

Trabajo Fin de Grado

Ingeniería de Telecomunicación

Pasarela sobre Raspeberry Pi entre intermediario MQTT y plataforma DBaaS

Autor: Alfredo Montilla Pérez

Tutor: Germán Madinabeitia Luque

Dep. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018



Trabajo Fin de Grado
Ingeniería de Telecomunicación

Pasarela sobre Raspeberry Pi entre intermediario MQTT y plataforma DBaaS

Autor:

Alfredo Montilla Pérez

Tutor:

Germán Madinabeitia Luque

Profesor colaborador

Dep. de Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Pasarela sobre Raspberry Pi entre intermediario MQTT y plataforma DBaaS

Autor: Alfredo Montilla Pérez

Tutor: Germán Madinabeitia Luque

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

El final de este aprendizaje está llegando a su fin y me gustaría agradecer el apoyo y la inversión que ha realizado mi familia en mí, haciendo posible el estudio de esta carrera en la Universidad de Sevilla, facilitándome todo lo posible el camino.

Por supuesto he de agradecer también la labor de los profesores de esta Institución, así como al departamento de Telemática, especialidad por la que opté y que tenía clara desde el principio. No puede faltar el agradecimiento a mi tutor Germán Madinabeitia Luque, por ofrecerme y guiarme en este proyecto que me ha proporcionado nuevos conocimientos y experiencias, y que me ha planteado nuevos retos que finalmente he logrado afrontar.

Así pues, gracias a todos los que me han acompañado en esta etapa, que ha sido tanto de aprendizaje y formación para el futuro profesional como de desarrollo personal.

Alfredo Montilla Pérez
Ingeniería de Telecomunicaciones
Sevilla, 2018

Resumen

El objetivo de este proyecto es entrar en contacto con el mundo del *Internet of Things (IoT)*, a través de las plataformas IoT de grandes compañías tecnológicas como Google, Amazon o Microsoft, y estudiar la viabilidad de utilizar un dispositivo de bajo coste como es la Raspberry Pi, para recopilar datos de un sensor de temperatura y transmitirlos a la nube; para lo que este dispositivo actuará como pasarela utilizando el protocolo de comunicación *Message Queuing Telemetry Transport (MQTT)*, del que se hará un profundo estudio sobre su estructura y funcionamiento así como de sus características.

Se detallarán los pasos necesarios para conseguir conectar un dispositivo con cada una de las plataformas, con el fin también de facilitar la tarea para futuros proyectos. Una vez recopilados y transmitidos los datos, se realizará una serie de configuraciones para enrutar las tramas enviadas hacia las bases de datos que ofrecen dichas plataformas, para conseguir almacenar y gestionar los datos.

Finalmente se propone un programa escrito en Python para conseguir aunar la funcionalidad de cada plataforma en un servicio que permita conectarse con cualquiera de ellas, según la elección del usuario, desde un mismo sitio haciendo transparente al cliente la heterogeneidad de las mismas.

Abstract

The aim of this project is to begin in the Internet of Things (IoT) field, through the IoT platforms owned by the most important technological companies like Google, Amazon or Microsoft, and study the feasibility about using a low cost device like Raspberry Pi, to recover temperature sensor data and send it to the Cloud. For that purpose, the device will run as a gateway using the Message Queuing Telemetry Transport (MQTT) protocol, which is going to be the object of a deep study about its structure, functionality and characteristics.

The necessary steps for connecting a device with any of the platforms will be detailed, also with the objective of make this task easier for future projects. Once the data has been recovered and transmitted, the next steps will be to configure some rules to route the data to the databases of the platforms for store and manage the data.

Finally a Python code is presented as a solution to join the functionality of each platform in one service that allows connecting with any of them, according to the client's choice, from a unique place making transparent the heterogeneity of the platforms.

Agradecimientos	ix
Resumen	x
Abstract	xi
Índice	xii
Índice de tablas	xiv
Índice de figuras	xv
1 Introducción	1
1.1 <i>Terminología</i>	2
1.1.1 Mensaje de aplicación	2
1.1.2 Cliente	2
1.1.3 Suscripción	2
1.1.4 Tema	2
1.1.5 Filtro por tema	2
1.1.6 Paquete de control MQTT	2
1.2 <i>Representación de los datos</i>	2
2 Funcionamiento del protocolo	3
2.1 <i>Calidades de servicio</i>	3
2.1.1 QoS 0: "At most once"	3
2.1.2 QoS 1: "At least once"	3
2.1.3 QoS 2: "Exactly once"	4
2.2 <i>Caracteres utilizados en los temas</i>	5
2.3 <i>Formato de los paquetes de control MQTT</i>	5
2.3.1 Cabecera fija	5
2.3.2 Cabecera variable	7
2.3.3 Carga útil	7
2.4 <i>Paquetes de control MQTT</i>	8
2.4.1 Connect: Cliente → Servidor	8
2.4.2 Connack: Servidor → Cliente	9
2.4.3 Publish: Cliente → Servidor o Servidor → Cliente	10
2.4.4 Puback: Cliente → Servidor o Servidor → Cliente	11
2.4.5 Pubrec: Cliente → Servidor o Servidor → Cliente	11
2.4.6 Pubrel: Cliente → Servidor o Servidor → Cliente	12
2.4.7 Pubcomp: Cliente → Servidor o Servidor → Cliente	12
2.4.8 Subscribe: Cliente → Servidor	12
2.4.9 Suback: Servidor → Cliente	13
2.4.10 Unsubscribe: Cliente → Servidor	13
2.4.11 Unsuback: Servidor → Cliente	13
2.4.12 Pingreq: Cliente → Servidor	14
2.4.13 Pingresp: Servidor → Cliente	14
2.4.14 Disconnect: Cliente → Servidor	14
2.5 <i>Comparativa versión del protocolo</i>	15

2.6	<i>Seguridad del protocolo</i>	16
3	Configuración previa Raspberry Pi	17
3.1	<i>Hardware y Software necesarios</i>	17
3.2	<i>Esquema cliente MQTT</i>	18
4	Servicios en la nube	21
4.1	<i>Importancia de las plataformas IoT</i>	22
4.2	<i>Definición y funcionalidades de las plataformas IoT</i>	22
4.3	<i>Proveedores utilizados</i>	23
4.3.1	Amazon IoT Core	23
4.3.2	Google IoT Core	33
4.3.3	Azure IoT Hub	43
4.3.4	Alternativa a las grandes plataformas	50
4.4	<i>Comparativa de las plataformas</i>	55
5	Servicio en conjunto	57
5.1	<i>Desarrollo del código servicio</i>	57
	Conclusiones	61
	Referencias	62
	Glosario	64
	Anexo A	65
	Anexo B	66
	Anexo C	69
	Anexo D	71

ÍNDICE DE TABLAS

Tabla 2-1. Formato de los paquetes de control MQTT.	5
Tabla 2-2. Tipos de paquetes de control MQTT.	6
Tabla 2-3. Valores de bandera para cada paquete de control.	6
Tabla 2-4. Códigos de respuesta para connect.	10
Tabla 2-5. Cambios de la versión MQTT 3.1.1 respecto a 3.1.	15
Tabla 4-1. Variable rc.	26
Tabla 4-2. Comparativa de las características de las plataformas.	55
Tabla 4-3. Comparativa de costes para las plataformas.	56

ÍNDICE DE FIGURAS

Figura 1-1. MQTT sobre modelo OSI.	1
Figura 2-1. Paso de mensajes para QoS 0.	3
Figura 2-2. Paso de mensajes para QoS 1.	4
Figura 2-3. Paso de mensajes para QoS 2.	4
Figura 3-1. Instalación Eclipse Paho MQTT.	17
Figura 3-2. Constructor Client().	18
Figura 3-3. Configuración tls.	18
Figura 3-4. Función connect().	19
Figura 3-5. Publicación de los datos.	19
Figura 3-6. Desconexión de la plataforma.	19
Figura 4-1. Liderazgo en el mercado [10]	21
Figura 4-2. Estructura plataforma IoT [12]	22
Figura 4-3. Conexión entre dispositivos, AWS IoT y servicios AWS.	23
Figura 4-4. Interfaz AWS IoT	24
Figura 4-5. Certificados AWS IoT Core.	24
Figura 4-6. Política Seguridad IoT Amazon.	25
Figura 4-7. Asociación entre certificado, política y dispositivo.	25
Figura 4-8. Definición funciones.	26
Figura 4-9. Fragmento main ().	27
Figura 4-10. Configuración tls y conexión.	27
Figura 4-11. Publicar mensaje con temperatura.	28
Figura 4-12. Publicar datos propios del usuario	29
Figura 4-13. Creación regla AWS.	29
Figura 4-14. Selección de la acción para la Regla.	30
Figura 4-15. Creación Tabla DynamoDB.	30
Figura 4-16. Configuración de la acción.	31
Figura 4-17. Formato de los datos enviados.	31
Figura 4-18. Datos del dispositivo en la Base de Datos de la nube.	32
Figura 4-20. Estructura plataforma Google.	33
Figura 4-21. Interfaz principal IoT Core.	34
Figura 4-22. Registro dispositivo.	34
Figura 4-23. Dispositivo añadido al registro.	35
Figura 4-24. Parámetros para conexión Google IoT Core.	35
Figura 4-25. Definición funciones Google.	36
Figura 4-26. Función para crear JWT.	36
Figura 4-27. Configuración cliente Google.	37

Figura 4-28. Subir datos de temperatura.	37
Figura 4-29. Subir datos del cliente.	37
Figura 4-30. Desconexión de Google IoT Core.	38
Figura 4-31. Tema Pub/Sub.	38
Figura 4-32. Suscripción Pub/Sub.	38
Figura 4-33. Exportar a BigQuery.	39
Figura 4-34. Tarea Dataflow.	39
Figura 4-35. Creación dataset en BigQuery.	40
Figura 4-36. Creación tabla en BigQuery.	40
Figura 4-37. Creación segmento en Storage.	40
Figura 4-38. Parámetros tarea Dataflow.	41
Figura 4-39. Información tarea Dataflow en ejecución.	41
Figura 4-40. Consulta SQL en BigQuery.	42
Figura 4-41. Visualización de los datos en Google.	42
Figura 4-42. Costes Cloud IoT Core.	42
Figura 4-43. Presentación Azure.	43
Figura 4-44. Algunos clientes de Azure.	43
Figura 4-45. Recurso IoT Hub.	44
Figura 4-46. Suscripción para el recurso IoT Hub.	44
Figura 4-47. Creación del dispositivo y tipo de autenticación.	45
Figura 4-48. Punto de conexión.	45
Figura 4-49. Ruta para los datos.	46
Figura 4-50. Parámetros para conexión con Microsoft.	46
Figura 4-51. Definición funciones para Microsoft.	47
Figura 4-52. Creación del token para Microsoft.	47
Figura 4-53. Inicialización cliente, configuración tls y conexión.	48
Figura 4-54. Selección de los datos a enviar.	48
Figura 4-55. Visualización datos Azure.	49
Figura 4-56. Costes para la plataforma Azure.	49
Figura 4-57. Interfaz principal del panel de control de Carriots.	50
Figura 4-58. Registro dispositivo en Carriots.	51
Figura 4-59. Apikey Carriots.	51
Figura 4-60. Clase CarriotsMqttClient().	52
Figura 4-61. Datos de temperatura en Carriots.	52
Figura 4-62. Datos del usuario en Carriots.	53
Figura 4-63. Visualización de los datos desde Carriots.	53
Figura 4-64. Costes para la plataforma Carriots.	54
Figura 5-1. Importación de los paquetes.	57
Figura 5-2. Mensajes iniciales.	58

Figura 5-3. Selección de plataforma por el cliente.	58
Figura 5-4. Transmisión datos temperatura a todas las plataformas.	59
Figura 5-5. Transmisión datos propios a todas las plataformas.	59
Figura 5-6. Recordatorio de las opciones disponibles.	60
Figura Anexo a-1. Interfaz principal del servicio.	65
Figura Anexo b-1. Interacción cliente Amazon.	66
Figura Anexo b-2. Interacción cliente Amazon depuración.	66
Figura Anexo b-3. Interacción cliente Carriots.	67
Figura Anexo b-4. Interacción cliente Azure.	67
Figura Anexo b-5. Interacción cliente Google.	68
Figura Anexo c-1. Notificación Amazon.	69
Figura Anexo c-2. Notificación Carriots.	69
Figura Anexo c-3. Notificación Google.	70

1 INTRODUCCIÓN

Lo único seguro en la industria de la tecnología es el cambio.

- Marc Benioff -

Mqtt es un protocolo de transporte de mensajes basado en el paradigma cliente-servidor. Se caracteriza por requerir poco ancho de banda, ser de libre utilización y tener un consumo muy bajo. Tanto el cliente como el servidor pueden publicar mensajes, o suscribirse a uno o varios temas para recibir las publicaciones relacionadas con el mismo. Estas características hacen de él un protocolo ideal para ser usado en el nuevo marco en vías de desarrollo conocido como Internet de las Cosas, derivado del inglés “Internet of Things”, en adelante IoT.

Este protocolo se basa en la pila de TCP/IP y algunas de sus funcionalidades son:

- La capacidad para ser implementado en hardware de dispositivos altamente limitados.
- Ser implementado en redes con un ancho de banda de alta latencia.
- El uso de paquetes para publicar o suscribirse a mensajes de aplicación.
- Transportar mensajes con independencia de la carga útil de los mismos.
- Permite la distribución de uno a muchos, fundamental en IoT.

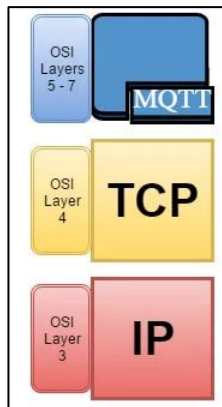


Figura 1-1. MQTT sobre modelo OSI.

Además implementa tres calidades de servicio, “At most once”, “At least once” y “Exactly once”, desarrolladas más adelante, para ajustarse al uso de la aplicación.

Originariamente MQTT fue inventado y desarrollado por la empresa multinacional IBM a finales de los 90 y su aplicación original era conectar sensores de oleoductos con satélites. A finales del 2014 se convierte en un estándar abierto de [OASIS](#), consorcio sin ánimo de lucro que impulsa el desarrollo y adaptación de estándares abiertos para la sociedad de la información global. Lo que facilita su uso para la comunidad de desarrolladores y favorece su crecimiento, convirtiéndose así en un candidato para el futuro de IoT. Finalmente en Junio de 2016 el estándar de OASIS es aprobado por la ISO (ISO/IEC 20922). [1][2][3]

1.1 Terminología

Esta sección recoge los términos más generales referentes al protocolo, que serán recurrentes a lo largo del documento.

1.1.1 Mensaje de aplicación

Hace referencia a los datos transportados por MQTT a través de la red para la aplicación. Estos mensajes llevan asociados un tema o “topic” y una calidad de servicio.

1.1.2 Cliente

Puede ser tanto un programa como un dispositivo que use MQTT y es el que siempre establece la conexión con el servidor. Algunas de sus capacidades son publicar y suscribirse a mensajes de aplicación o desconectarse del servidor.

1.1.3 Suscripción

Realizada por un cliente, indica sobre qué tema o topic tiene interés en recibir los mensajes de aplicación. Una suscripción va unida a una única sesión mientras que una sesión puede tener más de una suscripción.

1.1.4 Tema

Cadena de caracteres que cataloga el tipo de datos que se va a transmitir.

1.1.5 Filtro por tema

Expresión contenida en una suscripción para indicar interés en las publicaciones de uno o más temas.

1.1.6 Paquete de control MQTT

Paquete con información que se envía a través de la red. Hay 14 tipos definidos que se desarrollarán en detalle más adelante. Definen de forma explícita el comportamiento del protocolo.

1.2 Representación de los datos

Para representar los datos se hace uso del formato de codificación UTF-8 recogido en la [RFC 3629](#).

Es importante tener en cuenta que si un cliente o servidor recibe un paquete de control con una mala codificación UTF-8, debe cerrar la conexión. Al igual que no se debe incluir el carácter null (U+0000), pues en este caso el que lo reciba, cliente o servidor, deberá cerrar la conexión.

2 FUNCIONAMIENTO DEL PROTOCOLO

La ética cambia con la tecnología.

- Larry Niven -

En este capítulo se detallan las características del protocolo MQTT que determinan su correcto funcionamiento. Así pues se expone el formato de los paquetes de control, mencionados en el capítulo 1, los tipos de paquetes que conforman el protocolo y la calidad de servicio entre otras. [4]

2.1 Calidades de servicio

En esta sección se exponen las características de las tres calidades de servicio, en adelante QoS.

2.1.1 QoS 0: "At most once"

Con este tipo de calidad de servicio, el paquete publish, paquete que contiene el mensaje de aplicación, se envía como máximo una vez. Cuando se recibe el paquete no se envía confirmación y el reenvío no es posible con este nivel de servicio. Es posible que el mensaje no llegue al receptor.

