uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Telemática
2019-2020

Trabajo Fin de Grado

"PASARELA ZIGBEE PARA ENTORNO IOT"

Javier González Fuentes

Tutor

Prof.ª María Calderón Pastor

Leganés, noviembre de 2020

RESUMEN

Este proyecto propone el diseño de una arquitectura para crear una pasarela Zigbee para un entorno de lo que se conoce como **Internet de las Cosas**. El diseño propuesto está pensado para ser flexible y poder adaptar la infraestructura para cumplir requisitos determinados para una red domótica.

El proyecto se basa en el protocolo wireless Zigbee para crear una red de sensores y actuadores. Este protocolo se basa en el estándar IEEE 802.15.4 desarrollado para crear dispositivos que operan con batería y que requieren un consumo bajo de energía. Además de Zigbee, se utilizan diferentes tipos de tecnologías y herramientas que son muy utilizadas en la actualidad para el desarrollo de proyectos relacionados con dispositivos inteligentes.

El proyecto se divide en varios capítulos que guían al lector durante todo el proceso. Se expone la base teórica de las tecnologías utilizadas, se explica en detalle la arquitectura propuesta, se describe la configuración de las herramientas usadas, el desarrollo de la lógica de una red domótica y finalmente se analizan los resultados del tráfico dentro de toda la infraestructura desplegada.

Palabras clave: Zigbee, MQTT, domótica, wireless, Node-RED, red de sensores y actuadores.

ÍNDICE GENERAL

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS. 1
1.1 INTRODUCCIÓN. 1
1.2 OBJETIVOS Y PLANTEAMIENTO DEL PROBLEMA. 1
1.3 MARCO REGULADOR. 2
1.4 ESTRUCTURA DE LA MEMORIA. 2
CAPÍTULO 2. BACKGROUND Y ESTADO DEL ARTE. 4
2.1 SISTEMAS DOMÓTICOS. 4
2.2 MQTT. 5
2.3 PROTOCOLOS DE COMUNICACIÓN IOT. 5
2.4 ZIGBEE. 7
2.4.1 Zigbee Network. 7
2.4.2 Zigbee Stack. 9
2.4.3 Zigbee Profiles. 10
CAPÍTULO 3. ARQUITECTURA PROPUESTA. 12
3.1 DESCRIPCIÓN DE LA RED ZIGBEE. 14
3.2 ZIGBEE2MQTT. 15
3.3 MOSQUITTO. 16
3.4 AMAZON WEB SERVICES. 18
3.5 NODE-RED. 19
CAPÍTULO 4. CONFIGURACIÓN Y DESARROLLO. 21
4.1 HARDWARE UTILIZADO. 21
4.2 LA RED ZIGBEE. 21
4.3 ZIGBEE2MQTT. 22
4.4 MOSQUITTO. 24
4.5 AWS. 25
4.6 CONFIGURACIÓN DE NODE-RED. 29
4.7 FUNCIONALIDAD Y DESCRIPCIÓN DEL FLUJO DE NODE-RED. 32
4.7.1 Motion sensor. 32
4.7.2 Climate sensor. 35
4.7.3 Contact sensor. 35
4.7.4 Switch & Bulb. 38
CAPÍTULO 5. ANÁLISIS DE RESULTADOS Y PRUEBAS. 43
5.1 TRÁFICO ZIGBEE. 43
5.2 LOGS DE ZIGBEE2MQTT. 47
5.3 BROKER MQTT LOCAL Y EL BRIDGE A AWS. 51
5.4 DEBUG EN NODE-RED. 53
5.5 DASHBOARDS DE MONITORIZACIÓN Y CONTROL. 58
CAPÍTULO 6. GESTIÓN DEL PROYECTO. 61

6.1 PLANIFICACIÓN 61
6.2 PRESUPUESTO. 63
6.2.1 Gastos materiales y digitales. *63*
6.2.2 Gastos de personal. *65*
6.3 IMPACTO SOCIOECONÓMICO. 65
CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO. 67
CAPÍTULO 8. REFERENCIAS BIBLIOGRÁFICAS. 69
SUMMARY. 71
CONTEXT AND OBJECTIVES. 71
BACKGROUND AND STATE OF THE ART. 71
Home Automation systems. *71*
IoT communication protocols. *72*
Zigbee network. *73*
Zigbee Stack. *73*
Zigbee Profiles. *73*
PROPOSED ARCHITECTURE. 73
CONFIGURATION AND DEVELOPMENT. ANALYSIS OF RESULTS AND TESTS 74
PROJECT MANAGEMENT 74
Planning. *74*
Budgets. *75*
Socio-economic impact. *75*
CONCLUSIONS. 76

ÍNDICE DE FIGURAS

Figura 1. Ejemplo de una red Zigbee (topología de malla) [12]. 8
Figura 2. Esquema de la arquitectura de pila Zigbee [14]. 9
Figura 3. Dispositivos especificados en Zigbee HA [16]. 11
Figura 4. Esquema global de la arquitectura propuesta. 13
Figura 5. Parte local del esquema de la arquitectura propuesta. 14
Figura 6. Parte cloud del esquema de la arquitectura propuesta. 18
Figura 7. Flujo de Node-RED para la lógica domótica. 20
Figura 8. Parte del fichero de configuración de Zigbee2MQTT. 22
Figura 9. Parte del fichero de configuración de Zigbee2MQTT. 23
Figura 10. Parte del fichero de configuración de Zigbee2MQTT. 23
Figura 11. Parte del fichero de configuración para el bridge MQTT de local a cloud. 24
Figura 12. Parte del fichero de configuración para el bridge MQTT de local a cloud. Especificación de topics. 25
Figura 13. Certificado para la conexión con IoT Core 25
Figura 14. Política creada para permitir al bridge realizar acciones en IoT Core 26
Figura 15. Role de servicio para permitir EC2 conectar con IoT Core. 27
Figura 16. Instancia de EC2 con el role de servicio añadido. 27
Figura 17. Conexión de los usuarios a través del ELB. 28
Figura 18. ELB con la instancia en servicio. 28
Figura 19. Parte del fichero de configuración de Node-RED 29
Figura 20. Nodos mqtt in y mqtt out de Node-RED 30
Figura 21.Configuración del nodo mqtt-broker. 30
Figura 22. Configuración del nodo tls-config. 31
Figura 23. Nodo json de Node-RED 31
Figura 24. Nodo debug de Node-Red 31
Figura 25. Nodo function de Node-RED 32
Figura 26. Nodos text, gauge, switch y slider del paquete node-red-dashboard. 32
Figura 27. Flujo de Node-RED para el sensor de movimiento. 32
Figura 28. Nodo gauge del flujo del sensor de movimiento. 33
Figura 29. Nodo function del flujo del sensor de movimiento. 34
Figura 30. Nodo text del flujo del sensor de movimiento. 34
Figura 31. Flujo de Node-RED para el sensor de temperatura y humedad. 35
Figura 32. Flujo de Node-RED para el sensor de contacto. 35
Figura 33. Nodo funtion del flujo del sensor de contacto. Estado de la batería. 36
Figura 34. Nodo text del flujo del sensor de contacto. Estado de la batería. 36
Figura 35. Nodo funtion del flujo del sensor de contacto. Estado de la puerta. 37

Figura 36. Nodo text del flujo del sensor de contacto. Estado de la puerta. 37
Figura 37. Flujo de Node-RED para el Switch y la bombilla. 38
Figura 38. Nodo function del Switch. 39
Figura 39. Parte del flujo de la bombilla para representar el estado y el brillo. 40
Figura 40. Parte del flujo de la bombilla para crear los actuadores virtuales. 40
Figura 41. Nodo function para recuperar el estado de la bombilla. 41
Figura 42. Nodo function para establecer el brillo de la bombilla. 42
Figura 43. Captura de tráfico Zigbee del Switch al coordinador. 44
Figura 44. Captura de tráfico Zigbee del coordinador al Switch. 45
Figura 45. Captura de tráfico Zigbee del coordinador a la bombilla. 46
Figura 46. Captura de tráfico Zigbee de la bombilla al coordinador. 47
Figura 47. Logs de Zigbee2MQTT 48
Figura 48. Logs de Zigbee2MQTT 48
Figura 49. Logs de Zigbee2MQTT 49
Figura 50. Logs de Zigbee2MQTT 50
Figura 51. Logs de Zigbee2MQTT 50
Figura 52. Logs de Zigbee2MQTT 51
Figura 53. Logs de Zigbee2MQTT. 51
Figura 54. Logs del bridge entre Mosquitto y AWS IoT Core. 52
Figura 55. Logs del bridge entre Mosquitto y AWS IoT Core 52
Figura 56. Nodo debug (en verde) a la salida de mqtt-in del Switch. 53
Figura 57. Mensaje recibido en el nodo debug a la salida de mqtt-in del Switch. 53
Figura 58. Nodo debug (en verde) a la salida del nodo json del Switch. 54
Figura 59. Mensaje recibido en el nodo debug a la salida del nodo json del Switch. 54
Figura 60. Nodo debug (en verde) a la salida del nodo function del Switch. 54
Figura 61. Mensaje recibido en el nodo debug a la salida del nodo funtion del Switch. 55
Figura 62. Nodo debug (en verde) a la salida de los nodos function del flujo de la bombilla. 55
Figura 63. Mensaje recibido a la salida del nodo mqtt-in de la bombilla. 55
Figura 64. Mensaje a la salida del nodo json de la bombilla. 56
Figura 65. Mensaje a la salida del nodo function (get brightness). 56
Figura 66. Mensaje a la salida del nodo function (get state). 56
Figura 67. Switch virtual. 56
Figura 68. Mensaje a la salida del Switch virtual. 57
Figura 69. Dimmer virtual. 57
Figura 70. Mensaje a la salida del Dimmer virtual. 57
Figura 71. Mensaje a la salida del nodo function (set brightness). 57
Figura 72. Logs de Zigbee2MQTT. 58

Figura 73. Logs de Zigbee2MQTT. 58
Figura 74. Menú de navegación de para los dashboards de la red domótica. 58
Figura 75. Dashboard de monitorización y control del alumbrado. 59
Figura 76. Dashboard de monitorización del clima. 59
Figura 77. Dashboard de monitorización del sensor de movimiento. 60
Figura 78. Dashboard de monitorización de la puerta. 60
Figura 79. Diagrama de Gantt 62

ÍNDICE DE TABLAS

Tabla 1. Presupuestos materiales y digitales 64
Tabla 2. Presupuestos materiales con amortización. 64
Tabla 3. Tabla de gastos de personal. 65

LISTADO DE ABREVIATURAS

Internet de las Cosas ---------- (IoT)
Wireless sensor and actuator networks ---------- (WSAN)
Message Queuing Telemetry Transport ---------- (MQTT)
Bluetooth Low Energy ---------- (BLE)
Personal Area Networks Identifier ---------- (PAN ID)
Application Support Sub-Layer ---------- (APS)
Zigbee Device Objects ---------- (ZDO)
Zigbee Cluster Library ---------- (ZCL)
Home Automation ---------- (HA)
Amazon Web Services ---------- (AWS)
Elastic Cloud Computing ---------- (EC2)
Identity and Access Management ---------- (IAM)
Elastic Load Balancer ---------- (ELB)

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS.

1.1 Introducción.

El “Internet de las Cosas” (IoT) es un tema bastante popular hoy en día. El hecho de que disímiles objetos físicos se puedan conectar en una red, coleccionar y compartir datos a través de internet, abre un gran abanico de posibilidades al desarrollo en muchos campos como medicina, transporte, industria, agricultura, energía, medio ambiente, etc. Los dispositivos inteligentes son cada vez más variados y se desarrollan con relación a diferentes aplicaciones de uso: aplicaciones organizativas, aplicaciones industriales, aplicaciones de infraestructura, aplicaciones militares y aplicaciones de consumo [1].

Una de las aplicaciones de consumo más conocidas y utilizadas del IoT es la casa inteligente o Smart Home. Esta deriva de la introducción de los dispositivos inteligentes en el amplio concepto de home automation (HA) o domótica que se refiere a la construcción de sistemas y procesos de automatización para el hogar. Los sistemas de HA actuales que convierten un hogar en un “Smart Home”, se basan en una central o hub que es capaz de leer los datos de una red de sensores, controlar dispositivos y electrodomésticos inteligentes y funcionar como pasarela o gateway de la red domótica a Internet para que los usuarios tengan control remoto de los procesos a través de aplicaciones web y móviles.

El HA es una parte bastante importante dentro IoT y es un negocio que crece cada vez más. Son muchos fabricantes desarrollando disímiles tipos de sistemas de automatización utilizando diferentes tecnologías y software para crear productos domésticos inteligentes y conectarlos en una red.

1.2 Objetivos y planteamiento del problema.

El marco de desarrollo de este proyecto lo mantenemos en el contexto de una red de sensores y actuadores que utilizan Zigbee como protocolo de comunicación. Para esta red, necesitamos obtener y procesar los datos de lectura de dichos sensores y los comandos enviados a los dispositivos actuadores.

El objetivo de este proyecto es el diseño y desarrollo de una arquitectura con una gateway para una red Zigbee. La arquitectura permite que dicha red sea accesible a través de Internet y se desarrollan diferentes dashboards para la monitorización y el control de la red domótica. Apoyándonos en diferentes protocolos de comunicación, hardware y software, creamos un

entorno IoT donde somos capaces de definir y controlar la lógica del funcionamiento y la interacción de los diferentes dispositivos IoT.

1.3 Marco regulador.

Para definir el marco regulador de este proyecto vamos a comentar varios aspectos importantes, por una parte, referente a licencias de uso respecto a las tecnologías, herramientas y softwares utilizados y, por otra parte, referente a la protección de los datos de usuarios.

Este proyecto que iremos explicando con detalle a lo largo de la memoria está desarrollado utilizando herramientas y softwares de código libre como son Zigbee2MQTT, Mosquitto o Node-RED. Por otro lado, Zigbee es un estándar abierto gestionado por la Zigbee Alliance y funciona en la banda de frecuencia 2.4GHz que no tienen licencia, es decir, no requieren ningún permiso especial para su uso. Los dispositivos de sensores y actuadores que utilizamos son dispositivos Zigbee y los fabricantes se encargan de cumplir con el estándar.

Otro aspecto importante en el campo de Internet de las Cosas derivado del amplio espectro de dispositivos que existen y sus funcionalidades y uso, son las dudas que se plantean acerca de la protección de los usuarios y de sus datos, su identidad y su seguridad. La protección de las personas físicas en relación con el uso de sus datos viene definida en la **Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales** [2]. En el contexto en el que se desarrolla este proyecto y dada la naturaleza de los datos que se manejan en la red Zigbee, este aspecto no nos afecta. Estos datos son muy ligeros y no son de carácter personal. Por otro lado, para el acceso a los dashboards de monitorización y control de la red Zigbee se utiliza un sistema de autenticación. Esta parte del proyecto se despliega sobre Amazon Web Services (AWS) y está protegido por sus políticas de compliance, definiendo todos los datos como datos de cliente. Nuestro proyecto desarrolla una arquitectura Ad Hoc donde el usuario es el dueño de todos los datos y el único que puede acceder a ellos.

1.4 Estructura de la memoria.

La memoria de este trabajo se divide en 8 capítulos con el objetivo de organizar la estructura y ofrecer un hilo conductor coherente que guíe al lector de una manera clara, fácil y agradable a través de todo el proyecto que se explica.

Capítulo 1. Introducción y objetivos: En este capítulo se expone una visión general del tema que se va a tratar en el proyecto, los objetivos y el marco regulador.

Capítulo 2. Background y estado del arte: En este capítulo buscamos explicar de manera sencilla conceptos y protocolos de comunicación en los que se basa el desarrollo del proyecto. Se describe de manera simple el panorama actual relacionado con los sistemas domóticos, el protocolo MQTT y diferentes protocolos wireless de comunicación muy relacionados con el mundo de IoT, centrándonos más en Zigbee.

Capítulo 3. Arquitectura propuesta: Este capítulo se centra en explicar cada parte del diseño de la arquitectura que se propone para crear un entorno domótico basado en una red de sensores y actuadores que utilizan Zigbee como protocolo de comunicación.

Capítulo 4. Configuración y desarrollo: En esta sección se explica la configuración necesaria de los diferentes hardware y software que forman parte de la arquitectura propuesta, la configuración cloud y el desarrollo de la lógica del entorno domótico.

Capítulo 5. Análisis de resultados y pruebas: En este capítulo se describe cómo es el funcionamiento de la plataforma analizando el tráfico extremo a extremo dentro de la arquitectura propuesta.

Capítulo 6. Gestión del proyecto: En este capítulo se presenta la gestión del proyecto y para ello se divide en tres puntos fundamentales: planificación, presupuestos e impacto socioeconómico.

Capítulo 7. Conclusiones: Finalmente, en este capítulo se exponen las conclusiones a las que se ha llegado a lo largo del desarrollo de este proyecto y se plantean líneas de trabajo futuro.

Capítulo 8. Referencias bibliográficas.

CAPÍTULO 2. BACKGROUND Y ESTADO DEL ARTE.

2.1 Sistemas domóticos.

Los sistemas domóticos son sistemas desarrollados para el control de dispositivos que son de uso común en el hogar, ya sean electrodomésticos, el alumbrado, sistemas de seguridad, climatización o sistemas de entretenimiento. Existe muchísima variedad capaz de adaptarse a cualquier necesidad. La automatización del hogar permite aprovechar la funcionalidad de alta tecnología y el lujo, algo que no era tan viable en el pasado. A medida que el desarrollo de la tecnología continúa expandiéndose, también lo harán las posibilidades de que los sistemas domóticos hagan la vida más fácil y agradable.

En los últimos años se ha trabajado y avanzado mucho en el aumento de la flexibilidad y compatibilidad de los dispositivos, en la centralización del control de todos los dispositivos en un sitio, en la seguridad, en el control remoto de funciones del hogar, etc.

A muy alto nivel, típicamente la arquitectura de un sistema domótico consiste en una red de sensores y actuadores (Wireless sensor and actuator networks, WSAN [3]) que se comunican con un hub a través de un protocolo de comunicación, por lo general, inalámbrico. Esta central es la encargada de recibir, procesar y enviar datos entre los diferentes dispositivos de la red domótica y, funcionando como gateway, se conecta a las aplicaciones de visualización, monitorización y control para los usuarios.

Los hubs vienen en todos los tipos y formas, pero típicamente tratan de cumplir el propósito, automatizar todos los sensores y dispositivos inteligentes del hogar. Ejemplos de los hubs más utilizados por usuarios que buscan los beneficios de un Smart Home son Hubitat, Samsung SmartThings, HomeKit, OpenHab, y Wink. Estos hubs compiten en puntos como amigabilidad con el usuario (user friendly) respecto al dispositivo, experiencia de usuario en la aplicación, acceso remoto y sobre todo la compatibilidad con varios protocolos de comunicación que amplía el espectro de utilización de diferentes dispositivos.

Una opción para usuarios más avanzados es Home Assistant, un software open source que corre en un ordenador. Generalmente los usuarios lo instalan en una Raspberry Pi que puedes mantener encendida todo el día en casa. Es rápido debido a que corre todo dentro de la red local, pero está limitado a los protocolos de comunicación de la Raspberry Pi y no tienes un acceso remoto.

Estos problemas los vamos a solventar en este proyecto. Nuestra red domótica tendrá acceso local y remoto, crearemos nuestro hub utilizando también una Raspberry Pi añadiendo modificaciones que permitan la comunicación con Zigbee y tendremos la libertad de desarrollar completamente la lógica de la comunicación y haciéndola tan compleja como se requiera.

2.2 MQTT.

MQTT (Message Queuing Telemetry Transport) es una parte importante en este proyecto para permitir la comunicación remota con nuestra red domótica. MQTT es un protocolo de publicación-suscripción extremadamente ligero que transporta mensajes entre dispositivos y por lo general corre sobre una pila de protocolos TCP/IP. Está diseñado para conexiones con localizaciones remotas y es ampliamente utilizado en IoT en una gran variedad de industrias sobre todo en la automovilística. [4]

MQTT tiene dos entidades, un broker de mensajes y uno o varios clientes. El broker es un servidor que establece la comunicación, a nivel de aplicación, entre los diferentes clientes. Recibe todos los mensajes y los enruta al cliente apropiado suscrito a un topic específico. [5]

MQTT es ligero y eficiente, permite comunicaciones bidireccionales, permite la entrega confiable de mensajes definiendo 3 niveles de QoS y facilita el cifrado de mensajes mediante TLS y la autenticación de clientes mediante protocolos de autenticación modernos como OAuth. En próximos capítulos veremos cómo integramos y utilizamos MQTT en este proyecto [6].

2.3 Protocolos de comunicación IoT.

Según lo que se ha investigado, existen varios protocolos de comunicación wireless relacionados con la implementación de un sistema domótico. Decidir sobre qué protocolo es mejor para los dispositivos IoT que se van a utilizar en el desarrollo de una red IoT es una discusión que depende completamente de los objetivos que se quieran alcanzar. Estos son algunos de los protocolos más utilizados y, teniendo en cuenta que son protocolos muy bien documentados, solo comento algunas de las características a tener en cuenta a la hora de utilizar Zigbee sobre otros protocolos wireless.

Bluetooth Low Energy (BLE) [7] es una versión independiente del Bluetooth BR/EDR (clásico) con la que se consigue un consumo de energía significativamente más reducido y a la vez mantiene un rango similar de comunicación. BLE se utiliza en situaciones en las no

necesita manejar grandes flujos de datos y donde las baterías deben durar mucho tiempo. BLE también es compatible con muchos sistemas operativos, incluidos Android, iOS, Windows y Mac OS lo que lo hace más fácil de utilizar.

WiFi es otro protocolo ampliamente utilizado para la comunicación entre dispositivos IoT y aunque es un protocolo muy adecuado para la comunicación, consume mucha energía para sus operaciones. Es el protocolo más poderoso para transferencias de datos entre la mayoría de los dispositivos IoT en la actualidad, pero su consumo no lo hace adecuado para dispositivos alimentados por baterías.

Z-Wave [8] es un protocolo de comunicación de radiofrecuencia de baja potencia y está diseñado principalmente para sistemas de automatización del hogar y dispositivos electrónicos como controladores de lámparas y sensores. Su alcance es de unos 24 metros en interior y unos 90 metros al aire libre. Las redes Z-Wave están limitadas a 232 dispositivos y la frecuencia de trabajo es de 900MHz, con lo cual es despreciable la interferencia con protocolos como WiFi, Bluetooth y Zigbee.

6LoWPAN [9] define mecanismos de encapsulación y compresión de cabeceras que permiten enviar y recibir paquetes IPv6 sobre redes basadas en el estándar IEEE 802.15.4. Este protocolo permite que los dispositivos se puedan interconectar mediante redes IP y fue diseñado específicamente para manejar situaciones en las que es necesario conectar muchos dispositivos.

Zigbee [10] es un protocolo de comunicación inalámbrica de corto alcance basado en el estándar IEEE 802.15.4 [11] desarrollado específicamente para dispositivos que operan con batería. Los principales atributos que hacen que Zigbee sea adecuado para una comunicación eficaz entre dispositivos de IoT son el bajo consumo de energía, la alta escalabilidad, la seguridad y la durabilidad. Zigbee no consume mucho ancho de banda y, al igual que Z-Wave, opera utilizando una red de malla (mesh network) que les permite aumentar su rango de la señal transmitiendo la información de un dispositivo a otro hasta llegar al hub. La red de Zigbee también soporta las topologías de estrella y árbol.

Zigbee tiene la capacidad de soportar teóricamente hasta 65535 nodos, distribuidos en subredes de 255 nodos. La tecnología es ideal para aplicaciones de baja potencia: sensores, electrodomésticos inteligentes, etc. No consume mucho ancho de banda y está diseñado idealmente para su uso en domótica y grandes sitios industriales donde se requiere poca energía e intercambio de datos.

¿Por qué Zigbee es una buena opción?

En la investigación realizada y según los objetivos del proyecto, resalta Zigbee como candidato principal a ser el protocolo de comunicación de nuestro ecosistema IoT. Zigbee, BLE y WiFi comparten frecuencia de transmisión y, a diferencia de BLE y WiFi, Zigbee no tiene soporte de sistemas operativos, lo cual no es un inconveniente para nuestro proyecto. En comparación con Z-Wave, Zigbee teóricamente no tiene límites de saltos entre un dispositivo y el controlador de la red, mientras que Z-Wave admite hasta cuatro. A diferencia de WiFi, Zigbee y BLE son bastante similares desde el punto de vista de ancho de banda y consumo de energía, aunque Zigbee es más eficiente en redes de mayor rango y la eficiencia energética era un punto de peso importante.

Otro de los puntos que se tuvo en cuenta para la decisión fue la posibilidad de acceder a dispositivos más variados y económicos respecto a los otros protocolos. Hasta hace unos años Zigbee era una tecnología utilizada en dispositivos que no eran tan económicos. Esto es algo que cambió en 2018 cuando IKEA comenzó a fabricar y vender dispositivos Zigbee más asequibles consiguiendo en 2019 entrar a formar parte de la junta directiva de la Zigbee Alliance. Hoy en día, Xiaomi, Lumi, Tuya, etc, con la fabricación de productos Zigbee, han extendido ese rango y variedad de dispositivos económicos. Zigbee es una de las tecnologías más utilizadas en los ecosistemas de Smart Home.

2.4 Zigbee.

2.4.1 Zigbee Network.

En Zigbee, hay tres tipos diferentes de dispositivos: dispositivo final (End Device), enrutador (Router) y coordinador (Coordinator). Una red Zigbee siempre tiene solo un coordinador y puede tener varios routers y dispositivos finales.

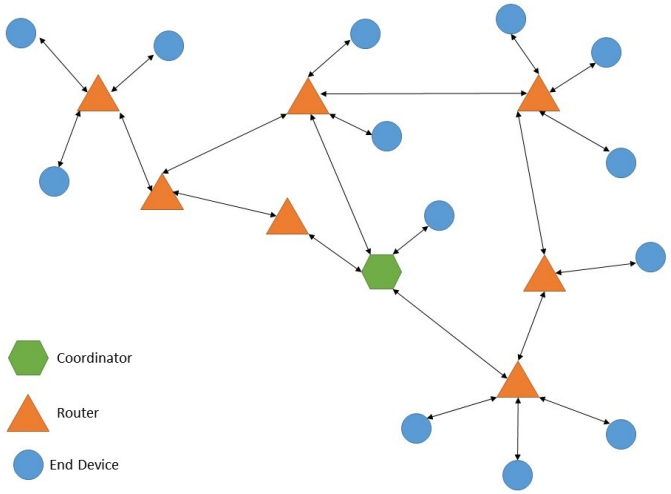

Figura 1. Ejemplo de una red Zigbee (topología de malla) [12].

