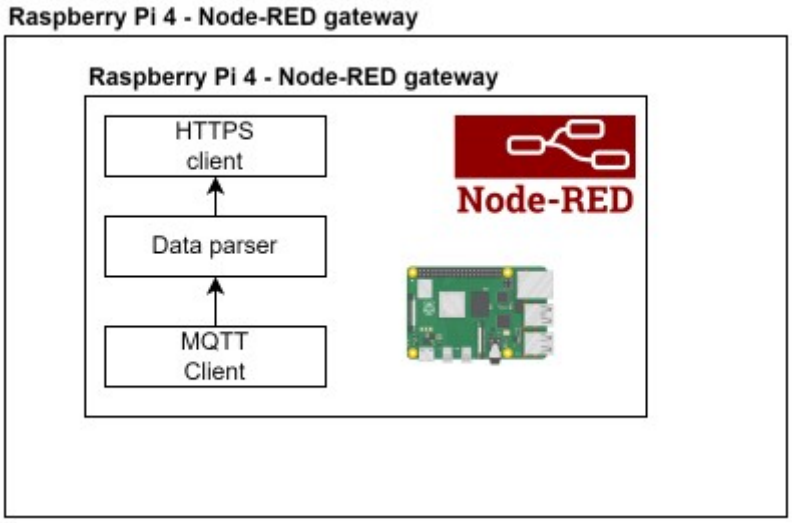

Figura 5.6: Infraestructura general - Raspberry Pi 4 - Node-Red Gateway

El siguiente entorno configurado y preparado trata de un dispositivo *Raspberry Pi 4* que actúa como *gateway*[1] entre el entorno visto en el apartado 5.1.1 hacia el entorno de desarrollo 5.1.3. Su función principal es traducir los datos recibidos por el protocolo *MQTT* en crudo (*raw data*) hacia un formato entendible por el *Context Broker* de la plataforma *FIWARE*.

Conseguimos establecer conexión con el entorno 5.1.1 a través de una conexión *VPN* con un usuario/contraseña proporcionado por el proyecto la cual nos da acceso a todos los dispositivos involucrados según la Figura 4.4 y obtenemos una *IP* privada.

La Figura 5.6 muestra el programa utilizado y el proceso de traducción y envío de datos desde el protocolo *MQTT* al protocolo *HTTPS*. *Node-RED* [31] es una herramienta de programación visual que se implementa en dispositivos *hardware*. Trabaja mostrando de manera visual las funciones y relaciones. Cuenta con una gran cantidad de nodos de comunicación y funciones para modificar/enviar/recibir datos de manera sencilla. Este sistema de representación es capaz de ayudar al usuario a visualizar gráficamente el flujo de datos.

La Figura 5.7 muestra el flujo de datos realizado y especificado de manera más corporativa en la Figura 5.6. Tenemos un flujo individual para cada tópico de envío de información, la cual representa los valores y características de un sensor en concreto.

[1]Puerta de enlace que actúa de interfaz de conexión entre aparatos o dispositivos. Traduce el protocolo utilizado en una red inicial al protocolo usado en la red de destino.

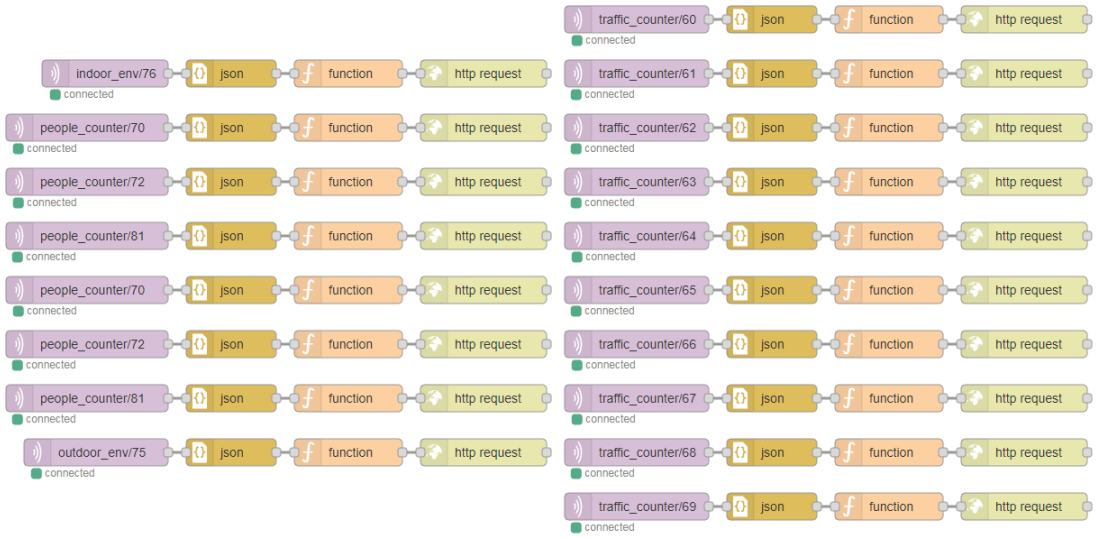

Figura 5.7: Flow de datos - Node-RED

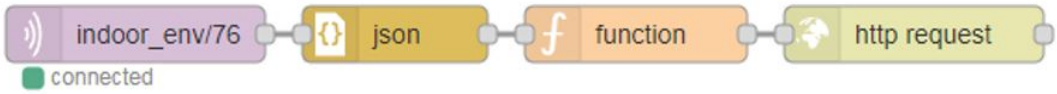

Figura 5.8: Representación flow de datos - Node-RED

A continuación, se definen cada uno de los procesos utilizados por cada flujo de información para conseguir recibir y enviar los datos correctamente en sus protocolos correspondientes.

MQTT Client

El primer recuadro de la Figura 5.8 representa el cliente *MQTT*. Este cliente está conectado al tópico correspondiente a cada sensor asignado y utiliza un sistema de usuario/contraseña para establecer la conexión y poder recibir los datos correctamente.

JSON parser

Los datos son recibidos por el cliente *MQTT* en formato *YAML*. El segundo recuadro de la Figura 5.8 convierte el formato del mensaje recibido a formato *JSON*. Este paso no es necesario aunque nos ayudará más adelante en la modificación del protocolo de envío de datos.

Function

El tercer recuadro de la Figura 5.8 representa la función principal que actuará sobre los datos recibidos a través del protocolo *MQTT*, estos datos son agregados a un *payload* que formará parte del envío al entorno de desarrollo a través del protocolo *HTTPS*. El Listing 5.1 muestra un ejemplo de adaptación y agregación de los *headers* necesarios para el envío de los datos.

Además, para poder enviar los datos al entorno de la plataforma *FIWARE* será necesario añadir los *tokens* correspondientes. El token **'X-Auth-Token'** representa el token generado por *keyrock* para el envío de datos a través del *Pep-Proxy*, el token **'DLT-Token'**

será el generado para registrar y actualizar la cadena de bloques de *blockchain* referente al objeto modificado. Estos conceptos se verán en profundidad en la Secciones 5.1.3 y 5.1.3.

El Listing 5.1 muestra un ejemplo de adaptación y agregación de los *headers* necesarios para el envío de los datos.

```
var message = {}

message.headers = {{'Content-Type': 'application/json'},
                   {'X-Auth-Token': 'keyrocktoken'},
                   {'DLT-Token': 'dlttoken'}};

message.payload =
     {"SBX_BATT":{"type":"Number", "value": msg.payload.SensorData.SBX_BATT},
      "SBX_PV":{"type":"Number", "value": msg.payload.SensorData.SBX_PV},
      "TEMP":{"type":"Number", "value": msg.payload.SensorData.TEMP}};

return message;
```

Listing 5.1: Ejemplo payload para envío de datos a través del protocolo HTTPS.

HTTP Request

El último recuadro de la Figura 5.8 muestra la petición de envío *HTTPS* hacia el entorno de desarrollo. En este caso la dirección de envío se compone de los siguientes campos:

- **Protocolo de envío**: HTTPS
- **Dirección IP**: 192.168.1.144
- **Puerto de envío**: 4430
- **Ruta**: /orion/v2/entities/
- **Parámetros principales**: Nombre entidad (sensor). Ej.: /people_counter_1/
- **Parámetros secundarios**: /attrs/

Ejemplo de petición:

https://192.168.1.144:4430/orion/v2/entities/people_counter_1/attrs

5.1.3. Entorno de desarrollo - Plataforma FIWARE y Canis Major

En el entorno de desarrollo se desenvuelve el grueso de proyecto. Se compone de una máquina virtual que consta de un sistema operativo **Ubuntu 20.04** que funciona como servidor central de almacenamiento y procesamiento de datos, además, contiene la lógica necesaria para proporcionar acceso a los clientes que deseen obtener los datos referidos y el historial *blockchain* de cada entidad creada. De nuevo, destacamos la novedad de implementación de una solución basada en *blockchain* enfocada a entornos *Edge-Cloud Computing*. Gracias a la posibilidad de poder **identificar el historial de transacciones y actualizaciones de los datos**, será posible encontrar una viabilidad a este proyecto al generar confianza sobre los clientes finales que reciben los datos, pues estos no podrán

ser modificados una vez la red descentralizada *blockchain* confirme estas transacciones en su cadena de bloques.

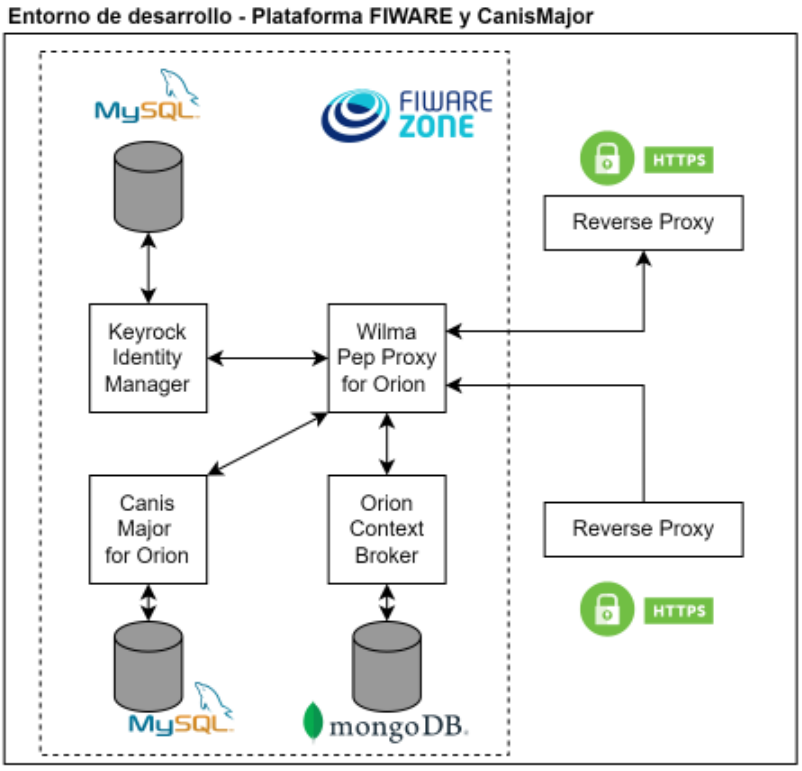

Figura 5.9: Entorno de desarrollo - Plataforma FIWARE y CanisMajor

La Figura 5.9 muestra la infraestructura interna del entorno de desarrollo principal. A continuación, se desarrollan las funcionalidades de cada elemento y cómo influyen el flujo de compartición y administración de los datos recibidos.

