

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS  
INDUSTRIALES Y DE TELECOMUNICACIÓN**

**UNIVERSIDAD DE CANTABRIA**



*Trabajo Fin de Grado*

**IMPLEMENTACIÓN DE UNA PLATAFORMA  
DE ANÁLISIS Y VISUALIZACIÓN DE DATOS  
IOT MULTI-PROTOCOLO EN TIEMPO REAL  
SOBRE NODE-RED**

*Implementation of a Platform for Analysis and Visualization  
of IoT Multi-Protocol Data in Real Time on Node-Red*

Para acceder al Título de  
**Graduado en Ingeniería de Tecnologías de Telecomunicación**

**Autor:** Eder Ollora Zaballa

**Septiembre 2021**



# Agradecimientos

Quisiera que este momento que simboliza la finalización de los estudios de Ingeniería de Tecnologías de Telecomunicación y la entrega de este TFG en la Universidad de Cantabria esté dedicada, principalmente, a mi tía y mi tío: Monse Ollora y Pruden Aja. Mis tíos han supuesto un soporte familiar y económico para mi hermano y para mí desde nuestra infancia. Este apoyo es algo que hoy día se ve materializado en la culminación de unos estudios universitarios. Además, parte de mi infancia ha estado ligada a Cantabria, a Riva de Ruesga, uniendo mi vida a este territorio. Por estas razones, tiene sentido que el principal agradecimiento en el TFG de esta titulación cursada en Santander tenga a ambos como mi mayor referencia. Mi hermano y yo siempre estaremos agradecidos por su ayuda.

También querría agradecer el apoyo de mi familia, a mi hermano, a mis amamas, tíos y primos de Navarra pero sobre todo, a mis padres. Es complicado explicar cuán importantes han sido en mi educación. Además, este momento nunca hubiese ocurrido sin Antonia y Juanjo. Es complicado resumir lo que ambos han supuesto para nuestra familia, la que hemos creado junto a ellos. Han sido soporte fundamental y todo agradecimiento será poco para ellos.

Hablar de mi experiencia en Santander no tendría sentido si no menciono a Irene Zamora. Es ella, probablemente, una de las razones por las que haber ido a Santander mereciese la pena. Dankeschön. Du bist eine bessere Lehrerin als ich ein Schüler bin.

No me olvido de las personas que conocí mientras viví en Santander, como Jonathan Cárdenas y David Valero. Fue una experiencia muy buena compartir con ellos

---

y me agrada que, aún con la distancia, sigan siendo parte de alegres charlas de vez en cuando. También guardo muy buenos recuerdos de mis compañeros allí como Nerea, Martín, Cristina y Jacobo.

Para terminar, gracias a Luis Sánchez y a Pablo Sotres, supervisores de este trabajo fin de grado. En especial, agradezco a Luis que haya aceptado ser el supervisor y proponerme desarrollar este tema para completar el grado, incluso con un relativamente tan ajustado.

# Resumen

Los retos que presentan las tecnologías de la Internet de las Cosas (IoT) se incrementan según se incorporan nuevas tecnologías a un ecosistema tremendamente heterogéneo y fragmentado. Cada vez más, los desarrolladores de soluciones deben tratar no solo con la propia lógica de negocio de sus servicios inteligentes basados en la capacidad de medida y monitorización de fenómenos, sino que deben tener una comprensión multidisciplinar dentro del ámbito de la IoT. Desarrollar un sistema extremo-a-extremo donde toman parte sensores, redes inalámbricas, pasarelas (*gateway*), servidores y la monitorización es extremadamente complejo. Esto hace necesario plantear un sistema fácil y de rápido despliegue para recibir la información de los sensores, almacenarla y visualizarla.

Por lo tanto, este Trabajo de Fin de Grado (TFG) pretende desarrollar un sistema de programación visual del lado del servidor que reciba la información de sensores y plataformas IoT heterogéneas, la almacene en una base de datos y la visualice. En particular, se utilizarán diferentes fuentes de datos que provienen de sensores que utilizan cuatro plataformas de gestión de la información IoT distintas. Estos datos se enviarán sobre diferentes protocolos de aplicación y el servidor desplegado usando Node-RED recibirá todos los datos. También se almacenarán los datos en InfluxDB a corto o medio plazo, y se utilizará MongoDB como base de datos histórica, para después mostrarse en Grafana usando tableros dinámicos.

Los conceptos anteriormente mencionados se validan comprobando el transporte de datos extremo-a-extremo, así como se comprueba el almacenamiento a corto y largo plazo y su visualización dinámica. Se confirma, por lo tanto, que es posible desplegar

---

una red completa de aplicaciones basadas en contenedores en pocos segundos, para ofrecer un entorno de programación visual así como almacenamiento y visualización.

# Abstract

The more Internet-of-Things (IoT) technologies advance, the more considerable challenges appear. Current IoT developers do not limit themselves to develop a tiny part of sensor and phenomena monitoring services. It is necessary to incorporate a multidisciplinary understanding of all IoT areas. It is challenging to develop an end-to-end system that incorporates sensors, wireless networks, gateways, servers, and a visualization scheme. This fact requires new and easy deployment techniques in order to collect, store and visualize sensor information.

Therefore, this thesis pretends to develop and implement a visual programming system that receives, stores and visualizes measurements directly sent from sensors or proxies. In particular, different data sources will be used, which belong to sensors that employ four different IoT data management platforms to handle their observations. The data generated by these sensors will ultimately be sent using different application layer protocols. Still, the server deployed using Node-RED will be able to process it. Data will also be stored in an InfluxDB database for the short or medium term, while MongoDB is used as a historical database. Grafana later queries the data stored at InfluxDB to show the information received by the server using dynamic dashboards at runtime.

The concepts aforementioned will eventually be validated, verifying the end-to-end data transport and checking that data is appropriately stored in short and long-term databases, ultimately using dynamic means of visualization. The thesis demonstrates that it is possible to deploy a complete container-based scenario in a few seconds to offer a visual programming environment, storage, and visualization capabilities.

# Contenido

<b>Agradecimientos</b>	<b>II</b>
<b>Resumen</b>	<b>IV</b>
<b>Abstract</b>	<b>VI</b>
<b>1 Introducción</b>	<b>1</b>
1.1 Contexto . . . . .	3
1.2 Objetivos . . . . .	4
1.3 Metodología . . . . .	6
1.4 Planteamiento y resumen ejecutivo . . . . .	7
<b>2 Fundamentos tecnológicos</b>	<b>9</b>
2.1 Redes de la Internet de las cosas (IoT) . . . . .	9
2.1.1 Tecnologías de redes de baja potencia y área amplia (LPWAN)	9
2.2 Protocolos de comunicación . . . . .	13
2.2.1 Message Queuing Telemetry Transport (MQTT) . . . . .	13
2.2.2 Constrained Application Protocol (CoAP) . . . . .	15
2.2.3 Advanced Message Queuing Protocol (AMQP) . . . . .	16
2.2.4 Hypertext Transfer Protocol (HTTP) . . . . .	17
2.3 Plataformas y lenguajes de programación visual . . . . .	19
2.3.1 Blockly . . . . .	19
2.3.2 Scratch . . . . .	21
2.3.3 Node-RED . . . . .	22
2.4 Almacenamiento de datos . . . . .	22
2.4.1 Bases de datos relacionales o SQL . . . . .	23
2.4.2 Base de datos de siguiente generación . . . . .	24
2.5 Modelado de datos y contextualización . . . . .	26
2.5.1 Next Generation Service Interface (NGSI) . . . . .	26
<b>3 Diseño y desarrollo</b>	<b>29</b>
3.1 Despliegue de la plataforma . . . . .	29
3.1.1 Entorno y contenerización . . . . .	29
3.1.2 Despliegue . . . . .	31



---

3.2	Diseño e implementación de los casos de uso . . . . .	34
3.2.1	Caso 1: <i>hook</i> HTTP para SmartSantander . . . . .	34
3.2.2	Caso 2: hook HTTP para el Orion Context Broker . . . . .	47
3.2.3	Caso 3: cliente MQTT para Lora . . . . .	54
3.2.4	Caso 4: cliente activo de HTTP (polling) . . . . .	59
<b>4</b>	<b>Validación</b>	<b>67</b>
4.1	Revisión de despliegue . . . . .	67
4.2	Adaptación de modelos de información . . . . .	68
4.3	Verificación extremo-a-extremo de la información . . . . .	70
4.4	Visualización . . . . .	73
4.5	Almacenamiento a largo plazo . . . . .	77
<b>5</b>	<b>Discusión</b>	<b>80</b>
<b>6</b>	<b>Conclusiones</b>	<b>83</b>

# Índice de Figuras

Figura 1.1	Visión a alto nivel de los pasos principales en la metodología.	6
Figura 1.2	Planteamiento inicial de la estructura del proyecto. . . . .	7
Figura 1.3	Planteamiento final de la estructura del proyecto. . . . .	8
Figura 2.1	Mapa de la cobertura mundial de Sigfox . . . . .	11
Figura 2.2	Mapa de la cobertura mundial de LoRaWAN . . . . .	11
Figura 2.3	Ejemplo de despliegue IoT con la red de LoRa . . . . .	11
Figura 2.4	Modos de despliegue para NB-IoT . . . . .	13
Figura 2.5	Arquitectura y organización de una red MQTT. . . . .	14
Figura 2.6	Ejemplo del uso del protocolo CoAP . . . . .	15
Figura 2.7	Ejemplo de arquitectura sobre la que funciona AMQP. . . . .	16
Figura 2.8	Ejemplo de un caso de uso con clientes y servidores HTTP . . . . .	18
Figura 2.9	Interfaz de usuario en Blockly. . . . .	20
Figura 2.10	Caso de uso con Scratch. . . . .	21
Figura 2.11	Ejemplos de flujos Node-RED . . . . .	23
Figura 2.12	Ejemplo para visualizar un diagrama de entidad-relación . . . . .	24
Figura 2.13	Representación estructural de las bases de datos NoSQL. . . . .	25
Figura 2.14	Estructura de NGSiv2. . . . .	27
Figura 2.15	Equivalencia de modelo y formato NGSiv2. . . . .	28
Figura 3.1	Representación completa de la distribución de contenedores. . . . .	34
Figura 3.2	Representación de todas las entidades relevantes en el Caso 1. . . . .	35
Figura 3.3	Visión general de la interfaz de usuario para el Caso 1. . . . .	36
Figura 3.4	Visión de todos los fenómenos recibidos de SmartSantander. . . . .	36
Figura 3.5	Datos introducidos en el Caso 1 para enviar una subscripción. . . . .	37
Figura 3.6	Listado de todas las subscripciones mostrando también la última creadas. . . . .	38
Figura 3.7	Demostración visual de una subscripción no activa. . . . .	38
Figura 3.8	Demostración de una subscripción activa después de enviar una petición de activación. . . . .	39
Figura 3.9	Contador de mensajes recibidos para el Caso 1. . . . .	39
Figura 3.10	Configuración de la localización y radio de filtrado para la subscripción. . . . .	40
Figura 3.11	Flujo de inicio para el caso y recolección del listado de fenómenos. . . . .	41

Figura 3.12	Petición de las unidades del fenómeno elegido y definición de variables globales. . . . .	41
Figura 3.13	Bloques del envío de la petición a SmartSantander. . . . .	41
Figura 3.14	Automatización del proceso de actualización del listado de suscripciones. . . . .	42
Figura 3.15	Flujo para la comprobación manual o automática del estado de la suscripción. . . . .	43
Figura 3.16	Flujo para habilitar la suscripción a SmartSantander. . . . .	43
Figura 3.17	Flujo de Node-RED para eliminar la suscripción seleccionada de la lista desplegable. . . . .	43
Figura 3.18	Mapa de la infraestructura IoT de SmartSantander. . . . .	44
Figura 3.19	Bloque de recepción y adaptación de datos para el Caso 1. . . . .	45
Figura 3.20	Parámetros de configuración para InfluxDB en el Caso 1. . . . .	47
Figura 3.21	Parámetros de configuración de una nueva suscripción para Orion. . . . .	48
Figura 3.22	Listado de suscripciones enviadas por el servidor de contexto Orion. . . . .	49
Figura 3.23	Nodos en Node-RED para suscribirse en Orion. . . . .	49
Figura 3.24	Flujo que actualiza la lista de suscripciones de la entidad que envíe la petición. . . . .	50
Figura 3.25	Flujo que incluye un inyector o botón que desencadena la eliminación de una suscripción. . . . .	50
Figura 3.26	Captura de las colecciones que Orion crea en la base de datos MongoDB. . . . .	52
Figura 3.27	Atributos de las entidades insertados por Orion en la base de datos MongoDB. . . . .	52
Figura 3.28	Flujo de nodos que reciben mensaje de Orion, adaptan el payload y lo almacenan en InfluxDB. . . . .	54
Figura 3.29	Tabla de mensajes MQTT recibidos por cada ID y tema en el Caso 3. . . . .	56
Figura 3.30	Nodos que cuentan los mensajes recibidos y los muestran en la UI. . . . .	56
Figura 3.31	Sensor emulado que envía datos usando MQTT. . . . .	57
Figura 3.32	Consumidor MQTT suscrito a broker de los sensores LoRa. . . . .	58
Figura 3.33	Interfaz de usuario completa del Caso 4. . . . .	60
Figura 3.34	Nodo contador de mensajes y bloque UI de la fuente de datos (1). . . . .	61
Figura 3.35	Nodos que hacen una petición de los sensores en Aarhus para la fuente (2). . . . .	61
Figura 3.36	Listado de todos los sensores de tráfico en Aarhus como parte de la fuente (2). . . . .	62
Figura 3.37	Nodos que adquieren los parámetros del formulario, detienen las consultas y cuentan los mensajes recibidos. . . . .	62

Figura 3.38	Cliente polling para la fuente (1) que adquiere datos del repositorio abierto de SmartSantander. . . . .	64
Figura 3.39	Cliente polling que adquiere datos de tráfico del servidor de Aarhus en la fuente (2). . . . .	64
Figura 3.40	Cliente polling para OpenWeatherMap que adquiere datos meteorológicos desde la fuente (3). . . . .	65
Figura 4.1	Listado de contenedores activos y funcionando mostrados por Portainer (InfluxDB_CLI funciona una vez y se detiene solo). . . . .	68
Figura 4.2	Parámetros seleccionables en InfluxDB en el Caso 1. . . . .	71
Figura 4.3	Parámetros seleccionables en InfluxDB en el Caso 2. . . . .	71
Figura 4.4	Parámetros seleccionables en InfluxDB en el Caso 3. . . . .	72
Figura 4.5	Demostración en tabla de los parámetros seleccionados en la Figura 4.4. . . . .	72
Figura 4.6	Parámetros seleccionables en InfluxDB en el Caso 4. . . . .	72
Figura 4.7	Demostración en tabla de los parámetros seleccionados en la Figura 4.6. . . . .	72
Figura 4.8	Demostración gráfica de la medida, etiquetas y campo seleccionados en la Figura 4.6. . . . .	73
Figura 4.9	Consulta Flux en Grafana para adquirir todos los buckets almacenados en InfluxDB. . . . .	74
Figura 4.10	Gráfico de dependencias de las variables en el tablero de Grafana. . . . .	74
Figura 4.11	Listado de variables seleccionadas en el tablero de Grafana. . . . .	75
Figura 4.12	Se muestran los datos de temperatura para SmartSantander (Caso 1). . . . .	75
Figura 4.13	Se muestran los datos de temperatura del sensor emulado desde Orion (Caso 2). . . . .	76
Figura 4.14	Se muestran los datos de luminancia de un sensor LoRa que envía la información usando MQTT (Caso 3). . . . .	76
Figura 4.15	Se muestran los datos de humedad en Ámsterdam por OpenWeatherMap (Caso 4). . . . .	77
Figura 4.16	Diagrama conceptual y UI para el almacenamiento de datos a largo plazo. . . . .	78

# Índice de Listados

Listado 3.1	Configuración YAML para Docker compose . . . . .	33
Listado 3.2	Mensaje del nodo “ADAPTADOR” a la base de datos InfluxDB	46
Listado 3.3	Payload del mensaje POST enviado a Orion. . . . .	51
Listado 3.4	Payload del mensaje PATCH enviado a Orion. . . . .	51
Listado 3.5	Formato JSON de la información almacenada en MongoDB .	53
Listado 3.6	Mensaje JSON que se recibe en el servidor HTTP del Caso 2.	55
Listado 3.7	Transformación de la estructura del mensaje para InfluxDB .	55
Listado 3.8	Función JavaScript que crea un mensaje para InfluxDB . . .	59
Listado 4.1	Información de OpenWeatherMap usando el modelo NGSIv2 .	79

# Capítulo 1

## Introducción

La investigación, desarrollo y comercialización de dispositivos y redes de la Internet de las Cosas o *Internet-of-Things* (IoT), ha supuesto un marco para una nueva generación de dispositivos que intercambian información a través de redes de acceso e internet. Las aplicaciones prácticas y casos de uso donde la IoT es relevante, son muy variados [1]. Por ejemplo, los Entornos Inteligentes (*Smart Environments*) son uno de los campos donde las tecnologías de la IoT son aplicables. Casos donde se instalan sensores de detección de incendios en bosques, sensores para detectar el nivel de nieve en montaña, sensores de polución, etc. Por otro lado, IoT también tiene casos de usos en el ámbito de la medición en la Red Eléctrica Inteligente (REI o *Smart Grid*). En los casos de medición inteligente, los sensores se usan para medir y monitorizar el consumo de energía eléctrica. Son particularmente útiles para detectar el uso ilegal de los recursos de energía eléctrica por terceras partes.

Otro de los casos de uso principales de la IoT son las Viviendas Inteligentes (*Smart Homes*). La función esencial es gestionar y administrar los servicios de la vivienda. Funciones como automatización de persianas, luces y electrodomésticos son las primordiales en las Viviendas Inteligentes. Como extensión al concepto de Viviendas Inteligentes, también existen los Edificios Inteligentes (*Smart Buildings*), que se caracterizan por usar sensores y dispositivos que automatizan funciones relacionadas con la seguridad del inmueble o mantenimiento de recursos compartidos. Existen, además, Sistemas de Transporte Inteligente (STI, o *Intelligent Transportation Systems*) que se caracterizan por usar sensores y dispositivos para mejorar la eficiencia

y la optimización. Es así en sistemas como la conducción autónoma donde se mide la proximidad con otros vehículos, Sistemas de Posicionamiento Global (GPS) para la localización, Lidar para medir distancia de una forma precisa y protocolos de comunicación basados en el concepto Vehículo-a-Vehículo (*Vehicle-to-Vehicle*) o Vehículo-a-Red (*Vehicle-to-Network*), hacen posible que ocurra.

La automatización de la industria o Industria 4.0 es otro de los sectores donde la IoT está logrando una relevancia muy importante. El hecho de que la mayoría de los modernos procesos industriales necesiten de una automatización de la robotización, implica que los últimos avances en tecnologías IoT también necesitan implementarse. Por ejemplo en este sector es normal necesitar protocolos de sincronización (e.g. *Precision Time Protocol* o PTP) que proveen una sincronización del término del nanosegundo. Para proporcionar una sincronización tan precisa y poder gestionar procesos industriales de una forma automatizada, la IoT juega un papel crucial.

Sin ser menos importante que otros sectores, la Medicina Inteligente o *Smart Healthcare* tiene una importancia significativa. En los últimos años se han transferido ciertos procesos de medicina a las casas de los pacientes, como la monitorización y seguimiento de ciertas enfermedades o procesos de recuperación. Esto significa que los pacientes necesitan las últimas tecnologías para comunicarse con los servicios de medicina y poder monitorizar en tiempo real cuáles son las constantes vitales de los pacientes. En estos casos es importante que la calidad en los servicios de comunicación de voz y vídeo, así como la transmisión de datos por redes inalámbricas sea de una calidad excelente.

Por último, es importante analizar uno de los casos principales para la IoT, como son las ciudades inteligentes. Este caso es importantes para este Trabajo de Fin de Grado (referido como TFG o trabajo), ya que gran parte de los datos que se adquieren en la implementación y desarrollo del sistema se consiguen mediante sensores que se

han desplegado como parte de la red de una ciudad inteligente.

Esta introducción sirve para enmarcar los casos de uso más típicos para la IoT. Puede entenderse que tantos casos de uso con tantos protocolos y tecnologías implica un conocimiento muy profundo de múltiples áreas. Esto no solo significa un reto para desarrolladores actuales, sino también para presentes y futuros despliegues de tecnologías IoT. Sin embargo, la complejidad de los diferentes aspectos de la IoT implica que se necesitan explicaciones mucho más extensas para abarcar la arquitectura, diseño y protocolos más relevantes. Por eso, la siguiente secciones servirán como introducción a los temas más importantes en la IoT como las tecnologías inalámbricas, protocolos, modelos de datos etcétera.

Sin embargo, y antes de proceder con las explicaciones más técnicas sobre la IoT, es necesario comprender varios aspectos de este trabajo como la contextualización, los objetivos, la metodología y el planteamiento y resumen jerárquico.

## 1.1. Contexto

Teniendo en cuenta la introducción de este trabajo, el lector puede entender que existen varios problemas que se deben abordar. Por ejemplo, la gran heterogeneidad de tecnologías inalámbricas, protocolos, modelos de datos y tecnologías de aplicaciones en el mundo de la IoT. Este trabajo se enfoca, sobre todo, en el problema de desplegar una red que sea capaz de recibir, guardar y visualizar información de varios sensores o dispositivos. Para poder desplegar servicios que sean capaces de comunicarse con sistemas intermedios para la gestión de la información, es necesario facilitar el despliegue de todos estos sistemas del lado del servidor.

Como ya se ha mencionado anteriormente, existe un gran problema a la hora de



delegar las funciones de desarrollo, diseño e implementación a un mismo desarrollador. Desplegar todas estas funciones implica un conocimiento muy extenso de todas las tecnologías que toman parte en las redes IoT. Para poder desplegar servicios que sean capaces de comunicarse con sistemas intermedios de la IoT de una manera ágil, eficiente y amigable, es más intuitivo y recomendable utilizar un sistema de programación visual que sea capaz de ofrecer posibilidades más amplias y automatizadas para estudiantes y desarrolladores.

## 1.2. Objetivos

El proyecto que se expondrá en este TFG se centrará en desarrollar una plataforma para integrar los servicios del lado del servidor (*backend*), para la recepción y análisis de información. Se trata de especificar mediante abstracciones visuales de programación, un diagrama que represente la funcionalidad de un servidor encargado de recibir información, que podría venir de sensores o sistemas intermedios de la IoT.

La plataforma también pretende describir mediante la programación visual, la interconexión de entidades que elaboran funciones de servidor. Interpretando sus conexiones, se podrán entonces traducir a un código que se ejecuta en el servidor. Mediante estas abstracciones visuales, este proyecto pretende crear entidades que manejen diferentes protocolos de la capa de aplicación al recibir información de entidades externas como: *Message Queuing Telemetry Transport* (MQTT), *Hypertext Transfer Protocol Secure* (HTTP, HTTPS), arquitecturas de tipo *Representational State Transfer* (REST) que también usa, generalmente, HTTP como protocolo de aplicación.