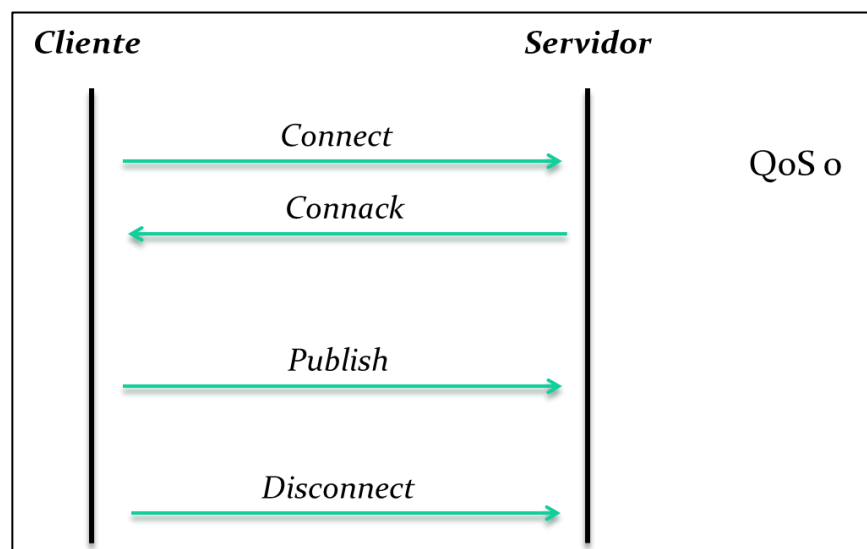


Figura 2-1. Paso de mensajes para QoS 0.

2.1.2 QoS 1: "At least once"

Este nivel de servicio asegura que el paquete publish llegue al menos una vez. Sin embargo es posible que se reciba un paquete duplicado.

El transmisor debe asignar un identificador de paquete cada vez que publique un nuevo mensaje de aplicación. El paquete publish en su cabecera fija debe incluir los parámetros QoS=1 y DUP=0, definidos en [el apartado 2.5.3](#). El transmisor puede enviar varios paquetes publish mientras espera la confirmación de los mismos.

El receptor debe responder mediante un paquete puback con el mismo identificador que el del paquete recibido.

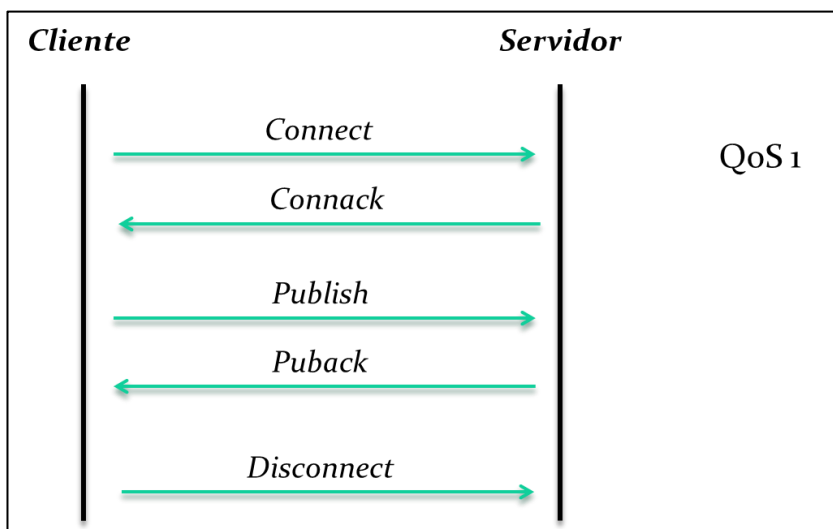


Figura 2-2. Paso de mensajes para QoS 1.

2.1.3 QoS 2: "Exactly once"

Este es el nivel más alto, se utiliza cuando no se quiere permitir ni la pérdida ni la duplicación de mensajes. Con esta calidad aseguramos que la entrega del paquete publish se realiza exactamente una sola vez.

El transmisor, al igual que antes, debe asignar un identificador de paquete para cada mensaje de aplicación que vaya a publicar. En este caso los parámetros de su cabecera fija serán QoS=2 y DUP=0.

El receptor entonces deberá responder mediante un paquete pubrec con el mismo identificador que el del paquete publish. Una vez recogido dicho paquete por el transmisor, éste deberá responder mediante el paquete pubrel, de nuevo con el mismo identificador que el publish original.

Una vez enviado el pubrel, el transmisor no debe reenviar el paquete publish, evitando así entregas duplicadas. Por último, el receptor deberá responder al pubrel mediante un paquete pubcomp con el mismo identificador.

En este punto el receptor deberá tratar cualquier otro paquete publish con el mismo identificador como una publicación nueva.

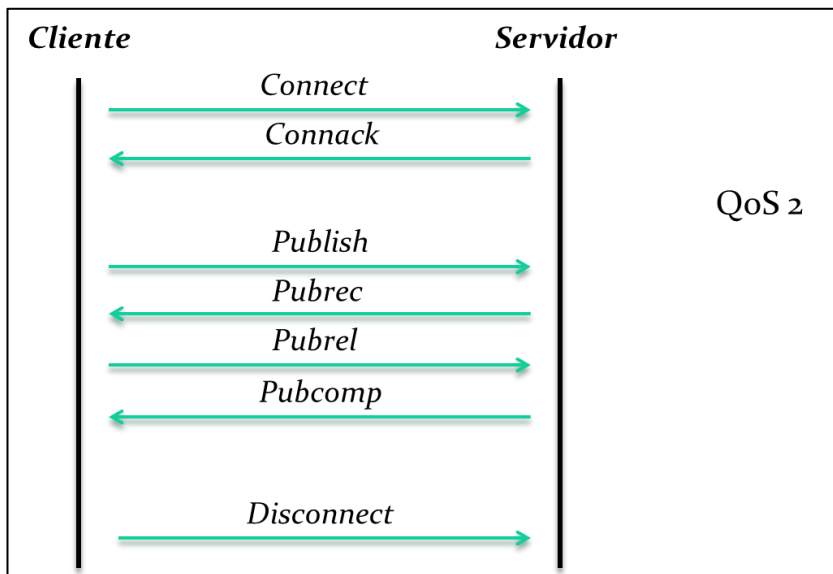


Figura 2-3. Paso de mensajes para QoS 2.

2.2 Caracteres utilizados en los temas

- El carácter ‘/’ se utiliza para separar los temas entre sí de forma jerárquica, por ejemplo “Casa/Salón/Sensor1”. Si un cliente se suscribe al topic anterior, recibirá los mensajes de aplicación que publique el Sensor1.
- El carácter ‘#’ es multinivel, por lo que permite suscribirnos a más de un tema a la vez. Así, si tenemos una suscripción como “Casa/PlantaBaja/Salón/#”, recibiremos mensajes publicados por:
 - “Casa/PlantaBaja/Salón/Sensor1”
 - “Casa/PlantaBaja/Salón/Sensor2”
 - “Casa/PlantaBaja/Salón/Sensor3/Sensor3.1”
- El carácter ‘+’ permite la suscripción en un mismo nivel, es decir si tenemos la suscripción “Casa/+” recibiremos mensajes de “/Casa/PlantaBaja” y “Casa/PlantaAlta”, pero no de “Casa/PlantaBaja/Salón”.

Hay que tener en cuenta que tanto los temas y los filtros de tema son sensibles a las mayúsculas y minúsculas, y el carácter espacio está permitido.

2.3 Formato de los paquetes de control MQTT

MQTT, como cualquier otro protocolo de red, funciona intercambiando una serie de mensajes, en este caso los paquetes de control, en un orden definido. En esta sección se explica el formato de estos paquetes.

Se dividen en tres partes:

Tabla 2-1 Formato de los paquetes de control MQTT

Cabecera fija	Presente en todos los paquetes de control MQTT
Cabecera variable	Presente en algunos de los paquetes de control
Carga útil	Presente en algunos de los paquetes de control

2.3.1 Cabecera fija

La cabecera fija se subdivide en tres apartados, repartidos en 2 octetos.

- En el primer octeto del bit 7 al 4: *Tipo de paquete de control MQTT.*
- Primer octeto del bit 3 al 0: *Banderas específicas para cada tipo de paquete de control.*
- En el segundo octeto: *Longitud restante.*

2.3.1.1 Tipos de paquetes de control MQTT

Este apartado de la cabecera fija especifica el tipo de paquete del que se trata. Como se menciona en la terminología, existen 14 tipos distintos representados por 4 bits sin signo. Estos valores se representan en la siguiente tabla.

Tabla 2-2 Tipos de paquetes de control MQTT

Nombre	Valor	Dirección del flujo de datos	Descripción
Reservado	0	Prohibido	Reservado
CONNECT	1	Cliente → Servidor	Cliente pide conexión con el Servidor
CONNACK	2	Servidor → Cliente	Confirmación para la conexión
PUBLISH	3	Cliente → Servidor Servidor → Cliente	Mensaje de publicación
PUBACK	4	Cliente → Servidor Servidor → Cliente	Confirmación para la publicación
PUBREC	5	Cliente → Servidor Servidor → Cliente	Publicación recibida
PUBREL	6	Cliente → Servidor Servidor → Cliente	Publicación lanzada
PUBCOMP	7	Cliente → Servidor Servidor → Cliente	Publicación completada
SUBSCRIBE	8	Cliente → Servidor	Petición para suscribirse del cliente
SUBACK	9	Servidor → Cliente	Confirmación de la suscripción
UNSUBSCRIBE	10	Cliente → Servidor	Petición para darse de baja
UNSUBACK	11	Servidor → Cliente	Confirmación petición de baja
PINGREQ	12	Cliente → Servidor	Petición PING
PINGRESP	13	Servidor → Cliente	Respuesta PING
DISCONNECT	14	Cliente → Servidor	Desconexión del cliente
Reservado	15	Prohibido	Reservado

2.3.1.2 Banderas para cada paquete de control

En el primer octeto, del bit 3 al 0 encontramos el campo “Flags” o banderas. Son específicas de cada tipo de paquete de control MQTT y sus valores están recogidos en la siguiente tabla.

Es importante que los valores de los campos reservados sean los especificados. Si se recibe una bandera errónea, el receptor debe cerrar la conexión.

Tabla 2-3 Valores de bandera para cada paquete de control

Paquete de control	Banderas	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reservado	0	0	0	0
CONNACK	Reservado	0	0	0	0
PUBLISH	Usado en MQTT 3.1.1	DUP	QoS	QoS	RETAIN

PUBACK	Reservado	0	0	0	0
PUBREC	Reservado	0	0	0	0
PUBREL	Reservado	0	0	1	0
PUBCOMP	Reservado	0	0	0	0
SUBSCRIBE	Reservado	0	0	1	0
SUBACK	Reservado	0	0	0	0
UNSUBSCRIBE	Reservado	0	0	1	0
UNSUBACK	Reservado	0	0	0	0
PINGREQ	Reservado	0	0	0	0
PINGRESP	Reservado	0	0	0	0
DISCONNECT	Reservado	0	0	0	0

Donde:

- DUP: Entrega duplicada de un paquete de control publish.
- QoS: Calidad del servicio definida para publish.
- RETAIN: Bandera de retención.

Como se observa, todos los paquetes exceptuando el paquete publish tienen valores reservados para un uso futuro. Los tres parámetros del paquete publish se especifican en el apartado 2.2.3, correspondiente al [paquete publish](#).

2.3.1.3 Longitud restante

En el segundo octeto de la cabecera fija se encuentra el campo *longitud restante*, indica el número de octetos restantes para el paquete actual, incluyendo los datos de la cabecera variable y la carga útil.

2.3.2 Cabecera variable

El contenido de este campo varía según el tipo del paquete de control, aunque la mayoría de estos paquetes tienen en común un campo denominado identificador de paquete que ocupa dos octetos.

Este identificador debe ser único para cada paquete que se envía, repitiéndose únicamente si el paquete se reenvía, lo cual dependerá de la calidad de servicio establecida. Una vez se confirma el paquete, el identificador vuelve a estar disponible.

El resto de contenidos de este campo se verá en detalle para cada tipo de paquete.

2.3.3 Carga útil

Este campo es obligatorio para los paquetes “connect”, “subscribe”, “suback” y “unsubscribe”. Para el paquete “publish” es opcional, el resto no contienen este campo.

2.4 Paquetes de control MQTT

En esta sección se exponen de forma detallada todos los tipos de paquetes de control y los campos que los componen.

2.4.1 Connect: Cliente → Servidor

Debe ser el primer paquete enviado desde el cliente hacia el servidor tras haberse establecido la conexión entre ambos. Un cliente sólo puede enviar un paquete connect, de lo contrario el servidor procesará ese segundo connect como una violación del protocolo y desconectará al cliente.

Como se ha visto anteriormente en la [tabla 2-1](#), estos paquetes se estructuran en tres campos, la cabecera fija, la cabecera variable y la carga útil.

2.4.1.1 Cabecera fija

En la cabecera fija se especifica en la primera sección del octeto1 el tipo de paquete de control, en este caso con valor 1, codificado como 0001. Y en la segunda sección encontramos el campo bandera, para el que obtenemos el valor “Reservado” codificado con 0000. Estos valores están recogidos en la [tabla 2-2](#) y [tabla 2-3](#) respectivamente.

2.4.1.2 Cabecera variable

En cuanto a la cabecera variable para “connect”, está formada por cuatro campos. En el siguiente orden: nombre del protocolo, versión del protocolo, banderas y “keep alive”.

1. Nombre del protocolo: Codificación en UTF-8 que representa el nombre del protocolo MQTT. Es importante destacar que un nombre erróneo del protocolo podría hacer que el servidor desconectara al cliente.
2. Versión del protocolo: Este campo representa la versión del protocolo que utiliza el cliente mediante 8 bits. Para este proyecto se usará la versión 3.1.1, cuyo valor correspondiente para este campo es 4 (0x04).

Si el servidor no soporta la versión del cliente, debe responder al connect mediante el código 0x01 y desconectar posteriormente al cliente.

3. Banderas: El campo banderas contiene una serie de parámetros que especifican el comportamiento de la conexión MQTT.

- 3.1. Clean session: Este bit se utiliza para controlar el tiempo de vida del estado de una sesión.

Si está configurado a 0, el servidor debe reanudar la comunicación con el cliente en función del estado de la sesión actual. Si no existiese dicha sesión, el servidor debe crear una nueva.

Si el bit está a 1, cliente y servidor deben descartar cualquier sesión previa y comenzar una nueva. Esta sesión tendrá la misma duración que la conexión de red.

- 3.2. Will flag: Si se configura a 1, el servidor debe almacenar un mensaje asociado a la conexión de red, que será publicado cuando se cierre la conexión. A no ser que se elimine dicho mensaje como consecuencia de un paquete disconnect.

- 3.3. Will QoS: Especifica mediante dos bits la calidad de servicio que se quiere utilizar al publicar el mensaje elegido. Si el bit *will flag* se establece a 0, *will QoS* debe valer 0 (0x00). Mientras que si se establece a 1, el valor *de will QoS* puede ser 0 (0x00), 1 (0x01) o 2 (0x02), correspondientes a las tres calidades de servicio disponibles.

- 3.4. Will retain: Si este bit se establece a 1, indica que el servidor debe almacenar el mensaje y el tema en el que se publica. Así, cada cliente que se suscriba a un tema que coincida con el del mensaje retenido, recibirá dicho mensaje inmediatamente después de la suscripción. El bit *will flag* debe estar a 1 para poder activar este bit, de lo contrario *will retain* debe valer 0.
- 3.5. User name: Si está a 1, un nombre de usuario debe aparecer en el campo carga útil.
- 3.6. Password: Si el bit vale 1, una contraseña deberá estar configurada en la carga útil. Si no hay nombre de usuario, no puede haber contraseña.
4. Keep alive: Intervalo de tiempo medido en segundos, expresado mediante 16 bits. Representa el tiempo máximo permitido entre el instante en el que termina de enviar un paquete de control, y el instante en el que empieza a enviar el siguiente.

El servidor cerrará la conexión si se excede 1.5 veces el tiempo establecido con “keep alive” sin recibir ningún paquete de control.

2.4.1.3 Carga útil

Por último en el campo carga útil o “payload” encontramos uno o varios parámetros, según los valores de las banderas de la cabecera variable anterior. Estos campos son “client identifier”, “will topic”, “will message”, “user name” y “password”.

1. Client identifier: Como se mencionó en la [sección 2.3.2](#), este identificador aparece en la mayoría de los paquetes de control MQTT. Identifica de forma única al cliente frente al servidor, ambos lo utilizan para identificar la sesión MQTT que comparten.

Este identificador debe estar presente en la carga útil de los paquetes “connect” y siempre en primer lugar, codificado mediante UTF-8.

Como caso excepcional, si un cliente envía un identificador con longitud cero, el servidor podría aceptarlo asignándole un identificador único, siempre que “clean session” esté a 1. De lo contrario el servidor responderá con el código 0x02 (identificador rechazado) y cerrará la conexión.

2. Will topic: Cadena codificada mediante UTF-8 siempre que la bandera “will flag” esté activada (bit a 1).
3. Will message: Mensaje al que se hacía referencia cuando se activa la bandera “will flag”. Define el mensaje de aplicación que será publicado.
4. User name: Cadena codificada en UTF-8 si la bandera “user name” está a 1.
5. Password: Contraseña que puede ocupar de 0 a 65535 octetos de datos binarios.

2.4.2 Connack: Servidor → Cliente

Paquete enviado por el servidor en respuesta de un paquete “connect”, debe ser el primer paquete que envíe el servidor.

2.4.2.1 Cabecera fija

Al igual que en el paquete anterior, en la cabecera fija se encuentra el tipo de paquete de control al que corresponde “connack”, en este caso el valor 2, su bandera correspondiente y la longitud de su cabecera variable. De nuevo estos valores se hallan en la [tabla 2-2](#) y [tabla 2-3](#).