Reverse proxy

Existen dos imágenes docker personalizadas encargadas de recibir las peticiones *HTTPS*, una de ellas está enfocada a recibir los datos por parte del nodo *gateway* y otra a procesar las peticiones por parte del cliente final. El concepto de *reverse proxy* es utilizado para generar una comunicación externa a través de certificados SSL. Una vez la petición llega y se procesa dentro del entorno privado, la petición es redirigida al elemento correspondiente a través del protocolo *HTTP*.

Dependiendo del puerto que se utilice para realizar la llamada *HTTPS* se utilizará un *reverse proxy* u otro. El único *reverse proxy* que tendrá el puerto abierto para poder recibir peticiones externas será el enfocado a la parte del cliente. A continuación, en la tablas 5.1 y 5.2 se muestran las rutas permitidas en las llamadas al *reverse proxy* y la redirección que realizan.

Reverse proxy gateway :443 ->:443 (HTTPS)		
Ruta	**Elemento de redirección**	**Ruta redirección**
https://<private_ip>/orion/	Wilma Pep-Proxy	http://orion-proxy:1027

Tabla 5.1: Rutas permitidas *reverse proxy* gateway

Reverse proxy cliente :4430 ->:443 (HTTPS)		
Ruta	**Elemento de redirección**	**Ruta redirección**
https://<public_ip>/orion/	Wilma Pep-Proxy	http://orion-proxy:1027
https://<public_ip>/keyrock/	Keyrock	http://keyrock:3005
https://<public_ip>/canismajor/	Canis Major	http://canismajor:4000

Tabla 5.2: Rutas permitidas *reverse proxy* cliente

En el Apéndice A.1 se detalla el archivo de configuración utilizado para la creación de la imagen del *reverse-proxy* para el cliente a modo de ejemplo.

Wilma Pep-Proxy for Orion

La imagen docker de *Wilma Pep-Proxy for Orion* es la encargada de la comunicación entre los *reverse proxy* y la plataforma *FIWARE*. Este elemento de comunicación se encarga de procesar las llamadas *HTTP* recibidas y comprobar si los requisitos de autenticación (**Keyrock token** y **Canis Major token**) son correctos.

Para la realización de este entorno que integra la validación por *blockchain* se ha utilizado una imagen docker modificada [32] que permite el reenvío de información a *Canis Major* en caso de que el *token* de DLT exista y sea correcto.

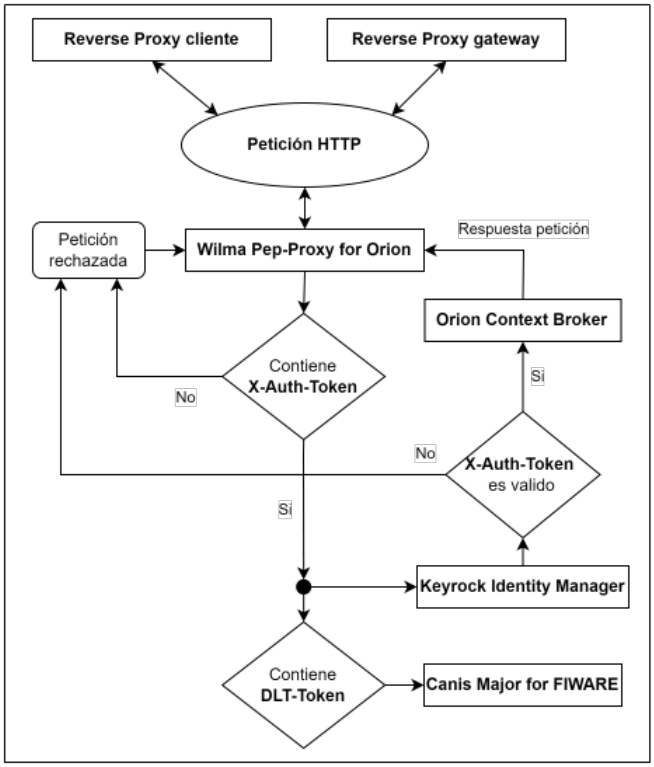

Figura 5.10: Diagrama de flujo comunicación *Wilma Pep-Proxy*

Cuando *Wilma Pep-Proxy* recibe una petición *HTTP* envía la información relativa al token de autenticación **'X-Auth-Token'** a la imagen *keyrock* que se encargará de aceptar o denegar la petición. En el caso de aceptar la petición, enviará la información recibida a la imagen *Orion Context Broker* y en caso de que exista el token **'DLT-Token'** enviará la

información a la imagen *Canis Major for Orion*. Una vez realizado el proceso, devolverá el resultado de la operación.

El Diagrama 5.10 muestra el proceso de comunicación entre **Wilma Pep-Proxy** y las demás imágenes involucradas en la plataforma *FIWARE*.

Keyrock Identity Manager

Keyrock Identity Manager (en adelante *keyrock*) es el responsable de la gestión de identidad de usuarios y dispositivos asociados a la plataforma *FIWARE*. Contiene las credenciales (usuario/contraseña) de todos los clientes y dispositivos asociados, además de las credenciales *OAuth2* correspondientes al servicio de *Orion Context Broker*. Un cliente deberá enviar una petición a *keyrock* para obtener el token **'X-Auth-Token'** para poder enviar peticiones de creación/modificación/eliminación de entidades.

Además, cuenta con una interfaz para dar de alta/baja a los distintos usuarios y dispositivos. La Figura 5.11 muestra la información relativa al *Pep-proxy* utilizado. La Figura 5.12 muestra la información relativa a las credenciales de uso del servicio *Orion Context Broker*. Las Figuras 5.13 y 5.14 muestran la información relativa a los clientes y sensores habilitados para su uso.

Figura 5.11: Keyrock - Información Wilma Pep-Proxy

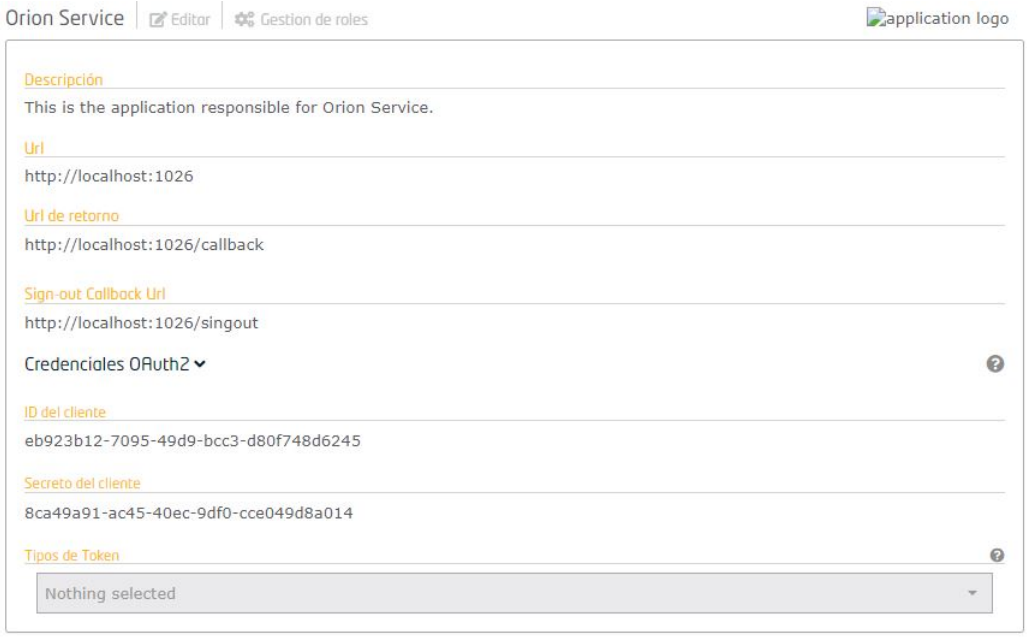

Figura 5.12: Keyrock - Información servicio Orion Context Broker

Figura 5.13: Keyrock - Información clientes habilitados

Figura 5.14: Keyrock - Información Dispositivos habilitados

Orion Context Broker

Orion Context Broker (en adelante *Orion*) es el principal y único componente obligatorio en cualquier solución desarrollada bajo la plataforma *FIWARE*. Permite administrar la información de contexto, consultarla y actualizarla.

Cuando recibe una petición de creación o modificación de una entidad desde *Wilma Pep-Proxy*, la identifica y si los datos son correctamente enviados los almacena en una base de datos **Mongo-DB** (base de datos no relacional o NoSQL). Cuando *Wilma Pep-Proxy* realiza una petición de consulta, *Orion* devuelve la información relativa a la entidad consultada.

En resumen, se ha utilizado una imagen genérica de *Orion Context Broker* junto a una base de datos no relacional *Mongo-DB*.

Canis Major

Canis Major es un adaptador de cadena de bloques que admite varios DLT[2], tiene como objetivo enviar los datos a DLT y funciona para almacenar información relativa a entidades de datos. La imagen Canis Major creada recomienda el uso de **contratos AEI para los clientes *Ethereum***. Estos tipos de contratos son creados a partir de una nueva imagen agregada escrita en *Solidity* utilizando el estándar *ERC721* (NFT).

Este Trabajo Fin de Máster está enfocado en un entorno de desarrollo de un Plan de Producción (PdP) y tiene como objetivo simular la producción de forma aislada y en un entorno privado. Por este motivo, no utilizaremos un sistema *Ethereum* real asignado una cartera (*wallet* en inglés) de criptomonedas. En cambio, se ha utilizado una imagen definida como **Ganache-cli**.

