UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UNA APLICACIÓN IOT CON ALMACENAMIENTO Y PROCESADO DE DATOS EN LA NUBE MEDIANTE EL PROTOCOLO MQTT

Salvador Vives Parra

Tutor: Antonio León Fernández

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 27 de mayo de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

Este trabajo consiste en la elaboración de una aplicación de Internet de las Cosas con almacenamiento y procesado en la nube mediante el protocolo MQTT. Dicha aplicación se encargará de la obtención de datos, siguiendo con el procesado de estos y finalmente el estudio de ellos.

Para el desarrollo se pretende utilizar los dispositivos Raspberry Pi y NodeMCU que permitirán la creación de una red de comunicación con el protocolo MQTT a nivel local. Dicha red estará conectada directamente a la nube de Amazon Web Services, que actualmente lidera en el mercado del Cloud Computing.

Con este trabajo se busca alcanzar los conocimientos suficientes para poder elaborar una aplicación IoT profundizando en los conceptos teóricos tanto del Internet de las Cosas como del protocolo MQTT, al mismo tiempo que se busca el aprendizaje de las distintas tecnologías utilizadas con el fin de poder resolver y afrontar los retos que supone la realización del proyecto.

Finalmente, se espera poder materializar todos los conceptos y conocimientos aprendidos durante la realización de este trabajo, con el fin de obtener una aplicación que ofrezca una solución a la problemática de una situación real.

Resum

Aquest treball consisteix en l'elaboració d'una aplicació d'Internet de les Coses amb emmagatzematge i processat en el núvol mitjançant MQTT. Aquesta aplicació serà l'encarregada de l'obtenció de les dades, seguint amb el processament i l'estudi d'aquestes.

Per al desenvolupament d'aquesta, es pretén utilitzar els dispositius Raspberry Pi i NodeMCU que ens permetran la creació d'una xarxa de comunicació mitjançant el protocol MQTT en l'àmbit local. Aquesta xarxa estarà connectada directament amb el núvol, el qual s'accedirà per mitjà de la plataforma Amazon Web Services, que actualment lidera en el mercat del Cloud Computing.

Amb aquest treball es busca obtindre els coneixements suficients per a poder elaborar una aplicació IoT profunditzant en els conceptes teòrics tant de la internet de les Coses com del protocol MQTT, al mateix temps que es busca l'aprenentatge de les diferents tecnologies utilitzades amb el fi de poder resoldre i afrontar els reptes que suposa la realització del projecte.

Finalment, s'espera poder materialitzar tots els conceptes i coneixements apresos durant la realització d'aquest treball, amb el fi d'obtindre una aplicació que oferisca una solució a la problemàtica d'una situació real.

Abstract

This project is based on developing an IoT application through cloud computing and cloud storaging using MQTT protocol. This application will be in charge of obtaining and processing the data, which after that will make a research of all the information gained.

There will be some required devices to use in order to develop this project, which are: Raspberry Pi and NodeMCU. These devices will allow the creation of a local area communication's network with MQTT protocol. This network will be connected directly to the Amazon Web Services cloud system, which currently leads the cloud computing market.

The main goal of this project is to gain the necessary knowledge to develop an IoT application carrying out a deep understanding of theorical concepts about the Internet of Things and the MQTT protocol. At the same time this project looks for a complete learning of the several technologies used on it in order to be able to solve the challenges faced on this project.

At last, it is expected to materialize all the concepts and knowledge learned while developing this project, in order to obtain as a result an application that offers a solution to a real problem.

Índice

Capítulo 1. Introducción y objetivos

1.1 Motivación

La motivación del siguiente trabajo viene impulsada por la gran evolución tecnológica que vivimos actualmente, así como la incorporación y aceptación de las tecnologías en nuestras vidas cotidianas.

Unas de las principales razones por las que he decidido hacer una aplicación IoT de procesado y almacenamiento de datos en la nube con comunicación mediante el protocolo MQTT, es el gran auge y la inmensa variedad de posibilidades de implementación que ofrece la tecnología 'Internet of Things', traducido al español 'Internet de las Cosas'.

Internet de las Cosas (IoT) busca conectar un gran número de dispositivos, como pueden ser sensores, los cuales se encargan de recolectar y enviar los datos captados para su procesado y posterior análisis. Dichos datos son de tamaño reducido y se suelen generar con una alta frecuencia, lo que permite tener un conocimiento cercano y actualizado de la situación. Según la necesidad, los datos pueden ser procesados y analizados a tiempo real, lo que permite actuar de manera inmediata, o bien pueden ser almacenados pasa su posterior análisis, de manera que a partir de los datos se puedan estudiar situaciones y comportamientos que permitan detectar patrones y poder predecir en cierta manera comportamientos futuros. Internet de las cosas está presente en muchos sectores, como el de la Industria 4.0 o el de las Ciudades Inteligentes o Smart Cities, debido a que es capaz de cubrir un amplio abanico de posibilidades y ofrecer alternativas y mejoras que, hasta la fecha, o eran imposibles de implementar con otras tecnologías, o bien, eran mucho más costosas a la hora de implementar y de mantener. Otra ventaja de IoT, es la relación que existe con los servicios en la nube, ya que estos nos permiten concentrar toda la información en Internet y poder acceder a ella desde cualquier lugar y en cualquier momento con tan solo disponer de una conexión a internet, de manera que podamos centralizar las fuentes de datos sin importar el lugar de procedencia de los datos.

Todas estas características hacen que me despierte una gran curiosidad por dicha tecnología, la cual, según recientes estudios, y junto a la llegada de las redes 5G en un futuro próximo, apunta a que la presencia de esta aumentará notablemente en los próximos años. Al mismo tiempo, se presenta como un reto y a la vez una oportunidad de explorar nuevos horizontes de mi carrera personal, así como profesional.

1.2 Objetivos

El objetivo principal de la realización del proyecto es la creación de una aplicación IoT completa, desde la instalación y configuración de los sensores, los cuales facilitarán la obtención de datos con los que la aplicación trabajará, hasta el análisis de ellos mediante herramientas que permiten representarlos de manera gráfica y dotarlos de valor.

Dicho objetivo se divide a su vez en otros subobjetivos de manera que permite estructurar y trabajar los distintos aspectos que requiere el proyecto y de este modo alcanzar todos los aspectos a cubrir. Los subobjetivos son los siguientes:

- Estudiar y enriquecer los conocimientos propios de Internet de las Cosas, así como los conceptos principales, con la intención de poder desarrollar una aplicación desde cero y poder escalarla en un futuro próximo.
- Adquirir los conocimientos imprescindibles sobre la placa de desarrollo NodeMCU, así como la incorporación de sensores, configuración de estos y la exploración de las distintas opciones que ofrecen.
- Implementación del protocolo de comunicación Message Queing Telemetry Transport, también conocido como MQTT, el cual se ha vuelto muy popular entre los dispositivos

IoT. Este protocolo está basado en TCP/IP y está compuesto por tres interpretes principales.

- Aprendizaje del lenguaje de programación Python, con el cual se pretende desarrollar la implementación de la comunicación MQTT de la placa de desarrollo NodeMCU con la Raspberry Pi. Esta comunicación permitirá el envío y recepción de los datos obtenidos entre ambos dispositivos.
- Adquirir conocimientos básicos de Cloud Computing, así como del uso de microservicios de la nube. Para ello, se pretende utilizar Amazon Web Servicies, plataforma de servicios en la nube que permite una fácil implementación de servicios y escalabilidad. Algunos de estos servicios permiten la utilización de procesos basados en el procesado de datos, así como la realización de análisis de estos. Otros, ofrecen almacenamiento e incluso disponen de funcionalidades como el envío de avisos y notificaciones en caso de que se encuentren fallos en los datos.
- Estudio de ventajas y desventajas de distintos casos de uso, de manera que gracias a dicho estudio permita encontrar un caso de uso que cumpla con los criterios y requisitos del trabajo.
- Cuantificación de los costes en referencia a la realización e implementación de una aplicación IoT.

1.3 Estructura de la memoria

Esta memoria está compuesta por 7 capítulos, en los cuales se tratan los diversos aspectos y enfoques en los que se centra el trabajo realizado. Empezando, por una introducción donde se resume a grandes rasgos el protocolo MQTT y su relación con Internet de las Cosas, hasta la colección de fuentes de información, las cuales han servido de base para la realización del mismo.

En primer lugar, en el **Capítulo 1** se explica en que consiste el concepto de Internet de las Cosas, cómo se ha visto aumentada su popularidad y su relación con el protocolo MQTT. Al mismo tiempo, se presentan los objetivos que se pretenden alcanzar en realización de este trabajo, junto con las motivaciones que han llevado a su elección, sin olvidar la estructura y metodología de este.

Seguidamente, se encuentra el **Capítulo 2**, donde tras indicar algunas de las características principales del protocolo MQTT y del Internet de las Cosas, se presenta el caso de uso, en el cual se ha basado el desarrollo de la aplicación, acompañado de la estructura que lo compone.

En el **Capítulo 3**, se describen las tecnologías utilizadas, también, se detallan las ventajas de estas frente a otras alternativas, así como su importancia en la realización del trabajo.

Seguidamente, se encuentra el **Capítulo 4**, en el cual se detalla el desarrollo llevado a cabo en la elaboración de la aplicación de Internet de las Cosas. El capítulo alberga tres módulos principales, en ellos se explica la configuración y función de estos, así como el valor que aportan a la aplicación. El conjunto de dichos módulos conforma la aplicación al completo. Adicionalmente, se detalla la batería de pruebas realizas en la aplicación con la finalidad verificar su correcto funcionamiento.

En el **Capítulo 5** se recogen todas las conclusiones obtenidas a partir de la elaboración de la aplicación, así como el cálculo de los costes de los recursos utilizados y las opciones de escalabilidad en un futuro próximo. De este modo, se puede valorar en qué forma se han cumplido las expectativas del trabajo, posibles contratiempos surgidos durante el desarrollo y si las medidas adoptadas para solucionar los contratiempos han enriquecido el trabajo o bien, si por lo contrario han supuesto unos inconvenientes no esperados a la hora de diseñar el proyecto.

Le sigue el **Capítulo 6**, donde se adjuntan todas las fuentes de información que han proporcionado los conocimientos necesarios para el diseño y la posterior realización del trabajo, así como para

la resolución de posibles contratiempos con éxito. En cada entrada, se adjunta el enlace de acceso a la fuente y también, la fecha y hora del último acceso.

Y, por último, se encuentra una serie de **anexos**, en los cuales se recoge las definiciones de los conceptos más recurrentes a lo largo del trabajo, traducciones y significado de las siglas, así como guías de los pasos a seguir en la instalación y configuración de las partes que conforman la aplicación.

1.4 Metodología

Para definir la metodología del trabajo, partimos de una idea principal como es el desarrollo de una aplicación de Internet de las Cosas con almacenado y procesado de datos en la nube mediante el uso del protocolo MQTT. A partir de esta idea, se establece una serie de fases ordenadas cronológicamente, de manera que permita trazar una ruta o camino a seguir para abordar la resolución con éxito del trabajo. Los pasos o pautas seguidas han sido las siguientes:

- **Tarea 1**: **Investigación** exhaustiva acerca de Internet de las Cosas y la relación con el protocolo MQTT, así como la búsqueda de ejemplos o aplicación ya existentes con el fin de encontrar un caso de uso que cumpla con las condiciones y expectativas del trabajo.
- **Tarea 2**: A partir de la **elección** del caso de uso, investigación de elementos candidatos a ser utilizados en el desarrollo de la aplicación, así como la compatibilidad entre ellos y las funciones que van a llevar a cabo dentro de la aplicación.
- **Tarea 3**: Tras decidir los elementos participes, aprender a configurarlos al mismo tiempo que utilizarlos, pero lo más importante, **comprobar** que realmente cumplen con las condiciones y requisitos esperados.
- **Tarea 4**: Realizar un **análisis**, con el fin de diseñar el esquema interno de la aplicación. Para ello, dividir en módulos, cada uno con una función principal, las distintas funcionalidades que va a llevar a cabo cada elemento y definir las relaciones entre los distintos módulos.
- **Tarea 5**: Una vez definida la estructura de la aplicación, empezar con el **desarrollo** de esta, para ello empezar por los módulos que componen la base de la aplicación, en este caso la obtención de datos y centralización en flujos, seguir por los módulos encargados de la gestión y el procesado de dichos datos, y finalizar por los módulos encargados de representar o actuar en función de los datos procesados.
- **Tarea 6**: Finalmente, definir una serie de **pruebas** a seguir con el objetivo de garantizar que los resultados obtenidos son correctos y concuerdan con lo esperado, de este modo poder localizar posibles errores o fallos en la aplicación y poder corregirlos o incluso mejorar el funcionamiento para garantizar una aplicación de calidad.

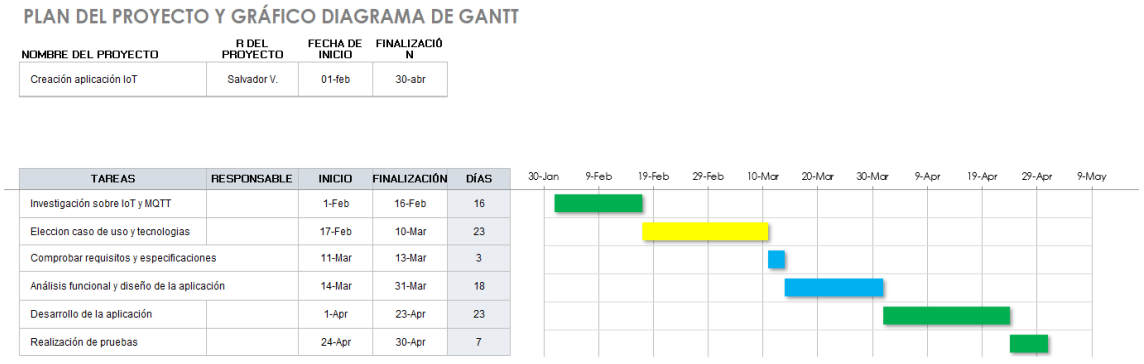

Figura 1. Plan de proyecto y gráfico diagrama de Gantt

Capítulo 2. Presentación

2.1 Introducción

La función de este capítulo es exponer la idea y el funcionamiento de la aplicación desarrollada, al mismo tiempo que se detallan algunas de sus principales características. Por otro lado, el capítulo pretende definir el caso de uso en el que se basa el diseño de la aplicación y el cual ha servido como punto de inicio del trabajo. Finalmente, se describe de manera detallada la estructura sobre la que se apoya la aplicación, así como sus bases y su funcionamiento a nivel interno.

2.2 Características generales

La aplicación se caracteriza principalmente por llevar a cabo una serie de funcionalidades o procesos muy concretos, cada uno con un fin diferente. De esta manera, la aplicación permite partir de un punto inicial, como es la obtención de miles de datos que en un principio no aportan valor, hasta llegar a un punto final en el que se obtiene como resultado distintas representaciones de estos, así como patrones de comportamiento y modelos predictivos que permiten actuar en consecuencia. Otras de las características más importantes de la aplicación son las siguientes:

- Obtención de los datos a partir de la creación de una red a nivel local formada por una gran cantidad de sensores con la finalidad de recopilar distintos tipos de datos de manera frecuente.
- Fácil implementación de las conexiones y comunicaciones entre dispositivos gracias a la utilización del protocolo MQTT.
- Centralización de los datos obtenidos por los sensores en flujos, lo que permite tener un menor número de flujos en la red y por tanto menos conexiones activas.
- Cifrado de la conexión entre la red local y la nube mediante certificado con claves publica y privadas, lo que garantiza una mayor seguridad en el movimiento de los flujos de datos a través de internet.
- Clasificación y tratamiento de los datos mediante reglas y canalizaciones con el fin de agrupar los datos en base a distintos criterios y poder así realizar estudios en base a dichas agrupaciones.
- Disponibilidad de un sistema de detección de fallos encargado detectar y avisar de posibles averías en los sensores que componen la aplicación.

2.3 Caso de uso

El caso de uso se define a partir de una problemática real localizada en una gran cantidad de infraestructuras o edificios que albergan a un gran número de persona como bien puede ser edificios de oficinas y universidades.

La problemática reside en las distintas condiciones cambiantes de temperatura y humedad y su regulación mediante calefacción y aire acondicionado. La mayoría de estos edificios cuentan con calefacción y aire acondicionado de manera centralizada, con control remoto y normalmente programados en base a distintos criterios, como la hora y estación del año, la cual cosa provoca que en unas ocasiones se consigan unas condiciones de temperatura y humedad ideales y en otras no. Esto se debe a que por muchos criterios que se tengan en cuenta a la hora de crear un programa de regulación de la temperatura siempre existen factores espontáneos y sumamente cambiantes que hacen que en ciertos momentos no se ajuste correctamente la temperatura.

La aplicación realizada en este trabajo tiene el objetivo de monitorizar las distintas partes del edificio de forma que cada un breve periodo de tiempo, aproximadamente 10-15 minutos, se

obtenga un informe detallado de las condiciones de temperatura y humedad, de este modo poder regular dichas condiciones más acorde a las necesidades demandadas.

Un ejemplo práctico de este caso de uso sería un edificio de oficinas con revestimiento acristalado en un día cualquiera de entre semana del mes de diciembre. Según el tiempo se espera un cielo nublado y una temperatura de aproximadamente de 11-13 ºC a lo largo de la mañana mientras que por la tarde se esperan una temperatura de 9-12 ºC. Por último, el intervalo de tiempo del almuerzo se considera de 10:30 a 11:00h de la mañana. En base a dichas condiciones se ajusta las distintas temperaturas de la calefacción para crear unas condiciones óptimas de temperatura dentro del edificio para ese día en concreto.

A lo largo del día, se comprueba de manera periódica las condiciones tanto de temperatura como de humedad de los distintos espacios del edifico. A primera hora del día, cuando los trabajadores llegan a la oficina se encuentran con una temperatura agradable que les permite pasar de una sensación de frio de la calle a una sensación cálida. A medida que el día avanza, las nubes desaparecen provocando que el sol irradie con más fuerza sobre el edifico y a mitad mañana los trabajadores empiezan a notar una sensación de calor y de agobio. Por otro lado, los trabajadores de la segunda planta tienen la costumbre de almorzar a las 9:30 en lugar de a las 10:30 como se prevé en un primer momento, esto hace que cuando vuelven de almorzar se encuentran con que la calefacción está a baja potencia con el fin de descender la temperatura del edificio lo que hace que durante el periodo de las 10:30 a las 11:00 horas los trabajadores de la segunda planta sientan una sensación más fría. Por último, se detecta que sobre las 12:30h en las salas de un lado del edificio las temperaturas son bastante más altas con respecto a las salas del otro lado y esto es debido a que las salas del lado con mayor temperatura les da el sol durante toda la mañana mientras que en el otro lado apenas les da. En definitiva, todos estos desajustes de temperatura hacen que muchos de los trabajadores enfermen o simplemente tengan dolores de cabeza y con consecuencia no desarrollen una labor empresarial óptima.

Por tanto, mediante el uso de la aplicación, así como del despliegue de la red de sensores necesaria para el uso de esta, se puede seguir de cerca el comportamiento y las condiciones de temperatura y de humedad de todo el edificio, de esta manera mediante software de terceros o un sistema de control, regular la calefacción a fin de obtener una temperatura controlada en todo momento.

Gracias al caso de uso descrito, el cual se ha utilizado en la realización del trabajo, se ha logrado identificar la base y los pilares fundamentales, a partir de los cuales se ha diseñado y desarrollado de la aplicación. Esto ha permitido la creación de un modelo de aplicación, adaptable a una gran cantidad de casos de uso que requieran procesado de datos distribuido, como por ejemplo la monitorización de la velocidad de las cintas transportadoras en una cadena de producción o la monitorización de la calidad de aire de una zona geográfica, como puede ser una gran ciudad afectada por la contaminación.

2.4 Estructura de la aplicación

La aplicación realizada en este trabajo está dividida fundamentalmente en tres partes principales. Una primera parte se basa en la obtención de los datos mediante sensores y placas de desarrollo NodeMCU. Le sigue otra parte encargada de la centralización de todos los datos en un único flujo, el cual envía los datos a la nube. Y, por último, existe una tercera parte alojada sobre la plataforma Amazon Web Services, en la cual se ha desarrollado una aplicación encargada del procesado y tratamiento de los datos, así como la transformación en resultados finales.