2.4.2.2 Cabecera variable

La cabecera variable para connack está formada por dos campos:

1. Connect acknowledge flags: Se divide en dos:
 - a. Del bit 7 al 1 está reservado y debe valer 0.
 - b. El bit 0 es para *session present flag*.

Si el servidor acepta una conexión con *clean session* a 1, la bandera *session present* debe configurarse a 0 pues se generará una nueva sesión. De lo contrario si *clean session* vale 0, el valor de *session present* dependerá de si el servidor tiene o no guardado el estado de la sesión con el cliente.

2. Connect return code: Este campo representa los posibles códigos de respuesta para los paquetes connect. Dichos valores quedan recogidos en la tabla 2-4.

Tabla 2-4 Códigos de respuesta para “connect”.

Valor	Código de respuesta	Descripción
0	Conexión aceptada	Conexión aceptada
1	Conexión rechazada, versión del protocolo no aceptada	El servidor no soporta la versión del protocolo MQTT utilizada por el cliente
2	Conexión rechazada, identificador rechazado	El identificador de cliente es correcto según UTF-8, pero el servidor no lo acepta.
3	Conexión rechazada, Servidor no disponible	La conexión de red se ha establecido, pero el servicio MQTT no está disponible
4	Conexión rechazada, campo nombre de usuario o contraseña no válido	Los datos de nombre de usuario o contraseña están mal formados
5	Conexión rechazada, no autorizado	El cliente no tiene autorización para conectarse
6-255	-----	Reservado para el futuro

Si ninguno de estos códigos fuese aplicable en alguna situación, el servidor debe cerrar la conexión sin enviar el paquete connack.

2.4.2.3 Carga útil

Este tipo de paquetes no incluyen el campo de carga útil.

2.4.3 Publish: Cliente → Servidor o Servidor → Cliente

Los paquetes publish se encargan de transportar los mensajes de aplicación que pueden publicar tanto los clientes como los servidores.

2.4.3.1 Cabecera fija

La cabecera fija para estos paquetes incluye el tipo de paquete correspondiente, 3 en este caso. Del bit 3 al 0 del primer octeto encontramos 3 campos, DUP flag, QoS y Retain.

Por último, para el segundo octeto de la cabecera fija, se encuentra la longitud de la cabecera variable.

1. DUP flag: Si está a 0, indica que es la primera vez que se envía el paquete publish. Si se encuentra a 1 indica que el paquete debe ser reenviado al servidor, si el cliente es el que publica o viceversa. Hay que tener en cuenta que si la calidad de servicio está establecida en 0, DUP flag debe valer 0.
2. QoS: 2 bits que indican la calidad de servicio.
3. Retain: Si se configura con valor 1, el servidor debe almacenar el mensaje de aplicación y la calidad de servicio para que futuros suscriptores reciban dicha información, si sus suscripciones coinciden con el tema o topic correspondiente.

2.4.3.2 Cabecera variable

La cabecera variable contiene dos parámetros:

1. Topic name: Identifica al canal de información en el que se publicarán los datos.
2. Packet identifier: El identificador de paquete aparece solo si la calidad de servicio es 1 o 2.

2.4.3.3 Carga útil

Por último, el campo carga útil contiene el mensaje de aplicación que se publicará. El contenido y formato de dicho mensaje dependerá de la aplicación.

2.4.4 Puback: Cliente → Servidor o Servidor → Cliente

Paquete de respuesta a un publish con calidad de servicio igual a 1.

2.4.4.1 Cabecera fija

Para la cabecera fija se tiene el valor 4 para el tipo de paquete, la bandera reservada y el valor 2 para la longitud restante del paquete.

2.4.4.2 Cabecera variable

La cabecera variable se compone únicamente del identificador de paquete, ocupando 2 octetos.

2.4.4.3 Carga útil

Este tipo de paquetes no contiene el campo de carga útil.

2.4.5 Pubrec: Cliente → Servidor o Servidor → Cliente

Paquete de respuesta a un publish con calidad de servicio igual a 2.

2.4.5.1 Cabecera fija

En la cabecera fija se encuentra el valor 5 para el tipo de paquete, la bandera correspondiente y una longitud de 2 octetos restantes para el paquete.

2.4.5.2 Cabecera variable

Al igual que el anterior paquete, la cabecera variable se compone únicamente del identificador de paquete.

2.4.5.3 Carga útil

No contiene el campo de carga útil.

2.4.6 Pubrel: Cliente → Servidor o Servidor → Cliente

Paquete que se envía como respuesta a un paquete pubrec.

2.4.6.1 Cabecera fija

Cabecera fija similar al paquete anterior con la salvedad del tipo de paquete, en este caso 6. Los valores de la bandera del bit 3 al 0 también están reservados y deben valer 0010.

2.4.6.2 Cabecera variable

En cuanto a la cabecera variable, vuelve a estar compuesta únicamente por el identificador de paquete de 2 octetos.

2.4.6.3 Carga útil

No contiene el campo de carga útil.

2.4.7 Pubcomp: Cliente → Servidor o Servidor → Cliente

Paquete de respuesta a un paquete pubrel previo.

2.4.7.1 Cabecera fija

Similar en cuanto a los tres campos de la cabecera fija del paquete anterior, exceptuando el valor del tipo de paquete, 7, y el de la bandera. Ambos recogidos en las [tabla 2-2](#) y [tabla 2-3](#).

2.4.7.2 Cabecera variable

En la cabecera variable se encuentra el identificador de paquete.

2.4.7.3 Carga útil

No tiene carga útil

2.4.8 Subscribe: Cliente → Servidor

El tipo de paquetes subscribe se envía desde el cliente hacia el servidor para crear una o más suscripciones. Cada suscripción registra el interés del cliente en uno o más temas.

Así, cuando se producen mensajes de aplicación, el servidor en función de las suscripciones de los clientes, reenvía dichos mensajes, mediante publish, relacionados con un tema a aquellos clientes suscritos al mismo.

2.4.8.1 Cabecera fija

Para la cabecera fija se tiene en el primer octeto el tipo de paquete 8 y el valor reservado de bandera 0010. Y una longitud restante de 2 octetos.

2.4.8.2 Cabecera variable

La cabecera variable está formada por el identificador de paquete.

2.4.8.3 Carga útil

La carga útil de un paquete subscribe consiste en una lista con los temas o “topic” a los que se desea suscribirse, codificados mediante UTF-8. Cada tema que se especifique debe ir acompañado de la calidad de servicio máxima que puede recibir el cliente.

Este campo debe contener como mínimo una pareja de valores tema/calidad de servicio, de lo contrario se

tratará al paquete como una violación del protocolo.

2.4.9 Suback: Servidor → Cliente

Cuando el servidor recibe un paquete subscribe de un cliente, debe responder mediante un paquete suback. Este paquete tiene el mismo identificador que el del paquete subscribe.

Para cada par de valores tema / QoS debe haber un código de respuesta en el que se muestre la calidad del servicio máxima concedida por el servidor para esa suscripción. Este valor podría ser menor que el solicitado por el cliente.

2.4.9.1 Cabecera fija

Similar a los paquetes anteriores, cambiando el valor al tipo 9 y su bandera correspondiente.

2.4.9.2 Cabecera variable

Formada por el identificador de paquete.

2.4.9.3 Carga útil

En la carga útil se encuentra una lista con los códigos de respuesta,

0x00 - Correcto – Calidad de servicio máxima 0

0x01 - Correcto – Calidad de servicio máxima 1

0x02 - Correcto – Calidad de servicio máxima 2

0x80 – Fallo

Códigos distintos a los especificados están reservados y no se deben usar.

2.4.10 Unsubscribe: Cliente → Servidor

Paquete enviado por el cliente al servidor para darse de baja en un tema.

2.4.10.1 Cabecera fija

Formato similar a los paquetes anteriores.

2.4.10.2 Cabecera variable

Similar a paquetes anteriores.

2.4.10.3 Carga útil

Al contrario que en los paquetes subscribe, en el campo carga útil se encuentra una lista de los temas a los que el cliente le gustaría darse de baja. Debe haber al menos un valor en la lista, de lo contrario se tratará como una violación del protocolo.

2.4.11 Unsuback: Servidor → Cliente

El servidor responde a los paquetes unsubscribe con un unsuback.

2.4.11.1 Cabecera fija

La cabecera fija mantiene la estructura de paquetes anteriores.

2.4.11.2 Cabecera variable

Este paquete tiene el mismo identificador que el del paquete unsubscribe al que responde, contenido en el campo de cabecera variable.

2.4.11.3 Carga útil

No aplica en estos paquetes

2.4.12 Pinreq: Cliente → Servidor

Este tipo de paquetes se puede utilizar para indicarle al servidor que el cliente sigue activo, si no se están enviando otros tipos de paquetes de control. Otra utilidad es confirmar que el servidor sigue activo así como la conexión de red.

2.4.12.1 Cabecera fija

Cabecera fija similar a la estructura de los paquetes anteriores, identificador de tipo con valor 12 y bandera reservada.

2.4.12.2 Cabecera variable

No aplica en estos paquetes.

2.4.12.3 Carga útil

No aplica.

2.4.13 Pinresp: Servidor → Cliente

Paquete en respuesta del anterior, indica al cliente que el servidor sigue activo.

2.4.13.1 Cabecera fija

Cabecera fija con tipo de paquete 13 y bandera reservada en el primer octeto. En el segundo octeto se encuentra la longitud restante con valor 0.

2.4.13.2 Cabecera variable

Como se deduce del campo anterior, al tener una longitud restante 0, este campo no aplica para estos paquetes.

2.4.13.3 Carga útil

No aparece en los paquetes pinresp.

2.4.14 Disconnect: Cliente → Servidor

Este es el último paquete de control enviado desde el cliente hacia el servidor. Indica que el cliente se va a desconectar.

2.4.14.1 Cabecera fija

En la cabecera fija el valor del tipo de paquete corresponde a 14, valores de la bandera reservados y longitud restante 0.

2.4.14.2 Cabecera variable

No forma parte de estos paquetes.

2.4.14.3 Carga útil

Tampoco aplica para disconnect.

Tras enviar el paquete disconnect, el cliente debe cerrar la conexión y no enviar más paquetes de control por dicha conexión. Si el cliente no cerrase la conexión, el servidor debería hacerlo.

2.5 Comparativa versión del protocolo

Tanto para la documentación sobre el protocolo MQTT como para el código desarrollado en Python para los clientes, se ha utilizado la versión del protocolo 3.1.1. A continuación se muestra una tabla con las principales diferencias respecto a su antecesor, la versión 3.1.

Tabla 2-5. Cambios de la versión MQTT 3.1.1 respecto a 3.1. [5]

Cambios	Descripción	Referencia en documentación de la versión 3.1.1
1	Las cadenas deben verificarse para que sean válidas según UTF-8	1.5.3 UTF-8 cadenas codificadas
2	Cerrar la conexión si la cadena transmitida contiene NULL	1.5.3 UTF-8 cadenas codificadas
3	El nombre del protocolo ha cambiado a MQTT	3.1.2.1 Nombre del protocolo
4	El valor del <i>campo nivel del protocolo</i> cambia a 0x04	3.1.2.2 Nivel del protocolo
5	El id del cliente puede contener más de 23 octetos codificados	3.1.3.1 Identificador de cliente
6	El id del cliente puede tener 0 octetos de longitud	3.1.3.1 Identificador de cliente
7	El id del cliente debe mantener formato UTF-8	3.1.3.1 Identificador de cliente
8	El id del cliente puede contener únicamente caracteres alfanuméricos	3.1.3.1 Identificador de cliente
9	El paquete Connack tiene una nueva bandera, "sesión present"	3.2.2.2 Session Present
10	DUP no puede estar activado para QoS=0	3.3.1.1 DUP
11	Suback puede indicar fallos	3.9.3 Suback Payload
12	Los clientes pueden enviar paquetes de control adicionales inmediatamente después de enviar un paquete connect	3.1.4 Respuesta

2.6 Seguridad del protocolo

Finalizando con las especificaciones que definen el funcionamiento del protocolo MQTT, se cierra el capítulo mencionando brevemente algunos aspectos de seguridad. Hay que recordar en este término que MQTT es un protocolo de transporte y su función es la transmisión de mensajes, recae por tanto la responsabilidad de adecuar medidas de seguridad en el implementador del mismo.

Frente a las posibles amenazas como el robo de los datos guardados en los dispositivos clientes o servidores de forma física, ataques de denegación de servicio o comunicaciones interceptadas, se proponen algunas soluciones. Entre otras, la autenticación y autorización de usuarios y dispositivos o comprobar la integridad de los paquetes de control y los datos que portan.

Normalmente se utiliza TLS para cubrir algunas de estas amenazas, así el servidor puede autenticar a los clientes mediante los certificados SSL enviados por los mismos. El protocolo MQTT no proporciona mecanismos para que los clientes autentiquen al servidor, por lo que también se utilizan estos certificados entre servidores y clientes.

Si alguno de estos certificados de cliente o servidor se pierde o se ve comprometido, deberían ser revocados utilizando para ello CRLs u OCSP, recogidos en las RFC [5280](#) y [6960](#) respectivamente.

3 CONFIGURACIÓN PREVIA RASPBERRY PI

El mayor riesgo es no correr ningún riesgo.

- Mark Zuckerberg -

En este capítulo se detallan los pasos seguidos para configurar la pasarela, Raspberry Pi 3 Modelo B, desde cero. Se instalará un sistema operativo y un cliente MQTT del que se hará uso en el siguiente capítulo, para conectar este dispositivo con las plataformas IoT y enviar los datos de temperatura que se obtendrán de un sensor.

3.1 Hardware y Software necesarios

Tras adquirir el dispositivo Raspberry Pi 3 Modelo B se necesita una tarjeta microSD en la que se instalará el sistema operativo. Entre los distintos sistemas compatibles se opta por instalar Raspbian pues es el sistema operativo oficial y contiene software preinstalado orientado a la programación, como los editores Gedit o Python Idle.

Para dicha instalación se hace uso del programa Etcher que facilita la tarea de instalar la imagen del Sistema Operativo, descargada de la página oficial de Raspberry, en la microSD. Este programa es compatible con Windows, Linux y Mac. Dicha imagen tiene un tamaño aproximado de 4.6 Gb por lo que se recomienda una tarjeta microSD de al menos 8Gb y clase 6, lo que implica escritura a 6Mb/seg. [6][7]

El objetivo de esta pasarela es recopilar y transmitir datos hacia las plataformas IoT mediante el protocolo MQTT. Estos datos a enviar serán los valores de la temperatura de la CPU de la Raspberry Pi o texto establecido por el usuario final de forma dinámica.

Para obtener los valores de la temperatura se utiliza un sensor que trae incorporado el modelo Raspberry Pi 3. Este sensor escribe de forma continua los datos en el fichero `temp` ubicado en la ruta `sys/class/thermal/thermal_zone0/temp`. Por tanto para acceder a los valores de temperatura se accede a este fichero y se guardan los valores en una variable.

Queda pendiente la instalación del cliente MQTT para conectarse y transmitir estos datos a las plataformas en la nube. Consiste simplemente en abrir un terminal en el dispositivo e introducir el siguiente comando:

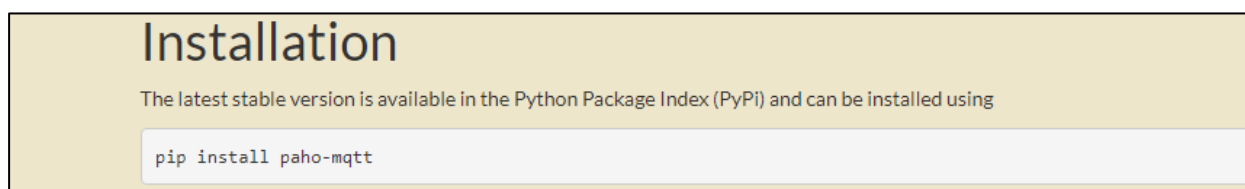


Figura 3-1. Instalación Eclipse Paho MQTT.

3.2 Esquema cliente MQTT

Para implementar este cliente será necesario revisar previamente la documentación desde la página oficial de Eclipse Paho. Los clientes para las plataformas se desarrollarán en el lenguaje de programación Python.

El desarrollo del cliente será visto en detalle en el capítulo 4 con los casos de usos concretos, pero la estructura que mantienen es la siguiente. [8]

1. Se comienza creando una instancia del cliente mediante el constructor `Client()`.

```
Client()
Client(client_id="", clean_session=True, userdata=None, protocol=MQTTv311, transport="tcp")
```

Figura 3-2. Constructor `Client()`.

Donde:

- `Client_id`: Representa el identificador único del cliente para la plataforma.
- `Clean_session`: Determina si se guarda la información del cliente al desconectarse o no.
- `Userdata`: Datos pasados del usuario.
- `Protocol`: Define qué protocolo utilizar en la conexión MQTTv3.1 o MQTTv3.1.1
- `Transport`: Para enviar MQTT a través de WebSockets.

2. Configuración de encriptación y autenticación mediante `tls_set()`.

```
tls_set()
tls_set(ca_certs=None, certfile=None, keyfile=None, cert_reqs=ssl.CERT_REQUIRED,
        tls_version=ssl.PROTOCOL_TLS, ciphers=None)
```

Figura 3-3. Configuración `tls`.

Donde:

- `Ca_certs`: Debe especificar la ruta hasta el certificado CA (Certificate Authority), certificado que ha sido emitido por una entidad certificadora. [9]
- `Certfile`: Ruta hasta el certificado de tipo PEM.
- `Keyfile`: Fichero con la clave privada.
- `Cert_reqs`: Define los certificados requeridos que el cliente impone a la plataforma.
- `Tls_version`: Especifica la versión del protocolo SSL/TLS a utilizar.
- `Ciphers`: Especifica tipos de cifrado permitidos, "None" para utilizar los establecidos por defecto.