Ganache [33] es un simulador de Ethereum que hace que el desarrollo de aplicaciones Ethereum sea más accesible a proyectos de desarrollo, fácil y seguro. Incluye todas las

[2]Distributed Ledger Technology (DLT) o Tecnología de Contabilidad Distribuida es un sistema electrónico o base de datos para registrar información que no es ejecutada por una sola entidad. Esta nos permiten almacenar y usar datos que pueden ser descentralizados y distribuidos tanto de forma privada o pública.

funcionalidades y características populares de RPC (como eventos) y es posible manejarlo de forma determinista en el desarrollo. A partir de la imagen creada de *ganache-cli* se nos proporciona diferentes cuentas *ethereum* simuladas junto a sus claves privadas, que serán en conjunto los argumentos necesarios para la creación de nuestro token para *Canis Major* **'DLT-Token'**. Cuando tengamos el conjunto de claves (pública/privada) podremos crear la clave encriptándola en *'Base64'*[3] y podrá ser añadida al encabezado de las peticiones por parte de nuestro *gateway.*

Cuando se recibe el token **'DLT-Token'** adjunto a una petición por parte de *Wilma Pep-Proxy* se compara con las claves (publica/privada) adquiridas por *ganache-cli* y si estas son correctas se almacena la información relativa a la entidad, junto el *hash*de blockchain generado por la transacción, por último, se almacena el historial de modificaciones de la entidad tratada.

5.1.4. Cloud - Google Sheets y Geomaps

Como última sección a desarrollar en el proceso de instalación y configuración tenemos el lado del cliente o sistema cloud. La Figura 5.15 muestra las herramientas utilizadas para la creación del cloud en este TFM, explicadas previamente en la Sección 4.3.2. Procederemos a explicar el proceso técnico de uso e instalación de la parte cloud.

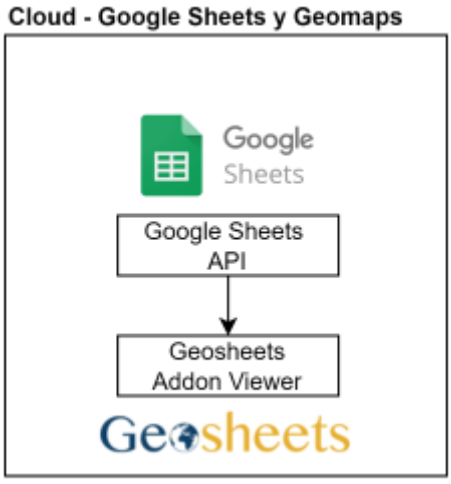

Figura 5.15: Capa cloud - Google Sheets y Geomaps

Script python automatización de recolección de datos

Para simular los requisitos de recolección de datos de un cliente real, se ha utilizado un *script* escrito en el lenguaje de programación *python*. Utilizando la librería *python* denominada **pygsheets** (explicada en la Sección 4.3.2) somos capaces de exportar los datos recibidos por las peticiones *HTTPS* a una hoja de configuración de *Google Sheets*. El apéndice A.2 muestra el script utilizado (resumido).

[3]Sistema de encriptación posicional que utiliza el número 64 como base. Es la mayor potencia que puede ser representada utilizando sólo los caracteres imprimibles ASCII.

Google Sheets

Una vez hemos utilizado la librería *python* para agregar los datos de manera automática al archivo de google sheets, obtendremos los resultados reflejados en la Figura 5.16.

Geosheets

https://www.geosheets.com/map/s:QAQOvrb9/TFM

				People Counter Attributes								Traffic Counter Attributes				
Location	Attribute_Name	Icon	Color	Right_	Left_T	Right_	Left_T	DIFF	TEMP	SBX_E	SBX_F	TEMP	Left_C	Left_A	Right_	Right_
39.4627,-0.3223	People_Counter_1	pitch	blue	1	2	3	4	5	6	7	8					
39.4614,-0.3302	People_Counter_2	pitch	blue	8	7	6	5	4	3	2	1					
39.4602,-0.3321	People_Counter_3	pitch	blue	4	5	6	7	8	9	10	11					
39.4599,-0.3327	Traffic_Counter_1	car	crimson							1	2	3	4	5	6	7
39.4626,-0.3243	Traffic_Counter_2	car	crimson							8	7	6	5	4	3	2
39.4630,-0.3287	Traffic_Counter_3	car	crimson							4	5	6	7	8	9	10
39.4553,-0.3296	Traffic_Counter_4	car	crimson							4	5	6	7	8	9	10
39.4556,-0.3278	Traffic_Counter_5	car	crimson							1	2	3	4	5	6	7
39.4594,-0.3326	Traffic_Counter_6	car	crimson							8	7	6	5	4	3	2
39.4563,-0.3304	Traffic_Counter_7	car	crimson							4	5	6	7	8	9	10
39.4556, -0.3300	Traffic_Counter_8	car	crimson							4	5	6	7	8	9	10
39.4545,-0.3292	Traffic_Counter_9	car	crimson							4	5	6	7	8	9	10
39.4543,-0.3290	Traffic_Counter_10	car	crimson							4	5	6	7	8	9	10

Figura 5.16: Capa cloud - Google sheets

Geosheets

Una vez tenemos todos los datos, agregaremos el *add-on Geosheets* y seleccionando los datos a generar nos aparecerá de manera gráfica una página web con los datos actualizados en tiempo real. La Figura 5.17 muestra un ejemplo de representación final de los datos en tiempo real.

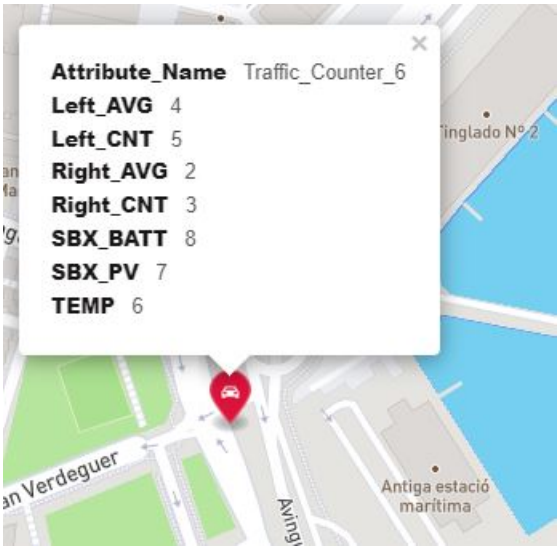

Figura 5.17: Capa cloud - Ejemplo representación visual entidad

5.2 Despliegue y ejecución

A continuación, se realiza el proceso de despliegue y ejecución de la infraestructura de la solución propuesta. Realizaremos un proceso gradual, dividiendo el desarrollo en puntos de actuación.

1. Despliegue sensores y entorno LivingFog

Se considerará desplegado el entorno *LivingFog* una vez se ha instalado y configurado correctamente tal y como se refiere en la Sección 5.1.1.

En la Tabla 5.3 se muestra la información referida a cada dispositivo de sensorización utilizado, el servidor *MQTT* que utiliza y el tópico *MQTT* que utiliza como canal de comunicación individual.

Nº	Nombre Sensor	Pico Cluster	Servidor MQTT & puerto	Tópico MQTT
1	People_counter_1	Cluster 3	192.168.9.30:1883	people_counter/70
2	People_counter_2	Cluster 3	192.168.9.30:1883	people_counter/72
3	People_counter_3	Cluster 1	192.168.9.10:1883	people_counter/81
4	Traffic_counter_1	Cluster 1	192.168.9.10:1883	traffic_counter/60
5	Traffic_counter_2	Cluster 1	192.168.9.10:1883	traffic_counter/61
6	Traffic_counter_3	Cluster 1	192.168.9.10:1883	traffic_counter/62
7	Traffic_counter_4	Cluster 1	192.168.9.10:1883	traffic_counter/63
8	Traffic_counter_5	Cluster 1	192.168.9.10:1883	traffic_counter/64
9	Traffic_counter_6	Cluster 1	192.168.9.10:1883	traffic_counter/65
10	Traffic_counter_7	Cluster 2	192.168.9.20:1883	traffic_counter/66
11	Traffic_counter_8	Cluster 2	192.168.9.20:1883	traffic_counter/67
12	Traffic_counter_9	Cluster 2	192.168.9.20:1883	traffic_counter/68
13	Traffic_counter_10	Cluster 2	192.168.9.20:1883	traffic_counter/69
14	Indoor_env_1	Cluster 3	192.168.9.30:1883	indoor_env/76
14	Outdoor_env_1	Cluster 2	192.168.9.20:1883	outdoor_env/75

Tabla 5.3: Características MQTT dispositivos sensorización

2. Despliegue entorno de desarrollo

Una vez tenemos el entorno LivingFog enviando los datos mediante el protocolo *MQTT* listos para ser procesados y almacenados, procederemos a desplegar el entorno de desarrollo, explicado en la Sección 5.1.3.

En primer lugar, hemos creado dos archivos de configuración **Docker-compose**. *Docker-compose* es una aplicación que ayuda a definir y compartir aplicaciones de varios contenedores. Gracias a *docker-compose* podemos crear un archivo *YAML*[4] para definir servicios, redes privadas de comunicación entre contenedores y realizar un despliegue controlado de los distintos contenedores junto a sus variables de entorno.

El primer archivo de configuración contiene los contenedores necesarios para crear un entorno de desarrollo simulado en *Ethereum* tal y como hemos descrito en la Sección 5.1.3. El Apéndice A.3 muestra el archivo de configuración utilizado utilizando los contenedores *ganache-cli* y *AEI-Contract*.

[4]YAML es un formato de archivos de serialización de datos legible y manipulable por desarrolladores.

El segundo archivo de configuración contiene los 9 contenedores restantes que componen la plataforma *FIWARE* junto a *Canis Major* y los *reverse proxy*. El Apéndice A.4 muestra el archivo de configuración utilizado, junto a la *network* creada. Además, se adjunta en el Apéndice A.5 el archivo de variables utilizadas en ambos archivos de configuración.