Adicionalmente, el sistema debe ser capaz de proporcionar un procedimiento para el almacenamiento y análisis de datos usando una base de datos de series temporales y una interfaz de visualización de los datos en tiempo real. Además, el sistema de

almacenamiento debe integrar un procedimiento para reducir el dimensionamiento de los datos almacenados en la base datos principal.

De esta forma, el proyecto ofrece dos vertientes de aplicabilidad. La primera, una interfaz de aprendizaje ágil para programadores sin experiencia en los servicios del lado del servidor, de manera que la docencia sería la primera área de aplicación. Asimismo, esta plataforma también ofrece una interfaz para un despliegue rápido de los servicios del lado del servidor, abstrayendo la programación y configuración subyacente.

El objetivo principal de este trabajo es desarrollar una plataforma de análisis y visualización de datos que se desglosa en los siguientes objetivos específicos:

- Se analizarán diversas fuentes de datos a lo largo del TFG que impliquen la heterogeneidad imperante en la IoT y se procederá a su integración utilizando el modelo de datos definido por *Next Generation Service Interface* versión 2 (NGSIv2) para homogeneizar los respectivos modelos de información de cada una de las cuatro fuentes utilizadas.
- Se empleará un modelo de contenerización que permita despliegue y replicación fluido de la plataforma desarrollada en el TFG.
- Definición, localización e interconexión de las entidades visuales que representan las instancias relevantes en el entorno del lado del servidor.
  - Identificar y configurar (o en su defecto desarrollar) parámetros de cada una de las entidades del entorno de programación visual.
- Habida cuenta de que la validación del sistema implementado en el TFG se hará empleando para ello fuentes de datos reales, se pretende desplegar un entorno completo en el cual integrar las diferentes fuentes de datos identificadas al inicio del TFG.

- Se llevará a cabo la implementación de una serie de interfaces visuales de tipo *Dashboard* que sirvan como prueba de concepto y, a la vez, de validación funcional de los desarrollos realizados en el TFG.

### 1.3. Metodología

Para el desarrollo de este TFG se ha seguido una metodología que implica una serie de pasos concretos. A lo largo de cada una de las distintas fases se han estudiado y asimilado todos los conceptos necesarios para, posteriormente, poder diseñar, implementar y validar el sistema, que pretende completar todas las funciones listadas en la sección previa.

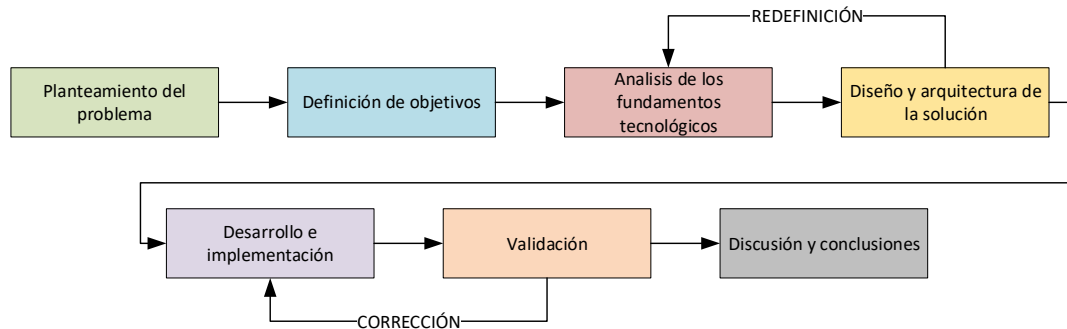


Figura 1.1: Visión a alto nivel de los pasos principales en la metodología.

En la Figura 1.1, se resumen las fases en las que se divide la metodología empleada y como se ha procedido a avanzar a lo largo de ellas para alcanzar los objetivos de este TFG. Se puede apreciar como el proceso se inicia con un análisis del caso que se quiere tratar en el TFG, su problemática y los objetivos a cumplir para poder abordar la misma. Después, se estudia y analizan las tecnologías principales, que incluye trabajos e investigación previa así como estándares, protocolos de comunicación y herramientas disponibles. Una vez se completan estos pasos, se diseña la solución para poder completar los objetivos. Esta solución, que implica un diseño particular, se ajusta en el tiempo, incorporando nuevas tecnologías del paso anterior. Después,

se desarrolla e implementa la solución, que deberá validarse más tarde y que servirá como retroalimentación para corregir las implementaciones y los desarrollos completados hasta el momento. Finalmente, se discutirán los resultados y se llegará a una o varias conclusiones.

## 1.4. Planteamiento y resumen ejecutivo

La metodología utilizada desencadena ciertos cambios mientras se desarrolla el TFG, en comparación con la primera propuesta. Sin embargo, estos cambios no han supuesto una modificación troncal de los conceptos y objetivos, pero sí del desarrollo.

La Figura 1.2 muestra el planteamiento inicial para el proyecto aquí desarrollado. Las diferencias con la Figura 1.3 (en color verde) son principalmente: el adaptador NGSIV2 (que pasa a la parte de base de datos histórica), la fuente de datos de FIWARE-Orion que pasa a ser un servidor local y finalmente, se añaden más fuentes de HTTP *polling*.

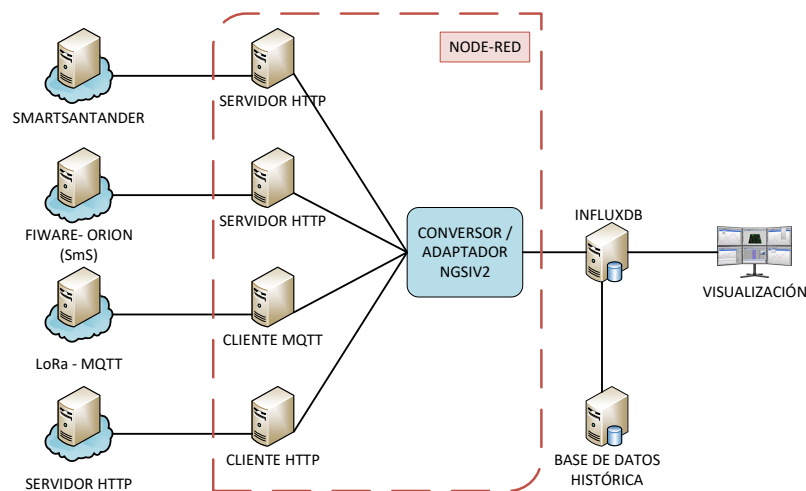


Figura 1.2: Planteamiento inicial de la estructura del proyecto.

Después de los cambios en el planteamiento final, la estructura del TFG se compone de los siguientes capítulos. El Capítulo 2 revisa los fundamentos tecnológicos y los aspectos más importantes que sirven como base conceptual para el TFG. El

Capítulo 3 presenta el diseño y desarrollo de la plataforma del lado del servidor que se encarga de recibir, analizar, almacenar y visualizar la información. El Capítulo 4 analiza y valida las acciones como el despliegue, el almacenamiento de la información o la visualización se desarrollen apropiadamente. El Capítulo 5 ofrece una reflexión de los conceptos clave del TFG y sobre el impacto en el desarrollo del TFG. Finalmente, el Capítulo 6 incluye las conclusiones del trabajo realizado en el TFG. La Figura 1.3 ofrece una relación entre los capítulos (y sus secciones) citados anteriormente con las entidades principales de la estructura del TFG.

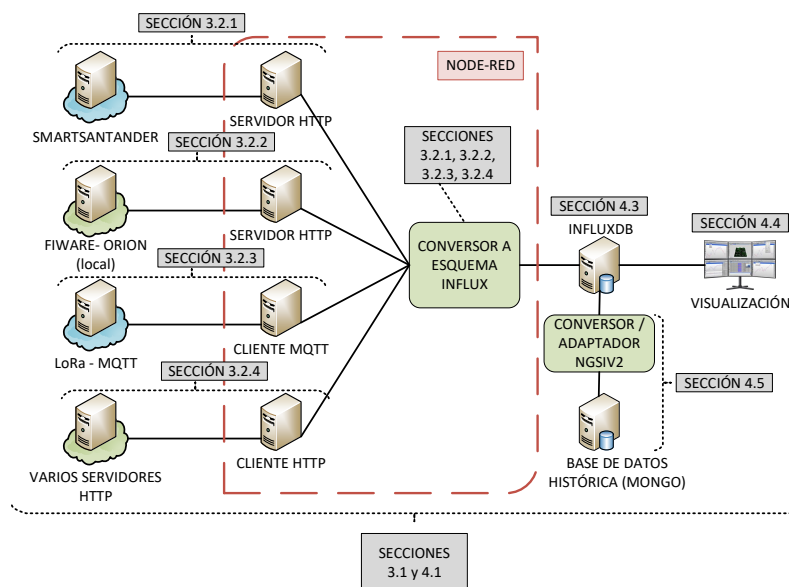


Figura 1.3: Planteamiento final de la estructura del proyecto.

# Capítulo 2

## Fundamentos tecnológicos

### 2.1. Redes de la Internet de las cosas (IoT)

Hoy en día existen una gran variedad de redes inalámbricas para los dispositivos de la IoT. Es posible usar redes inalámbricas como Wi-Fi, 3G o 4G para los dispositivos existentes. Sin embargo, estas tecnologías no ofrecen ciertas ventajas comparadas con las que se presentan en las siguientes secciones. De hecho, existen dispositivos que usan redes como *General Packet Radio Service* (GPRS) o los propios SMS para enviar y recibir información. No obstante, si se pretende abarcar un espacio muy amplio de casos de uso es normal que otras tecnologías tengan más importancia, ya que aspectos como el consumo de energía y una amplia cobertura son claves. Además, ciertas tecnologías inalámbricas hacen uso de espacios de frecuencia libres que no necesitan una licencia o un costoso despliegue de estaciones base, siendo tecnologías más económicas.

Una de las razones por las que se van a introducir las redes inalámbricas de baja potencia (*low-power wide-area network* o LPWAN) en las siguientes secciones es porque funcionan en un rango de cobertura muy amplio y los dispositivos que las usan tienen un consumo de energía relativamente bajo.

#### 2.1.1. Tecnologías de redes de baja potencia y área amplia (LPWAN)

Esta sección pretende explicar cuáles son las redes de baja potencia que tienen más popularidad en el ámbito de la IoT y cuáles son sus características. También se pretenden comparar ciertos aspectos de sus capacidades y cuáles pueden ser los casos

de uso que son mejores para cada tipo de red inalámbrica de baja potencia. Aunque el TFG no se concentra tanto en los diferentes tipos de redes inalámbricas de baja potencia, es importante entender cuáles son sus características, ya que su implantación es clave en todo el proceso de medición de datos y transferencia a la nube.

### **Sigfox**

Sigfox [2] utiliza una tecnología patentada de banda ultra estrecha (*Ultra Narrowband* o UNB). Tal y como muestra la Figura 2.1, se despliegan estaciones base propietarias en diferentes países, en una banda sin licencia por debajo del GHz [3]. Particularmente, utiliza los 868 MHz en Europa, los 915 MHz en Norteamérica, o los 433 MHz en Asia. Los dispositivos utilizan una modulación BPSK en una banda ultra estrecha de 100 Hz y a una transmisión máxima de 100 bps. Debido a la utilización de la mencionada banda ultra estrecha, Sigfox consigue usarla con una gran eficiencia y por eso sufre de niveles de ruido muy bajos. De esa forma logra una consumición de energía muy baja, además del uso de diseños de antenas de bajo coste.

Otra de las características es que al principio solo transmitía en el enlace ascendente (*uplink*) pero después soportó la comunicación bidireccional. El número máximo de mensajes que se pueden enviar al día es de 140 y el tamaño máximo de la carga (*payload*) es de 12 bytes. Los mensajes que ahora se soportan en el enlace descendente (*downlink*) no soportan mensajes de reconocimiento (*acknowledgement*) y están limitados a un máximo de 4 al día. La razón de la limitación de los mensajes y de la carga es optimizar la consumición de energía en los dispositivos.

### **LoRaWAN**

Alrededor del año 2015 la alianza de LoRa estandarizó un protocolo llamado LoRaWAN que usaba una banda que no tenía licencia por debajo del GHz. Dado el tipo de modulación que se utiliza con Lora, se garantiza una comunicación bidireccional y

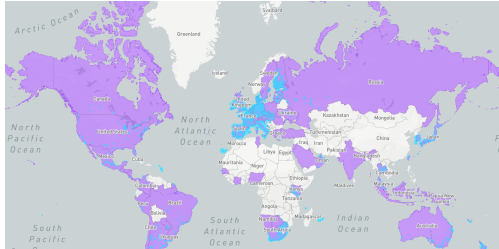


Figura 2.1: Mapa de la cobertura mundial de Sigfox, definida por un azul claro para zonas con cobertura en funcionamiento y morado oscuro para zonas ahora en despliegue [4].

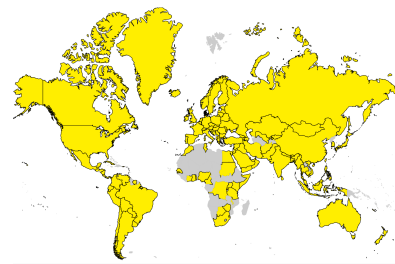


Figura 2.2: Mapa de la cobertura mundial de LoRaWAN. Se especifica en su página países con operadores públicos y cobertura de la comunidad [5].

también ofrece un nivel de ruido bajo, como lo hace Sigfox. Además, es complicado detectar o imposibilitar la comunicación usando su protocolo y ofrece un grado relativamente alto de recuperación y resiliencia frente a comunicaciones afectadas por interferencias [3]. El término LoRa se usa para referirse a la capa física, y LoRaWAN al protocolo y arquitectura del sistema. Las Figuras 2.1 (Sigfox) y 2.2 (LoRa) muestran una comparación de despliegue en diferentes de países. La Figura 2.3 muestra un ejemplo de red que incorpora nodos que utilizan LoRa.

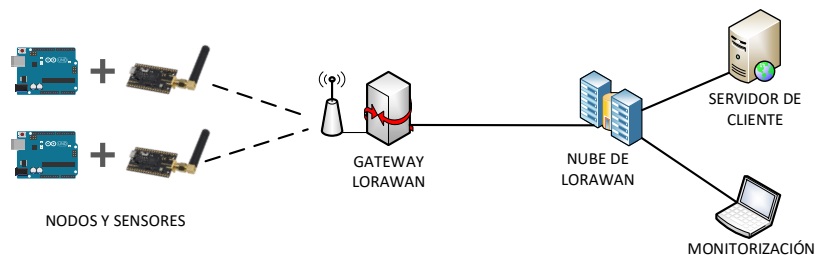


Figura 2.3: Un ejemplo de un caso de uso para la red de LoRaWAN. Utiliza sensores para enviar información a la nube de LoRa, haciéndolo accesible a los clientes.

Dependiendo del factor de propagación (*spreading factor*) que se use en Lora, se consigue una transmisión de datos y un rango de transmisión mayor o menor. En general, la tasa de transmisión de datos se encuentra entre los 30 bps y los 50 kbps, y la carga máxima de los paquetes es de 243 Bytes. Cuando los dispositivos transmiten mensajes, estos se reciben por todas las estaciones base en el rango. Dado que



se explota este tipo de redundancia, la fiabilidad de la transmisión es muy alta. En LoraWAN hay diferentes clases en lo que a la comunicación se refiere, cada una de las cuales ofrece diferentes recursos y es capaz de ofrecer una latencia más baja o más alta.

### Narrowband IoT

Narrowband IoT (NB-IoT) es una tecnología de banda estrecha dentro de las redes de las LPWAN. NB-IoT coexiste con tecnologías como 4G *Long Term Evolution* (LTE) o *Global System for Mobile Communications* (GSM) en las bandas de frecuencia bajo licencia. NB-IoT necesita 200 kHz de ancho de banda, lo que corresponde a un bloque de recursos en una transmisión mediante GSM o LTE [6]. Teniendo en cuenta los bloques de recursos que NB-IoT puede llegar a utilizar, existen varios modos de operación. Como muestra la Figura 2.4, los modos de operación se dividen en el modo con banda independiente (*stand alone*), banda de protección (*guard band*) y en banda (*in-band*). Aunque existe la posibilidad de desplegar NB-IoT como parte de las bandas independientes de GSM, el *3rd. Generation Partnership Project*(3GPP) recomienda desplegar NB-IoT en conjunción con LTE.

NB-IoT reutiliza muchas de las características de LTE, así como varios de los principios y bloques sobre los que está construido LTE. Sin embargo NB-IoT reduce muchas de las funcionalidades de LTE al mínimo para adecuar los casos de uso de la IoT. Por ejemplo, NB-IoT no requiere información relacionada con las medidas de la calidad de radio o agregación. De este modo, NB-IoT optimiza la comunicación para poder usar la mínima cantidad de energía [7]. Por ser más específicos con las partes técnicas, es importante mencionar que NB-IoT usa la modulación de *Single-carrier Frequency-division multiple access* (SC-FDMA) en el enlace ascendente y *Orthogonal frequency-division multiplexing* (OFDMA) en el descendente. La diferencia con el resto de tecnologías es que el máximo de la tasa de transmisión son 200 Kbps en el enlace descendente y 20 Kbps en el enlace ascendente.

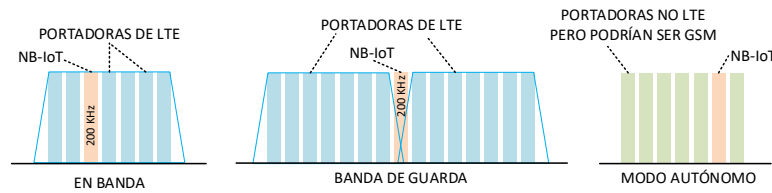


Figura 2.4: Modos de despliegue para NB-IoT: modo en banda, banda de guarda y autónomo.

## 2.2. Protocolos de comunicación

Aunque es importante mencionar cuáles son las tecnologías inalámbricas más relevantes hoy en día para la IoT, este TFG se enfoca más en la capa de aplicación. Esto significa que los servicios desarrollados en este TFG usan varios de protocolos de la capa aplicación, los cuales son importantes describir. Aunque no todos los protocolos listados en esta sección se utilizan en el desarrollado TFG, son relevantes para entender cuál es la visión general en esta temática.

### 2.2.1. Message Queuing Telemetry Transport (MQTT)

Es un protocolo estándar de transporte para mensajes entre dispositivos y un intercambiador (se referirá como *broker* de aquí en adelante) que sigue la función de publicación y suscripción, como puede verse en la Figura 2.5. La principal función de MQTT es minimizar el ancho de banda necesario para transmitir los mensajes enviados con este protocolo y también los recursos necesarios para una transmisión segura. Gracias a las características de MQTT es posible transferir mensajes sobre condiciones de redes con poca fiabilidad y se pueden llegar a consumir pocos recursos de energía, como ya han medido previos estudios [8]. MQTT corre sobre el protocolo *Transmission Control Protocol* (TCP), aunque puede usar algún otro protocolo que provea el mismo tipo de características, como por ejemplo, envío de datos en orden y conexiones bidireccionales y sin pérdidas. MQTT también puede correr sobre protocolos como *Transport Layer Security* (TLS) usando un puerto diferente (8883) al que se usa por defecto (1883).

La particularidad de MQTT es que se compone principalmente por tres entidades. Primero, la fuente de datos o publicador(es), es decir, el cliente MQTT que envía los datos. Siguiendo una cadena lógica, después de que el cliente envíe los datos, estos llegan al intercambiador conocido como broker, que se encarga de distribuir los datos sobre otros clientes (consumidores) interesados en estos. Los mensajes que llegan al broker se componen principalmente de datos que envió el sensor (productor), pero también otra información como el tema (*topic*) sobre el que se van a publicar estos datos que se reciben en el broker. Es así como un tercer cliente, también llamado consumidor, es capaz de recibir los datos después de haberse suscrito a un tema o varios.

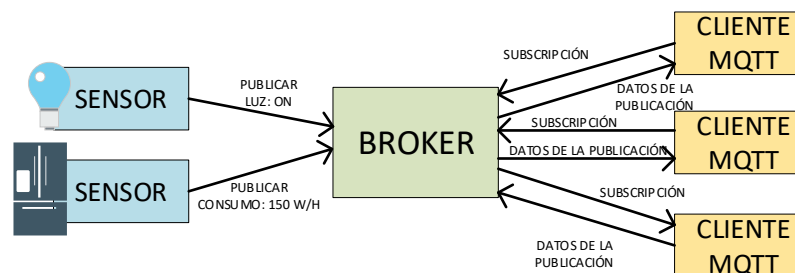


Figura 2.5: Arquitectura y organización de una red MQTT.

Sin hacer una descripción muy detallada de los temas MQTT, es necesario mencionar que estos se componen por términos independientes. El tema se forma con una cadena de términos que se enlazan mediante separadores ("/"). De esta forma los consumidores pueden llegar a suscribirse sobre un tema que abarcar varios temas multinivel o temas específicos, que se definen como temas subsecuentes enlazados por un separador. Por ejemplo, un suscriptor puede llegar a suscribirse a todos los mensajes de un tema como " Casa/Madrid/\* " o puede ser más específico y solo suscribirse a los temas de un lugar y tipo concretos como " Casa/Madrid/Planta1/baño/luz ". La organización de los temas depende de cada despliegue y caso de uso.

### 2.2.2. Constrained Application Protocol (CoAP)

Otro protocolo relevante en lo que se refiere a la capa de aplicación es el Constrained Application Protocol, conocido como CoAP [9]. Su estructura es muy similar a la de HTTP, pero corre sobre el protocolo UDP (véase la Figura 2.6). CoAP también utiliza arquitecturas tipo REST, usando métodos GET, POST, PUT o DELETE (entre otros). CoAP incluye una cabecera de tamaño reducido de 4 bytes con ciertos parámetros opcionales como el uso de un *token*. Este protocolo soporta métodos basados en petición/respuesta (*request/response*) o publicación/suscripción, parecido al anteriormente mencionado MQTT. El tipo de casos en los que se usa un modelo de publicación/suscripción es interesante para poder usar la mínima cantidad de energía.

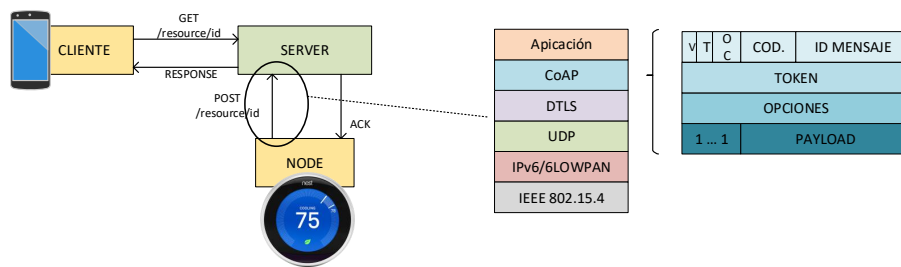


Figura 2.6: Ejemplo del uso del protocolo CoAP con sensores IoT y estructura de la cabecera del protocolo.