3. Se definen los parámetros concretos de cada plataforma para establecer la conexión mediante Connect().

```
connect()
connect(host, port=1883, keepalive=60, bind_address="")
```

Figura 3-4. Función connect().

Parámetros:

- Host: Nombre o dirección IP de la plataforma.
- Port: Puerto del servidor, por defecto 1883 (no seguro).
- Keepalive: Periodo máximo en segundos permitido entre comunicaciones.
 - ❖ Referente al campo *keep alive* de la cabecera variable del tipo de paquete connect.
- Bind_address: IP de una interfaz local para vincular al cliente, suponiendo múltiples interfaces.

4. Se publican los datos mediante publish().

```
publish()
publish(topic, payload=None, qos=0, retain=False)
```

Figura 3-5. Publicación de los datos.

Con:

- Topic: Tema en el que publicar el mensaje.
 - ❖ Este topic, en relación con la documentación de MQTT, es el campo *topic name* de la cabecera variable de los paquetes publish.
- Payload: Los datos del mensaje en sí.
 - ❖ Hace referencia al campo “carga útil” de los paquetes publish.
- Qos: Nivel de calidad de servicio.
 - ❖ Equivale al campo QoS de la cabecera fija.
- Retain: Si se actualiza a “True” el mensaje se almacenará en el servidor, asociado al topic utilizado, para futuros suscriptores.
 - ❖ Este parámetro se corresponde con “retain”, también de la cabecera fija.

5. Desconexión de la plataforma con disconnect().

```
disconnect()
disconnect()
```

Figura 3-6. Desconexión de la plataforma.

4 SERVICIOS EN LA NUBE

El espíritu humano debe prevalecer sobre la tecnología.

- Albert Einstein -

En este capítulo se define la importancia de las plataformas en la nube, se exponen las características y los pasos seguidos para conseguir establecer conexión entre nuestro dispositivo y la nube. Así como la utilización de bases de datos para almacenar los datos enviados a las mismas.

Para el proyecto se ha realizado una selección de las plataformas a utilizar en base a las soluciones más adoptadas y avanzadas en el mercado, como pueden ser el servicio de Amazon llamado “AWS IoT Core”, “Azure” de Microsoft o “Google Cloud IoT” de Google. El objetivo de esta selección es ofrecer al cliente la posibilidad de elegir entre los mejores servicios posibles.

La siguiente figura muestra una división del mercado según Gartner (Abril 2018), en la que se puede observar como destacan las tres plataformas recién mencionadas sobre el resto.

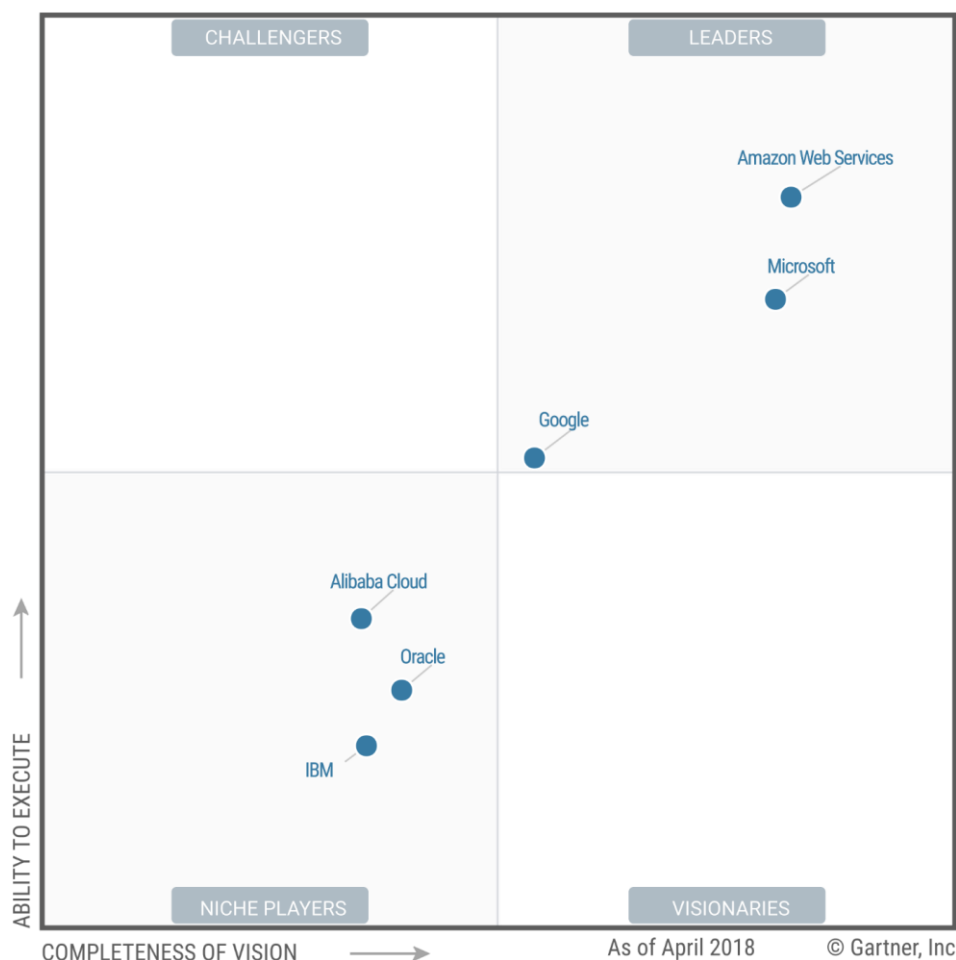


Figura 4-1. Liderazgo en el mercado [10]

4.1 Importancia de las plataformas IoT

Antes de desarrollar la implementación de las plataformas utilizadas, se expone la importancia que están adquiriendo las mismas en el mercado. Según [Gartner](#), empresa consultora y de investigación de las tecnologías de información con sede en Stamford, Connecticut (Estados Unidos), en 2020 habrá alrededor de 20.415 millones de dispositivos conectados en todo el mundo. Esto genera el desafío de cómo gestionar el flujo de los datos así como el almacenamiento y análisis de los mismos, donde entraría en juego otro de los grandes campos actuales, el Big Data.

Se convierten así estas plataformas en una herramienta indispensable para la gestión de este volumen de datos cada vez mayor. La empresa [Berg Insight](#), dedicada a la investigación de mercado M2M/IoT con sede en Suecia, afirma que estas plataformas IoT generarán un volumen de negocio alrededor de los 3.000 millones de euros en 2021, frente a los 610 millones generados en 2015. [11]

4.2 Definición y funcionalidades de las plataformas IoT

Las plataformas IoT son un servicio software basado en la nube que nos permite recoger todos los datos generados por nuestros sensores, actuadores u otros dispositivos vía internet, y a través de una serie de reglas pasar a almacenarlos en una base de datos o incluso analizarlos en tiempo real para tomar decisiones más rápidas.

Aunque cada proveedor desarrolla su plataforma de manera independiente suelen mantener una estructura similar, reflejada en la siguiente figura.

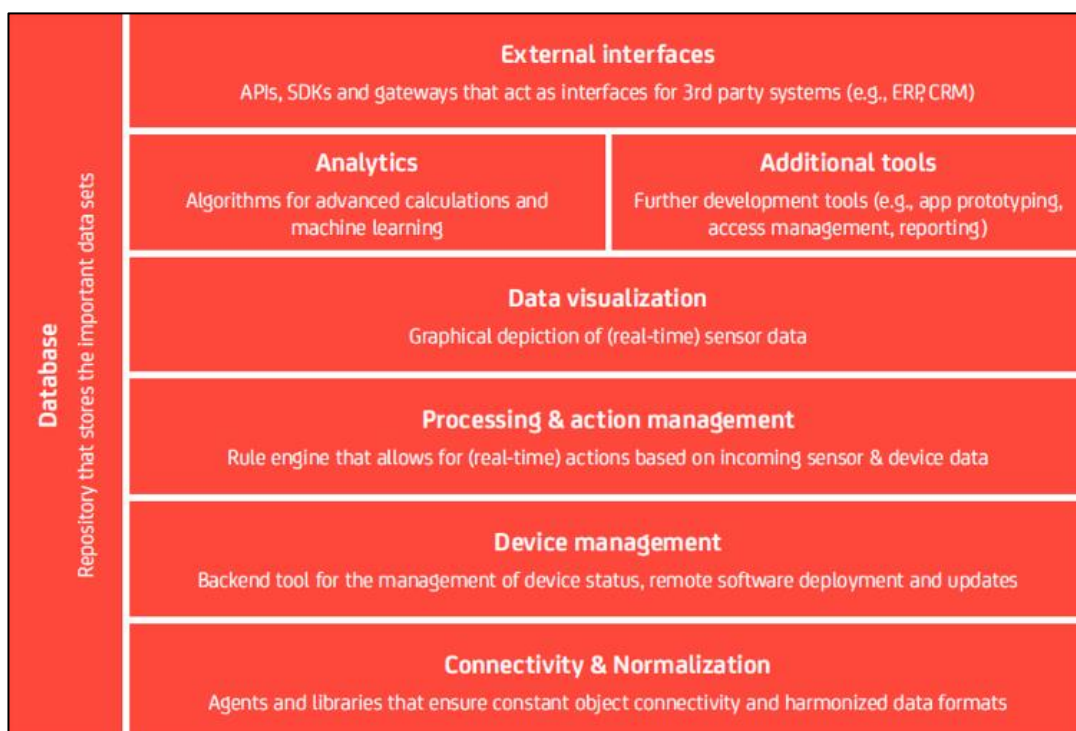


Figura 4-2. Estructura plataforma IoT [12]

Donde en primer lugar es necesario realizar la conexión del dispositivo con la plataforma, y para ello es necesario registrarlo previamente en la misma. Una vez conectado el dispositivo y comprobado que se están recibiendo los datos en la plataforma, es necesario crear una serie de reglas que nos permiten conectar lo que se denomina, normalmente en la mayoría de plataformas, centro de IoT o IoT Core con los servicios de almacenamiento y análisis de datos así como su visualización.

4.3 Proveedores utilizados

En esta sección se detallan las características de los proveedores de servicios en la nube para IoT utilizados para este proyecto, así como los pasos necesarios para conectar nuestro dispositivo, Raspberry Pi, con la nube y proceder con el envío y almacenamiento de datos a través del protocolo MQTT.

4.3.1 Amazon IoT Core

Amazon presenta su solución como AWS IoT Core, es una plataforma en la nube administrada que permite a los dispositivos conectados interactuar con facilidad y seguridad con las aplicaciones en la nube y otros dispositivos.

Afirma que su plataforma admite miles de millones de dispositivos y billones de mensajes. Como se ha visto en la [estructura general de las plataformas](#), este IoT Core se puede conectar con multitud de servicios de AWS (Amazon Web Services) como AWS Kinesis, Amazon CloudWatch, AWS CloudTrail, Amazon DynamoDB o Amazon Elasticsearch. [13] [14]

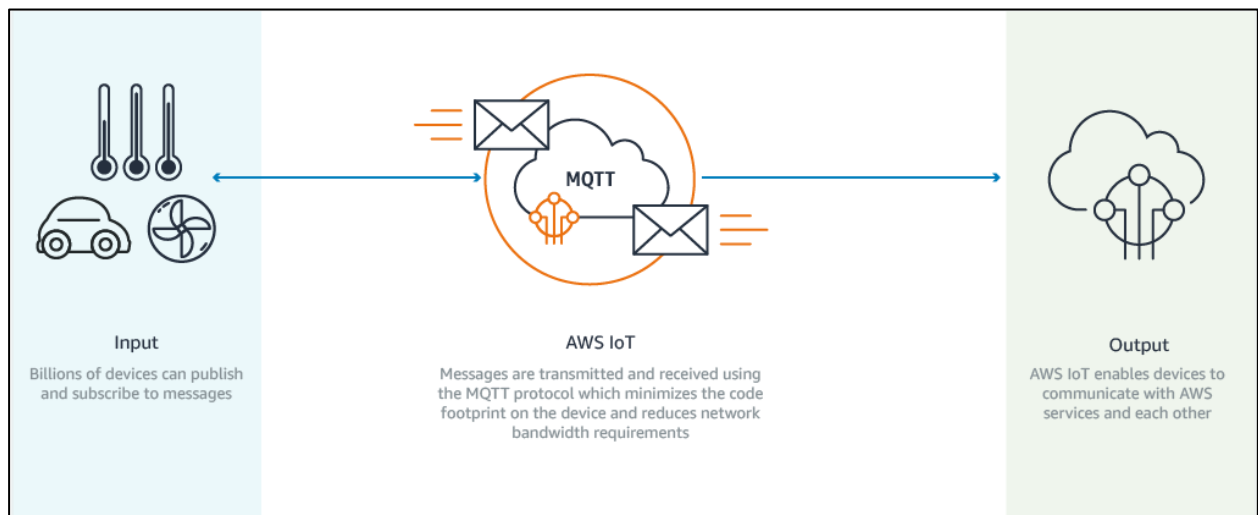


Figura 4-3. Conexión entre dispositivos, AWS IoT y servicios AWS.

Entre estos destacan AWS Kinesis para el análisis de los datos en tiempo real y Amazon DynamoDB para el almacenamiento de los datos recibidos en bases de datos en la nube, el cual se utiliza en este proyecto.

Admite HTTP, WebSockets y el protocolo MQTT y proporciona mecanismos de autenticación y cifrado integral en todos los puntos de la conexión.

4.3.1.1 Creación de cuenta en Amazon Web Services

En primer lugar es necesario registrarse en AWS para poder acceder a los servicios. Amazon ofrece una cuenta de desarrollador válida durante doce meses de forma gratuita, aunque se requiere una tarjeta de crédito para el proceso de registro.

4.3.1.2 Registro de la Raspberry Pi

Será necesario:

- 1) Registrar el dispositivo en la plataforma
- 2) Crear certificados [X.509](#) (estándar UIT-T)
- 3) Crear una Política
- 4) Asociar Política y Certificados al dispositivo registrado

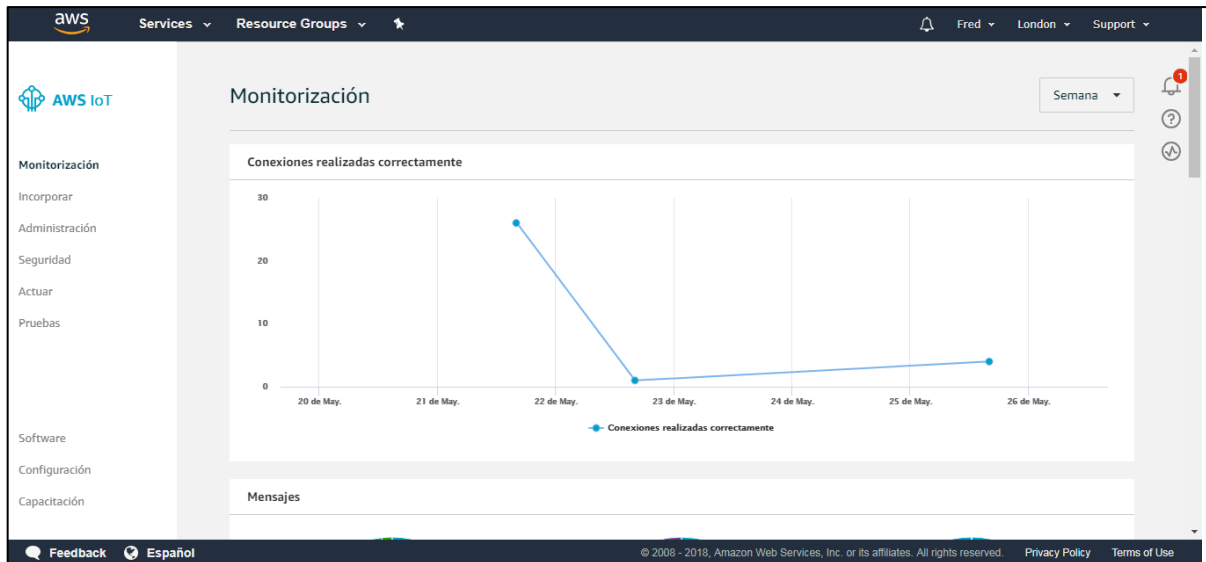


Figura 4-4. Interfaz AWS IoT

1. Se procede entonces a registrar el dispositivo accediendo al apartado Administración, donde únicamente se solicitará un nombre para el dispositivo.
2. Una vez definido el dispositivo en la plataforma es necesario crear los certificados, los cuales se generan a través de la misma desde la sección de seguridad, en “Crear”. En la siguiente figura se ilustra la interfaz con los certificados creados, que hay que descargar y almacenar en nuestro dispositivo en el directorio en el que se desarrolle posteriormente el código para subir los datos.

Estos certificados X.509 se utilizan para proteger la comunicación entre el dispositivo y la plataforma de manera que los datos lleguen de forma segura al sitio correcto desde el dispositivo adecuado.