Por último, el Apéndice A.6 muestra el *script services.sh* de automatización de ejecución y parada del entorno de desarrollo de manera controlada y visual escrito en *bash script*. A continuación explicamos el proceso de despliegue de la infraestructura:

1. Desplegamos el entorno virtual *Ethereum* y creamos el la dirección del contrato de la cartera virtual.

```
tfmraul@tfmraul:~/Desktop/tfm-raul$ ./services network
Create contract address ETH:
Creating tfm-raul_ganache-cli_1 ... done
Creating tfm-raul_aei-contract_1 ... done
```

Figura 5.18: Despliegue entorno virtual *Ethereum* y dirección del contrato

2. Conseguimos el contrato del contenedor creado *tfm-raul_aei-contract_1*.

```
2_asset_contract.js
===================

   Deploying 'Assets'
   ------------------
   > transaction hash:    0x712f4cede2b9d4d84c957006b2b12614f9c143f8bb33896f982ca1f45bff8abc
- Blocks: 0            Seconds: 0
   > Blocks: 0            Seconds: 0
   > contract address:    0x004B66c27ea5e55525957958b113D362B31cd638
   > block number:        71
   > block timestamp:     1662405773
   > account:             0x3423f4d100f8646aaF6829cE32Cf801996f7007B
   > balance:             98.95065054
   > gas used:            2644377 (0x285999)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.05288754 ETH

- Saving migration to chain.
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.05288754 ETH
```

Figura 5.19: Adquisición dirección del contrato *Ethereum*

3. Conseguimos las claves públicas y privdas simuladas de nuestra cartera *Ethereum* del contenedor creado *tfm-raul_ganache-cli_1*.

```
Available Accounts
==================
(0) 0x3423f4d100f8646aaF6829cE32Cf801996f7007B (100 ETH)
(1) 0x609A10e108B61786E1a1F6Dae2A90694e1ab8Aa3 (100 ETH)
(2) 0x16994785552C97cABfE64c06Bf1611aa5e864Be0 (100 ETH)
(3) 0xa96feB064c1b5AC996D17e8Ec1Ee635B40f131B4 (100 ETH)

Private Keys
==================
(0) 0x6e8f202ae50d774850d0678fb83a730e501ada8d2a6cda5851cdb42b27a4f45b
(1) 0x4410685860d7a40d117bff4f75879cf76a632e2503dd305bb67a48c985412346
(2) 0x8066b3a791a9e920bd4932dea859f4bfc738ec82f5df060521f15caf7eee2749
(3) 0x4b45f2c72b3951653617733a38b97ce4e11578e24f13524433219ae4d377dbed
```

Figura 5.20: Adquisición claves públicas/privadas cartera *Ethereum*

4. Agregamos el contrato conseguido en la variable de entorno *'CONTRACT_ADDRESS'* de la configuración del contenedor *Canis Major* en el archivo de configuración *Docker-compose.yml*

```
environment:
    - NODE_ENV=development
    - CM_PORT=${CANIS_MAJOR_PORT}
    - DB_HOST=mysql-db2
    - DB_PORT=${MYSQL_DB_PORT}
    - DB_NAME=cm
    - DB_USERNAME=root
    - DB_PASSWORD=secret
    - DLT_TYPE=eth
    - RPC_ENDPOINT=http://192.168.1.144:${GANACHE_PORT}
    - DEFAULT_GAS=3000000
    - AEI_CONTRACT_MODE=true
    - CONTRACT_ADDRESS=0x004B66c27ea5e55525957958b113D362B31cd638
    - STORAGE_TYPE=ipfs
    - IPFS_HOST=ipfs.infura.io
    - IPFS_PORT=5001
    - IPFS_PROTOCOL=https
    - IOTAMAM_HOST=https://nodes.devnet.iota.org
    - IOTAMAM_MODE=public
```

Figura 5.21: Agregación dirección contrato a *Canis Major*

5. Una vez tenemos toda la configuración previa realizada, desplegamos el entorno *FIWARE* y los *reverse-proxy*.

```
tfmraul@tfmraul:~/Desktop/tfm-raul$ ./services start
Starting containers:
- Orion is the context broker
- MongoDB is the database
- Node-red is the node-red

Creating db-mysql  ... done
Creating db-mongo          ... done
Creating db-mysql2      ... done
Creating fiware-keyrock    ... done
Creating fiware-canismajor ... done
Creating fiware-orion      ... done
Creating fiware-orion-proxy ... done
Creating rproxy            ... done
Creating rproxy-out        ... done
```

Figura 5.22: Despliegue plataforma *FIWARE, Canis Major y Reverse proxy*

6. (OPCIONAL) Cuando queramos realizar un apagado controlado del entorno creado, tendremos la opción dentro del script de automatización.

```
tfmraul@tfmraul:~/Desktop/tfm-raul$ ./services stop
Stopping containers
7d2feb580a19
e10385614630
04e267df9d8b
f28fd2e4007b
5303b82a3c9a
ea7fc2275cfd
a3b70faf0b56
beeda01adf58
62c30f7d7e7f
145b0fb83976
3997210db0da
Removing old volumes
fcbf5a6d7fb748d3418a92386ff038a1c61e566bc3c67cd4f7a01a0240b92d77
```

Figura 5.23: Apagado controlado entorno de desarrollo

Una vez hemos realizado estos pasos, hemos conseguido desplegar correctamente el entorno de desarrollo, incluyendo la plataforma *FIWARE*, *Canis Major* y *Reverse proxys*, dando por finalizado el despliegue del entorno de desarrollo.

Despliegue Node-RED gateway

Una vez hemos logrado desplegar satisfactoriamente el entorno de desarrollo, habremos obtenido las claves públicas y privadas de la cartera *Ethereum*, que serán esenciales para conseguir el token **'DLT-Token'**. Además, podremos realizar una llamada al servicio activa *Wilma Pep-Proxy* para conseguir el token **'X-Auth-Token'** que nos proporcionará *keyrock*. El desarrollo es el siguiente:

1. Adquirimos una de las claves públicas y privadas conseguidas previamente en los pasos anteriores (Figura 5.20) y la encriptamos en *Base-64*, con este proceso obtenemos el token **'DLT-Token'**.

```
tfmraul@tfmraul:~/Desktop/tfm-raul$ echo -n \
> 0x3423f4d100f8646aaF6829cE32Cf801996f7007B:0x6e8f202ae50d774850d0678fb83a730e501ada8d2a6cda5851cdb42b27a4f45b \
> | base64
MHgzNDIzZjRkMTAwZjg2NDZhYUY2ODI5Y0UzMkNmODAxOTk2ZjcwMDdCOjB4NmU4ZjIwMmFlNTBk
Nzc0ODUwZDA2NzhmYjgzYTczMGU1MDFhZGE4ZDJhNmNkYTU4NTFjZGI0MmIyN2E0ZjQ1Yg==
```

Figura 5.24: Obtención DLT-Token

2. Adquirimos el token **'X-Auth-Token'** gracias a un script en *python* registrado en el Apéndice A.7.

```
tfmraul@tfmraul:~/Desktop/tfm-raul/client$ python3 getToken.py
80e21728af961f0755c9d5cb6f547c74929715d3
```

Figura 5.25: Obtención X-Auth-Token

Una vez conseguidos ambos tokens podemos incluirlos en el payload descrito en el Listing 5.1. Una vez agregados los tokens en todos los payloads correspondientes habríamos desplegamos correctamente el *Gateway Node-RED*.

Despliegue Cloud - Google sheets y Geomaps

Bastaría con ejecutar el *script* por parte del sistema cliente introducido en el Apéndice A.2. En la instalación y configuración quedaría reflejado el resto de infraestructura, vista en la Sección 4.3.2.

5.3 Interfaces de comunicación y uso de la plataforma

5.3.1. Interfaces de comunicación

En toda solución distribuida será necesario contar con interfaces de comunicación para poder interactuar de manera directa con la aplicación utilizada. Cuanto más estándar sea la solución, más sencillo será su utilización y adaptación a su uso.

En esta sección se detallan las distintas interfaces de comunicación que podemos realizar desde el cliente final hacia nuestro entorno *Edge-Cloud Computing* mediante el protocolo de comunicación HTTPS.

Keyrock Identity Manager

Método	Endpoint	Descripción
POST	/oauth2/token	Devuelve un objecto JSON que contiene un token de acceso y un token para regenerar el tiempo de expiración del token.

Tabla 5.4: Interfaces de comunicación Keyrock Identity Manager

Orion Context Broker

Método	Endpoint	Descripción
GET POST DELETE	/entities/	Devuelve/actualiza/elimina un objecto JSON que contiene todas las entidades actualizadas almacenadas en Orion Context Broker
GET POST DELETE	/entities/<id_entity>/	Devuelve/actualiza/elimina un objecto JSON que contiene la información relativa a la entidad determinada.
GET POST	/entities/<id_entity>/ attrs	Devuelve/actualiza un objeto JSON que contiene los atributos almacenados a la entidad determinada.
GET POST	/entities/<id_entity>/ attrs/<attr>	Devuelve/actualiza un objeto JSON que contiene un atributo de un conjunto de atributos almacenados a la entidad determinada.

Tabla 5.5: Interfaces de comunicación Orion Context Broker

Canis Major

Método	Endpoint	Descripción
GET	/entity?entityId= <entity_name>	Devuelve/actualiza/elimina un objecto JSON que contiene todas las transacciones realizadas junto a la información relativa obtenida y almacenada por los bloques blockchain.
GET	/entity/<id_entity>/dlt	Devuelve el último elemento generado de la cadena blockchain de la entidad seleccionada.
GET	/ipfs/<blockchain_data>	Devuelve/actualiza un objeto JSON que contiene la verificación de datos introducidos en blockchain. Verificación de IPFS.

Tabla 5.6: Interfaces de comunicación Canis Major

5.3.2. Uso de la plataforma

En esta sección se muestra, a modo de ejemplo, un conjunto de opciones que posee el cliente final para trabajar con la plataforma a través de sus **REST API endpoints**. Cabe destacar que el cliente no obtendrá el token **'DLT-Token'**, por lo tanto, aunque consiguiera modificar o eliminar datos relativos al estado de alguna entidad, esta operación no quedaría reflejada en la cadena de bloques.

Obtener token de Keyrock Identity Manager

La Figura 5.26 representa la petición *REST API* que establece un cliente final con sus credenciales de acceso a la plataforma que previamente el responsable de la aplicación deberá crear en el gestor de entidades *keyrock*, visto en la Sección 5.1.3. Además, deberá incluir el token de autorización que le proporcionará la plataforma compuesto por una clavé pública y una privada encriptada en el algoritmo criptográfico *base64*.

Este será el primer paso para poder gestionar y realizar las demás peticiones a la plataforma *FIWARE*.