Es relevante mencionar que existen cuatro tipos de mensajes soportados por CoAP: confirmables (CON), no confirmables (NON), confirmaciones (ACK) y reset (RST). Asimismo, CoAP puede, opcionalmente, soportar el envío de información fidedigna en lo que a la fiabilidad se refiere. La mayor diferencia entre las transferencias que son seguras y fiables y las que no es que aquellas que son seguras se confirman mediante un mensaje ACK y las que no son seguras no necesitan recibir un mensaje ACK. Para las transmisiones seguras, si no se recibe un ACK en un periodo de tiempo concreto, el emisor volverá a transmitir el mensaje pero sin sobrepasar ciertos límites de retransmisión (como el número máximo de retransmisiones). Como CoAP usa el protocolo de transporte UDP, las confirmaciones que se han mencionado en los párra-

fos previos se consiguen mediante el protocolo de aplicación y no el de transporte. Por último, es importante mencionar que CoAP también puede transferirse usando protocolos de encriptados como *Datagram Transport Layer Security* (DTLS).

### 2.2.3. Advanced Message Queuing Protocol (AMQP)

Advanced Message Queuing Protocol (AMQP) [10] es un estándar de protocolo abierto que también funciona sobre el paradigma de publicación/suscripción, tal y como muestra la Figura 2.7. Se diseñó con el objetivo de habilitar la interoperabilidad en un rango amplio de diferentes aplicaciones y sistemas sin importar su diseño interno. Originalmente se diseñó con el objetivo de que fuese un sistema de mensajería comercial entre aplicaciones de organizaciones. La mencionada característica de interoperabilidad es importante, ya que permite que diferentes plataformas programadas con diversos lenguajes intercambien mensajes entre ellas, lo cual es favorable para sistemas heterogéneos [11].

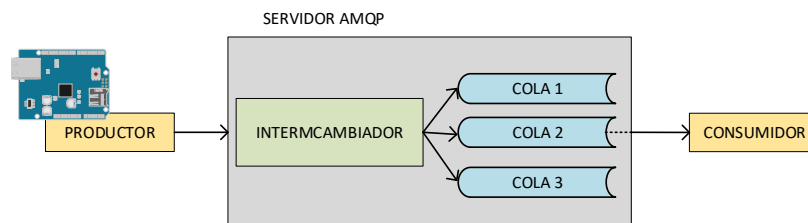


Figura 2.7: Ejemplo de arquitectura sobre la que funciona AMQP.

Aunque AMQP se diseñó con el propósito de proporcionar un mecanismo de publicación/suscripción, ahora mismo existen varias versiones de este protocolo. Por ejemplo la versión 0.9.1 ofrece, únicamente, un modelo de publicación/suscripción. En este, un intercambiador o broker como se le denominará de ahora en adelante, reenvía los mensajes a aquellos clientes que se suscribieron a un tema en particular. En versiones posteriores, como la 1.0, el protocolo no estaba sujeto solo al método de publicación/suscripción, sino también al método entre pares (*peer-to-peer*). Por lo

tanto, en esta última versión los clientes de AMQP podían publicar y suscribirse a ciertos temas o comunicarse entre ellos directamente, sin la intermediación de un broker. En consecuencia, el broker solo se necesita para cuando una entidad intermedia precisa almacenar los datos momentáneamente para reenviarlos al resto de suscriptores. Con el enfoque de la versión 1.0, AMQP consigue concentrarse en diferentes mecanismos como la comunicación cliente a cliente, cliente a broker y broker a broker.

En lo que se refiere a los detalles técnicos, AMQP utiliza TCP como protocolo de transporte y por lo tanto existe cierta fiabilidad a la hora de enviar los datos. Así como MQTT, AMQP también tiene mecanismos para enviar mensajes encriptados sobre protocolos tales como TLS y autenticación utilizando *Simple Authentication and Security Layer* (SASL). La mayor desventaja de este protocolo frente a los demás es que es relativamente pesado y, por lo tanto, los dispositivos que lo usan consumen más energía y necesitan algo más de capacidades de procesamiento.

#### 2.2.4. Hypertext Transfer Protocol (HTTP)

*Hypertext Transfer Protocol* (HTTP) es, con diferencia, el protocolo más utilizado en internet, y por lo tanto, el más conocido en esta sección. Es el principal protocolo en lo que se refiere a servicios web y la gran mayoría de servicios que funcionan con navegadores y servidores lo utilizan. La versión más extendida es HTTP/1.1, pero la versión 2 ya está ampliamente soportada por la mayoría de servidores y navegadores. Una de las mayores diferencias es que HTTP/2 es capaz de multiplexar diferentes conexiones sobre el mismo canal (*stream*) de TCP. Existen diferencias más específicas, como la posibilidad de que el servidor sea capaz de enviar recursos sin tener que ser pedidos primero por el cliente (como era típico hasta ahora) y que existe una mejor compresión en las cabeceras.

En los últimos años ha existido un esfuerzo notable para combinar IoT con el protocolo HTTP y con REST. Esto se debe a que el protocolo HTTP está global-

mente extendido y es soportado por la mayoría de los servidores. Con un estilo de arquitectura software como REST es posible transferir información sobre el estado de los dispositivos de una forma estandarizada para poder crear, leer, actualizar y borrar datos. Estas operaciones son conocidas como CRUD (*create, read, update, and delete*). De acuerdo estas operaciones, es posible crear un enlace directo entre estas y los métodos HTTP, haciendo posible que el modelo REST se pueda implantar en dispositivos IoT.

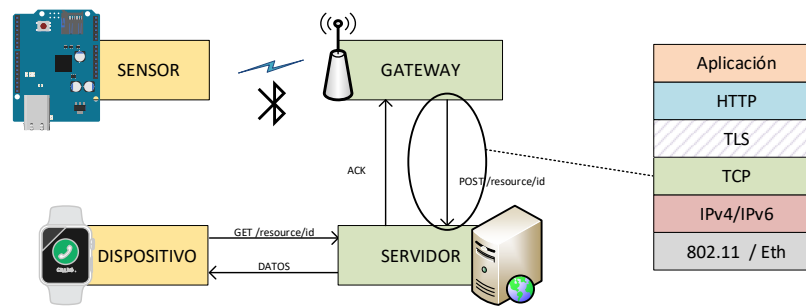


Figura 2.8: Caso de uso con clientes y servidores HTTP. También se muestra la estructura de las cabeceras, que incluyen los protocolos HTTP y TLS.

En la Figura 2.8 se muestra que el protocolo HTTP usa TCP como protocolo de transporte, para garantizar el envío de información de una forma fiable. Sin embargo, es cierto que el uso de HTTP y el protocolo TCP supone un esfuerzo mayor para los dispositivos, debido al procesamiento que requieren. Por ejemplo establecer un stream TCP supone que el cliente y servidor deben intercambiar ciertos mensajes para establecer una conexión, lo que se conoce como el intercambio de tres fases (*3-way-handshake*). Se debe mencionar que el protocolo HTTP funciona, generalmente, para los servicios web sobre el protocolo TLS. Por lo tanto, proporciona un enlace encriptado entre el cliente y el servidor aunque esto supone un mayor esfuerzo de procesamiento para los dispositivos IoT.

Es importante aclarar que el modelo de datos no es está ligado necesariamente al

protocolo HTTP o al formato de carga, pero es muy común encontrar dispositivos que transfieran la información usando formatos de datos de tipo JSON (JavaScript Object Notation) o Extensible Markup Language (XML) sobre HTTP. De hecho, es necesario explicar que existe cierta confusión con REST y las *Application Programming Interface* (APIs) HTTP. Lo cierto es que si se atiende a la tesis doctoral de Fielding [12], primero se formula el estilo y arquitectura de REST, pero HTTP no es necesariamente una parte intrínsecamente ligada a este estilo de arquitectura software.

## 2.3. Plataformas y lenguajes de programación visual

Una vez se han descrito los protocolos de aplicación más relevantes en el mundo de la IoT, es importante remarcar que en el TFG la programación visual tiene una importancia mayúscula. Uno de los mayores objetivos es llegar a definir una forma de programar servicios del lado del servidor para que se pueda recibir, procesar, almacenar y visualizar información de los dispositivos IoT, usando solo entidades visuales. Aunque este trabajo utiliza Node-RED para definir el diseño de los servicios del lado del servidor, hay otras herramientas para programar visualmente. Las más populares que existen hoy en día se describen brevemente en las siguientes secciones, y se analiza las características de estas herramientas.

### 2.3.1. Blockly

Blockly es una librería [13] que añade un editor de código a aplicaciones web y móviles. Blockly utiliza representaciones de bloque y establece conexiones entre ellos, representando variables, expresiones lógicas, iteraciones, etc. Usando este tipo de bloques, como los que se muestran en la Figura 2.9, el desarrollador no necesita preocuparse por la sintaxis, ya que la formulación de un programa se hace enteramente con bloques. Los desarrolladores pueden crear sus propios bloques e integrar el editor que



ya se incorpora por defecto a herramientas terceras que se puedan añadir. Al final, el objetivo de esta librería es generar código a partir de expresiones visuales.

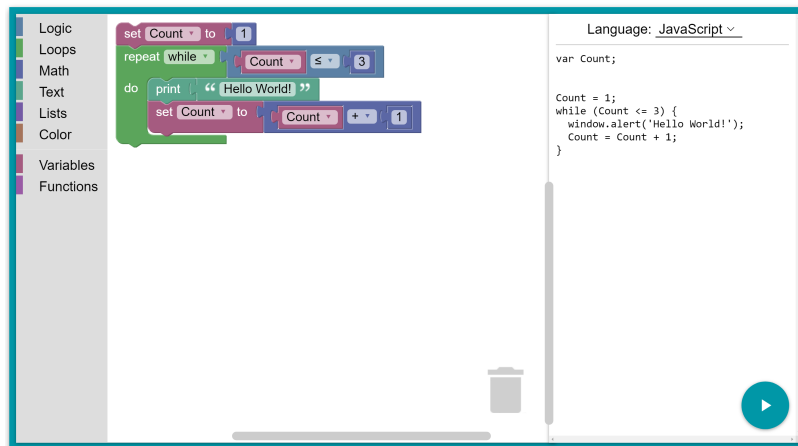


Figura 2.9: Un conjunto de bloques en Blockly que se traducen a código JavaScript y se ejecutan en el servidor.

Blockly, que es un proyecto de código abierto, incorpora multitud de ventajas como la posibilidad de que el código (texto) se pueda exportar directamente desde el editor. Como ya se mencionó anteriormente, se pueden extender los bloques que ya se incorporan por defecto e incluso generar programas complejos a partir de los ya definidos. Para las personas que aprendan a programar o estudiantes esta librería se ha traducido ya a más de 40 idiomas.

Aunque Blockly es una librería de gran utilidad, la programación de ciertos servicios del lado del servidor implicaría un detalle mucho mayor del que necesita este trabajo. Es decir, no se pretende crear un programa que actúe, literalmente, como un servidor HTTP real y que extraiga cabeceras o procese métodos de una forma muy detallada. Más bien, se pretende proporcionar un nivel más alto de programación en lo que abstracciones de funcionalidades y flujo de datos se refiere.

### 2.3.2. Scratch

Scratch [14] es un lenguaje de programación visual que también se basa en bloques. El objetivo principal de esta herramienta son estudiantes muy jóvenes que empiezan a programar. Por lo tanto, se usa como una herramienta para la enseñanza de la programación. Así como ocurre con la herramienta anterior, es posible programar mediante una interfaz web, que se muestra en la Figura 2.10.

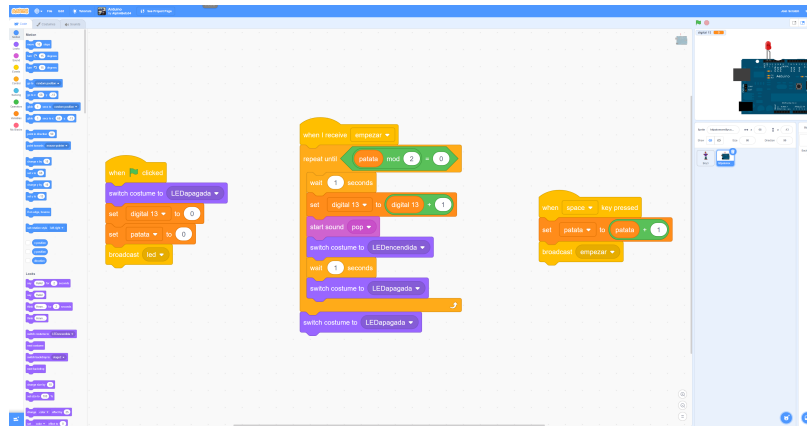


Figura 2.10: Caso de uso con Scratch para programar un Arduino y su led.

Al igual que lo hace Blockly, Scratch también utiliza bloques visuales que forman su sintaxis. Dependiendo de la forma de los bloques, se puede verificar si algunos de ellos son capaces de conectar con otros. De esta forma es fácil de comprobar si la combinación de varios bloques supone la creación de una entidad de mayor abstracción de programación. En este lenguaje también se pueden representar funciones, disparadores (*triggers*), estructuras de bloque, comandos, etc. Asimismo, los tipos de datos en Scratch también se pueden representar como booleano, numérico o cadenas de caracteres.

Aunque este lenguaje de programación visual proporciona herramientas muy útiles, no sería posible crear los escenarios necesarios para este trabajo, por lo que se debe buscar una herramienta de programación visual que sea capaz de enfocarse sobre

todo en los casos de IoT.

### 2.3.3. Node-RED

Node-RED [15] no puede considerarse el mismo tipo de herramienta que Blockly o Scratch, pero se ha decidido integrar en la misma sección porque sigue representando una forma de programar visualmente. Node-RED es un editor de flujos que también utiliza el navegador para poder interconectar nodos. Esta herramienta incorpora nodos que representan funcionalidades muy básicas como disparadores para generar un estímulo en un flujo o herramientas muy complejas, como un broker o cliente MQTT. Node-RED se basa principalmente en crear funciones de código JavaScript para poder generar, procesar o alterar mensajes que se reciben en el lado del servidor. Una de las principales características de Node-RED es que es capaz de generar flujos muy complejos. Por ello, una herramienta de este tipo es de gran utilidad cuando se trata de recibir, alterar, adaptar y almacenar datos que llegan de dispositivos IoT.

La razón por la que esta herramienta es tan importante en este trabajo es que está especialmente enfocada a casos de uso de la IoT. Es posible generar servicios del lado del servidor sin necesidad de herramientas especializadas o conocer la configuración de las mismas.

En la Figura 2.11, se muestran varios ejemplos de flujos en Node-RED. Los 3 flujos superiores, de arriba a abajo, crean 3 servidores que responden con el código de un sitio web simple que accede a los encabezados de la solicitud y escucha un formulario, respectivamente. El par inferior de flujos, de arriba a abajo, generan una solicitud GET con parámetros en una Uniform Resource Locator (URL) y procesan la salida JSON de una solicitud GET, respectivamente.

## 2.4. Almacenamiento de datos

El objetivo de esta sección no es aportar detalles muy específicos sobre diferentes tecnologías de bases de datos que existen en la actualidad. Sin embargo, es necesario

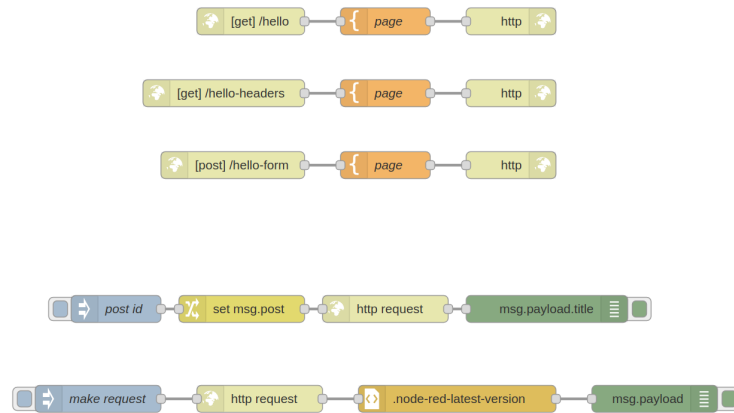


Figura 2.11: Ejemplos de flujos Node-RED.

detallar las diferencias de los grupos de bases de datos más relevantes en lo que almacenamiento de datos se refiere.

### 2.4.1. Bases de datos relacionales o SQL

Las bases de datos de tipo Structured Query Language (SQL), pertenecen al grupo de sistemas que siguen un modelo relacional. El software que gestiona la información en una base de datos relacional (BDR) se denomina como *Relational Database Management System* (RDBMS). Un modelo de datos relacional implica que las estructuras de datos se pueden relacionar con otras estructuras en la misma base de datos. Este concepto es originario del año 1970, a partir de un trabajo de investigación de IBM dónde se proponía un modelo relacional para grandes bancos de datos compartidos [16]. Generalmente, este tipo de bases de datos organizan su información en unas estructuras conocidas como tablas, compuestas por columnas (*columns*) e hileras (*rows*). Las columnas se utilizan como atributos a la hora de estructurar la información y cada hilera se suele denominar también como registro. Se puede ver un ejemplo de esta estructura en la Figura 2.12. Las bases de datos de tipo relacional se caracterizan por incluir unas propiedades conocidas como ACID, que significa Atomicidad, Consistencia, Aislamiento y Durabilidad (*Atomicity, Consistency, Isolation, Durability*).

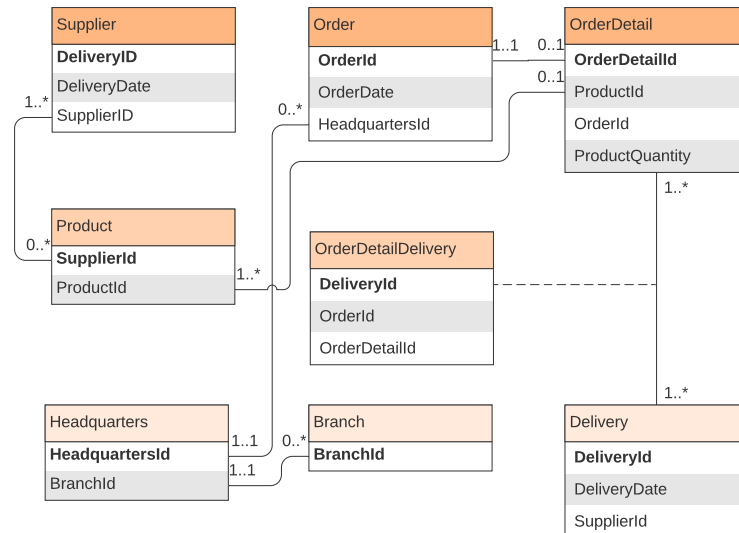


Figura 2.12: Ejemplo de un diagrama entidad-relación para una base de datos relacional (RDBMS).

Como se ha introducido anteriormente, este tipo de base de datos se manipulan (consultas y mantenimiento) utilizando un lenguaje de consulta estructurada llamado SQL [17], o una versión derivada del lenguaje estandarizado (e.g., PL/SQL [18], pgSQL [19], etc.). La sintaxis del lenguaje SQL se caracteriza por estar dividida en diferentes partes. Principalmente, se compone de comandos, operadores, cláusulas y funciones. La combinación de estos elementos posibilita la consulta y manipulación de los datos en la BDR.

#### 2.4.2. Base de datos de siguiente generación

En la primera mitad de la década de los 2000, hubo un gran desarrollo de bases de datos relacionales. A mediados de la primera década de los años 2000 no parecía que ninguna revolución en las bases de datos fuese a proponer nuevos sistemas o modelos. Sin embargo, la innovación en las arquitecturas cliente-servidor y la masificación de las aplicaciones web cambiaron el futuro de las bases de datos [20]. Estos efectos

acarrearon innovaciones con nuevas características incrementales que las bases de datos relacionales no podían cubrir.

Aunque al principio este tipo de bases de datos no tenían una denominación específica se acabó por asignarles el nombre de *Distributed Non-Relational Database Management System* (Distributed Non-RDBMS). Sin embargo, el término *NoSQL* [21] fue el que acaparó la mayor atención, para referirse a todas aquellas bases de datos que no seguían el modelo relacional de las RDBMS. Las bases de datos NoSQL son de diferente tipo. Mayoritariamente, son de tipo documental, orientadas a grafos, o clave/valor, como puede verse en la Figura 2.13. Las siguientes son algunas de las bases de datos más populares en el ámbito NoSQL: MongoDB (documental), CouchDB (documental) Couchbase (clave/valor), Cassandra (clave/valor), Redis (clave/valor), Neo4J (grafo).

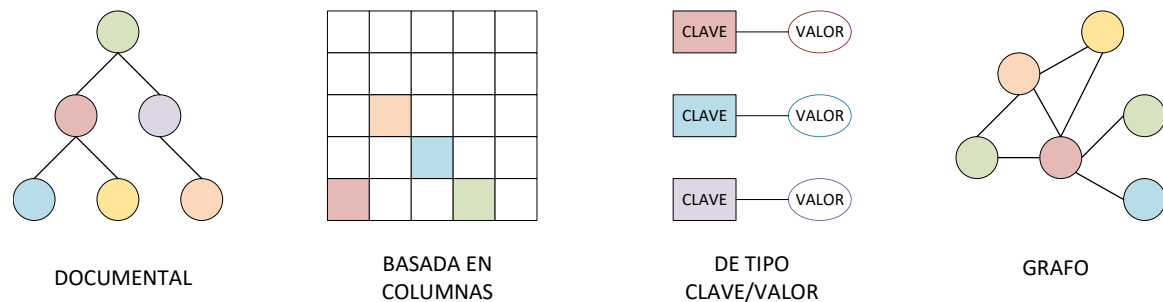


Figura 2.13: Representación estructural de las bases de datos NoSQL.

Unos años después de que NoSQL se convirtiese en un tema popular, un nuevo término denominado *NewSQL* adquirió notoriedad. Este tipo de bases de datos no suponían una disrupción tan grande como NoSQL frente a las bases de datos relacionales. Igualmente, mejoran ciertas características de las RDBMS e incorporan algunas propiedades de las bases de datos que típicamente parte de NoSQL. CockroachDB, VoltDB o Google Spanner son algunas de las bases de datos más conocidas.

## 2.5. Modelado de datos y contextualización

En este TFG se trabaja con varios modelos de datos en lo que se refiere a la capa de aplicación. Cuando los dispositivos envían información y los servidores o *proxies* intermedios (si es que hay alguno) la distribuyen hasta el servidor desarrollado en el TFG, lo hacen siguiendo ciertas reglas. Es decir, siguen un modelo de datos estandarizado o propietario para que todas las entidades que procesan los datos extremo-a-extremo puedan contextualizar la información. Frecuentemente, se suele confundir el formato de los datos, como puede ser XML o JSON, con el modelo de datos. Un modelo de datos siempre podrá expresar su definición de recursos y sintaxis con uno o varios formatos de datos. Por lo tanto, ambos conceptos están relacionados pero son independientes. Buscando una analogía simple, podría decirse que una persona siempre podrá pronunciar el fonema /n/ tanto si habla inglés como si habla chino, pero la presentación es diferente en inglés (“n”) y en chino (“泥”).