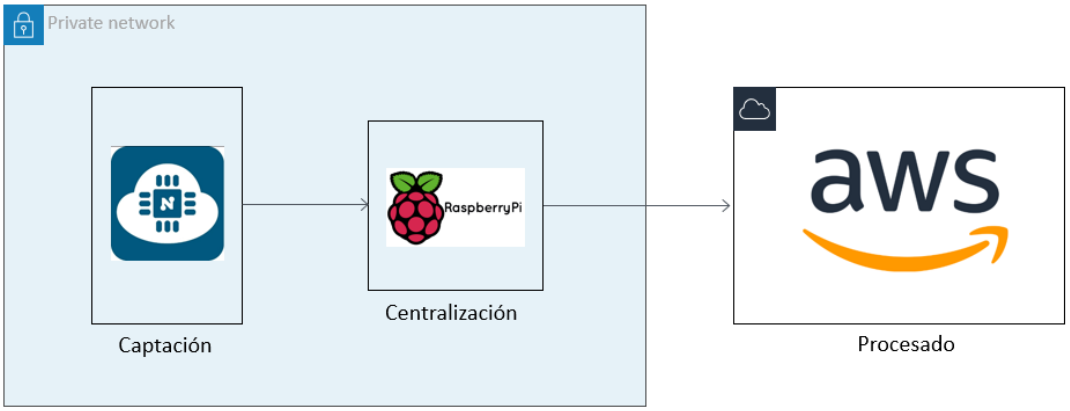

Figura 2. Configuración y tecnologías utilizadas

Respecto a la primera parte que hace referencia a la **captación** de datos, se realiza mediante la conexión de múltiples dispositivos NodeMCU a una red WiFi privada. Estos dispositivos, se encargan de obtener los datos obtenidos por los sensores conectados en función del tipo de dato que se desea obtener y de enviar dichos datos a un tema del bróker MQTT alojado dentro de la red privada.

En el caso de espacios más dispersos, como ciudades o zonas geográficas de gran tamaño, se considera la utilización de otro tipo de redes más afines a este escenario, encargadas de la interconexión de los sensores, como es el caso de redes LPWAN y LoRaWAN, las cuales ofrecen un mayor alcance de la cobertura frente a las redes WiFi.

La segunda parte, encargada de la **centralización** de los datos, hace uso de un dispositivo Raspberry Pi para albergar el bróker MQTT, el cual dispondrá de un tema en el que se publicarán todos los datos obtenidos por los sensores. La Raspberry Pi, al igual que los dispositivos NodeMCU, estará conectada a una red de área local privada con conexión a internet y su función será mantener una conexión MQTT con la aplicación IoT localizada en la nube. Dicha conexión estará cifrada punto a punto mediante el uso de certificado y claves públicas y privadas y se utilizará para enviar todos los datos de todos los sensores de la red privada a la nube para su posterior tratamiento.

Dado un aumento notable del tráfico y en consecuencia la carga de mensajes soportada por el servidor MQTT se podrían implementar distintas soluciones, como un posible traslado del broker alojado en la Raspberry Pi a un dispositivo más potente o la instalación de brokers adicionales en distintos dispositivos Raspberry Pi conectados a la red, con el fin de separar el tráfico y disminuir la carga.

Finalmente, existe una tercera parte, quizá algo más compleja que las anteriores, encargada del **procesado** y transformación de los datos en resultados finales. Esta parte, se desarrolla íntegramente en la plataforma AWS e incorpora diferentes servicios relacionados entre ellos y cada uno con una función en concreto, de este modo se facilita el tratamiento y preparación de los datos para su posterior estudio. Entre los servicios utilizados, se encuentran servicios de almacenamiento como bases de datos y repositorios, servicios de gestión y tratamiento de datos, servicios de alertas y notificaciones y servicios de análisis de datos. En la figura 3 se muestran los servicios utilizados, así como la relación entre ellos, que se han utilizado en la elaboración del trabajo.

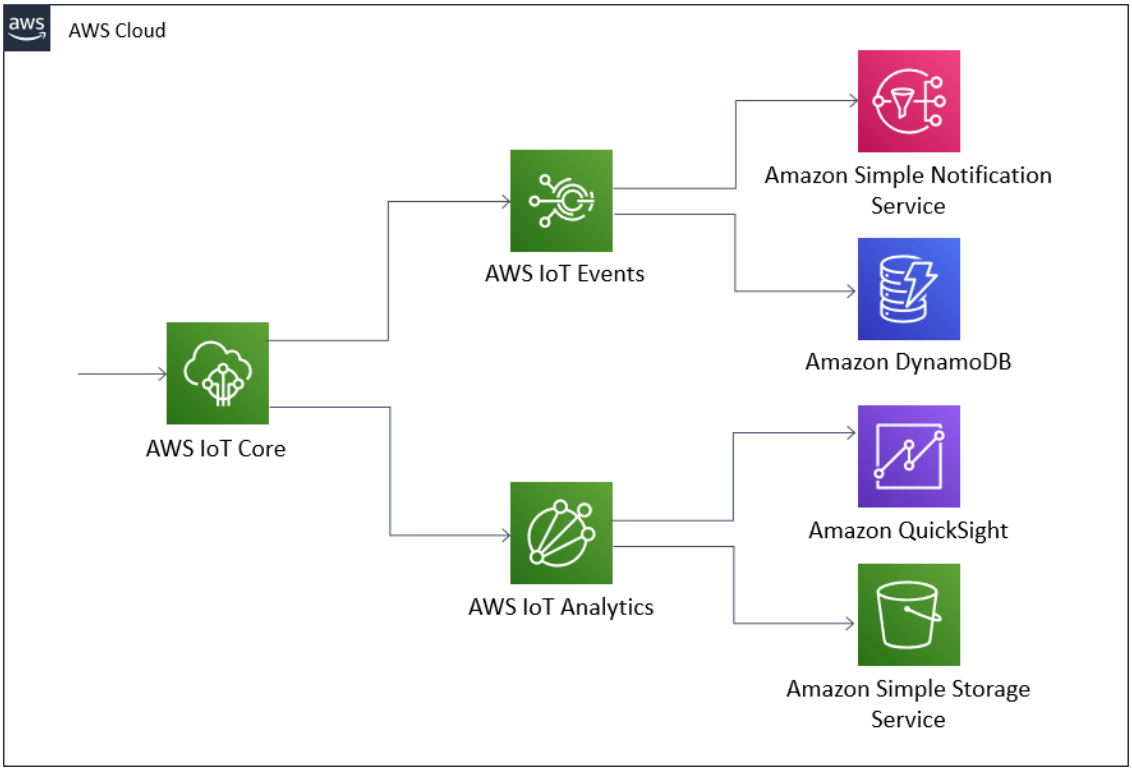

Figura 3. Modelo funcional implantado en AWS

Capítulo 3. Tecnologías

3.1 Raspberry Pi

Raspberry Pi es un ordenador de baja potencia de tamaño reducido y con unas dimensiones similares a las de un disco duro sólido. Está formado por una placa base, la cual dispone principalmente de un microprocesador, chip gráfico y memoria RAM. Otra de sus características principales es que no dispone de disco duro o unidad de almacenamiento, lo que favorece a su tamaño tan reducido. Por lo contrario, dispone de una ranura para tarjetas microSD que a su vez tiene la función de unidad de almacenamiento, la cual albergará el sistema operativo. Adicionalmente, dispone de una salida HDMI, una salida de audio jack 3.5mm, varios puertos USB, un conector RJ45 y una tarjeta de red integrada para conexiones inalámbricas. Estas características se pueden apreciar en la figura 4, la cual muestra el modelo de Raspberry Pi que ha sido utilizada en el proyecto.

El sistema operativo está basado en Linux y se instala mediante distribuciones del mismo, las más comunes son Ubuntu, Raspbian, Android o Firefox Os. Todas las distribuciones se caracterizan por estar basadas en un sistema de código abierto lo que permite la modificación de este y opciones de personalización más amplias frente a los sistemas operativos más comunes como Microsoft Windows o Apple Os X.

Adicionalmente, Raspberry Pi es compatible con protocolos de comunicación como MQTT y con una gran cantidad de sensores, y junto a su bajo coste, hace que su popularidad se haya visto incrementada significativamente, en especial en el ámbito del Internet de las Cosas.

Figura 4. Dispositivo Raspberry Pi 3B+

En ente trabajo se ha utilizado el modelo ‘Raspberry Pi 3B+’, el cual se muestra en la figura 4. Dicho modelo se caracteriza por ser uno de los últimos modelos lanzados al mercado por la marca, solo superado por el modelo ‘Raspberry Pi 4’, el cual dispone de un par de funcionalidades extras.

Las características específicas, se detallan en la siguiente tabla, donde destaca su bajo precio junto a la gran cantidad de puertos y funcionalidades disponibles del modelo en concreto.

Raspberry Pi 3B+	
PROCESADOR	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
FRECUENCIA DE RELOJ	1,4 GHz
GPU	
MEMORIA	1GB LPDDR2 SDRAM
CONECTIVIDAD INALÁMBRICA	2.4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 4.2, BLE
CONECTIVIDAD DE RED	Gigabit Ethernet over USB 2.0 (300 Mbps de máximo teórico)
PUERTOS	GPIO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación) Power over Ethernet (PoE)
FECHA DE LANZAMIENTO	14/3/2018
PRECIO	44,98 eur

Tabla 1. Características técnicas Raspberry Pi 3B+

3.2 NodeMCU

NodeMCU es una placa de desarrollo, de un tamaño equivalente al de un dispositivo pendrive, basada en OpenSource, o lo que es lo mismo, código abierto. Esta placa integra principalmente un módulo WiFi basado en el modelo SystemOnChip que permite la conexión a redes inalámbricas. Este módulo, es el chip integrado ESP8266, el cual tiene como función principal la conexión WiFi y se caracteriza por su bajo consumo. ESP8266 es compatible con la arquitectura TCP/IP y permite la conectividad de los microcontroladores con una red, de manera que estos puedan trasmitir los datos de manera inalámbrica sin tener la necesidad de incorporar un puerto RJ45 que permita la conexión a la red por medio de cableado.

Existen distintas versiones, las cuales implementan diferentes módulos WiFi, pero todas comparten las mismas características principales, entre ellas, la existencia de terminales de tipo pines que permiten la conexión de módulos complementarios y la existencia de un puerto micro USB, que permite la programación de la placa de manera sencilla, al mismo tiempo que realiza la función de fuente de alimentación.

Su popularidad se ha visto incrementada notablemente gracias al gran surtido de software, lenguajes de programación, frameworks y librerías que permiten la fácil programación de esta, así como, a su bajo coste con relación al potencial que ofrece.

En la Figura 5 se muestra el modelo utilizado durante el desarrollo del trabajo, en este se puede apreciar el chip ESP8266, así como, el puerto micro USB utilizado para la programación del mismo.

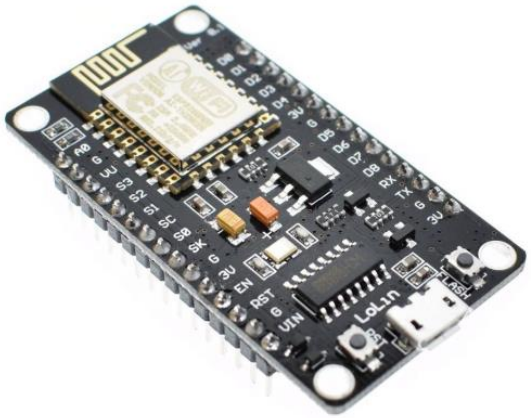

Figura 5. Dispositivo NodeMCU V3 basado en ESP8266

3.3 Amazon Web Services (AWS)

Amazon Web Services, también conocida como AWS, es una plataforma de cloud computing o lo que es lo mismo, informática en la nube. Dicha plataforma ofrece una gran cantidad de servicios muy variados permitiendo al cliente utilizar solo aquellos que realmente demandan sus necesidades. AWS dispone de centros de datos distribuidos por regiones en todo el mundo, la cual cosa garantiza prácticamente una disponibilidad del 100% del servicio, al mismo tiempo la distribución y cercanía de dichos centros permite reducir notablemente las latencias, especialmente en los casos en que se requiere ejecución en tiempo real.

AWS cuenta con servicios que van desde el desarrollo de tecnologías de infraestructura, almacenamiento y bases de datos hasta servicios que facilitan el desarrollo de tecnologías punteras y recientes como Internet de las Cosas, Big Data o Inteligencia Artificial.

La seguridad es otra de las características fuertes de AWS, ya que esta es una de las principales preocupaciones cuando se habla de aplicaciones en internet. AWS cuenta con un amplio abanico de herramientas de seguridad, además es compatible con muchos otros estándares de seguridad y garantiza la integridad de los datos mediante el cifrado de estos en todos sus servicios con el fin de garantizar y cumplir con los requisitos de confidencialidad más exigentes a día de hoy, lo que permite disponer de un entorno seguro.

Finalmente, AWS cuenta con el respaldo de una gran cantidad de clientes que la posicionan como líder en el mercado de la informática en la nube.

3.3.1 AWS Iot Core

AWS IoT Core es uno de los servicios que integra la plataforma Amazon Web Services y que permite conectar una gran cantidad de dispositivos IoT, del orden de millones, con el fin de operar con otros servicios de la plataforma, o simplemente conectar con otros dispositivos.

Este servicio integra cifrado de datos mediante el uso de certificados y claves públicas y privadas con el objetivo de garantizar una conexión segura de los dispositivos con la nube. Además, AWS IoT Core tiene la capacidad de procesar billones de mensajes mediante el uso de reglas, de tal

manera que se pueda redirigir los datos a los distintos servicios compatibles de AWS. Esto permite gestionar distintos tipos de aplicaciones como son aplicaciones basadas en tiempo real o en el análisis y estudio de los datos.

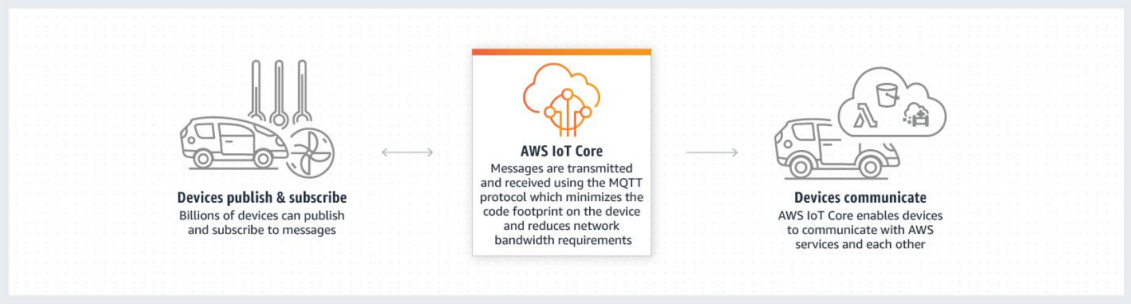

Figura 6. Utilidad servicio AWS IoT Core

3.3.2 AWS Iot Analytics

AWS IoT Analytics es otro de los servicios integrados en la plataforma AWS, el cual permite el análisis y gestión de enormes cantidades de datos IoT, de manera sencilla. Al mismo tiempo permite crear aplicaciones fácilmente sin tener que administrar el hardware ni la infraestructura requerida por estas.

El servicio lleva a cabo diferentes fases ordenadas con el objetivo de garantizar el correcto análisis de los datos. Para ello, cuando los datos llegan al servicio se aplica un filtrado, el cual tiene como finalidad eliminar los datos dañados o con errores de lecturas falsas y así evitar que dichos datos provoquen un análisis desacertado. A continuación, los datos son transformados y enriquecidos con metadatos que facilitan un mejor estudio, de modo que el resultado es un conjunto de datos listos para ser analizados. Finalmente, dicho conjunto se almacena en contenedores temporales a los que los servicios encargados del análisis acceden, con el fin de obtener los datos y llevar a cabo el estudio deseado. También, pueden ser extraídos mediante consultas SQL, con el fin de realizar estudios más complejos.

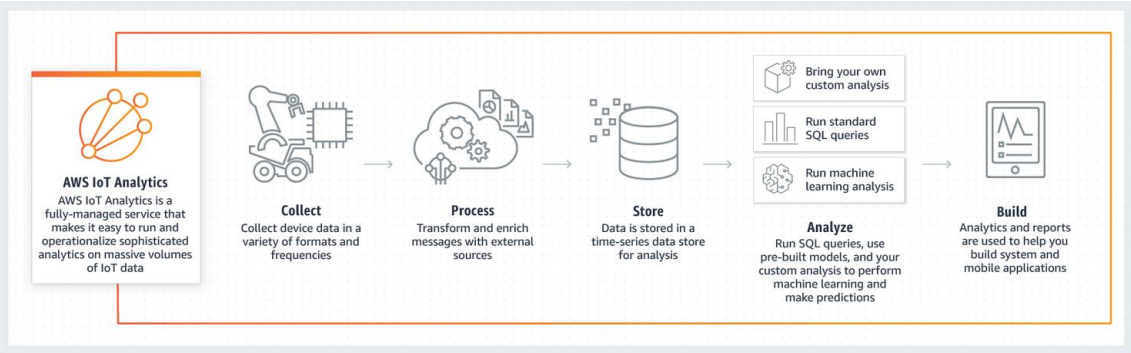

Figura 7. Funcionamiento servicio AWS IoT Analytics

3.3.3 AWS Iot Events

AWS IoT Events es un servicio que tiene como objetivo la monitorización de los datos, la detección de anomalías y la actuación frente a ellas. El servicio es compatible con millones de dispositivos IoT y con los servicios AWS IoT Core y AWS IoT Analytics, lo que permite detectar circunstancias inesperadas de manera temprana. Un ejemplo sería la detección de un

sobrecalentamiento en uno de los motores de una flota de camiones monitorizada por sensores, de este modo permite actuar de manera rápida y así, corregir el problema.

El procedimiento utilizado por AWS IoT Events es el siguiente, se define el origen o fuente de datos, a continuación, se define la lógica que permite actuar frente a determinadas situaciones con estructura de condicionantes "if-then-else" y, por último, se definen alarmas o acciones que se ejecutarán en caso de detectar una de las situaciones predefinidas. Dichas situaciones se componen a partir de patrones de los datos, de forma que cuando se detecta un patrón se activan ciertas acciones o alarmas en función del patrón.

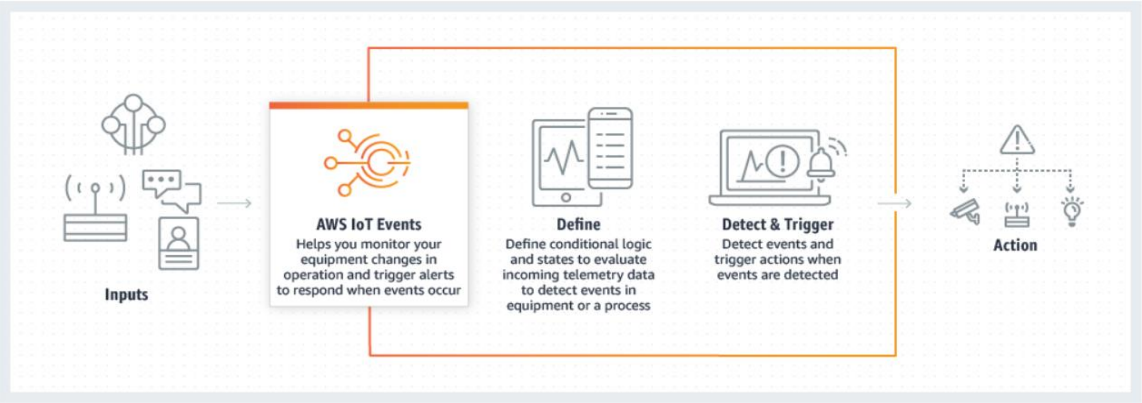

Figura 8. Funcionamiento servicio AWS IoT Events

3.3.4 Amazon QuickSight

Amazon QuickSight es un servicio empresarial integrado en la nube que permite la creación de paneles y gráficos interactivos a partir de un conjunto o fuente de datos. El hecho de ser un servicio alojado en la nube facilita el acceso a los recursos a cualquier usuario con permisos, más allá de donde se encuentre. Al mismo tiempo, permite el acceso a la información mediante él envió de informes por correo electrónico y la integración de los paneles interactivos en aplicaciones y sitios web.

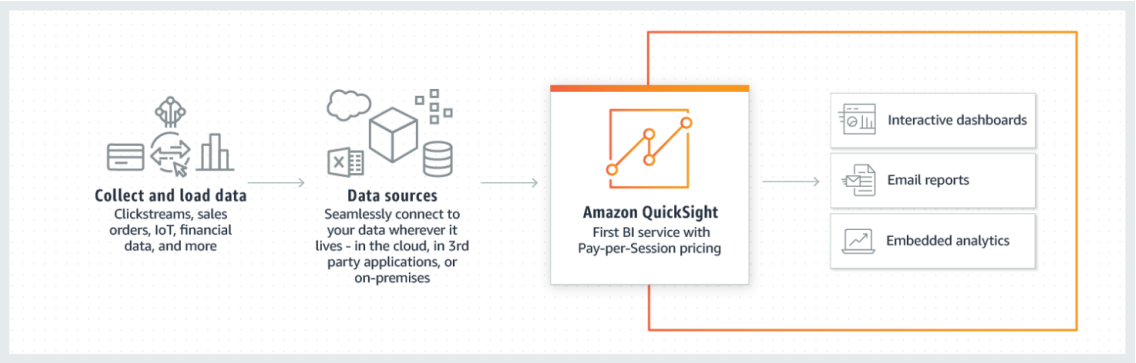

Figura 9. Funcionamiento servicio Amazon QuickSight

3.3.5 Amazon Simple Notification Service

Amazon Simple Notification Service, también conocido como Amazon SNS, es un servicio de mensajería basado en un modelo publicador/subscriptor, el cual hace uso de temas con el objetivo

de distribuir los mensajes a los diferentes puntos de enlace de los suscriptores. Al mismo tiempo, permite el uso de envío y distribución de notificaciones a usuarios finales, mediante notificaciones de inserción móviles, SMS y correo electrónico.