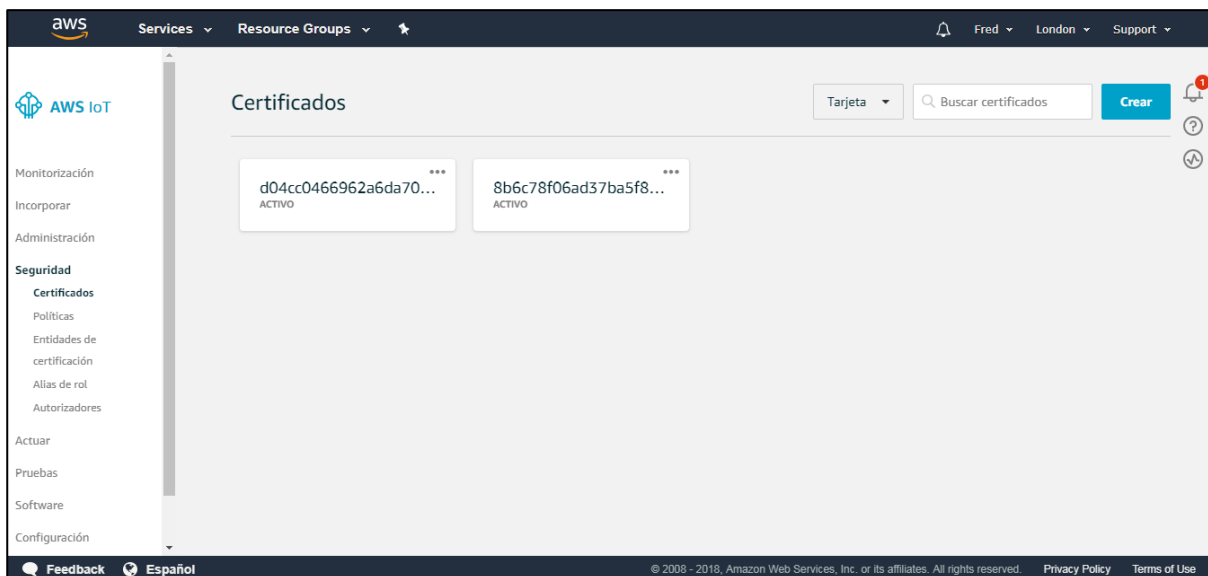


Figura 4-5. Certificados AWS IoT Core.

3. Se crea entonces una nueva política en el subpartado Políticas de Seguridad donde se especifica un nombre para la misma. La acción en este caso se define como “iot:*” la cual nos permite realizar todas las acciones. El recurso ARN (Nombres de Recursos de Amazon) que identifica de forma unívoca un recurso, con valor “*” y por último el efecto de la política “allow” o permitir.



Figura 4-6. Política Seguridad IoT Amazon.

4. Se prosigue emparejando con el certificado tanto la política como el dispositivo Raspberry Pi, como se muestra en la siguiente figura.

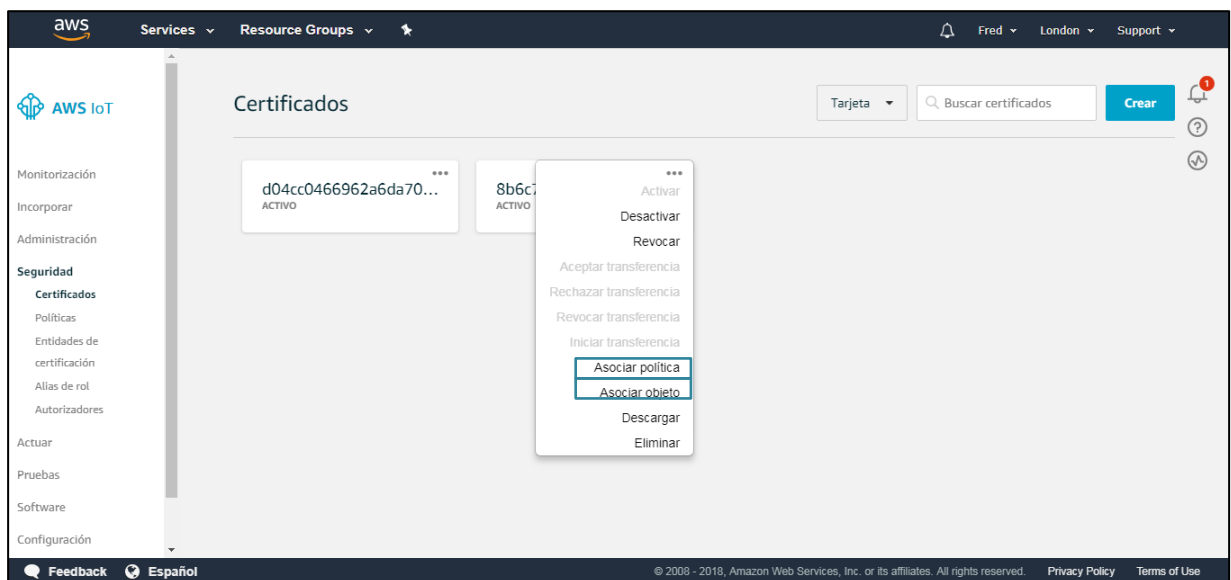


Figura 4-7. Asociación entre certificado, política y dispositivo.

Una vez realizadas estas configuraciones se procede con el dispositivo físico, desarrollando el código para el cliente mqtt.

4.3.1.3 Cliente MQTT para Amazon

Para realizar la conexión con la plataforma desde la Raspberry Pi se utiliza el cliente MQTT Paho de Eclipse descrito en el capítulo 3.

En primer lugar se definen algunas funciones, disponibles en la librería de paho como `on_connect`, `on_disconnect`, `on_log` y `error_str`.

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

    # Se comprueba la conexión ha tenido éxito
    if rc == 0:
        global connflag
        connflag = True
    # Bandera para notificar conexión establecida

# Cuando el cliente se desconecta de la plataforma
def on_disconnect(client,userdata, rc):
    # Un valor distinto de 0 indica conexión inesperada
    if rc != 0:
        print("Desconexión inesperada\n")

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

# Muestra información sobre el intercambio de mensajes, útil solo para desarrollador
def on_log(client, userdata, level, buf):
    print(buf)

# Convierte los códigos de error de paho a formato cadena para hacerlos legibles por el usuario
def error_str(rc):
    return '{}: {}'.format(rc, mqtt.error_string(rc))
```

Figura 4-8. Definición funciones.

- La función `on_connect` se utiliza cuando el cliente recibe un Connack desde la plataforma, tras haber enviado un mensaje Connect.

Para comprobar si el resultado de la conexión ha sido satisfactorio se encuentra la variable “rc” cuyos valores determinarán la respuesta de la plataforma. Si lo ha sido, se actualiza la bandera `connflag` a verdadero, esta bandera será útil para distinguir si estamos conectados o no, antes de enviar un mensaje tipo Publish.

Los valores de la variable “rc” (connection result) se muestran en la siguiente tabla.

Tabla 4-1. Variable rc.

Variable rc:	0	1	2	3	4	5
Significado	Conexión satisfactoria	Versión del protocolo incorrecta	Identificador de cliente no válido	Servidor no disponible	Usuario o contraseña incorrecto	Conexión no autorizada

- La función `on_disconnect` se utiliza cuando el cliente se desconecta de la plataforma. Si el parámetro “rc” es 0 significa que la desconexión ha sido satisfactoria, de lo contrario cualquier valor distinto de 0 significará que el cliente se ha desconectado de la plataforma de manera inesperada. Al desconectar el cliente de la plataforma se actualiza la bandera `connflag` a “False”.
- La función `on_log` sirve para mostrar información sobre los mensajes que se están intercambiando entre el dispositivo y la plataforma. Así, en una conexión normal encontraríamos en el siguiente orden los mensajes Connect, Connack, Publish y Disconnect.
Esta función es útil para el desarrollador a modo de depuración, pero para el cliente no tendría valor por lo que se desactiva su funcionamiento una vez depurado el código.
- Por último se encuentra `error_str` que se encarga de traducir los códigos de error de Eclipse Paho a formato cadena para que sean legibles por el usuario.

Se continúa entonces con la función principal **main** () la cual recibe dos parámetros, “subirDatosPropios” y “datos”.

```
def main(subirDatosPropios, datos):

    # Cliente mqtt version 3.1.1
    mqttc = mqtt.Client(protocol=mqtt.MQTTv311)

    mqttc.on_connect = on_connect          # Asignacion a on_connect
    mqttc.on_disconnect = on_disconnect    # Asignacion a on_disconnect
    mqttc.on_log = on_log                  # Asignacion a on_log, util para depurar r
    mqttc.error_str = error_str           # Asignacion a error_str

    ### Parametros ###
    awshost = "amgbu8oes223z.iot.eu-west-2.amazonaws.com" # Direccion de la plataforma
    awsport = 8883                                         # Puerto

    clientId = "Raspberry"                               # Id del cliente
    thingName = "Raspberry"                              # Nombre del objeto

    caPath = "Certificados/root-CA.crt"                   # Ruta hasta el certificado Root-CA
    certPath = "Certificados/Raspberry.cert.pem"         # Ruta hasta el certificado .cert.pem
    keyPath = "Certificados/Raspberry.private.key"       # Ruta hasta la clave privada .private.key
```

Figura 4-9. Fragmento main ().

En esta función se empieza inicializando el cliente mqtt, en este caso mqttc, llamando al constructor Client() de Eclipse Paho y se especifica la versión del protocolo a utilizar (3.1.1). A continuación se asignan las cuatro funciones definidas con las de Paho y se definen los parámetros necesarios para realizar la conexión con Amazon Web Services.

- **awshost**: indica la URL en la que se encuentra el servicio.
- **awsport**: 8883 es el puerto para MQTT sobre SSL. En este caso se utilizan certificados para la autenticación, por lo que se utiliza este puerto. Si no se utilizase la configuración por tls se podría usar el puerto 1883, también estándar para MQTT, pero no seguro.
- **clientId**: Hace referencia al nombre del objeto definido previamente en la plataforma como “Raspberry”.
- **thingName**: De nuevo su valor es el nombre del objeto.
- **caPath**: Ruta hasta el certificado root-CA.crt
- **certPath**: Indica la ruta hasta el certificado Raspberry.cert.pem.
- **keyPath**: Ruta hasta la clave privada.

Definidos los certificados y sus ubicaciones se realiza la configuración tls mediante “tls_set()”.

```
### Configuracion tls ###
mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.CERT_REQUIRED,
             tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)

# Para conectarse con aws
mqttc.connect(awshost, awsport, keepalive=60)
mqttc.loop_start()                                     # Inicia un hilo en segundo plano para llamar a loop()
                                                         # que procesa los eventos de red
```

Figura 4-10. Configuración tls y conexión.

Se observa también la conexión mediante “mqttc.connect(awshost, awsport, keepalive=60)” donde los dos primeros parámetros son los definidos anteriormente y el último, keepalive, representa el tiempo máximo en segundos permitido entre comunicaciones.

Por último se encuentra la función “loop_start()” que abre un hilo en segundo plano para procesar los eventos de red.

El cliente dispone de dos opciones en cuanto a la transmisión de los datos:

1. Subir los datos del sensor de temperatura de la CPU de la Raspberry Pi, para lo que el parámetro “subirDatosPropios” sería 0 y el segundo parámetro “datos” sería una cadena vacía.
2. Subir su propia cadena a la nube, para lo que el primer parámetro valdría 1 y el segundo sería igual al contenido de la cadena introducida por teclado.

Para la primera opción se tiene:

```
# ---Se envian los datos de temperatura---#
if subirDatosPropios == 0:
    # Pausa de dos seg para dar tiempo a la conexion
    sleep(2)

    if connflag == True:

        fichero = open("/sys/class/thermal/thermal_zone0/temp")    # Se accede al fichero temp, donde se almacenan los
        temp = fichero.read(5)
        temp = float(temp)/1000    # Se divide entre 1000 por el formato
        cadena = str(temp) + " Grados Centigrados"    # Se forma la cadena a enviar
        fecha = time.strftime("%c")    # Fecha de la consulta, %c para formato fecha y hora

        # Datos a enviar
        datosCadena = {'Datos':cadena, 'Fecha': fecha}

        # Se convierte a formato Json para permitir el acceso a los datos desde
        # la base de datos "dynamoDB" de Amazon
        datosJson = json.dumps(datosCadena)

        # Se publican los datos en la nube con qos = 1
        # o calidad de servicio, indica que el mensaje se manda "al menos una vez"
        mqttc.publish("raspberry/temperatura/cpu", datosJson, qos=1)

        print("Mensaje enviado: temperatura " + cadena + "\n")    # Se imprime por pantalla el mensaje publicado
    else:
        print("No se pudo establecer la conexion\n")
```

Figura 4-11. Publicar mensaje con temperatura.

Se empieza por una espera de dos segundos, periodo para dar tiempo a que se establezca la conexión tras haber llamado a “connect()” y se comprueba el valor de la bandera “connflag” para conocer el estado de la conexión. Por tanto si el valor es “True” o verdadero se continúa accediendo al fichero “temp” ubicado en la ruta “/sys/class/thermal/thermal_zone0”.

En este fichero se almacena periódicamente el valor de la temperatura de la CPU obtenido del sensor que trae integrado de serie el modelo B de Raspberry Pi 3. Como se observa en el código, se leen los 5 primeros caracteres y se dividen por 1000. Esta división es necesaria ya que por defecto si una temperatura de ejemplo podría ser 50°C, en el fichero aparecería como 50.000.

A la cifra de la temperatura se le añade la unidad en la que se mide, grados centígrados, convirtiendo el conjunto en una cadena. Cada vez que se realiza una medición de la temperatura, se registra la fecha mediante “time.strftime(“%c”)” donde “%c” es el formato para fecha y hora. Finalmente se convierte a formato Json, conversión necesaria para acceder y almacenar posteriormente los datos a través de la plataforma.

Por último se publica el mensaje ya configurado mediante “mqttc.publish()” donde como primer parámetro se encuentra el “Topic” en el que publicar, los datos como segundo parámetro y la calidad del servicio (qos) establecida en 1 como último parámetro. Con esta calidad se asegura el envío “at least once”, es decir se envía al menos una vez, lo que supone que podría haber algún mensaje duplicado.

Ésta es la calidad de servicio máxima que soporta Amazon Web Services. [15]

En cuanto a la segunda opción, se subirá a la plataforma la cadena que indique el cliente por teclado:

```
# ----Se envian los datos pasados por el usuario----#
else:
    # Pausa de seg seg para esperar la conexion
    sleep(2)

    if connflag == True:

        fecha = time.strftime("%c")

        # En este caso "datos" contiene la cadena pasada por el usuario
        datosCadena = {'Datos':datos, 'Fecha': fecha}
        datosJson = json.dumps(datosCadena)

        mqttc.publish("raspberrypi/temperatura/cpu", datosJson, qos=1)

        print("Mensaje '" + datos + "' se ha publicado en la nube de Amazon\n")

    else:
        print("No se ha conseguido establecer conexion\n")

# Desconexion con la plataforma
mqttc.disconnect()

# Espera de dos seg para desconexion
sleep(2)
```

Figura 4-12. Publicar datos propios del usuario

Al igual que en la anterior figura se empieza comprobando el valor de “connflag”. Si hay conexión, se almacena la fecha y se envía conjuntamente con el parámetro “datos” que contiene la cadena introducida por el cliente. Se utiliza el mismo “Topic” y calidad de servicio que en la primera opción.

Finalmente, con independencia de los datos enviados, se cierra la conexión con “mqttc.disconnect()” desconectando así el cliente de la plataforma y realizando una espera de dos segundos para dar tiempo a la desconexión.

4.3.1.4 Conexión AWS IoT con DynamoDB

Llegados a este punto seríamos capaces de transmitir datos desde el dispositivo hasta la plataforma, pero para visualizar y almacenar los mismos es necesario integrar el funcionamiento del módulo DynamoDB.

Para conectar AWS IoT con DynamoDB es necesaria la creación de una regla. Ésta nos permitirá recoger los mensajes MQTT enviados desde el dispositivo y escribirlos en una tabla de DynamoDB.

Desde el apartado “Actuar” de la interfaz de AWS IoT, se accede a la administración y creación de las reglas. Para la creación de una regla se solicita un nombre así como un atributo y un filtro de tema o “Topic Filter”. Para este proyecto se establece como **atributo** “*” de forma que todo el contenido del mensaje se escribirá en la tabla, y como filtro de temas, “raspberrypi/temperatura/cpu”, el mismo que se utilizó desde el cliente para publicar los datos.



Instrucción de consulta de regla