### 2.5.1. Next Generation Service Interface (NGSI)

El modelo de datos de referencia que se va a utilizar a lo largo de este TFG es conocido como *Next Generation Service Interface* (NGSI), versión 2. Es un estándar de la OpenMobile Alliance (OMA) y soporta la recepción, procesamiento, contextualización y publicación de datos IoT. De acuerdo a la especificación de NGSI, la API define varios conceptos [22]:

- Un modelo de datos.
- Una interfaz de datos de contexto.
- Una interfaz de disponibilidad de datos de contexto.

El modelo de datos organiza los elementos de información de contextualización y estandariza como se relacionan entre ellos. La interfaz de datos de contexto define como intercambiar información sobre consultas, suscripciones y operaciones de

actualización. Asimismo, la interfaz de disponibilidad, posibilita el intercambio de información para descubrir cómo se obtienen los datos de contexto. El modelo de datos de contexto se organiza, principalmente, sobre entidades, atributos y metadatos. Estos términos siguen cierta jerarquía que se muestra en la Figura 2.14.

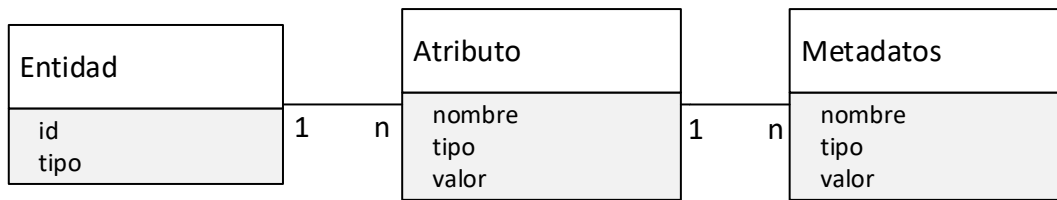


Figura 2.14: Estructura de NGSIV2.

Las entidades de contexto son la piedra angular del modelo de información. La especificación define a una entidad de una forma generalizada como una “cosa” u objeto físico o lógico. Por ejemplo, una entidad podría ser un sensor, una persona o una habitación. Las entidades se representan mediante su **ID**. Además, la especificación define el **tipo** de entidad que podría ser, por ejemplo, un *sensor\_de\_temperatura*. La combinación del ID y tipo genera una identificación unívoca de una entidad. El atributo (o varios de ellos) definen propiedades de las entidades. Por ejemplo, la “temperatura” podría ser un atributo de la entidad “sensor123”. Los atributos se definen por los tres denominadores listados en la figura anterior: nombre, tipo (tipo del valor NGSIV) y valor. Estos denominadores podrían equivaler, en un caso práctico, a temperatura, entero (*integer*) y 25. La estructura de entidades, atributos y metadatos pueden verse descrita en la Figura 2.14.

Además del modelo de datos ya introducido existen otros conceptos en la especificación que son necesarios mencionar. Por ejemplo, la especificación [22] también define modelos simplificados para la representación de entidades, así como representaciones parciales y restricciones a la hora de definir ciertos campos.



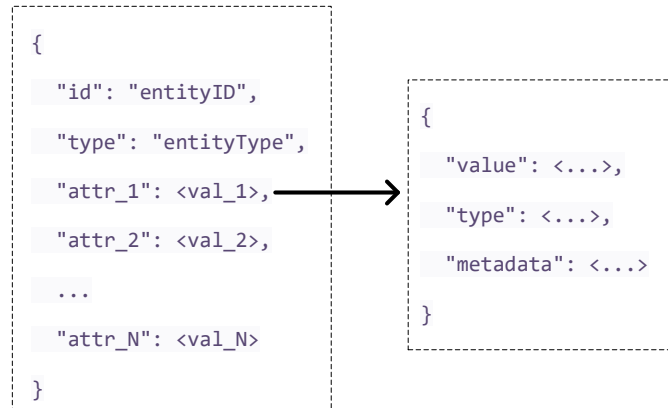


Figura 2.15: Equivalencia entre la especificación NGSIv2 y un ejemplo práctico.

Como se introducía en esta sección, la API de NGSIv2 no solo define un modelo de datos de contexto, sino también ciertas interfaces. Es decir, crea una referencia sobre cómo consultar datos de contexto, empezando por un punto de referencia en la API que define cómo consultar datos de recursos. En otras palabras, el servidor responde con enlaces en un cuerpo JSON (véase la Figura 2.15), sobre cómo un cliente debe consultar la información y dónde encontrar los recursos. Por ejemplo, el punto de entrada de la API responderá sobre qué enlace un cliente puede listar las entidades disponibles, así como gestionar sus propias suscripciones. No se pretende dar una explicación muy extensa de cómo funciona la interfaz, ya que el Capítulo 3 incluye una larga explicación de cómo se han recolectado los datos del servidor de SmartSantander o de un servidor propio Orion, que siguen un modelo como el aquí explicado.

# Capítulo 3

## Diseño y desarrollo

Esta sección describe, en primer lugar, el proceso de despliegue de todos los componentes que se usan para establecer servicios del lado del servidor. Posteriormente, se detallan todos los casos de uso que se pretenden analizar y cuál es el desarrollo para poder ponerlos en funcionamiento.

### 3.1. Despliegue de la plataforma

Uno de los objetivos de este TFG es hacer que el desarrollo de servicios del lado del servidor sea lo más sencillo y automatizado posible. Por lo tanto, desplegar todas las herramientas no puede ser una actividad que necesite de un trabajo tedioso por parte de los desarrolladores. Aunque es posible seguir ciertos pasos de clonación, compilación y ejecución de todas las herramientas, supondría un tiempo demasiado largo para alguien que necesita una herramienta desplegada en segundos. Por lo tanto, esta sección pretende explicar cuál es el papel que juegan los contenedores y las ventajas frente a otras opciones.

#### 3.1.1. Entorno y contenerización

Las máquinas virtuales han supuesto un medio muy popular para virtualizar los recursos de la infraestructura. En general, tienen un tamaño en el orden de gigabytes (GB) y suelen incluir su propio sistema operativo, asignando ciertos recursos de la infraestructura a ellas (como memoria RAM, o núcleos de una CPU, entre otros). De esta forma se pueden tener varios sistemas operativos a la vez en una misma entidad de computación, compartiendo los recursos. Los centros de datos más grandes del

mundo se caracterizan por ofrecer el alquiler de máquinas virtuales como servidores a desarrolladores. Se denominan como Servidor Virtual Privado (Virtual Private Server o VPS). Presentan una función muy útil a la hora de desplegar máquinas virtuales para cumplir una función muy concreta.

Por otro lado, los contenedores son una forma de virtualización de los servicios ofrecidos sobre un sistema de computación. Generalmente, ocupen del orden de megabytes (MB) y suelen alojar aplicaciones y los archivos necesarios para su ejecución. Después de que se añadieran los *cgroups*, diferentes proyectos emergieron para desplegar procesos contenerizados. Algunos ejemplos son Linux Containers (LXC), *containerd*, *libcontainer*, *rkt* o *appc*.

Docker es probablemente la plataforma de virtualización más famosa y Kubernetes la plataforma de orquestación más extendida. Cuando el orquestador de contenedores Kubernetes [23] fue publicado, Docker era una parte estática del mismo. Cuando Kubernetes se hizo popular, empezó a buscar otros sistemas *runtime* aparte de Docker. Ahí es donde varias empresas colaboraron para definir el *Container Runtime Interface* (CRI), que otorgaba a Kubernetes la posibilidad de soportar varios *runtimes*. Por ser más específicos, en el CRI hay dos principales elementos: *containerd* y *CRI-O*. Estos dos *runtime* fueron una de las principales razones para que Google decidiese dejar de soportar *Dockershim*, un componente necesario para que *containerd* (que es parte de Docker) trabaje correctamente con Kubernetes. El hecho de que Docker no sea compatible con CRI hace que Google decida dejar de soportarlo como runtime, pues necesita herramientas extra (*Dockershim*) para usarlo [24]. Igualmente, Docker sigue estando ampliamente extendido y es parte fundamental del TFG.

En resumen, la naturaleza menos pesada de los contenedores frente a las máquinas virtuales, su portabilidad y su fácil despliegue hacen de este tipo de virtualización la mejor para este proyecto.

### 3.1.2. Despliegue

Como ya se ha comentado anteriormente, en este TFG se han usado contenedores para establecer el funcionamiento de todas las entidades. Particularmente, se desplegarán los siguientes contenedores:

- Node-RED
- InfluxDB
  - InfluxDB CLI
- Grafana
- Orion
- MongoDB
- Telegraf
- Portainer

No todas las entidades listadas son necesarias y su utilización o no depende del caso de uso. Por ejemplo, Telegraf, es un agente para la recopilación de datos y reporte de métricas para InfluxDB. El gran número de *plugins* de los que dispone para la recepción de datos y su arquitectura *plug-and-play* hacen que, fácilmente, se puedan recopilar métricas. Aunque para este TFG no es una parte fundamental, es común desplegarla como parte del paquete TIG (Telegraf, InfluxDb, Grafana) y en un futuro podría ser de ayuda para visualizar y recolectar datos IoT.

En este TFG se usa específicamente el *runtime* Docker y se adquieren los contenedores usando su repositorio principal. Sin embargo, su despliegue no es manual sino automatizado usando una utilidad llamada **Docker Compose**. Se ha creado un único archivo de configuración en formato YAML (YAML Ain't Markup Language)

para que Docker Compose despliegue todos los contenedores y servicios con un orden específico. Además, el archivo de configuración define el tipo de red interna que se debe crear para interconectar todos los contenedores en la misma subred.

Como se puede ver en el extracto de código del Listado 3.1, la configuración para este trabajo usa imágenes diferentes a las que InfluxDB provee en su repositorio oficial de Docker. Sin embargo, la de Node-RED es la que se publica por defecto y se usa la última versión. Es importante decir que Node-RED solo se despliega una vez que InfluxDB ya está operativa. Ocurre igual para Telegraf o el InfluxDB CLI, debido a que se deben ejecutar algunos comandos automáticamente solo cuando InfluxDB ya esté en funcionamiento. También se puede comprobar que cada servicio exporta ciertos puertos y que expone como parte de la máquina que corre los contenedores. Esto es importante porque terceras aplicaciones nunca podrían contactar con la base de datos InfluxDB o con Orion si no se publicasen dichos puertos. Sin embargo, para la mayoría de casos, el único puerto que necesita ser expuesto al exterior es el de Node-RED, ya que el resto de comunicaciones ocurren internamente. Por eso, se define una red para todos los servicios, para que puedan comunicarse entre ellos y cada uno pueda tener su propio nombre de máquina u *hostname*. Así, los servicios que necesiten acceder a la base de datos no necesitan saber su dirección IP sino su *hostname*.

```
1 version: '3'
2 services:
3   nodered:
4     image: nodered/node-red:latest
5     depends_on:
6       - influxdb
7     links:
8       - influxdb
9     ports:
10      - '1880:1880'
11     networks:
12      - interna_tfg
13     volumes:
14      - node_red_data:/data
15     container_name: nodered
```

```
16 influxdb:
17   image: quay.io/influxdb/influxdb:v2.0.3
18   volumes:
19     - influxdbv2:/influxdbv2
20   ports:
21     - "8086:8086"
22   networks:
23     - interna_tfg
24   hostname: influxdb
25   container_name: influxdb
26   #
27   # Otros servicios: Grafana, Orion, etc.
28   #
29 networks:
30   interna_tfg:
31 volumes:
32   influxdbv2:
33   grafana_data:
34   node_red_data:
```

Listado 3.1: Configuración YAML de Docker compose para desplegar de forma automatizada el escenario de desarrollo.

Los servicios listados en el archivo de configuración YAML cumplen funciones específicas a la hora de desplegar servicios IoT del lado del servidor. Por ejemplo, Node-RED se encarga de desplegar las principales funciones como: la interfaz de usuario (*User Interface* o UI) de alto nivel; los servidores, receptores o clientes de tipo *pull* (e.g., HTTP); la transformación de los datos; y los agentes de la base de datos. InfluxDB almacena los datos enviados por Node-RED y se establece como la base de datos de lectura para la aplicación de visualización Grafana. Grafana construye un tablero de instrumentación (*dashboard*) dinámico que altera los datos de visualización dependiendo de la elección del usuario. Por lo tanto, y sin cambiar el tablero, el usuario puede visualizar los resultados de diferentes fuentes de datos como pudiese ser SmartSantander u OpenWeatherMap. La ubicación de cada contenedor desplegado dentro de la arquitectura de red utilizada en el TFG puede verse en la Figura 3.1.

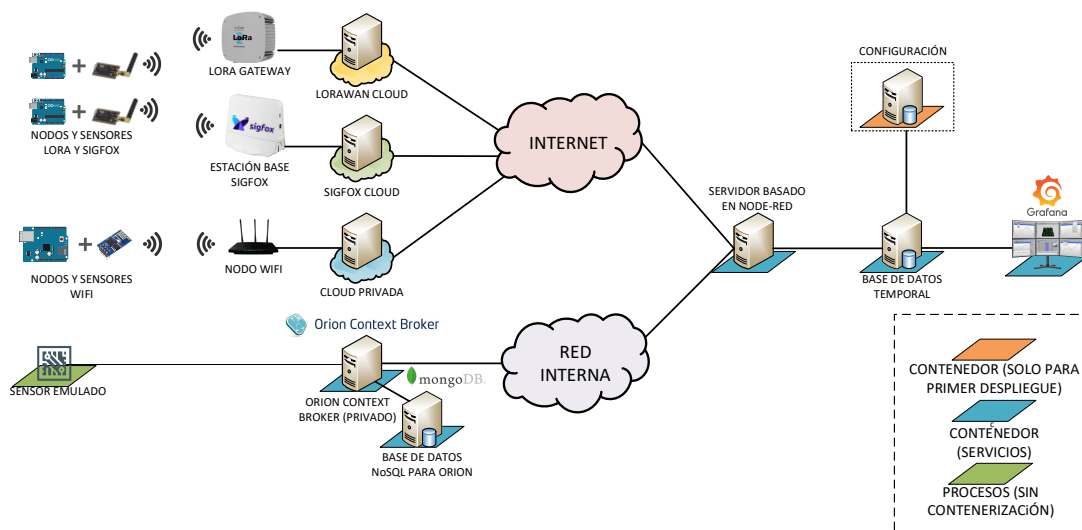


Figura 3.1: Representación completa de la distribución de contenedores.

## 3.2. Diseño e implementación de los casos de uso

En esta sección se explicará cada caso de uso, su diseño y la implementación de todas las partes relevantes del flujo. No todos los casos de uso se han implementado de la misma forma. Por ejemplo, en el Caso 1 se ha programado una interfaz de usuario para gestionar el contexto de las suscripciones, pero el Caso 3 no necesita de una interfaz del mismo tipo, ni tan compleja. Se debe a que la recolección de datos en los Casos 3 y 4 es más simple, comparada con los Casos 1 o 2. Todas las particularidades de cada caso se describen en las siguientes secciones.

### 3.2.1. Caso 1: *hook* HTTP para SmartSantander

Siendo este uno de los principales casos de uso, esta sección incluirá partes con una explicación más detallada sobre la programación de la UI. Para poder crear todos los servicios necesarios del lado del servidor, se deben generar suscripciones a los datos que SmartSantander puede proveer y se debe crear un servidor para recibirlos. Los datos no se adquieren directamente desde el servidor, sino que se genera una suscripción para que la información llegue a un destino solo si ciertos parámetros se cumplen. Estos parámetros son incluidos a la hora de enviar la suscripción desde

el cliente a SmartSantander. La visión general del caso de uso puede verse en la Figura 3.2.

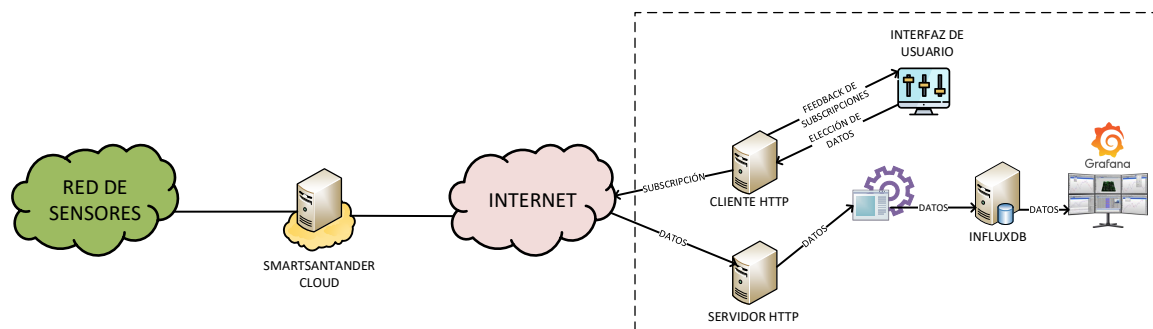


Figura 3.2: Representación de todas las entidades relevantes en el Caso 1.

### Interfaz de usuario

La interfaz de usuario en este caso de uso cumple una función fundamental. No solo permite que un usuario pueda suscribirse a cierto tipo de información, sino que permite seleccionar el tipo de datos que desea monitorizar en tiempo real. También es posible seleccionar los parámetros de alerta, el servidor destino o la localización de filtrado. Además monitoriza si una suscripción está o no activa una vez que la misma se crea. Es una funcionalidad clave porque si el servidor deja de funcionar o no funciona correctamente, el servidor de SmartSantander lo detectará y dejará de enviar información. Por lo tanto, con esta herramienta es más fácil monitorizar las suscripciones.

La interfaz de usuario creada para este caso de uso, puede verse en la Figura 3.3.

La interfaz de usuario desarrollada muestra los parámetros de configuración y el estado de la suscripción, así como el estado de la suscripción, el ID de la última suscripción creada y los mensajes recibidos por el servidor si se activa la misma.

Para poder crear una suscripción, nuestro servidor debe primero consultar qué tipo de fenómenos soporta, tal y como se muestra en la Figura 3.4. Esto se consigue pulsando el botón de actualización (“ACT.” en la UI) o mediante un inyector de forma manual. Una vez que se obtiene esta información utilizando la correspondien-



SMARTSANTANDER

FENÓMENO

UNIDAD

DESTINATION URL

CONDITION

Lat. **43.467064** Long. **-3.813624** Radius (m) **329**

SELECCIONAR SUB ID

Automático

SUB ACTIVA?

FECHA EXPIRACION:

ULT. SUB ID CREADA:

Contador de mensajes:

Figura 3.3: Visión general de la interfaz de usuario para el Caso 1.

te API de SmartSantander, el usuario selecciona un tipo de fenómeno. Entonces, el servidor automatiza la petición de unidades asociadas al mismo. Así, el usuario ya puede seleccionar el fenómeno y la unidad (por ejemplo, “temperature:ambient” y “degreeCelsius”).

SMARTSANTANDER

FENÓMENO

UNIDAD

DESTINATION UR

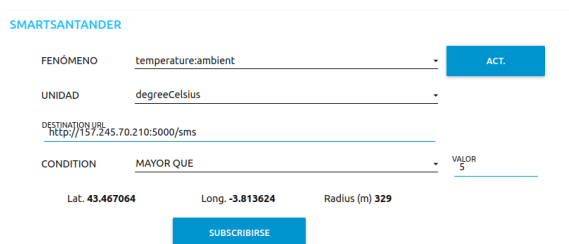
CONDITION

Lat. **43.4670**

Figura 3.4: Visión de todos los fenómenos recibidos de SmartSantander.

Una vez se elige el fenómeno y la unidad, el usuario debe introducir la URL de destino para que el servidor de SmartSantander envíe la correspondiente notificación cada vez que las condiciones impuestas por el usuario se cumplan. A la hora de esta-

blecer estas condiciones, es posible determinar si el valor del fenómeno debe ser igual, mayor e igual o menor e igual a un determinado valor, como muestra la Figura 3.5. Además, el usuario también puede determinar la localización en la cual los sensores se deben encontrar, imponiendo un radio desde las coordenadas geográficas definidas. De esta forma, cuando SmartSantander reciba un nuevo mensaje de un sensor que se encuentre en el radio definido y su valor cumpla la condición determinada por el usuario, la correspondiente notificación será enviada a la URL de destino introducida por el usuario. En la Figura 3.5 se puede comprobar el proceso mediante una interfaz de usuario y la ejecución manual con bloques de Node-RED. Ambas son compatibles y pueden funcionar a la vez, pero será el usuario quien determine la forma más rápida y útil.



The image shows a web form titled "SMARTSANTANDER" for creating a subscription. It contains several input fields and buttons:

- FENÓMENO:** A dropdown menu with "temperature:ambient" selected.
- UNIDAD:** A dropdown menu with "degreeCelsius" selected.
- DESTINATION URL:** A text input field containing "http://157.245.70.210:5000/sms".
- CONDITION:** A dropdown menu with "MAYOR QUE" selected.
- VALOR:** A text input field containing "5".
- Coordinates:** "Lat. 43.467064" and "Long. -3.813624".
- Radius:** A text input field containing "329".
- Buttons:** A blue "ACT." button next to the "FENÓMENO" field, and a blue "SUBSCRIBIRSE" button at the bottom.

Figura 3.5: Datos introducidos en el Caso 1 para enviar una subscripción.

Una vez el usuario decide suscribirse (botón “SUBSCRIBIRSE” en la interfaz de usuario) o decida hacerlo manualmente mediante un inyector, el servidor de SmartSantander habrá creado una subscripción. El ID de esta subscripción se muestra en la página automáticamente. Cuando se envía la petición de subscripción, un segundo bloque de despliegue mostrará todas las subscripciones ya creadas en SmartSantander con nuestra llave API (*API KEY*).

Una vez se selecciona el ID de la subscripción (véase la Figura 3.6), el cliente HTTP de Node-RED pedirá la información al servidor de SmartSantander. Como se ve en la siguiente Figura 3.7, la interfaz de usuario muestra con un color rojo la subscripción, porque no está activa o habilitada. La fecha de caducidad de la subscripción muestra

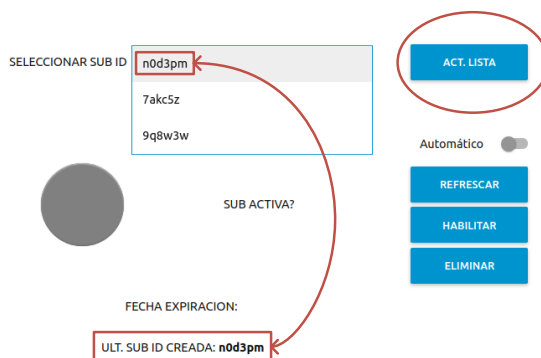


Figura 3.6: Listado de todas las suscripciones mostrando también la última creadas.

una fecha ya pasada, aunque este es el valor por defecto para todas las suscripciones nuevas que aún no han sido activadas.