Amazon SNS tiene un funcionamiento similar al protocolo MQTT, en ambos casos se utiliza el modelo publicador/subscriptor de manera que un publicador envía un mensaje a un determinado tema, luego Amazon SNS se encarga de filtrar los mensajes por temas y finalmente los mensajes son enviados a los suscritores que cumplen con las políticas de seguridad y se han subscrito al tema con anterioridad.

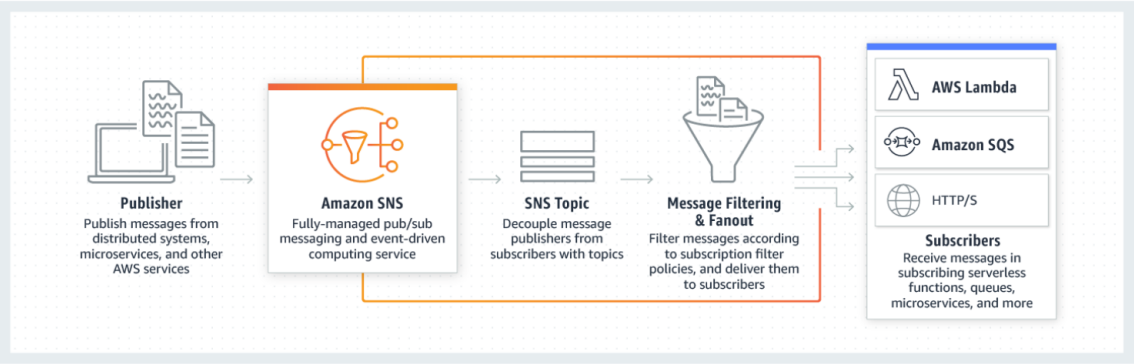

Figura 10. Funcionamiento servicio Amazon Simple Notification Service

3.3.6 Amazon DynamoDB

Amazon DynamoDB es un servicio que ofrece bases de datos con una tipología tipo clave-valor y que soporta millones de solicitudes a lo largo del día, tanto de lectura como de escritura. Así mismo, dispone de mecanismos seguridad que garantizan la integridad de los datos, a la vez, que cuenta con respaldo de copias de seguridad y mecanismo de restauración de los datos por tal de garantir la disponibilidad de estos y así, evitar posibles pérdidas.

Una de las ventajas que ofrece este servicio, es que al ser un servicio auto escalable permite soportar grandes picos sin afectar al rendimiento ni a la disponibilidad de los datos. Esto se debe a que el servicio gestiona de manera automática la capacidad en función del número de solicitudes que recibe y por tanto ajusta los recursos a los requerimientos del momento.

3.3.7 Amazon S3

Amazon S3 es un servicio de almacenamiento en la nube basado en la utilización de objetos como modo de clasificación de los distintos tipos de recursos almacenados. El servicio está diseñado con la finalidad de soportar un gran volumen de datos procedentes desde distintos tipos de fuentes, como son aplicaciones móviles, sitios web y aplicaciones empresariales de gestión de archivos tradicionales, pero también de fuentes de tecnologías más recientes, como son aplicaciones IoT y análisis Big Data, las cuales generan una gran cantidad de datos en una fracción muy reducida de tiempo.

Otra de las ventajas de Amazon S3 es la fácil escalabilidad y la disponibilidad de los datos, lo que permite su uso a todo tipo de clientes, desde clientes pequeños a grandes corporaciones que cuentas con una gestión compleja. Además, el servicio cuenta con un motor de gestión de control de acceso a los datos mediante permisos específicos, esto, permite tener una mayor seguridad respecto a la protección de los datos.

Finalmente, el servicio es compatible con una gran cantidad de servicios de la plataforma que permiten la extracción y manejo de los datos de manera fácil, de esto modo, se pueden obtener los datos necesarios para poder llevar a cabo el análisis con éxito.

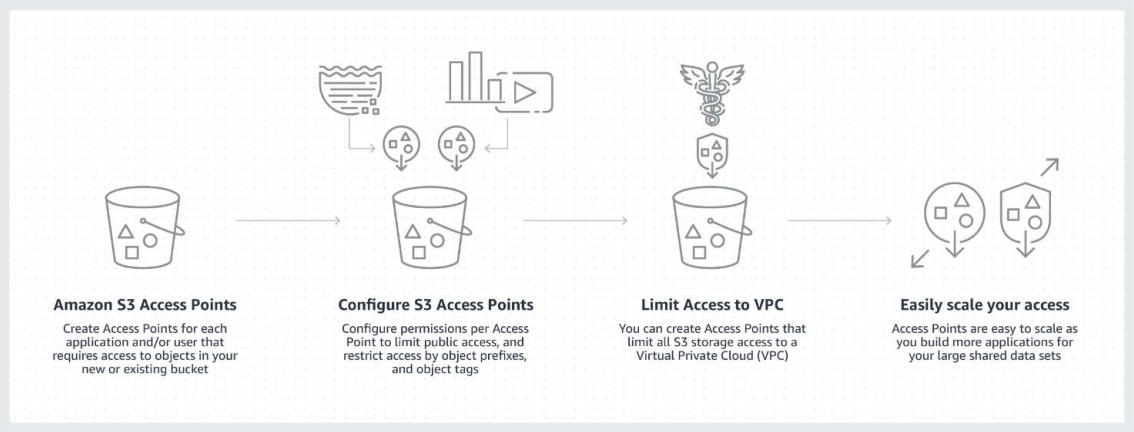

Figura 11. Funcionamiento servicio Amazon S3

3.4 MQTT

El protocolo MQTT es un protocolo de comunicación basado en M2M (Machine to Machine) el cual se implementa sobre la pila TCP/IP.

Dicho protocolo se ha convertido en uno de los más utilizados en el ámbito del Internet de las Cosas, esto es debido a las múltiples ventajas que ofrece, entre ellas, destaca la pequeña cantidad de ancho de banda que consume y la fácil implementación de este, o lo que es lo mismo la reducida cantidad de recursos necesarios para su correcto funcionamiento, esto hace que sea compatible con un gran número de dispositivos del mercado.

MQTT implementa un esquema basado en el modelo PUB/SUB, tal y como se muestra en la figura 12, en el que intervienen tres componentes o participes, los cuales son el Publisher, el Subscriber y el Broker, cada uno de ellos encargado de llevar a cabo una función en concreto.

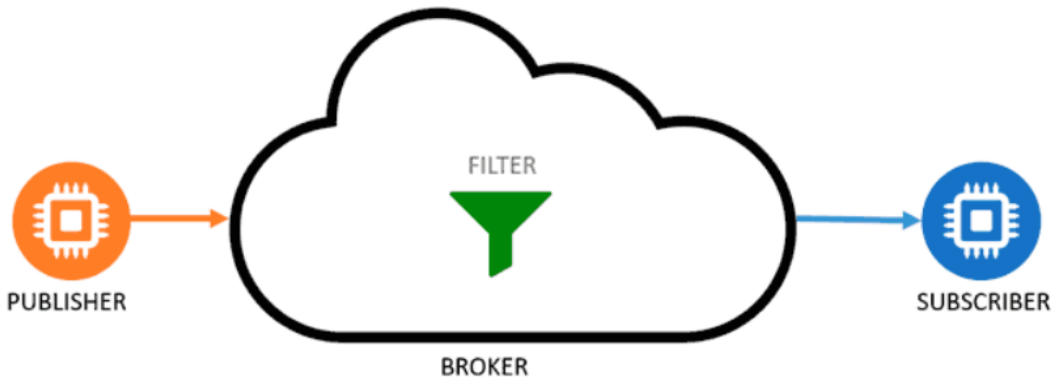

Figura 12. Esquema protocolo MQTT

Las funciones de cada elemento son las siguientes:

- Publisher (Publicador): se encarga de enviar los mensajes a un topic localizado en el bróker.
- Subscriber (Subscriptor): recibe todos los mensajes enviados al topic al cual se ha subscrito con anterioridad.
- Broker: se encarga de administrar los mensajes recibidos por parte del Publisher y hacerlos llegar a los Subscribers indicados, para ello clasifica los mensajes recibidos en topics y los reenvía a los subscribers que se han suscrito a dichos topics en cuestión.

Funcionamiento:

En primer lugar, se debe establecer una conexión TCP/IP entre el cliente y el bróker, dicha conexión permanecerá activa hasta que el cliente decida finalizarla. La conexión se realizará por medio del envío, por parte del cliente, de un mensaje CONNECT, el cual contendrá la información necesaria para realizar la conexión, Una vez el cliente haya recibido un mensaje CONNACK, por parte del bróker confirmando la conexión, esta estará lista para operar. Los puertos utilizados en la conexión son el puerto 1883 para una conexión normal y el puerto 8883 si la conexión se realiza sobre TLS.

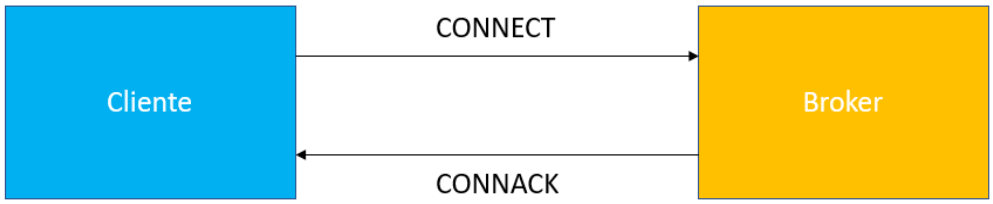

Figura 13. Establecimiento conexión MQTT

El cliente puede adquirir el papel de Publisher, de Subscriber o bien de ambos a la vez, dependiendo de las necesidades demandadas por parte del cliente.

Por otro lado, el cliente podrá subscribirse o darse de baja de un topic, para ello, se utilizarán los mensajes SUBSCRIBE/UNSUBSCRIBE respectivamente, que serán enviados por el cliente. Tanto el mensaje SUBSCRIBE, como el mensaje UNSUBSCRIBE albergarán en su interior el nombre del topic sobre el cual se desea realizar la acción. Una vez, recibido el mensaje de confirmación, SUBACK/UNSUBACK la acción se habrá realizado con éxito.

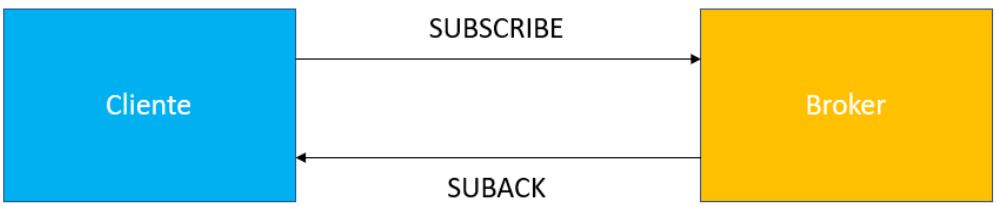

Figura 14. Subscripción a un tema MQTT

Finalmente, en cuanto a la publicación de mensajes por parte del cliente en un topic del bróker, se realizará por medio del mensaje PUBLISH, el cual contendrá el nombre del topic, en el que se desea publicar, y el mensaje a publicar.

Figura 15. Publicación en un tema MQTT

Formato del mensaje:

Los mensajes enviados en el protocolo MQTT están compuestos por tres partes, de las cuales dos son opcionales, estas partes son:

- Cabecera: su tamaño varía entre 2 y 5 bytes y es obligatoria para todos los mensajes. El primer byte contiene la información de control, de los ocho bits que lo conforman se dividen en dos grupos de cuatro bits, el primer grupo alberga información de calidad de servicio QoS, de duplicidad y retención, mientras que el segundo grupo indica el tipo de mensaje. Los bytes sobrantes indican el tamaño del mensaje.
- Cabecera opcional: no es obligatoria, por lo que dependerá del tipo de mensaje enviado. Su función es albergar información complementaria sobre el mensaje.
- Datos: contiene el mensaje real del protocolo, si por ejemplo se tratase de un mensaje SUBSCRIBE los datos serían el topic y el mensaje a publicar.

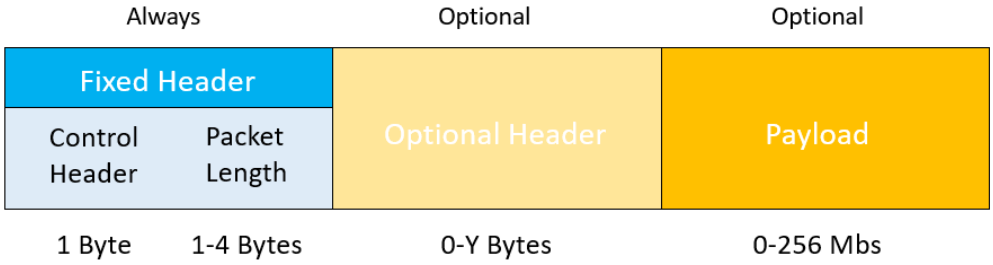

Figura 16. Formato mensaje MQTT

Calidad de servicio:

MQTT incorpora mecanismos para garantizar la calidad de servicio QoS, para ello dispone de tres niveles:

- QoS 0: el mensaje se envía sin que se garantice la recepción de este.

- QoS 1: el mensaje se envía hasta que se recibe un mensaje confirmando la recepción de este, debido al tiempo que puede pasar entre el envío de este y la recepción de la confirmación, puede que el receptor reciba mensajes duplicados.
- QoS 2: se entrega una sola vez el mensaje al receptor y se verifica que le ha llegado.

Seguridad:

Respecto al ámbito de la seguridad, cabe remarcar que MQTT soporta varios mecanismos de seguridad como son, autentificación por medio de usuario y contraseña, así como, la comunicación cifrada por medio de SSL/TLS. La utilización de este último requiere la utilización del puerto 8883, mediante el cual, el cliente valida el certificado perteneciente al servidor y se establece la comunicación cliente-servidor.

3.5 Eclipse Mosquitto

Eclipse Mosquitto es un programa de código abierto que permite la instalación de un servidor MQTT, también conocido como broker MQTT, de manera sencilla y rápida. El software, es compatible con una gran cantidad de sistemas operativos, entre los cuales se encuentra Microsoft Windows y MacOS, así como distintas distribuciones de Linux, como son Raspian, Debian y Ubuntu.

La instalación del programa creará un servidor MQTT, el cual se mantendrá activo y en ejecución mientras disponga de una conexión a la red. Adicionalmente, tras el proceso de instalación, se dispondrá de clientes MQTT, tanto publishers como subscribers que permitirán realizar las pruebas necesarias sin tener la necesidad de instalar software adicional.

Figura 17. Logo Eclipse Mosquitto

Capítulo 4. Desarrollo

4.1 Configuración NodeMCU

4.1.1 Generación datos y envío a la Raspberry Pi

Con la finalidad de obtener datos de temperatura y humedad, provenientes de las lecturas de los sensores se ha desarrollado un sencillo programa, llamado 'NodeMCU.py', mediante el uso del lenguaje de programación Python. Este programa tiene la finalidad de obtener datos de los sensores y a partir de ellos, construir el mensaje que contendrá el campo 'payload' de la trama MQTT, la cual será enviada al tema 'iot/node' alojado en el broker de la Raspberry Pi. En este caso, la placa NodeMCU tomará el papel de publicador MQTT.

Dado que las lecturas provenientes de los sensores apenas varían debido a las condiciones del entorno, se ha modificado el código del programa para que este genere datos dentro de un rango asignado y así disponer de datos más variados.

En resumen, se ha desarrollo un pequeño programa que se ejecutará en la placa de desarrollo NodeMCU, el cual será el encargado de generar los datos y enviarlos a la Raspberry mediante el uso del protocolo MQTT. En los anexos de este documente, se muestran las distintas partes del código, así como una breve explicación de estas.

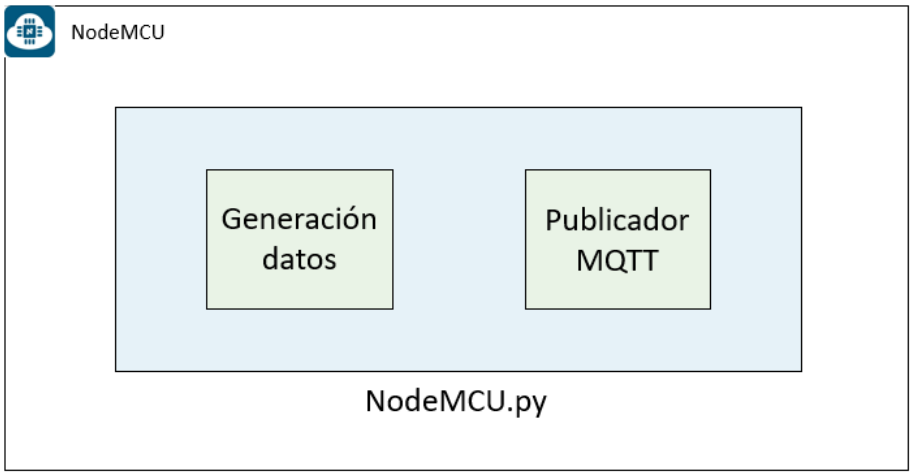

Figura 18. Funciones implementadas en dispositivo NodeMCU

4.2 Configuración Raspberry Pi

4.2.1 Instalación Broker MQTT

Con el fin de recibir mensajes de las lecturas tomadas por los sensores, desde las distintas placas NodeMCU a la Raspberry, se ha instalado un servidor MQTT, también conocido como broker, que tiene la funcionalidad de reenviar los mensajes recibidos en los distintos temas que alberga.

En este caso, el broker ha sido instalado en el dispositivo Raspberry Pi mediante la instalación del software 'Eclipse Mosquitto', adicionalmente se ha asignado una dirección IP estática a la Raspberry Pi para así evitar cambios indeseados en la configuración de los clientes MQTT. En los anexos de este documento se encuentra una recopilación de los pasos a seguir para instalar el software en la Raspberry Pi.

4.2.2 Recepción datos y reenvío a la plataforma Amazon Web Services

Como se explica en el cuarto punto del capítulo 2, la Raspberry tiene como finalidad la centralización de todos los flujos de datos provenientes de los distintos sensores en un único flujo de datos, el cual será enviado a la plataforma Amazon Web Services. Para ello se han desarrollado dos pequeños programas, llamados 'RaspberryPi.py' y 'publicador_aws.py', por medio del lenguaje de programación Python.

El primer programa, tiene como objetivo, recibir todos los mensajes del tema 'iot/node', alojado en el broker, en el cual publican todas las placas NodeMCU, y reenviarlos al segundo, que será el encargado de publicarlos en el tema 'iot/lecturasSensores/SUR' alojado en la nube de AWS. De este modo, al haber un solo dispositivo conectado con la nube de AWS se requieren un número menor de conexiones cifradas y, por tanto, se consigue un ahorro notable de recursos. En el capítulo 7 de este documento, se encuentran los códigos de los programas 'PaspberryPi.py' y 'publicador_aws.py', junto a una explicación de estos y sus distintas funcionalidades.

En este caso la Raspberry Pi, por un lado, hace el papel de broker MQTT, el cual mantiene las conexiones con las placas de desarrollo NodeMCU, mientras que, por otro, hace los papeles de subscriptor, el cual tiene la finalidad de recibir los mensajes publicados por las placas NodeMCU, y de publicador, que tiene la finalidad de reenviar los mensajes al broker alojado en la nube.