Solicitud

```
curl --location --request POST 'https://192.168.1.144/keyrock/oauth2/token' \
--header 'Authorization: Basic ZWI5MjNiMTItNzA5NS00OWQ5LWJjYzMtZDgwZj \
c0OGQ2MjQ1OjhjYTQ5YTkxLWFjNDUtNDBlYy05ZGYwLWNjZTA0OWQ4YTAxNA' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Accept: application/json' \ \
--data-raw 'username=raul@test.com&password=1234&grant_type=password'
```

Respuesta

```
{
    "access_token": "06c1b83b4ac7f1c65a31afa42ffd10f727dcf636",
    "token_type": "bearer",
    "expires_in": 3599,
    "refresh_token": "1fde800a1d59724ea773c331c7e6f4bb19ce66c1",
    "scope": [
        "bearer"
    ]
}
```

Figura 5.26: Uso de la plataforma - Obtención token IDM

*Access_token generado por el usuario para el token **'X-Auth-Token'**

Obtención de una entidad

La Figura 5.27 representa una petición de obtención de los parámetros registrados de una entidad específica. Un cliente puede realizar esta petición para obtener los valores de manera específica de los dispositivos de sensorización y con ellos manejar su propia base de datos. Es importante haber conseguido el token **'X-Auth-Token** obtenido en el paso anterior.

Solicitud

```
curl --location --request GET
'https://192.168.1.144/orion/v2/entities/indoor_env_1/' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'X-Auth-Token: ae0039f9e5fb7bffa59c617730ca3849628339d2'
```

Respuesta


```
{
    "id": "indoor_env_1",
    "type": "indoor_env",
    "Humidity": {
        "type": "Number",
        "value": 75,
        "metadata": "{}"
    },
    "Humidity_unit": {,
        "type": "Text",
        "value": "%RH",
        "metadata": "{}"
    },
    "Oxygen": {,
        "type": "Number",
        "value": 94,
        "metadata": "{}"
    },
    "Oxygen_unit": {,
        "type": "Text",
        "value": "%",
        "metadata": "{}"
    },
    .
    .
    .
    "Temperature": {,
        "type": "Number",
        "value": 22.3,
        "metadata": "{}"
    },
    "Temperature_unit": {,
        "type": "Text",
        "value": "ºC",
        "metadata": "{}"
    },
    "Pressure": {,
        "type": "Number",
        "value": 1021,
        "metadata": "{}"
    },
    "Pressure_unit": {,
        "type": "Text",
        "value": "mbar",
        "metadata": "{}"
    }
}
```


Figura 5.27: Uso de la plataforma - Obtención de una entidad

Obtención de recibo de transacción de blockchain (CanisMajor)

La Figura 5.28 representa la obtención de los datos de todas las transacciones *blockchain* que ha registrado una entidad en específico. Estos datos incluyen información sobre los parámetros registrados, *hash* de las transacciones, información del contrato de *Ethereum* utilizado y fechas de creación y última modificación.

Además, con esta información es posible obtener el historial de transacciones de cada una de las actualizaciones que se han registrado en la cadena de bloques, esta información es inmutable y provee al cliente de la seguridad de que los valores **no han sido manipulados** por terceros.

Solicitud

```
curl --location --request GET 'https://192.168.1.144/canismajor/entity?
entityId=indoor_env_1'
```

Respuesta


```
{
    "offset": 0,
    "limit": 25,
    "count": 1,
    "records": [
        {
            "id": 2,
            "entityId": "indoor_env_1",
            "txDetails": {
                "to": "0xcd125237903865f39caf6443209c89ba70a4a375",
                "from": "0x3423f4d100f8646aaf6829ce31cf801996f7007b",
                "keys": [
                    "id",
                    "type",
                    "AmbientLight",
                    "AmbientLight_unit",
                    "CO2e",
                    "CO2e_unit",
                    "CarbonDioxide",
                    "CarbonDioxide_unit",
                    "Humidity",
                    "Humidity_unit",
                    "Oxygen",
                    "Oxygen_unit",
                    "PM10.5",
                    "PM10.5_unit",
                    "PM1.0",
                    "PM1.0_unit",
                    "PM2.5",
                    "PM2.5_unit",
                    "PM4.0",
                    "PM4.0_unit",
                    "Pressure",
                    "Pressure_unit",
                    "Temperature_unit",
                    "VolatileOrganicCompounds",
                    "VolatileOrganicCompounds_unit",
                    "bVOC",
                    "bVOC_unit"
                ],
```

```
                "logs": [
                    {
                        "id": "log_7b54619e",
                        "data":
"0x0000000000000000000000003423f4e100f8646aaf6829ce32cf801996f7007b75726e3a6e6773692d6
                        "type": "mined",
                        "topics": [

"0x117ef0a3787baaa508b007db020a6dc877e9f3e78883d885d11e272070e45175"
                        ],
                        "address": "0xcd12523790b865f39caf64553209c89bA70a4A385",
                        "removed": false,
                        "logIndex": 0,
                        "blockHash":
"0x3027007f150f19e239ed335d6035ad0798393f072d94e58c228bb4d1585b28a4",
                        "blockNumber": 6,
                        "transactionHash":
"0xc3bf2b372682f5a2ed4a6b15fde00eaeeccd94e838814dc4a567dcd15807602b",
                        "transactionIndex": 0
                    }
                ],
                "status": true,
                "dltType": "eth",
                "gasUsed": 110000,
                "version": "ngsi-ld",
                "blockHash":
"0x4038207f150f19e239ed325d6035ad0798393f072d94e58c228bb4d1585b28a4",
                "logsBloom":
"0x00000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000
                "objectType": "asset",
                "txSignMode": false,
                "blockNumber": 6,
                "storageType": "ipfs",
                "encyptionMode": false,
                "contractAddress": "0x004B66c27ea5e55525957958b113D362B31cd638",
                "transactionHash":
"0xc4af2b382682f5a2ed4a6b15fde00eaeeccd94e838814dc4a567dcd15807602b",
                "transactionIndex": 0,
                "cumulativeGasUsed": 110100
            },
            "createdAt": "2022-09-06T17:05:28.000Z",
            "updatedAt": "2022-09-06T17:05:28.000Z"
        }
    ]
}
```


Figura 5.28: Uso de la plataforma - Obtención de recibo de transacción de blockchain (CanisMajor)

Obtención de datos de blockchain (Canis Major)

A continuación y en relación con la petición anterior, es posible obtener el *hash* de registro de cada interacción de actualización de una entidad. A través del identificador de cada actualización, la Figura 5.29 muestra la petición realizada para la obtención del *hash* generado por la tecnología distribuida DLT.

Solicitud

```
curl --location --request GET 'https://192.168.1.144/canismajor/entity/2/dlt'
```

Respuesta

```
[
    "cePuB3ZSPgwsSR122qz9PCxSMxnNWxwNqpdyNEnVs6k1Mb1B1"
]
```

Figura 5.29: Uso de la plataforma - Obtención de datos de blockchain (Canis Major)

Verificación de datos de blockchain

Por último, una vez hemos conseguido en el paso anterior el recibo de transacción de *blockchain*, podemos verificar la transacción realizada y registrada en la cadena de bloques *blockchain* de la tecnología *Ethereum*. La Figura 5.30 muestra un ejemplo de verificación de registro agregando el recibo de transacción obtenido previamente.

Solicitud

```
curl --location --request GET
'https://192.168.1.144/canismajor/ipfs/cePuB3ZSPgwsSR122qz9PCxSMxnNWxwNqpdyNEnVs6k1Mb1
```

Respuesta


```
{
    "Humidity": {
        "type": "Number",
        "value": 75,
        "metadata": "{}"
    },
    "Humidity_unit": {,
        "type": "Text",
        "value": "%RH",
        "metadata": "{}"
    },
    "Oxygen": {,
        "type": "Number",
        "value": 94,
        "metadata": "{}"
    },
    "Oxygen_unit": {,
        "type": "Text",
        "value": "%",
        "metadata": "{}"
    },
```

```
    .
    .
    .
    "Temperature": {,
        "type": "Number",
        "value": 22.3,
        "metadata": "{}"
    },
    "Temperature_unit": {,
        "type": "Number",
        "value": "ºC",
        "metadata": "{}"
    },
    "Pressure": {,
        "type": "Number",
        "value": 1021,
        "metadata": "{}"
    },
    "Pressure_unit": {,
        "type": "Text",
        "value": "mbar",
        "metadata": "{}"
    }
}
```


Figura 5.30: Uso de la plataforma - Verificación de datos de blockchain

CAPÍTULO 6
Conclusiones y propuestas

En este capítulo se expondrán las conclusiones obtenidas después de la realización del TFM. Además, se propondrán trabajos futuros relacionados con la temática tratada.

6.1 Conclusiones

En el desarrollo de este TFM se ha evaluado características principales de las tecnologías basadas en el paradigma *Edge-Cloud Computing* desde el punto de vista funcional, seguridad de las comunicaciones y veracidad de los datos transmitidos. Todos los experimentos han sido realizados con objeto de cumplir los objetivos principales de la Sección 1.2 y así poder colaborar, junto a la literatura actual, al análisis y puesta en marcha de un *framework* de seguridad adaptado a las tecnologías *Edge-Cloud Computing*.

En orden, en primer y segundo objetivo tratan sobre el estudio de las características principales y el éxito que acompañan a las tecnologías basadas en el el paradigma emergente de entornos *Edge-Cloud Computing* y el análisis de los sistemas de seguridad asociados. Ya en el Capítulo 2 se habla sobre los antecedentes y el contexto dónde se sitúan estos entornos, generando un punto de inflexión sobre la utilización de tecnologías basadas en el *Internet Of Things (IoT)*. A continuación, en la Sección 2.2 se ha determinado que, entre otros factores, el uso de tecnologías basadas en entornos *Edge-Cloud Computing* empiezan a implementar mecanismos de seguridad basados en el protocolo AAA (autenticación, autorización y contabilización), lo que abre diversas oportunidades de mercado y viabilidad en los entornos actuales. Por último, en el Capítulo 5 se ha evaluado y determinado que determinados entornos basados en *Edge-Cloud Computing* proporcionan soluciones que ofrecen facilidad de instalación, configuración y uso viables para su utilización utilizando tecnologías emergentes como el *blockchain*. En conclusión, las prestaciones obtenidas y la posibilidad de habilitar mecanismos de seguridad añadidos a estas tecnologías hacen que sean una alternativa viable para el mercado y las necesidades tecnológicas actuales.