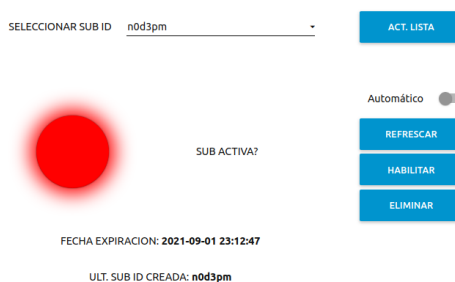


Figura 3.7: Demostración visual de una suscripción no activa.

Si se hace click para habilitar la suscripción (botón “HABILITAR” en la interfaz de usuario), el cliente HTTP realizará una petición de activación a través de la API de SmartSantander, como se muestra en la Figura 3.8. En el momento en el que un sensor envíe nueva información, que esté dentro del rango geográfico configurado y que cumpla las condiciones impuestas (por ejemplo, temperatura mayor que 5 grados Celsius), una notificación (implementada mediante una petición POST de HTTP) será enviada a la URL de destino incluida en la suscripción.

Cuando el servidor HTTP recibe mensajes del servidor de SmartSantander, la interfaz de usuario muestra un contador (mostrado en la Figura 3.9), Así, es posible comprobar la correcta comunicación entre los sensores y el servidor de SmartSantander, con el desplegado en Node-RED.



Figura 3.8: Demostración de una subscripción activa después de enviar una petición de activación.

ULT. SUB ID CREADA: **n0d3pm**

Contador de mensaies:

Figura 3.9: Contador de mensajes recibidos para el Caso 1.

Si los mensajes que se envían desde el servidor de SmartSantander no consiguen llegar al servidor de Node-RED (por ejemplo, por el NAT de un router), la subscripción quedará inactiva. SmartSantander detectará que nuestro servidor no es capaz de establecer una sesión TCP o no responde con un mensaje HTTP de tipo 2XX, y desactivará la subscripción.

## Interconexión de bloques con la UI en Node-RED

Para que todas las funcionalidades de la UI se ejecuten, es necesario tener bloques que representen todas las entidades visuales en la UI (y sus funciones). Por ejemplo, un botón se representará como un bloque con sus entradas y salidas (para el flujo de información). Además, esos bloques necesitan conexiones a terceros que establezcan tareas clave como mandar una petición HTTP cuando se pulsa un botón.

Antes de actualizar la lista de fenómenos, primero, es necesario modificar un bloque del flujo. Este se encarga de establecer las coordenadas y el radio para filtrar de qué sensores se desea recibir medidas. Esto se puede hacer también con campos

nuevos en el formulario, pero usando un mapa se vuelve una tarea más enfocada a la programación visual. Una vez se selecciona el lugar y el radio, las coordenadas (latitud y longitud) y el radio en metros puede verse en la UI (representados por los tres últimos bloques azules en la Figura 3.10). Para modificar la localización se debe modificar el bloque y volver a desplegar el flujo.

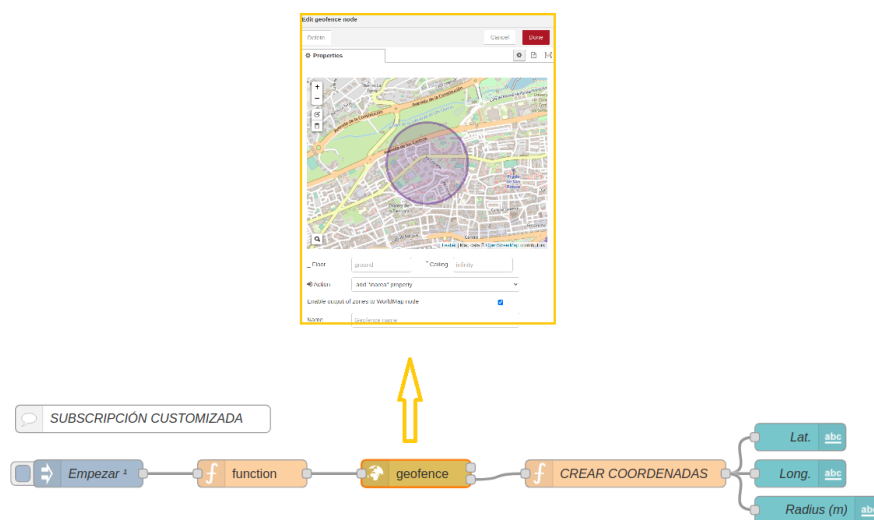


Figura 3.10: Configuración de la localización y radio de filtrado para la suscripción.

Para actualizar la lista de fenómenos soportados por SmartSantander, el botón “ACT.” se comunica con terceros nodos del mismo flujo, como se describe en la Figura 3.11. Como puede observarse, se puede activar con un botón (“ACT.”) o con un inyector (“CONS. FENOMENOS”). Una vez se activa el flujo, se manda una señal al cliente HTTP que crea una petición y recibe todos los fenómenos que forman parte de la API de SmartSantander. Una vez se recibe la información en el cliente HTTP, se pasa el mensaje a una función que genera la lista de elementos para el bloque que presenta una lista desplegable en la UI.

Una vez se selecciona el fenómeno en la UI, el flujo de Node-RED automatiza una petición de las unidades correspondientes al fenómeno. Como se puede ver en la Figura 3.12, el bloque “PHENOMENON” está conectado a un segundo cliente

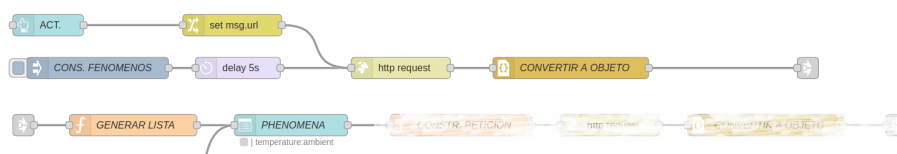


Figura 3.11: Flujo de inicio para el caso y recolección del listado de fenómenos.

HTTP que es el encargado de hacer la petición por la que se obtienen las unidades que corresponden con al fenómeno seleccionado. La respuesta a dicha petición, se envía a una función que genera una segunda lista desplegable. El resultado de esa lista así como el resto de campos del formulario se conectan a la vez a una función que se encarga de generar variables globales (de flujo) que guardan los valores seleccionados e introducidos en los campos.

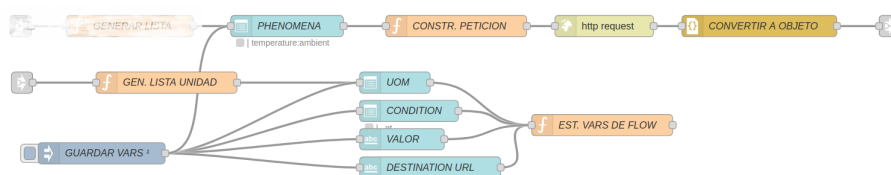


Figura 3.12: Petición de las unidades del fenómeno elegido y definición de variables globales.

Una vez se hace click sobre el botón “SUBSCRIBIRSE” (Figura 3.13) o se activa manualmente con un inyector, se genera la carga útil (o *payload* a partir de ahora) de la petición. En este payload (en formato JSON) se incluyen todos los parámetros recogidos a través de la UI para suscribirse a un fenómeno. Una vez el bloque que representa el cliente HTTP recibe la petición, la envía al servidor de SmartSantander.

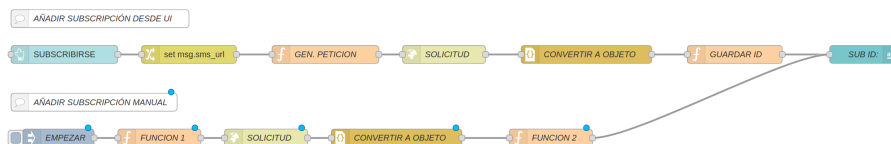


Figura 3.13: Bloques del envío de la petición a SmartSantander.

Una vez la suscripción se ha creado correctamente en el servidor de SmartSantander, los desarrolladores pueden hacer click sobre el botón “ACT. LISTA” para

que el servidor adquiriera todas las suscripciones ya creadas por dicho usuario. Este evento también se puede desencadenar manualmente con un inyector. La respuesta del servidor se incluirá en una lista desplegable para que los desarrolladores puedan comprobar el estado de la suscripción, como se muestra en la Figura 3.14.

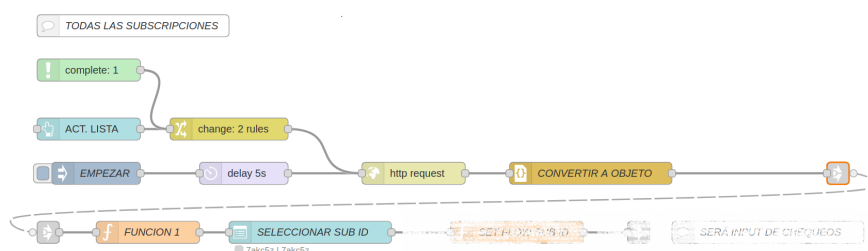


Figura 3.14: Automatización del proceso de actualización del listado de suscripciones.

Una vez se selecciona la suscripción a comprobar en el listado desplegable, el servidor en Node-RED vuelve a enviar una petición al servidor de SmartSantander para comprobar su estado. Una luz verde o roja se iluminará, dependiendo de si su estado es habilitado o deshabilitado, respectivamente. Esta comprobación, que se muestra en la Figura 3.15, se puede hacer cada vez que se pulsa un botón o si se habilita el botón de tipo *switch*. El botón *switch* habilita un *trigger* o disparador en Node-RED que repite la consulta cada 5 segundos, por lo que se consigue una comprobación automatizada que permitirá tener información en tiempo real sobre el estado de la suscripción. Esta es una herramienta útil para depurar si los mensajes de SmartSantander llegan o no al servidor de Node-RED, o si la suscripción ha quedado deshabilitada por algún error.

Tal y como muestra la Figura 3.16, es posible activar o habilitar la suscripción (si no lo está ya). Para ello se ha desplegado un botón (“HABILITAR”) para que el servidor de Node-RED mande una petición al servidor de SmartSantander. Una vez procesada la respuesta a la petición en Node-RED, se empezarán a enviar paquetes desde SmartSantander, recibidos por el bloque que actúa como servidor HTTP en Node-RED.

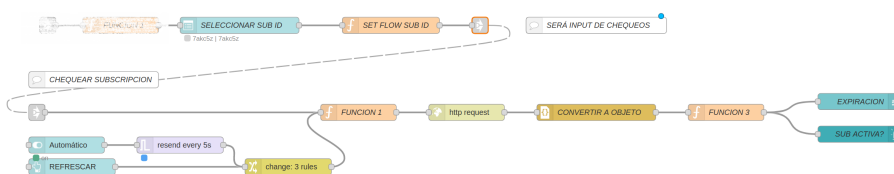


Figura 3.15: Flujo para la comprobación manual o automática del estado de la suscripción.

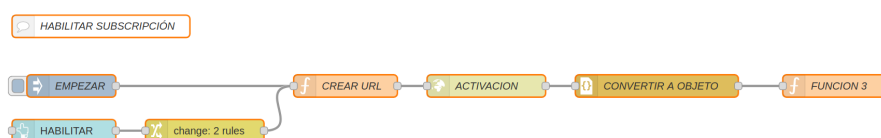


Figura 3.16: Flujo para habilitar la suscripción a SmartSantander.

Por último, es importante mencionar que también es posible eliminar suscripciones con la UI o con bloques inyectores, como puede observarse en la Figura 3.17. Pulsando el botón “ELIMINAR” se envía una petición al servidor de SmartSantander para eliminar la suscripción. Si se actualizan las suscripciones, el listado desplegable no incluirá la suscripción que se acaba de eliminar.

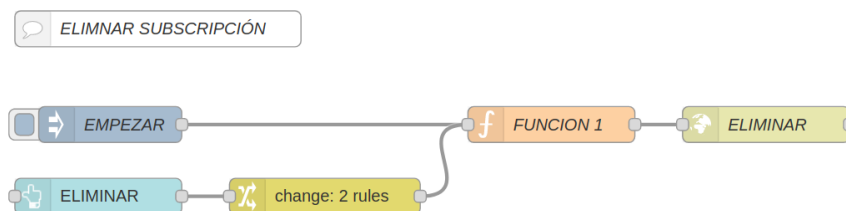


Figura 3.17: Flujo de Node-RED para eliminar la suscripción seleccionada de la lista desplegable.

## Generación de datos

En este caso, la generación de los datos ocurre gracias a sensores y dispositivos IoT desplegados en la ciudad de Santander. No es necesario, por lo tanto, generar



ningún tipo de dato emulado para poder testear el servidor de Node-RED. Para este caso, se pueden adquirir datos de todos los sensores desplegados en la ciudad y que sean parte de la infraestructura. La Figura 3.18 muestra parte de la infraestructura de SmartSantander, de la cual se puede conseguir información.



Figura 3.18: Mapa de la infraestructura IoT de SmartSantander.

### Recepción, extracción y adaptación de los datos de sensores

Una vez se ha explicado cómo crear una suscripción, cómo se monitoriza si una suscripción sigue activa y cómo eliminarla, la parte de recepción de datos es relativamente diferente a estas. El servidor de SmartSantander envía las medidas cuando se cumplen los requisitos establecidos en la suscripción. El protocolo usado es HTTP (definido en la URL de destino de la suscripción), por lo que se necesita configurar un servidor HTTP. De hecho, hasta ahora, todas las funciones de comunicación con SmartSantander se han hecho con clientes HTTP. Para recibir la información de cada sensor desde SmartSantander, el bloque de Node-RED debe funcionar como un servidor. Este bloque se conoce como *http in*. La forma en la que los bloques que funcionan como servidores HTTP puedan escuchar peticiones, se basa en la URL que se define

como un parámetro de configuración dentro del bloque en Node-RED. En particular, se define la ruta que identifica a nuestro servidor. Por ejemplo, la ruta “/sms” de todos los paquetes que lleguen al puerto 1880 de Node-RED serán enrutados al bloque que lo había definido a la hora de desplegar el flujo.

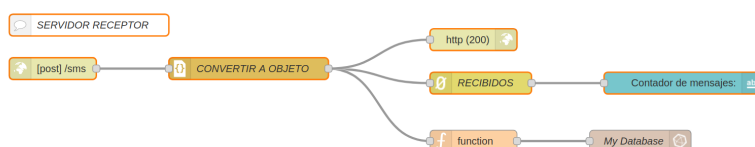


Figura 3.19: Bloque de recepción y adaptación de datos para el Caso 1.

Como puede observarse en la Figura 3.19, el servidor se encarga de recibir los datos que envía SmartSantander, y extraer el payload del paquete HTTP. El siguiente bloque del flujo convierte la cadena de caracteres con formato JSON a un objeto de JavaScript para que la siguiente función del flujo pueda procesarlo. Esta función extrae los valores que envía el sensor así como los metadatos que envía el servidor de SmartSantander como parte del mensaje (p. ej., una marca de tiempo (*timestamp*), o el ID de la suscripción que desencadenó el mensaje).

Una parte importante del servidor implica responder con un mensaje que incluya un código 2XX cuando se reciben correctamente los datos desde SmartSantander. El bloque encargado de ello puede verse también en la Figura 3.19 y se denomina *http response*. Este paso es necesario, ya que si no se confirma con un código de estado (*status code*) de tipo “2xx: Peticiones correctas”, se deshabilitará la suscripción después de varios mensajes enviados.

## Base de datos

Una vez se extraen los datos del paquete HTTP que llega al servidor en Node-RED, se debe adaptar a la base de datos. InfluxDB tiene una estructura muy particular.

Los conceptos más relevantes en InfluxDB son *bucket* (se usará este término de ahora en adelante), medidas (*measurements*), etiquetas (*tags*) y campos (*fields*). Dado que InfluxDB no tiene una estructura como la que tienen las bases de datos relacionales (con tablas) o incluso bases de datos flexibles como NoSQL (con documentos), se debe adaptar la información. La función denominada “ADAPTADOR” se encarga de extraer etiquetas como la latitud, la longitud, la unidad de medida, el ID del sensor y el tiempo en el que se hizo la medida. En el Listado 3.2 puede observarse una representación del mensaje enviado a la base de datos.

```
1  [
2    {
3      "temperature:ambient":18.7
4    },
5    {
6      "id":"urn:x-iot:smartsantander:u7jcfa:t453",
7      "unit":"degreeCelsius",
8      "measured_time":"2021-09-03T01:24:59.698020+02:00",
9      "latitude":-3.80402,
10     "longitude":43.46212
11   }
12 ]
```

Listado 3.2: Mensaje del nodo “ADAPTADOR” a la base de datos InfluxDB

Como el campo más relevante, se guarda el nombre del fenómeno y su valor como una combinación de clave-valor (primer objeto del *array* en el Listado 3.2). Antes de insertar el campo y las etiquetas, se debe establecer cuál será el bucket y el nombre de la medida. Ambos parámetros se pueden establecer como parte de la configuración inicial del bloque que representa la base de datos en el flujo de Node-RED. También se puede definir mediante lenguaje JavaScript, en una de las funciones previas al bloque de la base de datos.

Es necesario explicar que la primera vez que se establece el bloque de la base de datos InfluxDB en el flujo se deben configurar dos parámetros: la URL de la base de datos y el *token*. Estos parámetros, así como el bucket, se han preestablecido y

configurado en el despliegue con Docker Compose. En la Figura 3.20 pueden verse los parámetros para configurar el bloque de InfluxDB en Node-RED (recuadro izquierdo), así como los parámetros necesarios a la hora de insertar información (recuadro derecho).

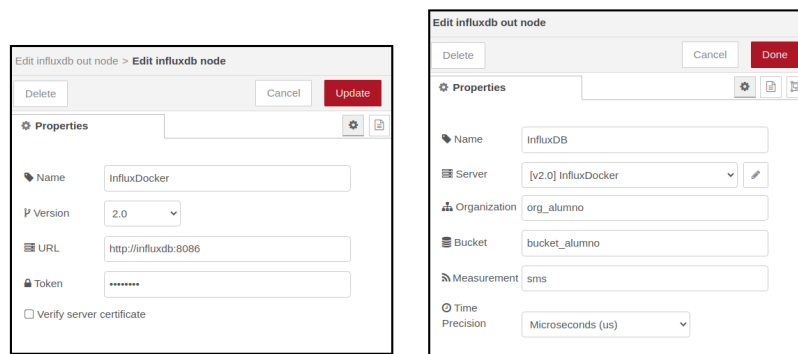


Figura 3.20: Parámetros de configuración para InfluxDB en el Caso 1.

### 3.2.2. Caso 2: hook HTTP para el Orion Context Broker

Este caso es similar al Caso 1, porque el sistema de suscripción y adquisición de datos es análogo (salvando las diferencias en cuanto a la API que ofrecen ambas plataformas). No se pretende repetir las mismas explicaciones que en el caso anterior, pues los conceptos más importantes ya se han descrito. Sin embargo, es necesario advertir los retos, dificultades y desarrollos específicos que se han hecho para este caso de uso.

#### Interfaz de usuario

Para este caso de uso también se ha desarrollado una interfaz de usuario como en el Caso 1, aunque con ciertas diferencias que se pueden observar en la Figura 3.21. Sin embargo, esta es más simple que la anterior. Para poder suscribirse a la información de contexto, Orion ofrece la oportunidad de enviar una petición para evitar un *polling* continuo. Los campos disponibles para rellenar en la interfaz de usuario

se limitan, principalmente, a la (des)suscripción y monitorización de paquetes recibidos. Rellenando los parámetros de la suscripción, el servidor enviará una petición al servidor de contexto Orion. Cuando un sensor con ID y tipo iguales a los indicados envíe una nueva medida y el valor de la condición (por ejemplo la presión o *pressure*) haya cambiado, se enviará un mensaje con el valor del atributo que se haya incluido en la petición a la URL de destino.

The screenshot shows a web form titled "ORION" for configuring a subscription. The form includes the following fields and controls:

- SELECCIONAR SUB ID:** A dropdown menu with "Select option" and a blue "ACT. LISTA" button.
- URL acumulador:** A text field containing "http://172.18.0.5:1880/orion".
- ID de la Entidad:** A text field containing "Room1".
- Tipo de la Entidad:** A text field containing "Room".
- Condición:** A text field containing "pressure".
- Atributos:** A text field containing "temperature".
- Fecha de expiración:** A date field containing "09/24/2021" with a calendar icon.
- Descripción:** A text field containing "HHMM" and "--:-- --" with a clear icon.
- At the bottom, there are two blue buttons: "ENVIAR" and "CANCELAR".

Below the form, there is a "Contador de mensajes:" label and a status message at the bottom: "ULT. SUB CREADA: 61327177629f030f327d116b".

Figura 3.21: Parámetros de configuración de una nueva suscripción para Orion.

La condición que se incluye en la petición define el criterio que se debe interpretar cuando un valor cambia. Es decir, si se define un atributo como *pressure* como la condición, Orion interpretará que cada vez que el valor de *pressure* cambie, se deberá enviar un mensaje. Este mensaje incluirá uno o más atributos incorporados (como *temperature*) en el campo "Atributo" del formulario previamente mostrado. Por lo tanto, cuando el valor de *pressure* cambie, Orion enviará la información de *temperature* a la URL de destino.

Una vez se envía la petición, es posible actualizar la lista desplegable de IDs de las subscripciones. Pulsando el botón “ACT. LISTA” se podrán observar las subscripciones activas, como se muestra en la Figura 3.22.

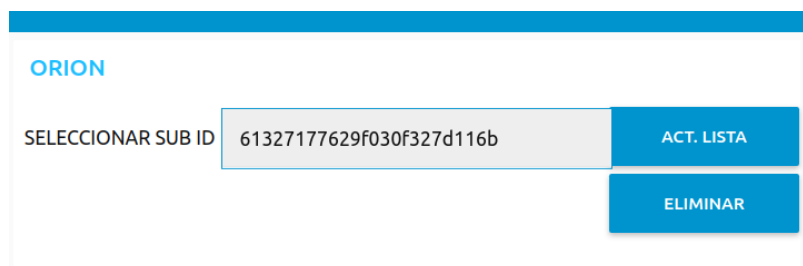


Figura 3.22: Listado de subscripciones enviadas por el servidor de contexto Orion.

Una vez que se actualiza y selecciona el ID de subscripción, es también posible eliminarla. No se ha creado la interfaz para activar y desactivar la subscripción, pero existe una API propia de Orion para ello.

### Interconexión de bloques con la UI en Node-RED

Cuando se han introducido los valores apropiados en todos los campos del formulario, el flujo que se describe en la Figura 3.23 se encarga de enviarlo a Orion.