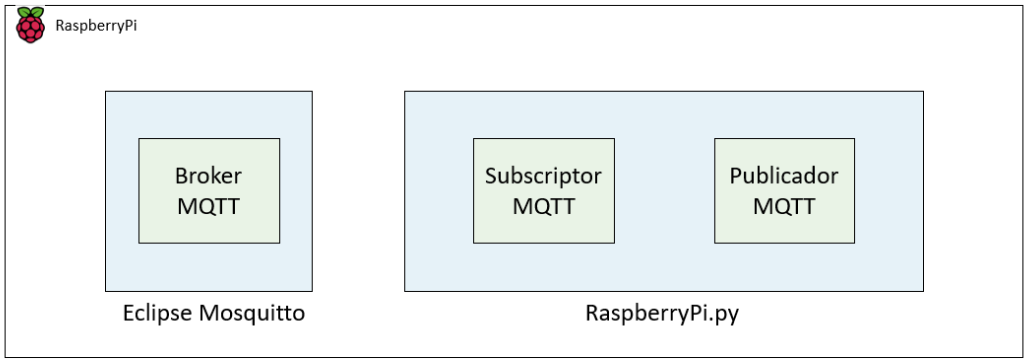

Figura 19. Funciones implementadas en dispositivo Raspberry Pi

4.3 Creación de la aplicación en Amazon Web Service

4.3.1 AWS Iot Core

La comunicación de la plataforma que permite la creación de aplicaciones en la nube (AWS), con el dispositivo Raspberry Pi se realiza mediante el servicio ASW IoT Core, para ello se deben crear los recursos necesarios para poder llevarla a cabo. Los recursos de este servicio permiten garantizar la integridad de los datos, así como servir de punto de partida en el proceso de tratamiento y procesado de estos. A continuación, se muestran los recursos utilizados:

Creación de un objeto:

En primer lugar, se debe crear un recurso llamado 'Objeto', el cual tiene la función de representante del dispositivo que se pretende conectar a la nube, en este caso la Raspberry Pi. Cualquier acción que desee llevar a cabo el dispositivo deberá ser enviada al objeto, el cual se encargará de su realización. Por tanto, si el dispositivo conectado desea publicar un mensaje en un tema de AWS, el dispositivo enviará la petición con los datos al objeto y este, será el encargado de hacerlo.

En este trabajo se ha creado el objeto 'edificioSUR', el cual representa a la Raspberry Pi del edifico de oficinas 'Sur'.

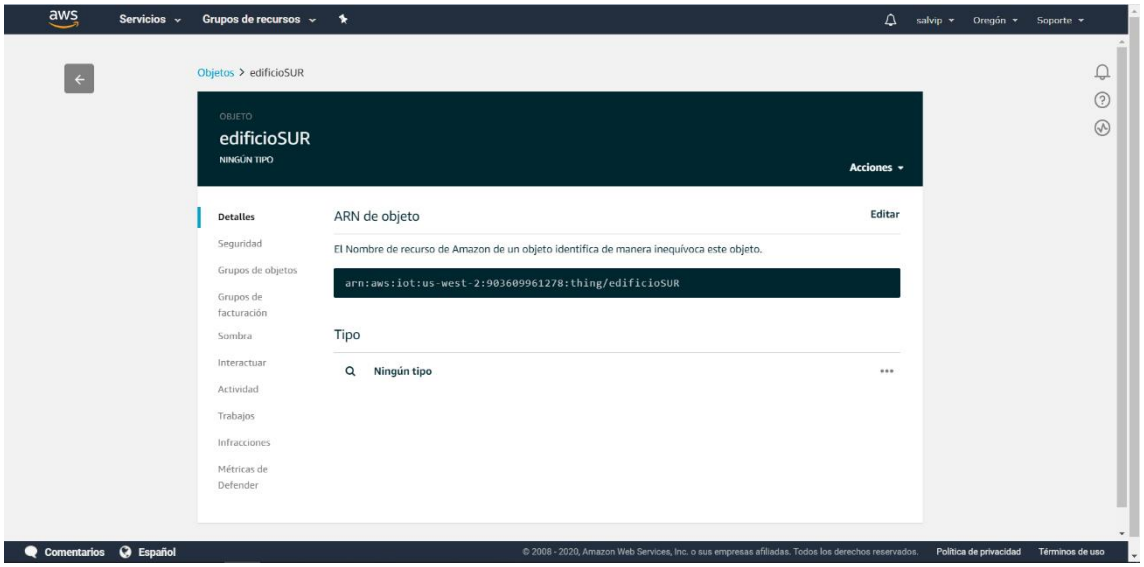

Figura 20. Objeto servicio AWS IoT Core

Creación de un certificado y su asociación al objeto 'edificioSUR':

Con el fin de garantizar la integridad y la privacidad de los datos entre el dispositivo Raspberry Pi y el objeto 'edificioSur' se debe crear un certificado digital con las claves públicas y privadas que nos permita cifrar los datos de dicha conexión y posteriormente asignarlo al objeto correspondiente.

Dicho certificado ha sido creado mediante la opción 'Creación de un certificado' que se encuentra dentro del apartado 'Seguro → Certificados'. Al pulsar sobre dicha acción automáticamente se ha creado un certificado y sus respectivas claves, las cuales junto al certificado han sido descargadas y guardadas. Tras crear el certificado, se ha accedido al certificado y en el apartado 'Objetos' se le ha asociado al objeto 'EdificioSUR' con el fin de hacer uso de este y así cifrar los datos de la conexión. Cabe destacar que un certificado puede ser asignado a más de un objeto, sin embargo, un objeto no puede disponer de más de un certificado.

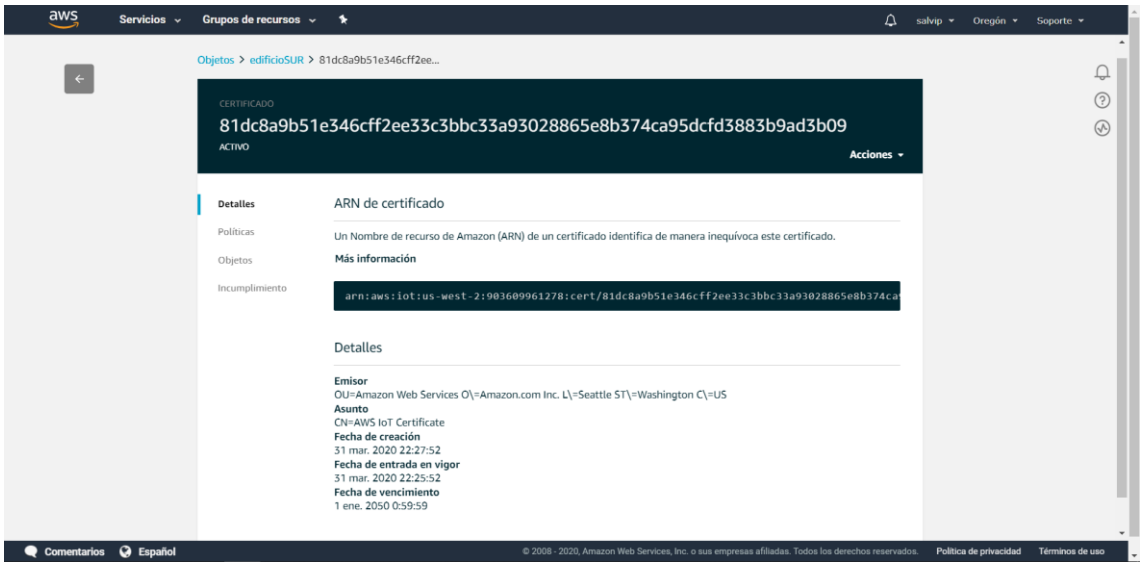

Figura 21. Certificado servicio AWS IoT Core

Creación de una política y su asociación al certificado:

Con el objetivo de aumentar la seguridad en la nube, la plataforma AWS obliga la utilización de políticas de seguridad, las cuales definen una pautas y acciones que regulan el acceso a los recursos y servicios de la plataforma.

Por tanto, se ha creado una política accediendo al apartado 'Seguro → Políticas' y pulsando en la opción 'Crear'. En dicha opción se le ha asignado el nombre 'edificioSUR_politica' y adicionalmente se han definido las pautas, las cuales definen una acción, a quien afecta la acción y si se permite o no la acción. En este caso se ha permitido la conexión y la acción de publicar en el tema 'topic/iot/lecturasSensores/SUR' a todos los recursos que tengan asociada esta política. Finalmente, la política se le ha asociado al certificado que a su vez está asociado al objeto 'edificioSUR', para ello se ha accedido al certificado y en el apartado 'políticas' se le ha asociado la política. Cabe remarcar, que la política es asociada a un certificado y en consecuencia a un objeto, por tanto, todos los objetos que compartan el mismo certificado tendrán la misma política. Al mismo tiempo, el certificado puede tener asociadas distintas políticas.

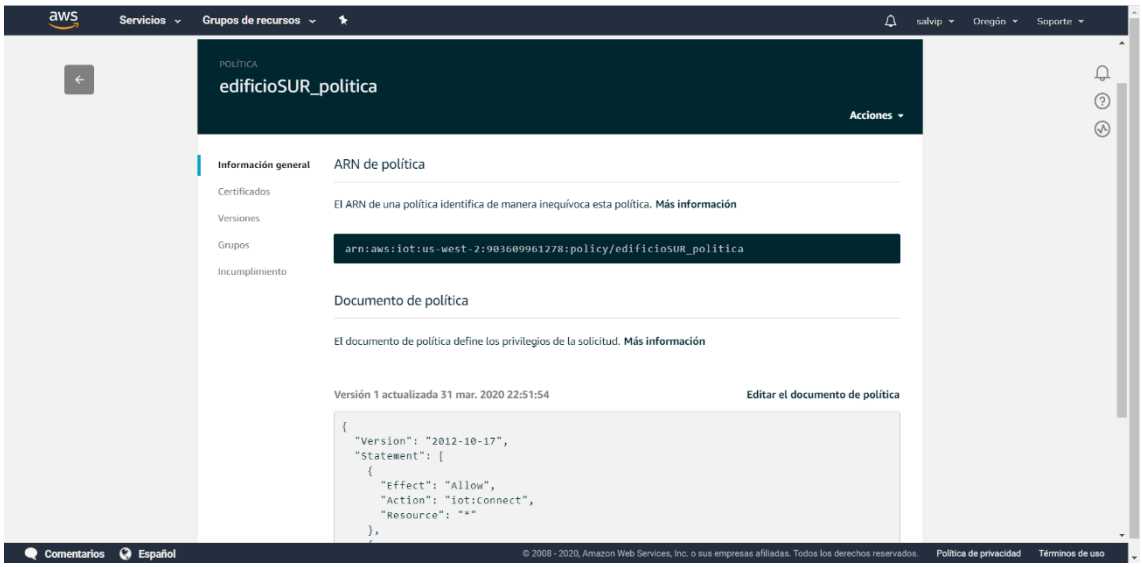

Figura 22. Política de seguridad servicio AWS IoT Core

Creación de reglas para el reenvío de mensajes:

Mediante el uso de reglas se puede filtrar mensajes de un tema MQTT y reenviarlos a distintos servicios de la plataforma AWS, o bien realizar distintas acciones, como publicar en un tema diferente. Para ello, se utilizan consultas SQL con el fin de seleccionar los mensajes o datos del mensaje que cumplen con los criterios especificados en la consulta.

En la realización del trabajo se han creado dos reglas una que permite el reenvío de los mensajes al servicio AWS IoT Analytics y otra que permite el reenvío al servicio AWS IoT Events. Para ello se ha accedido a la opción 'Crear' del apartado 'Actuar → Reglas' y se ha configurado el nombre, la consulta SQL para la obtención de los mensajes y la acción a realizar.

Para la primera, se ha configurado de manera que reenvíe todos los mensajes del tema 'topic/iot/lecturasSensores/SUR' al canal 'datos_sur_channel' creado en el punto 4.3.2 de este trabajo, el cual corresponde con la creación de un canal del servicio AWS IoT Analytics.

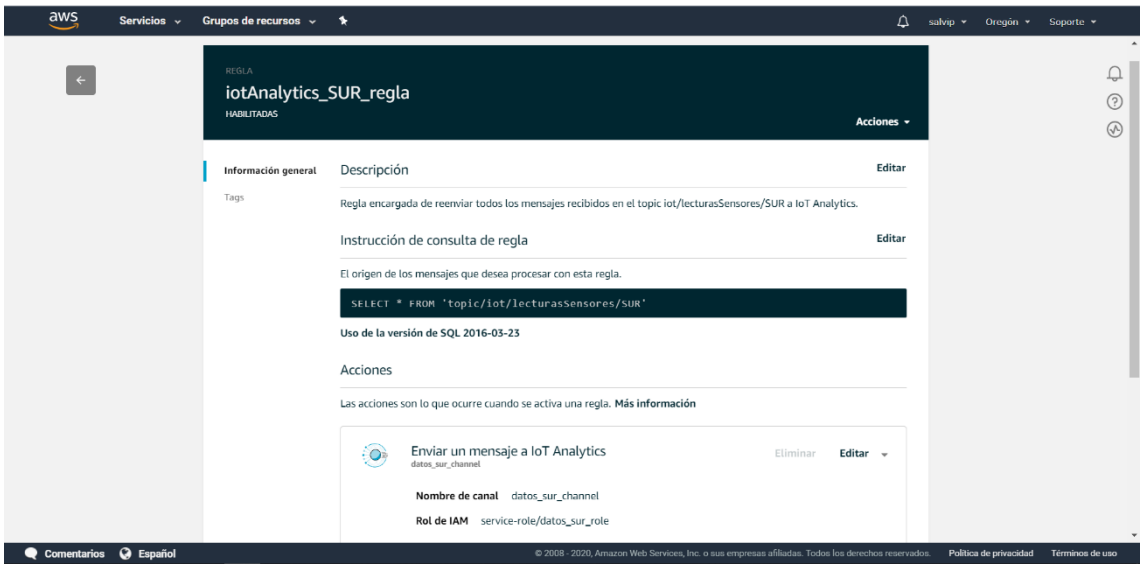

Figura 23. Regla de reenvío de mensajes al servicio AWS IoT Analytics

Mientras que la segunda, se ha configurado de manera que reenvíe todos los mensajes del tema 'topic/iot/lecturasSensores/SUR' a la entrada 'mensaje' del modelo detector creado en el punto 4.3.3 de este trabajo.

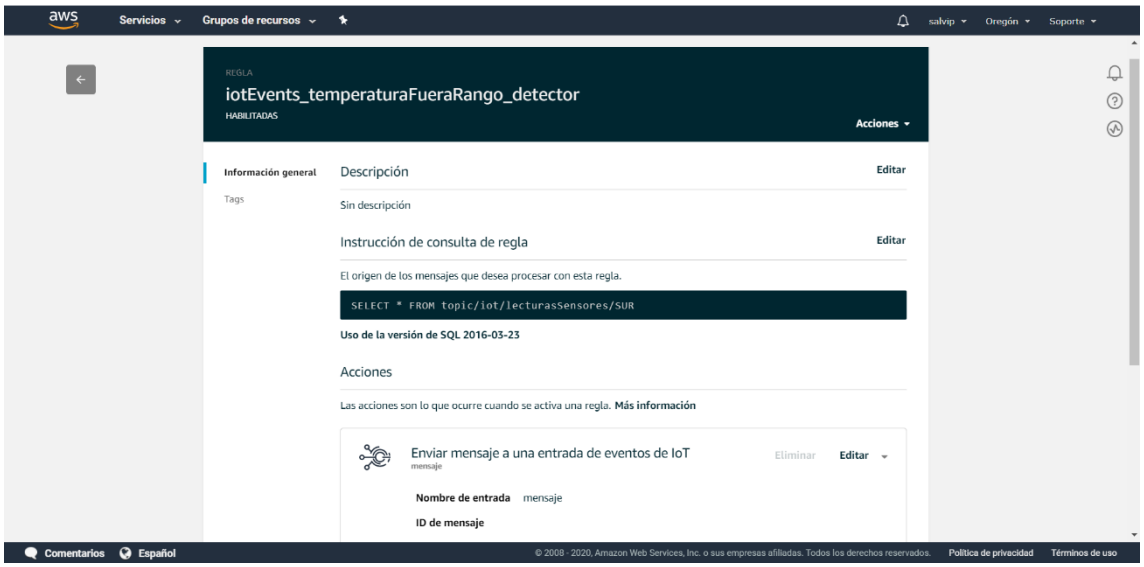

Figura 24. Regla de reenvío de mensajes al servicio AWS IoT Events

4.3.2 AWS Iot Analytics

Los recursos del servicio AWS IoT Analytics permiten el filtrado y el procesado de los datos mediante el uso de canales y canalizaciones, con el fin de crear conjuntos de datos que son guardados en almacenes de datos para su posterior análisis.

En este trabajo su finalidad es dividir el flujo de datos perteneciente al edificio de oficinas 'Sur' en flujos más pequeños, en relación con las distintas plantas o alturas del edificio y de este modo, poder llevar a cabo estudios e informes por plantas.

Creación de un canal:

El objetivo principal del canal es recibir todos los flujos de datos enviados por el servicio AWS IoT Core, los cuales guardan una misma relación, en este caso pertenecen al mismo edificio.

En cuanto a la creación del canal, se debe acceder al apartado 'Canales' y acceder a la opción 'Crear'. Seguidamente se debe asignar un nombre al canal, el tipo de almacén del canal y el periodo de retención de los datos, en este caso se le ha asignado el nombre 'datos_sur_channel', ya que este será el encargado de albergar todos los flujos de datos obtenidos del edificio de oficinas 'Sur'. Adicionalmente, se ha seleccionado el tipo de almacén 'administrado por el servicio' y un periodo 'indefinido' para la retención de los datos.

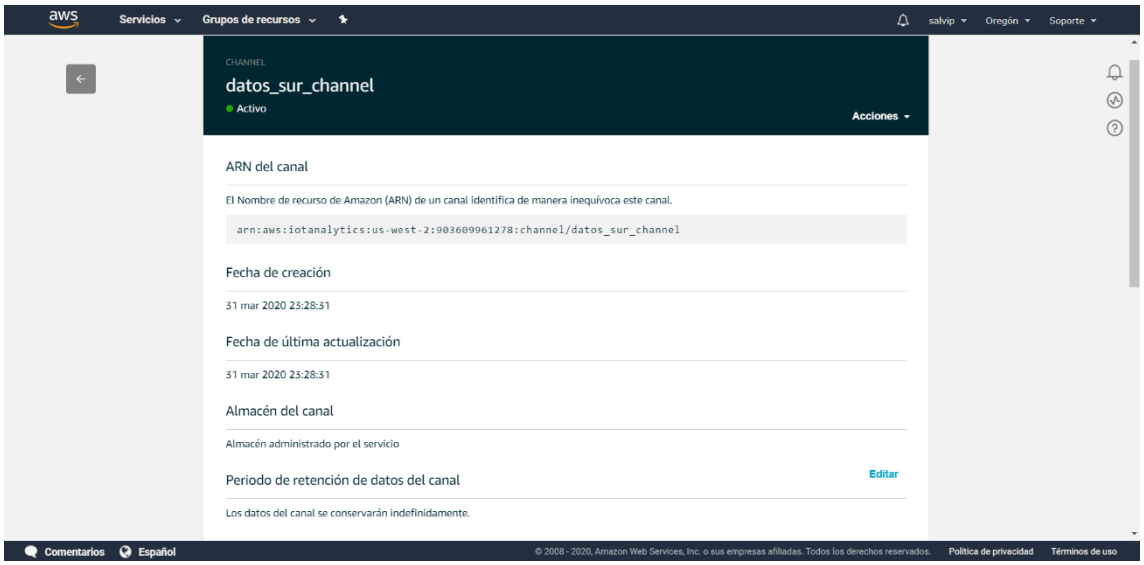

Figura 25. Canal servicio AWS IoT Analytics

Creación de canalizaciones en función de la planta:

Las canalizaciones permiten filtrar los mensajes o datos de un canal en basa a criterios, esto permite obtener flujos de datos más pequeños con las mismas características.

Dado que la finalidad de este trabajo es obtener estudios de las distintas plantas del edificio, se debe crear una canalización por planta, para ello, se deben dividir los mensajes de las distintas plantas, de manera que todas tengan como fuente común el canal 'datos_sur_channel' y como destino un almacén específico para cada planta.

Para la creación de la canalización, por ejemplo, de la primera planta, se ha asignado el nombre 'datos_sur_planta001_canalización', la fuente de datos 'datos_sur_channel' y el almacén 'datos_sur_planta001_almacendatos'. Además, se ha configurado una actividad de filtrado, por lo que solo se obtienen los mensajes de la planta 001 del edificio Sur. Las canalizaciones restantes se han creado de la misma forma, la única diferencia entre ellas es el número de planta, por lo que en los nombres de los recursos y en el criterio de filtrado el número de planta será distinto.

En los anexos de este trabajo, se encuentra un esquema del modelo implementado en el servicio AWS IoT Analytics, el cual muestra los recursos utilizados, así como la relación entre ellos. Este tiene la finalidad de representar de manera visual los recursos utilizados y de este modo favorecer a una mejor comprensión.