```
SELECT * FROM 'raspberrypi/temperatura/cpu'
```

Atributo ?

*

Filtro de temas ?

raspberrypi/temperatura/cpu

Figura 4-13. Creación regla AWS.

Tras crear la regla hay que asignarle una acción, entre los distintos servicios que se ofrecen se selecciona “DynamoDB”.

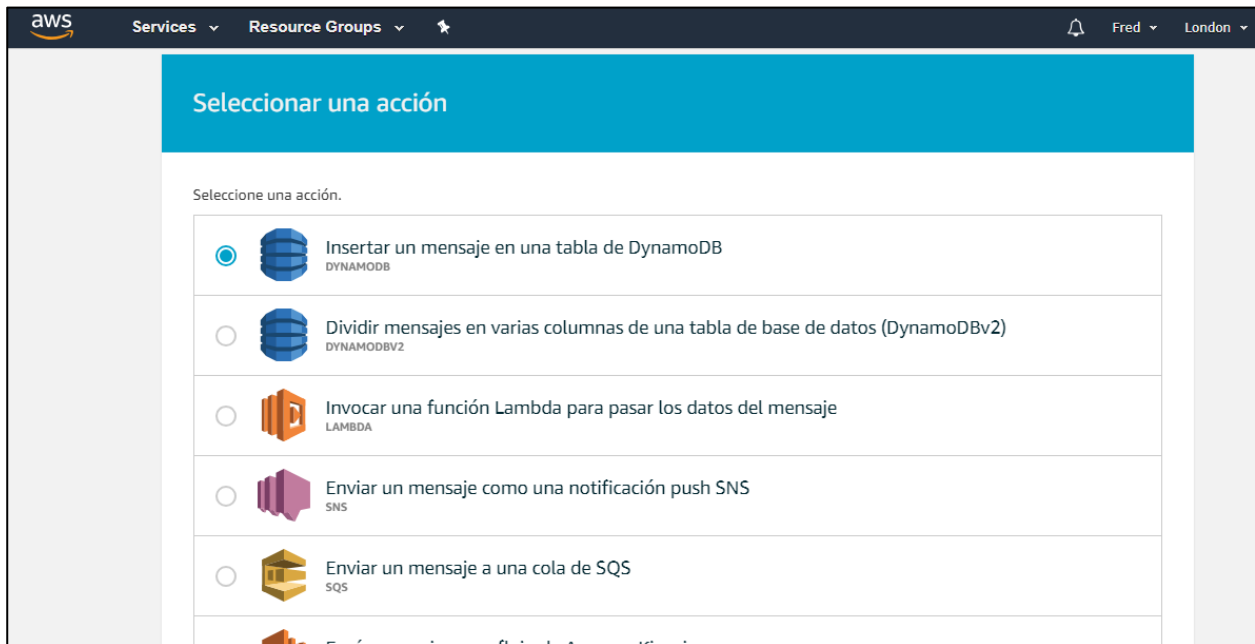


Figura 4-14. Selección de la acción para la Regla.

Ahora es necesario crear una tabla antes de continuar, desde el servicio DynamoDB se especifica el nombre de la tabla y la clave primaria así como sus tipos.

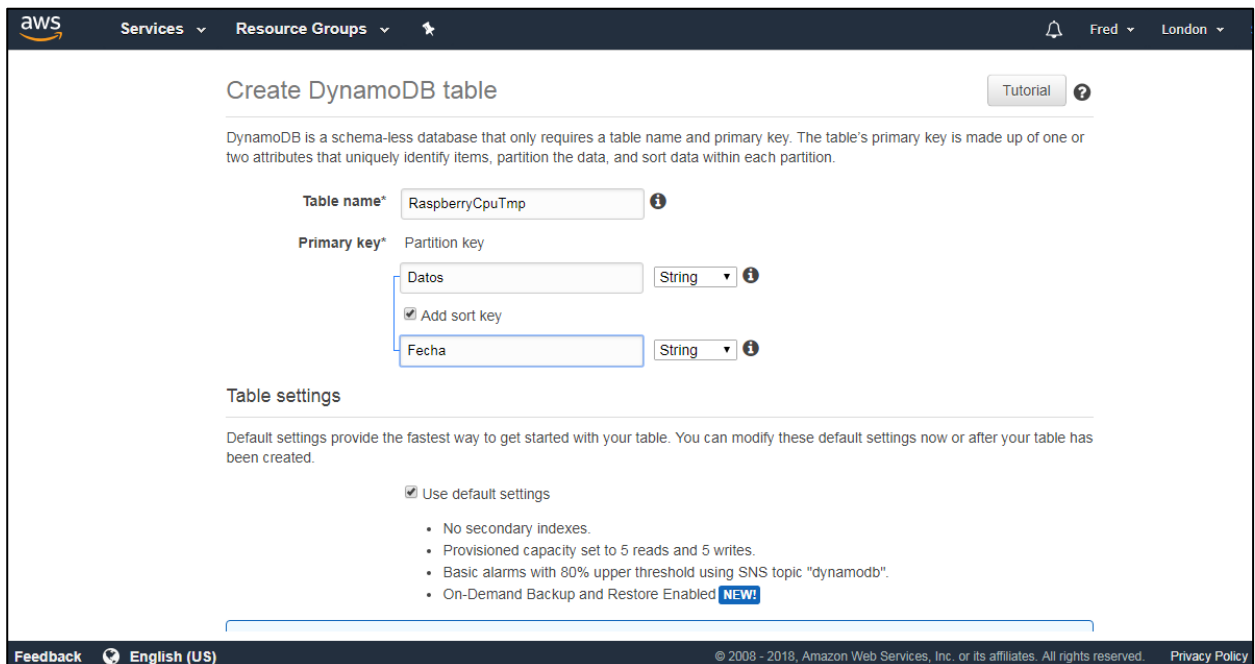
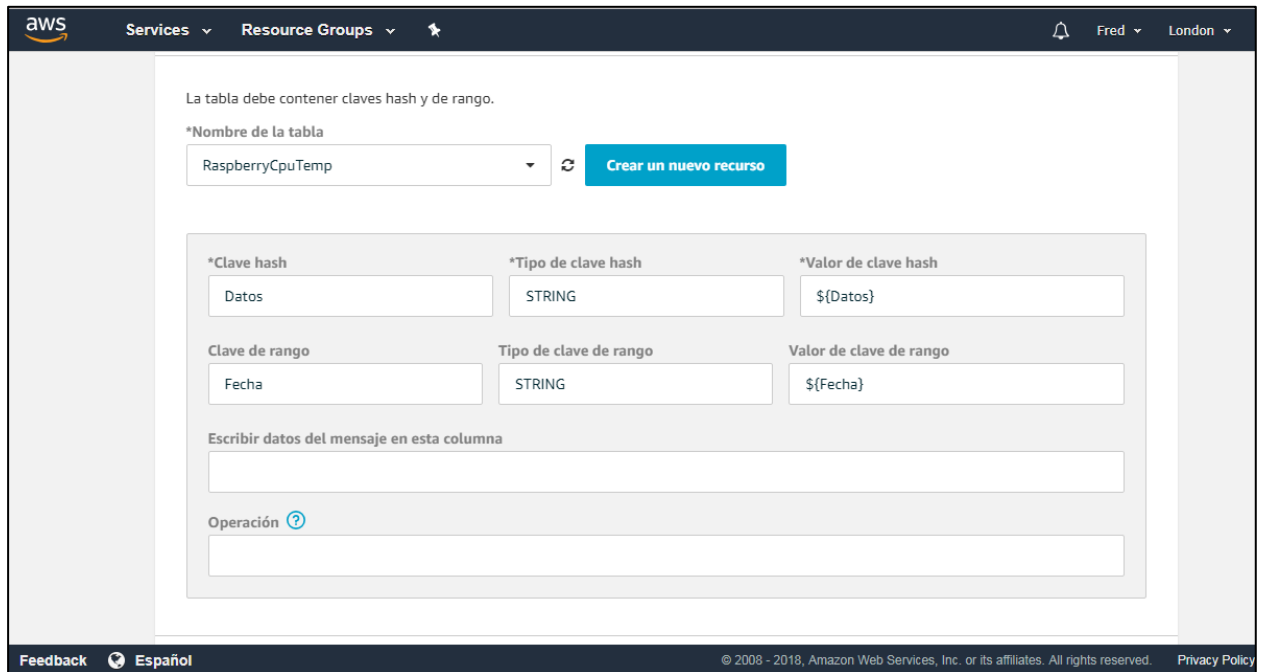


Figura 4-15. Creación Tabla DynamoDB.

Se continúa configurando la acción para lo que se selecciona la tabla recién creada. Aquí es donde se denota la importancia de haber convertido a Json el formato de los datos enviados desde el dispositivo, pues en el campo “Hash key value” se introduce “\${Datos}”, indicando así a la regla que debe tomar el valor del atributo Datos del mensaje MQTT. Se actúa de igual forma para el atributo Fecha.



La tabla debe contener claves hash y de rango.

*Nombre de la tabla
RaspberryCpuTemp [Crear un nuevo recurso](#)

*Clave hash	*Tipo de clave hash	*Valor de clave hash
Datos	STRING	\${Datos}
Clave de rango	Tipo de clave de rango	Valor de clave de rango
Fecha	STRING	\${Fecha}

Escribir datos del mensaje en esta columna

Operación [?](#)

Feedback Español © 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy

Figura 4-16. Configuración de la acción.

Recordando la parte del cliente vemos como los atributos eran “Datos” y “Fecha”:

```
# Datos a enviar
datosCadena = {'Datos':cadena, 'Fecha': fecha}

# Se convierte a formato Json para permitir el acceso a los datos desde
# la base de datos "dynamoDB" de Amazon
datosJson = json.dumps(datosCadena)
```

Figura 4-17. Formato de los datos enviados.

Realizada la creación de la regla, la tabla y las configuraciones respectivas queda todo dispuesto para realizar un uso completo del servicio, desde el envío de datos del dispositivo hasta la recepción, visualización y almacenamiento de los mismos en la plataforma.

4.3.1.5 Visualización de los datos en Amazon

Si se accede al servicio DynamoDB desde la consola de Amazon encontramos la tabla RaspberryCpuTemp, creada unas páginas atrás. Tras realizar un envío de datos desde el dispositivo, la regla haría su efecto enrutando los datos de AWS IoT hacia la base de datos. Encontraríamos entonces los mensajes enviados en la tabla como se muestra en la siguiente figura:

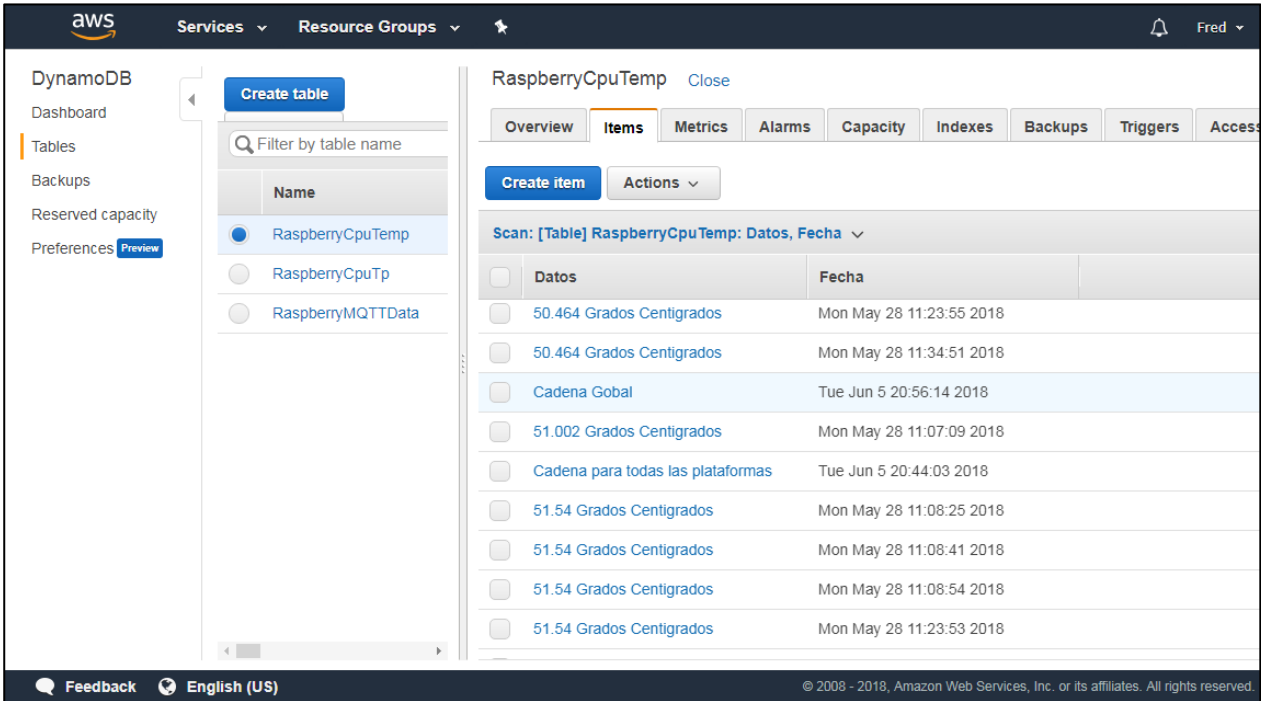


Figura 4-18. Datos del dispositivo en la Base de Datos de la nube.

Como se observa, quedan recogidos los valores de la temperatura de la CPU del dispositivo Raspberry Pi, junto a la fecha en la que se consultaron esos valores. Se distinguen también entre los datos, las cadenas introducidas por teclado, transmitidas por el usuario junto a la fecha de envío.

4.3.1.6 Costes

El servicio AWS que presta Amazon presenta una calculadora para hacer una estimación de los costes. Para ilustrar un ejemplo se toma 1 dispositivo con un envío de 400.000 mensajes al día por dispositivo y un tamaño por mensaje de 5 KB. Estos parámetros serán útiles para realizar una comparación con el resto de plataformas posteriormente.

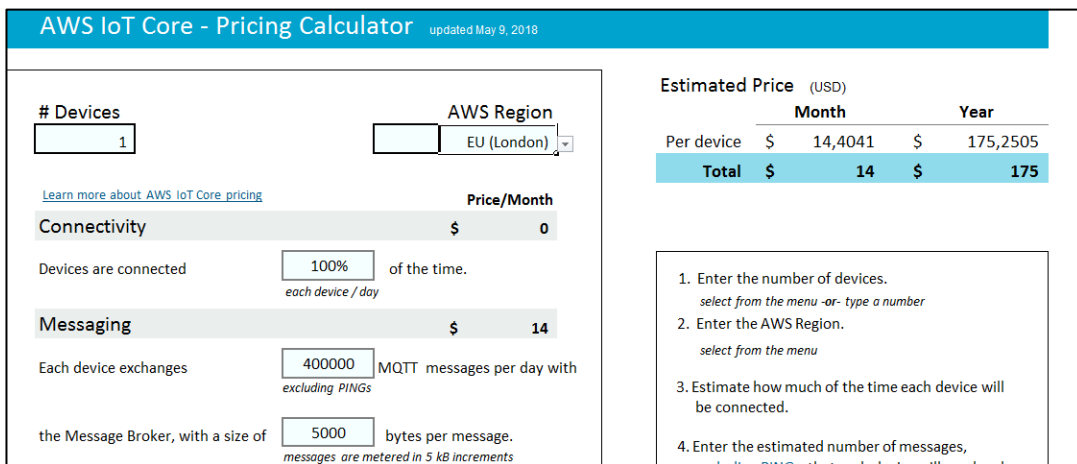


Figura 4-19. Costes para la plataforma Amazon.

4.3.2 Google IoT Core

Google ofrece su propuesta bajo el nombre de Google IoT Core, basada en la computación mediante máquinas virtuales escalables, es un servicio completamente administrado que al igual que el anterior, nos permite conectar y administrar datos de millones de dispositivos.

Esta solución ofrece multitud de servicios relacionados con el almacenamiento de datos y el análisis de los mismos. Además soporta el protocolo objeto de este proyecto, MQTT, para ser implantado en pasarela utilizando el dispositivo Raspberry Pi. [16] [17]

De forma gráfica se muestra la estructura de la plataforma IoT que ofrece Google de forma ordenada, es decir mostrando la conexión de cada uno de los distintos módulos:

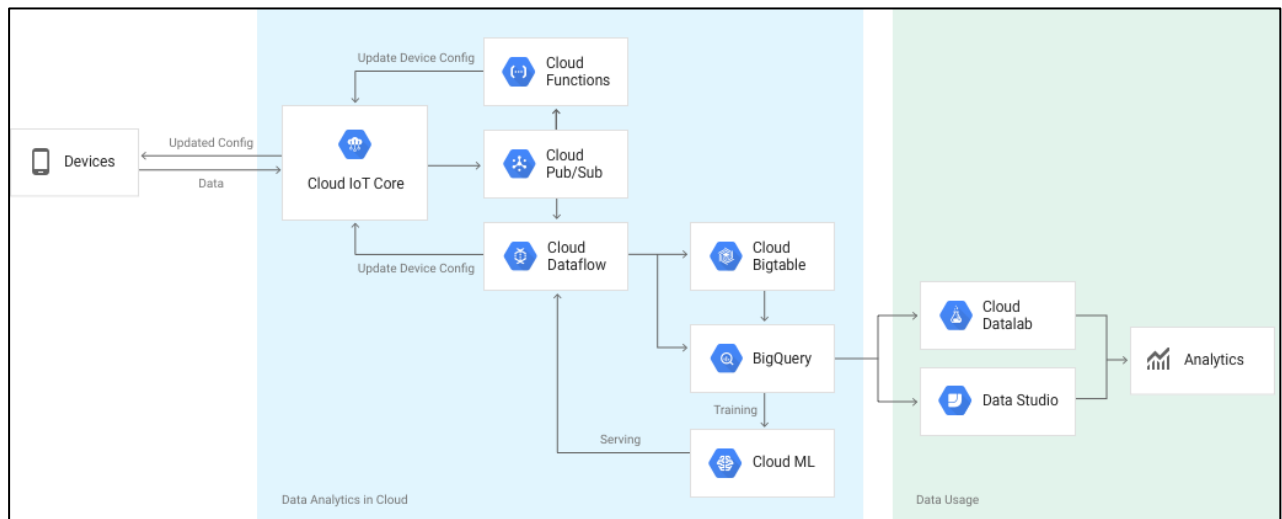


Figura 4-20. Estructura plataforma Google.

En este proyecto se utiliza el módulo central “Cloud IoT Core” para recopilar directamente los datos del dispositivo. Tiene dos componentes principales:

1. **Administrador de dispositivos:** Para administrar y configurar de forma segura y por partes los dispositivos a través de consola o de forma automatizada.
2. **Puente de protocolo:** Se encarga de la conexión entre los dispositivos y la plataforma para el envío de datos.

Este módulo se conecta con el segundo, “Cloud Pub/Sub” que sirve para recopilar los datos de dispositivos dispersos en un sistema global. Posteriormente se pasa por el módulo “Cloud Dataflow” cuya función será encaminar los datos al último módulo, “BigQuery”. Su función será almacenar el flujo de datos entrantes en tablas SQL.

La plataforma cuenta con socios como Intel, Cisco o Helium y en cuanto a la seguridad en las comunicaciones, ofrece seguridad extremo a extremo a través de autenticación de clave simétrica mediante TLS 1.2 y certificados CA para verificar la propiedad de los dispositivos.

4.3.2.1 Cuenta para los servicios IoT de Google

Al igual que Amazon, ofrece una cuenta de prueba de manera gratuita válida para un periodo de 12 meses con un saldo inicial de 270€. De nuevo al igual que la plataforma anterior, requiere una tarjeta de crédito para darse de alta en el servicio.

4.3.2.2 Registro del dispositivo Raspberry Pi

Los pasos necesarios son:

- 1) Crear el registro del dispositivo
- 2) Registro del dispositivo y gestión de los certificados X.509

Se presenta la interfaz principal de Google IoT Core, desde donde se creará el registro de los dispositivos a utilizar:

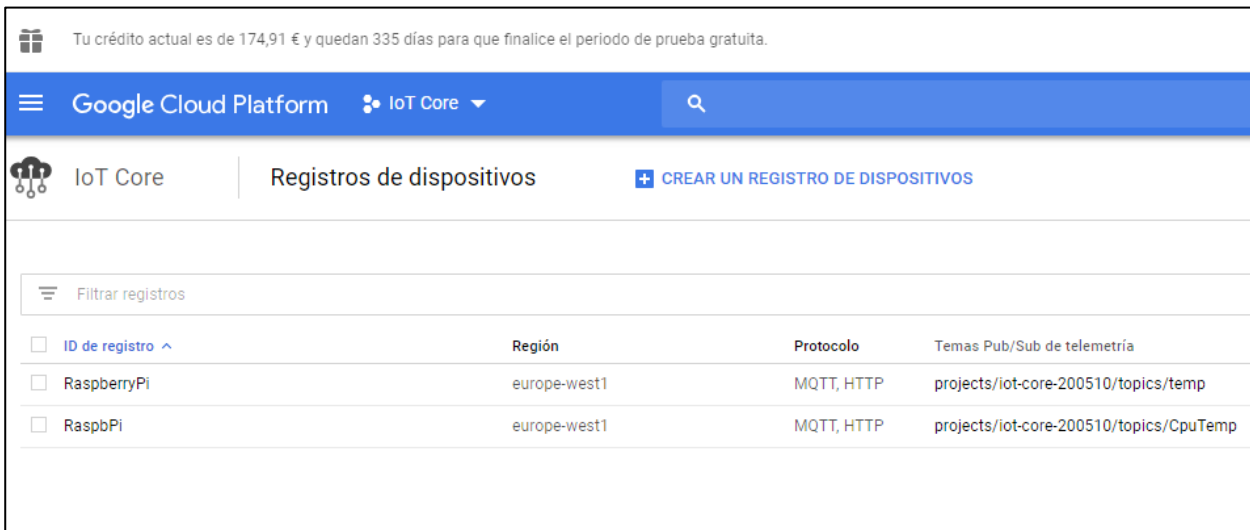


Figura 4-21. Interfaz principal IoT Core.

Por tanto para realizar el **primer paso** será necesario declarar un nombre para el registro y establecer la región en la que se desea que se almacenen los datos. Cabe destacar que para evitar un aumento en los costes es preferible establecer aquella región más cercana a donde tengamos los dispositivos, pues esta decisión es permanente.

Se seleccionan también los protocolos que usarán los dispositivos y se añade un tema o “Topic” para el módulo “Cloud Pub/Sub”, en el que se publicarán los datos. Así pues el resultado sería el siguiente.

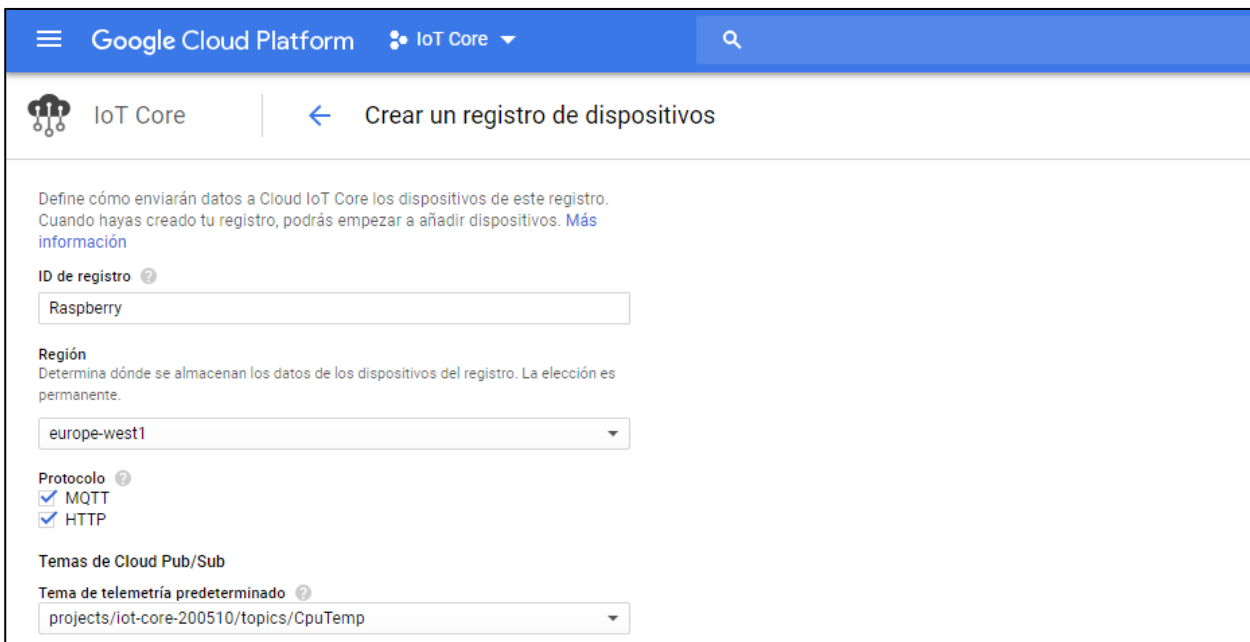


Figura 4-22. Registro dispositivo.

Una vez creado el registro, como **segundo paso**, se accede al mismo para asignarle un dispositivo, habilitar la comunicación con Google Cloud y asignar el fichero que contiene la clave pública.

Creado el registro y asociado el dispositivo al mismo, y con la autenticación mediante los certificados realizada quedaría la configuración dispuesta a la espera de que el cliente comience la transmisión de datos. Sin embargo sin conectar el resto de módulos mencionados con anterioridad, no habrá forma de visualizar los datos transmitidos, por lo que será necesario configurar el módulo “Dataflow” y “BigQuery” posteriormente.

La siguiente captura muestra cómo añadir el dispositivo al registro:

Figura 4-23. Dispositivo añadido al registro.

4.3.2.3 Cliente MQTT para Google

En cuanto al cliente, se vuelve a utilizar el ya comentado Paho Client de Eclipse al igual que con el cliente para la plataforma Amazon. Empiezo declarando las variables necesarias para la conexión con la plataforma. [18]