Los dispositivos finales no enrutan el tráfico y tienen la funcionalidad de poder dormir, lo que hace que los dispositivos finales sean una opción adecuada para los dispositivos que funcionan con batería. Un dispositivo final solo tiene un padre, ya sea el coordinador o un enrutador que generalmente es el dispositivo más cercano cuando se emparejó. Todas las comunicaciones hacia y desde el dispositivo final se realizan a través de su padre. Si un enrutador principal se desconecta, todo el tráfico hacia sus hijos cesará hasta que esos dispositivos finales intenten encontrar un nuevo padre.

Los routers son responsables de enrutar el tráfico entre diferentes nodos y no deberían dormir. Actúan como puentes que comunican y retransmiten datos entre router y dispositivos finales o coordinadores. Son responsables de recibir y almacenar mensajes destinados a sus hijos y de permitir que nuevos nodos se unan a la red. Como tal, los routers no son una opción adecuada para dispositivos que funcionan con baterías.

Un coordinador es un router especial que además es responsable de formar la red. Para hacer eso, debe seleccionar el canal apropiado, modo de seguridad de la red, el personal area networks identifier (PAN ID) y dirección de red extendida (extended PAN ID).

El PAN ID es un número de 16 bits único para identificar cada red y que es común entre todos los dispositivos de la red. Los dispositivos Zigbee pueden estar preconfigurados con un PAN iD para unirse o pueden descubrir redes cercanas y seleccionar un PAN ID para unirse. EL PAN ID utiliza como un campo de direccionamiento de capa MAC en todas las transmisiones entre dispositivos en una red Zigbee. Sin embargo, debido al espacio de direccionamiento limitado del PAN ID de 16 bits (65.535 posibilidades), existe la posibilidad de que varias redes Zigbee puedan utilizar el mismo identificador, así que para evitar esto, la Zigbee Alliance creó

un PAN ID de 64 bits en la versión de Zigbee PRO (2007). Este parámetro de la red se conoce como extended PAN ID y todos los dispositivos de la misma red Zigbee deben compartir los mismos valores de PAN ID de 64 y 16 bits [13].

2.4.2 Zigbee Stack.

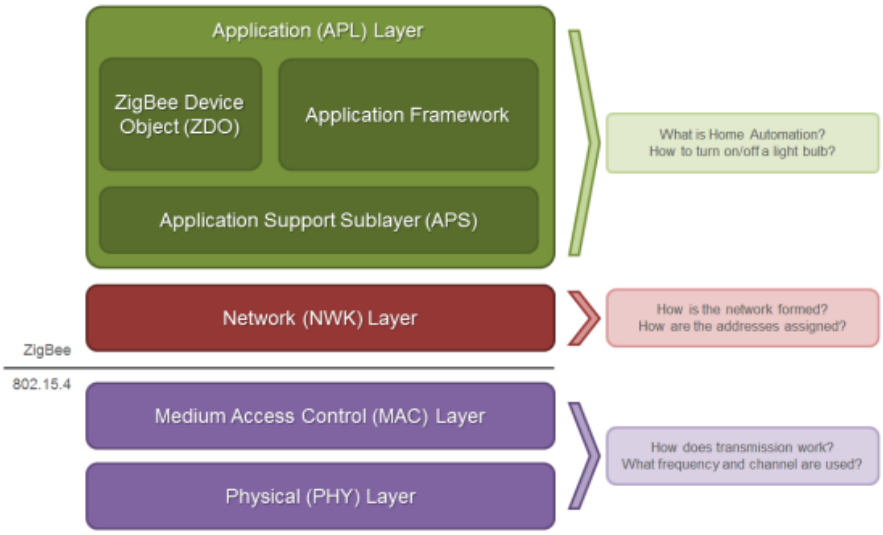

Figura 2. Esquema de la arquitectura de pila Zigbee [14].

Al igual que la mayoría de los protocolos de red, Zigbee también utiliza el concepto de capas para separar diferentes componentes y funciones en módulos independientes que se pueden ensamblar de diferentes maneras. Zigbee se basa en la capa física (PHY) y la subcapa de control de acceso al medio (MAC) definida en el estándar IEEE 802.15.4 las cuales manejan operaciones de red de bajo nivel como direccionamiento y transmisión/recepción de mensajes.

La especificación Zigbee define la capa de red (NWK) que se encarga de la estructura, el enrutamiento y la seguridad de la red. La capa de aplicación (APL) de Zigbee consta de la subcapa Application Support Sub-Layer (APS), el Zigbee Device Objects (ZDO) y las aplicaciones definidas por el usuario o fabricante que le dan al dispositivo su funcionalidad específica.

APS proporciona una interfaz entre la capa de red (NWK) y la capa de aplicación (APL) a través de un conjunto general de servicios que son utilizados tanto por ZDO como por los objetos de aplicación definidos por el fabricante.

Application Framework es el entorno en el que se alojan los objetos de la aplicación en los dispositivos Zigbee.

ZDO se encuentra entre el Application Framework y la APS y representan una clase base de funcionalidad que proporciona una interfaz entre los objetos de la aplicación, el device profile y la APS. Proporciona funciones de detección de dispositivos y servicios y capacidades de gestión de red avanzadas. Satisface los requisitos comunes de todas las aplicaciones que operan en una pila de protocolo Zigbee. [15]

2.4.3 Zigbee Profiles.

Zigbee tiene tres especificaciones Zigbee RF4CE, Zigbee PRO y la más reciente Zigbee IP y sobre estas la Zigbee Alliance crea diferentes soluciones. Este proyecto se centra en la especificación Zigbee PRO finalizada en 2007 y actualizada por última vez en 2015. Zigbee PRO tiene varios perfiles en dependencia de su aplicación de uso (Application Profile). Un perfil es como un espacio de dominio de aplicaciones y dispositivos que están relacionados. Los ID de perfil de aplicación son números de 16 bits y van de 0x0000 a 0x7fff para perfiles públicos y de 0xbf00 a 0xffff para perfiles específicos del fabricante. Algunos de los perfiles públicos definidos por la Zigbee Alliance son:

- (0x0101) Industrial Plant Monitoring (IPM),
- **(0x0104) Home Automation (HA)**
- (0x0105) Commercial Building Automation (CBA)
- (0x0107) Telecom Applications (TA)
- (0x0108) Personal Home & Hospital Care (PHHC)
- (0x0109) Advanced Metering Initiative (AMI)

Home Automation, es el perfil con el que se trabaja en este proyecto, es un perfil de aplicación público que define una amplia gama de dispositivos destinados a su uso en el hogar. Los dispositivos especificados en el perfil de HA se dividen en 5 grupos que se muestran en la Figura 3.

Device Name	Device ID
Generic	
On/off switch	0x0000
Level control switch	0x0001
On/off output	0x0002
Level control output	0x0003
Scene selector	0x0004
Configuration tool	0x0005
Remote control	0x0006
Lighting	
On/off light	0x0100
Dimmable light	0x0101
On/off light switch	0x0103
Dimmer switch	0x0105
Light sensor	0x0106
Occupancy sensor	0x0107
Closure	
Shade	0x0200
Shade controller	0x0201
HVAC	
Heating and cooling unit	0x0300
Thermostat	0x0301
Temperature sensor	0x0302
Pump	0x0303
Pump controller	0x0304
Pressure sensor	0x0305
Flow sensor	0x0306
Intruder Alarm System	
IAS control / indicating equipment	0x0400
IAS ancillary control equipment	0x0401
IAS warning device	0x0403

Figura 3. Dispositivos especificados en Zigbee HA [16].

Otro aspecto importante que destacar es la Zigbee Cluster Library (ZCL), introducida en la versión Zigbee 2006 anterior a Zigbee PRO. La ZCL está publicada bajo su propia especificación y no es más que un conjunto de funciones y comandos utilizados en los perfiles públicos desarrollados por la Zigbee Alliance para favorecer el desarrollo y estandarización.

CAPÍTULO 3. ARQUITECTURA PROPUESTA.

Para conseguir los objetivos de este trabajo se utilizaron varias tecnologías y herramientas tanto hardware como software. La idea sobre la que se trabajó fue desarrollar la lógica de una red domótica simulando funciones básicas de una Smart Home y conseguir un acceso remoto y seguro a dichas funciones para mostrar el planteamiento de este proyecto. La arquitectura que se expone en la Figura 4 es la solución que se propone para conseguir los objetivos del proyecto y a lo largo de este capítulo se describirá cada parte en detalle.

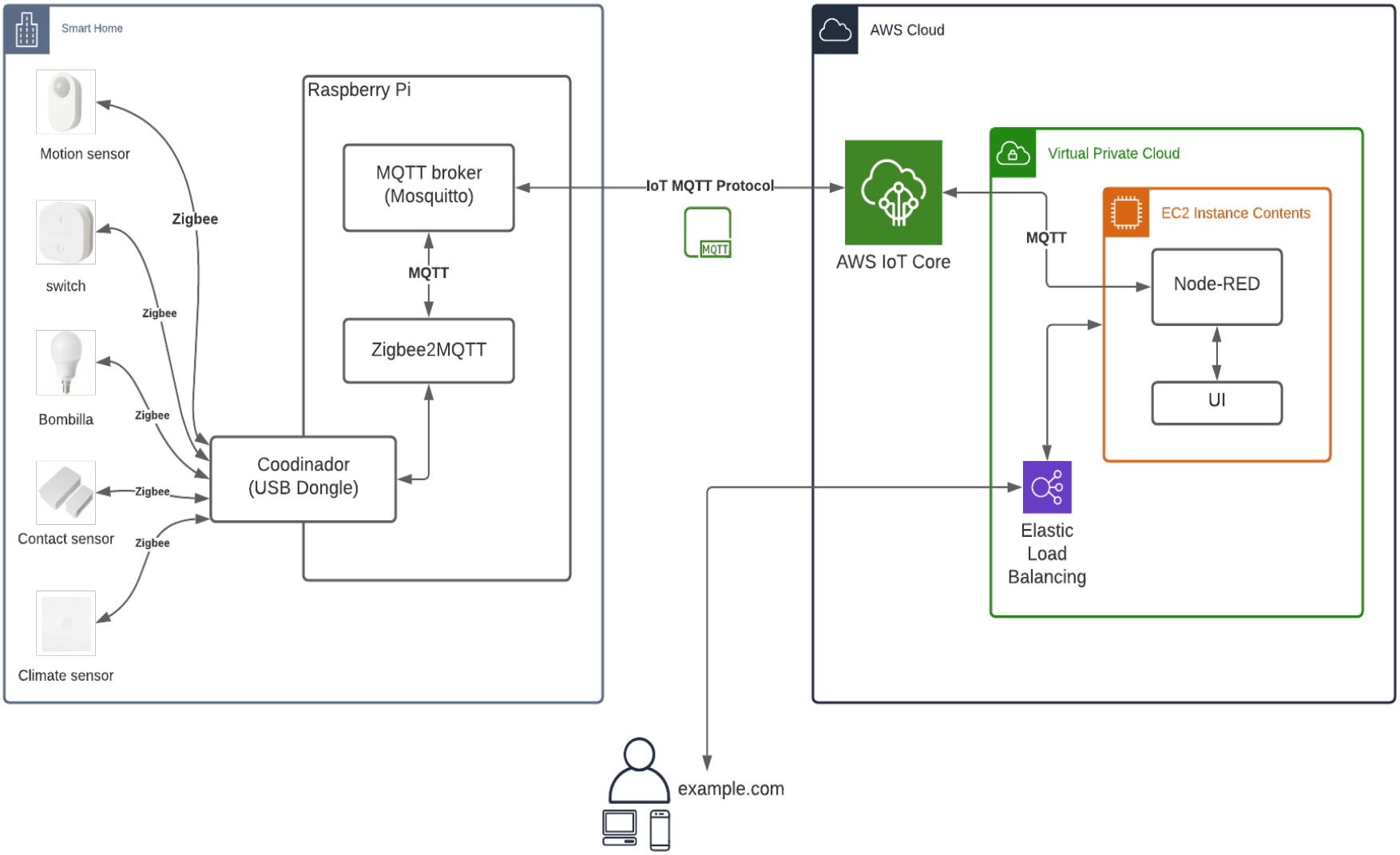

Figura 4. Esquema global de la arquitectura propuesta.

3.1 Descripción de la red Zigbee.

Parte de la arquitectura propuesta consiste en el despliegue de una red de sensores y actuadores inalámbricos (WSAN) los cuales funcionan con batería y utilizan Zigbee como protocolo de comunicación. Esta red está formada por un sensor de humedad, un sensor de contacto, un sensor de movimiento, una bombilla y un switch. Todos estos son dispositivos finales y no fue necesario introducir ningún dispositivo router en la red Zigbee.

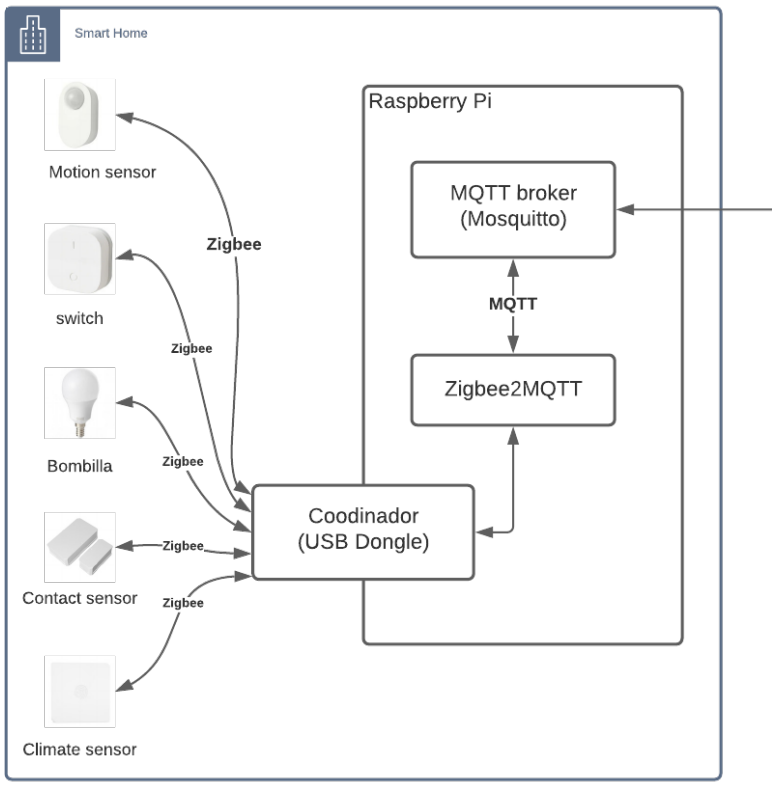

Figura 5. Parte local del esquema de la arquitectura propuesta.

Otro punto clave era tener un coordinador y el gateway para esta red Zigbee. Al no querer utilizar ningunos de los Hubs de fabricantes que hemos mencionado anteriormente, la decisión fue utilizar una Raspberry Pi. La idea era tener más flexibilidad a la hora de crear los flujos de interacción de los dispositivos IoT, algo que veremos más adelante. Una Raspberry Pi es básicamente un ordenador de bajo coste y tamaño reducido. Su alta capacidad de proceso en relación con su tamaño y su escaso coste, su capacidad para soportar diferentes módulos y periféricos y su flexibilidad para utilizar diferentes sistemas operativos, hacen que sea una de las placas más utilizadas en el desarrollo de proyectos de redes e informática en la actualidad. Para este proyecto se utilizó Raspberry Pi OS (anteriormente llamado Raspbian) como sistema operativo para la placa.

El contratiempo en este caso fue que una Raspberry Pi no tiene un módulo Zigbee integrado, así que se tuvo que investigar cómo se podía hacer para conseguir que utilizase Zigbee y así formar parte de la red domótica. Entre las varias opciones que se encontraron, la que más encajaba en el proyecto fue el USB Dongle CC2531. Es un USB stick integrado con el chip CC2531 desarrollado por Texas Instruments para comunicaciones Zigbee. A pesar de no ser tan potente para redes grandes de más de 30 dispositivos, fue suficiente para el rango de alcance y dispositivos que se necesitaba desplegar en la red Zigbee propuesta. Además, es bastante económico para las propiedades que necesitamos.

Este USB stick se flasheó con el firmware de coordinador del estándar ZigBee HA (Home Automation) y se utilizó como coordinador de la red Zigbee propuesta. Utilizando un segundo USB dongle CC2531 en otro ordenador y utilizando otras herramientas, se pudo capturar tráfico Zigbee y ver el funcionamiento de la red, pero hablaremos con más detalles sobre esto en otro capítulo.

3.2 Zigbee2MQTT.

En este punto, ya teniendo la red zigbee desplegada, surge la problemática de cómo poder obtener los datos de la red Zigbee para poder trabajar con ellos a través de una aplicación. También surgieron cuestiones sobre cómo vincular todo a algún servicio cloud para conseguir un acceso remoto. Después de unos días investigando soluciones para esto, se tomó la decisión de integrar Zigbee2MQTT [17] en la arquitectura.

Zigbee2MQTT es un proyecto open source desarrollado Koen Kanters y más de 268 contribuidores en Github [18]. Está disponible para varias plataformas y sistemas operativos y cuenta con el respaldo de una amplia comunidad. Es bastante utilizado en ecosistemas de home automation para facilitar la integración y control de dispositivos Zigbee a través de MQTT y sin necesidad de utilizar el hub del proveedor de dichos dispositivos. Zigbee2MQTT actúa como un puente entre los mensajes Zigbee y los mensajes MQTT. Soporta una lista inmensa de dispositivos y si alguno no está en la lista, es relativamente simple añadir la compatibilidad.

Zigbee2MQTT tiene su propio firmware desarrollado a partir del stack Zigbee HA 1.2. El firmware lleva una aplicación definida como parte de la capa de aplicación del stack de Zigbee y se encarga de enviar y recibir mensajes del controlador Zigbee2MQTT. Dicho controlador lo instalamos en la Raspberry y es el bridge entre Zigbee y MQTT, genera los mensajes MQTT a partir de los mensajes que recibe de Zigbee y viceversa.

El firmware de coordinador para el CC2531 fue el que utilizamos en el USB dongle y es compatible con 20 dispositivos conectados directamente al coordinador, lo cual es suficiente para nuestro proyecto, además, en el caso de que se configurasen routers en la red Zigbee, el tamaño de la red se podría ampliar.

3.3 Mosquitto.

Mosquitto [19] es un broker de mensajes open source que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Zigbee2MQTT necesita conectarse como cliente a un broker MQTT para enviar y recibir los mensajes. El broker se encarga de manejar todos los mensajes MQTT de los clientes que estén conectados a él.

Una vez que tenemos Zigbee2MQTT integrado en la infraestructura y corriendo como demonio sobre el sistema operativo de la Raspberry Pi, ya tenemos mensajes MQTT circulando en nuestra red local. En este punto ya nos plateamos como conseguir acceso remoto a nuestra red Zigbee.

Evidentemente, la aplicación y la lógica para el tratamiento de los dispositivos a través de mensajes MQTT, era necesario que estuviesen desplegadas en un servicio cloud. De ahí, cómo llevar los mensajes MQTT de la red local hasta donde se desplegará la aplicación fue el próximo reto.

Leyendo la documentación de Mosquitto vimos que tiene una funcionalidad de bridge. Esto significa que un broker MQTT como Mosquitto se puede conectar de manera segura como cliente a otro broker MQTT en remoto. Los detalles de la configuración se verán en capítulos posteriores.

Es válido pensar en la posibilidad de conectar directamente Zigbee2MQTT con un broker MQTT en un servicio cloud en lugar de conectarlo con Mosquitto en una red local, lo cuál es viable. Sin embargo, pensando en la flexibilidad de la arquitectura, se decidió utilizar la funcionalidad de bridge de Mosquitto. Pensemos en los siguientes casos que se describen y que reafirman la decisión que se tomó.

Si, por ejemplo, se quiere utilizar una parte de la red Zigbee y desarrollar funcionalidades que no requieren un acceso remoto sino sólo desde la red local, no es viable conectar simultáneamente Zigbee2MQTT a dos brokers MQTT, entonces no podríamos controlar parte de la red domótica en remoto y otra parte desde la red local. Sin embargo, si utilizamos un

broker MQTT local, podemos decidir que mensajes se envían a la lógica en cloud y cuáles se utilizan en una lógica en local.

Otro ejemplo, imaginemos que un cliente tiene desplegada una red domótica local con otros dispositivos y funcionalidades. Dicha red también utiliza un broker MQTT para las comunicaciones y solamente requiere tener acceso remoto a su red. Pues es viable con solo utilizar la parte del entorno cloud de la arquitectura propuesta vinculando el broker MQTT legacy con AWS IoT Core. De esta forma los datos de la red legacy estarían en la nube para poder desarrollar una lógica que dé una solución el cliente.

3.4 Amazon Web Services.

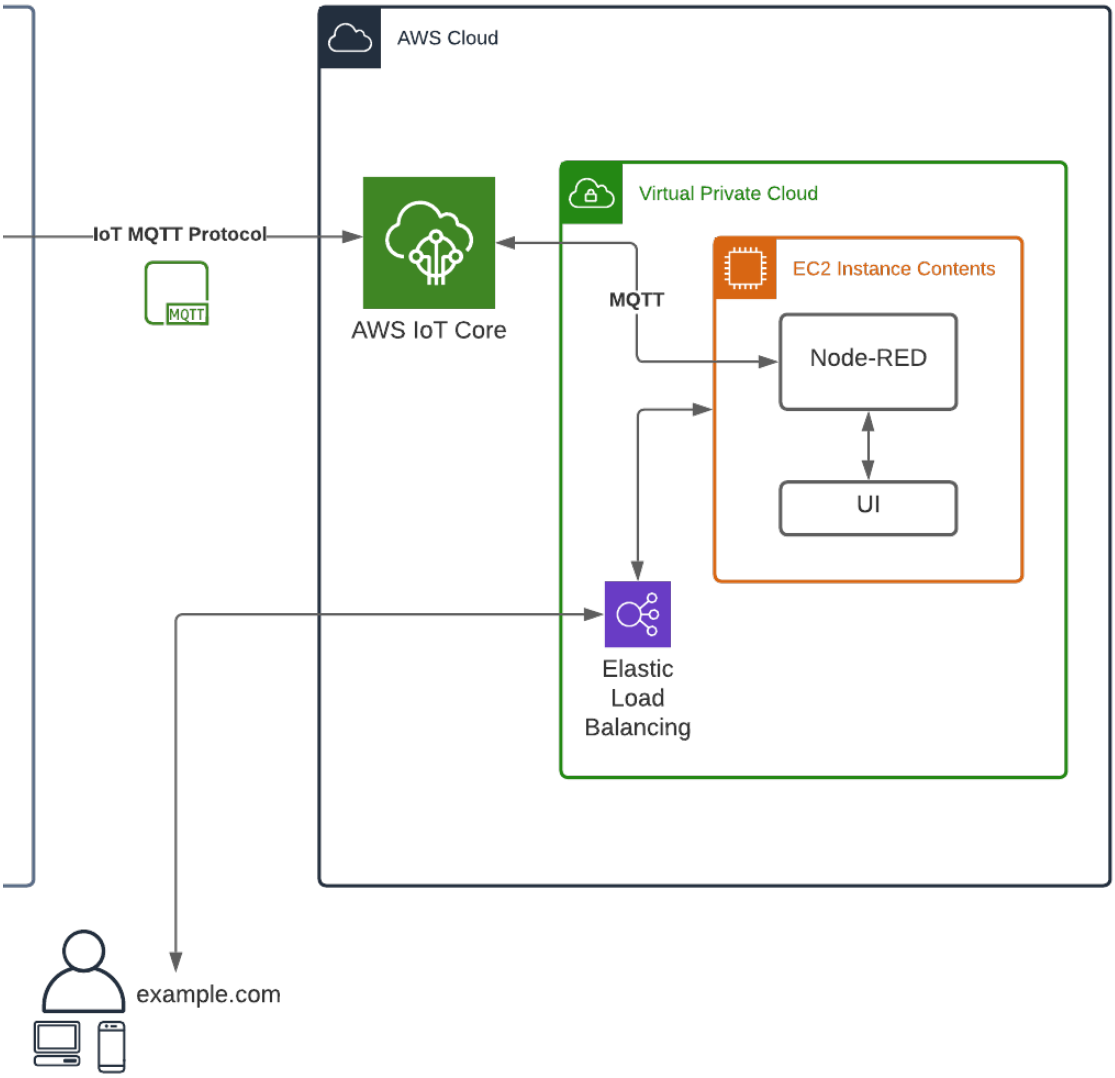

Figura 6. Parte cloud del esquema de la arquitectura propuesta.

Investigando varios servicios de cloud para encontrar qué opción se adecuaba más a este proyecto, se tomó la decisión de utilizar AWS [20] por 3 motivos principales: es líder de mercado, tienen un servicio específico para IoT y disponen, durante el primer año, de una capa de servicios gratuita con recursos limitados. Para el nivel de tráfico y datos que se manejan en esta arquitectura, los recursos ofrecidos de manera gratuita fueron suficientes.

Se utilizaron 3 servicios claves de AWS: Internet of Things Core (IoT Core), Identity and Access Management (IAM) y Elastic Cloud Computing (EC2).

IoT Core, fue el nexo remoto con el broker de Mosquitto de nuestra red local. Este servicio nos ofrece un endpoint personalizado para establecer comunicaciones MQTT con AWS IoT para posteriormente poder utilizar los datos en otros servicios. IoT Core nos permite generar certificados para establecer una conexión segura sobre TLS desde nuestro broker Mosquitto y crear políticas para permitir el acceso de otros servicios de la cuenta de AWS.

AWS Identity and Access Management (IAM) permite administrar el acceso a los servicios y recursos de AWS de manera segura. Este servicio nos permitió crear un role de servicio para poder acceder posteriormente desde EC2 a AWS IoT.

EC2 es unos de los servicios más utilizados y conocidos de AWS. Este potente servicio tiene amplias funcionalidades y es quien proporciona capacidad de cómputo escalable en la nube. En este proyecto se utilizó para poder tener un servidor dedicado en la nube donde estaría funcionando la aplicación de interacción con la red de domótica.

3.5 Node-RED.

Node-RED [21] fue la solución utilizada para gestionar los mensajes MQTT, poder crear los flujos de comportamiento de todos los dispositivos de la red domótica y crear los dashboards de monitorización e interacción para los usuarios de la Smart Home. Es una de las herramientas más utilizadas en el desarrollo de entornos IoT y se basa en el desarrollo de flujos para conectar dispositivos de hardware, API y servicios online. La figura 7 muestra la visión global de la lógica desarrollada en Node-RED para este proyecto y que se explicará en detalle en el siguiente capítulo.