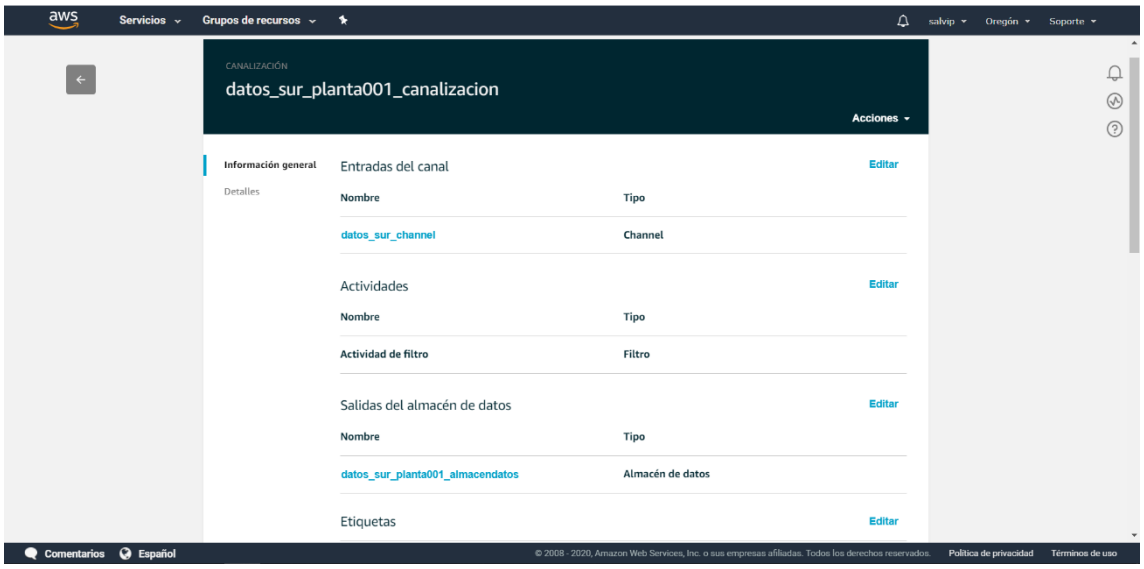

Figura 26. Canalización servicio AWS IoT Analytics

Creación de almacenes de datos en función de la planta:

Los almacenes de datos tienen la finalidad de albergar los datos y mensajes enviados a través de una canalización.

Dado que en este trabajo se desea dividir los datos en función de la planta del edificio, se ha creado un almacén de datos por planta, el cual almacenará todos los mensajes y datos enviados por la canalización de la planta correspondiente.

Para la creación del almacén de datos, por ejemplo, de la primera planta, se ha accedido a la opción 'Crear' del apartado 'Almacenes de datos', donde se le ha asignado el nombre 'datos_sur_planta001_almacendatos' y se le ha configurado un periodo de retención de datos de 10 días, de este modo, todos los mensajes y datos almacenados con más de 10 días son eliminados, lo que contribuye a un ahorro de tamaño del recurso y en consecuencia un ahorro en los costes de este. Los almacenes de datos restantes se han creado de la misma forma, con la única diferencia que cada uno está asociado a una planta distinta.

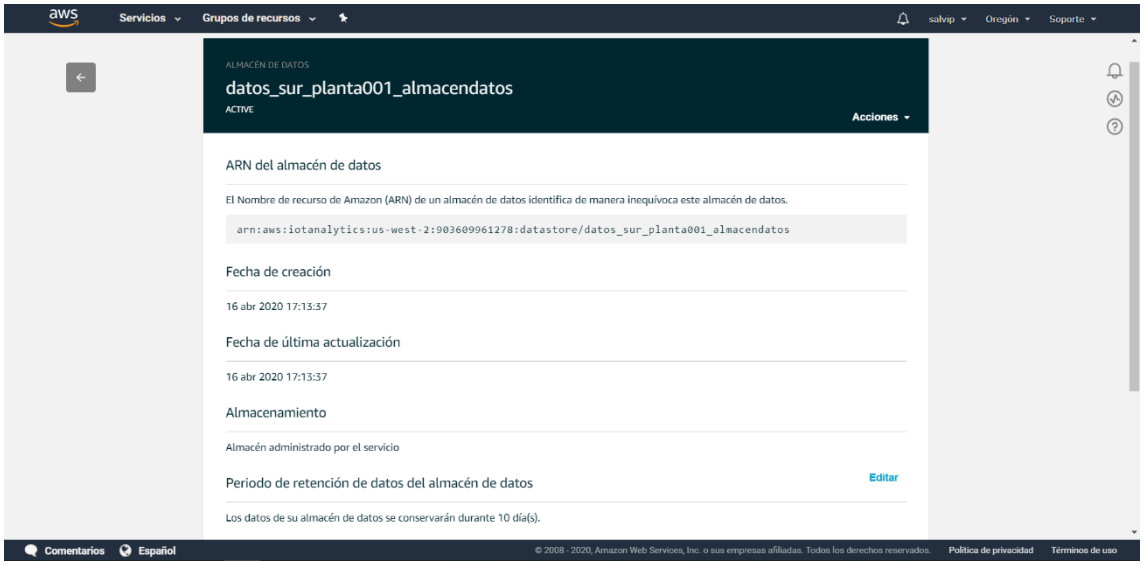

Figura 27. Almacén de datos servicio AWS IoT Analytics

Creación del conjunto de datos a analizar:

Tal y como indica el nombre, un conjunto de datos es un grupo de datos, en este caso con propiedades similares que son seleccionados en base a distintos criterios de manera que facilitan el análisis de estos.

En este trabajo, se han elaborado diversos conjuntos de datos en función de la planta del edificio y del parámetro del cual se desea estudiar el comportamiento.

Un ejemplo de los creados en este trabajo es la creación de un conjunto de datos formado por los campos 'sensor', 'temperatura' y 'timestamp' de los mensajes pertenecientes a la primera planta del edificio Sur. Para ello se ha accedido a la opción 'Crear' del apartado 'Conjuntos de datos' donde se le ha asignado el nombre 'datos_sur_planta001_temperatura_conjuntodatos' y se ha configurado una consulta SQL para la extracción de los datos del almacén 'datos_sur_planta001_almacendatos'. Adicionalmente se le ha configurado una programación encargada de actualizar los datos cada 15 minutos, un periodo de retención de los datos de 10 días y una regla de entrega de contenido al cubo creado en el punto 4.3.7 de este trabajo. De esta manera, cada vez que se actualicen los datos, estos serán enviados a la unidad de almacenamiento asignada donde serán almacenados de manera indefinida.

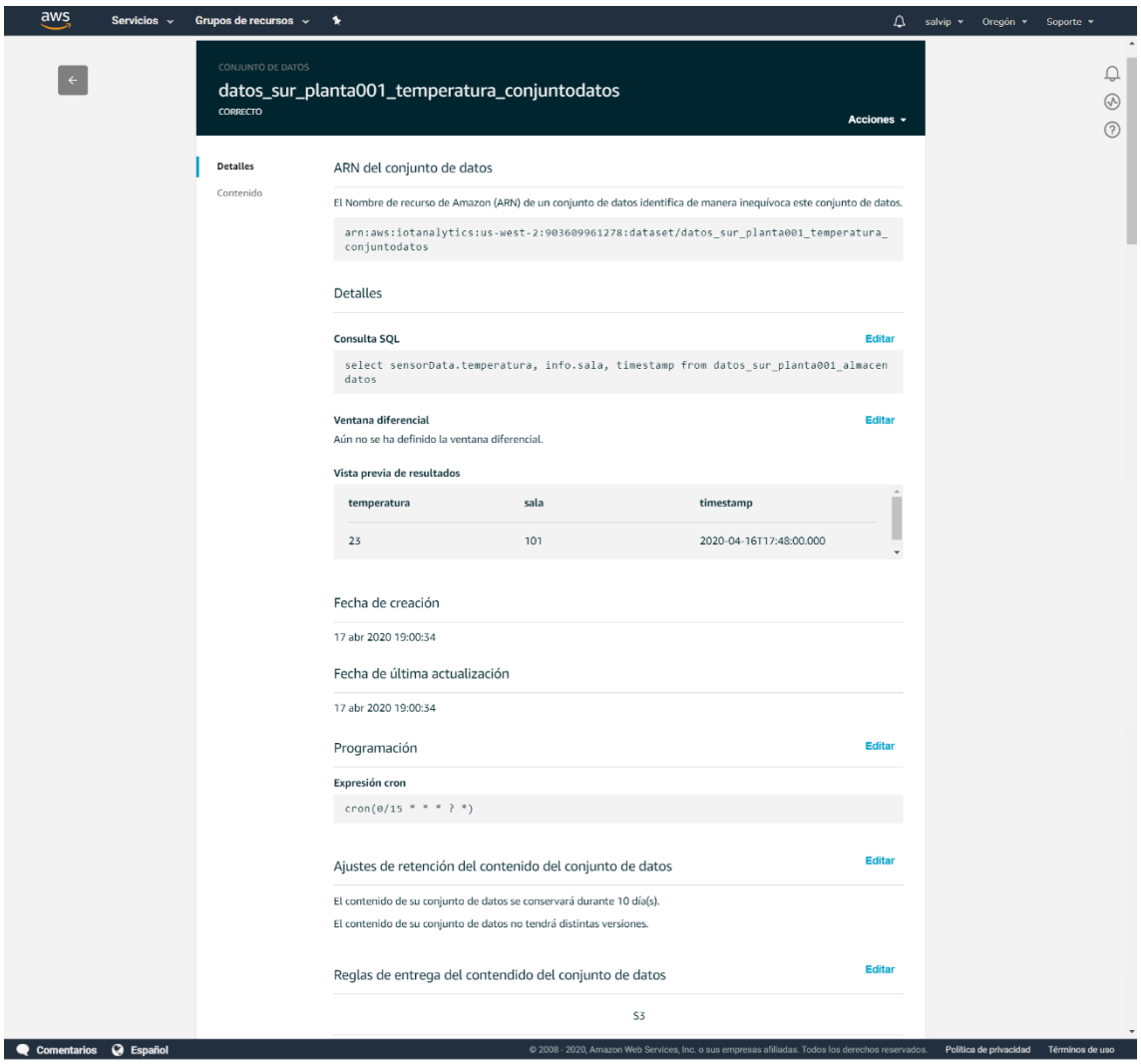

Figura 28. Conjunto de datos servicio AWS IoT Analytics I

4.3.3 AWS Iot Events

El servicio AWS Iot Events permite la activación y realización de acciones o eventos de manera automática. Dichas acciones son activadas cuando se cumplen una serie de condiciones definidas durante el proceso de creación de un modelo detector.

Creación de la entrada del modelo detector:

La creación de un modelo detector requiere la creación de una entrada de datos previa. Dicha entrada es un ejemplo del tipo de mensaje que el modelo recibirá y en base al cual se asientan las condiciones que activan los eventos.

Para la creación de la entrada del modelo detector se ha utilizado el tipo de mensaje que la regla 'iotEvents_temperaturaFueraRango_detector', definida anteriormente en el punto 4.3.1, envía al modelo, de tal forma que la estructura de los mensajes coincida con la estructura definida en la entrada y de este modo, se eviten posibles errores.

Creación del modelo detector:

Un modelo detector se basa en una estructura en la que se definen distintos estados y eventos de transición que permiten el cambio de estado. Al mismo tiempo, cada estado tiene definidas distintas condiciones, que en caso de cumplirse activan de manera automática diversos eventos o acciones, como el envío de notificaciones o la inserción de datos en una tabla DynamoDB.

Para este trabajo, se ha definido un modelo detector que tiene la finalidad de verificar que las lecturas de temperatura de los sensores se encuentren dentro de un rango definido, en caso de que alguna lectura no corresponda con los valores del rango, automáticamente, el modelo pasará del estado 'normal' al estado 'fueraDeRango' y en consecuencia se enviará un correo electrónico avisando del cambio de estado y con información adicional de la lectura que ha provocado el cambio. Adicionalmente, se insertará un nuevo registro en la tabla, creada en el punto 4.3.6 de este trabajo, con la información de la lectura. El modelo se ha configurado de manera que cuando esté en el estado 'fueraDeRango' no pueda cambiar al estado 'normal' hasta que no se obtenga un numero de lecturas consecutivas con valores dentro del rango definido, en caso de cumplir la condición el estado cambiará a 'normal' y se enviará un correo electrónico avisando del cambio de estado. En los anexos se encuentran los diagramas de la lógica interna de ambos estados.

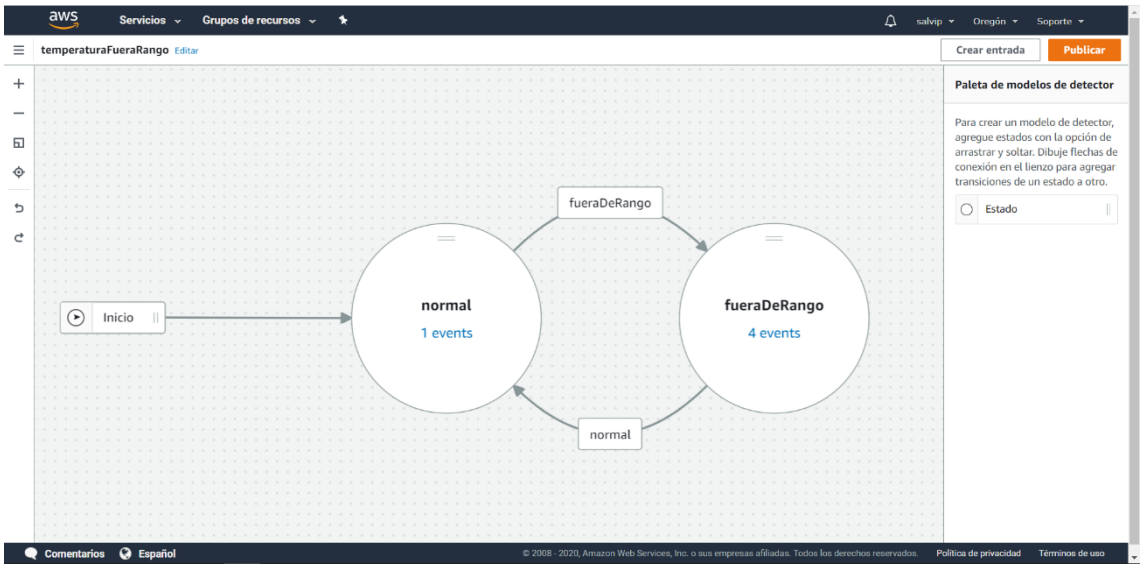

Figura 29. Modelo detector lecturas fuera de rango

4.3.4 Amazon QuickSight

QuickSight dispone de una gran cantidad de herramientas y recursos para la representación de los datos mediante informes y paneles interactivos.

Creación de un panel interactivo:

Uno de los paneles creados a lo largo de este trabajo ha sido el encargado de representar la variación de la temperatura obtenida por los distintos sensores distribuidos por salas de una planta en función del tiempo.

Para ello se ha creado un panel, el cual tiene como fuente de datos el conjunto de datos 'datos_sur_planta001_temperatura_conjuntodatos' creado en el punto 3.3.2. Seguidamente se ha configurado el eje 'x' en función del campo 'timestamp' correspondiente al tiempo y el eje 'y' en función del campo 'temperatura', finalmente se ha configurado la representación de los datos separados por salas, de manera que el resultado final es un gráfico de las evoluciones de la temperatura de las distintas salas.

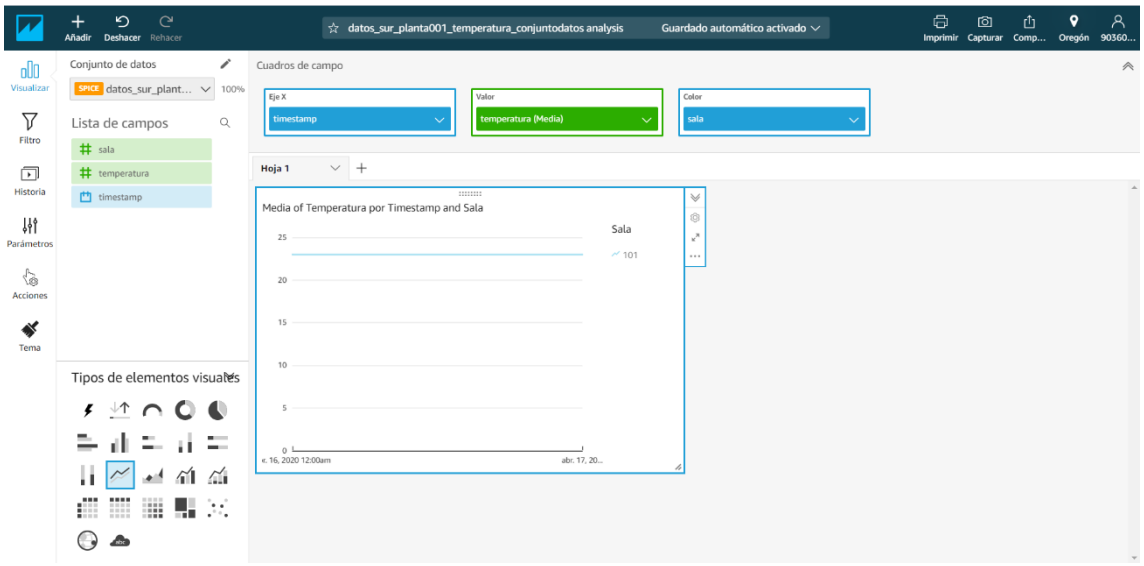

Figura 30. Gráfico representativo de la temperatura en función del tiempo

4.3.5 Amazon Simple Notification Service (SNS)

El servicio Simple Notification Service está basado en un modelo publicador/subscriptor y dispone de una gran variedad de utilidades para el reenvió de mensajes, como es el envío de notificaciones por medio del correo electrónico, función la cual ha sido utilizada en el desarrollo de este trabajo.

Creación de temas:

Para poder llevar a cabo los eventos de envió de notificaciones definidos en el modelo detector creado en el capítulo 4.3.3, por medio de envíos de correos electrónicos, ha sido necesaria la creación de dos temas, 'avisoTemperaturaFueraRango' y 'avisoTemperaturaRestablecida', de tal forma que cuando el modelo detector envíe un mensaje a alguno de estos temas, este será renviado a las subscripciones de estos. En la creación de ambos temas, se ha asignado un nombre y se ha dejado la configuración predeterminada.

Creación de suscripciones asociadas a los temas:

Con el fin de reenviar los mensajes recibidos por el tema a una dirección de correo electrónico ha sido necesario la creación de dos suscripciones asociadas a una cuenta de correo electrónico. Cada una de estas subscripciones, ha sido enlazada a uno de los temas creado anteriormente, de esta forma cuando un tema reenvíe un mensaje lo estará redirigiendo a la bandeja de entrada del correo electrónico suscrito.

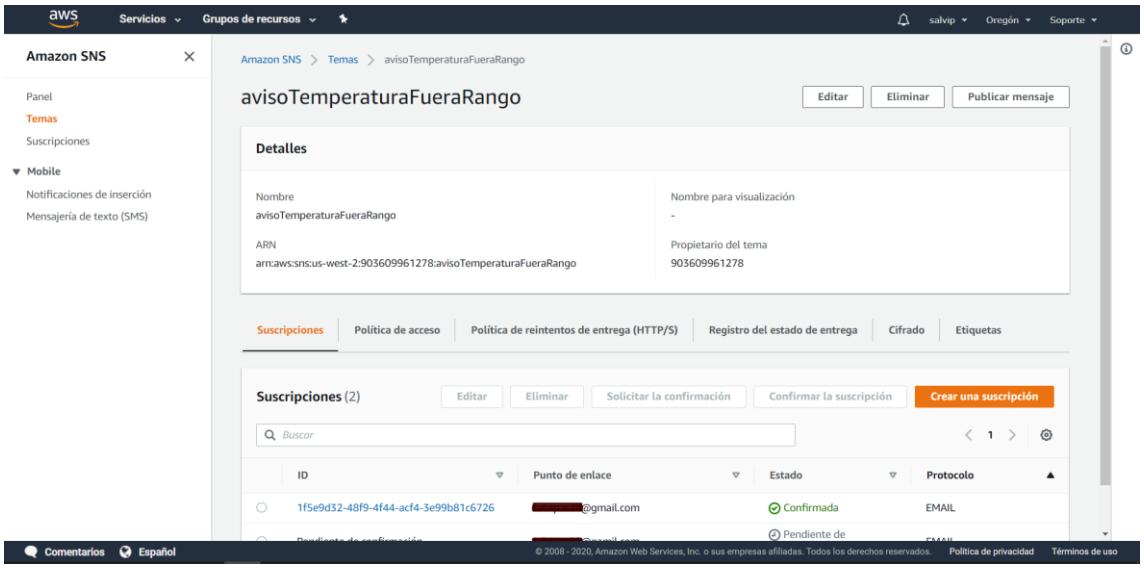

Figura 31. Tema y suscripción servicio Amazon SNS

4.3.6 Amazon DynamoDB

DynamoDB permite la creación de tablas NoSQL, muy útiles en el entorno del Internet de las Cosas debido al gran número de peticiones que esta soporta, tanto de lectura como de escritura.

Creación tabla:

Tal como se expone en el capítulo 4.3.3, durante el proceso de creación del modelo detector se define un evento encargado de insertar los registros de las lecturas de temperatura que están fuera del rango, en una tabla del servicio DynamoDB. Para ello, se ha creado una tabla llamada 'lecturasTemperaturasFueraRango' donde serán almacenados los mensajes enviado por el evento.

Cabe destacar que durante el proceso de creación se han mantenido las opciones predeterminadas, las cuales se pueden modificar en cualquier momento en función a los requisitos de la aplicación.