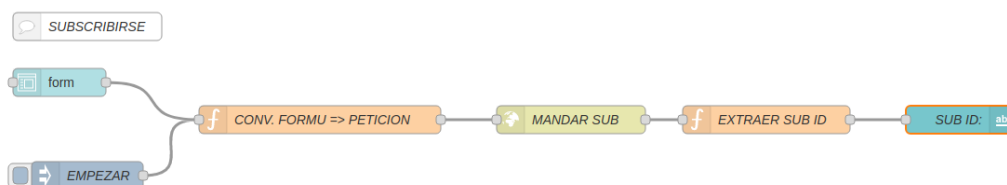


Figura 3.23: Nodos en Node-RED para subscribirse en Orion.

La función que se encuentra después del cliente HTTP se encarga de extraer el ID de la subscripción que se acaba de crear. Ese ID se muestra en la parte inferior de la

UI como se observa en la Figura 3.21. Para actualizar la lista de suscripciones, la UI ofrece un botón que, al pulsarlo, desencadena el flujo que se muestra a continuación en la Figura 3.24.

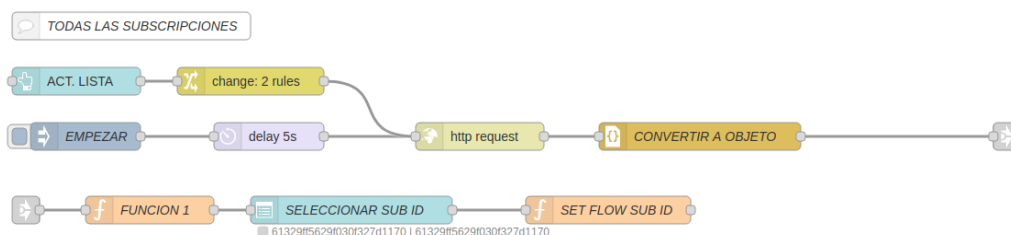


Figura 3.24: Flujo que actualiza la lista de suscripciones de la entidad que envíe la petición.

Por último, la UI ofrece la posibilidad de eliminar una suscripción. Como se ha mencionado antes, se puede desactivar en vez de eliminar una suscripción, pero no está actualmente implementado. El siguiente flujo en la Figura 3.25 representa los bloques que se encargan de generar la petición HTTP para eliminar una suscripción.

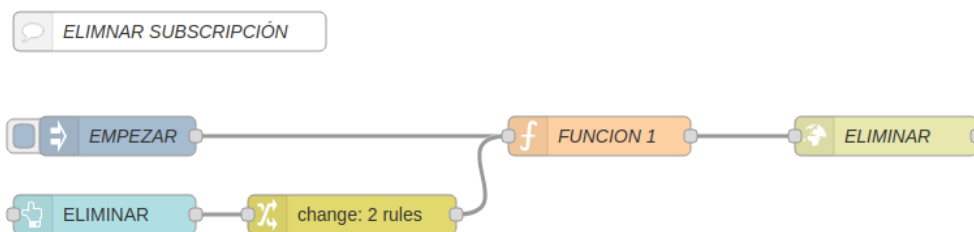


Figura 3.25: Flujo que incluye un inyector o botón que desencadena la eliminación de una suscripción.

## Generación de datos

Dado que el servidor Orion se ha desplegado localmente, se necesita generar datos que representen hipotéticas medidas hechas por sensores. En este caso, se ha creado un programa o *script* que genera una serie de datos emulados como la temperatura

o la presión atmosférica monitorizadas por un conjunto de 10 sensores dentro de una vivienda virtual. Cada cierto tiempo (5 o 10 segundos) se elige, aleatoriamente, una de las varias zonas de la vivienda como el tipo de la entidad. En ese momento, se generan los valores asociados al sensor de esa zona. Por ejemplo, si el script selecciona la zona del baño aleatoriamente, se generan valores para la temperatura y humedad dentro de un rango predefinido. Una vez que se generan los datos se forma la petición que se envía a Orion, como si de un sensor se tratase.

Dado que Orion se despliega junto a un contenedor de MongoDB, se puede verificar que se han generado entidades correspondientes a todas las zonas de la casa emulada. Estas entidades se crean a partir de las peticiones POST que el script crea al ejecutarse al principio y que envía a Orion, como muestra el Listado 3.3. Después el script genera peticiones PATCH para actualizar el estado de algunos atributos, tal y como muestra el Listado 3.4. La petición PATCH implica que Orion realice una actualización de los valores de los atributos que son parte de la petición. Por ejemplo, al actualizarse la temperatura y la humedad, los valores asociados a estos dos atributos deberán ser actualizados en la base datos MongoDB.

```
1 {
2   "id": "bathroom2",
3   "type": "bathroom",
4   "temperature": {
5     "value": 21.11,
6     "type": "Float"
7   },
8   "humidity": {
9     "value": 52.14,
10    "type": "Float"
11  }
12 }
```

Listado 3.3: Payload del mensaje POST enviado a Orion.

```
1 [
2   {
3     "temperature": {
4       "value": 21,
5       "type": "Float"
6     },
7     "humidity": {
8       "value": 53.72,
9       "type": "Float"
10    }
11  }
12 ]
```

Listado 3.4: Payload del mensaje PATCH enviado a Orion.

En la Figura 3.26, se puede comprobar cómo se han creado las colecciones de



subscripciones (*csubs*) y las de las entidades (*entities*).



Figura 3.26: Captura de las colecciones que Orion crea en la base de datos MongoDB.

Asimismo, en la Figura 3.27 se puede observar el total de documentos introducidos. En el Listado 3.5, también se muestra el documento exacto, tal y como se almacena en MongoDB después de que Orion inserte los datos.

entities > attrNames					
_id	attrNames	attrs	creDate	modDate	lastCorrelator
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630691413.8947942	1630708257.26	9b91f32e-0d06-11ec-9833-0242ac120007
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630716390.0652034	1630717285.47	a0cd9932-0d1b-11ec-b25c-0242ac120007
{ 3 fields }	[ 3 elements ]	{ 3 fields }	1630716390.0934403	1630717324.34	b7f83ac2-0d1b-11ec-b0cf-0242ac120007
{ 3 fields }	[ 3 elements ]	{ 3 fields }	1630716390.1021705	1630717220.20	79e63ce8-0d1b-11ec-bbe2-0242ac120007
{ 3 fields }	[ 3 elements ]	{ 3 fields }	1630716390.109675	1630717280.45	9dcfc75a-0d1b-11ec-a028-0242ac120007
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630716390.1150568	1630716707.79	487992fa-0d1a-11ec-8136-0242ac120007
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630716390.1242628	1630716697.77	4280f47e-0d1a-11ec-a2f5-0242ac120007
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630716743.2708843	1630717240.30	85dffdfc-0d1b-11ec-a6d0-0242ac120007
{ 3 fields }	[ 2 elements ]	{ 2 fields }	1630716743.2775588	1630717334.38	bd4f226-0d1b-11ec-b101-0242ac120007

Figura 3.27: Atributos de las entidades insertados por Orion en la base de datos MongoDB.

```

1  {
2      "_id" : {
3          "id" : "Room1",
4          "type" : "Room",
5          "servicePath" : "/"
6      },
7      "attrNames" : [
8          "temperature",
9          "pressure"
10     ],
11     "attrs" : {
12         "temperature" : {

```

```
13     "value" : 26.07,  
14     "type" : "Float",  
15     "mdNames" : [  
16  
17     ],  
18     "creDate" : 1630691413.8947942,  
19     "modDate" : 1630708257.2656958  
20 },  
21 "pressure" : {  
22     "value" : 883.0,  
23     "type" : "Float",  
24     "mdNames" : [  
25  
26     ],  
27     "creDate" : 1630691413.8947942,  
28     "modDate" : 1630708257.2657006  
29 }  
30 },  
31 "creDate" : 1630691413.8947942,  
32 "modDate" : 1630708257.2657082,  
33 "lastCorrelator" : "9b91f32e-0d06-11ec-9833-0242ac120007"  
84 }
```

Listado 3.5: Contenido en formato JSON almacenado por MongoDB en lo que se refiere a la información sobre Room1.

### Recepción, extracción y adaptación de los datos de sensores

Este proceso tiene un funcionamiento muy parecido al del Caso 1, que ya disponía de datos con un formato JSON. En este caso, no se usa el modelo de información utilizado por SmartSantander, pero se utiliza uno parecido, ya que los datos distribuidos por Orion Context Broker se basan en el estándar NGSIv2. Dado que el Orion Context Broker proporciona una modalidad de suscripción, los clientes no necesitan consultar (*polling*) la información cada cierto tiempo, sino que Orion actúa como cliente HTTP para enviar la información cuando ciertas condiciones se cumplen. Como puede verse en la Figura 3.28, un servidor HTTP recibe los mensajes de Orion y se encarga de transformar la información, contabilizar los mensajes e insertar la información en InfluxDB.

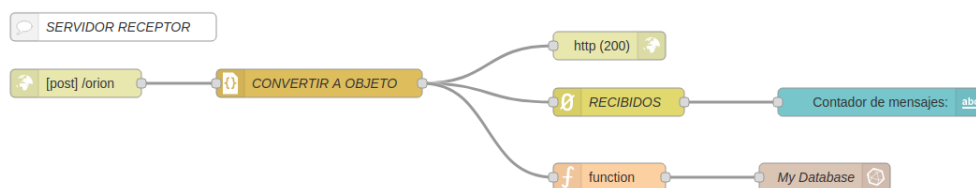


Figura 3.28: Flujo de nodos que reciben mensaje de Orion, adaptan el payload y lo almacenan en InfluxDB.

Es necesario mencionar que la función “CONVERTSOR” adapta el mensaje de Orion a la estructura que InfluxDB espera, particularmente, los campos y las etiquetas. El bucket, así como la medida, se definen en el bloque de InfluxDB, pero se pueden editar en una función JavaScript como parámetros de la variable *msg*.

### Base de datos

Para este caso existe una base de datos MongoDB que es parte del despliegue estructural del Orion Context Broker. Sin embargo, en esta sección solo se explicará la función de InfluxDB como parte principal del flujo del servidor. Para adaptar la información que llega del broker Orion (Listado 3.6), el servidor extrae los componentes más relevantes y crea la estructura que envía a InfluxDB (Listado 3.7).

### 3.2.3. Caso 3: cliente MQTT para Lora

Este caso es diferente a los dos anteriores, ya que se utiliza un protocolo distinto en la capa de aplicación. Además, se han usado nodos LoRa para enviar los datos a la nube. Asimismo, el número de bloques en Node-RED disminuye, ya que no es necesaria una extensa interfaz de usuario.

### Interfaz de usuario

A diferencia de los casos anteriores (Caso 1 y Caso 2), no es necesario generar una suscripción del mismo tipo que la del servidor de SmartSantander u Orion.

```

1  {
2  "subscriptionId":
3  ↵ "6132...1171",
4  "data": [
5  {
6  "id": "bedroom1",
7  "type": "bedroom",
8  "temperature": {
9  "type": "Float",
10 "value": 19.75,
11 "metadata": {}
12 }
13 ]
14 }

```

Listado 3.6: Mensaje JSON que se recibe en el servidor HTTP del Caso 2.

```

1  [
2  {
3  "value": 19.75
4  },
5  {
6  "topic": "temperature",
7  "id": "bedroom1"
8  }
9  ]

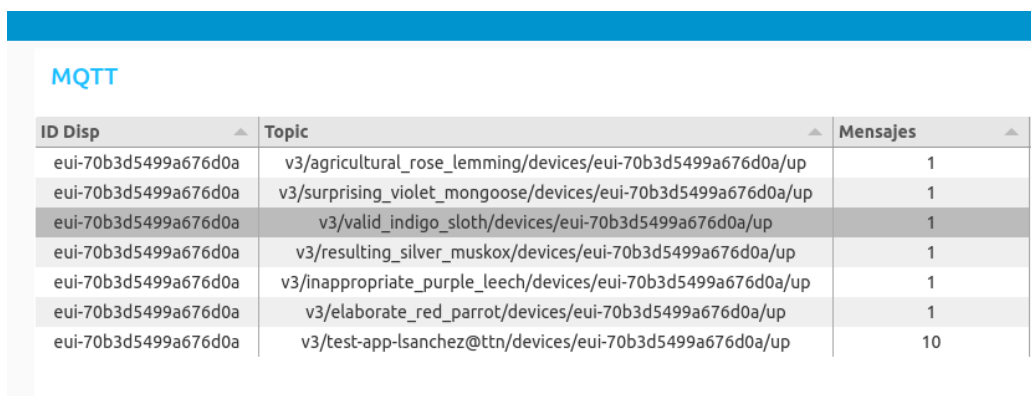
```

Listado 3.7: Transformación de la estructura del mensaje para InfluxDB

Aunque es cierto que el cliente MQTT en el flujo de Node-RED necesita subscribirse a un tema (*topic*). Como los temas de MQTT siguen una jerarquía, el cliente MQTT en Node-RED permite subscribirse a varios temas usando diferentes comodines (*wildcards*) para seleccionar uno o varios niveles jerárquicos. Por ejemplo, subscribiéndose a “v3/+/devices/+/up”, el cliente se subscribe a cualquier tema de segundo y cuarto nivel. Por eso recibiría, asíncronamente, mensajes de temas como “v3/usuario1/devices/dispositivo1/up” o “v3/usuario3/devices/dispositivo2/up”.

El problema es que el cliente de MQTT en Node-RED no acepta una entrada para poder definir diferentes subscripciones, sino que, depende primero de la jerarquía de los temas y también en el uso de comodines para cubrir diferentes casos de uso. Para varias subscripciones con diferentes temas no correlacionables con comodines, se usarán más bloques de clientes MQTT. Por esa razón, no es necesaria una interfaz muy compleja dado que gran parte de la configuración se hace en los bloques MQTT propios de Node-RED.

Sin embargo, se ha creado una tabla que se actualiza con cada mensaje nuevo. Por cada combinación de ID de dispositivo y tema, se muestra un contador de mensajes recibidos tal y como se ve en la Figura 3.29.



ID Disp	Topic	Mensajes
eui-70b3d5499a676d0a	v3/agricultural_rose_lemming/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/surprising_violet_mongoose/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/valid_indigo_sloth/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/resulting_silver_muskox/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/inappropriate_purple_leech/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/elaborate_red_parrot/devices/eui-70b3d5499a676d0a/up	1
eui-70b3d5499a676d0a	v3/test-app-lsanchez@ttn/devices/eui-70b3d5499a676d0a/up	10

Figura 3.29: Tabla de mensajes MQTT recibidos por cada ID y tema en el Caso 3.

### Interconexión de bloques con la UI en Node-RED

La única interconexión entre la UI y el flujo, es la que se produce con la tabla. Cuando el cliente MQTT recibe un mensaje desde el broker MQTT, el mismo se procesa y se extraen dos elementos clave: el ID del dispositivo y el tema. Ambos forman una clave única para contar mensajes. Se podría generar una clave aún más larga con otros parámetros incluidos en el mensaje MQTT desde el broker si se pretende extender el caso.



Figura 3.30: Nodos que cuentan los mensajes recibidos y los muestran en la UI.

Como se muestra en la Figura 3.30, el bloque “CONTADOR” se encarga de generar una clave única por ID de dispositivo y tema y guardarlo como variable global como parte del flujo. La función es capaz de detectar si la clave única se ha detectado

anteriormente y de generar un contador para ella si no existe. El valor del contador también se guarda como variable global como parte del flujo.

### Generación de datos

Se desplegó un conjunto de nodos Lora que generasen datos y se enviasen mediante gateways LoRa. Sin embargo, también se ha incluido un cliente emulado como parte del flujo en Node-RED (Figura 3.31). Este solo es necesario cuando no se tenga acceso a nodos o sensores LoRa (y su nube que soporte MQTT) o para cuando se necesite generar datos continuamente. Junto al emulador, se usa el broker público de HiveMQ [25] para publicar los datos desde el sensor emulado, aunque es posible desplegar un broker en el propio Node-RED si se quiere extender el caso de uso.

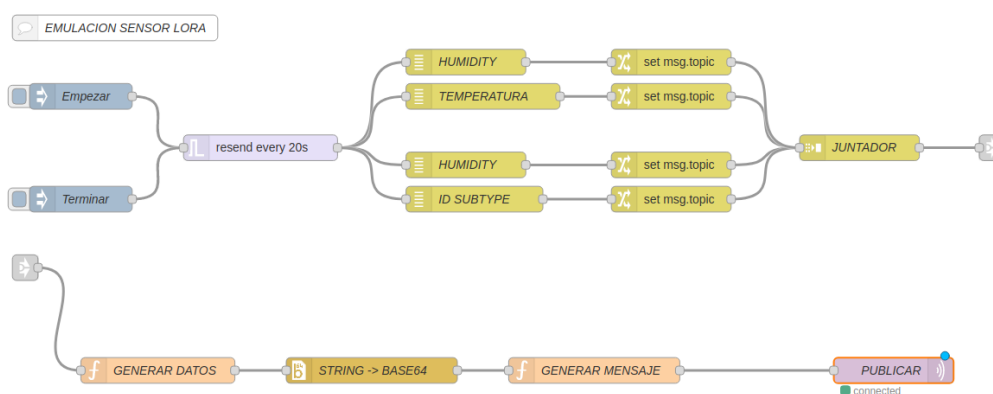


Figura 3.31: Sensor emulado que envía datos usando MQTT.

### Recepción, extracción y adaptación de los datos de sensores

Para recibir los datos del broker MQTT se ha añadido un cliente MQTT que se suscribe a un tema en particular, como se muestra en la Figura 3.32. Cuando se recibe en el broker un mensaje que coincide con el tema al que se ha suscrito, este se envía al cliente MQTT. Una vez se recibe en el cliente, estos datos se pasan a varios conversores. El primer conversor (“CONVERTIR A OBJETO”), convierte el payload en JSON a un objeto JavaScript. Con esto se consigue que funciones de Node-RED

puedan leer y editar los campos. El segundo bloque de conversión, convierte una parte del payload que envió el sensor, en base64, a cadena de caracteres (que tiene formato JSON). En este caso particular, aunque ya se había convertido el mensaje del broker a tipo objeto, los datos del sensor estaban codificados en base64 y necesitan ser decodificados (una vez más) a una cadena de caracteres. Por último, el segundo bloque de conversión “CONVERTIR A OBJETO” se encarga de convertir solo la parte que se había convertido de base64 a cadena de caracteres. Una vez se tiene el mensaje completo del broker y también los datos del sensor en formato objeto de JavaScript, la función “ADAPTADOR” lo convierte al objeto que la base datos espera.

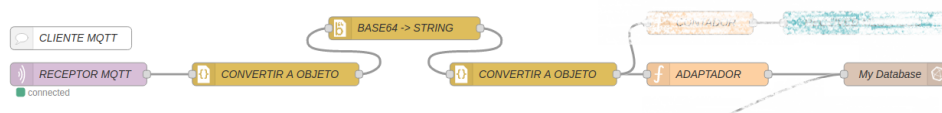


Figura 3.32: Consumidor MQTT suscrito a broker de los sensores LoRa.

## Base de datos

Tanto en este como en los dos casos anteriores, se sigue utilizando InfluxDB. En este paso, lo importante es convertir el mensaje recibido en el cliente MQTT a un formato que sirva para la base datos. Como se ha explicado en la sección anterior, los datos que originalmente envía el sensor LoRA llegan finalmente al cliente MQTT codificados en base64. Una vez se obtiene el formato JSON y su estructura en JavaScript, la función “ADAPTADOR” introduce todos los atributos como campos, y la dirección del dispositivo, su ID y el tema como etiquetas.

La creación del objeto de envío para InfluxDB puede verse en el Listado 3.8.

```

1   msg.payload = [{
2     illuminance : msg.payload.uplink_message.frm_payload.illuminance.v,
3     atmosphericPressure :
4     ↪ msg.payload.uplink_message.frm_payload.atmosphericPressure.v,
5     relativeHumidity :
6     ↪ msg.payload.uplink_message.frm_payload.relativeHumidity.v,

```

```
5     temperature : msg.payload.uplink_message.frm_payload.temperature.v
6   },
7   {
8     dev_addr: msg.payload.end_device_ids.dev_addr,
9     dev_eui: msg.payload.end_device_ids.device_id,
10    topic: msg.topic,
11    latitude: msg.payload.uplink_message.frm_payload.latitude,
12    longitude: msg.payload.uplink_message.frm_payload.longitude,
13
14  }];
15
16  msg.measurement = "sanchez_LoRa"
```

Listado 3.8: Función en JavaScript para formar el mensaje enviado a InfluxDB partiendo del mensaje de los sensores LoRa.

### 3.2.4. Caso 4: cliente activo de HTTP (polling)

En este último caso, se ha implementado el proceso más básico en el que un cliente HTTP consulta los datos de un servidor cada cierto tiempo. En los últimos años, con el desarrollo de utilidades y servidores de contextualización como Orion, se ha pasado a un modo asíncrono. Aun así, el HTTP polling es un modo ampliamente extendido y que es ampliamente soportado por servidores IoT. Las siguientes secciones describirán 3 fuentes de datos y casos de uso diferentes en los que se hace, de manera proactiva, la consulta explícita de la información y cómo se asocia esto a la generación y al almacenamiento. Por lo tanto, se analizarán 3 fuentes: el repositorio de Datos Abiertos de Santander (1), el repositorio de Datos Abiertos de Aarhus con información de tráfico (2) y OpenWeatherMap para consultar la información meteorológica (3).

#### Interfaz de usuario

En esta ocasión se ha decidido simplificar la UI, que solo contiene la monitorización de recepción de mensajes para cada caso y una utilidad extra para elegir el lugar sobre el cual consultar la información de la tercera fuente (3). Para el repositorio



(1) se provee un contador de mensajes recibidos. Para el repositorio (2) se ofrece un selector de sensor entre todos los sensores de tráfico en Aarhus. Como puede verse en la Figura 3.33, existen diferentes utilidades de UI para cada repositorio.

The screenshot shows a mobile application interface with a blue header labeled "PULL". It displays three data sources:

- SMARTSANTANDER OPENDATA**: Shows a message counter of 283.
- AARHUS REALTID TRAFFIK**: Features a sensor selector dropdown menu currently set to "7 -> Hasselager - Genvejen" and a message counter of 108.
- OPENWEATHERMAP**: Includes a configuration section with fields for "Ciudad" (Amsterdam), "Pais" (NL), "Petición cada (s)" (30), and "Llave API" (0da00d39614cefcf872d804c965877f7). Below these fields are "CONFIGURAR" and "CANCELAR" buttons, and a red "X DETENER" button.

At the bottom, a summary table shows the current state:

Ciudad	Pais	Mensajes
Amsterdam	NL	88

Figura 3.33: Interfaz de usuario completa del Caso 4.