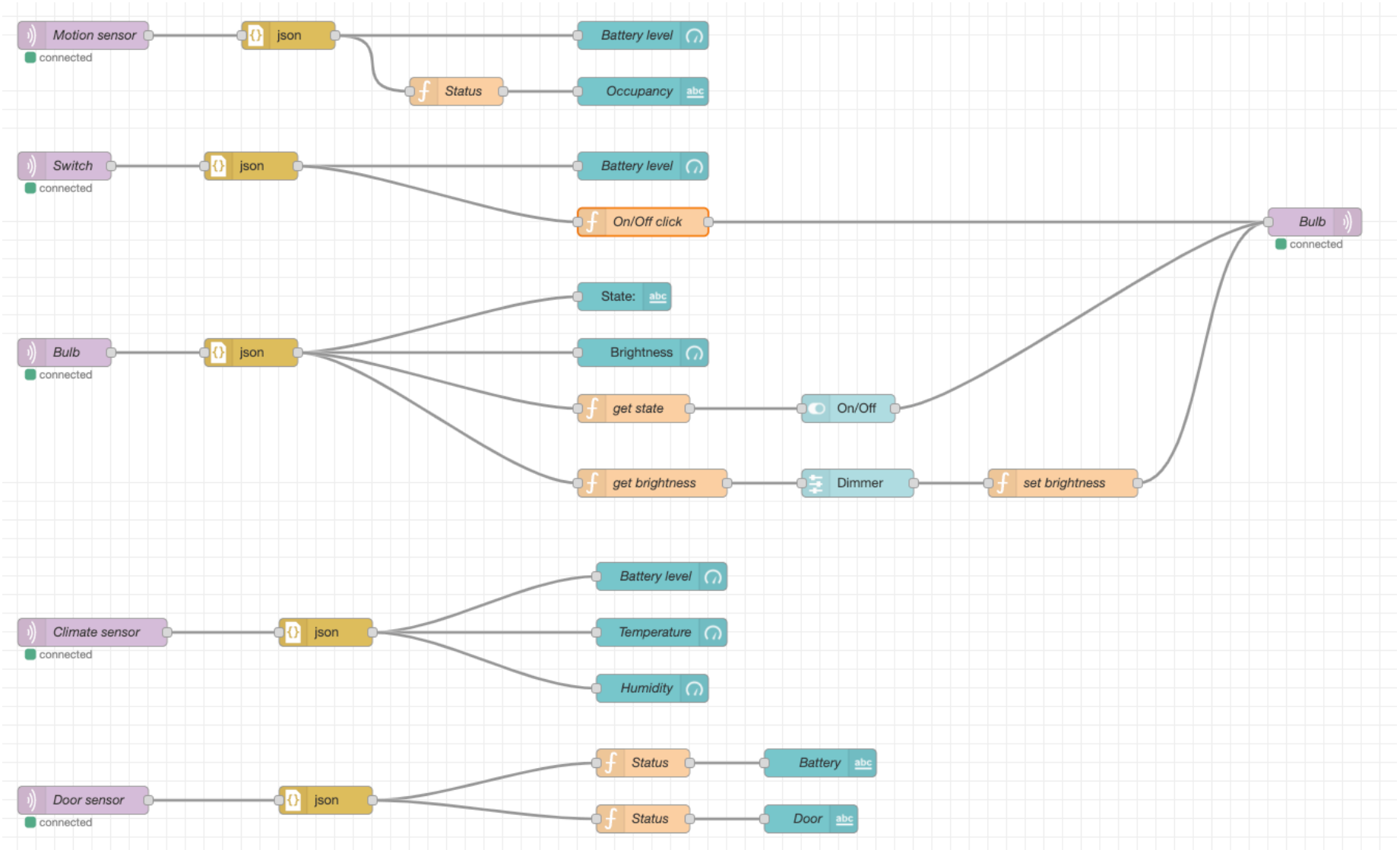

Figura 7. Flujo de Node-RED para la lógica domótica.

CAPÍTULO 4. CONFIGURACIÓN Y DESARROLLO.

4.1 Hardware utilizado.

En este apartado se listan y describen brevemente los dispositivos hardware adquiridos para el desarrollo del proyecto.

- Raspberry Pi 4 Model B [22].
- USB dongle CC2531.
  Este dispositivo consiste en la adaptación de un conector USB con el chip Zigbee CC2531 de Texas Instruments.
- CC Debugger [23].
  Este dispositivo se utiliza principalmente para conectar los dispositivos con un software de depuración y programación para la instalación de firmwares.
- CC2531 downloader cable.
  Este dispositivo es el conector entre el CC Debugger y el USB dongle CC2531.
- LED1738G7 (IKEA TRADFRI LED bulb E12 600 lumen).
  Este dispositivo Zigbee es una bombilla LED con espectro blanco regulable entre cálido y frío.
- E1525 / E1745 (IKEA TRADFRI motion sensor).
  Este dispositivo Zigbee es un sensor de movimiento con un alcance máximo de 5 metros y un ángulo de 120° y también puede configurarse para modo diurno o nocturno.
- E1743 (IKEA TRADFRI ON/OFF switch).
  Este es un dispositivo Zigbee con funcionalidad de switch y dimmer.
- SNZB-02.
  Este dispositivo Zigbee es un sensor para medir los valores de temperatura y humedad del ambiente.
- SNZB-04.
  Este dispositivo Zigbee es un sensor de contacto.

4.2 La red Zigbee.

Para poder crear la red Zigbee lo principal fue instalar el firmware personalizado de Zigbee2MQTT en el USB Dongle CC2531 y de esta forma tener preparado el coordinador. Para esto se utilizó el CC Debugger conectado a un ordenador con sistema operativo Windows donde utilizamos el programa gratuito SmartRF Flash programmer de Texas Instruments para flashear el chip CC2531 con el firmware descargado previamente. La

conexión entre el CC Debugger y el USB Dongle fue a través del Downloader cable CC2531. Existen varios tutoriales paso a paso en internet y la documentación de Zigbee2MQTT sobre cómo utilizar estos dispositivos.

La Raspberry Pi es nuestro dispositivo central donde estará la gateway de la red Zigbee con lo cual es donde conectamos el nuevo coordinador, el USB Dongle CC2531.

4.3 Zigbee2MQTT.

En la documentación de este software viene bien detallado los pasos para instalarlo [24]. La mejor opción para nuestro caso y la variante que se utilizó fue correrlo como un demonio con systemctl en el sistema operativo de la Raspberry Pi. También existen opciones para utilizarlo en un entorno virtual de Python o en un contenedor de Docker.

Después de leer en la documentación las opciones de configuración [25], dedicamos un tiempo a configurar nuestro bridge Zigbee2MQTT. El fichero de configuración se escribe en formato YAML y está ubicado en el directorio denominado “data”, dentro del directorio donde se instaló. A continuación, presentamos y explicamos, de forma progresiva, cómo quedó el fichero de configuración Zigbee2MQTT.

```
homeassistant: false
permit_join: false
mqtt:
  base_topic: zigbee2mqtt
  server: 'mqtt://192.168.1.39:1883'
```

Figura 8. Parte del fichero de configuración de Zigbee2MQTT.

Siguiendo las indicaciones de los fabricantes de los dispositivos Zigbee sobre cómo activar el modo pairing, conseguimos parear los dispositivos con el coordinador de la red Zigbee previamente conectado a la Raspberry Pi. Para esto, la opción de configuración permit_join debe estar a true y al acabar de añadir los dispositivos que se quieran, se vuelve a poner a false para mantener la red segura y evitar que se unan otros dispositivos Zigbee. La opción de configuración homeassistant es obligatoria y se debe poner a “true” o “false” dependiendo de si se utiliza o no el plugging de integración de Zigbee2MQTT con el software Home Assistant [26] (ver figura 8).

La opción de configuración bajo mqtt (ver figura 8) hace referencia al topic principal de los mensajes MQTT que se crean y a la ubicación para conectar con un broker MQTT, que como hemos mencionado para nuestro caso se ha utilizado Mosquitto. Hay opciones para hacer

que la conexión con el broker pueda ser sobre TLS y con autenticación, pero en este punto no lo consideramos necesario pues seguimos trabajando dentro de nuestra red privada. Más adelante al conectar con la nube si que será necesario trabajar sobre la seguridad de la conexión.

```
serial:
  port: /dev/ttyACM0
advanced:
  log_level: info
```

Figura 9. Parte del fichero de configuración de Zigbee2MQTT.

Esta parte de la configuración (ver figura 9) hace referencia al puerto USB de la Raspberry Pi donde se encuentra conectado el dispositivo USB dongle CC2531, que es el coordinador de la red Zigbee. También se especifica el nivel de detalle de los logs de Zigbee2MQTT.

```
devices:
  '0x680ae2fffe80371c':
    friendly_name: switch
    retain: false
  '0x680ae2fffe91d661':
    friendly_name: bulb
    retain: false
  '0x00124b001f90f708':
    friendly_name: contact_sensor
    retain: false
  '0xccccccfffedcf96d':
    friendly_name: motion_sensor
    retain: false
  '0x00124b001f84cd82':
    friendly_name: t&h_sensor
    retain: false
```

Figura 10. Parte del fichero de configuración de Zigbee2MQTT.

Bajo la opción de configuración devices (ver figura 10) se encuentran todos los dispositivos que se han unido a la red. Aquí fue muy importante desactivar la opción de configuración MQTT retain en cada dispositivo. Esto lo que provoca es que los mensajes MQTT que se generan para cada uno de los dispositivos vengan con el flag retain desactivado. El flag retain implica que el broker retenga el último mensaje enviado a un topic y lo envíe a todos los futuros suscriptores de dicho topic. AWS IoT no soporta esta especificación de MQTT 3.1.1 con lo cual rechazaba la conexión desde nuestro broker local porque los mensajes por defecto se generaban con el flag retain activo [27].

4.4 Mosquitto.

Instalar el broker Mosquitto es muy simple en un sistema linux ya que está disponible a través del repositorio principal. En la documentación de Eclipse Mosquitto se indica como instalarlo para varios sistemas operativos. La configuración fundamental fue la que se hizo para poder establecer el bridge con AWS IoT. Para esto se creó un fichero de configuración independiente dentro del directorio de configuración (ver figura 11 y 12).

```
# ================================================================
# Bridge to AWS IOT
# ================================================================

# Bridge connection name and MQTT client Id,
# enabling the connection automatically when the broker starts.

connection awsiot
address [redacted]iot.eu-west-1.amazonaws.com:8883

cleansession true
try_private true
clientid bridgeawsiot
start_type automatic
notifications false
log_type all

# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false

# ================================================================
# Certificate based SSL/TLS support
# ================================================================

#Path to the rootCA
bridge_cafile /etc/mosquitto/certs/bridges/rootCA.pem

# Path to the PEM encoded client certificate
bridge_certfile /etc/mosquitto/certs/bridges/cert.crt

# Path to the PEM encoded client private key
bridge_keyfile /etc/mosquitto/certs/bridges/private.key
```

Figura 11. Parte del fichero de configuración para el bridge MQTT de local a cloud.

Las descripciones de las opciones de configuración establecidas son fáciles de encontrar en la documentación [28]. Lo importante en este punto fue poder establecer una conexión segura con AWS IoT utilizando el endpoint y los certificados generados en el propio servicio IoT Core.

```
# Specifying which topics are bridged and in what fashion
topic zigbee2mqtt/switch both 1
topic zigbee2mqtt/bulb both 1
topic zigbee2mqtt/bulb/set both 1
topic zigbee2mqtt/motion_sensor both 1
topic zigbee2mqtt/t&h_sensor both 1
topic zigbee2mqtt/contact_sensor both 1

#END of bridge.conf
```

Figura 12. Parte del fichero de configuración para el bridge MQTT de local a cloud. Especificación de topics.

Por otro lado, fue necesario definir los patrones de los topics que se querían compartir entre los dos brokers, el sentido de la conexión en el que se comparten y el nivel de QoS. En este punto nos apoyamos en MQTT Explorer [29] para ver la estructura completa de los topics que se deseaban enviar por el bridge.

4.5 AWS.

En IoT Core es necesario realizar dos acciones fundamentales. Primero, generar los certificados para el cifrado de la conexión entre el broker local y el broker en la nube. La figura 13 muestra el certificado creado para conectar Mosquitto con IoT Core.

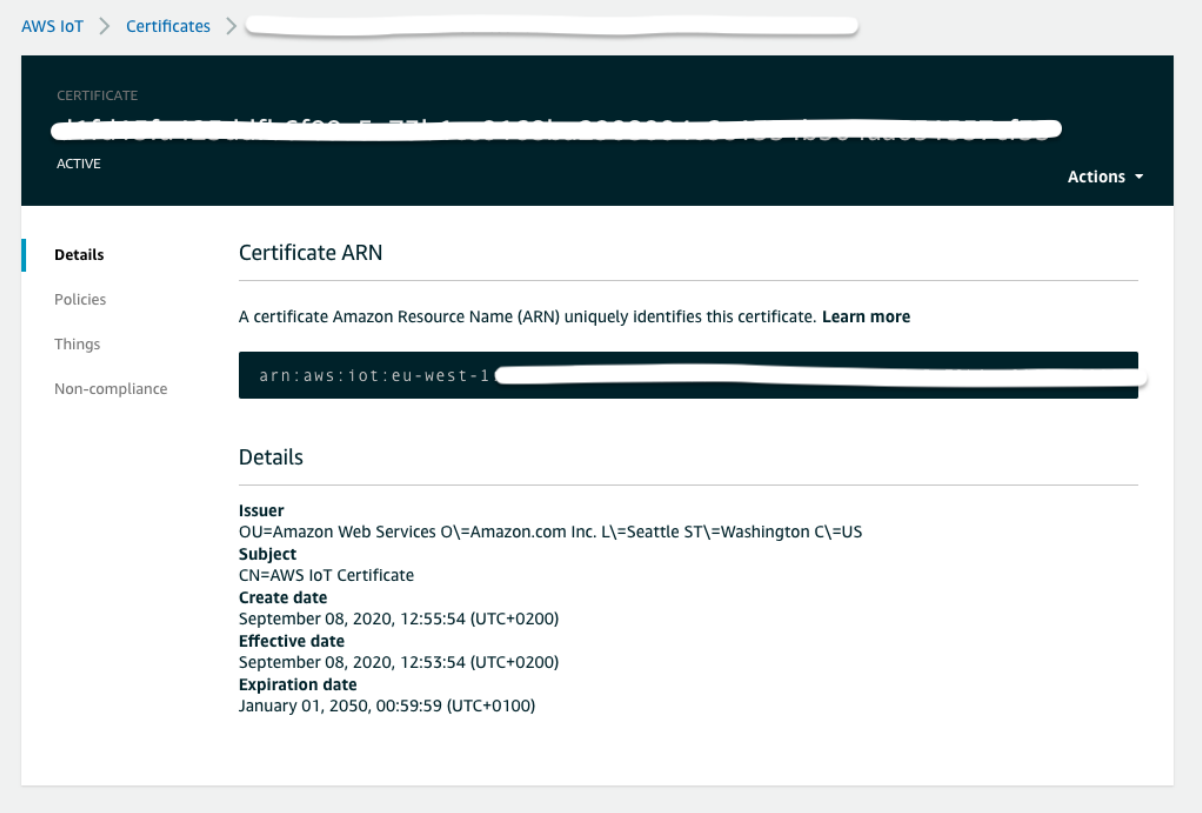

Figura 13. Certificado para la conexión con IoT Core

Segundo, es necesario crear una política para permitir al bridge realizar cualquier acción sobre los recursos de AWS (ver figura 14). Es posible restringir la política para realizar solo determinadas acciones sobre algún recurso específico, pero para mantener las cosas simples permitimos cualquier acción de Mosquitto sobre cualquier recurso de AWS.

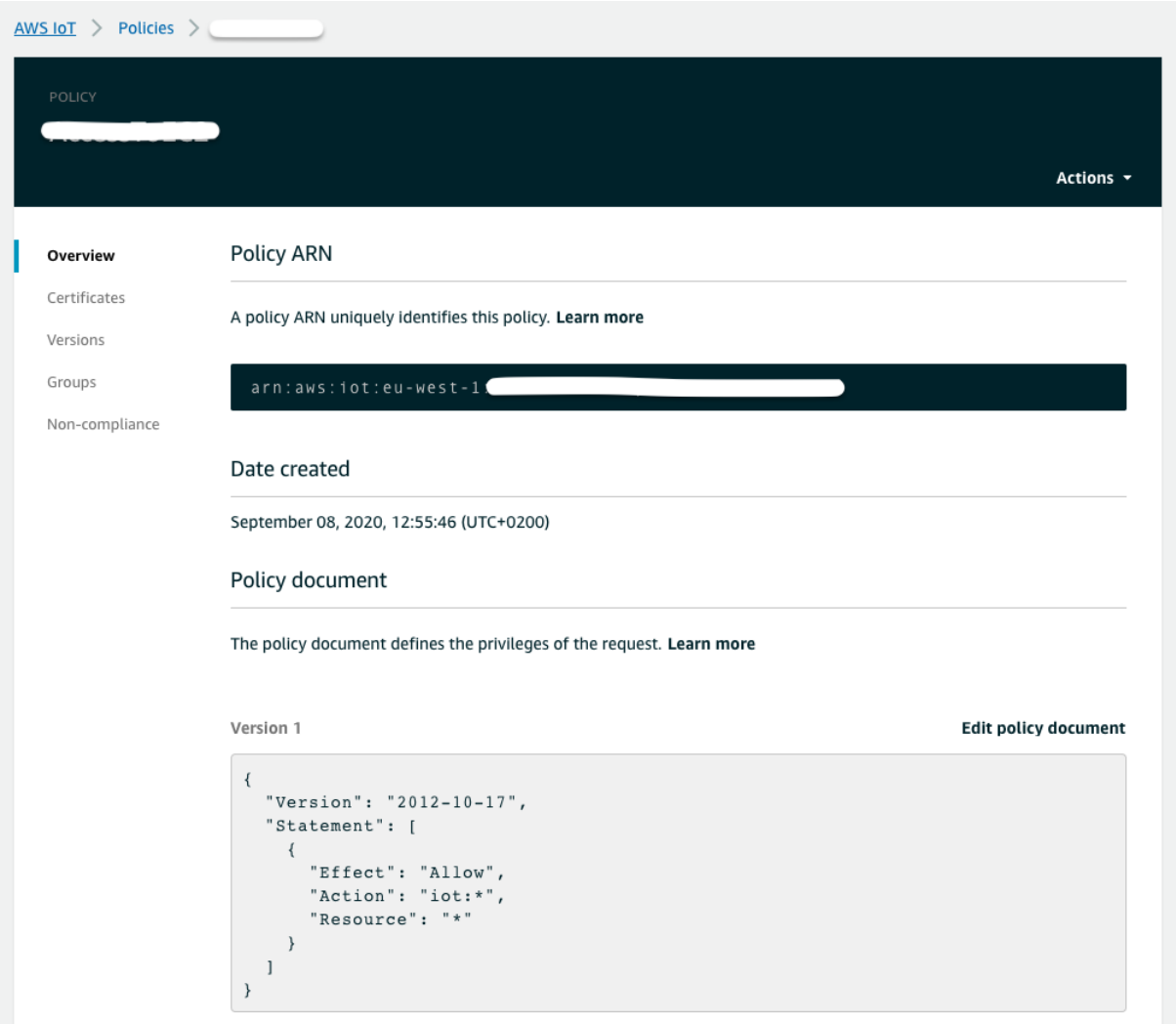

Figura 14. Política creada para permitir al bridge realizar acciones en IoT Core

Con AWS EC2 creamos una instancia donde estaría alojada la aplicación de Node-RED. La instancia creada fue una t2.micro que cuenta con los recursos suficientes para poder utilizar Node-RED. Ubuntu fue la opción de sistema operativo por la que nos decantamos para dicha instancia. Lo más importante a destacar es la necesidad de que los mensajes MQTT llegasen a Node-RED, con lo cual tuvimos que crear y adjuntar un role de servicio a la instancia para que EC2 pudiese comunicar con IoT Core. Para esto utilizamos el servicio de IAM. La figura 15 muestra el rol de servicio creado con la política AWSIoTConfigAccess, que es una política administrada por AWS. En la Figura 16 se muestra el rol añadido a la instancia EC2 y a partir de este punto ya hay comunicación entre ambos servicios de AWS.

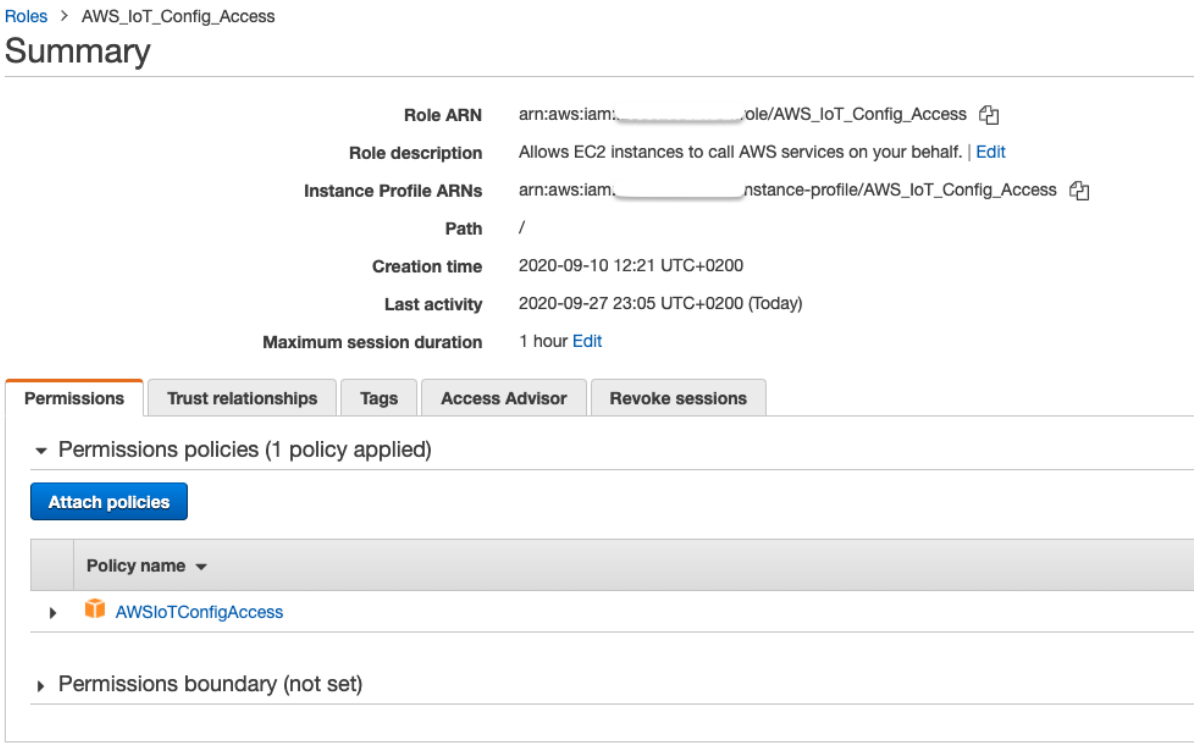

Figura 15. Role de servicio para permitir EC2 conectar con IoT Core.

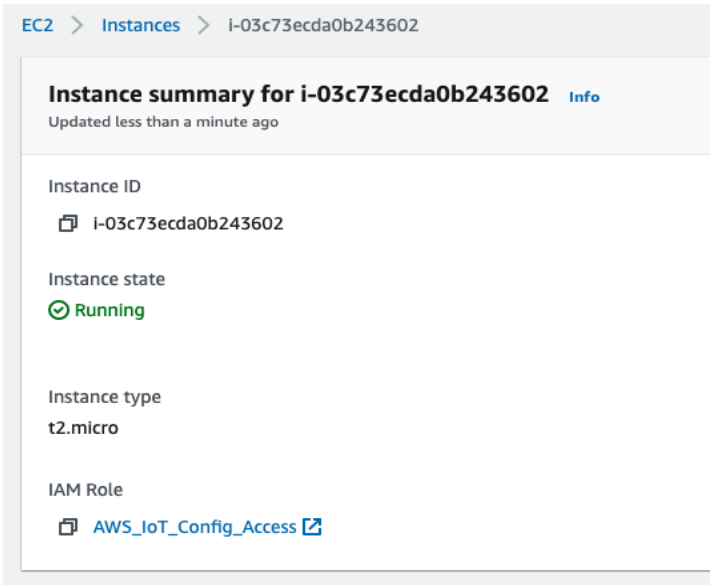

Figura 16. Instancia de EC2 con el role de servicio añadido.

Dentro del servicio de EC2 se utiliza un recurso más, un Elastic Load Balancer (ELB), el cual consiste en un balanceador de carga. La función por la que se utiliza es para apuntar, al ELB, nuestro dominio de acceso al dashboard de control de la Smart Home, tal como se muestra

en la figura 17. De esta manera evitamos exponer la IP pública de nuestro servidor y el tráfico que llegue al ELB es redirigido a la instancia.

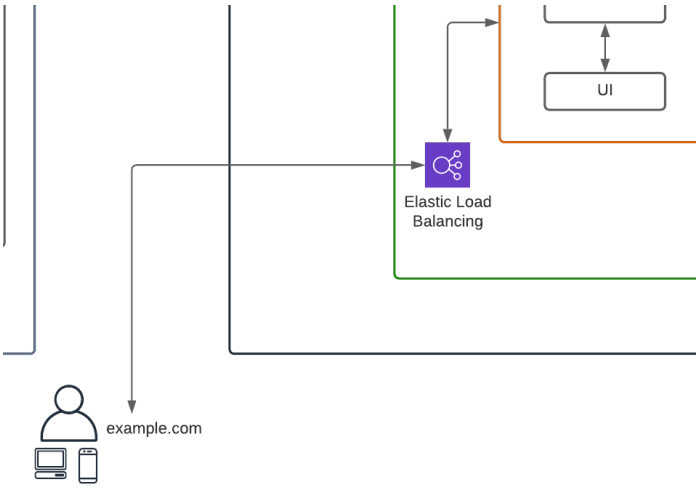

Figura 17. Conexión de los usuarios a través del ELB.

La figura 18 muestra la instancia de EC2 habilitada en el ELB, lo que indica que el tráfico recibido en el ELB es reenviado a dicha instancia en servicio.