4.3.7 Amazon S3

El servicio S3 ofrece un gran número de opciones en cuanto a almacenamiento respecta, es por ello, por lo ha sido elegido a la hora de desarrollar este trabajo.

Creación cubo S3:

En el desarrollo del presente trabajo, se ha creado una unidad de almacenamiento llamada 'datos-sur-s3' cuya función es albergar todos los datos recopilados y procesados por el servicio AWS IoT Events. El objetivo es poder acceder a todos los datos procesados sin importar su antigüedad con el fin de poder elaborar informes y paneles interactivos complementarios a los utilizados habitualmente para monitorizar las temperaturas de las distintas salas del edificio.

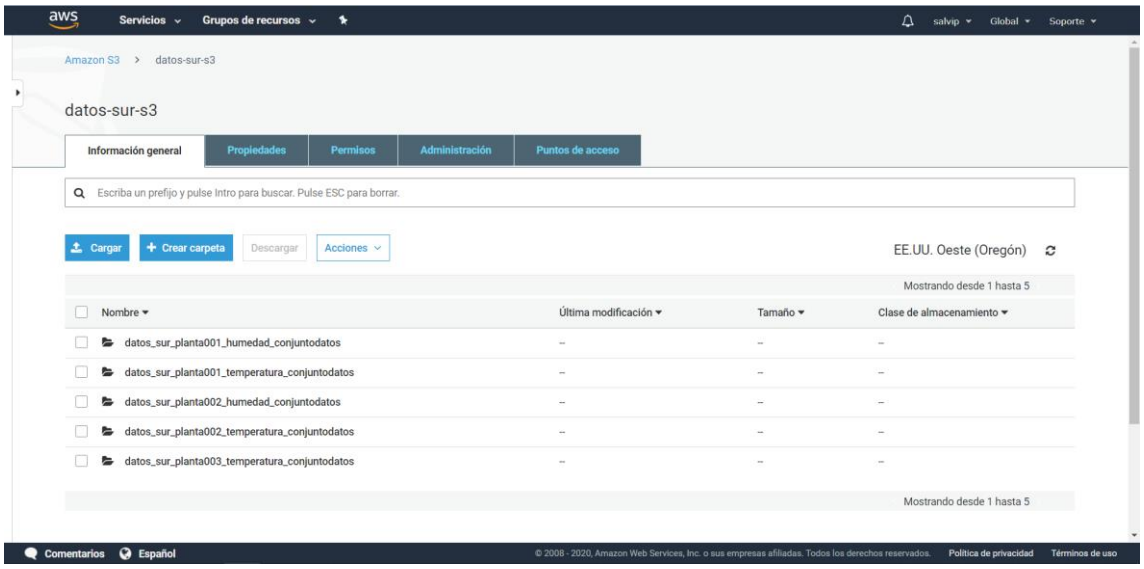

Figura 32. Unidad de almacenamiento servicio Amazon S3

4.4 Pruebas del desarrollo elaborado

Con el fin de garantizar el correcto funcionamiento de la aplicación desarrollada, se ha realizado una serie de pruebas y de comprobaciones con el objetivo de solucionar posibles errores que se hayan pasado por alto en el proceso de diseño y desarrollo. Así mismo, se ha comprobado que las acciones que se llevan a cabo se realizan sin ningún tipo de problema.

Comprobación de la correcta publicación de mensajes, procedentes de la placa NodeMCU, en el tema 'iot/node' del broker alojado en el dispositivo Raspberry Pi:

Para llevar a cabo la comprobación se ha ejecutado el programa 'NodeMCU.py' en la placa de desarrollo NodeMCU, mientras que en el dispositivo Raspberry Pi, mediante el uso de la herramienta de consola, se ha hecho uso del cliente MQTT que ofrece el programa Eclipse Mosquitto para suscribirse al tema 'iot/node'. De esta forma, se ha podido comprobar la correcta recepción de los mensajes enviados por la placa de desarrollo al broker que alberga la Raspberry Pi.

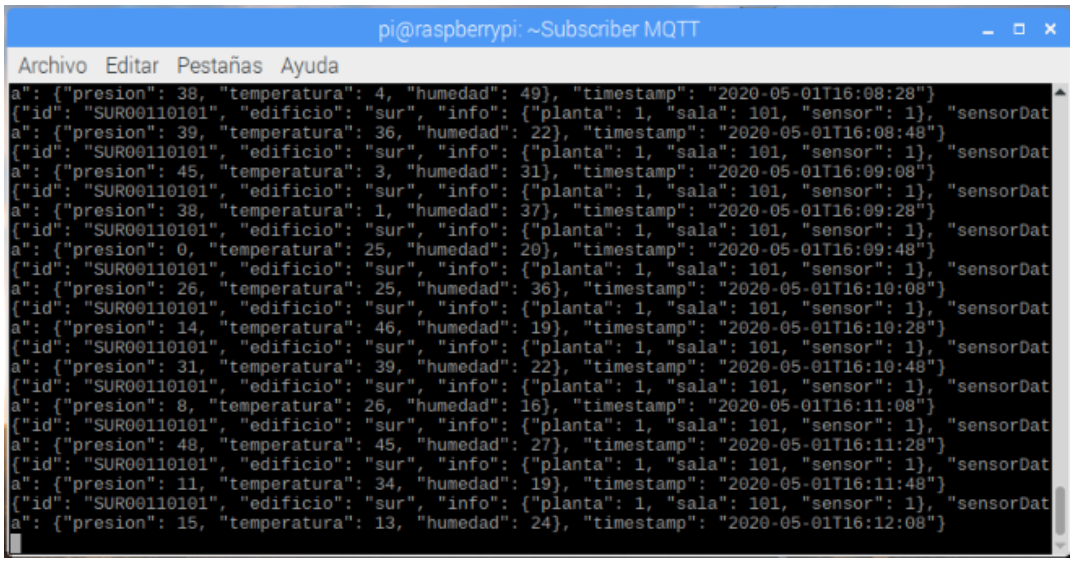

Figura 33. Cliente MQTT

Comprobación del correcto reenvío de los mensajes publicados en el tema 'iot/node' del broker, instalado en la Raspberry Pi, al tema 'iot/lecturasSensores/SUR' del servicio AWS IoT Core:

Esta comprobación se ha realizado en dos partes. La primera parte se ha realizado mediante el uso de los paneles interactivos del apartado 'Monitorización' del servicio AWS IoT Core, en ellos hemos comprobado las distintas conexiones que se han producido, el protocolo que se ha utilizado en las conexiones, así como el número de mensajes publicados y de reglas ejecutas.

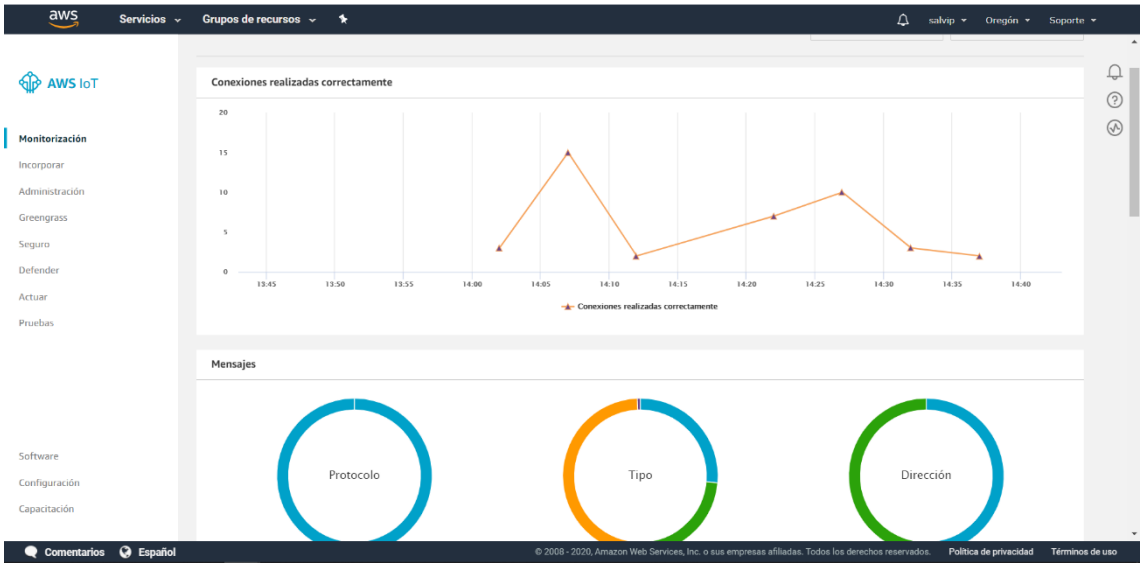

Figura 34. Conexiones realizadas en función del tiempo

En la segunda parte, se ha hecho uso del cliente MQTT que dispone el servicio, el cual ha sido suscrito al tema 'iot/lecturasSensores/SUR', de esta forma se ha podido comprobar la recepción de los mensajes enviados por el dispositivo Raspberry Pi. Cabe destacar que en todo momento el programa 'RaspberryPi.py' se encuentra en ejecución el dispositivo Raspberry Pi.

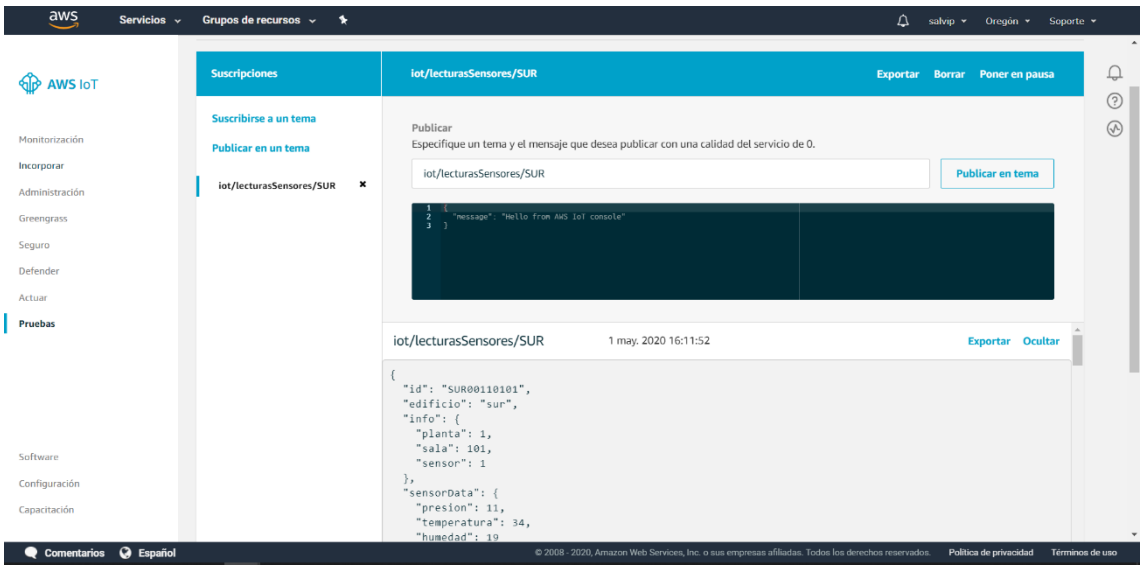

Figura 35. Subscriptor MQTT

Comprobación del contenido de los conjuntos de datos del servicio AWS IoT Analytics:

Esta comprobación es sumamente importante ya que permite comprobar el funcionamiento del modelo implementado en el servicio AWS IoT Analytics. Para ello se accede a los conjuntos de datos creados y configurados con anterioridad, y se comprueba que el contenido de dichos conjuntos corresponde con los datos esperados.

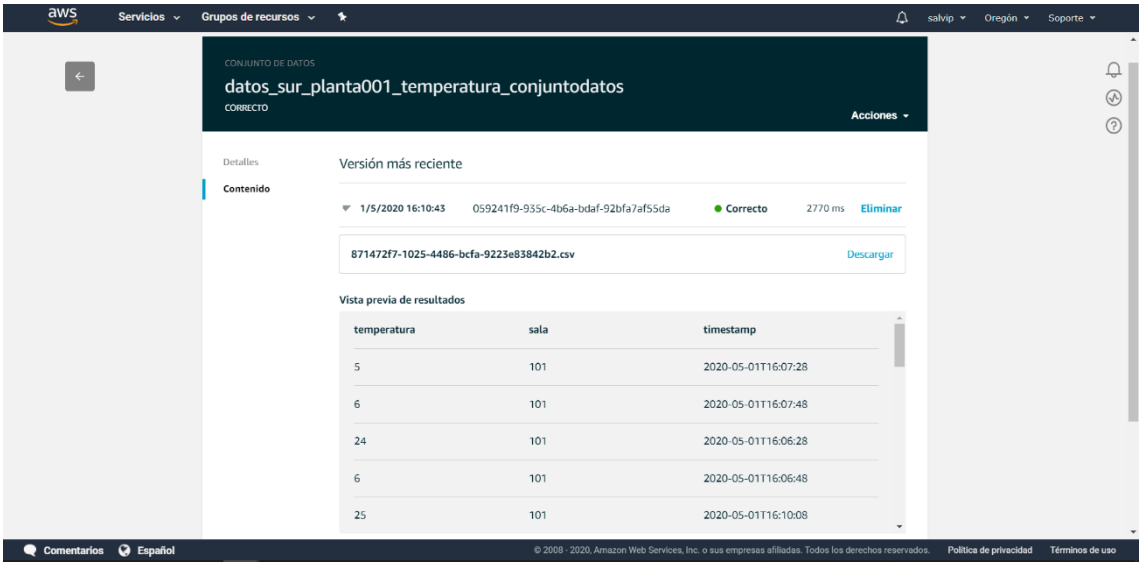

Figura 36. Contenido conjunto de datos

Comprobación de la correcta representación de los datos en los paneles interactivos creados con el servicio Amazon QuickSight:

Una vez, comprobado el contenido de los conjuntos de datos, se comprueba que la representación gráfica, mediante paneles interactivos, corresponda con los datos que forman el conjunto de datos a representar.

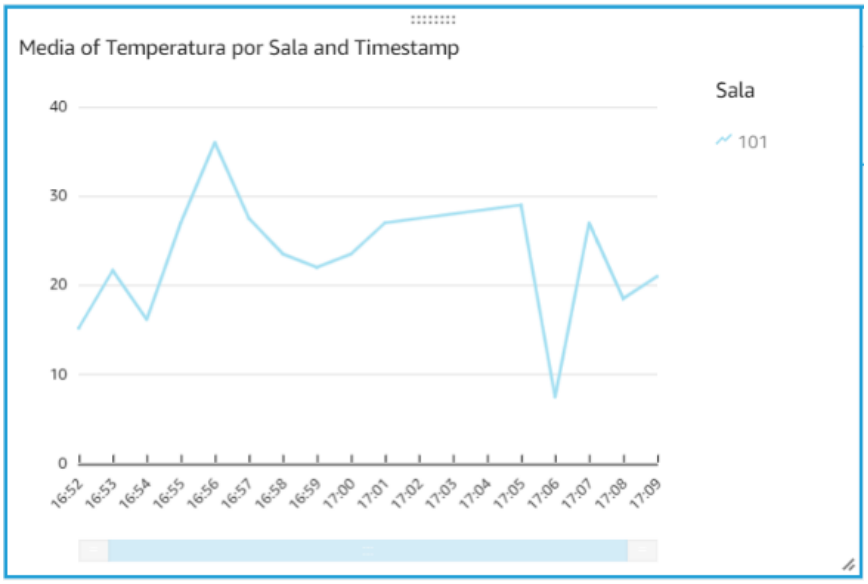

Figura 37. Panel representativo de la temperatura en función del tiempo

Comprobación almacenaje en S3 de las copias de los conjuntos de datos:

De igual forma que la anterior comprobación, tras verificar el contenido de los conjuntos de datos, se comprueba el correcto almacenamiento de las distintas versiones. Para ello, se accede al servicio Amazon S3, donde se localiza la unidad de almacenamiento 'datos-sur-s3' y se comprueba que dicha unidad de almacenamiento contiene las distintas versiones de todos los conjuntos de datos que tienen configurado la unidad.

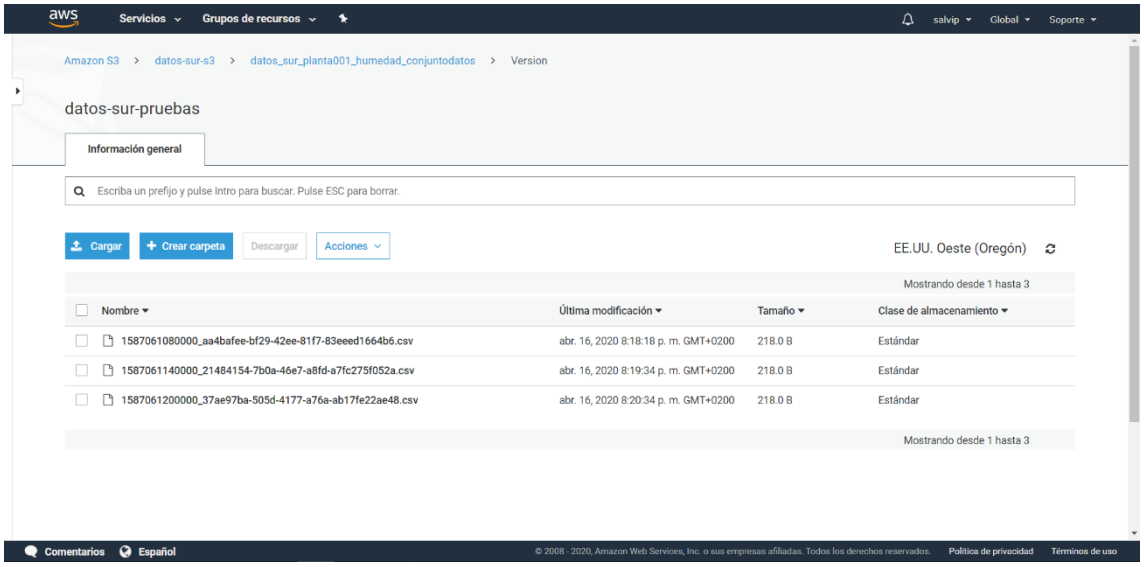

Figura 38. Versiones contenido del almacén de datos

Comprobación del correcto funcionamiento del modelo detector y de sus correspondientes eventos:

Para esta comprobación se han monitorizado los mensajes publicados en el tema 'iot/lecturasSensores/SUR' con el fin de comprobar el comportamiento de los eventos programados en el modelo detector. Paralelamente, se comprueba la bandeja de entrada del correo electrónico asignado como destinario de las notificaciones y se comprueba la recepción de las notificaciones de los cambios de estado mediante mensajes electrónicos.

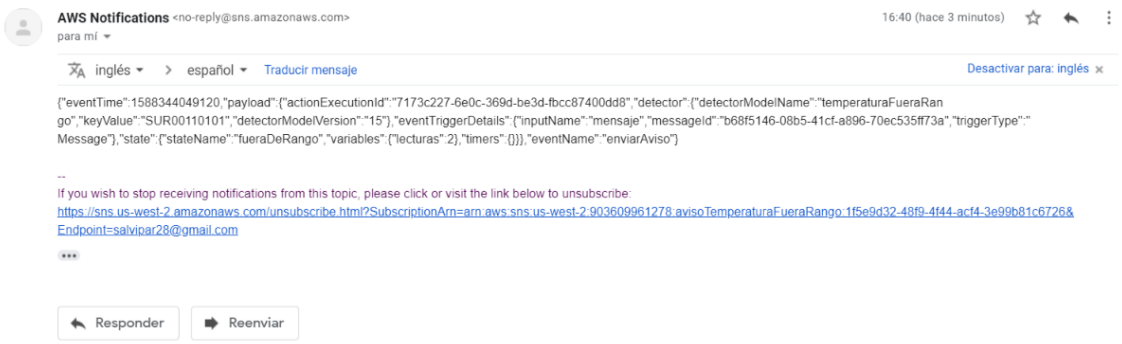

Figura 39. Mensaje de aviso cambio de estado a fueraDeRango

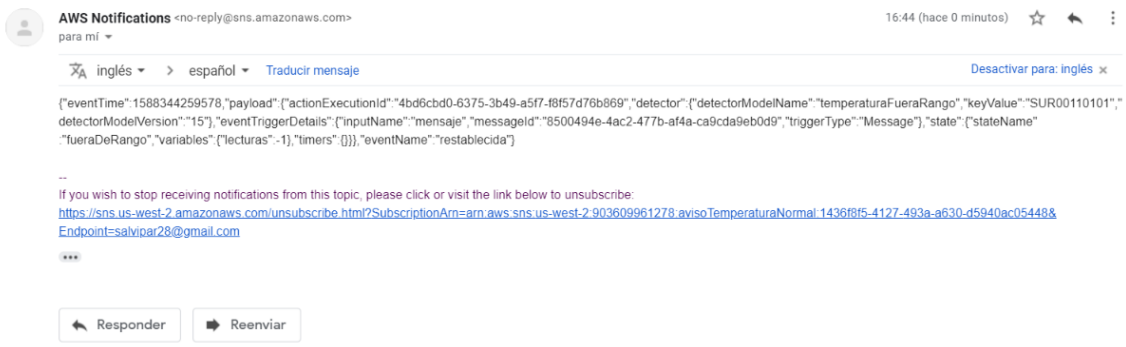

Figura 40. Mensaje de aviso cambio de estado a normal

Por otra parte, se verifica que los mensajes que provocan un cambio del estado ‘normal’ al estado ‘fueraDeRango’ se registran en la tabla ‘lecturasTemperaturasFueraRango’ del servicio Amazon DynamoDB.