El tercer objetivo trata sobre la puesta en marcha de un sistema real basado en el paradigma *Edge-Cloud Computing*. Como se ha nombrado anteriormente, en el Capítulo 5 se agrega la implementación y despliegue de una solución propuesta utilizando un sistema basado en dispositivos de virtualización reales, alojados según lo definido en la Sección 4.1.1.

El cuarto y último objetivo habla sobre el análisis y puesta en marcha de implementaciones de seguridad sobre la infraestructura creada. En la Sección 5.1.3 se han identificado diversos mecanismos de seguridad basados en las comunicaciones seguras (protocolos

cifrados), autenticación externa y como novedad en este trabajo, el uso de *blockchain* para asegurar al cliente final la veracidad de los datos recibidos.

Destacar que todos los objetivos han sido sido **elegidos y dedicados en base a la implementación de mecanismos de ciberseguridad y ciberinteligencia**. Por este motivo, este TFM tiene su total enfoque y desarrollo en base a la mejora y producción de un entorno de ciberseguridad (más específicamente en la creación de un *framework* de seguridad) que a la implementación y análisis de un sistema *Edge-Cloud Computing*.

Como conclusión final, en el desarrollo de este trabajo, además de conseguir analizar y ejecutar un entorno *Edge-Cloud Computing* satisfactoriamente, hemos conseguido implementar mecanismos novedosos de seguridad en estos entornos, creando una nueva linea de investigación y mercado dedicado a la seguridad *IoT*. Además, podemos decir sin miedo a equivocarnos, que se ha **integrado la primera solución de un *framework* de seguridad basado en *blockchain* en entornos reales sobre el paradigma *Edge-Cloud Computing***.

6.2 Trabajo futuro

En esta sección, se describen las principales líneas de trabajo futuro en la implementación de *frameworks* de seguridad basados en entornos *Edge-Cloud Computing*.

1. **Evaluación de escalabilidad**. Hasta ahora, hemos comprobado la implementación de sistemas de seguridad agregados a entornos basados en el paradigma *Edge-Cloud Computing*. Se plantea como trabajo futuro un análisis de escalabilidad y viabilidad de esta implementación en sistemas reales que transfieran gran cantidad de datos sobre varios clientes.

2. **Migración entorno desarrollo**. Una de las ventajas que presenta el hecho de trabajar con contenedores es la gran versatilidad y portabilidad que presentan. Es posible cambiar el entorno desarrollado en *docker* a infraestructuras más avanzadas y desarrolladas como pueden ser *kubernetes* o *helm* y tener un repositorio de fácil instalación y despliegue, además de contar con mayores medidas de seguridad.

3. **Evaluación de seguridad**. En el desarrollo de este proyecto, se ha utilizado la tecnología *blockchain* desde un entorno de desarrollo, en una propuesta futura se plantea como trabajo utilizar carteras de criptomonedas reales para realizar un estudio económico de la viabilidad del proyecto.

Bibliografía

[1] Red Hat. "¿Qué es el edge computing?". Página web. https://www.redhat.com/es/topics/edge-computing/what-is-edge-computing. Último acceso, Mayo 2022.

[2] Red Hat. "¿Qué es la nube híbrida?". Página web. https://www.redhat.com/es/topics/cloud-computing/what-is-hybrid-cloud. Último acceso, Mayo 2022.

[3] Universitat Politècnica de València. Sistemas y Aplicaciones en Tiempo Real Distribuido. Página web. https://satrd.es/. Último acceso, Mayo 2022.

[4] Jadeja, Y., Modi, K. Cloud Computing - Concepts, Architecture and Challenges. *International Conference on Computing, Electronics and Electrical Technologies*, 1:4, diciembre, 2012.

[5] Digital Guide IONOS. "¿Qué es un servidor?". Página web. https://www.ionos.es/digitalguide/servidores/know-how/que-es-un-servidor-un-concepto-dos-definiciones/. Último acceso, Mayo 2022.

[6] Dillon, T., Wu, C., Chang, E. Cloud computing: issues and challenges. *IEEE international conference on advanced information networking and applications*, pp. 27-33, junio, 2010.

[7] Mela, J. L., Cedeño, G. D., Herrera, E. C. Edge Computing: Aplicaciones y desafíos actuales. *Visión Antataura*, pp. 75-91, 2021.

[8] FIWARE. FIWARE. Página web. https://www.fiware.org/about-us/. Último acceso, Mayo 2022.

[9] Azure. Guía de protocolos y tecnologías IoT. Página web. https://azure.microsoft.com/es-es/overview/internet-of-things-iot/iot-technology-protocols/. Último acceso, Mayo 2022.

[10] FIWARE-IDM. Identity Manager - Keyrock. Página web. https://fiware-idm.readthedocs.io/en/latest/. Último acceso, Junio 2022.

[11] FIWARE-PEP-PROXY. PEP Proxy - Wilma. Página web. https://fiware-pep-proxy.readthedocs.io/en/latest/. Último acceso, Junio 2022.

[12] Criptonoticias. ¿Qué es una cadena de bloques (blockchain)? Página web. https://www.criptonoticias.com/criptopedia/que-es-una-cadena-de-bloques-block-chain/. Último acceso, Junio 2022.

[13] González Takmovtseva, I. Trabajo Fin de Máster. *Desarrollo de un Marketplace para la Internet de las Cosas en Blockchain*. 2020.

[14] Francia, J. ¿Qué es Scrum? Página web. https://www.scrum.org/resources/blog/que-es-scrum. Último acceso, Junio 2022.

[15] Malathi, M. Cloud computing concepts. *International Conference on Electronics Computer Technology. IEEE*, pp. 236-239, junio, 2011.

[16] Weinhardt, C., Anandasivam, A., Blau, B., Borissov, N., Meinl, T., Michalk, W. Cloud computing–a classification, business models, and research directions. *Business Information Systems Engineering*, pp. 391-399, 2009.

[17] Ammous, S. Blockchain technology: What is it good for? *SSRN 2832751*.. 2018

[18] Tilooby, A. The impact of blockchain technology on financial transactions. *Business Administrations Dissertations*. 2018.

[19] Gong, J., Navimipour, N. J. An in-depth and systematic literature review on the blockchain-based approaches for cloud computing. *Cluster Computing*. pp 1-18. 2021.

[20] Diez Arnal, J. El puerto de Valencia. Página web. http://www.jdiezarnal.com/valenciapuertodevalencia.html. Último acceso, Junio 2022.

[21] LoRaWAN ¿Qué es LoRaWAN? Página web. https://lorawan.es/. Último acceso, Junio 2022.

[22] MQTT Protocolo MQTT. Página web. https://mqtt.org/. Último acceso, Junio 2022.

[23] Computer Weekly. Docker. Página web. https://www.computerweekly.com/es/definicion/Docker. Último acceso, Julio 2022.

[24] Reinosa Simón, R. Trabajo Fin de Grado. Universidad de Castilla-La Mancha (UCLM). *Explorando las prestaciones de Singularity vs Docker*. 2020.

[25] Red Hat. ¿Qué es un clúster de kubernetes? Página web. https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster. Último acceso, Julio 2022.

[26] AlfaIOT. Chirpstack. Página web. https://alfaiot.com/tecnologias-iot/lorawan/chirpstack/. Último acceso, Julio 2022.

[27] Urteaga, JC. Orion Context Broker. Página web. https://cudi.edu.mx/boletin/2017/FIWARE_Taller/README_JCarlosUrteaga.pdf. Último acceso, Julio 2022.

[28] FIWARE. Canis Major. Página web. https://github.com/FIWARE/CanisMajor. Último acceso, Julio 2022.

[29] Nithinmurali. Pygsheets. Página web. https://pygsheets.readthedocs.io/en/stable/. Último acceso, Julio 2022.

[30] Geosheets. Página web. https://www.geosheets.com/. Último acceso, Julio 2022.

[31] Sinelec. ¿Qué es Node-RED y para qué sirve? Página web. https://blog.gruposinelec.com/actualidad/que-es-node-red-y-para-que-sirve/. Último acceso, Agosto 2022.

[32] Preet Singh, H. Fiware Blockchain Pep Proxy. Página web. https://github.com/FIWARE-Blockchain/fiware-pep-proxy. Último acceso, Septiembre 2022.

[33] Ganache. Página web. https://github.com/trufflesuite/ganache/tree/master. Último acceso, Septiembre 2022.

[34] FogGuru. FogGuru Project.Training the Next Generations of Europeans Fog Computings Experts. Página web. http://www.fogguru.eu/livingfog/. Último acceso, Septiembre 2022.

APÉNDICE A
Archivos de configuración

A continuación se adjuntan los archivos de configuración utilizados para el desarrollo del Trabajo Fin de Máster (TFM).

A.1 Archivo de configuración ejemplo creación de reverse proxy

```
#
# Raul Reinosa - TFM
#

proxy_cache_path /var/cache/nginx/liferay_cache levels=1:2 keys_zone=
    liferay_cache:10m inactive=60m max_size=256M;

server {
  listen 443 ssl;
  server_name orion;

  ssl                  on;
  ssl_certificate      /etc/nginx/ssl/orion.crt;
  ssl_certificate_key  /etc/nginx/ssl/orion.key;

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  proxy_connect_timeout 240;
  proxy_send_timeout 600;
  proxy_read_timeout 600;
  send_timeout 600;

  #Liferay URls are too long.
  large_client_header_buffers 4 16k;


  location /orion {
     return 302 /orion/;
  }

  location /orion/ {
    proxy_pass http://orion-proxy:1027/;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }

  location /keyrock {
     return 302 /keyrock/;
```

```
40    }
41
42    location /keyrock/ {
43      proxy_pass http://keyrock:3005/;
44      proxy_set_header X-Forwarded-Host $host;
45      proxy_set_header X-Forwarded-Server $host;
46      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
47    }
48
49    location /canismajor {
50       return 302 /canismajor/;
51    }
52
53    location /canismajor/ {
54      proxy_pass http://canismajor:4000/;
55      proxy_set_header X-Forwarded-Host $host;
56      proxy_set_header X-Forwarded-Server $host;
57      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
58    }
59
60 }
```