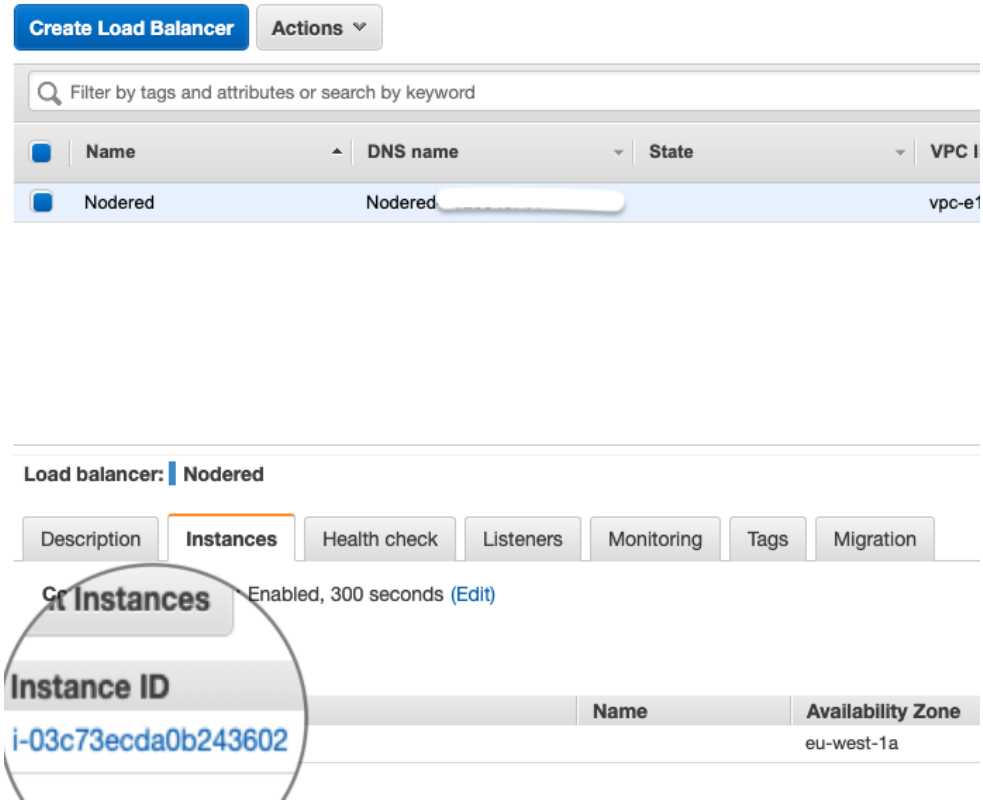

Figura 18. ELB con la instancia en servicio.

Si bien en nuestra arquitectura solamente tenemos una instancia, con lo cual no aprovechamos la ventaja real del ELB que es repartir la carga de tráfico, es válido considerar que, si la infraestructura crece y necesitamos más recursos, habría que buscar una solución diferente.

4.6 Configuración de Node-RED.

La instalación de esta herramienta en AWS puede ser complicada o bastante sencilla dependiendo de los requisitos que se tengan. En nuestro caso, que era utilizarla dentro de una instancia Ubuntu, fue simple. En la documentación de Node-RED se explica muy bien todos estos pasos y las diferentes opciones que se pueden tener [30].

En cuanto a la configuración lo fundamental fue poder securizar Node-RED (ver figura 19). Esto se trabajó en dos puntos, encriptar la comunicación habilitando HTTPS y habilitar la autenticación basado en username/password del editor y del Dashboard de Node-RED. Las contraseñas tanto para el usuario administrador del editor como para el usuario del acceso a los dashboards de control son hash encriptados con la herramienta de líneas de comandos node-red-admin. Para la generación de los certificados utilizamos acme.sh [31], un shell script que actúa como cliente para emitir y renovar automáticamente los certificados gratuitos de Let's Encrypt.

```
// property can be used. See http://nodered.org/docs/security.html for details.
adminAuth: {
    type: "credentials",
    users: [{
        username: "[redacted]",
        password: "[redacted]",
        permissions: "*"
    }]
},

httpNodeAuth: {user:"[redacted]",pass:"[redacted]"},
httpStaticAuth: {user:"[redacted]",pass:"[redacted]"},

https: {
    key: require('fs').readFileSync('/home/ubuntu/.acme.sh/kriogman.com/kriogman.com.key'),
    cert: require('fs').readFileSync('/home/ubuntu/.acme.sh/kriogman.com/fullchain.cer')
},

// be redirected to HTTPS.
requireHttps: true,
```

Figura 19. Parte del fichero de configuración de Node-RED

Node-RED ofrece muchísimas opciones y ventajas para nuestro desarrollo y podemos conseguir la flexibilidad para crear un sistema domótico tan complejo como deseemos. Una vez configurado todo, comenzamos a desarrollar los flujos de interacción de los dispositivos Zigbee y los dashboards de monitorización y control de la red domótica. Para el desarrollo del flow “My Home Station” trabajamos con diferentes tipos de nodos:

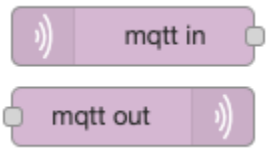

Figura 20. Nodos mqtt in y mqtt out de Node-RED

Los nodos mqtt in y mqtt out (ver figura 20) son los necesarios para capturar y enviar, respectivamente, mensajes MQTT a los diferentes topics dependiendo del dispositivo con el que vamos a trabajar.

Los nodos mqtt in y mqtt out dependen de un nodo mqtt-broker que es el que configuramos para establecer la conexión con nuestro broker de AWS IoT Core a través de un cliente mqtt. Este nodo a su vez depende de un nodo tls-config que proporciona la configuración para que el nodo mqtt-broker pueda encriptar su conexión.

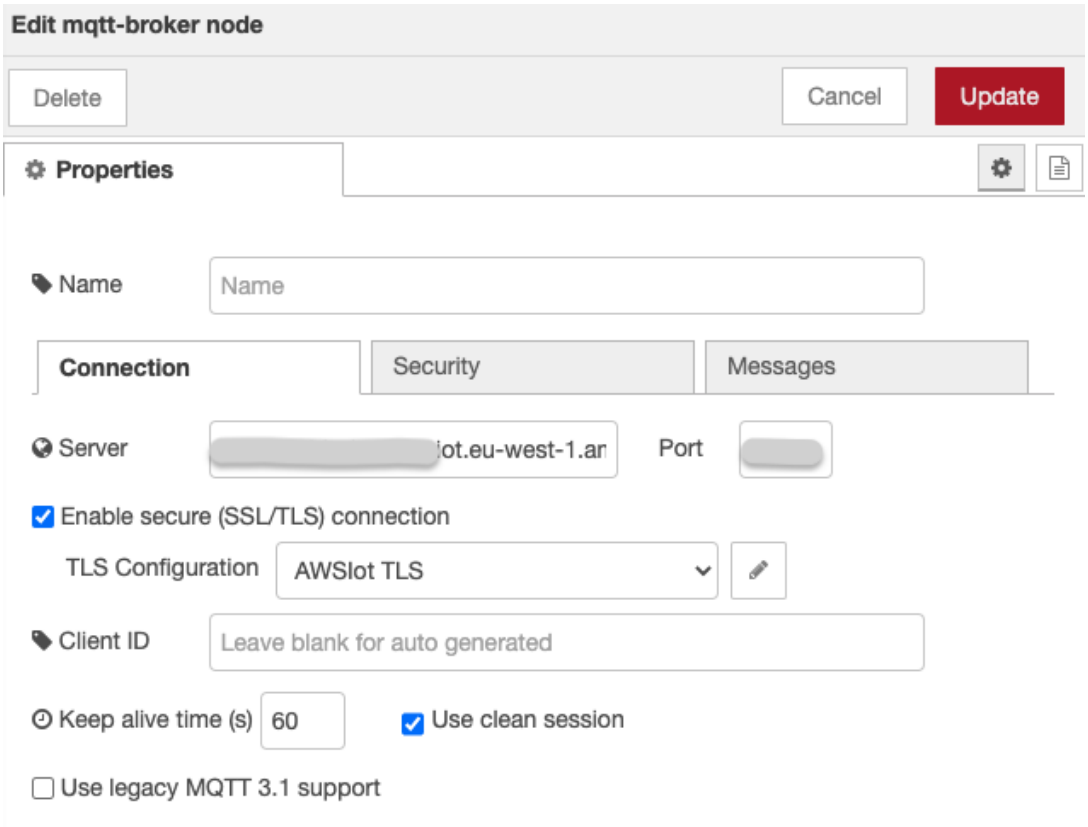

Figura 21.Configuración del nodo mqtt-broker.

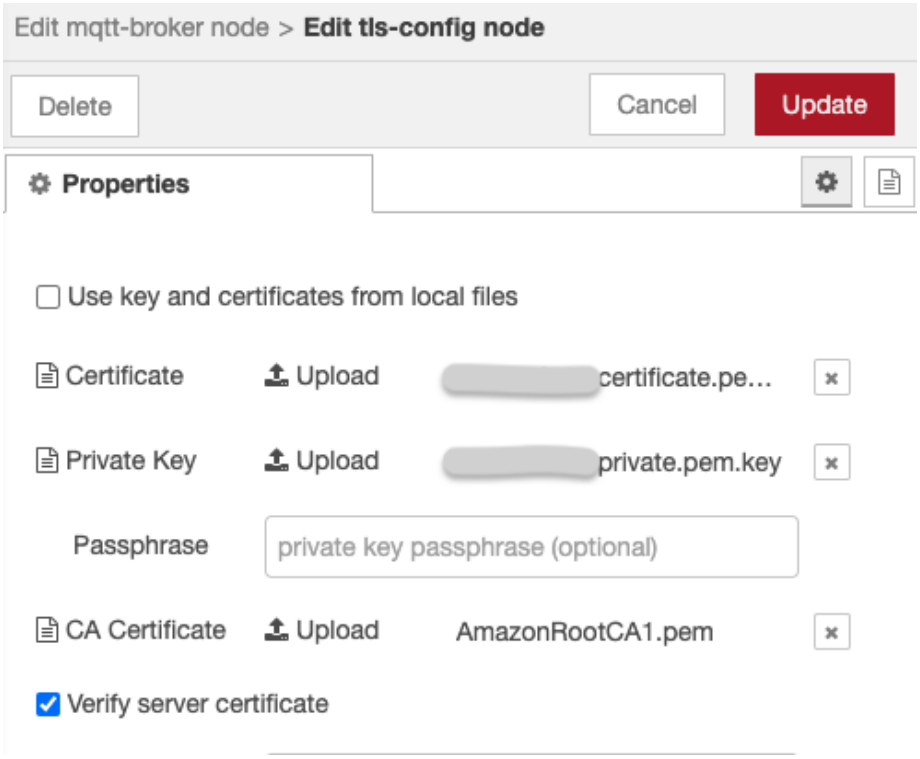

Figura 22. Configuración del nodo tls-config.

Figura 23. Nodo json de Node-RED

El nodo json (ver figura 23) es un parser para modificar un mensaje. Si a la entrada se le pasa un JSON string, el nodo devuelve el objeto JavaScript equivalente. Al contrario, si al nodo se le pasa un objeto JavaScript, éste lo convierte a JSON.

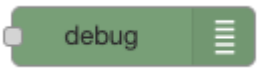

Figura 24. Nodo debug de Node-Red

El nodo debug (ver figura 24) nos permite verificar el valor del mensaje que se mueve por el flujo, en el instante y en la posición que deseamos.

Figura 25. Nodo function de Node-RED

El nodo function (ver figura 25) permite ejecutar código JavaScript contra un objeto msg que acepta a la entrada y devolver cero o más objetos msg a la salida.

El paquete node-red-dashboard (ver figura 26) provee diversos tipos de nodos que nos permiten generar diferentes dashboards para monitorear e interactuar con los datos con los que estamos trabajando.

Figura 26. Nodos text, gauge, switch y slider del paquete node-red-dashboard.

4.7 Funcionalidad y descripción del flujo de Node-RED.

Vamos a explicar la funcionalidad y a describir los flujos de la lógica desarrollada para los diferentes dispositivos de la red domótica.

4.7.1 Motion sensor.

En primer lugar, en la figura 27 se muestra el flujo Node-Red para interactuar con el sensor de movimiento.

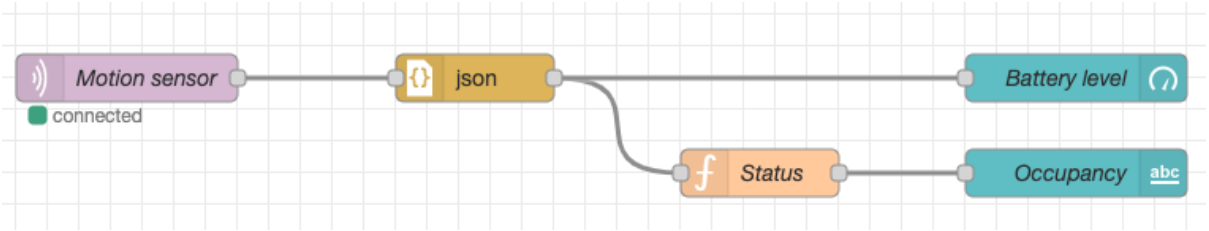

Figura 27. Flujo de Node-RED para el sensor de movimiento.

El flujo de nuestro sensor de movimiento es de monitorización. El nodo Motion sensor que es de tipo mqtt in, se conectaría al nodo mqtt-broker y se configuraría su suscripción al topic

zigbee2mqtt/motion_sensor para obtener los datos del sensor. El mensaje a la salida del nodo Motion sensor es un JSON script, así que deberíamos pasarlo por un nodo json para obtener el objeto JavaScript que lo representa y que por convenio se le llama msg. Dicho objeto tiene una property payload con los datos. Una vez que se modifica el formato del mensaje lo utilizaríamos para generar dos salidas. Una que directamente se pasaría a un nodo gauge para generar un gráfico que muestre el nivel de batería del sensor.

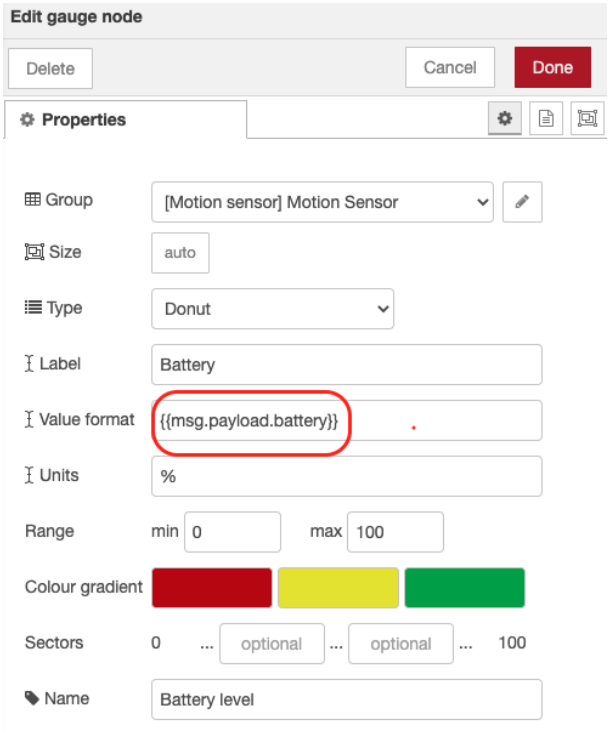

Figura 28. Nodo gauge del flujo del sensor de movimiento.

La otra salida es la que se pasaría a un nodo function para modificar el valor del objeto msg antes de enviarlo a un nodo text que muestre el estado de la habitación donde se encuentra el sensor (ver figura 29 y 30).

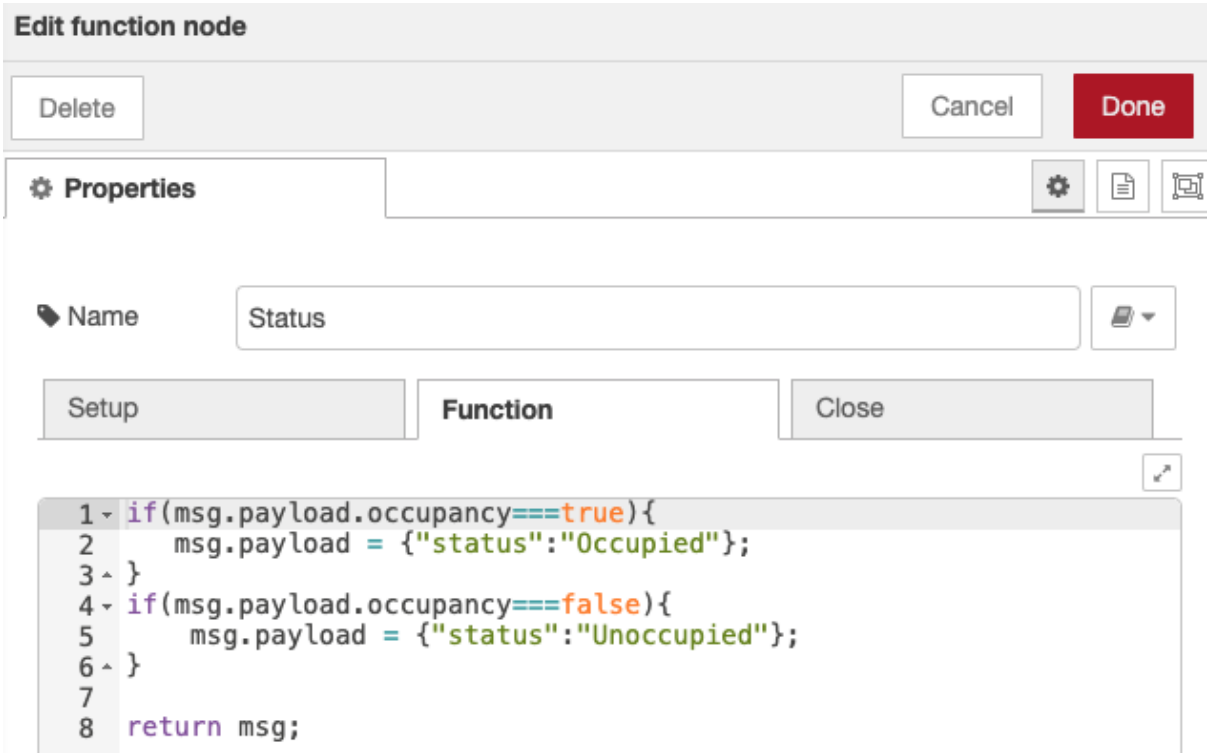

Figura 29. Nodo function del flujo del sensor de movimiento.

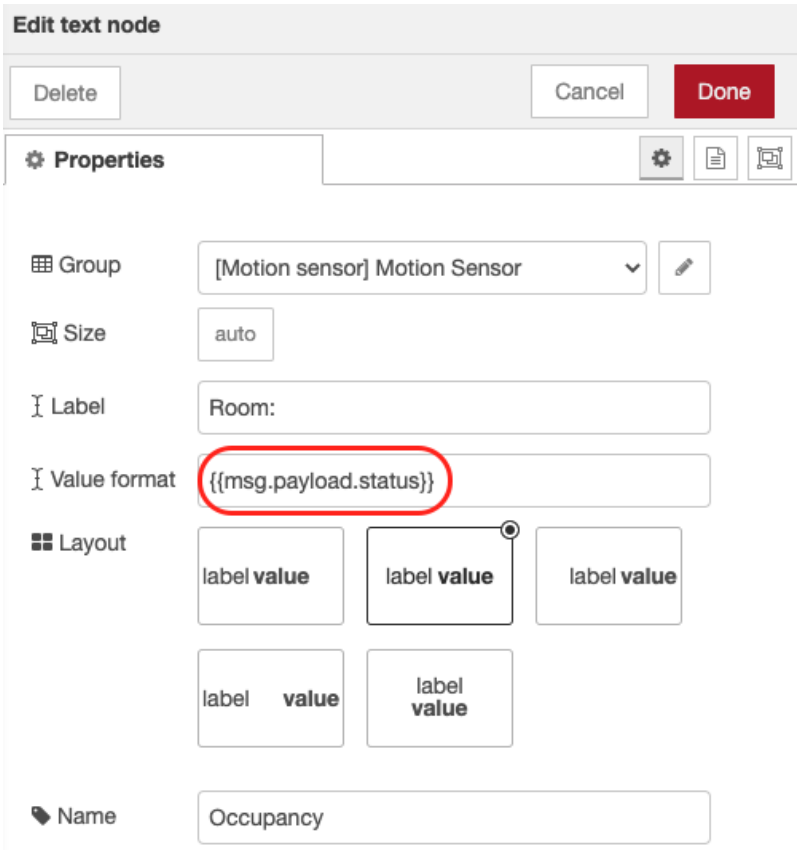

Figura 30. Nodo text del flujo del sensor de movimiento.

4.7.2 Climate sensor.

En la figura 31 se muestra el flujo Node-Red para interactuar con el sensor de temperatura y humedad.

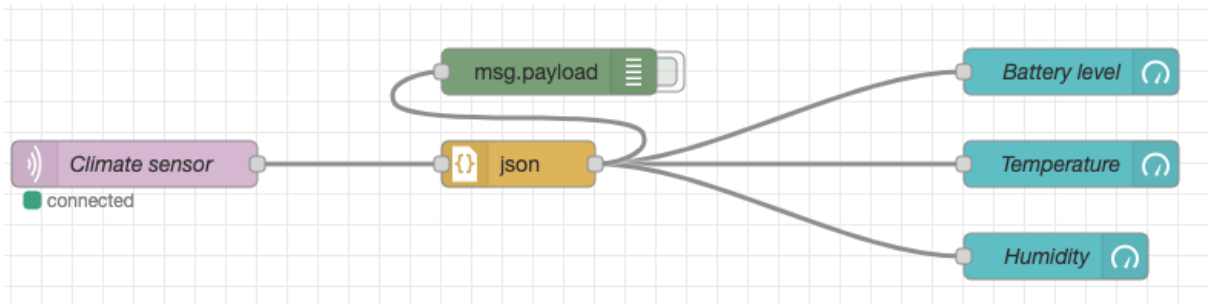

Figura 31. Flujo de Node-RED para el sensor de temperatura y humedad.

El flujo para nuestro sensor de temperatura y humedad está pensado para generar gráficas de monitorización de dichas variables climáticas para conocer el estado de la habitación donde se encuentre el sensor. En este caso el nodo Climate sensor también se conectaría al nodo mqtt-broker suscribiéndose al topic zigbee2mqtt/t&h_sensor. El mensaje de salida de dicho sensor también se pasaría por un nodo json para obtener el objeto msg que representa dicho mensaje. Luego se enviaría la salida a tres nodos gauge para pintar los gráficos de nivel de batería del sensor, temperatura y humedad en función del valor que utilicen del mensaje de entrada a cada nodo; msg.payload.battery, msg.payload.temperature y msg.payload.humidity respectivamente.

4.7.3 Contact sensor.

En la figura 32 se muestra el flujo Node-Red para detectar el estado de la puerta implementado con un sensor de contacto.

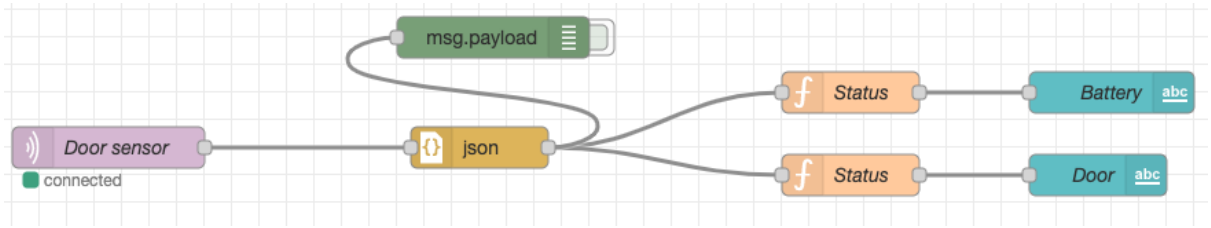

Figura 32. Flujo de Node-RED para el sensor de contacto.

El flujo del sensor de contacto también es un flujo de monitorización. El nodo Door sensor se conectaría al nodo mqtt-broker y se configuraría su suscripción al topic zigbee2mqtt/contact_sensor para leer los mensajes que el sensor envía a ese topic. Los

mensajes pasarían nuevamente por un nodo json para obtener los objetos JavaScript equivalentes y los nuevos mensajes lo utilizamos como entrada a dos nodos function que van a modificar el objeto msg para utilizarlos como entrada en los nodos text. Estos nodos generan un dashboard que mostraría en texto el estado de la batería y el estado del contacto del sensor. En este caso se mostraría en texto porque los mensajes que genera el sensor para representar dichas variables son booleanos. Por ejemplo, en el caso de la batería del sensor de contacto, se muestra en la figura 33 la configuración del nodo function y en la figura 34 la configuración del nodo text que utiliza como entrada la salida del nodo function anterior:

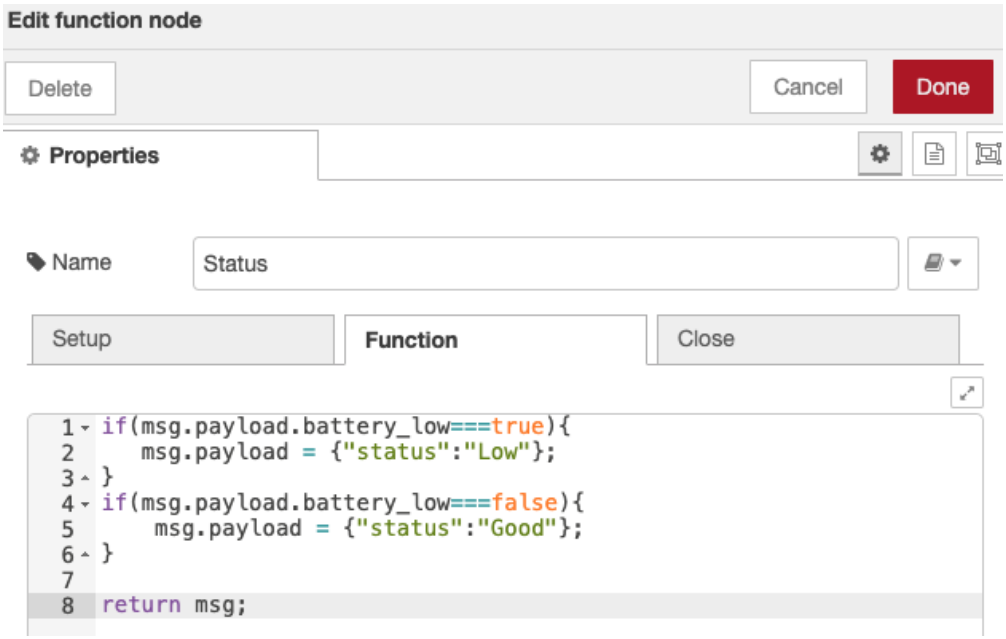

Figura 33. Nodo funtion del flujo del sensor de contacto. Estado de la batería.