```
##### Variables para establecer la conexion con Google Cloud IoT Core #####
project_id = "iot-core-200510"           # Id del proyecto
cloud_region = "europe-west1"          # Define la region de los servidores de la plataforma a usar
registry_id = "RaspbPi"
device_id = "RasperryPi"

#---Certificados---#
private_key_file = 'rsa_private-Google.pem'
algorithm = "RS256"                    # Algoritmo de encriptacion para la creacion del Json Web Token
ca_certs = 'rootsGoogle.pem'

mqtt_bridge_hostname = "mqtt.googleapis.com" # Direccion de la plataforma
mqtt_bridge_port = "8883"               # Puerto MQTT sobre SSL

connflag = False                        # Bandera para la conexion

##### Fin Variables establecer la conexion #####
```

Figura 4-24. Parámetros para conexión Google IoT Core.

Se especifica el identificador del proyecto así como el id del registro y del dispositivo, campos que se crearon en el apartado anterior. También es necesario especificar el parámetro “cloud_region” para establecer la región de los servidores a usar que debe ser la que se estableció a la hora de crear el registro.

En el apartado de certificados encontramos la ruta hasta el fichero con la clave privada en “private_key_file” y el certificado CA. El campo “algorithm” especifica el algoritmo de encriptación a utilizar para crear el “Json Web Token” o JWT utilizado para la autenticación. Un JWT es una codificación de [Token](#) basada en Json.

Por último en el apartado de variables encontramos “mqtt_bridge_hostname” y “mqtt_bridge_port” para especificar la dirección de la plataforma y el puerto a usar. El parámetro “connflag” es la bandera para conocer el estado de la conexión, al igual que en la anterior plataforma.

Continuando con el código encontramos la definición de las funciones “on_connect”, “on_disconnect”, “on_log” y “error_str” definidas y utilizadas también en la [sección de Amazon](#):

```
# Entra cuando se establece la conexión con la plataforma (llega el mensaje CONNACK)
def on_connect(unused_client, unused_userdata, unused_flags, rc):
    print("Conectando a Google IoT Core\n")
    print("Resultado: " + error_str(rc))
    # Se notifica al usuario que se esta conectando
    # Resultado de la conexión

    # Se comprueba si el resultado ha sido satisfactorio
    if rc == 0:
        global connflag
        connflag = True
    # Bandera que notifica conexión establecida

# Se llama cuando el cliente se desconecta de la plataforma
def on_disconnect(unused_client, unused_userdata, rc):
    # rc distinto de 0 implica desconexión inesperada
    if rc != 0:
        print("Desconexión inesperada")

    print("Desconectando de la plataforma")
    print("Resultado: " + error_str(rc) + "\n")
    global connflag
    connflag = False
    # Actualización de la bandera de conexión

# Muestra información sobre el intercambio de mensajes, para depurar
def on_log(client, userdata, level, buf):
    print(level)
    print(buf)

# Devuelve los códigos de errores de paho en formato cadena
def error_str(rc):
    return '{}: {}'.format(rc, mqtt.error_string(rc))
```

Figura 4-25. Definición funciones Google.

Como salvaguarda para el cliente de Google se implementa una función más, “create_jwt”, encargada de crear el JWT mediante el algoritmo especificado, en este caso [RS256](#) (estándar RFC). Para el token, se especifica el momento de creación, el momento en el que expirará y el proyecto al que está asociado.

```
##### Creación del Json Web Token para conseguir una comunicación segura por la red #####
def create_jwt(project_id, private_key_file, algorithm):

    token = {
        # Momento en el que se crea el token
        'iat': datetime.datetime.utcnow(),
        # Tiempo en el que expirará el token
        'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
        # Este campo debe ser siempre el id del proyecto
        'aud': project_id
    }

    # Lee el fichero de clave privada
    with open(private_key_file, 'r') as f:
        private_key = f.read()

    print("Creando el JWT usando el algoritmo {} y el fichero {}".format(
        algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)
##### Fin JWT #####
```

Figura 4-26. Función para crear JWT.

A continuación se encuentra la función **main** que recibe de nuevo dos parámetros, subirDatosPropios y datos. Esta función inicializa el cliente mqtt estableciendo un nombre de usuario (no usado) y una contraseña para la autenticación. Esta contraseña es realmente el JWT que se acaba de definir.

Se configuran las opciones de encriptación y autenticación con “tls_set”, se realizan las asignaciones a las funciones definidas y se configura el “topic” o tema en el que publicar para el que Google establece un formato concreto indicado en la siguiente captura.

```
def main(subirDatosPropios, datos):

    # Se inicializa el cliente MQTT, Google IoT Core necesita el id de cliente con el siguiente formato
    client = mqtt.Client(client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(project_id,
                                                                                          cloud_region,
                                                                                          registry_id,
                                                                                          device_id),protocol=mqtt.MQTTv311)

    client.on_connect = on_connect
    client.on_disconnect = on_disconnect
    client.on_log = on_log
    client.error_str = error_str

    # En Google Cloud IoT Core, el parametro username se ignora
    # El campo contraseña se utiliza para transmitir el Json Web Token (JWT)
    client.username_pw_set(username='unused',password=create_jwt(project_id, private_key_file, algorithm))

    # Configuración tls para comunicación segura por la red
    client.tls_set(ca_certs=ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)

    # Se realiza la conexión con Google Cloud IoT Core
    client.connect(mqtt_bridge_hostname, mqtt_bridge_port)
    client.loop_start() # Procesa los eventos de red

    # Tema en el que se publicaran los mensajes
    sub_topic = 'state'
    mqtt_topic = '/devices/{}/{}'.format(device_id, sub_topic)
```

Figura 4-27. Configuración cliente Google.

De nuevo al igual que con el cliente de Amazon y los sucesivos se distinguen dos situaciones, una en la que el cliente decide enviar los datos de la temperatura del sensor y otra en la que decide enviar sus propios datos. Se distinguen ambas con el parámetro “subirDatosPropios”.

Para estas dos situaciones se actúa de igual forma que la explicada para Amazon, a excepción del tema especificado en el que publicar. Se muestran a continuación las capturas de ambas situaciones.

Si los datos para subir a la plataforma son los del sensor de temperatura ([explicado en sección de Amazon](#)):

```
if subirDatosPropios == 0: # Se accede entonces al sensor para obtener datos
    # Espera de dos seg para establecer la conexión
    sleep(2)

    if connflag == True:

        fichero = open("/sys/class/thermal/thermal_zone0/temp") # Se accede al fichero temp, donde se almacenan los valores de t
        temp = fichero.read(5) # Se leen los 5 primeros caracteres
        temp = float(temp)/1000 # Se divide entre 1000 por el formato

        payload = str(temp) + " Grados Centigrados" # Se forma la cadena a enviar
        fecha = time.strftime("%c") # Registro de la fecha de la consulta, %c establece fecha y hora

        datosEnviar = {'Cputemp': payload, 'Fecha': fecha} # Se convierte a formato Json para permitir el almacenamiento...
        datosJson = json.dumps(datosEnviar) # ...en la base de datos de Google "BigQuery"

        client.publish(mqtt_topic, datosJson, qos=1) # Publicación de los datos en el Tema establecido y con
        print("Datos " + datosJson + " publicados\n") # calidad de servicio (qos) 1, se publica "al menos 1 vez"

    else:
        print("no se pudo establecer la conexión\n")
```

Figura 4-28. Subir datos de temperatura.

Si los datos son los especificados por el cliente desde el teclado ([explicado en sección de Amazon](#)):

```
else: # Se envía a la nube la cadena especificada por el usuario
    # Pausa de dos seg para la conexión
    sleep(2)

    print("Publicando datos\n")

    if connflag == True: # Se comprueba que haya conexión

        fecha = time.strftime("%c")
        payload = datos # Los datos son los especificados por el usuario mediante teclado

        datosEnviar = {'Cputemp': payload, 'Fecha': fecha} # Al igual que antes, se convierte a Json para permitir...
        datosJson = json.dumps(datosEnviar) # ... el acceso a los datos desde la nube

        client.publish(mqtt_topic, datosJson, qos=1) # Se publican los datos proporcionados por el usuario
        print("Datos " + datosJson + " publicados\n")

    else:
        print("No se pudo establecer conexión\n")
```

Figura 4-29. Subir datos del cliente.

Por último independientemente de los datos transmitidos se realiza la desconexión.