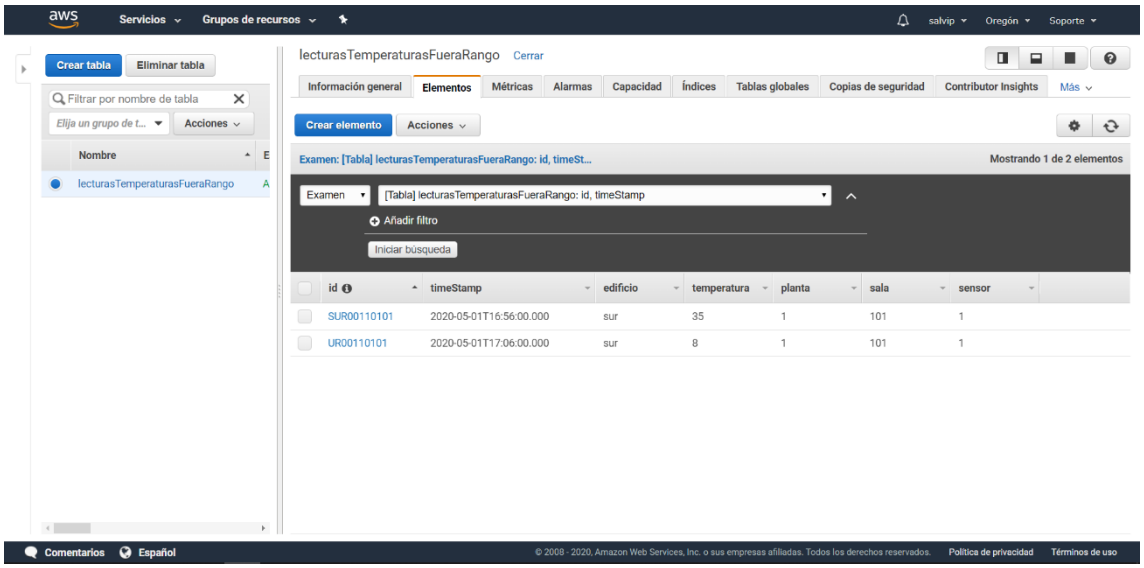

Figura 41. Registros lecturas fuera de rango

Capítulo 5. Aspectos finales y conclusiones

5.1 Opciones de escalabilidad

Uno de los objetivos de este trabajo es poder trasladar la aplicación a distintos casos de uso, sin importar el tamaño y el tráfico a soportar por los recursos que la componen. Para ello, se parte del ejemplo del edificio de oficinas, con el fin de establecer la base y los pilares fundamentales de la aplicación. A partir de este punto se idea un modelo común valido para distintos casos de usos con la misma finalidad y características similares.

En este apartado, se pretende exponer diversas situaciones o puntos de vista en referencia a un posible crecimiento tanto del tráfico como del número de fuentes de datos de la aplicación y cómo afrontarlos.

Un punto crítico a la hora de aumentar el número de sensores y, por tanto, el tráfico generado por las fuentes de datos es la centralización de estos, tarea llevada a cabo por el dispositivo Raspberry Pi. En el momento en el que el dispositivo soporta un tamaño de carga cercano al máximo y un rendimiento alto, se deben buscar alternativas con el fin de evitar posibles pérdidas de información, así como errores. Para ello, la aplicación ha sido diseñada con la finalidad de soportar soluciones alternativas sin afectar ni causar un gran impacto en la estructura de esta.

Una de las soluciones propuestas para llevar a cabo la escalabilidad de esta fase es la sustitución del dispositivo Raspberry Pi, encargado de la centralización de los datos, por un dispositivo o equipo de mayor potencia que permita soportar una mayor carga, la cual no afecte al rendimiento. Para la implementación de esta solución, se debe disponer de un dispositivo con mayor potencia que sea compatible con el protocolo MQTT, el cual se deberá configurar de igual forma que el dispositivo Raspberry Pi.

Otra de las soluciones posibles, es la instalación de un segundo dispositivo Raspberry Pi y en consecuencia la instalación de un segundo servidor MQTT, tal y como se representa en la figura 42. Este tendrá la función de centralizar los datos obtenidos por los nuevos sensores incorporados a la red, en un nuevo flujo de datos. Dicho flujo, deberá tener configurado el mismo destino que el flujo de datos creado por la Raspberry Pi principal.

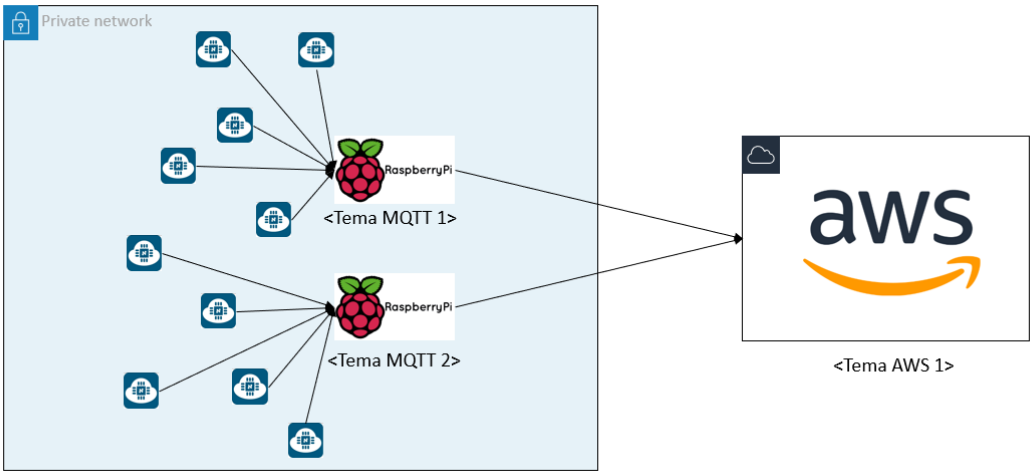

Figura 42. Modelo escalabilidad nivel local

Gracias a estas soluciones, se podría resolver la problemática causada por un posible aumento del tamaño de la red formada por los distintos sensores.

Otra de las situaciones relacionadas con la escalabilidad es la incorporación de nuevas redes privadas, formadas por sensores, y consigo nuevos flujos de datos. En este trabajo, se ha utilizado el caso de uso de un solo edificio de oficinas, pero la aplicación esta ideada con el fin de incorporar más de una fuente de datos, como pueden ser distintos edificios dispersos de manera geográfica. Estas fuentes tienen como destino la publicación en distintos temas del servicio AWS IoT Core. Adicionalmente, por cada fuente de datos, proveniente de una red privada formada por distintos sensores, se debe crear los recursos necesarios para el procesado de los datos, así como las reglas para reenviar los mensajes a los distintos servicios de AWS. Dichos recursos deben ser creados acorde a los recursos ya existentes.

Por tanto, se podrían incorporar tantas fuentes de datos como se desee con tan solo crear los recursos necesarios en la plataforma AWS.

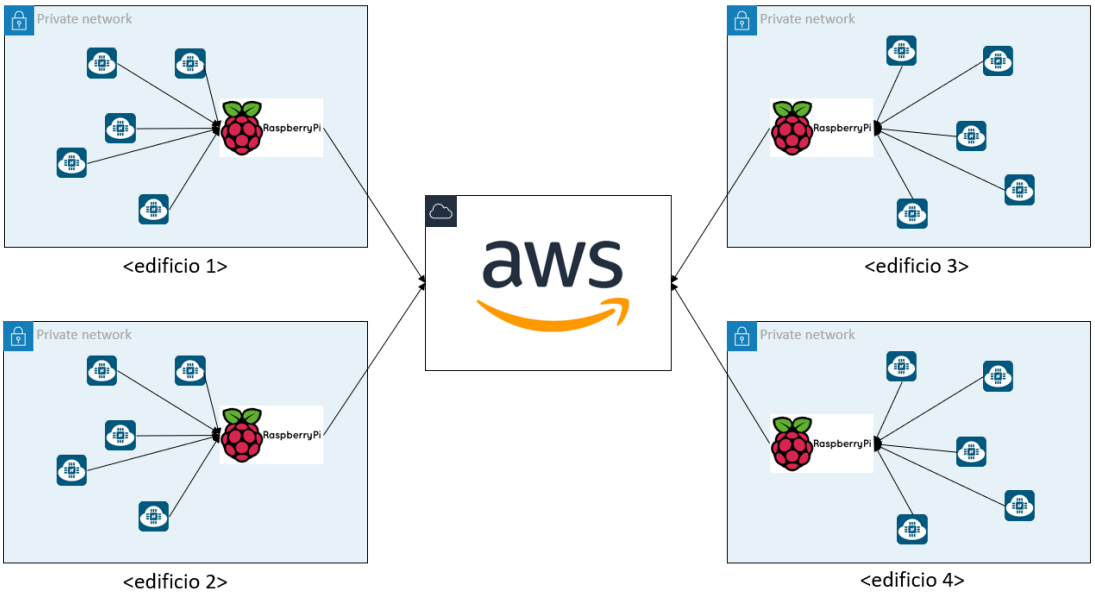

Figura 43. Modelo escalabilidad nivel global

Finalmente, otra de las opciones de escalabilidad de la aplicación, es la incorporación de nuevos servicios de la plataforma AWS con diversos fines. La plataforma cuenta con una gran cantidad opciones, las cuales permiten relacionar e incorporar nuevos servicios de manera sencilla y de esta forma, añadir funcionalidades nuevas a la aplicación sin afectar a la estructura de la aplicación.

5.2 Presupuesto y costes

El presupuesto de la aplicación IoT se basa principalmente en la suma de los precios de los componentes y dispositivos utilizados en la realización de esta, entre los cuales se encuentran, los sensores de temperatura y humedad, la placa de desarrollo NodeMCU y el dispositivo Raspberry Pi. Por tanto, el total del presupuesto dependerá del número de dispositivos y componentes que se utilicen en el desarrollo.

Por otra parte, existe un coste de mantenimiento mensual con referencia al uso de los servicios de la plataforma Amazon Web Services, el cual puede variar en función del uso y rendimiento de la aplicación.

Precios componentes y dispositivos:

- Modulo sensor temperatura y humedad (1,03 – 1,30€ Aliexpress)

Figura 44. Precio sensor temperatura y humedad

- Dispositivo NodeMCU (4,90€ BricoGeek)

Figura 45. Precio dispositivo NodeMCU

- Precio dispositivo Raspberry Pi 3 B+ (44,89€ PcComponentes)

Figura 46. Precio dispositivo Raspberry Pi 3 B+

Costes servicios Amazon Web Services:

A continuación, se detalla una lista con los precios de los recursos, utilizados en la elaboración de este trabajo, pertenecientes a los distintos servicios que ofrece la plataforma AWS.

- Costes AWS IoT Core:
  - Servicio de conectividad: 0,08 USD por millón de minutos de conexión
  - Tratamiento de mensajes: 0,70 USD por millón de mensajes
  - Servicios de registro y sombra del dispositivo: 1,25 USD por millón de operaciones
  - Reglas activadas: 0,15 USD por millón de reglas activadas
  - Reglas ejecutadas: 0,15 USD por millón de reglas ejecutadas

- Costes AWS IoT Analytics:
  - Datos procesados: 0,20 USD por GB
  - Datos procesados almacenados: 0,03 USD por GB
  - Datos escaneados: 6,50 USD por TB

- Costes AWS IoT Events:
  - Datos evaluados (hasta 5.000.000 de mensajes): 1,10 USD por cada 10.000 mensajes
  - Datos evaluados (más de 5.000.000 de mensajes): 0,80 USD por cada 10.000 mensajes

- Costes Amazon QuickSight:
  - Autores: 18 USD por usuario por mes
  - Lectores: 0,30 – 5,00 USD por sesión por usuario por mes

- Costes Amazon Simple Notification Service:
  - Notificaciones email: 2,00 USD por cada 100.000
  - Transferencia entrante de datos: gratuito
  - Transferencia saliente de datos: 0,05-0,09 USD por GB

- Costes Amazon DynamoDB:
  - Escrituras: 1,25 USD por millón de escrituras
  - Lecturas: 0,25 USD por millón de lecturas
  - Almacenamiento: 0,25 USD por GB

- Costes Amazon S3:
  - Hasta 50 TB: 0,023 USD por GB
  - De 50 TB hasta 500 TB: 0, 022 USD por GB
  - Mas de 500 TB: 0,021 USD por GB

Las tarifas de la plataforma se aplican en dólares (USD) pero existe la posibilidad de pagar en euros.

5.3 Propuesta de trabajo futuro

Como propuesta de trabajo futuro, se plantean diversas mejoras, así como la incorporación de nuevos servicios que aporten valor a la aplicación. El objetivo es, por un lado, robustecer y complementar la aplicación con funcionalidades nuevas y, por otro lado, dotarla de una mayor seguridad.

Para ello, se definen tres puntos claves a atajar que permitirán llevar a cabo la siguiente fase con éxito:

El primer punto, es la optimización de los recursos del servicio AWS IoT Analytics, para ello se pretende optimizar el modelo actual de manera que permita mejorar la utilización de los recursos y de este modo se requieran un menor número de estos.

El segundo punto, hace referencia a la seguridad en las distintas partes de la aplicación. Con el fin de garantizar un alto nivel de seguridad se deberá implementar el cifrado sobre las conexiones MQTT de la red privada, de manera que las conexiones entre las placas de desarrollo y el dispositivo Raspberry Pi estén cifradas y consigo los datos trasmitidos. Adicionalmente, se deberán definir políticas de seguridad y roles más estrictos en los servicios de la plataforma AWS, de este modo restringir los accesos y las acciones permitidas. Por último, se incorporará el servicio AWS IoT Device Defender con el fin de garantizar una mayor seguridad y detectar posibles vulnerabilidades o posibles ataques y de esta forma actuar de manera temprana.

Por último, el tercer punto, plantea la incorporación de nuevos servicios, como AWS Kinesis o Lamda, los cuales permiten elaborar un modelo de monitorización en tiempo real para los casos en que se requiera una monitorización y procesado continuo de los datos con la menor latencia posible.

5.4 Conclusiones

Gracias a la elaboración de este trabajo se ha logrado superar una serie de objetivos definidos al principio de este. Entre los cuales se encuentra, la planificación y el diseño de las distintas fases de un proyecto, así como el diseño y desarrollo de una aplicación IoT.

Respecto a la planificación y el diseño del proyecto se ha partido de una fase inicial, la cual tenía como objetivo encontrar una idea que a la vez sirviese de punto de partida, seguida de una fase de investigación a la que le ha seguido una fase de diseño y análisis, y finalmente se han llevado a cabo, las fases de desarrollo y pruebas. Esto ha permitido separar las distintas tareas en diferentes fases permitiendo así, una mayor gestión de tiempo, pero, sobre todo una mejor organización.

Por otra parte, el hecho de desarrollar una aplicación IoT completa ha permitido un mayor aprendizaje, debido al trabajo en profundidad realizado en las fases de obtención, centralización, procesado y análisis de los datos. De esta forma, se ha podido conocer más de cerca los procedimientos realizados en cada una de las fases.

Finalmente, se ha logrado incrementar los conocimientos en relación con las tecnologías utilizados en el desarrollo del trabajo, como son las placas de desarrollo NodeMCU, el dispositivo Raspberry Pi, el protocolo MQTT y los distintos servicios de la plataforma AWS.

Capítulo 6. Bibliografía

[1] ABC tecnología, "Qué es Raspberry Pi y para qué sirve" https://www.abc.es/tecnologia/informatica-hardware/20130716/abci-raspberry-como-201307151936.html [En línea]. 17:15 12/04/2020

[2] Amazon Web Services, "Introducción a AWS IoT Core" https://docs.aws.amazon.com/es_es/iot/latest/developerguide/iot-gs.html [En línea]. 18:13 05/04/2020

[3] Amazon Web Services, "Guía del usuario de AWS IoT Analytics" https://docs.aws.amazon.com/es_es/iotanalytics/latest/userguide/analytics-ug.pdf [En línea]. 16:13 07/04/2020

[4] Amazon Web Services, "¿Qué es AWS?" https://aws.amazon.com/es/what-is-aws/ [En línea]. 14:14 10/04/2020

[5] Amazon Web Services, "Internet de las cosas" https://aws.amazon.com/es/iot/ [En línea]. 16:48 10/04/2020

[6] Amazon Web Services, "Información general AWS IoT Core" https://aws.amazon.com/es/iot-core/ [En línea]. 16:49 10/04/2020

[7] Amazon Web Services, "¿Qué es AWS IoT Analytics?" https://docs.aws.amazon.com/es_es/iotanalytics/latest/userguide/welcome.html [En línea]. 10:08 11/04/2020

[8] Amazon Web Services, "Detecte y responda a eventos de IoT" https://aws.amazon.com/es/iot-events/ [En línea]. 11:41 11/04/2020

[9] Amazon Web Services, "Amazon QuickSight" https://aws.amazon.com/es/quicksight/ [En línea]. 11:44 11/04/2020

[10] Amazon Web Services, "Amazon S3" https://aws.amazon.com/es/s3/ [En línea]. 11:52 11/04/2020

[11] Amazon Web Services, "AWS | Servicio de notificaciones Push (SNS)" https://aws.amazon.com/es/sns/ [En línea]. 17:48 11/04/2020

[12] Amazon Web Services, "AWS | Servicio de base de datos gestionada NoSQL (DynamoDB)" https://aws.amazon.com/es/dynamodb [En línea]. 10:18 12/04/2020

[13] Amazon Web Services, "Getting started with the AWS IoT Events console" https://docs.aws.amazon.com/es_es/iotevents/latest/developerguide/iotevents-getting-started.html [En línea]. 17:43 16/04/2020

[14] Amazon Web Services, "Como crear una tabla NoSQL y realizar consultas - AWS" https://aws.amazon.com/es/getting-started/hands-on/create-nosql-table/ [En línea]. 17:09 18/04/2020

[15] Amazon Web Services, "Creación de una regla con una acción de DynamoDB – AWS IoT" https://aws.amazon.com/es/getting-started/hands-on/create-nosql-table/ [En línea]. 21:13 19/04/2020

[16] Amazon Web Services, "Precios de Amazon QuickSight" https://aws.amazon.com/es/quicksight/pricing/#FAQs [En línea]. 19:02 07/05/2020

[17] Amazon Web Services, "Precios de AWS IoT Core" https://aws.amazon.com/es/iot-core/pricing/ [En línea]. 18:53 07/05/2020

[18] Amazon Web Services, "Precios de AWS IoT Analytics" https://aws.amazon.com/es/iot-analytics/pricing/ [En línea]. 18:58 07/05/2020

[19] Amazon Web Services, "Precios de AWS IoT Events" https://aws.amazon.com/es/iot-events/pricing/ [En línea]. 18:58 07/05/2020

[20] Amazon Web Services, "Amazon DynamoDB: Precios" https://aws.amazon.com/es/dynamodb/pricing/ [En línea]. 18:59 07/05/2020

[21] Amazon Web Services, "Precios Amazon Web Services S3 | Amazon Simple Storage Service" https://aws.amazon.com/es/s3/pricing/ [En línea]. 19:00 07/05/2020

[22] Amazon Web Services, "Precios de Amazon Simple Notificacion Service (SNS) - Amazon Web Services (AWS)" https://aws.amazon.com/es/sns/pricing/ [En línea]. 19:01 07/05/2020

[23] BricoGeek, "NodeMCU v3 – ESP8266" https://tienda.bricogeek.com/wifi/1033-nodemcu-v3-esp8266.html [En línea]. 18:09 06/05/2020

[24] Computer Hoy, "Lo que debes saber sobre la Raspberry Pi 4 antes de lanzarte a comprar una" https://computerhoy.com/listas/tecnologia/debes-saber-raspberry-pi-4-antes-lanzarte-comprar-446889 [En línea]. 17:15 12/04/2020

[25] Clouding.io, "Introducción a Eclipse Mosquitto" https://clouding.io/hc/es/articles/360010704900-Introducci%C3%B3n-a-Eclipse-Mosquitto [En línea]. 16:58 13/04/2020

[26] Eclipse Mosquitto, "Eclipse Mosquitto TM An open source MQTT broker" https://mosquitto.org/ [En línea]. 16:58 13/04/2020

[27] Luis Llamas, "NodeMCU, la popular placa de desarrollo con ESP8266", https://www.luisllamas.es/esp8266-nodemcu/ [En línea]. 17:00 02/04/2020

[28] Luis Llamas, "¿Qué es MQTT? Su importancia como protocolo IoT", https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ [En línea]. 17:17 02/04/2020

[29] PcComponentes, "Raspberry Pi 3 Model BPlus" https://www.pccomponentes.com/raspberry-pi-3-modelo-bplus [En línea]. 18:12 06/05/2020

[30] Programafacil.com, "NodeMCU tutoría paso a paso desde cero" https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/#Conectar_el_NodeMCU_al_PC [En línea]. 11:52 02/01/2020

[31] Promotec, "Modelos ESP8266" https://www.prometec.net/modelos-esp8266/ [En línea]. 12:25 13/04/2020

[32] Redes zone, "El enorme mundo de Raspberry Pi" https://www.redeszone.net/raspberry-pi/enorme-mundo-raspberry-pi/ [En línea]. 18:42 12/04/2020

[33] Silicon Labs, "CP210x USB to UART Bridge VCP Drivers" https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/#Conectar_el_NodeMCU_al_PC [En línea]. 11:52 02/01/2020

Capítulo 7. Anexos

7.1 Glosario

- **AWS** (Amazon Web Services): Plataforma digital basada en la nube, que ofrece una gran cantidad y variedad de servicios enfocados al desarrollo de las tecnologías.
- **Cloud computing**: Tecnología que agrupa los servicios relacionados con la informática y la programación en el ámbito del internet.
- **IoT** (Internet of Things): Tecnología que permite interconectar miles de dispositivos y objetos, como pueden ser sensores o dispositivos electrónicos, mediante el uso de una red, bien sea privada o pública.
- **MQTT** (Message Queue Telemetry Transport): Protocolo de comunicación basado en un modelo publicador/subscriptor, que permite la comunicación de miles de dispositivos.
- **Open Source**: Se asocia normalmente a un tipo de software, el cual está libre de restricciones, por lo que permite su estudio, modificación y distribución de este a cualquier usuario.
- **QoS** (Quality of Service): Mecanismo que permite la optimización de la red, mediante el etiquetado del tráfico en distintos niveles, con el fin de asegurar la calidad del servicio.
- **TCP/IP**: Grupo de protocolos que permite la conexión y la transferencia de datos entre sistemas informáticos e internet de manera fiable.