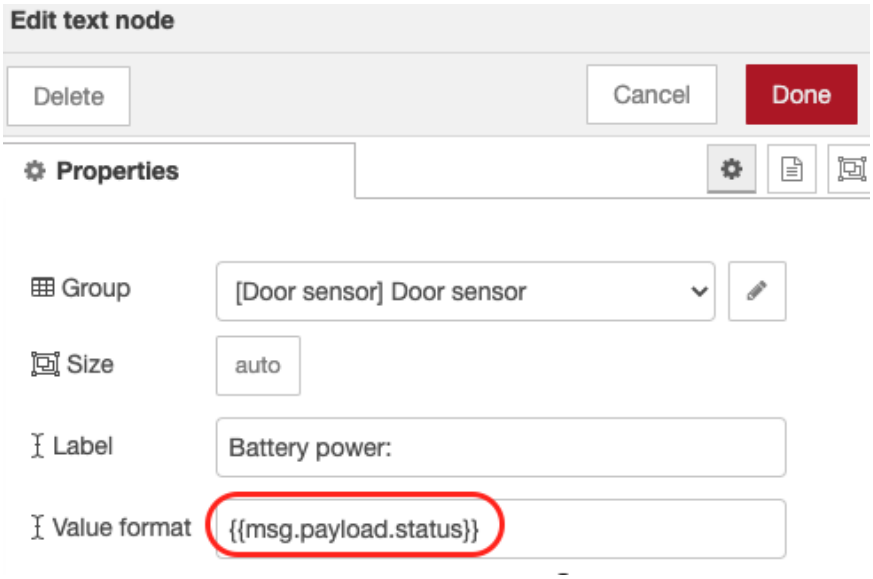

Figura 34. Nodo text del flujo del sensor de contacto. Estado de la batería.

En el caso del estado del contacto es similar la configuración de los nodos utilizados y mostrados en las figuras 35 y 36:

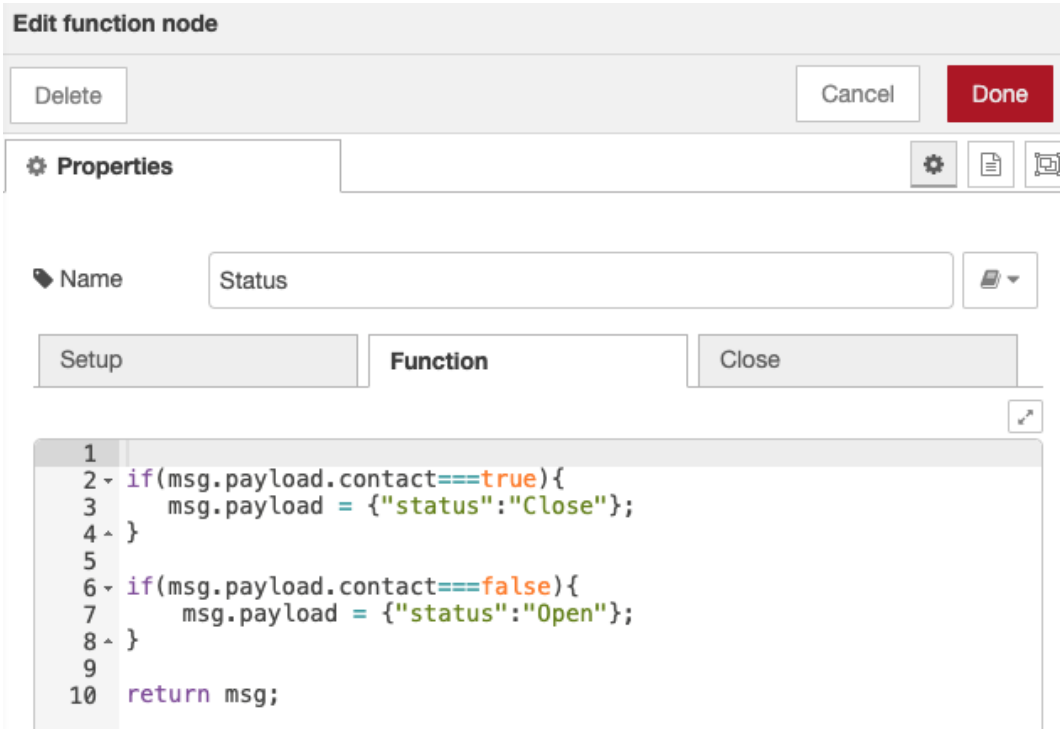

Figura 35. Nodo funtion del flujo del sensor de contacto. Estado de la puerta.

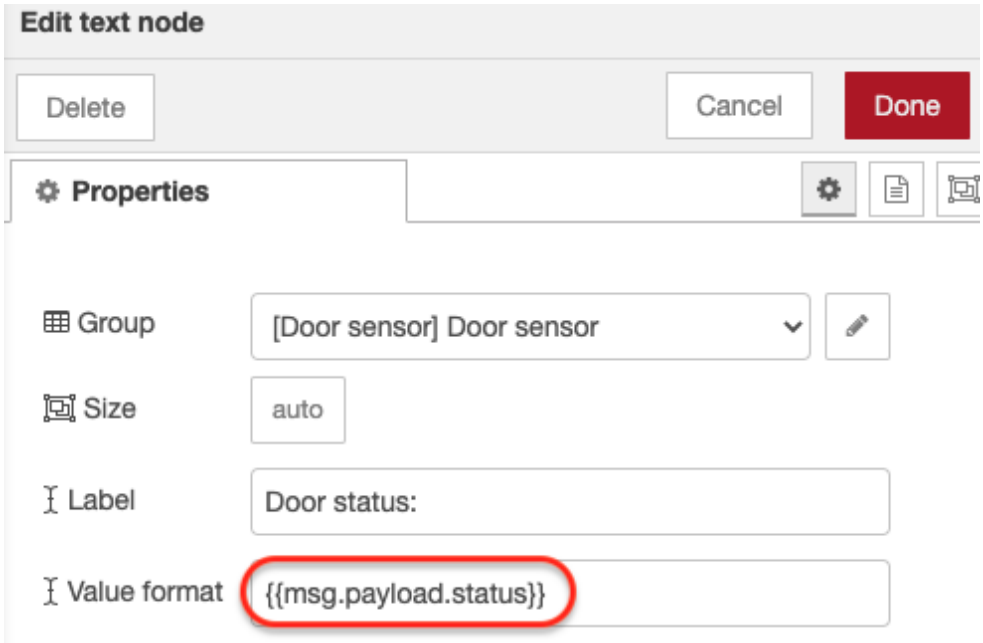

Figura 36. Nodo text del flujo del sensor de contacto. Estado de la puerta.

4.7.4 Switch & Bulb.

En la figura 37 se muestra el flujo Node-Red para controlar el estado de la bombilla, pudiendo modificarse con el switch Zigbee, con un switch virtual y con un regulador de intensidad (dimmer) virtual para variar el brillo de la bombilla.

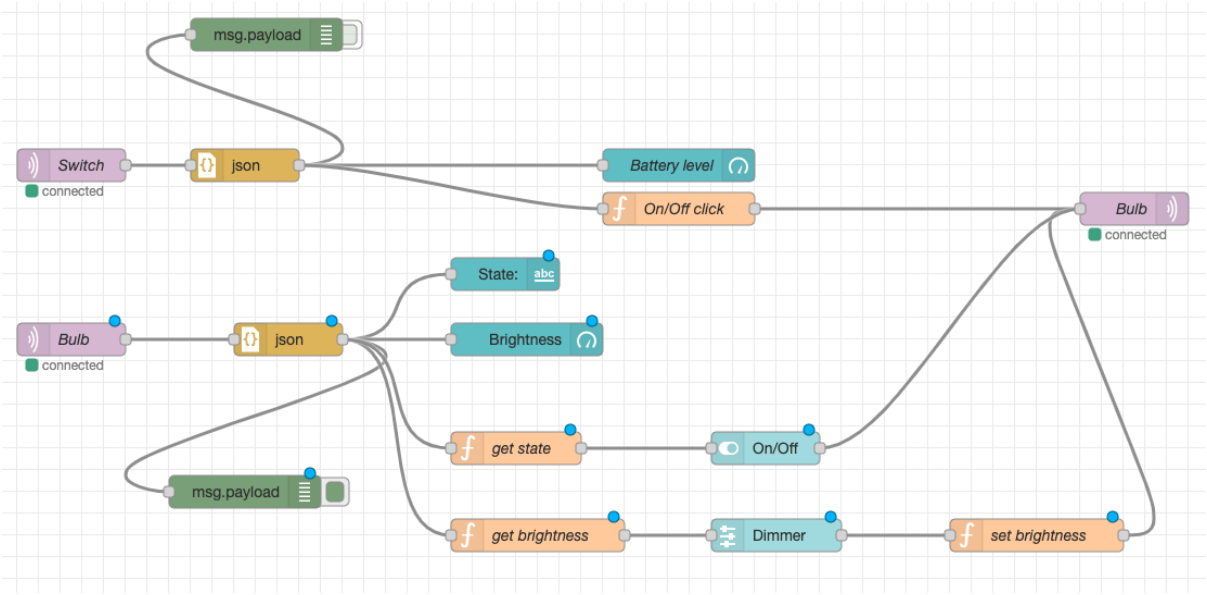

Figura 37. Flujo de Node-RED para el Switch y la bombilla.

El flujo del switch y la bombilla es más complejo ya que hay actuadores que interactúan con un sensor y por lo tanto debemos recibir y enviar mensajes mqtt.

Comencemos describiendo el flujo desde el nodo Switch hasta el nodo Bulb (mqtt out). El nodo Switch se conectaría al nodo mqtt-broker y se configuraría su suscripción al topic zigbee2mqtt/switch para recibir los mensajes del dispositivo actuador que es un switch. Los mensajes a la salida de dicho nodo se pasarían por un nodo json para obtener los objetos JavaScript que nos permitirían luego trabajar y modificar el mensaje. El mensaje a la salida del nodo json lo utilizaríamos de dos formas. Una para generar un gráfico gauge con los valores obtenidos en {{msg.payload.battery}} representando así el porcentaje de batería del switch y la otra salida la pasaríamos como entrada a un nodo function.

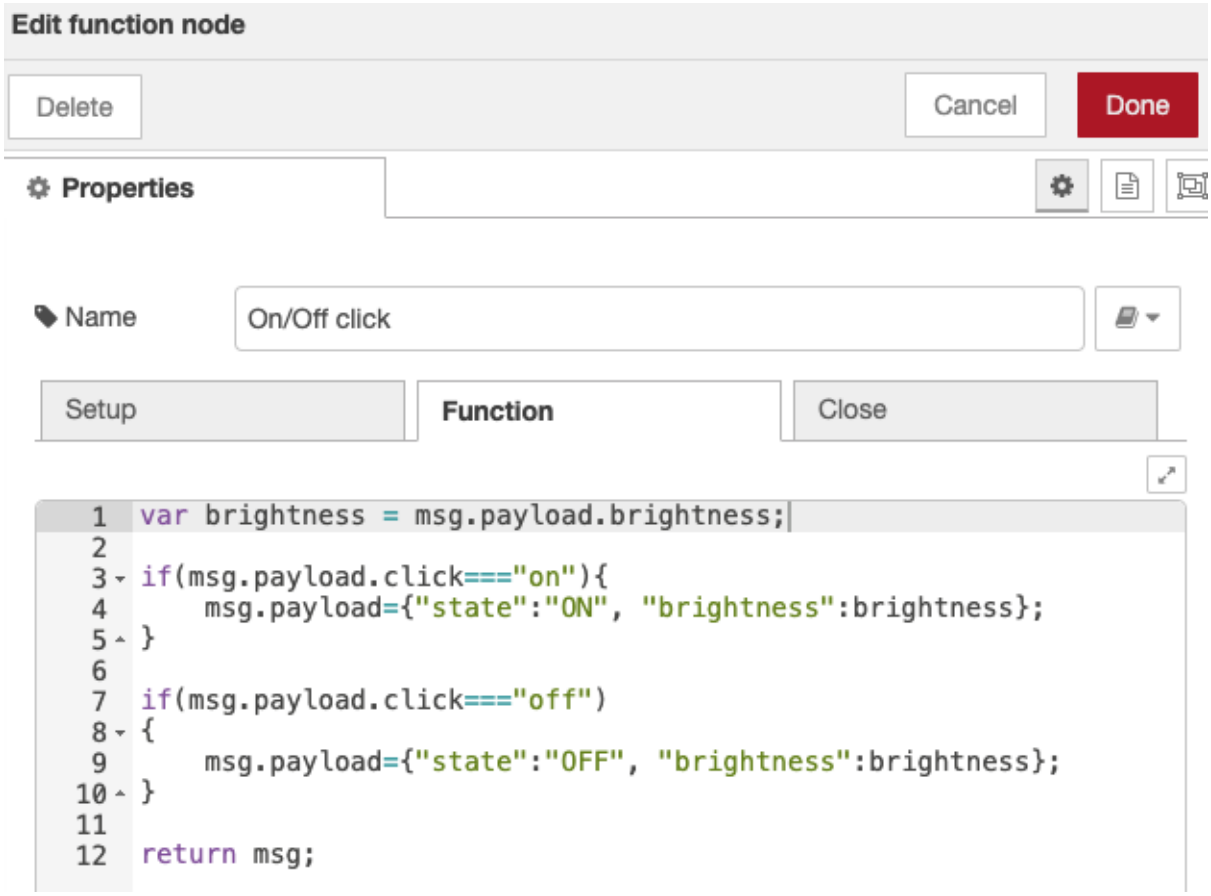

Figura 38. Nodo function del Switch.

Como se muestra en la figura 38, el nodo function lo utilizaríamos para modificar el payload del objeto msg y generar un mensaje que pueda ser interpretado por la bombilla. El mensaje a la salida del nodo function lo enviaríamos al nodo Bulb (mqtt out) el cual estaría conectado al mqtt-broker y suscrito al topic zigbee2mqtt/bulb/set. Con esto conseguiríamos que, si se recibiese un click on u off del switch, la bombilla modifique su estado a ON u OFF respectivamente.

La otra parte de este flujo consiste en el nodo Bulb (mqtt in) que también terminaría conectando con el nodo Bulb (mqtt out). El nodo Bulb (mqtt in) se conectaría al nodo mqtt-broker y se configuraría su suscripción al topic zigbee2mqtt/bulb, que es el topic donde se reciben los mensajes que muestran el estado de la bombilla y el brillo. Estos mensajes los pasaríamos nuevamente por un nodo json para obtener los objetos msg que representan a dichos mensajes. La salida del nodo json la utilizaríamos de cuatro formas.

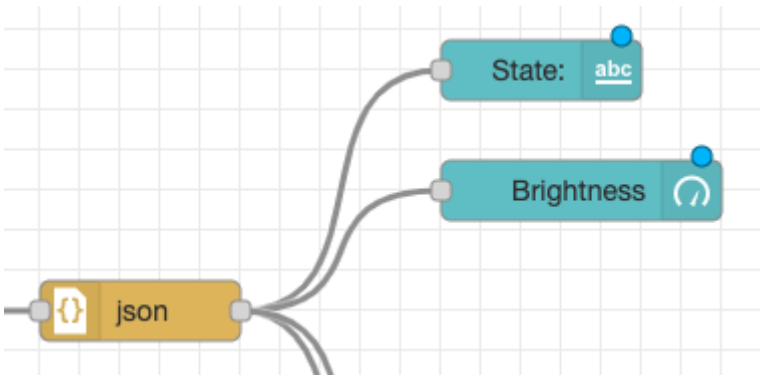

Figura 39. Parte del flujo de la bombilla para representar el estado y el brillo.

Las dos primeras se utilizarían para crear un dashboard de monitorización de la bombilla. Una salida conectaría con la entrada de un nodo text para representar el estado ON/OFF de la bombilla leyendo el valor {{msg.payload.state}} del mensaje que recibe. La segunda conectaría con un nodo gauge para crear un gráfico que represente el brillo actual de la bombilla leyendo el valor {{msg.payload.brightness}} del mismo mensaje.

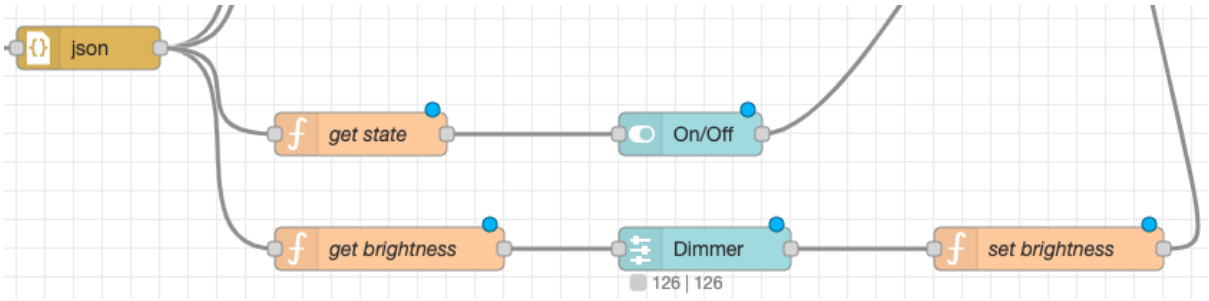

Figura 40. Parte del flujo de la bombilla para crear los actuadores virtuales.

Las otras dos salidas se utilizarían para crear un dashboard de control para poder encender y apagar la bombilla y modificar su brillo. Una de las salidas conectaría con un nodo function (get state) que modifica el valor del payload del objeto msg tal como se muestra en la figura 41.

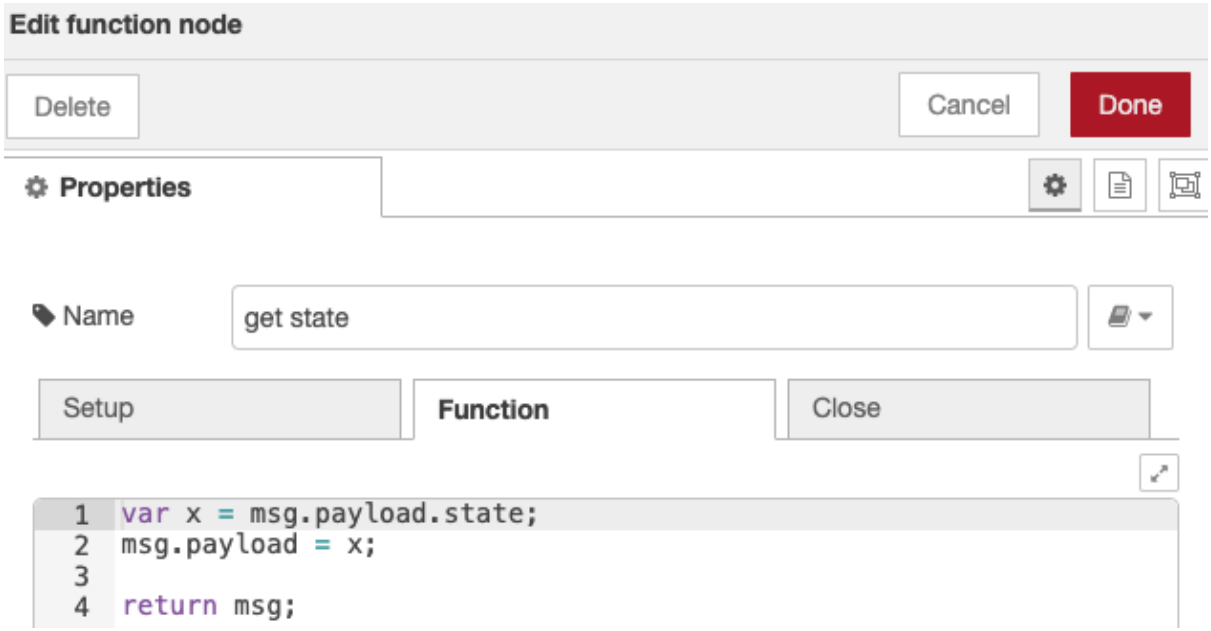

Figura 41. Nodo function para recuperar el estado de la bombilla.

La salida de este nodo conectaría con un nodo switch (On/Off) como se muestra en la figura 40. El nodo On/Off sería un switch virtual que funcionaría como actuador sobre la bombilla. De la forma en que se realizaría la conexión, el switch virtual mantendría la coherencia en el flujo al recibir como entrada el estado actual de la bombilla. Este nodo Switch conectaría finalmente su salida con el nodo Bulb (mqtt out) y de esta forma, al modificarse el estado del switch virtual a través del dashboard en la interfaz gráfica, se enviaría un mensaje a la bombilla para modificarse su estado (encender/apagar).

La última de las salidas del nodo json conectaría con otro nodo function (get brightness) con el cual modificaríamos el valor del payload del objeto msg. Luego se enviaría el valor del brillo actual de la bombilla al nodo Dimmer de tipo slider, el cual representa un regulador de intensidad. De esta forma también mantendría una coherencia dentro del flujo respecto al estado del brillo de la bombilla.

Este dimmer también funcionaría como un actuador virtual sobre la bombilla y según un usuario modifique el valor del brillo en la interfaz gráfica, se generarían nuevos mensajes. Por lo tanto, a la salida del nodo Dimmer conectaríamos otro nodo function (set brightness) que es el encargado de modificar los mensajes para que puedan ser interpretados por la bombilla al enviarlos al topic zigbee2mqtt/bulb/set a través del nodo Bulb (mqtt out) (ver figura 42).

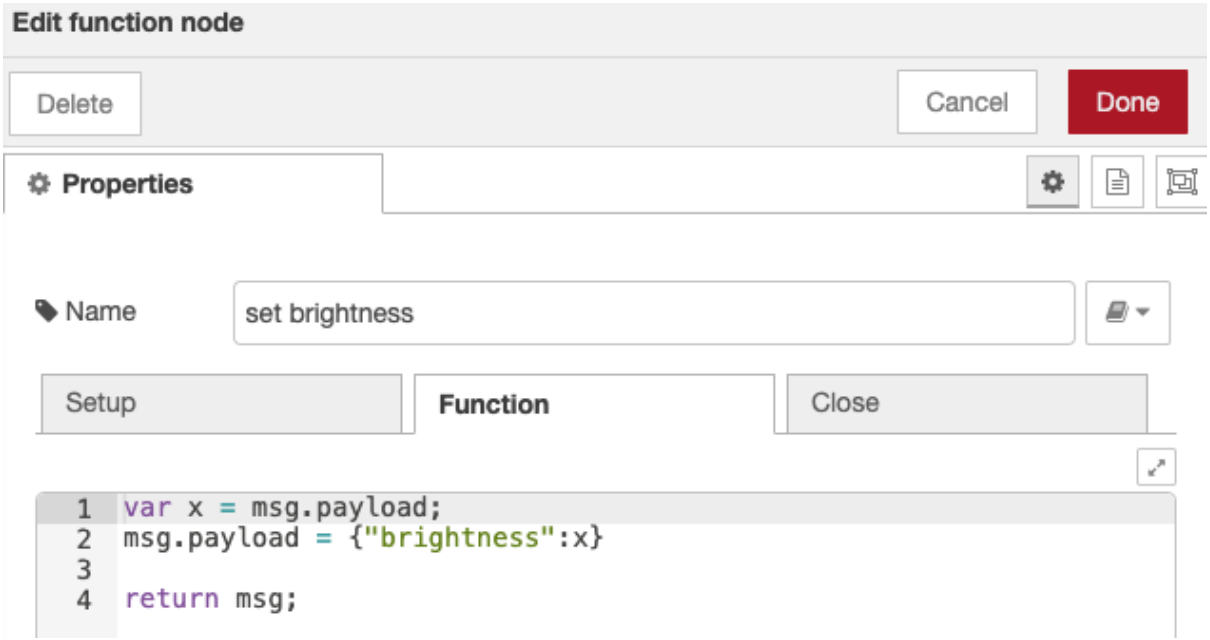

Figura 42. Nodo function para establecer el brillo de la bombilla.

Hasta aquí queda explicado el desarrollo de la lógica de la red domótica. Con los ejemplos desarrollados tendríamos funcionalidades básicas de una Smart Home. Con más tiempo de desarrollo vemos que el funcionamiento podría ser mucho más complejo incluso con los mismos dispositivos que tenemos. Por ejemplo, podríamos conseguir que el sensor de movimiento pudiese hacer de actuador sobre la bombilla, modificando el estado de esta dependiendo de si la habitación está ocupada o no. Sería posible también establecer horarios de encendido y apagado de la bombilla, si la casuística fuese de un alumbrado exterior. Las opciones de desarrollo son tan variadas y flexibles que permiten darle solución a cualquier requisito que se plantee.

CAPÍTULO 5. ANÁLISIS DE RESULTADOS Y PRUEBAS.

En este capítulo vamos a analizar el tráfico de extremo a extremo en toda la arquitectura propuesta para nuestra red domótica. Iremos mostrando los logs de cada uno de los puntos por donde pasa el tráfico real y mostrar su funcionamiento.

5.1 Tráfico Zigbee.

Para comprender el funcionamiento de Zigbee hicimos captura y análisis del tráfico de este protocolo. Para ello utilizamos ZBOSS Sniffer que es un software open source para analizar paquetes Zigbee y que se integra fácilmente con Wireshark. Fue necesario utilizar otro USB Dongle CC2531 como el que tenemos configurado como coordinador de nuestra red Zigbee, pero esta vez con un firmware diferente que provee ZBOSS. Utilizando esto y haciendo modificaciones para escuchar en el canal correcto de Zigbee y añadir las claves de encriptación de nuestra red para poder leer los mensajes, pudimos empezar a ver y analizar el tráfico de la red domótica que tenemos configurada.

Para mostrar de forma sencilla las capas del protocolo Zigbee y los mensajes intercambiados en la red domótica, vamos a trabajar en el caso particular donde encendemos la bombilla con el switch Zigbee.