### Interconexión de bloques con la UI en Node-RED

Para que el repositorio de Datos Abiertos de Santander (1) muestre los mensajes recibidos, existe un contador y un visualizador de texto que muestra cada uno de los mensajes. Esta petición se repite cada ciertos segundos, que son configurables en el nodo disparador (no mediante la UI). Se puede observar este proceso en la Figura 3.34:

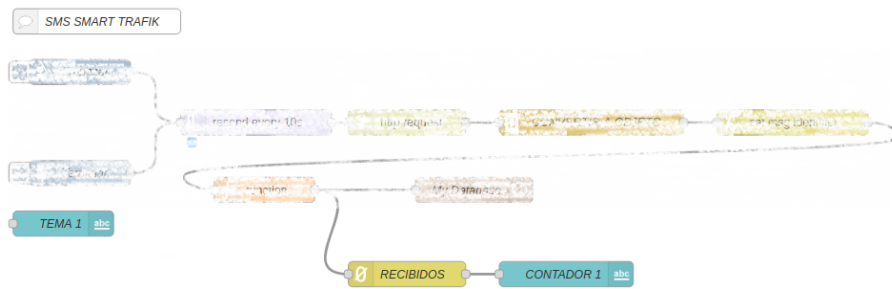


Figura 3.34: Nodo contador de mensajes y bloque UI de la fuente de datos (1).

Para que la UI del repositorio (2) ofrezca el listado de sensores, la Figura 3.35 muestra cómo se hace una petición al servidor y después se extraen del CSV todos los IDs de los sensores y su ubicación.

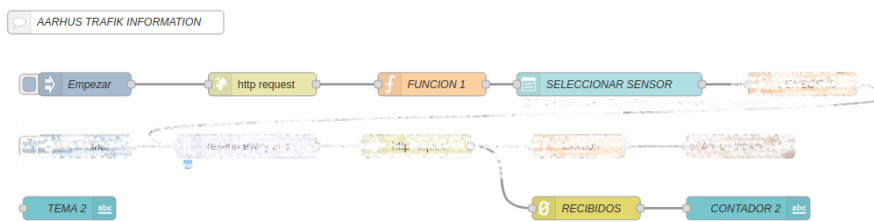


Figura 3.35: Nodos que hacen una petición de los sensores en Aarhus para la fuente (2).

Tal y como muestra la Figura 3.35, se juntan tres parámetros creando un ID personalizado y se crea una lista con todos los sensores para poder mostrarla en una lista desplegable, representada en la Figura 3.36. Además de esta información, se muestra un contador con todos los mensajes que se reciben del sensor de tráfico de Aarhus. Esto es útil para poder depurar si el polling HTTP funciona o no. El tiempo de actualización de las peticiones se debe modificar en el nodo disparador (o *trigger*) de Node-RED.

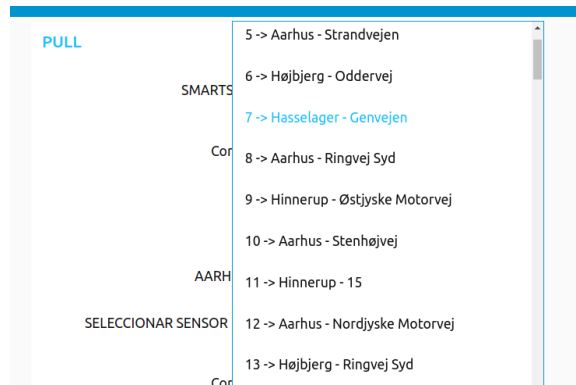


Figura 3.36: Listado de todos los sensores de tráfico en Aarhus como parte de la fuente (2).

En lo que a OpenWeatherMap se refiere (3), la lista de elementos en la UI es un poco más extensa. La UI presenta un formulario para poder subscribirse a los datos meteorológicos de una ciudad, especificando también su país. Además, se puede elegir el retardo de repetición a la hora de recolectar los datos. Esto es importante, ya que se necesita un registro gratuito para OpenWeatherMap y el máximo número de peticiones gratuitas es de 60 por minuto. El desarrollador debe calcular cada cuanto tiempo necesita enviar peticiones para no sobrepasar el límite (ratio menor o igual a 1 petición/s).



Figura 3.37: Nodos que adquieren los parámetros del formulario, detienen las consultas y cuentan los mensajes recibidos.

La primera línea de bloques en la Figura 3.37 representa el formulario y la adaptación de los parámetros. La segunda línea, que comienza con el bloque “DETENER”, ofrece la posibilidad de parar la recolección de información presionando un botón,

algo que se debe hacer mediante el bloque inyector en los casos (1) y (2). Finalmente, el último bloque contiene un contador y una tabla. El contador es una función en JavaScript que mantiene un contador con estado (*statefull*). Esto significa que la función puede contar mensajes por cada ciudad y país a la que se la haya enviado una petición. Y puede hacerlo aunque las peticiones cambien a otras ciudades, guardando el estado entre peticiones. Se actualiza cada vez que se recibe una respuesta a una petición a OpenWeatherMap y añadirá una línea por cada nueva ciudad y país.

### Generación de datos

En ninguno de estos casos se ha necesitado una emulación de sensores, ya que son repositorios abiertos o semiabiertos los que ofrecen la información. En el caso de los datos de tráfico de Santander (1) estos miden la ocupación, volumen o velocidad media [26]. Según el repositorio abierto de información de Aarhus [27] (2), se ofrecen al rededor de 400 puntos de medida en la ciudad, que miden cuando teléfonos con Bluetooth activado pasan por ciertos puntos de medida. En lo que a la tercera fuente de datos se refiere (3), OpenWeatherMap ofrece su propio modelo de predicción meteorológica [28]. Según la compañía, adquiere los datos de diversas fuentes: datos satelitales, estaciones de usuarios, estaciones de empresas, otros modelos globales de predicción o radares meteorológicos.

### Recepción, extracción y adaptación de los datos de sensores

En este caso (Caso 4), existen varias fuentes de datos y todos los flujos tienen su propio cliente HTTP, que recopila información de los servidores de cada fuente. Para la primera fuente de Datos Abiertos de Santander (1), se usa un cliente HTTP que repite la petición cada cierto tiempo, definido por el trigger (véase la Figura 3.38). Pulsando “TERMINAR”, se para la recolección continua de datos. Una vez el servidor responde, se convierte la respuesta a objeto JavaScript y el nodo de tipo *change* (posterior a la

función) define el ID del sensor que se desea filtrar (ya que llega la información de varios).

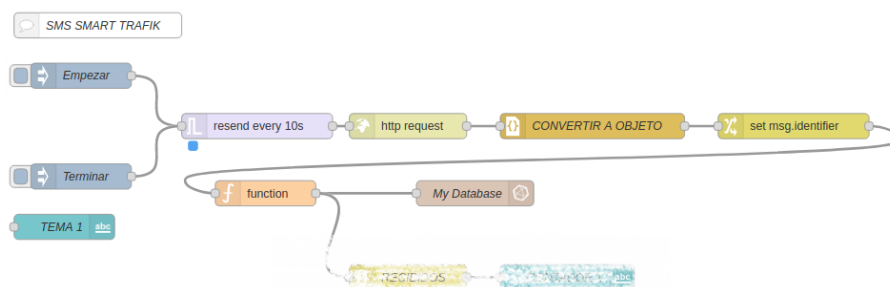


Figura 3.38: Cliente polling para la fuente (1) que adquiere datos del repositorio abierto de SmartSantander.

Como muestra el flujo de la Figura 3.39 (repositorio (2)), una vez se ha seleccionado el sensor sobre el cual recolectar la información, se envía una petición al servidor de datos de Aarhus. La respuesta se remite al siguiente bloque en el flujo, una función que se encarga de filtrar los datos de la respuesta a la petición HTTP, considerando el valor seleccionado anteriormente en el menú desplegable.

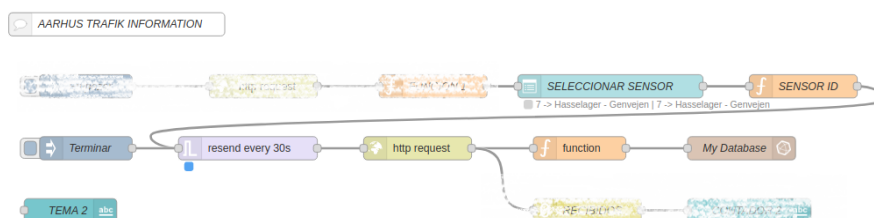


Figura 3.39: Cliente polling que adquiere datos de tráfico del servidor de Aarhus en la fuente (2).

En lo que a OpenWeatherMap (3) y la Figura 3.40 se refiere, el funcionamiento es el mismo que en los dos casos anteriores. Una vez se hace click sobre “CONFIGURAR”, el cliente HTTP envía la petición al servidor usando la API KEY del formulario. El trigger o disparador que se encuentra antes del bloque del cliente HTTP repite esta acción cada ciertos segundos, definido por el valor introducido por el usuario en el

formulario. Sin embargo, no se necesita filtrar la información, ya que los parámetros del formulario ya precisan el lugar para el cual recolectar la información meteorológica. La función posterior al cliente HTTP, se encarga de extraer información como la temperatura, humedad o presión atmosférica para guardarlas en la base de datos.

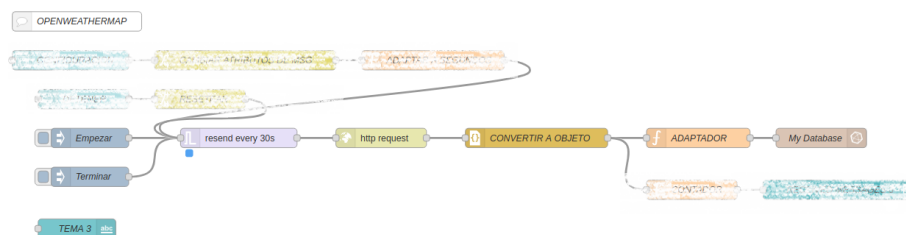


Figura 3.40: Cliente polling para OpenWeatherMap que adquiere datos meteorológicos desde la fuente (3).

## Base de datos

El uso de la base de datos es relativamente parecido en lo que a los servidores de las 3 fuentes se refiere. En todos los casos el servidor extrae la información relevante y asigna los campos y etiquetas oportunos para cada caso. Usando la primera fuente de datos (1), se usa solo la información del ID del sensor definido en el nodo change. Como campos más relevantes del sensor de tráfico, se introduce la ocupación y la velocidad media. El ID del sensor sirve como etiqueta para esta medida.

En lo que a la fuente de Aarhus (2) se refiere, se extrae la información del nodo elegido en el menú desplegable de la UI, y se obtiene información como el número de vehículos y la velocidad media como campos de la medida. El ID del sensor, que coincide con el seleccionado en el menú desplegable, será la única etiqueta de la medida que se introduce en la base de datos.

Para la última fuente de información meteorológica (3), se extraen una gran cantidad de campos relevantes para la medida, entre los que se encuentran la humedad o la

temperatura máxima y mínima. En este caso, se recibe más información en la consulta al servidor, como las coordenadas geográficas (latitud y longitud) o el nombre de la ciudad y su país. Existen muchos más parámetros, pero estos cuatro se utilizarán como etiquetas.

# Capítulo 4

## Validación

Para confirmar que este trabajo cumple con los objetivos y requisitos establecidos al principio, se debe ratificar que todos los componentes funcionan correctamente. También se debe validar que todas las entidades se despliegan adecuadamente y que funcionan de forma apropiada. Esto se puede comprobar con un test extremo-a-extremo, validando que los fenómenos e información generada por los sensores, se transmiten por todas las entidades de la red, se almacenan, y que principalmente, puedan ser visualizados.

### 4.1. Revisión de despliegue

Una de las partes principales para determinar si el sistema desarrollado en este TFG puede ser verificado por terceras personas (o desplegada por estudiantes) de nuevo, es comprobar otra vez el archivo de configuración YAML juntando todos los elementos descritos en el Capítulo 3. Aunque los elementos se han testeado y funcionan, es necesario determinar que un nuevo despliegue cumple con todos los requisitos para que un nuevo desarrollador pueda desplegar los mismos servicios del lado del servidor. Se despliega un sistema operativo (L)Ubuntu 21.04 (diferente al de desarrollo del TFG), con la aplicación Docker instalada.

En el archivo de despliegue final se han automatizado de principio a fin todos los pasos principales para que cada una de las entidades funcione. El objetivo es asegurar que el entorno de desarrollo necesita la mínima configuración. Se ha comprobado que los siguientes objetivos se cumplen:



- Todos los contenedores se encuentran configurados en la misma red interna.
- Los contenedores son accesibles, entre sí, usando el *hostname*.
- Node-RED se despliega junto con todos los *plugins* adicionales instalados durante el desarrollo de este trabajo.
- InfluxDB se despliega, automáticamente, con un usuario, contraseña, bucket y *token* predefinidos.
- Grafana se despliega, automáticamente, con un usuario y contraseña predefinidos.
- Grafana se autoconfigura con InfluxDB como fuente de datos con un archivo de configuración.

En la Figura 4.1 pueden verse todos los contenedores corriendo después de ejecutar el programa de despliegue:

Name	State	Quick actions	Stack	Image	Created	Published Ports
telegraf	healthy		apps	telegraf:1.19.3	2021-09-08 01:15:53	-
nodered	healthy		apps	nodered/node-red:2.0.6	2021-09-08 01:15:48	<a href="#">1880:1880</a> <a href="#">1880:1880</a>
orion	running		apps	fiware/orion:3.2.0-PRE-83	2021-09-08 01:15:48	<a href="#">1026:1026</a> <a href="#">1026:1026</a>
grafana	running		apps	grafana/grafana:8.0.2	2021-09-08 01:15:48	<a href="#">3000:3000</a> <a href="#">3000:3000</a>
influxdb	running		apps	quay.io/influxdb/influxdb:v2.0.3	2021-09-08 01:15:44	<a href="#">8086:8086</a> <a href="#">8086:8086</a>
mongo	running		apps	mongo:4.4	2021-09-08 01:15:44	<a href="#">27018:27017</a> <a href="#">27018:27017</a>
mongorion	running		apps	mongo:4.4	2021-09-08 01:15:44	<a href="#">27017:27017</a> <a href="#">27017:27017</a>
portainer	running		apps	portainer/portainer	2021-09-08 01:15:44	<a href="#">8000:8000</a> <a href="#">8000:8000</a> <a href="#">9000:9000</a> <a href="#">9000:9000</a>
influxdb_cli	stopped		apps	quay.io/influxdb/influxdb:v2.0.3	2021-09-08 01:15:48	-

Figura 4.1: Listado de contenedores activos y funcionando mostrados por Portainer (InfluxDB.CLI funciona una vez y se detiene solo).

## 4.2. Adaptación de modelos de información

Uno de los objetivos que propone este TFG es la adaptación o conversión a modelo de información basado en el estándar NGSIV2 y usarlo a la hora de almacenar toda la información en la base de datos InfluxDB. De esta forma la base de datos InfluxDB puede actuar como un acumulador de diferentes fuentes de datos, ofreciendo un sistema unificado. Si InfluxDB puede almacenar los datos de todas las fuentes siguiendo

un modelo de información común, entonces serviría un modelo único sobre datos que no fueron recolectados de la misma forma.

El principal reto a la hora de implementar este diseño es que InfluxDB necesitaría no solo brindar un esquema que englobase el diseño de NGSiv2, sino que debería ser capaz de ofrecer unos servicios parecidos a Orion. Es decir, precisaría poder almacenar los datos con un formato similar al explicado en el Capítulo 2 y ofrecerlos a diferentes clientes. Sin embargo, esto no es posible con InfluxDB.

En InfluxDB los elementos se almacenan como *time-structured merge tree* (TSM) y *time series index* (TSI). En resumidas palabras, los archivos TSM sirven para almacenar los datos en serie temporal, mientras que el TSI habilita a InfluxDB a almacenar índices en disco (sin usar necesariamente RAM). Con este método, se usan las páginas de caché del sistema operativo para obtener información que se necesita consultar sin apenas pérdidas de tiempo. La información que no es tan crítica queda almacenada en disco.

Teniendo en cuenta esta estructura, es necesario explicar que conceptos como bucket, medida, etiqueta o campo son específicos de InfluxDB. Son conceptos relativamente diferentes a una base de datos SQL (relacional) o una NoSQL. Esto significa que aunque hay equivalencia en los principales términos conceptuales de InfluxDB con una base de datos SQL, no es posible crear un esquema personalizado. En otras palabras, un diseño como el propuesto en NGSiv2 tiene sentido con una base de datos NoSQL como MongoDB. De hecho, no es casualidad que el Orion Context Broker use esta base de datos para almacenar la información. Por lo tanto, no es posible unificar diferentes formatos y diseños usando InfluxDB si se pretende seguir un diseño como el de NGISv2.

InfluxDB siempre almacena los datos siguiendo una organización de cierta jerarquía, en lo que a los conceptos previamente mencionados se refiere. Siguiendo una

equivalencia con una base de datos SQL, el bucket es análogo a una base de datos con una política específica de retención. La medida se puede identificar como una tabla en una base de datos SQL. Las etiquetas con columnas indexadas de una tabla SQL y los campos son columnas no indexadas de una base de datos SQL. Esta organización conceptual no puede modificarse como se modifica la estructura de un documento en una base de datos NoSQL o las tablas en una base de datos SQL. Por lo tanto no se ha podido efectuar una adaptación del diseño de los datos a NGSIv2 usando InfluxDB. Sin embargo, la Sección 4.5 ofrece una implementación complementaria que es capaz de ofrecer un diseño NGSIv2 en una base de datos de largo plazo.

### 4.3. Verificación extremo-a-extremo de la información

En esta sección se pretende analizar como la información se genera, transporta y almacena en la base de datos InfluxDB, comprobando que se cumple todo el proceso de recepción y adaptación de la información. Con el Caso 1 se hará una explicación más extensa que con los demás, para no que no resulte una sección repetitiva.

En lo que se refiere al primero de los casos de uso descritos en el Capítulo 2, los sensores de SmartSantander hacen mediciones de temperatura, humedad, tráfico, etc. Estos datos se envían mediante redes inalámbricas y usan tecnologías como IEEE 802.15.4 o LoRa para enviar los datos. Una vez el *gateway* recibe el mensaje, lo envía al servidor de SmartSantander localizado en la nube. El servidor desplegado en Node-RED, tal y como se ha descrito, recibirá estas medidas y observaciones a través de las notificaciones que se generan una vez hechas las correspondientes subscripciones al servidor de SmartSantander.

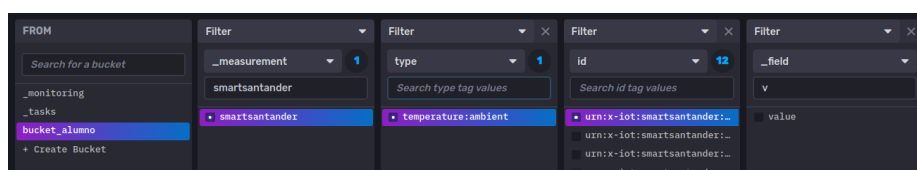


Figura 4.2: Parámetros seleccionables en InfluxDB en el Caso 1.

El servidor, entonces, se encarga de convertirlo al formato correcto para su almacenamiento en InfluxDB, extrayendo las etiquetas y campos correctos para con la estructura del mensaje recibido de SmartSantander. Cuando se constituye el mensaje, se envía a la base datos, mostrando la estructura mostrada en la Figura 4.2.

Para el Caso 2 el proceso es similar, con la diferencia de que los datos llegan de un sensor emulado y el servidor es Orion. Entonces, el servidor incluido en el flujo de Node-RED recibe y adapta los datos a la estructura impuesta por InfluxDB. En la Figura 4.3 se muestra la información y cómo se estructura en los diferentes bucket (bucket\_alumno), medida (orion) y etiquetas (id y \_field).

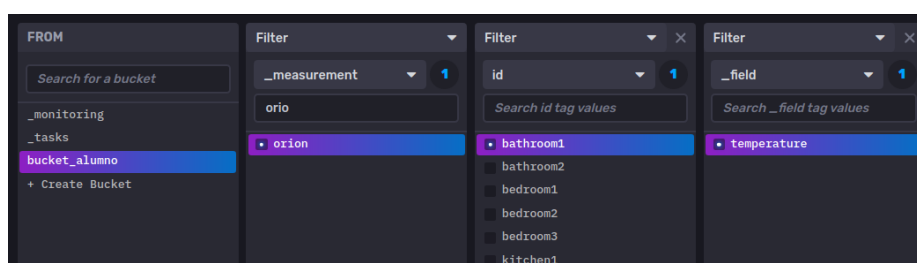


Figura 4.3: Parámetros seleccionables en InfluxDB en el Caso 2.

En el Caso 3, los datos se reciben desde el broker MQTT de la red LoRaWAN de The Things Network. La Figura 4.4 muestra la estructura de la base de datos InfluxDB. En este caso, se usa el mismo bucket que en los dos casos anteriores, pero una medida distinta (específica para cada fuente de datos), y por supuesto diferentes etiquetas (*dev\_addr*, *temperature*, *relativeHumidity*, etc.). Por su parte, en la Figura 4.5 se presenta un ejemplo del contenido incluyendo el resto de campos de la base de datos.

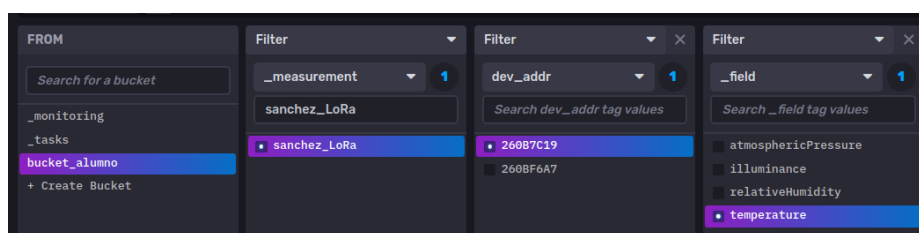


Figura 4.4: Parámetros seleccionables en InfluxDB en el Caso 3.

_start	_stop	_time	_value	_field	_measurement	dev_addr	dev_addr	topic
2021-09-07 03:03:50 GMT...	2021-09-07 04:03:50 GMT...	2021-09-07 03:11:30 GMT...	28.4	temperature	sanchez_LoRa	260B7C19	ed1-70b3d5499a2c19k9	v3/test-app-Isanchez@BT...
2021-09-07 03:03:50 GMT...	2021-09-07 04:03:50 GMT...	2021-09-07 03:31:50 GMT...	28.38	temperature	sanchez_LoRa	260B7C19	ed1-70b3d5499a2c19k9	v3/test-app-Isanchez@BT...
2021-09-07 03:03:50 GMT...	2021-09-07 04:03:50 GMT...	2021-09-07 03:42:00 GMT...	28.39	temperature	sanchez_LoRa	260B7C19	ed1-70b3d5499a2c19k9	v3/test-app-Isanchez@BT...
2021-09-07 03:03:50 GMT...	2021-09-07 04:03:50 GMT...	2021-09-07 03:52:10 GMT...	28.37	temperature	sanchez_LoRa	260B7C19	ed1-70b3d5499a2c19k9	v3/test-app-Isanchez@BT...
2021-09-07 03:03:50 GMT...	2021-09-07 04:03:50 GMT...	2021-09-07 04:02:20 GMT...	28.37	temperature	sanchez_LoRa	260B7C19	ed1-70b3d5499a2c19k9	v3/test-app-Isanchez@BT...