7.2 Desglose código NodeMCU.py

El código del programa NodeMCU.py se compone principalmente de tres partes, una primera parte formada por la función principal, seguidamente, de una segunda parte compuesta por las funciones de conectar y publicar, y finalmente, una última parte, formada por las funciones de generar el mensaje y de generar el time stamp.

El funcionamiento de la función principal, denominada main, es el siguiente: en primer lugar, inicializa las variables con los valores necesarios para llevar a cabo la conexión y la publicación de mensajes en el tema 'iot/node' del broker MQTT. Seguidamente hace una llamada a la función conectar y en el caso de no haber error sigue con la ejecución de un bucle 'while', el cual se ha condicionado de manera que se ejecute siempre. Finalmente, en dicho bucle, realiza las acciones de generar el mensaje y de publicarlo.

```
5
6   def main():
7       MQTT_BROKER = "192.168.30.97"
8       MQTT_PORT = 1883
9       MQTT_TOPIC = "iot/node"
10      mensaje = "hola"
11      cliente = mqtt.Client("")
12
13      conectar(cliente, MQTT_BROKER, MQTT_PORT)
14
15      i = 1
16      while i==1 :
17        mensaje = generarMensaje()
18        publicar(cliente, MQTT_TOPIC, mensaje)
19        time.sleep(60)
20
21      cliente.disconnect()
22
```

Figura 47. Función principal

Las funciones conectar y publicar tienen el objetivo de conectar con el servidor MQTT o broker y de publicar el mensaje recibido en el tema que se le indica mediante los parámetros recibidos, respectivamente.

```
22
23  def conectar(cliente, MQTT_BROKER, MQTT_PORT):
24      cliente.connect(MQTT_BROKER, MQTT_PORT)
25      print("Conectado al broker: ", MQTT_BROKER)
26
27  def publicar(cliente, MQTT_TOPIC, mensaje):
28      cliente.publish(MQTT_TOPIC, mensaje)
29      print("Publicado el mensaje: " + mensaje + " en el topic: " + MQTT_TOPIC)
30
```

Figura 48. Funciones conectar y publicar

Respecto la función 'generarMensaje' es la encargada de elaborar el mensaje que será publicado por la placa de desarrollo en el tema del broker alojado en el dispositivo Raspberry Pi. Esta función hace uso de la función 'generarTimeStamp', la cual tiene la función de obtener la fecha y hora del momento en que se ejecuta la función y de adaptar el formato de la variable 'timeStamp'.

En relación con la realización de este trabajo, los valores de las variables 'temperatura', 'humedad' y 'presión' se han generado de manera aleatoria de un rango definido de posibles valores de entre 1 y 40, en lugar de obtener los valores de lecturas de los sensores instalados. La razón principal es obtener valores distintos y de esta forma poder desarrollar y probar la aplicación en entornos menos probables.

```
31  def generarMensaje():
32      idSensor = "SUR00110101"
33      planta = 1
34      sala = 101
35      sensor = 1
36      temperatura = random.randrange(0, 40)
37      humedad = random.randrange(0, 50)
38      presion = random.randrange(0, 50)
39
40      timestamp = generarTimeStamp()
41
42      mensajeJSON = {
43          "id": idSensor,
44          "edificio" : "sur",
45          "info" : {
46              "planta" : planta,
47              "sala" : sala,
48              "sensor" : sensor
49          },
50          "sensorData": {
51              "presion": presion,
52              "temperatura": temperatura,
53              "humedad" : humedad
54           },
55          "timestamp" : timestamp
56      }
57
58      mensaje = json.dumps(mensajeJSON)
59      return mensaje
60
61  def generarTimeStamp():
62
63      fecha = time.strftime("%Y-%m-%d")
64      hora = time.strftime("%H:%M:%S")
65      timestamp = fecha + "T" + hora + ".000"
66      return timestamp
67
```

Figura 49. Funciones generación mensajes

7.3 Desglose código RaspberryPi.py

La función del programa RaspberryPi.py es suscribirse al tema 'iot/node' del broker MQTT, recibir los mensajes publicados en el tema y publicarlos en el tema 'iot/lecturasSensores/SUR'

del servicio AWS IoT Core. Para llevar a cabo las funciones relacionadas con la publicación de mensajes en el tema perteneciente al servicio AWS IoT Core se hace uso de otro programa desarrollado llamado 'publicador_aws.py'.

El código del programa RaspberryPi.py se estructura en dos partes, la primera alberga la función principal o main y la segunda parte alberga las funciones relacionadas con el protocolo MQTT.

La función main en primer lugar inicializa las variables de los parámetros necesarios para la conexión con el servidor MQTT, seguidamente realiza una llamada a la función conectar y si no se produce ningún error realiza una llamada a la función suscribirse, finalmente inicializa el loop y el programa queda a la espera de recibir los mensajes publicados en el tema 'iot/node'.

```
4
5   def main():
6       MQTT_BROKER = "192.168.30.97"
7       MQTT_PORT = 1883
8       MQTT_TOPIC = "iot/node"
9       cliente = mqtt.Client("")
10      cliente.on_message = on_message
11
12      conectar(cliente, MQTT_BROKER, MQTT_PORT)
13      subscribirse(cliente, MQTT_TOPIC)
14
15      cliente.loop_start()
16      #time.sleep(600)
17      #cliente.loop_stop()
18
```

Figura 50. Función principal RaspberryPi.py

La segunda parte del código RaspberryPi.py, está formado por la función 'conectar', la cual es la encargada de realizar la conexión con el servidor MQTT, la función 'subscribirse', encargada de subscribirse al tema 'iot/node' del broker y la función 'on_message', encargada de recibir los mensajes publicados del tema suscrito y pasárselos a la función 'publicar', la cual tiene el objetivo de ejecutar paralelamente el programa 'publicador_aws.py'.

```
18
19  def conectar(cliente, MQTT_BROKER, MQTT_PORT):
20      cliente.connect(MQTT_BROKER, MQTT_PORT)
21      print("Conectado al broker: ", MQTT_BROKER)
22
23  def subscribirse(cliente, MQTT_TOPIC):
24      cliente.subscribe(MQTT_TOPIC)
25      print("Suscrito al tema: ", MQTT_TOPIC)
26
27  def on_message(client, userdata, message):
28      mensajeRecibido = message.payload.decode("utf-8")
29      print("Mensaje recibido: " + str(mensajeRecibido))
30
31      publicar(mensajeRecibido)
32
33  def publicar(mensajeRecibido):
34      result = subprocess.run(["python", "publicador_aws.py"], input=mensajeRecibido.encode())
35
```

Figura 51. Funciones acciones protocolo MQTT

Por otro lado, el programa 'publicador_aws.py', el cual se ejecuta en paralelo, está estructurado de manera similar al programa 'RaspberryPi.py'. Este programa se divide en dos partes, una primera parte formada por la función principal o main y una segunda parte, que alberga las funciones relacionadas con la conexión con el servicio AWS IoT Core.

La función 'main', en primer lugar, inicializa las variables que van a ser utilizadas en el resto del programa con los valores que permiten la conexión y la publicación de mensajes, así como las rutas del certificado y las claves para cifrar los mensajes de la conexión. A continuación, hace

una llamada a la función 'input', la cual forma parte de las librerías de Python, que obtiene los parámetros adjuntos en la llamada de la ejecución de este programa, en este caso obtiene el mensaje a publicar. Seguidamente, hace uso de la función 'tls_set' que configura el cifrado de la conexión, de la función 'connect' definida en el código, e inicializa el bucle encargado de publicar en el tema 'iot/lecturasSensores/SUR', mediante el uso de la función 'publish' en el caso de que la conexión con el servicio AWS IoT Core se haya realizado con éxito, en caso contrario muestra un mensaje de aviso indicando que no se ha podido establecer la conexión.

```
def main():
    mqtt_clie = mqtt.Client(protocol=mqtt.MQTTv311)

    mqtt_clie.on_connect = on_connect
    mqtt_clie.on_disconnect = on_disconnect
    mqtt_clie.on_log = on_log
    mqtt_clie.error_str = error_str

    awshost = "aykwydm6rjuda-ats.iot.us-west-2.amazonaws.com"
    awsport = 8883
    topic = "iot/lecturasSensores/SUR"

    clientId = "edificioSUR"
    thingName = "edificioSUR"

    caPath = R"C:\Users\Salva\Downloads\certificado_edificioSUR\AmazonRootCA1.pem"
    certPath = R'C:\Users\Salva\Downloads\certificado_edificioSUR\81dc8a9b51-certificate.pem.crt'
    keyPath = R'C:\Users\Salva\Downloads\certificado_edificioSUR\81dc8a9b51-private.pem.key'

    mensajeRecibido = input()

    mqtt_clie.tls_set(caPath, certPath, keyPath, cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
    mqtt_clie.connect(awshost, awsport, keepalive=60)
    mqtt_clie.loop_start()

    time.sleep(2)

    if connflag == True:
        mqtt_clie.publish(topic, mensajeRecibido, qos=1)
        print("Mensaje publicado en " + topic + ": " + mensajeRecibido +"\n")
    else:
        print("No se pudo establecer conexion\n")
```

Figura 52. Función principal publicado_aws.py

Respecto, a la segunda parte, está compuesta por la función 'on_connect', que se encarga de conectar con el servicio AWS IoT Core, la función 'on_disconnect', que se encarga de cerrar la conexión establecida, la función 'on_log', encargada de mostrar los mensajes de las ejecuciones que el programa realiza y por último la función 'error_str', encargado de cambiar el formato de los errores recibidos. Todas las funciones, son utilizadas en la función principal.

```
def on_connect(client, userdata, flags, rc):
    print("\n")
    print("Conectando a AWS Amazon Web Service")
    print("Resultado de la conexion: " + error_str(rc) + "\n")

    if rc == 0: #Si es 0 la conexión ha tenido éxito
        global connflag
        connflag = True

def on_disconnect(client, userdata, rc):
    if rc != 0:
        print("Conexión inesperada\n")

    print("Resultado de la desconexion con la plataforma: " + error_str(rc))
    global connflag
    connflag = False
    print("Desconectado\n")

def on_log(client, userdata, level, buf):
    print(buf)

def error_str(rc):
    return '{}: {}'.format(rc, mqtt.error_string(rc))
```

Figura 53. Funciones publicador_aws.py

7.4 Instalación servidor MQTT mediante Eclipse Mosquitto

El proceso de instalación del servidor MQTT se lleva a cabo mediante la instalación del programa Eclipse Mosquitto. Para ello, se deben ejecutar los siguientes comandos en la consola del dispositivo Raspberry Pi.

En primer lugar, se debe importar la clave de firma del paquete del repositorio.

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
```

Seguidamente, se debe hacer disponible para apt.

```
cd /etc/apt/sources.list.d/
```

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
```

Y finalmente, se debe instalar

```
apt-get install mosquitto
```

Una vez termine el proceso de instalación, el servidor MQTT estará disponible para su uso.

7.5 Diseño estructura interna del servicio AWS Iot Core

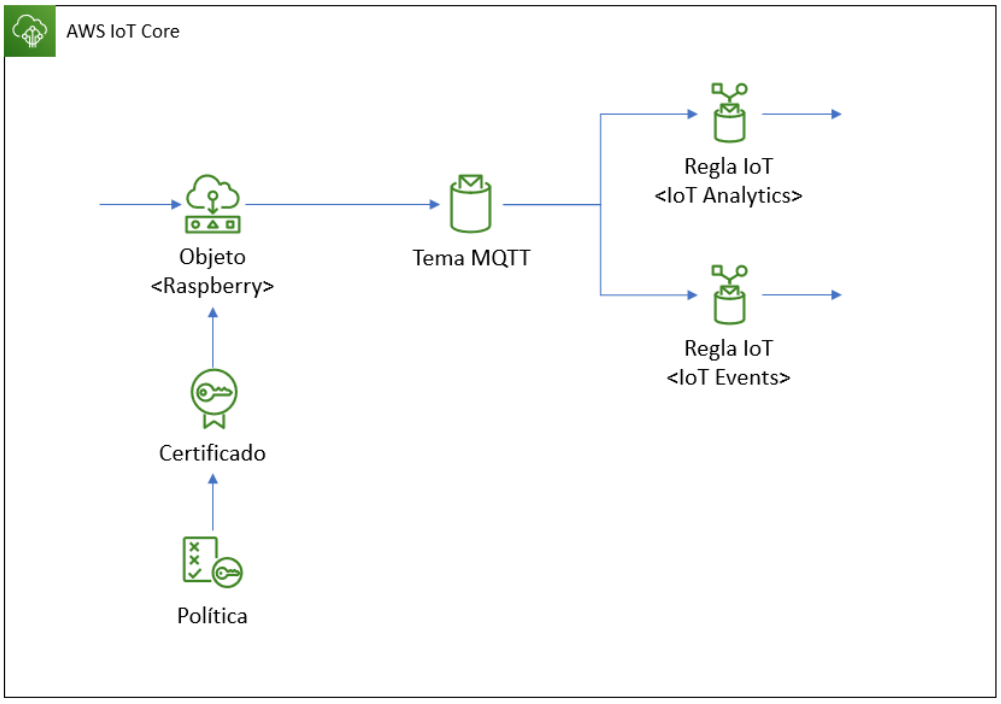

Figura 54. Modelo implementado en AWS IoT Core

7.6 Diseño estructura interna del servicio AWS Iot Analytics

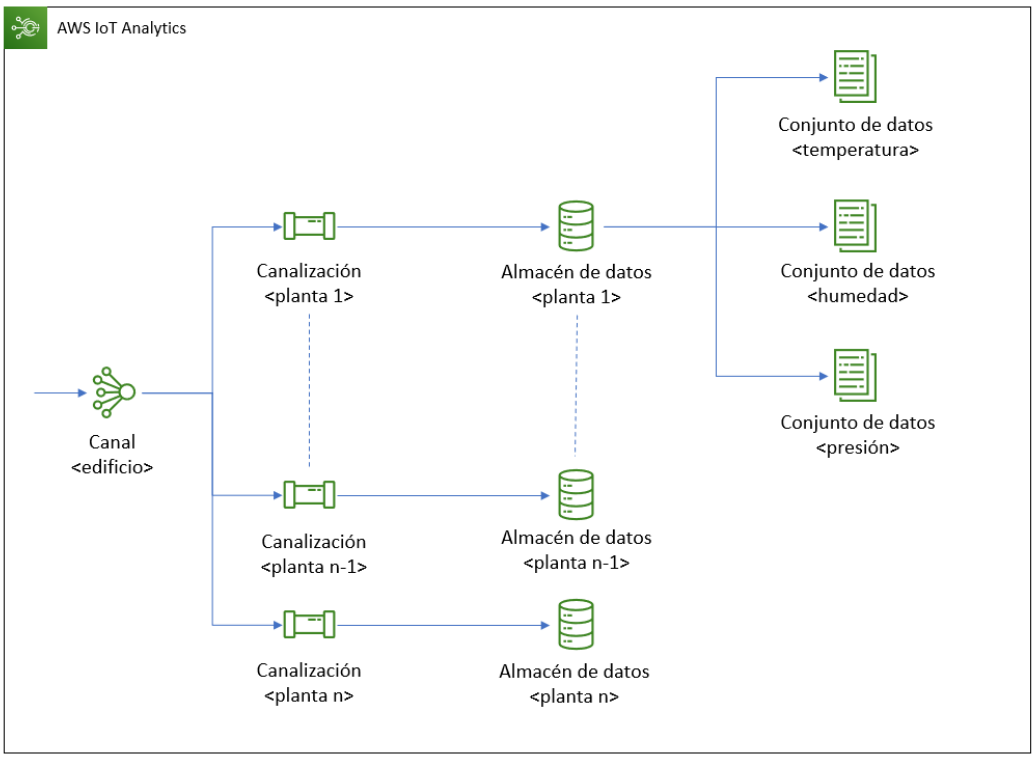

Figura 55. Modelo implementado en AWS IoT Analytics

7.7 Diseño estructura interna de los estados del servicio AWS Iot Events

Figura 56. Diagrama lógico estado 'normal'

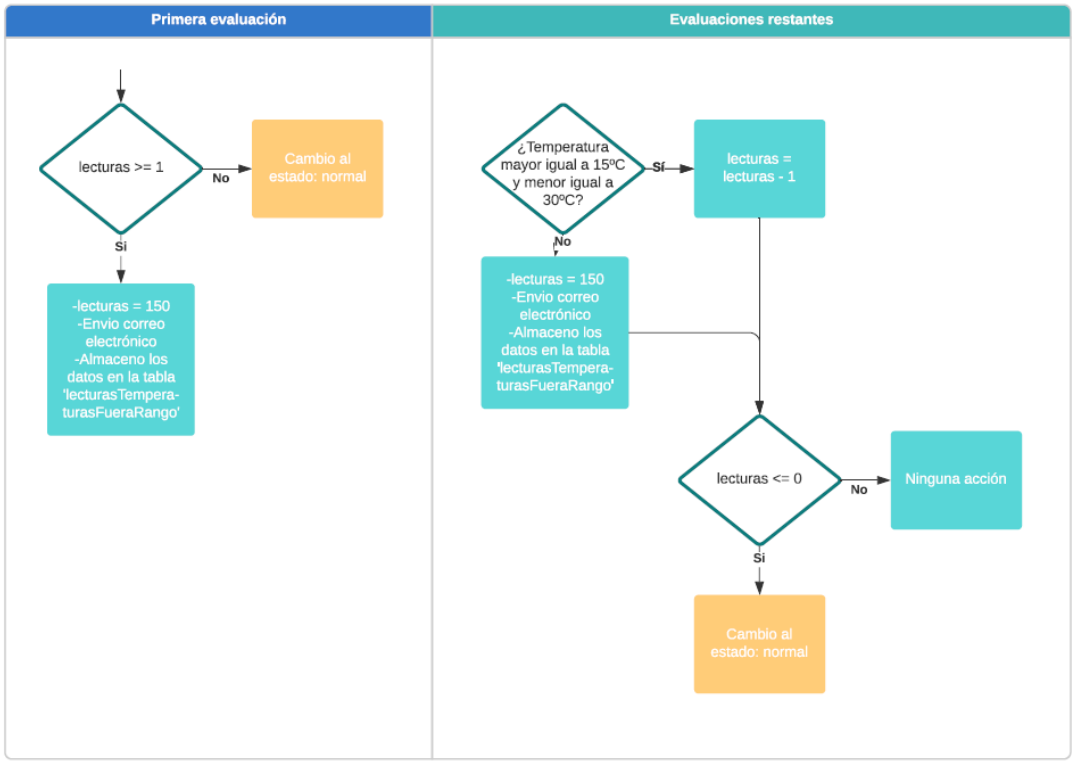

Figura 57. Diagrama lógico estado ‘fueraDeRango’