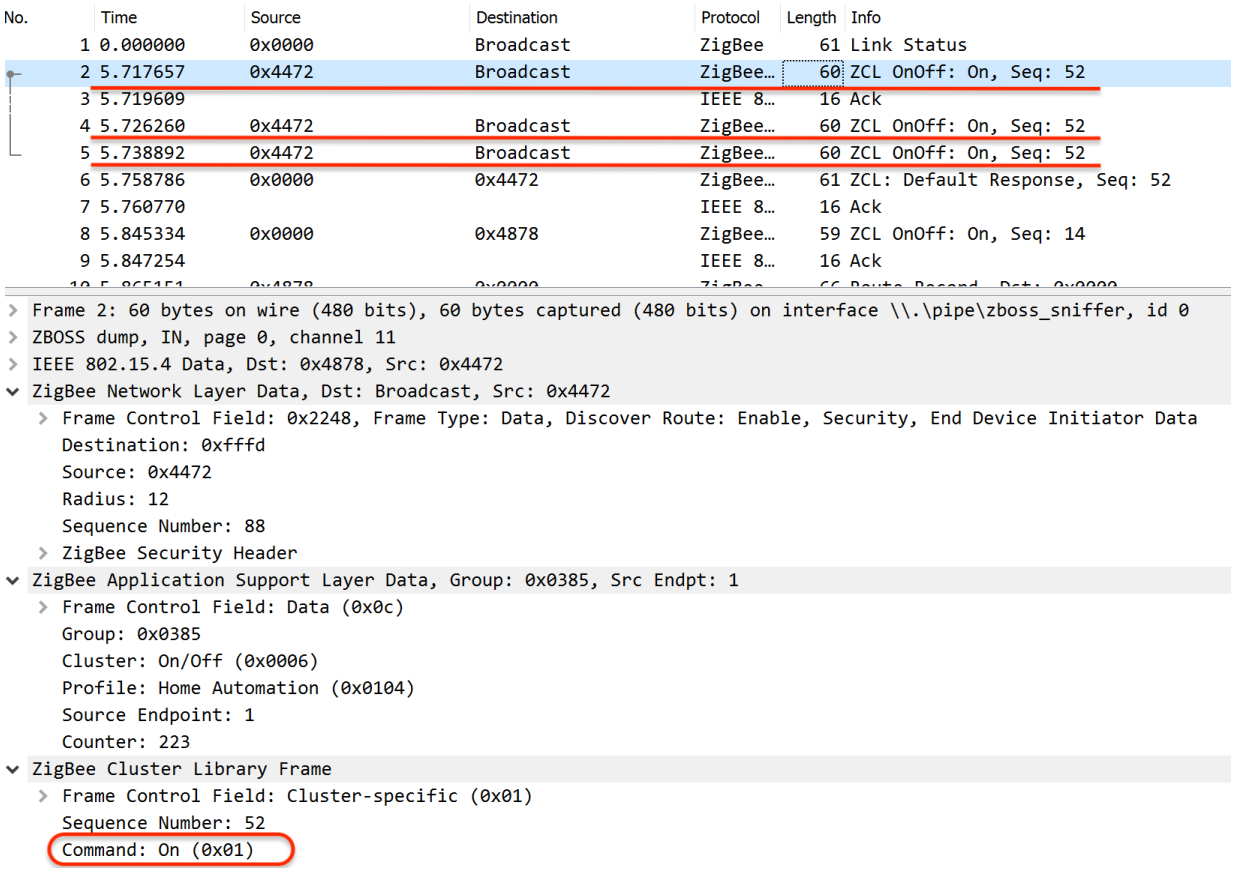

Figura 43. Captura de tráfico Zigbee del Switch al coordinador.

Tal como muestra la figura 43, cuando hacemos clic en el Switch para encender la bombilla vemos que el dispositivo final envía 3 mensajes a broadcast y espera recibir respuesta del coordinador de la red. En esta misma figura vemos en detalle las capas del protocolo Zigbee que hemos mencionado en capítulos anteriores. El profile del firmware del Switch, en la capa APS, muestra que es Home Automation con el cluster On/Off y en el apartado de ZCL se muestra el comando enviado.

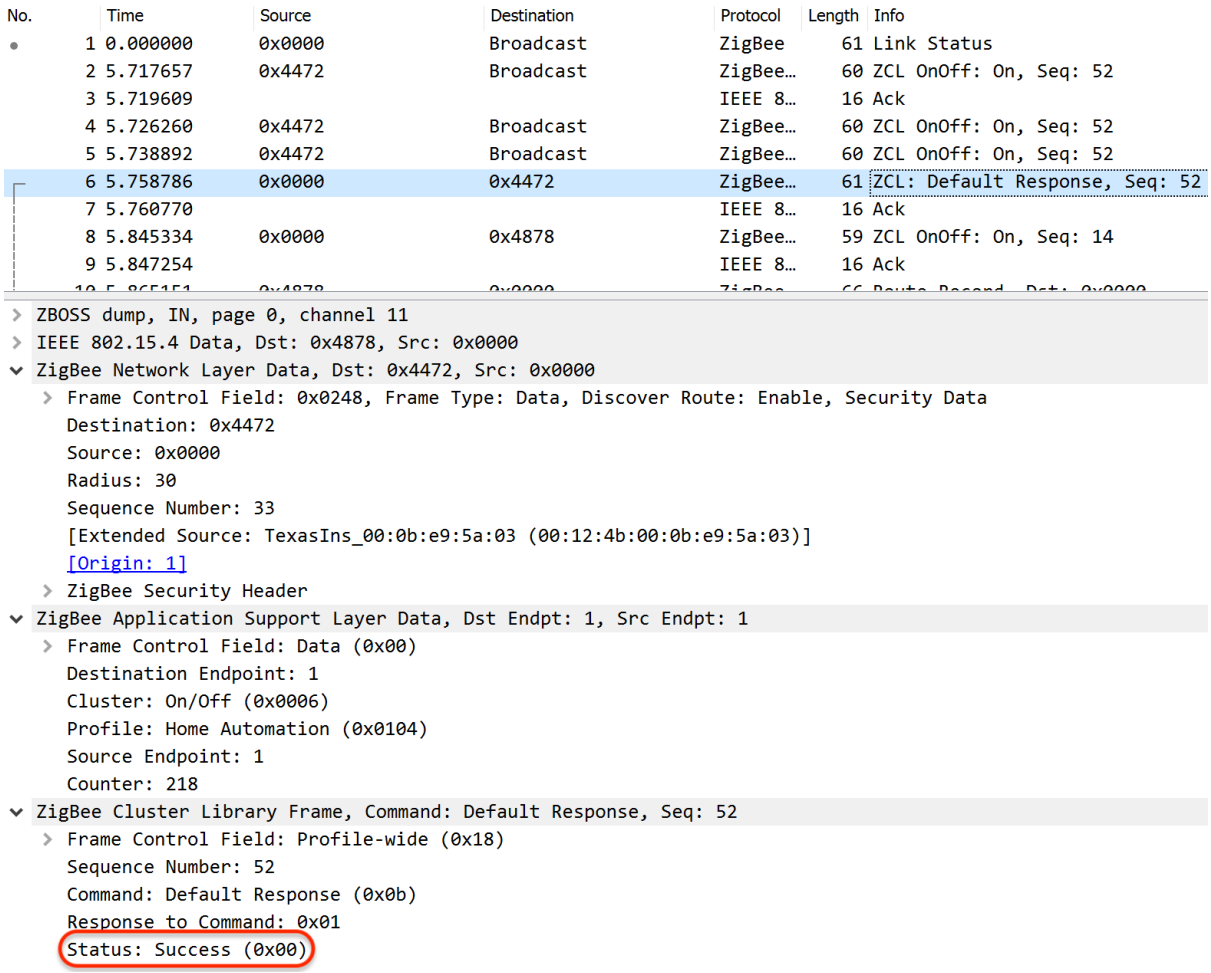

Figura 44. Captura de tráfico Zigbee del coordinador al Switch.

En la figura 44 vemos cómo el coordinador de la red responde al Switch para confirmar el comando recibido.

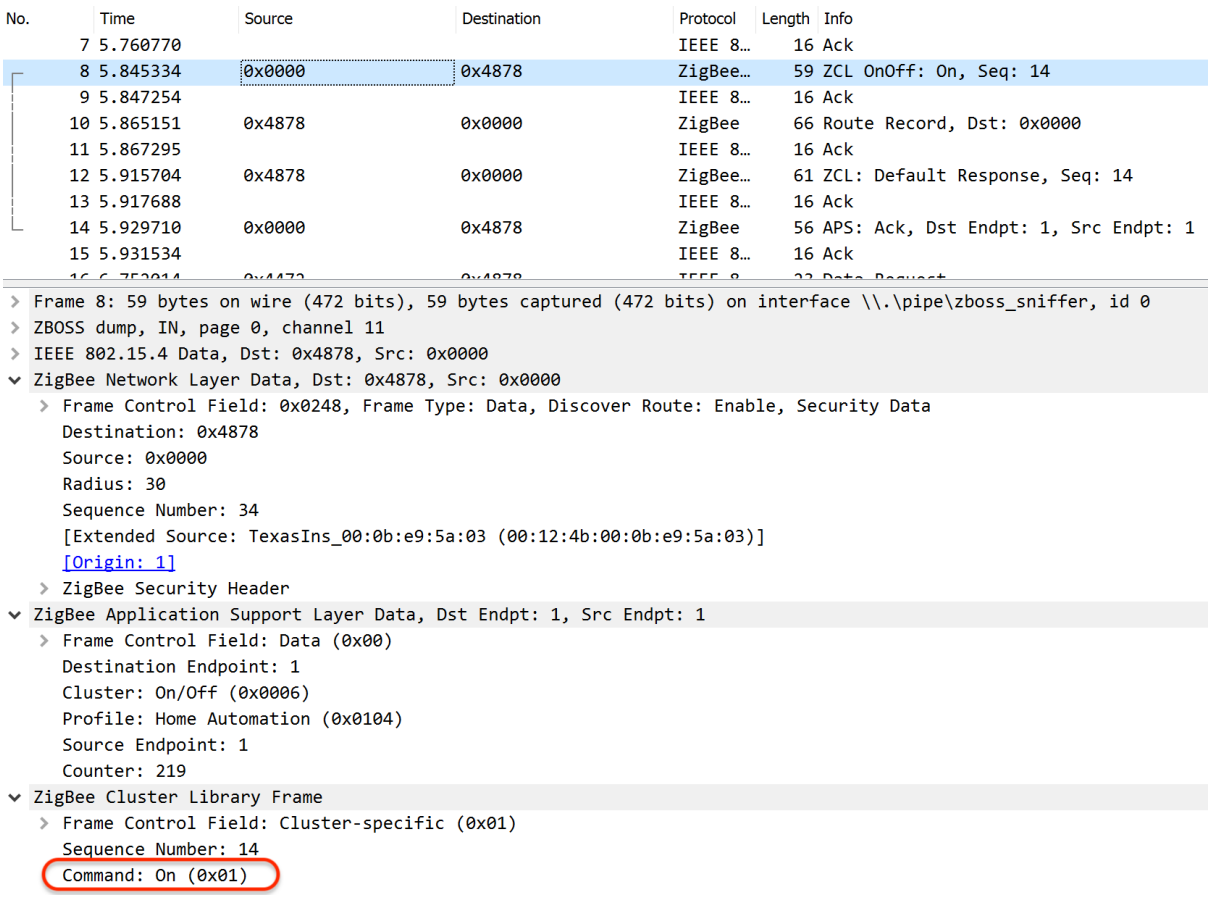

Figura 45. Captura de tráfico Zigbee del coordinador a la bombilla.

Luego que el mensaje recibido por el coordinador viajase por toda la arquitectura y que la lógica de nuestra aplicación determinase que hay que encender la bombilla, el coordinador envía un nuevo mensaje a la bombilla con el mismo comando que indicó el Switch (ver figura 45).

Finalmente, la figura 46 muestra cómo la bombilla envía un mensaje de respuesta por defecto al coordinador indicando el estado de la recepción del comando.

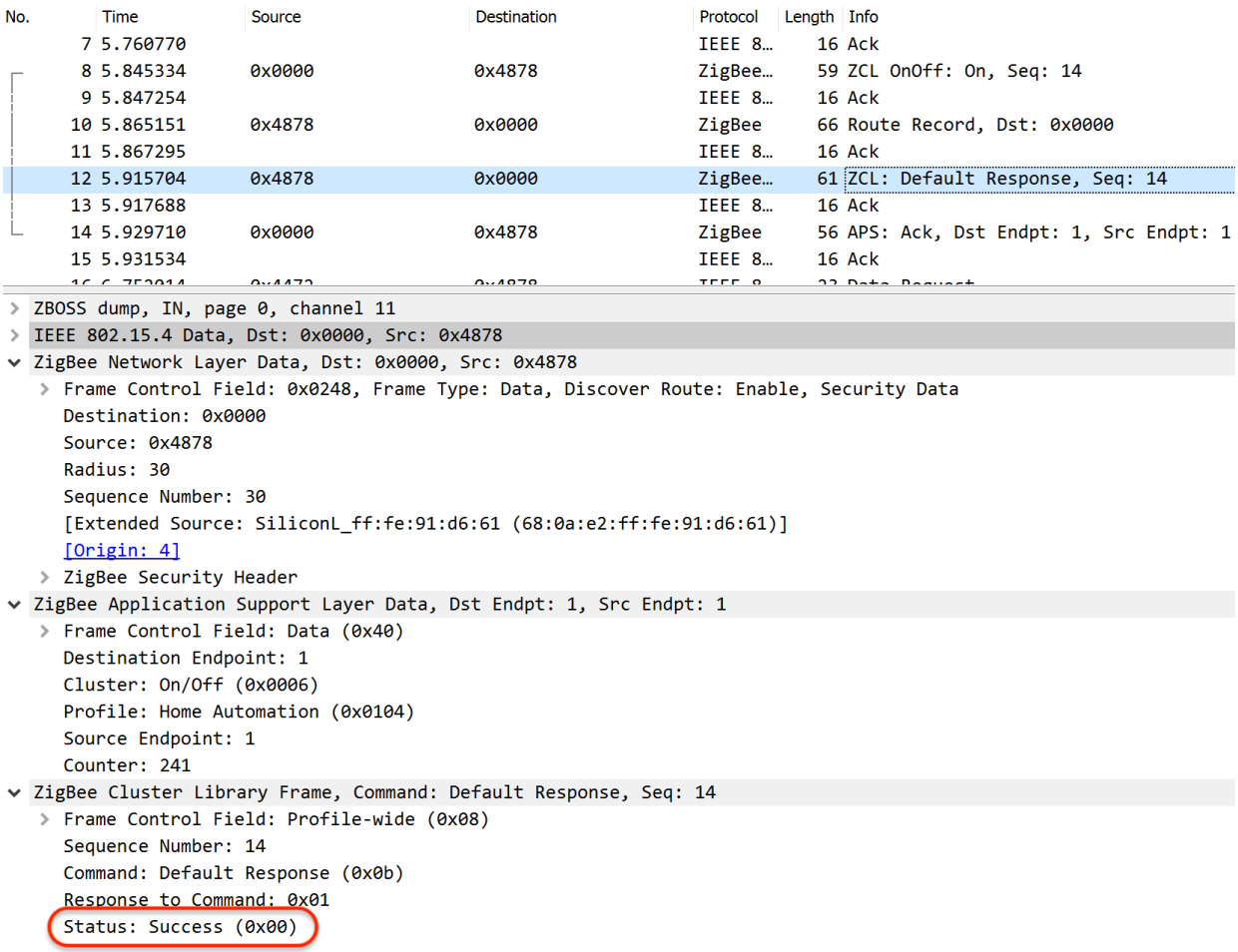

Figura 46. Captura de tráfico Zigbee de la bombilla al coordinador.

5.2 Logs de Zigbee2MQTT.

Como se ha comentado antes, Zigbee2MQTT está configurado para interpretar el tráfico que pasa a través del puerto donde está conectado el USB dongle con el firmware de coordinador de la red Zigbee.

Como se muestra en la figura 47, en las primeras trazas de log al arrancar el programa, vemos que crea el fichero log.txt donde registra todas las acciones. Luego carga el fichero state.json que guarda el último estado de los dispositivos conectados. A continuación, se muestra la versión de Zigbee2MQTT que se está utilizando. Zigbee2MQTT utiliza zigbee-herdsman que es una solución open source de una gateway Zigbee desarrollada para manejar la comunicación de Zigbee. Este módulo zigbee-herdsman arranca con diferentes parámetros que se obtienen del fichero de configuración de Zigbee2MQTT y otros valores por defecto si no son modificados en dicho fichero. Vemos que algunos de estos parámetros son el PAN ID, el extended PAN ID y el canal de la red Zigbee. También el puerto USB de donde está conectado el coordinador y ficheros donde se guardan datos. Luego que zigbee-herdsman

arranca, se muestra la versión del firmware del coordinador que vemos que se basa en zStack1.2 que es el Home Automation. Por último, se muestran los parámetros de la red Zigbee creada por el coordinador.

```
pi@raspberrypi:/opt/zigbee2mqtt/data/log $ tail -100f 2020-10-07.13-26-12/log.txt
info  2020-10-07 13:26:12: Logging to console and directory: '/opt/zigbee2mqtt/
data/log/2020-10-07.13-26-12' filename: log.txt
debug 2020-10-07 13:26:12: Loaded state from file /opt/zigbee2mqtt/data/
state.json
info  2020-10-07 13:26:12: Starting zigbee2mqtt version 1.14.2 (commit #b91b456)
info  2020-10-07 13:26:12: Starting zigbee-herdsman...
debug 2020-10-07 13:26:12: Using zigbee-herdsman with settings: '{"network":
{"panID":6754,"extendedPanID":[221,221,221,221,221,221,221,221],"channelList":
[11],"networkKey":"HIDDEN"},"databasePath":"/opt/zigbee2mqtt/data/
database.db","databaseBackupPath":"/opt/zigbee2mqtt/data/
database.db.backup","backupPath":"/opt/zigbee2mqtt/data/
coordinator_backup.json","serialPort":{"path":"/dev/ttyACM0"},"adapter":
{"concurrent":null}}'
info  2020-10-07 13:26:15: zigbee-herdsman started
info  2020-10-07 13:26:15: Coordinator firmware version:
'{"type":"zStack12","meta":
{"transportrev":2,"product":0,"majorrel":2,"minorrel":6,"maintrel":3,"revision":2
0190608}}'
debug 2020-10-07 13:26:15: Zigbee network parameters:
{"panID":6754,"extendedPanID":"0xdddddddddddddddd","channel":11}
```

Figura 47. Logs de Zigbee2MQTT

En la figura 48 se muestran los dispositivos que se conectan a la red Zigbee y que ya se conocen a partir del estado guardado en el fichero state.json. Luego se deshabilita la opción de que se puedan unir otros dispositivos que no aparecen en el fichero de configuración configuration.yml. Esto es lo que se había configurado anteriormente con la opción permit_join: false

```
info  2020-10-07 13:26:15: Currently 5 devices are joined:
info  2020-10-07 13:26:15: bulb (0x680ae2fffe91d661): LED1733G7 - IKEA TRADFRI
LED bulb E14 600 lumen, dimmable, white spectrum, opal white (Router)
info  2020-10-07 13:26:15: motion_sensor (0xccccccfffedcf96d): E1525/E1745 - IKEA
TRADFRI motion sensor (EndDevice)
info  2020-10-07 13:26:15: switch (0x680ae2fffe80371c): E1743 - IKEA TRADFRI ON/
OFF switch (EndDevice)
info  2020-10-07 13:26:15: t&h_sensor (0x00124b001f84cd82): SNZB-02 - SONOFF
Temperature and humidity sensor (EndDevice)
info  2020-10-07 13:26:15: contact_sensor (0x00124b001f90f708): SNZB-04 - SONOFF
Contact sensor (EndDevice)
info  2020-10-07 13:26:15: Zigbee: disabling joining new devices.
```

Figura 48. Logs de Zigbee2MQTT

Como se verá en la figura 49, una vez que ya están conectados y se reconoce los dispositivos de la red Zigbee, Zigbee2MQTT se conecta el broker MQTT local especificado en el fichero de configuración y comienza a enviar los mensajes MQTT con los datos de cada dispositivo

a los topics que se configuran con el friendly name de cada uno, lo cual también se define en el fichero de configuración.

```
info  2020-10-07 13:26:15: Connecting to MQTT server at mqtt://192.168.1.39:1883
info  2020-10-07 13:26:15: Connected to MQTT server
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/bridge/state',
payload 'online'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/bulb', payload
'{"state":"OFF","brightness":0,"update_available":true,"linkquality":102}'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/motion_sensor',
payload
'{"occupancy":true,"linkquality":92,"update_available":false,"battery":60}'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/switch', payload
'{"update_available":false,"linkquality":78,"battery":87}'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/t&h_sensor', payload
'{"temperature":22.68,"linkquality":55,"humidity":55.09,"battery":100,"voltage":3
200}'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/contact_sensor',
payload '{"contact":true,"tamper":false,"battery_low":false,"linkquality":89}'
info  2020-10-07 13:26:15: MQTT publish: topic 'zigbee2mqtt/bridge/config',
payload '{"version":"1.14.2","commit":"b91b456","coordinator":
{"type":"zStack12","meta":
{"transportrev":2,"product":0,"majorrel":2,"minorrel":6,"maintrel":3,"revision":2
0190608}},"log_level":"debug","permit_join":false}'
```

Figura 49. Logs de Zigbee2MQTT

Para mostrar el comportamiento de Zigbee2MQTT y su función como bridge entre Zigbee y MQTT en nuestra arquitectura, en las siguientes trazas de log mostramos el ejemplo particular en la parte del flujo donde encendemos la bombilla a través del switch. Este ejemplo es ideal porque mostramos Zigbee2MQTT tanto recibiendo tráfico Zigbee y enviando mensajes MQTT, como recibiendo mensajes MQTT y enviando mensajes Zigbee (la orden a la bombilla para que se encienda).

Al hacer clic en el switch, lo primero que observamos en la figura 50 es que Zigbee2MQTT recibió un mensaje Zigbee. Seguidamente publicó un mensaje MQTT al topic 'zigbee2mqtt/switch' con un payload con la info que interpreta que envió el dispositivo. En la lógica de nuestra red, desarrollamos que si se recibe un mensaje del switch que contenga en el payload “click”:”on” o “click”:”off”, se envía un mensaje MQTT al topic 'zigbee2mqtt/bulb/set' para modificar el estado de la bombilla en correspondencia con el switch. Por esta razón, la siguiente línea de log indica que Zigbee2MQTT recibió un mensaje MQTT en el topic 'zigbee2mqtt/bulb/set' con los datos '{"state":"ON"}' y este mensaje lo publicó en la red Zigbee. Luego se observa que recibió otro mensaje Zigbee con el nuevo estado de la bombilla y lo publicó en el topic MQTT 'zigbee2mqtt/bulb'.

```
debug 2020-10-07 13:39:11: Received Zigbee message from 'switch', type
'commandOn', cluster 'genOnOff', data '{}' from endpoint 1 with groupID 901
info  2020-10-07 13:39:11: MQTT publish: topic 'zigbee2mqtt/switch', payload
'{"update_available":false,"linkquality":76,"battery":87,"click":"on"}'
debug 2020-10-07 13:39:11: Received MQTT message on 'zigbee2mqtt/bulb/set' with
data '{"state":"ON"}'
debug 2020-10-07 13:39:11: Publishing 'set' 'state' to 'bulb'
debug 2020-10-07 13:39:11: Received Zigbee message from 'bulb', type
'readResponse', cluster 'genLevelCtrl', data '{"currentLevel":182}' from endpoint
1 with groupID 0
info  2020-10-07 13:39:11: MQTT publish: topic 'zigbee2mqtt/bulb', payload
'{"state":"OFF","brightness":182,"update_available":true,"linkquality":78}'
```

Figura 50. Logs de Zigbee2MQTT

El tráfico relacionado con los otros dispositivos como el sensor de movimiento, el sensor de contacto y el sensor de temperatura y humedad, se reenvía en el bridge Zigbee2MQTT en la dirección desde la red Zigbee hasta crear los mensajes MQTT. Esto es porque no son dispositivos sobre los que ejercen cambios otros dispositivos actuadores en la lógica explicada en el capítulo anterior.

En la figura 51 se muestra cómo se reciben datos del sensor de movimiento cuando se activa y luego se publican en el topic zigbee2mqtt/motion_sensor.

```
debug 2020-10-07 14:36:51: Received Zigbee message from 'motion_sensor', type
'commandOnWithTimedOff', cluster 'genOnOff', data
'{"ctrlbits":0,"ontime":1800,"offwaittime":0}' from endpoint 1 with groupID 0
info  2020-10-07 14:36:51: MQTT publish: topic 'zigbee2mqtt/motion_sensor',
payload
'{"occupancy":true,"linkquality":57,"update_available":false,"battery":60}'
```

Figura 51. Logs de Zigbee2MQTT

En ejemplo mostrado en la figura 52 es similar el caso anterior. Vemos como se abre y cierra la puerta donde está el sensor de contacto. En cada paso se recibe un mensaje Zigbee con los datos del sensor y son publicados en un mensaje MQTT en el topic zigbee2mqtt/contact_sensor

```
debug 2020-10-07 14:36:56: Received Zigbee message from 'contact_sensor', type
'commandStatusChangeNotification', cluster 'ssIasZone', data
'{"zonestatus":1,"extendedstatus":0}' from endpoint 1 with groupID 0
info  2020-10-07 14:36:56: MQTT publish: topic 'zigbee2mqtt/contact_sensor',
payload '{"contact":false,"tamper":false,"battery_low":false,"linkquality":57}'
debug 2020-10-07 14:36:58: Received Zigbee message from 'contact_sensor', type
'commandStatusChangeNotification', cluster 'ssIasZone', data
'{"zonestatus":0,"extendedstatus":0}' from endpoint 1 with groupID 0
info  2020-10-07 14:36:58: MQTT publish: topic 'zigbee2mqtt/contact_sensor',
payload '{"contact":true,"tamper":false,"battery_low":false,"linkquality":60}'
```

Figura 52. Logs de Zigbee2MQTT

En este último ejemplo que se muestra en la figura 53, vemos cuando se reciben los datos del sensor de temperatura y humedad y luego se envía un mensaje MQTT al topic zigbee2mqtt/t&h_sensor.

```
debug 2020-10-07 14:32:09: Received Zigbee message from 't&h_sensor', type
'attributeReport', cluster 'msRelativeHumidity', data '{"measuredValue":5496}'
from endpoint 1 with groupID 0
info  2020-10-07 14:32:09: MQTT publish: topic 'zigbee2mqtt/t&h_sensor', payload
'{"temperature":23.1,"linkquality":68,"humidity":54.96,"battery":100,"voltage":30
00}'
```

Figura 53. Logs de Zigbee2MQTT.

5.3 Broker MQTT local y el bridge a AWS.

Mosquito, que es el broker MQTT local, recibe diferentes mensajes del cliente MQTT de Zigbee2MQTT. Muchos son referentes básicamente a la configuración del bridge Zigbee-MQTT, pero los que realmente nos interesan son los mensajes relacionados con los dispositivos Zigbee de nuestra red domótica. Dichos dispositivos están suscritos a una serie de topics que configuramos en el capítulo anterior para decidir qué mensajes navegan a través del bridge entre el broker local y el broker cloud.

En los siguientes logs que se muestran en la figura 54, vemos cómo Mosquitto conecta con el broker cloud de AWS IoT Core y envía la petición para suscribirse a los topics. Mosquitto utiliza un cliente MQTT para conectar con el broker cloud y establecer el bridge. Según hemos configurado en el capítulo anterior, el id de dicho cliente es ‘bridgeawsiot’.

```
1602089153: Connecting bridge (step 1) awsiot ([redacted]iot.eu-
west-1.amazonaws.com:8883)
1602089153: Connecting bridge (step 2) awsiot ([redacted]iot.eu-
west-1.amazonaws.com:8883)
1602089153: Bridge bridgeawsiot sending CONNECT
1602089153: Received CONNACK on connection local.bridgeawsiot.
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 1, Topic:
zigbee2mqtt/switch, QoS: 1)
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 2, Topic:
zigbee2mqtt/bulb, QoS: 1)
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 3, Topic:
zigbee2mqtt/bulb/set, QoS: 1)
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 4, Topic:
zigbee2mqtt/motion_sensor, QoS: 1)
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 5, Topic:
zigbee2mqtt/t&h_sensor, QoS: 1)
1602089153: Bridge local.bridgeawsiot sending SUBSCRIBE (Mid: 6, Topic:
zigbee2mqtt/contact_sensor, QoS: 1)
```

Figura 54. Logs del bridge entre Mosquitto y AWS IoT Core.