Listing A.1: Archivo de configuración reverse proxy cliente

A.2 Script recolección de datos capa cloud

```
1
2  import platform
3  import subprocess
4  import time
5  import json
6  import sys
7  import pygsheets
8  import pandas as pd
9
10 host = "192.168.1.144"
11 user = "raul@test.com"
12 password = "1234"
13
14 entities = "https://"+host+"/orion/v2/entities/"
15 sys.dont_write_bytecode = True
16
17 #authorization
18 gc = pygsheets.authorize(service_file='/home/tfmraul/Desktop/tfm-raul/client/
       creds.json')
19 # Create empty dataframe
20 gen_map = pd.DataFrame()
21 people_counter = pd.DataFrame()
22 traffic_counter = pd.DataFrame()
23 indoor_env = pd.DataFrame()
24 outdoor_env = pd.DataFrame()
25 #open the google spreadsheet (where 'PY to Gsheet Test' is the name of my sheet
       )
26 sh = gc.open('TFM_Geosheets_Map')
27 #select the first sheet
28 wks = sh[0]
29
30 gm = ['close','=GEO_MAP(B5:R21,"TFM")']
31 pc = ['RightToLeft','LeftToRight','RightToLeft_SUM','LeftToRight_SUM','DIFF','
       TEMP','SBX_BATT','SBX_PV']
32 #...
```

```
tf = ['SBX_BATT','SBX_PV','TEMP','Left0_CNT','Left0_AVG','Right0_CNT','
    Right0_AVG']

def ping(host):

    # Returns True if host (str) responds to a ping request.
    # Option for the number of packets as a function of
    param = '-n' if platform.system().lower()=='windows' else '-c'

    # Building the command. Ex: "ping -c 1 google.com"
    command = ['ping', param, '1', host]
    return subprocess.call(command,stdout=subprocess.DEVNULL) == 0

def getToken(host,user,password):
 # Return new token
 token_json = json.loads(subprocess.check_output("curl -s -k -X POST \
  'https://"+host+"/keyrock/oauth2/token' \
  -H 'Accept: application/json' \
  -H 'Authorization: Basic token' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data 'username="+user+"&password="+password+"&grant_type=password'",shell=
      True))
 return token_json["access_token"]

def getData(host,token):
 command = "curl -s https://"+host+"/orion/v2/entities --insecure -H 'X-Auth-
     Token: "+token+"'"
 return subprocess.check_output(command,shell=True)

def exportData(json_payload):
 data = json.loads(json_payload)

 gen_map['gen_map'] = [gm[0]]
 wks.set_dataframe(gen_map,start='B2',copy_head=False)

 # Create a column
 people_counter[pc[0]] = [data[1][pc[0]]['value'],data[2][pc[0]]['value'],data
     [3][pc[0]]['value']]
 people_counter[pc[1]] = [data[1][pc[1]]['value'],data[2][pc[1]]['value'],data
     [3][pc[1]]['value']]
 #...

 #update the first sheet with df, starting at cell B2.
 wks.set_dataframe(people_counter,start='F6',copy_head=False)
 wks.set_dataframe(traffic_counter,start='L9',copy_head=False)
 gen_map['gen_map'] = [gm[1]]
 wks.set_dataframe(gen_map,start='B2',copy_head=False)

token = getToken(host,user,password)
while ping(host) == True:
 json_payload = getData(host,token)
 if('Invalid' not in str(json_payload)):
  exportData(json_payload)
  time.sleep(60)
 else:
  token = getToken(host,user,password)
  time.sleep(5)
```

Listing A.2: Script python automatización de recolección de datos cliente

A.3 Archivo configuración networkprc.yml

```
#
# Raul Reinosa - TFM UPV
#
version: '3.5'
services:
  aei-contract:
        build:
            context: https://github.com/FIWARE-Blockchain/AEIContract.git
            args:
                RPC_ENDPOINT: ganache-cli
                RPC_PORT: ${GANACHE_PORT}
        command: bash -c "truffle migrate"
        depends_on:
            - ganache-cli

  ganache-cli:
        image: trufflesuite/ganache-cli:latest
        ports:
            - "${GANACHE_PORT}:${GANACHE_PORT}"
        expose:
            - "${GANACHE_PORT}"
        command: "ganache-cli \
                --hostname '0.0.0.0' \
                --db=/ganache_data \
                --mnemonic 'minimum symptom minute gloom tragic situate silver
                    mechanic salad amused elite beef' \
                --account 'hash, 100000000000000000000' \
                --account 'hash, 100000000000000000000' \
                --account 'hash, 100000000000000000000' \
                --account 'hash, 100000000000000000000' \
                "
        volumes:
            - ./ganache_data:/ganache_data
```

Listing A.3: Archivo configuración networkprc.yml

A.4 Archivo configuración docker-compose.yml

```
#
# Raul Reinosa - TFM UPV
#
version: "3.5"
services:
   # Orion is the context broker
  orion:
    image: fiware/orion:${ORION_VERSION}
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${ORION_PORT}"
    ports:
      - "${ORION_PORT}:${ORION_PORT}" # localhost:1026
    command: -dbhost mongo-db -logLevel DEBUG
    healthcheck:
      test: curl --fail -s http://orion:${ORION_PORT}/version || exit 1
      interval: 5s

```

```
24    # Database mongo-db
25  mongo-db:
26    image: mongo:${MONGO_DB_VERSION}
27    hostname: mongo-db
28    container_name: db-mongo
29    expose:
30      - "${MONGO_DB_PORT}"
31    ports:
32      - "${MONGO_DB_PORT}:${MONGO_DB_PORT}" # localhost:27017
33    networks:
34      - default
35    volumes:
36      - ./volumes/mongo-db:/data/db
37    healthcheck:
38      test: |
39        host=`hostname --ip-address || echo '127.0.0.1'`;
40        mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1 }).ok ? 0
              : 2)' && echo 0 || echo 1
41      interval: 5s
42
43     # Keyrock is an Identity Management Front-End
44  keyrock:
45    image: fiware/idm:${KEYROCK_VERSION}
46    container_name: fiware-keyrock
47    hostname: keyrock
48    networks:
49      default:
50        ipv4_address: 172.18.1.5
51    depends_on:
52      - mysql-db
53    ports:
54      - "${KEYROCK_PORT}:${KEYROCK_PORT}" # localhost:3005
55      - "${KEYROCK_HTTPS_PORT}:${KEYROCK_HTTPS_PORT}" # localhost:3443
56    environment:
57      - "DEBUG=idm:*"
58      - "IDM_DB_HOST=mysql-db"
59      - "IDM_DB_PASS_FILE=/run/secrets/my_secret_data"
60      - "IDM_DB_USER=root"
61      - "IDM_PORT=${KEYROCK_PORT}"
62      - "IDM_HOST=http://localhost:${KEYROCK_PORT}"
63      - "IDM_HTTPS_ENABLED=${IDM_HTTPS_ENABLED}"
64      - "IDM_HTTPS_PORT=${KEYROCK_HTTPS_PORT}"
65      - "IDM_ADMIN_USER=admin"
66      - "IDM_ADMIN_EMAIL=admin@test.com"
67      - "IDM_ADMIN_PASS=1234"
68      - "IDM_CORS_ENABLED=false"
69    secrets:
70      - my_secret_data
71    healthcheck:
72      interval: 5s
73
74     # Database keyrock
75  mysql-db:
76    restart: always
77    image: mysql:${MYSQL_DB_VERSION}
78    hostname: mysql-db
79    container_name: db-mysql
80    expose:
81      - "${MYSQL_DB_PORT}"
82    ports:
83      - "${MYSQL_DB_PORT}:${MYSQL_DB_PORT}"
84    networks:
85      default:
86        ipv4_address: 172.18.1.6
```

```
    environment:
      - "MYSQL_ROOT_PASSWORD_FILE=/run/secrets/my_secret_data"
      - "MYSQL_ROOT_HOST=172.18.1.5" # Allow Keyrock to access this database
    volumes:
      - ./volumes/mysql-db:/var/lib/mysql
    secrets:
      - my_secret_data

    # PEP Proxy for Orion
  orion-proxy:
    build:
      context: https://github.com/FIWARE-Blockchain/fiware-pep-proxy.git
    container_name: fiware-orion-proxy
    hostname: orion-proxy
    networks:
      default:
        ipv4_address: 172.18.1.10
    depends_on:
      - keyrock
    deploy:
      restart_policy:
        condition: on-failure
    ports:
      - "${ORION_PROXY_PORT}:${ORION_PROXY_PORT}" # localhost:1027
    expose:
      - "${ORION_PROXY_PORT}"
    environment:
      - PEP_PROXY_APP_HOST=orion
      - PEP_PROXY_APP_PORT=${ORION_PORT}
      - PEP_PROXY_PORT=${ORION_PROXY_PORT}
      - PEP_PROXY_IDM_HOST=keyrock
      - PEP_PROXY_HTTPS_ENABLED=false
      - PEP_PROXY_AUTH_ENABLED=false
      - PEP_PROXY_IDM_SSL_ENABLED=false
      - PEP_PROXY_IDM_PORT=${KEYROCK_PORT}
      - PEP_PROXY_APP_ID=eb923b12-7095-49d9-bcc3-d80f748d6245
      - PEP_PROXY_USERNAME=pep_proxy_1a192729-8196-45a2-b27b-8ba9d021aedf
      - PEP_PASSWORD=pep_proxy_2bfef585-8740-4363-931f-275f7af68c27
      - PEP_PROXY_PDP=idm
      - PEP_PROXY_MAGIC_KEY=1234
      - PEP_PROXY_PUBLIC_PATHS=/version
      - CANIS_MAJOR_URL=http://172.18.1.7:4000
    healthcheck:
      interval: 5s

    #Canismajor Eth blockchain
  canismajor:
        build:
          context: https://github.com/FIWARE-Blockchain/CanisMajor.git
        container_name: fiware-canismajor
        hostname: canismajor
        depends_on:
            - mysql-db2
        ports:
            - '${CANIS_MAJOR_PORT}:${CANIS_MAJOR_PORT}'
        networks:
            default:
                ipv4_address: 172.18.1.7
        environment:
            - NODE_ENV=development
            - CM_PORT=${CANIS_MAJOR_PORT}
            - DB_HOST=mysql-db2
            - DB_PORT=${MYSQL_DB_PORT}
            - DB_NAME=cm
```

```
151             - DB_USERNAME=root
152             - DB_PASSWORD=secret
153             - DLT_TYPE=eth
154             - RPC_ENDPOINT=http://192.168.1.144:${GANACHE_PORT}
155             - DEFAULT_GAS=3000000
156             - AEI_CONTRACT_MODE=true
157             - CONTRACT_ADDRESS=0x040CD324D9d286052Ac4A52DBE5096D3D06a52Aa
158             - STORAGE_TYPE=ipfs
159             - IPFS_HOST=ipfs.infura.io
160             - IPFS_PORT=5001
161             - IPFS_PROTOCOL=https
162             - IOTAMAM_HOST=https://nodes.devnet.iota.org
163             - IOTAMAM_MODE=public
164
165       # Database canismajor
166   mysql-db2:
167     restart: always
168     image: mysql:${MYSQL_DB_VERSION}
169     hostname: mysql-db2
170     container_name: db-mysql2
171     expose:
172       - "${MYSQL_DB_CM_PORT}"
173     ports:
174       - "${MYSQL_DB_CM_PORT}:${MYSQL_DB_PORT}"
175     networks:
176       default:
177         ipv4_address: 172.18.1.8
178     environment:
179       - "MYSQL_ROOT_PASSWORD=secret"
180       - "MYSQL_ROOT_HOST=172.18.1.7" # Allow canismajor to access this database
181     volumes:
182       - ./volumes/mysql-db2:/var/lib/mysql
183
184    # Reverse-proxy
185   rproxy:
186     image: rreisim/rproxy
187     container_name: rproxy
188     hostname: rproxy
189     networks:
190       default:
191         ipv4_address: 172.18.1.11
192     depends_on:
193       - orion
194     restart: always
195     ports:
196       - "4430:443"
197     volumes:
198       - ./certs/rproxy/:/etc/nginx/ssl/
199     logging:
200       options:
201         max-size: "2m"
202         max-file: "5"
203
204   # Reverse-proxy-out
205   rproxy-out:
206     image: rreisim/rproxy-out
207     container_name: rproxy-out
208     hostname: rproxy-out
209     networks:
210       default:
211         ipv4_address: 172.18.1.12
212     depends_on:
213       - orion
214     restart: always
```

```
215     ports:
216       - "443:443"
217     volumes:
218       - ./certs/rproxy-out/:/etc/nginx/ssl/
219     logging:
220       options:
221         max-size: "2m"
222         max-file: "5"
223
224 networks:
225   default:
226     ipam:
227       config:
228         - subnet: 172.18.1.0/24
229
230 secrets:
231   my_secret_data:
232     file: ./secrets/secrets.txt
```