Figura 4.5: Demostración en tabla de los parámetros seleccionados en la Figura 4.4.

En el último caso, el Caso 4, se ha seleccionado OpenWeatherMap. Los datos que se reciben como respuesta al cliente *pull* HTTP, tienen un formato concreto que necesita transformarse al formato impuesto por InfluxDB. Tal y como se puede ver en la Figura 4.6, Figura 4.7 y Figura 4.8, se han seleccionado la humedad, presión, temperatura y velocidad del viento como campos. Después se establecen la ciudad, el país, y las coordenadas como etiquetas.

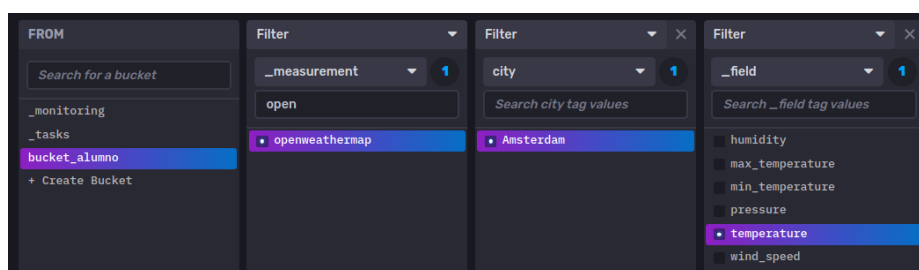


Figura 4.6: Parámetros seleccionables en InfluxDB en el Caso 4.

_start	_stop	_time	_value	_field	_measurement	city	country	latitude	longitude
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:07:20 G...	15.519999999999992	temperature	openweathermap	Amsterdam	NL	52.374	4.8897
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:08:00 G...	15.548888888888892	temperature	openweathermap	Amsterdam	NL	52.374	4.8897
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:08:30 G...	15.519999999999992	temperature	openweathermap	Amsterdam	NL	52.374	4.8897
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:09:00 G...	15.588888888888892	temperature	openweathermap	Amsterdam	NL	52.374	4.8897
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:09:30 G...	15.519999999999992	temperature	openweathermap	Amsterdam	NL	52.374	4.8897
2021-09-07 03:07:20 G...	2021-09-07 04:07:20 G...	2021-09-07 03:10:00 G...	15.519999999999992	temperature	openweathermap	Amsterdam	NL	52.374	4.8897

Figura 4.7: Demostración en tabla de los parámetros seleccionados en la Figura 4.6.



posibilitar la selección dinámica de medidas, etiquetas y campos.

Por lo tanto, se ha programado un tablero que usa variables seleccionables en Grafana para definir consultas a InfluxDB y, por consiguiente, autoconstruir un tablero personalizado. A estas variables se les asignan valores de respuesta de la consulta Flux. Cada variable contiene una lista de 0 o más valores que corresponde a la respuesta de InfluxDB sobre los valores consultados. Por ejemplo, como muestra la Figura 4.9, si se consultan todos los buckets, las variables asociadas a esa consulta almacenarán todos los nombres de los buckets, dejando al usuario la posibilidad de elegir uno de ellos.

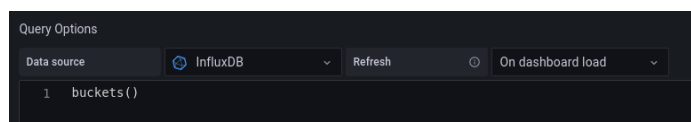


Figura 4.9: Consulta Flux en Grafana para adquirir todos los buckets almacenados en InfluxDB.

En el tablero existen variables que tienen una relación jerárquica entre ellas, como se observa en la Figura 4.10. Es decir, el usuario empieza eligiendo la variable de más alto nivel jerárquico para proceder con las siguientes de menor nivel.

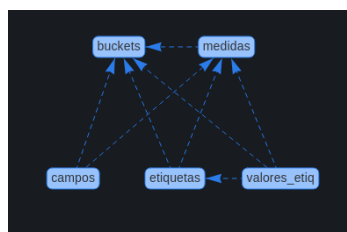


Figura 4.10: Gráfico de dependencias de las variables en el tablero de Grafana.

El usuario debe elegir, primero, el valor de uno de los buckets listados (que se consulta cuando se abre el tablero). Ese valor servirá como variable de filtrado a la hora de listar las medidas asociadas al bucket. A su vez, variable asociada a las medidas desplegará una lista de todas las que fueron respondidas por InfluxDB. Una

vez seleccionada la medida deseada, se consultan todas las etiquetas. Después, se selecciona una etiqueta en concreto para filtrar los datos y, para terminar, un campo. Por ejemplo, la Figura 4.11 muestra la selección del *bucket\_alumno* como bucket, *smartsantander* como medida, *unit* y *degreeCelsius* como la etiqueta y su valor, y por último *temperature:ambient* como el campo a mostrar. La elección de todas estas variables genera peticiones a InfluxDB con esas variables como parte de la consulta.

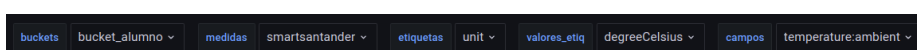


Figura 4.11: Listado de variables seleccionadas en el tablero de Grafana.

En la Figura 4.12, Figura 4.13, Figura 4.14 y Figura 4.15 se pueden ver diferentes ejemplos de las interfaces gráficas que se han generado en Grafana para visualizar la información que se recibe de los cuatro casos de uso que se han integrado en este TFG. Implícitamente, estas visualizaciones confirman que se ha conseguido recibir los datos de las cuatro diferentes fuentes, integrarlos en un único entorno, procesarlos para homogeneizar sus modelos de datos para poder almacenarlos en InfluxDB, y, finalmente, mostrarlos en Grafana.



Figura 4.12: Se muestran los datos de temperatura para SmartSantander (Caso 1).





Figura 4.13: Se muestran los datos de temperatura del sensor emulado desde Orion (Caso 2).



Figura 4.14: Se muestran los datos de luminancia de un sensor LoRa que envía la información usando MQTT (Caso 3).

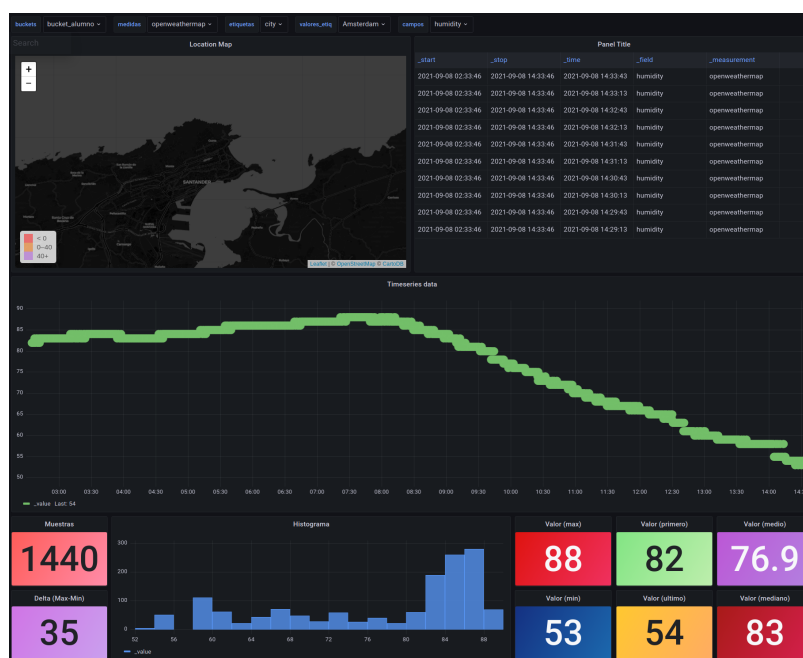


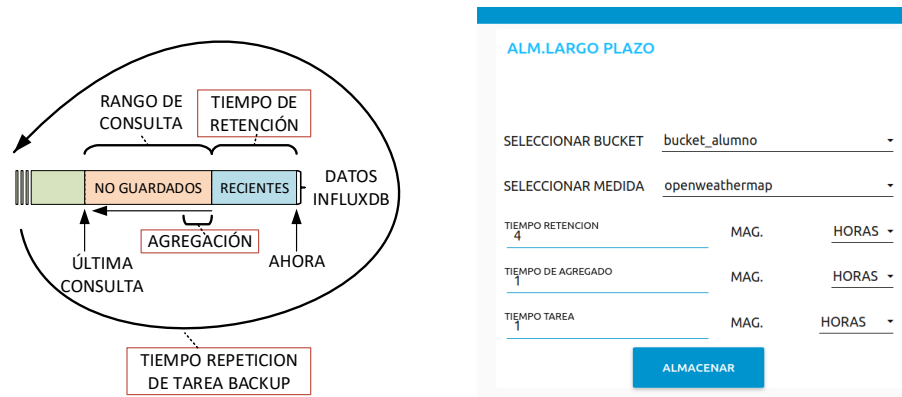
Figura 4.15: Se muestran los datos de humedad en Ámsterdam por OpenWeatherMap (Caso 4).

## 4.5. Almacenamiento a largo plazo

Como se explica en la Sección 4.2, no es posible adaptar todos los formatos a un único modelo, como el de NGSIv2. Debido a la estructura de InfluxDB y como gestiona la información, es necesario adaptar el modelo y formato de los datos a la estructura jerárquica de bucket, medida, etiquetas y campos que utiliza InfluxDB. Sin embargo, es cierto que toda información consultada de InfluxDB tendrá un único modelo, el impuesto por la propia base de datos. No obstante, si no hay un criterio único al transformar los datos cuando se insertan en InfluxDB, sería imposible determinar que la respuesta del propio InfluxDB siga algún tipo de modelo estandarizado. Igualmente, del modelo interno de datos de InfluxDB al que propone NGSIv2 hay una gran diferencia. Por eso se ha intentado trasladar la transformación a NGSIv2 a la base de datos que almacena los datos a largo plazo.

Los objetivos son varios: Determinar un periodo de retención, un periodo de agre-

gación y un periodo de ejecución de la tarea de respaldo o *backup*. Además, se transformarán los datos siguiendo el modelo de NGSiv2, en una forma muy parecida a como Orion almacena los datos en su base de datos Mongo. Los tres primeros objetivos se pueden ver resumidos en la Figura 4.16a.



(a) Diagrama conceptual de los conceptos más importantes en el almacenamiento a largo plazo. (b) UI para introducir los términos de configuración del respaldo de datos a largo plazo.

Figura 4.16: Diagrama conceptual y UI para el almacenamiento de datos a largo plazo.

Con la UI que se muestra en la Figura 4.16b, el desarrollador puede decidir qué *bucket* y medida se deben respaldar, así como el tiempo de retención, agregación y ejecución del respaldo. Para la transformación de los datos al modelo de referencia de NGSiv2, se deben primero insertar como etiquetas en InfluxDB, unos parámetros como el ID o el nombre del registro que se inserta. Estos parámetros son interpretados como ID y nombre de la entidad NGSiv2. Todos los campos son interpretados como atributos, con su nombre, tipo y valor. Para terminar, algunas etiquetas pueden llegar a ser atributos, meta datos o representar características como la localización. Si el flujo de Node-RED que respalda los datos es único para cualquier bucket y medida, se debe crear un criterio a la hora de almacenar datos en InfluxDB. Por ejemplo, que las etiquetas cuyo nombre empiece por un símbolo en particular sean consideradas como atributos y otro criterio para metadatos. El Listado 4.1 muestra los datos agregados de OpenWeatherMap para Ámsterdam en la base de datos MongoDB de almacena-

miento a largo plazo.

```
1 {
2   "_id" : ObjectId("613cfd15b07a6f0011dcc582"),
3   "id" : "2759794",
4   "type" : "Forecast",
5   "_start" : "2021-09-11T11:01:41.571Z",
6   "_stop" : "2021-09-11T15:01:41.587Z",
7   "_time" : "2021-09-11T13:00:00Z",
8   "temperature" : {
9     "type" : "Float",
10    "value" : 20.132916666666664
11  },
12  "location" : {
13    "type" : "geo:json",
14    "value" : {
15      "coordinates" : [
16        "52.374",
17        "4.8897"
18      ],
19      "type" : "Point"
20    },
21    "metadata" : {
22      "city" : {
23        "value" : "Amsterdam",
24        "type" : "Text"
25      },
26      "country" : {
27        "value" : "NL",
28        "type" : "Text"
29      }
30    }
31  }
32 }
```

Listado 4.1: Información de OpenWeatherMap (que proviene de InfluxDB) en formato NGSIv2 que se almacena en MongoDB.

# Capítulo 5

## Discusión

En el Capítulo 1, se mencionaba el problema sobre la heterogeneidad de sensores, tecnologías, protocolos y diseño de datos que existe cuando se intenta programar servicios del lado del servidor. Es preciso un conocimiento muy extenso de muchos de los dominios que abarcan las tecnologías IoT: desde redes inalámbricas hasta programación de servicios en la nube.

A lo largo de la descripción que se ha hecho en los capítulos anteriores sobre los desarrollos llevados a cabo en el marco de este TFG, se ha podido comprobar cómo es posible desplegar un entorno de programación para servicios del lado del servidor. Usando programas automatizados y configuraciones predefinidas se ofrece una herramienta completa para recibir, almacenar y visualizar información. Además, se ha conseguido mostrar qué flujos de programación visual son necesarios para ofrecer una programación coherente para cada fuente de datos. El correcto almacenamiento de los datos a corto y largo plazo, así como la visualización de la información de los sensores usando tableros dinámicos, son otros de los objetivos conseguidos. Aunque el proyecto se pueda considerar como un éxito en cuanto completar y conseguir los objetivos, existen ciertas partes que precisan de una discusión más profunda.

Los servicios del lado del servidor, aquellos que precisan de recibir mensajes de sensores o *proxies*, se programan usando Node-RED. El potencial de una herramienta como Node-RED abarca no solo desarrolladores que precisan establecer una validación de concepto (*proof-of-concept*) sino estudiantes que precisan una comprensión de

cómo funciona el *backend*. Esta herramienta es, sin embargo, un entorno que solo debe considerarse para casos que no precisen un despliegue de producción. La capacidad de Node-RED para recibir una cantidad ingente de datos es limitada, por lo que no debe ser utilizado más allá de los casos ya mencionados. Desarrollar servicios del lado del servidor conllevan un coste económico, y de tiempo, además de la búsqueda de personal altamente cualificado para programar los servidores. Sin embargo, la optimización y rendimiento son mayores. Por eso, esta herramienta (Node-RED) siempre deberá seguir un caso de uso enfocado al prototipado.

En cuanto a la proposición inicial de utilizar InfluxDB como base de datos que siguiese un modelo como el de NGSIV2, no ha sido el más adecuado. InfluxDB tiene unos requerimientos muy particulares a la hora de almacenar la información, y es complicado adaptarlo a un modelo como el de NGSIV2. Tal y como se comenta en la Sección 4.5, no es imposible intentar adaptar el modelo propio de InfluxDB a un modelo similar a NGSIV2. El problema de este método es que necesitaría de otro servidor (integrado junto a InfluxDB) que vuelva a adaptar los datos de la base de datos temporal al modelo de NGSIV2. Tener una base de datos principal como MongoDB habría posibilitado convertir y adaptar todos los formatos de datos de todas las fuentes a un diseño de almacenamiento personalizado (como lo hace Orion). Por esta razón, se decidió trasladar a la base de datos histórica la conversión de modelo y formato de NGSIV2. De esta forma, cualquier consulta a la base de datos tendrá como respuesta un mensaje con un modelo NGSIV2.

Para finalizar, se debe mencionar que Grafana ofrece una gran flexibilidad a la hora de utilizar tableros dinámicos. Es posible consultar la información de varios buckets y medidas, así como filtrar los datos por etiquetas y campos. Eligiendo los valores de los términos antes mencionados, no es necesario comprender cómo funciona la base de datos para visualizar la información recibida. Por otra parte, InfluxDB ofrece una utilidad de tableros muy parecida a la de Grafana. La integración de los tableros de

InfluxDB con la propia base de datos está mejor optimizada, dado que no se necesitan configuraciones adicionales (a diferencia de la necesaria para Grafana). Por esa razón, si el entorno y la arquitectura de desarrollo de un caso parecido al de este TFG incluye (solo) InfluxDB, tiene sentido utilizar en el propio tablero de InfluxDB. Es discutible si existe una ventaja utilidad/coste de Grafana frente a usar la propia de InfluxDB. La ventaja más obvia es que la visualización con Grafana es independiente a la base de datos usada, posibilitando una visualización de datos de varias bases de datos si fuese necesario.

# Capítulo 6

## Conclusiones

Si se observa la evolución de las tecnologías de la IoT, se puede comprobar que han progresado y ofrecen nuevas características a los sensores. Ahora es posible utilizar tecnologías inalámbricas que constituyen un gasto energético menor para los sensores que las utilizan. De la misma forma, existen nuevos avances con las tecnologías usadas en la nube, pero su desarrollo es complejo. Por esta razón, el alcance de los desarrollos llevados a cabo en este TFG se han concentrado en el lado del servidor.

Los objetivos principales de este proyecto se resumen en desplegar e implementar servicios del lado del servidor que puedan recibir, almacenar y visualizar los datos. Las secciones 3 y 4 demuestran que es posible desplegar estos servicios, así como almacenar datos de sensores a corto y largo plazo de una forma estructurada. Se ha comprobado que todos los datos enviados por diferentes fuentes son visualizables en tiempo de ejecución, según se reciben, sin importar el tipo de sensor o protocolos usados en su transmisión. Aun así, existen inconvenientes o problemas como la unificación de un modelo de datos usando NGSIv2 para InfluxDB. En conjunto, se recomienda usar la programación visual para implementar y prototipar las aplicaciones del lado del servidor. El tiempo de desarrollo disminuye significativamente si se compara con un desarrollo no orientado a la programación visual.

Este trabajo no debe considerarse como el límite al que este tipo de tecnologías deben llegar. Existen muchas extensiones donde trabajos futuros deberían enfocarse. Por ejemplo, se podría extender este trabajo implementando un servidor HTTP que



tradujese, directamente, una respuesta de la base de datos InfluxDB. De esta forma, se puede ofrecer un modelo de datos como el NGSIV2. Otra posible extensión podría incluir la adaptación del modelo de datos de diferentes fuentes, mezclándose con una traducción de protocolos. Se podría traducir de MQTT a AMQP, asegurando un modelo y formato de datos estandarizados.

Con la elaboración de este trabajo se espera que con los avances en el despliegue de herramientas como Node-RED e InfluxDB en forma de contenedores ayuden a desplegar estos servicios de una forma fluida. De hecho, el desarrollo de flujos para el procesamiento, almacenamiento y visualización de datos, supone para estudiantes y desarrolladores, un sistema óptimo para desplegar servidores IoT.

# Referencias

- [1] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, “A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures,” *IEEE Access*, vol. 7, pp. 82721–82743, 2019.
- [2] “Sigfox - The Global Communications Service Provider for the Internet of Things (IoT).” <https://www.sigfox.com/en>. (Consultada el 09/09/2021).
- [3] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, “Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT,” in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 197–202, 2018.
- [4] “Coverage — Sigfox.” <https://www.sigfox.com/en/coverage>. (Consultada el 08/30/2021).
- [5] “Homepage - LoRa Alliance.” <https://loro-alliance.org/lorawan-coverage/>. (Consultada el 08/30/2021).
- [6] R. Mozny, P. Masek, M. Stusek, K. Zeman, A. Ometov, and J. Hosek, “On the Performance of Narrow-band Internet of Things (NB-IoT) for Delay-tolerant Services,” in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 637–642, 2019.
- [7] R. K. Singh, P. P. Puluckul, R. Berkvens, and M. Weyn, “Energy Consumption Analysis of LPWAN Technologies and Lifetime Estimation for IoT Application,” *Sensors*, vol. 20, no. 17, 2020.
- [8] B. Mishra and A. Kertesz, “The Use of MQTT in M2M and IoT Systems: A Survey,” *IEEE Access*, vol. 8, pp. 201071–201086, 2020.

- 
- [9] M. A. Tariq, M. Khan, M. T. Raza Khan, and D. Kim, “Enhancements and Challenges in CoAP—A Survey,” *Sensors*, vol. 20, no. 21, 2020.
- [10] “Home — AMQP.” <https://www.amqp.org/>. (Consultada el 09/09/2021).
- [11] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration,” *ACM Comput. Surv.*, vol. 51, Jan. 2019.
- [12] R. T. Fielding, *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [13] “Blockly — Google Developers.” <https://developers.google.com/blockly>. (Consultada el 09/09/2021).
- [14] “Scratch - Imagine, Program, Share.” <https://scratch.mit.edu/>. (Consultada el 09/09/2021).
- [15] “Node-RED.” <https://nodered.org/>. (Consultada el 09/09/2021).
- [16] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.
- [17] Kriegel, Alex and Trukhnov, Boris M., *SQL Bible*. Hungry Minds, Incorporated, 1 ed., 2003.
- [18] “PL/SQL for Developers.” <https://www.oracle.com/database/technologies/appdev/plsql.html>. (Consultada el 09/09/2021).
- [19] “PostgreSQL: Documentation: 13: Chapter 42. PL/pgSQL — SQL Procedural Language.” <https://www.postgresql.org/docs/13/plpgsql.html>. (Consultada el 09/09/2021).
- [20] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, “SQL, NewSQL, and NOSQL Databases: A Comparative Survey,” in *2020 11th International Conference on Information and Communication Systems (ICICS)*, pp. 013–021, 2020.

- [21] J. Han, H. E, G. Le, and J. Du, “Survey on NoSQL database,” in *2011 6th International Conference on Pervasive Computing and Applications*, pp. 363–366, 2011.
- [22] “fiware-ngsiv2-2.0-2018\_09\_15.” <http://telefonicaid.github.io/fiware-orion/api/v2/stable/>, September 2018. (Se accedio el 08/30/2021).
- [23] “Kubernetes.” <https://kubernetes.io/>. (Consultada el 09/09/2021).
- [24] “Don’t Panic: Kubernetes and Docker — Kubernetes.” <https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/>. (Consultada el 09/01/2021).
- [25] “HiveMQ - Enterprise ready MQTT to move your IoT data.” <https://www.hivemq.com/>. (Consultada el 09/09/2021).
- [26] “Santander Facility.” <https://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary>. (Consultada el 09/06/2021).
- [27] “Real-time traffic data - Dataset.” <https://www.opendata.dk/city-of-aarhus/realtime-traffic-data>. (Consultada el 09/06/2021).
- [28] “Accuracy and quality of weather data - OpenWeatherMap.” <https://openweathermap.org/accuracy-and-quality>. (Consultada el 09/06/2021).