Luego de establecer el bridge entre los brokers local y cloud, si Mosquitto recibe un mensaje del cliente de Zigbee2MQTT (mqttjs_0df444c7) y dicho mensaje es de uno de los topics en cuestión, entonces lo reenvía al broker cloud a través del cliente 'bridgeawsiot' que establece la conexión. Veamos el siguiente ejemplo en la figura 55 cuando el sensor de movimiento y el sensor de contacto envían datos, el caso que se da cuando alguien entra en la habitación donde están ubicados dichos sensores.

```
1602089568: Received PUBLISH from mqttjs_0df444c7 (d0, q0, r0, m0,
'zigbee2mqtt/motion_sensor', ... (73 bytes))
1602089568: Sending PUBLISH to local.bridgeawsiot (d0, q0, r0, m0,
'zigbee2mqtt/motion_sensor', ... (73 bytes))
1602089568: Received PUBLISH from local.bridgeawsiot (d0, q0, r0, m0,
'zigbee2mqtt/motion_sensor', ... (73 bytes))
1602089616: Received PUBLISH from mqttjs_0df444c7 (d0, q0, r0, m0,
'zigbee2mqtt/contact_sensor', ... (69 bytes))
1602089616: Sending PUBLISH to local.bridgeawsiot (d0, q0, r0, m0,
'zigbee2mqtt/contact_sensor', ... (69 bytes))
1602089616: Received PUBLISH from local.bridgeawsiot (d0, q0, r0, m0,
'zigbee2mqtt/contact_sensor', ... (69 bytes))
```

Figura 55. Logs del bridge entre Mosquitto y AWS IoT Core

Para ambos casos, tanto para el sensor de movimiento como para el sensor de contacto, vemos que se reciben los mensajes PUBLISH del cliente MQTT de Zigbee2MQTT. Mosquitto al recibir estos mensajes envía PUBLISH al cliente que conecta con el broker MQTT cloud. El broker MQTT cloud, al recibir el mensaje PUBLISH, lo publica en el topic correspondiente para que los clientes conectados al IoT Core reciban los mensajes. Debido a esto, además de recibir los mensajes el cliente MQTT de Node-RED, también lo vuelve a recibir Mosquitto.

Es importante resaltar que aparece desactivado el flag de message retain (r0) debido a la configuración de Zigbee2MQTT que vimos en el capítulo anterior. Recordemos que esto era necesario pues AWS no soporta la funcionalidad de MQTT de retener los mensajes en el broker cloud.

5.4 Debug en Node-RED.

Una vez que establecida la conexión con el broker de AWS IoT Core y gracias a las configuraciones realizadas en AWS en cuanto a políticas y roles de servicios creados para poder conectar EC2 con IoT Core, ya fuimos capaces de poder manejar en cloud los mensajes de los dispositivos de la red domótica. En este apartado vamos a analizar los mensajes dentro de la lógica desarrollada en Node-RED. Para ello vamos a retomar el ejemplo de encender y apagar la luz con el switch Zigbee.

Una vez que tenemos configurado el nodo que se encarga de recibir los mensajes del topic del switch, ubicamos un nodo de debug (nodo verde) a la salida para ver los mensajes que llegaban (ver figura 56 y 57).

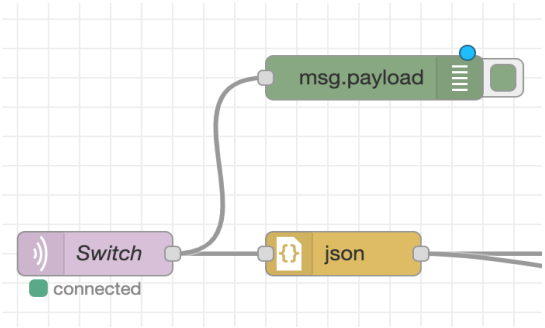

Figura 56. Nodo debug (en verde) a la salida de mqtt-in del Switch.

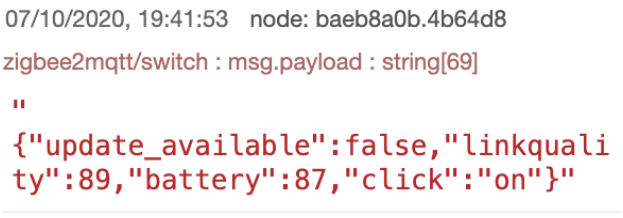

Figura 57. Mensaje recibido en el nodo debug a la salida de mqtt-in del Switch.

De esta forma nos dimos cuenta de que era necesario poder convertir a un objeto JavaScript el mensaje que recibimos para poder trabajar con él. Estos mensajes viajan desde la red Zigbee, pasando por el bridge Zigbee2MQTT y luego desde el broker MQTT local al cloud al que se conectan todos los nodos que utilizamos en la lógica desarrollada.

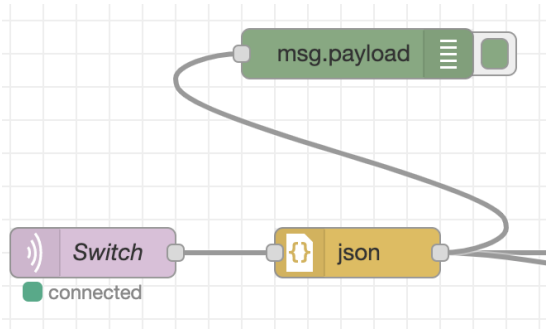

Figura 58. Nodo debug (en verde) a la salida del nodo json del Switch.

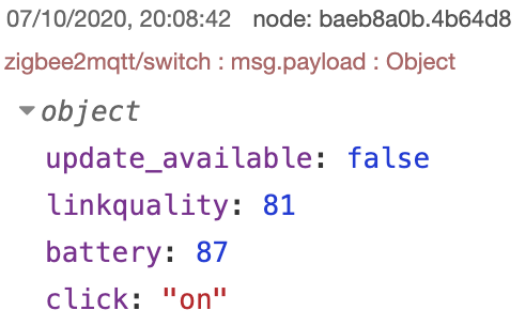

Figura 59. Mensaje recibido en el nodo debug a la salida del nodo json del Switch.

A la salida del nodo json se observa (ver figura 58 y 59) que ya podemos trabajar con el objeto msg para modificarlo y poder enviar un mensaje al topic de la bombilla.

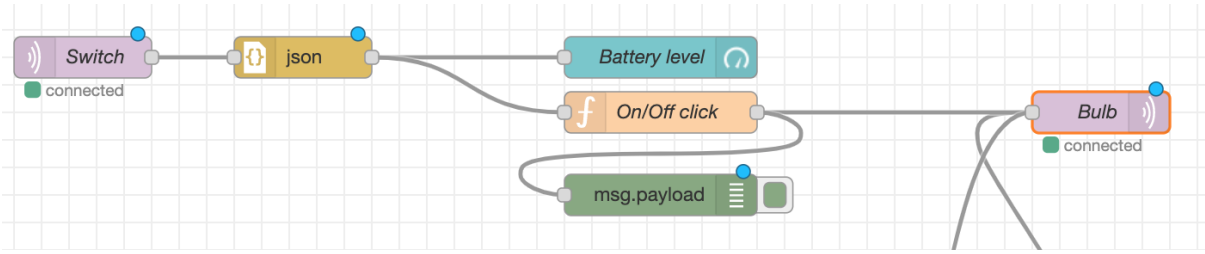

Figura 60. Nodo debug (en verde) a la salida del nodo function del Switch.

Un nodo debug a la salida de la función “On/Off click” (ver figura 60) nos muestra el mensaje modificado y listo para enviarlo al nodo Bulb (mqtt out) que como ya hemos visto está suscrito al topic ‘zigbee2mqtt/bulb/set’ (ver figura 61).

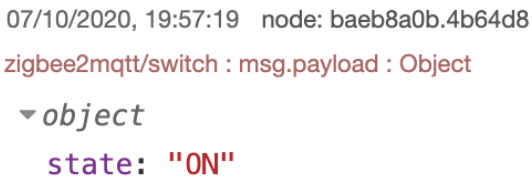

Figura 61. Mensaje recibido en el nodo debug a la salida del nodo funtion del Switch.

El mensaje enviado viaja del broker cloud al broker local, luego al bridge de Zigbee2MQTT y finalmente a la red Zigbee cambiando el estado de la bombilla. Al cambiar el estado, se envía un mensaje de actualización de estado que, como ya hemos visto, recibe el bridge Zigbee2MQTT y lo envía al broker local, quien a su vez lo reenvía al broker cloud a través del bridge MQTT.

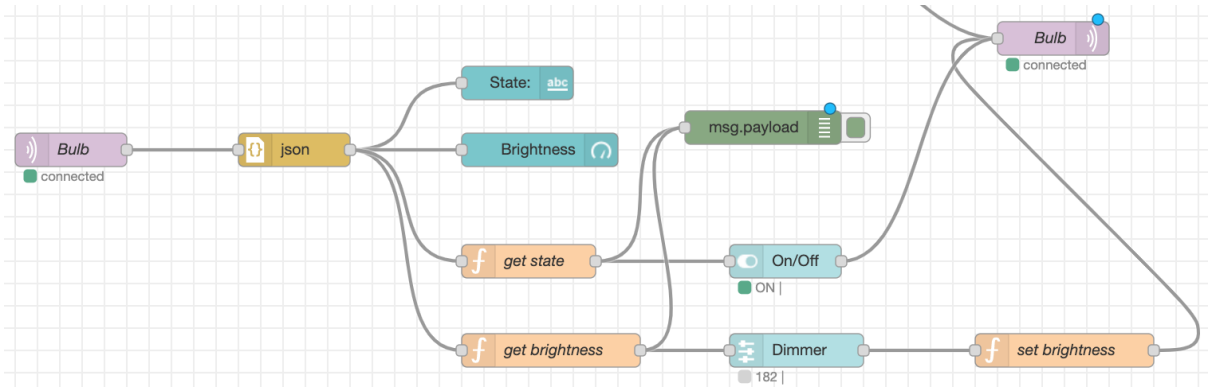

Figura 62. Nodo debug (en verde) a la salida de los nodos function del flujo de la bombilla.

El mensaje lo recibe el nodo Bulb (mqtt in) y se crea el objeto msg al pasar por el nodo json (ver figuras 63 y 64).

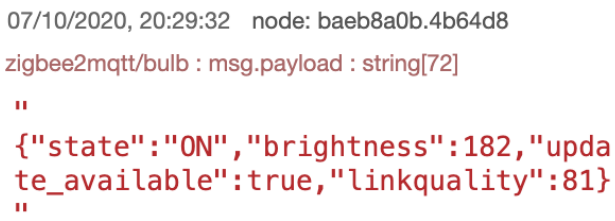

Figura 63. Mensaje recibido a la salida del nodo mqtt-in de la bombilla.

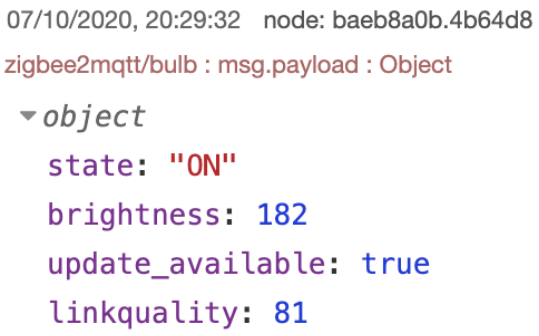

Figura 64. Mensaje a la salida del nodo json de la bombilla.

Teniendo el objeto msg podemos utilizarlo para obtener los valores del estado y el brillo de la bombilla con los nodos function “get state” y “get brightness” respectivamente (ver figura 62).

Figura 65. Mensaje a la salida del nodo function (get brightness).

Figura 66. Mensaje a la salida del nodo function (get state).

Tal como explicamos en el capítulo anterior, los valores a la salida de los nodos function “get state” y “get brightness” son los que mantienen la coherencia del estado en el switch y el dimmer virtual. Ambos son actuadores lógicos y se pueden modificar sus valores a través de los dashboards de control. Si movemos a estado On el switch virtual (ver figura 67), este envía un mensaje al nodo Bulb (mqtt out) para encender la bombilla (ver figura 68).

Figura 67. Switch virtual.

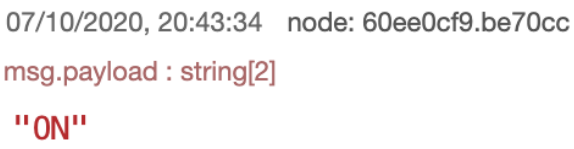

Figura 68. Mensaje a la salida del Switch virtual.

Por otra parte, si modificamos el valor del dimmer en el dashboard (ver figura 69), este envía un mensaje al nodo function “set brightness” para obtener el nuevo valor del brillo (ver figura 70).

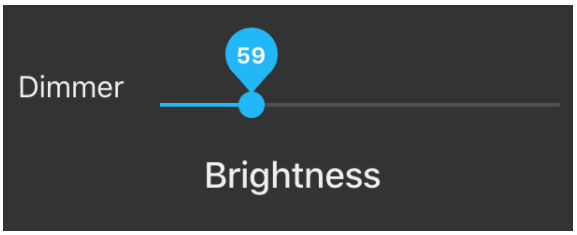

Figura 69. Dimmer virtual.

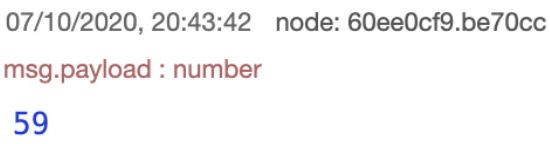

Figura 70. Mensaje a la salida del Dimmer virtual.

La función “set brightness” modifica el objeto msg para que se actualice el brillo de la bombilla al enviar el mensaje al nodo Bulb (mqtt out).

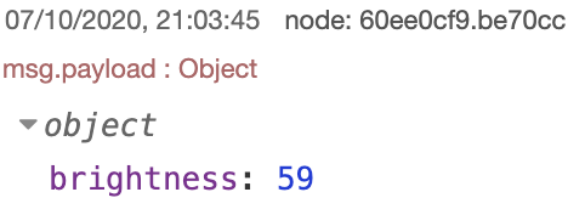

Figura 71. Mensaje a la salida del nodo function (set brightness).

Estos mensajes de los actuadores virtuales recorren la arquitectura desde el entorno cloud hasta la red Zigbee donde el último paso es el bridge Zigbee2MQTT como se muestra a continuación. En la figura 72 se muestra como se recibe primero el mensaje MQTT que

provoca que se publique en la red Zigbee el mensaje que enciende la bombilla y que seguidamente se publique un mensaje MQTT con el estado actualizado.

```
debug 2020-10-07 21:10:04: Received MQTT message on 'zigbee2mqtt/bulb/set'
with data 'ON'
debug 2020-10-07 21:10:04: Publishing 'set' 'state' to 'bulb'
info  2020-10-07 21:10:04: MQTT publish: topic 'zigbee2mqtt/bulb', payload
'{"state":"ON","brightness":161,"update_available":true,"linkquality":81}'
```

Figura 72. Logs de Zigbee2MQTT.

En la figura 73 se muestra cómo ocurre similar cuando se modifica el brillo de la bombilla.

```
debug 2020-10-07 21:06:19: Received MQTT message on 'zigbee2mqtt/bulb/set'
with data '{"brightness":59}'
debug 2020-10-07 21:06:19: Publishing 'set' 'brightness' to 'bulb'
info  2020-10-07 21:06:19: MQTT publish: topic 'zigbee2mqtt/bulb', payload
'{"state":"ON","brightness":59,"update_available":true,"linkquality":81}'
```

Figura 73. Logs de Zigbee2MQTT.

5.5 Dashboards de monitorización y control.

Este apartado está dedicado a mostrar el resultado final de los dashboards de monitorización y control que hemos creado para los usuarios de nuestra red domótica. Nuestro sistema cuenta con un menú de navegación para movernos entre los diferentes dashboards.

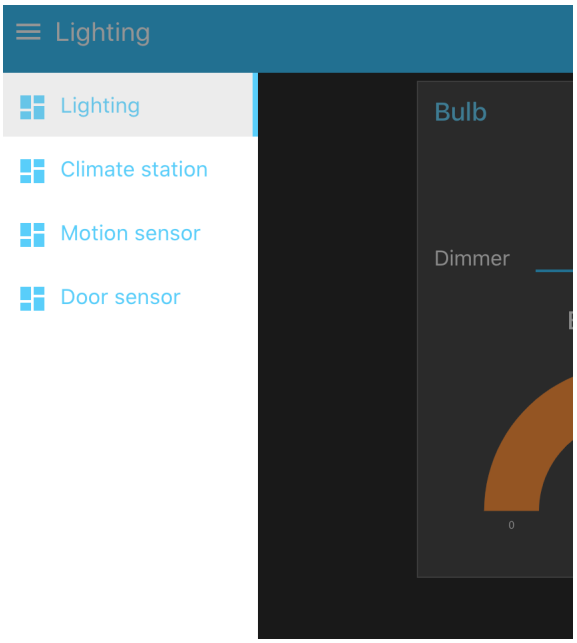

Figura 74. Menú de navegación de para los dashboards de la red domótica.

El dashboard de Lighting muestra el estado de la bombilla (ON/OFF), el gráfico para representar brillo y el gráfico de la batería del switch. Además, aquí se encuentran los controles de los actuadores virtuales del switch y el dimmer (ver figura 75).

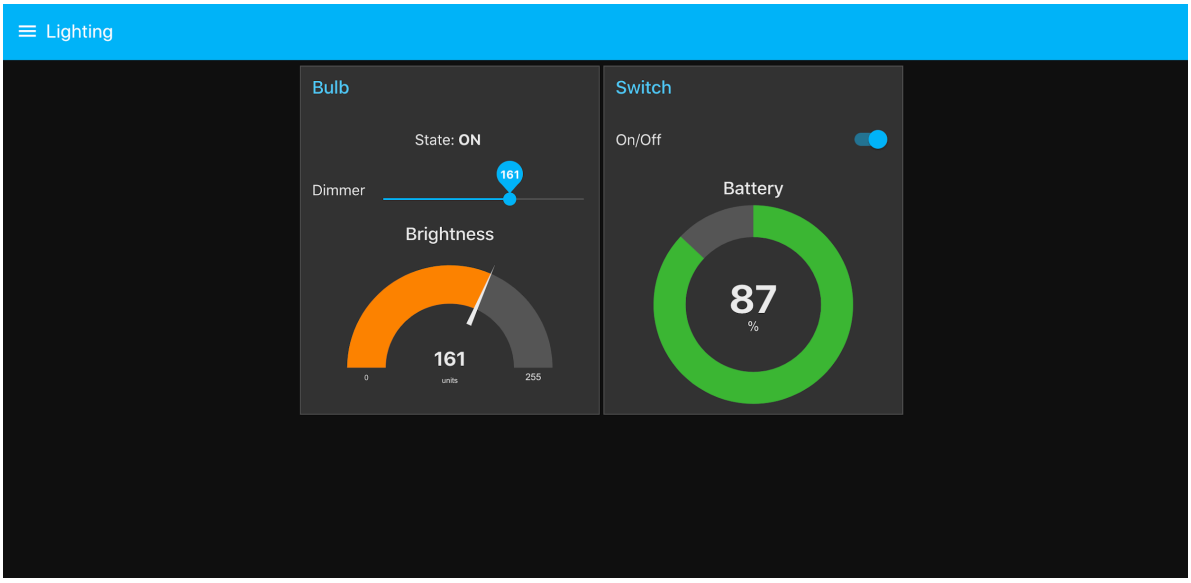

Figura 75. Dashboard de monitorización y control del alumbrado.

Climate station es el dashboard de monitorización que muestra los gráficos para las variables de temperatura y humedad y además el estado de la batería del sensor de temperatura y humedad (ver figura 76).

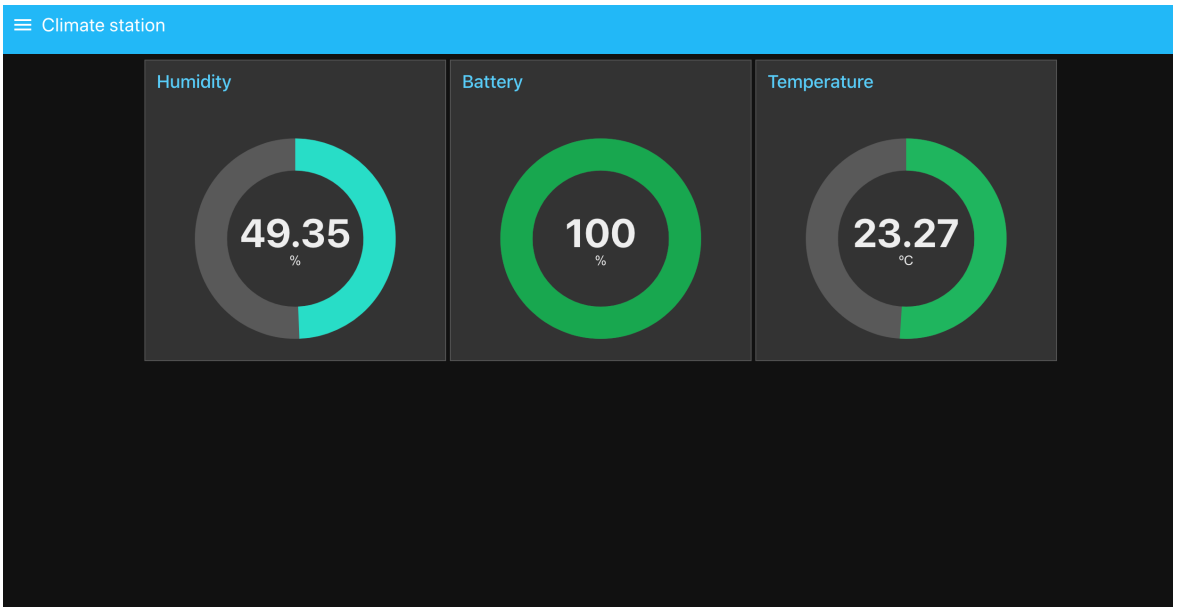

Figura 76. Dashboard de monitorización del clima.

El dashboard Motion sensor es para mostrar el estado de ocupación de la habitación (Occupied/Unoccupied) y el gráfico del estado de la batería del sensor de movimiento (ver figura 77).

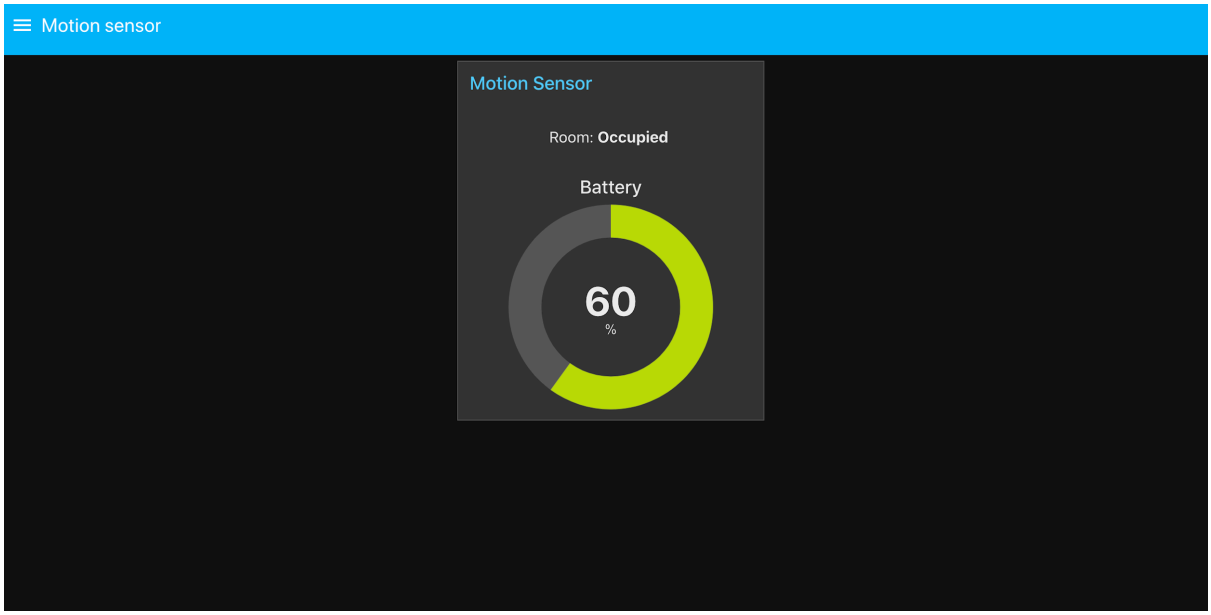

Figura 77. Dashboard de monitorización del sensor de movimiento.

Por último, en la figura 78 se ve el dashboard Door sensor que muestra de forma sencilla el estado de la batería del sensor de contacto (Good/Low) y el estado de la puerta (Open/Close)

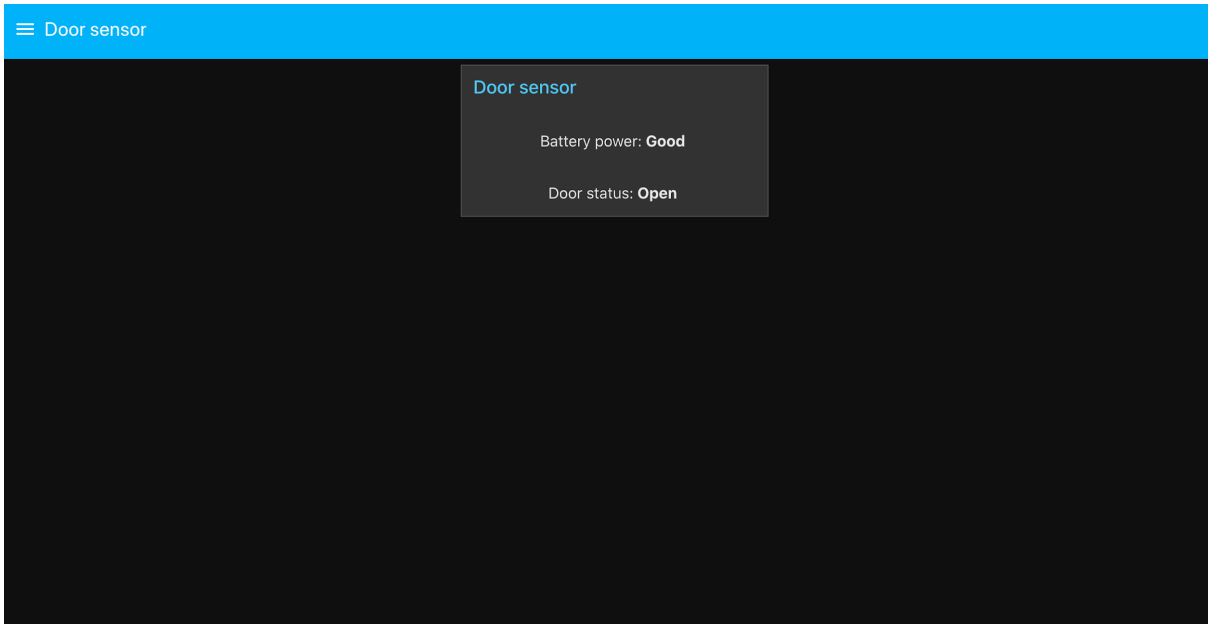

Figura 78. Dashboard de monitorización de la puerta.