Listing A.4: Archivo configuración docker-compose.yml

A.5 Archivo variables archivos de configuración

```
1  # Orion variables
2  ORION_PORT=1026
3  ORION_VERSION=2.5.2
4
5  # MongoDB variables
6  MONGO_DB_PORT=27017
7  MONGO_DB_VERSION=4.2
8
9  # IoT Agent Ultralight Variables
10 ULTRALIGHT_VERSION=1.15.0-distroless
11 IOTA_NORTH_PORT=4041
12 IOTA_SOUTH_PORT=7896
13
14 # Keyrock variables
15 KEYROCK_VERSION=7.8.1
16 KEYROCK_PORT=3005
17 KEYROCK_HTTPS_PORT=3443
18 IDM_HTTPS_ENABLED=false
19
20 # MySQL variables
21 MYSQL_DB_VERSION=5.7
22 MYSQL_DB_PORT=3306
23 MYSQL_DB_CM_PORT=3307
24
25 # PEP Proxy variables
26 WILMA_VERSION=7.9.1-distroless
27 ORION_PROXY_PORT=1027
28
29 # Canis Major variables
30 CANIS_MAJOR_PORT=4000
31
32 # Ganache-cli variables
33 GANACHE_PORT=8545
```

Listing A.5: Archivo variables archivos de configuración

A.6 Archivo configuración services.sh

```
#!/bin/bash
#
# Raul Reinosa - TFM
#
#

set -e

if (( $# < 1 )); then
    echo "Illegal number of parameters"
    echo "usage: services [create|start|stop]"
    exit 1
fi

loadData () {
 docker run --rm -v $(pwd)/import-data:/import-data \
  --network fiware_default \
  --entrypoint /bin/ash curlimages/curl import-data
 waitForOrion
 echo ""
}

stoppingContainers () {
 CONTAINERS=$(docker ps -aq)
 if [[ -n $CONTAINERS ]]; then
  echo "Stopping containers"
  docker rm -f $CONTAINERS
 fi
 VOLUMES=$(docker volume ls -qf dangling=true)
 if [[ -n $VOLUMES ]]; then
  echo "Removing old volumes"
  docker volume rm $VOLUMES
 fi
}

addDatabaseIndex () {
 printf "Adding appropriate \033[1mMongoDB\033[0m indexes for \033[1;34mOrion
     \033[0m  ..."
 docker exec  db-mongo mongo --eval '
 conn = new Mongo();db.createCollection("orion");
 db = conn.getDB("orion");
 db.createCollection("entities");' > /dev/null
 echo -e " \033[1;32mdone\033[0m"
}

waitForMongo () {
 echo -e "\ n   Waiting for \033[1mMongoDB\033[0m to be available\n"
 while ! [ `docker inspect --format='{{.State.Health.Status}}' db-mongo` == "
     healthy" ]
 do
  sleep 1
 done
}

waitForOrion () {
 echo -e "\ n   Waiting for \033[1;34mOrion\033[0m to be available\n"

 while ! [ `docker inspect --format='{{.State.Health.Status}}' fiware-orion` ==
      "healthy" ]
 do
   echo -e "Context Broker HTTP state: " `curl -s -o /dev/null -w %{http_code}
       'http://localhost:1026/version'` " (waiting for 200)"
```

```
    sleep 1
 done
}

waitForKeyrock () {
 echo -e "    Waiting for \033[1;31mKeyrock\033[0m to be available\n"

 while [ 'curl -s -o /dev/null -w %{http_code} 'http://localhost:3005/version'' -eq 000 ]
 do
  echo -e "Keyrock HTTP state: " 'curl -s -o /dev/null -w %{http_code} 'http://localhost:3005/version'' " (waiting for 200)"
  sleep 5
 done
 echo -e " \033[1;32mdone\033[0m"
}

displayServices () {
 echo ""
 docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}" --filter name=fiware-*
 echo ""
}

command="$1"
case "${command}" in
 "help")
        echo "usage: services [create|start|stop|network]"
        ;;
 "start")
  export $(cat .env | grep "#" -v)
  echo -e "Starting containers:"
  echo -e "- \033[1;34mOrion\033[0m is the context broker"
  echo -e "- \033[1mMongoDB\033[0m is the database"
  echo -e "- \033[1mNode-red\033[0m is the node-red"
  echo ""
  docker-compose --log-level ERROR -p fiware up -d --remove-orphans
  waitForMongo
  addDatabaseIndex
  waitForOrion
  loadData
  waitForKeyrock
  displayServices
  ;;
 "stop")
  export $(cat .env | grep "#" -v)
  stoppingContainers
  ;;
 "create")
  export $(cat .env | grep "#" -v)
  echo "Pulling Docker images"
  docker pull curlimages/curl
  docker-compose --log-level ERROR -p fiware up -d --remove-orphans
  ;;
 "network")
  export $(cat .env | grep "#" -v)
  echo -e "Create contract address ETH:"
  docker-compose --log-level ERROR -f networkrpc.yml up -d --remove-orphans
  ;;
 *)
  echo "Command not Found."
  echo "usage: services [create|start|stop]"
  exit 127;
  ;;
```

```
esac
```

Listing A.6: Archivo configuración services.sh

A.7 Script getToken.py

```
import subprocess
import json
import sys

sys.dont_write_bytecode = True
token = json.loads(subprocess.check_output("curl -s -k -X POST \
'https://192.168.1.144/keyrock/oauth2/token' \
-H 'Accept: application/json' \
-H 'Authorization: Basic ZWI5MjNiMTItNzA5NS00OWQ5LWJjYzMtZDgwZjc0 \
                         OGQ2MjQ1OjhjYTQ5YTkxLWFjNDUtNDBlYy05ZGYw \
                         LWNjZTA0OWQ4YTAxNA==' \
-H 'Content-Type: application/x-www-form-urlencoded' \
--data 'username=raul@test.com&password=1234&grant_type=password'",shell=True))

print(token["access_token"])
```

Listing A.7: Script getToken.py

APÉNDICE B
Objetivos de desarrollo sostenible

B.1 Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.			X	
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.		X		
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.			X	
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

B.2 Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El proyecto no está relacionado intrínsecamente con el contexto de los objetivos de desarrollo sostenible. De igual manera, se ha logrado relacionarlo con el objetivo de "ciudades y comunidades sostenibles" y, en menor medida, con "Industria, innovación e infraestructuras" y "Acción por el clima". El objetivo de este Trabajo de Fin de Máster (TFM) ha sido la implementación de un framework de seguridad en entornos Edge-Cloud Computing. Uno de los ejemplos más útiles de aplicación de estos entornos es en las denominadas "ciudades inteligentes". Gracias a la implementación de este modelo de entornos, es posible solucionar con rapidez los problemas que comprometen a sostenibilidad del medioambiente. Algunos ejemplos incluyen la optimización de gestión del tráfico, gestión del transporte, reparación de baches y recolección de residuos en tiempo y forma, optimización del alumbrado público, la identificación de vertidos químicos o gases. Para que este proyecto sea viable, necesitamos poder verificar estos datos y asegurarnos que no son modificados o manipulados por terceros y así poder actuar con cierta seguridad. La seguridad y veracidad de los datos nos ofrece una ventaja en la aplicación de dispositivos de prevención y actuación, si varios datos fueran corrompidos o manipulados, se podrían utilizar recursos innecesarios y tomar medidas de malgasto de servicios y elementos básicos, dando lugar a un escenario contraproducente al esperado. Directamente ligado al objetivo anterior, tenemos el objetivo denominado "acción por el clima" el cual no está estrechamente ligado con el pensamiento inicial del trabajo pero si con el objetivo de "ciudades y comunidades sostenibles". Esto se debe a que obteniendo información acerca del flujo del tráfico, uso de las carreteras y servicios públicos, será posible optimizar estos procesos y generar menor contaminación "innecesaria". Por último, el objetivo de "industria, innovación e infraestructuras" se encuentra ligeramente relacionado al estar este trabajo de fin de máster ligado al análisis y futura producción de proyectos dedicados a la industria e infraestructuras de manera segura y eficaz. Al utilizar blockchain como referente a las transacciones, nos ubicaremos en un entorno seguro y listo para poder ejecutar procesos industriales de manera eficiente y fiable.