CAPÍTULO 6. GESTIÓN DEL PROYECTO.

6.1 Planificación.

Para el desarrollo de este TFG se dividió el trabajo en diferentes fases y objetivos particulares, de modo que fuese viable evaluar la evolución en base a la consecución de hitos. El tiempo de trabajo se dividió en semanas con una planificación de una media de 20 horas semanales dedicadas al proyecto.

Fase 1: Definición general del problema a solucionar.

- Definir el marco de trabajo del problema que se quiere abordar.
- Definición de los objetivos que se quieren conseguir.
- Planificación del proyecto.

Fase 2: Investigación del entorno tecnológico.

- Investigación y evaluación de diferentes tecnologías y herramientas actuales en el campo en el que se plantea el problema.
- Definición de una arquitectura como solución para conseguir los objetivos definidos.

Fase 3: Costes y viabilidad.

- Evaluación de los costes y viabilidad del proyecto.
- Compra de los dispositivos necesarios para el desarrollo del proyecto.

Fase 4: Puesta en marcha.

- Instalación y configuración de software y herramientas que forman parte de la arquitectura propuesta.
- Desarrollo
- Pruebas y análisis.

Fase 5: Elaboración de la memoria.

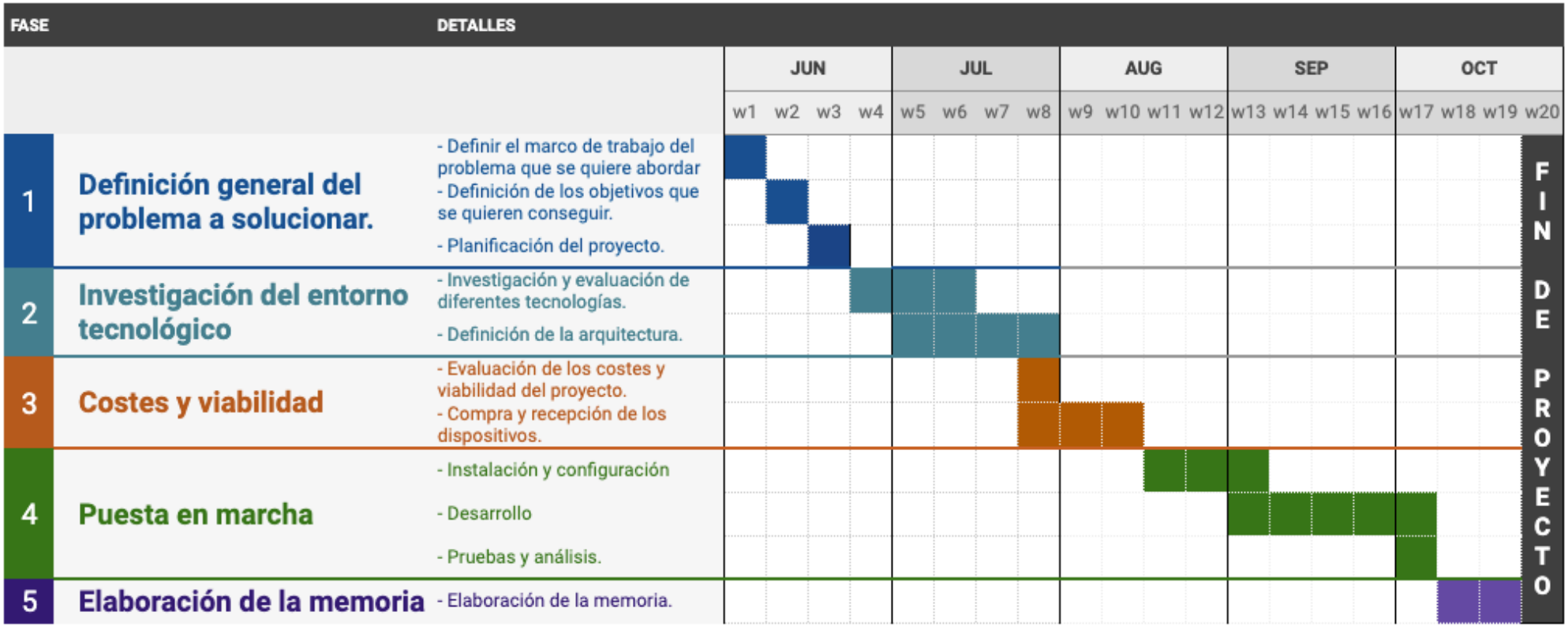

Figura 79. Diagrama de Gantt

6.2 Presupuesto.

El presupuesto para este proyecto lo dividimos en los gastos materiales y los gastos de personal. Los datos mostrados son solo un indicativo basado en la experiencia de implementar el proyecto durante el desarrollo del Trabajo de Fin de Grado y teniendo unos requisitos determinados.

6.2.1 Gastos materiales y digitales.

Estos gastos pueden variar en dependencia de los materiales que se quieran o necesiten para cumplir con los requisitos de un cliente.

Cantidad	**Dispositivo**	**Precio** (IVA incluido)
1	Raspberry Pi	39,95€
2	USB dongle cc2531	19,18€
1	CC Debugger	9,12€
1	CC2531 downloader cable	3,50€
1	LED1738G7	10,00€
1	E1525 / E1745	10,00€
1	E1743	6,00€

1	SNZB-02	7,18€
1	SNZB-04	7,18€
1	Dominio	5,00€ - 12,00€
		Total 124,11€

Tabla 1. Presupuestos materiales y digitales

Para el cálculo del coste del monitor, el teclado y el ratón se tuvo en cuenta la amortización. El coste es proporcional al tiempo de uso de los dispositivos respecto al tiempo de vida. Estimamos que de media estos dispositivos suelen tener una vida útil de 60 meses (5 años) y vimos que el tiempo de uso han sido de 5 meses. Entonces, el coste imputable de estos dispositivos durante el proyecto es de 8,9€.

Dispositivo	Precio	Coste imputable
Teclado y ratón	26,90€	2,24€
Monitor	80€	6,67€
		Total 8,9€

Tabla 2. Presupuestos materiales con amortización.

El gasto en cloud (AWS) es relativo al consumo de recursos y es algo que se puede ir controlando. En AWS, durante el primer año, se ofrece gratuitamente una capa de recursos limitados pero suficientes para poner en marcha este proyecto.

6.2.2 Gastos de personal.

Para el cálculo de los costes de realización de este proyecto en el aspecto de recursos humanos, se tuvo en cuenta el número de personas involucradas y las horas de trabajo dedicadas. El proyecto fue realizado por un estudiante de ingeniería bajo la supervisión de un ingeniero superior. El estudiante ha dedicado un total de 19 semanas trabajando una media de 20 horas semanales, lo que concluye en un total de 380 horas de trabajo. En las tareas de asesoramiento y revisión del proyecto, el ingeniero jefe dedicó un total de 10 horas.

	Horas dedicadas	Coste por hora (€)	Coste total (€)
Ingeniero superior Prof.ª María Calderón Pastor	10	40	400
Estudiante Javier González Fuentes	380	20	7600
			Total 8000€

Tabla 3. Tabla de gastos de personal.

El coste total del proyecto son 8133,01€.

6.3 Impacto socioeconómico.

En este trabajo se propone una arquitectura con gateway para una red Zigbee con el objetivo ofrecer a quién la utilice, la flexibilidad para poder monitorizar y controlar su entorno. Es un trabajo a partir del cual se pueden realizar proyectos ad hoc para satisfacer diferentes requisitos referentes a un entorno IoT. El proyecto es una alternativa más flexible a productos que ya existen en el mercado, aún así, el impacto socioeconómico que buscamos conseguir es el propio que viene generando este campo desde hace varios años.

Con este trabajo buscamos por una parte seguir generando conciencia sobre los beneficios del Internet de las Cosas y en particular de la domótica. Cuando somos capaces de monitorizar variables y controlar dispositivos del entorno que nos rodea somos capaces de conocer mejor cómo funcionamos y estudiar cómo mejorar procesos y hacerlos más eficientes. Desde una Smart Home hasta una Smart City, los beneficios que se derivan son

el impacto directo tanto en lo social brindando más comodidad y seguridad, como en el aspecto económico con algo tan simple como el ahorro energético.

CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO.

El desarrollo de este trabajo me ha permitido adentrarme y conocer un poco de un campo que me llamaba la atención desde hace un tiempo y que no había tenido la oportunidad de trabajar en ello. El Internet de las Cosas es muy interesante, brinda muchísimas oportunidades de desarrollo y creación y es un mercado que crece cada día más.

Durante el proceso de investigación tuve la oportunidad de conocer los fundamentos de varios protocolos de comunicación wireless y más en profundidad el protocolo Zigbee. Hasta ese momento algunos eran protocolos que desconocía. Para poder plantear una solución completa de una arquitectura domótica con gateway para una red Zigbee, tuve que leer mucha documentación y aprender de diferentes tecnologías y herramientas que son muy utilizadas en el mundo del desarrollo de entornos IoT.

La arquitectura que he propuesto es muy interesante porque ambos extremos de la infraestructura son libres para adaptarse a los requisitos que se propongan. Por un extremo, se pueden utilizar otros dispositivos diferentes a los planteados en el trabajo y por otro, se tiene la libertad de crear una lógica en Node-RED tan variada como se requiera. Esto hace que sea un proyecto fácil de extrapolar a diferentes entornos ya sea una casa, una fábrica, una farmacia o un restaurante.

Como trabajo futuro en este proyecto, y teniendo más tiempo de aprendizaje con Node-RED no sólo se podrían hacer mejoras y crear más funcionalidades en la lógica de la red domótica, sino que sería interesante la posibilidad de explorar la integración de otros protocolos wireless como los mencionados en capítulos anteriores. Esto abre aún más el espectro de dispositivos de diferentes fabricantes que se podrían utilizar para determinadas funciones dentro de la arquitectura que se ha propuesto, la cual aumentaría su flexibilidad de poder ser utilizada para resolver requisitos en más escenarios.

Otra línea en la que sería importante trabajar es respecto a la parte de la arquitectura en el entorno cloud. AWS brinda muchísimas opciones para variar el trabajo que hemos hecho. Si bien lo aprendido sobre AWS es bastante útil y nos sirvió para alcanzar los objetivos de este proyecto, es viable pensar en soluciones que se darían a un sistema que consuma muchos más recursos, que sea más grande y que necesite una infraestructura en alta disponibilidad.

Toda esta información aprendida me ha servido para crecer más profesionalmente y me ha abierto un poco más el espectro de trabajo al que me gustaría dedicar mi tiempo. Una

conclusión interesante que he experimentado es que mientras más “cosas” de nuestro entorno seamos capaces de monitorizar y controlar de manera segura, más ventajas tendremos en el día a día.

CAPÍTULO 8. REFERENCIAS BIBLIOGRÁFICAS.

[1] «Wikipedia (Internet de las Cosas),» 14 10 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Internet_of_things. [Último acceso: 15 10 2020].

[2] «Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.,» *BOE núm. 294, de 06/12/2018,* 25 06 2019.

[3] «Wikipedia (Red de sensores),» 5 10 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Red_de_sensores. [Último acceso: 10 20].

[4] «Wikipedia (MQTT),» 21 09 2020. [En línea]. Available: https://en.wikipedia.org/wiki/MQTT. [Último acceso: 10 2020].

[5] «HiveMQ (MQTT),» 19 01 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/. [Último acceso: 10 2020].

[6] L. Llamas, 17 04 2019. [En línea]. Available: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/. [Último acceso: 10 2020].

[7] «Wikipedia (Bluetooth Low Energy),» 5 10 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy. [Último acceso: 10 2020].

[8] «Wikipedia (Z-Wave),» 25 09 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Z-Wave. [Último acceso: 10 2020].

[9] «Wikipedia (6LoWPAN),» 14 10 2020. [En línea]. Available: https://en.wikipedia.org/wiki/6LoWPAN. [Último acceso: 10 2020].

[10] «Wikipedia (Zigbee),» 2 10 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Zigbee. [Último acceso: 10 2020].

[11] «Wikipedia (IEEE_802.15.4),» 2 07 2020. [En línea]. Available: https://en.wikipedia.org/wiki/IEEE_802.15.4. [Último acceso: 10 2020].

[12] «Insibe (Security in ZigBee communications),» Instituto Nacional de Ciberseguridad de España., 26 4 2016. [En línea]. Available: https://www.incibe-cert.es/en/blog/security-zigbee-communications. [Último acceso: 10 2020].

[13] «DIGI (PAN ID),» 15 5 2019. [En línea]. Available: https://www.digi.com/resources/documentation/Digidocs/90002002/Concepts/c_zb_pan_id.htm?TocPath=Zigbee%20networks%7CZigbee%20networking%20concepts%7CPAN%20ID%7C_____0. [Último acceso: 10 2020].

[14] «DIGI (Zigbee stack layers),» Digi International Inc, 15 05 2019. [En línea]. Available: https://www.digi.com/resources/documentation/Digidocs/90002002/Default.htm#Reference/r_zb_stack.htm. [Último acceso: 10 2020].

[15] « ZigBee Alliance (ZigBee Specification),» 5 08 2015. [En línea]. Available: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf. [Último acceso: 10 2020].

[16] A. Elahi y A. Gschwender, Wireless Sensor and Control Network, Pearson, 2009.

[17] K. Kanters, «Zigbee2MQTT,» [En línea]. Available: https://www.zigbee2mqtt.io/. [Último acceso: octubre 2020].

[18] K. Kanters, «Github (zigbee2mqtt),» [En línea]. Available: https://github.com/koenkk/zigbee2mqtt. [Último acceso: octubre 2020].

[19] «Eclipse Mosquitto™,» Eclipse Foundation, [En línea]. Available: https://mosquitto.org/. [Último acceso: 10 2020].

[20] «Amazon Web Services (Documentación),» [En línea]. Available: https://docs.aws.amazon.com/index.html. [Último acceso: 10 2020].

[21] «Node-RED,» [En línea]. Available: https://nodered.org/. [Último acceso: 10 2020].

[22] «Raspberry Pi 4,» Raspberry Pi Foundation, [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Último acceso: 10 2020].

[23] «Texas Instruments (CC Debugger User's Guide),» 4 2014. [En línea]. Available: https://www.ti.com/lit/ug/swru197h/swru197h.pdf. [Último acceso: 10 2020].

[24] K. Kanters, «Running Zigbee2MQTT,» [En línea]. Available: https://www.zigbee2mqtt.io/getting_started/running_zigbee2mqtt.html. [Último acceso: 8 2020].

[25] K. Kanters, «Configuration Zigbee2MQTT,» [En línea]. Available: https://www.zigbee2mqtt.io/information/configuration.html. [Último acceso: 8 2020].

[26] «Home Assistant,» [En línea]. Available: https://www.home-assistant.io/. [Último acceso: 10 2020].

[27] «AWS (MQTT 3.1.1),» [En línea]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html. [Último acceso: 10 2020].

[28] «mosquitto.conf Man Page,» Eclipse Foundation, [En línea]. Available: https://mosquitto.org/man/mosquitto-conf-5.html#. [Último acceso: 8 2020].

[29] T. Nordquist, «MQTT Explorer,» [En línea]. Available: http://mqtt-explorer.com/. [Último acceso: 8 2020].

[30] «Node-RED (Running on Amazon Web Services),» [En línea]. Available: https://nodered.org/docs/getting-started/aws. [Último acceso: 8 2020].

[31] «Github (acme.sh),» [En línea]. Available: https://github.com/acmesh-official/acme.sh. [Último acceso: 8 2020].

Summary.

The “Internet of Things” (IoT) is quite a popular topic today. The fact that dissimilar physical objects can be networked, collect and share data through the internet, opens a wide range of possibilities for development in many fields such as medicine, transport, industry, agriculture, energy, environment, etc. Smart devices are more and more varied and are developed in relation to different applications of use: organizational applications, industrial applications, infrastructure applications, military applications and consumer applications.

One of the best known and most used consumer applications of the IoT is the Smart Home. This derives from the introduction of smart devices in the broad concept of Home Automation (HA) or domotics that refers to the creation of home automation systems and processes. Current HA systems that turn a home into a “Smart Home” are based on a hub that is capable of reading data from a sensor network, controlling smart appliances and devices, and functioning as a gateway from the domotic network to the Internet so that users have remote control of processes through web and mobile applications.

HA is quite an important part of IoT and it is a business that is growing more and more. There are many manufacturers developing dissimilar types of automation systems using different technologies and software to create smart home products and connect them in a network.

Context and objectives.

We maintain the development framework of this project in the context of a network of sensors and actuators that use Zigbee as a communication protocol. For this network, we need to obtain and process the reading data from these sensors and commands from the actuator devices.

The objective of this project is the design and development of an architecture with a gateway for a Zigbee network. The architecture allows this network to be accessible through the Internet and different dashboards are developed for the monitoring and control of the home automation network. Relying on different communication protocols, hardware and software, we create an IoT environment where we are able to define and control the logic of the operation and interaction of the different IoT devices.

Background and state of the art.

Home Automation systems.

Home automation systems are developed to control devices that are commonly used in the home, whether they are electrical appliances, lighting, security systems, air conditioning or

entertainment systems. There is a huge variety capable of adapting to any need. Home automation allows you to take advantage of high-tech functionality and luxury, something that was not so viable in the past. As the development of technology continues to expand, so will the possibilities for home automation systems to make life easier and more enjoyable.

In recent years, much work and progress has been made in increasing the flexibility and compatibility of devices, in centralizing the control of all devices in one place, in security, in remote control of home functions, etc.

At a very high level, the architecture of a home automation system typically consists of a network of sensors and actuators (Wireless sensor and actuator networks, WSAN) that communicate with a hub through a communication protocol, usually wireless. This central is in charge of receiving, processing and sending data between the different devices of the home automation network and, functioning as a gateway, it connects to the visualization, monitoring and control applications for the users.

Hubs come in all shapes and forms, but typically they try to serve the purpose, to automate all of the smart home sensors and devices. Hubitat, Samsung SmartThings, HomeKit, OpenHab, Wink, among others, are examples of the hubs most used by users seeking the benefits of a Smart Home. They compete on points such as user friendly respect to the device, user experience in the application, remote access and above all the compatibility with several communication protocols that broadens the spectrum of use of different devices.

IoT communication protocols.

According to what has been investigated, there are several wireless communication protocols related to the implementation of a home automation system such as BLE, Z-Wave, WiFi, 6LoWPAN, Zigbee, etc. Deciding on which protocol is best for the IoT devices to be used in the development of an IoT network is a discussion that completely depends on the goals to be achieved. These are some of the most used protocols and, taking into account that they are very well documented protocols, I only comment on some of the characteristics to consider using Zigbee over other wireless protocols.

In the research carried out and according to the objectives of the project, Zigbee stands out as the main candidate to be the communication protocol of our IoT ecosystem. Zigbee, BLE and WiFi share transmission frequency and, unlike BLE and WiFi, Zigbee does not have operating system support, but that is not an inconvenience for our project. Compared to Z-Wave, Zigbee theoretically has no hop limits between a device and the network controller, while Z-Wave supports up to four. Unlike WiFi, Zigbee and BLE are quite similar from a

bandwidth and power consumption point of view, although Zigbee is more efficient on higher-range networks and energy efficiency was a major weight point.

Zigbee network.

In Zigbee, there are three different types of devices: End Device, Router, and Coordinator. A Zigbee network always has only one coordinator and can have multiple routers and end devices. Zigbee operates using a mesh network that allows them to increase their signal range by transmitting information from one device to another until it reaches the coordinator and also supports star and tree topologies

Zigbee Stack.

Zigbee is based on the Physical Layer (PHY) and the Media Access Control (MAC) sublayer defined in the IEEE 802.15.4 standard developed specifically for battery-operated devices. The network layer is responsible for the structure, routing, and security of the network. The Zigbee Application Layer (APL) consists of the APS sublayer, the ZDO, and the user or manufacturer-defined applications that give the device its specific function.

Zigbee Profiles.

Zigbee has three specifications Zigbee RF4CE, Zigbee PRO and the most recent Zigbee IP. This project focuses on the Zigbee PRO specification finalized in 2007 and last updated in 2015. Zigbee PRO has various application profiles that’s like a domain space of applications and devices that are related.

Home Automation is a public application profile with which we work in this project, and it defines a wide range of devices intended for use at home.

Proposed architecture.

To achieve the objectives of this work, various technologies and tools were used, both hardware and software. The idea was to simulate basic functions of a Smart Home and achieve remote and secure access to these functions to show the approach of this project.

Part of the proposed architecture consists of the deployment of a network of wireless sensors and actuators (WSAN) which are battery-operated and use Zigbee as the communication protocol. This network is made up of a humidity sensor, a contact sensor, a motion sensor, a light bulb and a switch. These are all end devices and it was not necessary to introduce any router devices into the Zigbee network. For the network coordinator we use a CC2531 USB dongle connected to a Raspberry Pi. This USB device was flashed with the ZigBee HA

standard coordinator firmware and was used as the coordinator of the proposed Zigbee network.

Another software we used was Zigbee2MQTT. This was the solution to be able to work with the zigbee data from the network through an application and to link everything to some cloud service to get remote access. Zigbee2MQTT acts as a bridge between Zigbee messages and MQTT messages. MQTT is an extremely lightweight publish-subscribe protocol that carries messages between devices and generally runs over a TCP/IP stack protocol. It is designed for connections to remote locations and is widely used in IoT. Zigbee2MQTT needs to connect as a client to an MQTT broker to send and receive messages. For this, we use Mosquitto, which is an open source MQTT message broker and it can handle all MQTT messages from clients that are connected to it.

As a cloud service provider we use AWS and 3 key services were used: IoT Core, IAM and EC2. La forma en que conectamos la parte local de la arquitectura con AWS fue a través de un puente configurado entre Mosquitto y AWS IoT Core.

Obviously, the application and the logic for the treatment of the devices through MQTT messages, it was necessary that they were deployed in a cloud service. For this we used Node-RED on a EC2 instance. Node-RED was the solution used to manage MQTT messages, to create the behavior flows of all the devices in the home automation network, and to create the monitoring and interaction dashboards for the Smart Home users. It is one of the most used tools in the development of IoT environments and is based on the development of flows to connect hardware devices, APIs and online services.

Configuration and development. Analysis of results and tests.

For the next 2 chapters we explained the necessary configuration of the different hardware and software that are part of the proposed architecture, the cloud configuration and the development of the logic of the home automation environment and we show how the platform works by analyzing end-to-end traffic within the proposed architecture.

Project management.

Planning.

The work was divided into different phases and particular objectives, so that it was feasible to evaluate the evolution based on the achievement of milestones. The working time was divided into weeks with a planning of an average of 20 hours per week dedicated to the project.

Budgets.

The budgets for this project are divided into material expenses and personnel expenses. The data shown is only an indication based on the experience of implementing the project during the development of the Final Degree Project and having certain requirements.

Material and digital costs may vary depending on the materials that are wanted or needed to meet a customer's requirements. Spending in the cloud (AWS) is relative to the consumption of resources and it is something that can be controlled. Our material and digital expenditure was a total of 124,11€.

For the calculation of the cost of the monitor, keyboard and mouse, depreciation was taken into account. The cost is proportional to the time of use of the devices with respect to the lifetime. We estimate that on average these devices usually have a useful life of 60 months (5 years) and we saw that the time of use has been 5 months. So, the chargeable cost of these devices during the project is 8,9€.

To calculate the costs of carrying out this project in the human resources aspect, the number of people involved, and the hours of work dedicated were taken into account. The project was carried out by an engineering student under the supervision of a senior engineer. The student has spent a total of 19 weeks working an average of 20 hours per week, which concludes in a total of 380 hours of work. Consulting and reviewing the project, the chief engineer spent a total of 10 hours. The cost of human resources was 8000€

The total cost of the project is 8133,01€.

Socio-economic impact.

In this work, an architecture with a gateway for a Zigbee network is proposed with the aim of offering whoever uses it the flexibility to be able to monitor and control their environment. It is a job from which ad hoc projects can be carried out to satisfy different requirements regarding an IoT environment. The project is a more flexible alternative to products that already exist in the market, even so, the socioeconomic impact we seek to achieve is the same that this field has been generating for several years.

With this work we seek, on the one hand, to continue generating awareness about the benefits of the Internet of Things and in particular of home automation. When we are able to monitor variables and control devices in the environment that surrounds us, we are able to better understand how we work and study how to improve processes and make them more efficient. From a Smart Home to a Smart City, the benefits that are derived are the direct impact both

on a social level, providing more comfort and security, and on the economic aspect with something as simple as energy saving.

Conclusions.

The development of this work has allowed me to enter and get to know a little of a field that has caught my attention for a while and that I had not had the opportunity to work on. The Internet of Things is very interesting, it offers many opportunities for development and creation and it is a market that grows every day.

As future work in this project, and having more learning time with Node-RED, not only we could improves and create more functionalities in the logic of the home automation network, but it would be interesting the possibility of exploring the integration of other wireless protocols such as those mentioned in previous chapters. This further opens the spectrum of devices from different manufacturers that could be used for certain functions within the architecture that has been proposed, which would increase its flexibility to be used to solve requirements in more scenarios.

Another line that would be important to work on is regarding the part of architecture in the cloud environment. AWS provides tons of options to vary the work we have done. Although what we learned about AWS is quite useful and helped us to achieve the objectives of this project, it is feasible to think of solutions that would be given to a system that consumes many more resources, is larger and needs an infrastructure in high availability.

All this information has helped me to grow more professionally and has opened up a little more the spectrum of work to which I would like to dedicate my time. An interesting conclusion that I have experienced is that the more “things” in our environment we are able to safely monitor and control, the more advantages we will have on a day-to-day basis.