```
# Desconexion con la plataforma
client.disconnect()

# Espera de dos seg para dar tiempo a la desconexion
sleep(2)
```

Figura 4-30. Desconexión de Google IoT Core.

Para la plataforma de Google la calidad de servicio (QoS) establecida cuando se publican los datos con publish, se mantiene a 1 ya que al igual que en Amazon, es la calidad máxima soportada. [19]

Desarrollado el cliente y configurada la plataforma queda pendiente la conexión del módulo Cloud IoT Core con el resto de módulos para la visualización de los datos publicados.

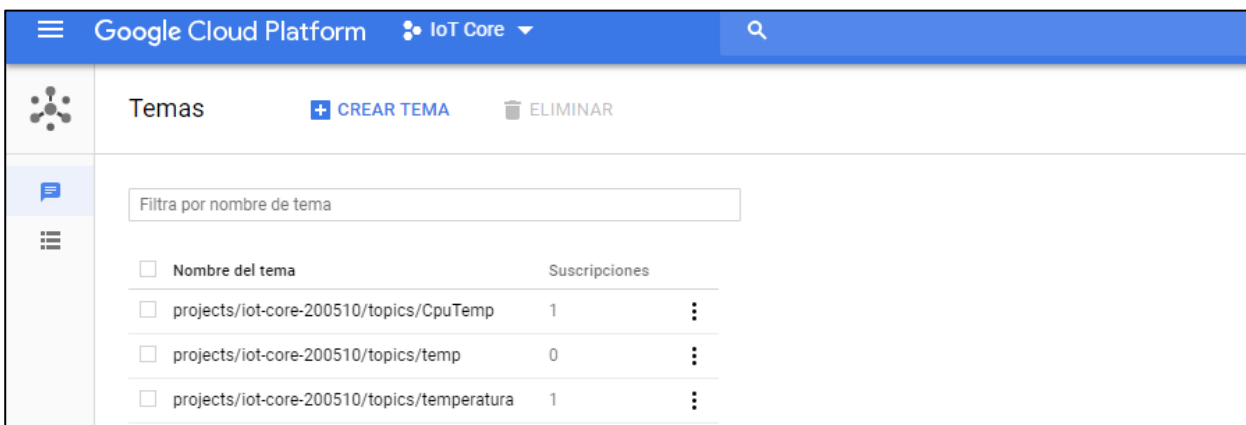
4.3.2.4 Conexión Cloud IoT Core con BigQuery

BigQuery es un módulo de análisis de Big Data empleado por todo tipo de empresas, desde startups hasta multinacionales. Nos permitirá almacenar los datos de forma administrada, accediendo a ellos mediante consultas SQL. [20]

Pero para llegar a él es necesario configurar previamente los módulos Cloud Pub/Sub y Dataflow.

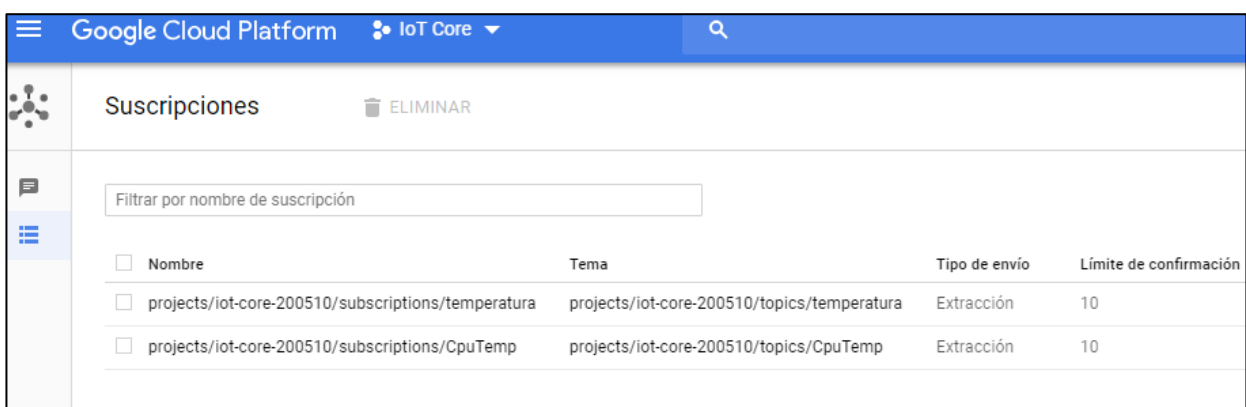
Cloud Pub/Sub es un servicio de mensajería administrado en tiempo real que permite enviar y recibir mensajes entre aplicaciones independientes, en este caso el dispositivo Raspberry Pi [21].

Desde la interfaz de este módulo creamos un tema y una suscripción al mismo para recuperar los datos y exportarlos a BigQuery mediante Dataflow.



Nombre del tema	Suscripciones	
<input type="checkbox"/> projects/iot-core-200510/topics/CpuTemp	1	⋮
<input type="checkbox"/> projects/iot-core-200510/topics/temp	0	⋮
<input type="checkbox"/> projects/iot-core-200510/topics/temperatura	1	⋮

Figura 4-31. Tema Pub/Sub.



Nombre	Tema	Tipo de envío	Límite de confirmación
<input type="checkbox"/> projects/iot-core-200510/subscriptions/temperatura	projects/iot-core-200510/topics/temperatura	Extracción	10
<input type="checkbox"/> projects/iot-core-200510/subscriptions/CpuTemp	projects/iot-core-200510/topics/CpuTemp	Extracción	10

Figura 4-32. Suscripción Pub/Sub.

Entrando en los detalles del tema creado se visualiza la opción de exportar al módulo BigQuery, opción que nos redirigirá primero a la configuración del módulo Dataflow.

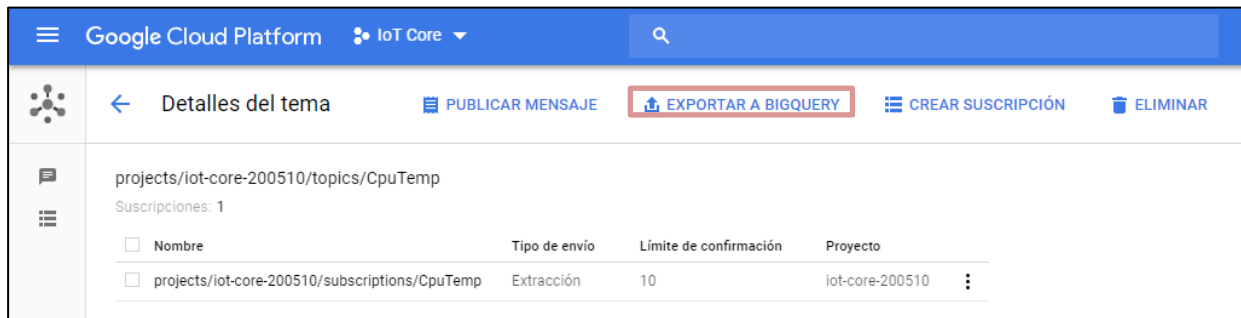


Figura 4-33. Exportar a BigQuery.

Dataflow es un servicio de procesamiento de datos totalmente administrado capaz de canalizar los datos, tanto en “streaming” o de forma continua como por lotes. [22]

En este módulo se crea una tarea que se encargará de canalizar los datos recogidos por Cloud Pub/Sub hacia el módulo BigQuery. Habrá que asignar un nombre único para la tarea y definir en qué región desplegar el servicio, para este proyecto “europe-west1” por ser la más cercana al dispositivo.

Además se escoge la plantilla “PubSub to BigQuery” que permite ejecutar flujos de procesamiento preinstalados con el objetivo de dirigir los datos a BigQuery.

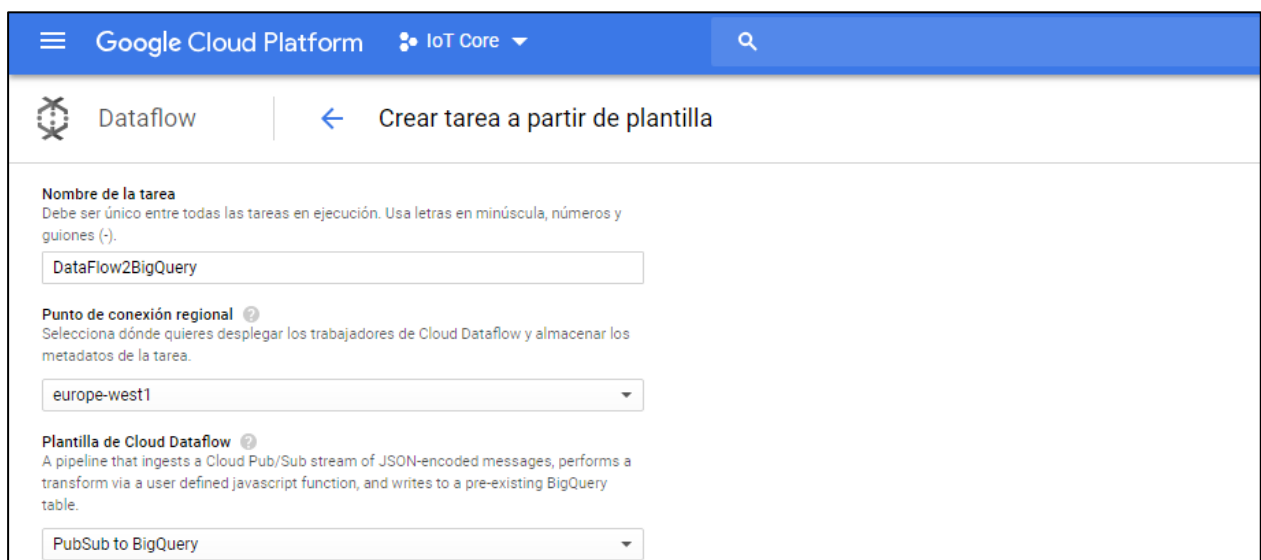


Figura 4-34. Tarea Dataflow.

Continuando con la creación de la tarea hay que definir tres parámetros:

1. **Cloud Pub/Sub input topic:** Es el tema o “topic” definido en el módulo anterior Cloud Pub/Sub del que se recogerán los datos.
2. **BigQuery output table:** Ruta en la que se encuentra la tabla de BigQuery en la que se desea que se escriban los datos. Debe tener el formato “<project>:<dataset>.<table_name>”, donde *project* es el nombre del proyecto, *dataset* es el conjunto de datos definido en BigQuery y *table_name* es el nombre de la tabla creada en el conjunto de datos anterior.
3. **Ubicación temporal:** Ruta para almacenar archivos temporalmente.

Pero antes de asignar los valores a estos parámetros hay que definir el conjunto de datos o “dataset” y crear la tabla en BigQuery.

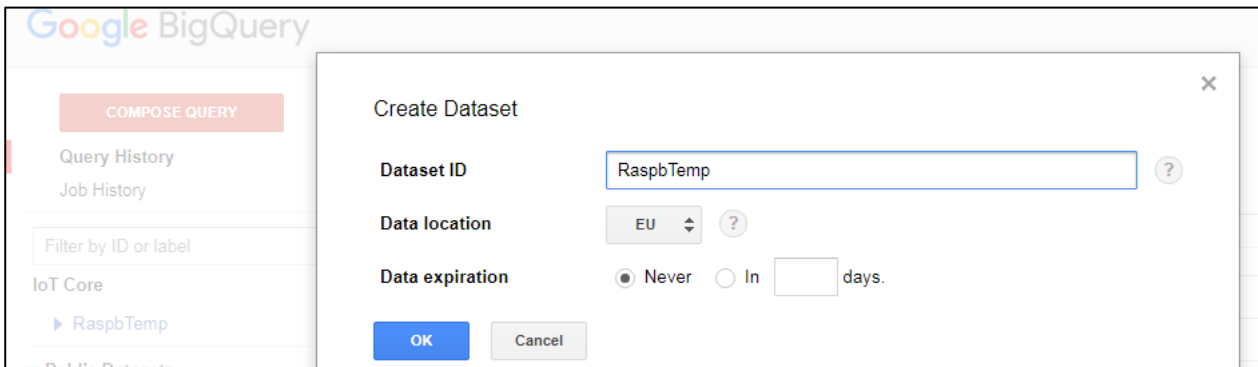


Figura 4-35. Creación dataset en BigQuery.

Y para la creación de la tabla dentro del dataset recién definido se establece un nombre para la misma y además, como columnas de la tabla se utilizan los nombres de los datos [definidos en el Json](#) al enviar los mismos desde el dispositivo, indicando así a la tabla de BigQuery qué debe almacenar.

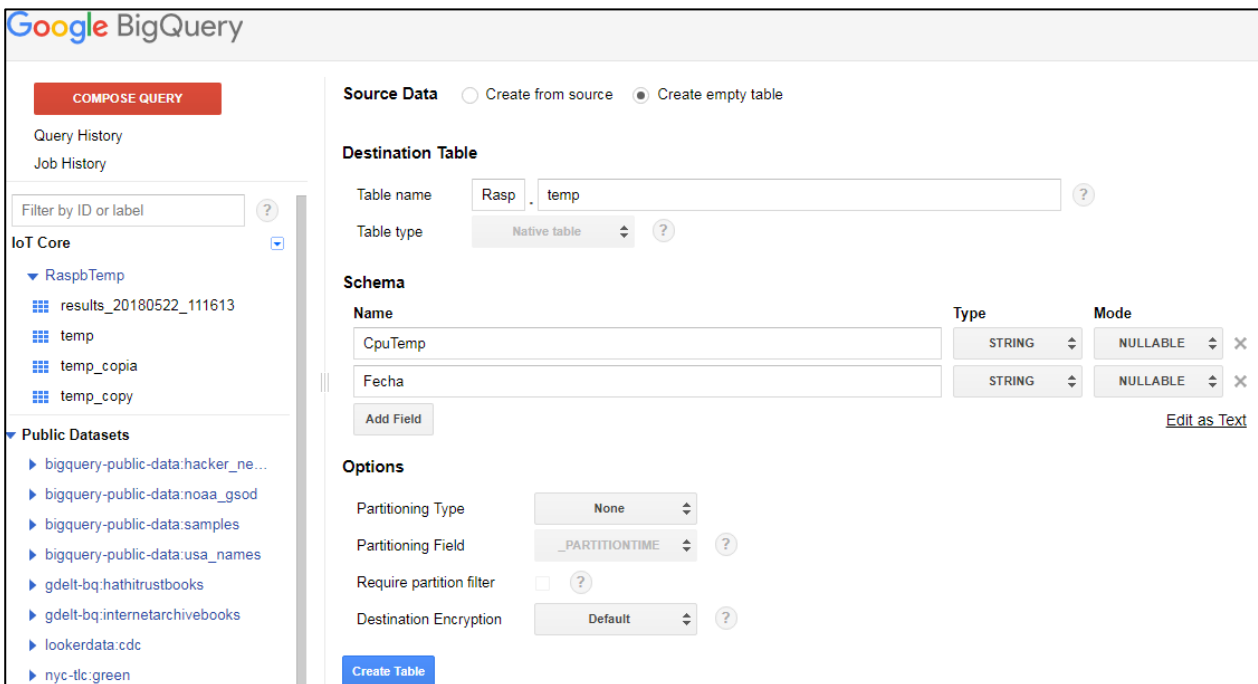


Figura 4-36. Creación tabla en BigQuery.

Para definir el último parámetro, “ubicación temporal”, hay que desplazarse hasta un nuevo módulo, Storage, para crear un segmento. En este caso *datostemp/CpuTemp* ubicado también en la región “europe-west1”.



Figura 4-37. Creación segmento en Storage.

Retomando entonces la configuración de la tarea, se definen los parámetros [expuestos anteriormente](#).

Google Cloud Platform IoT Core

Dataflow ← Crear tarea a partir de plantilla

Parámetros

Cloud Pub/Sub input topic
Cloud Pub/Sub topic to read the input from, in the format of 'projects/<project>/topics/<topic>'

GCS location of your Javascript UDF (Opcional)
The full URL of your .js file. Example: gs://my_bucket/my_function.js

The name of the javascript function you wish to call as your UDF (Opcional)
The function name should only contain letters, digits and underscores. Example: 'transform' or 'transform_udf1'.

BigQuery output table
BigQuery table location (<project>:<dataset>.<table_name>) to write the output to. The table's schema must match the input JSON objects.

Ubicación temporal
Prefijo de ruta y de nombre de archivo para escribir archivos temporales. Ejemplo: gs://MiSegmento/tmp

Figura 4-38. Parámetros tarea Dataflow.

Una vez creada la tarea, se pone en ejecución canalizando todos los datos del tema Cloud Pub/Sub definido hacia la tabla del dataset que sea acaba de crear. Al ejecutar la tarea desde Dataflow se muestra la siguiente información. A la izquierda se ven los pasos que siguen los datos hasta finalmente ser escritos en la tabla de BigQuery, y a la derecha se pueden ver datos de la tarea así como parámetros sobre la métrica.

Google Cloud Platform IoT Core

← dataflow2bigquery REGISTROS

Tarea

Resumen de tareas

Nombre de la tarea	dataflow2bigquery
ID de la tarea	2018-06-01_05_07_03-320769616126134198
Región	europa-west1
Estado de la tarea	En curso <input type="button" value="Detener tarea"/>
Versión del SDK	Apache Beam SDK for Java 2.4.0
Tipo de tarea	Streaming
Fecha y hora de inicio	1 jun. 2018 14:07:04
Tiempo transcurrido	26 s

Ajuste de escala automático
...

Métricas de recursos

vCPU actuales	0
Tiempo total de vCPU	0 vCPU h
Memoria actual	0 B
Tiempo total de memoria	0 GB h

Pipeline Diagram:

```

graph TD
    A[ReadPubsub - En curso] --> B[JavascriptTextTransfor... - En curso]
    B --> C[BigQueryConverters.Js... - En curso]
    C --> D[WriteBigQuery - En curso]
  
```

Figura 4-39. Información tarea Dataflow en ejecución.

Sobre las tareas destacar que si se declaran de tipo “streaming” o envío continuo, aunque no se estén enviando datos se tarifa por cada segundo en ejecución. Por tanto, si se conoce que habrá un periodo de inactividad es recomendable detener la tarea para evitar cargos innecesarios.

Configurado el registro, el dispositivo, el módulo Cloud Pub/Sub, Storage, Dataflow y BigQuery, cuando el cliente realice una transmisión de datos, estarán disponibles para su visualización y almacenaje.

4.3.2.5 Visualización de los datos en Google

Para consultar los datos transmitidos y almacenados se accede a la tabla “temp” desde el módulo BigQuery, donde será necesario realizar una consulta SQL especificando qué se desea visualizar. Para mostrar todo el contenido de la tabla utilizo la siguiente sentencia:



Figura 4-40. Consulta SQL en BigQuery.

Donde se utiliza el parámetro “*” que indica que se desean recoger todos los atributos de la tabla “temp”, la cual pertenece al dataset “RaspbTemp”. Por último añado la condición de que ordene los datos según la fecha mediante “order by Fecha”. Así se listarán los datos de temperatura del sensor, o los enviados por el cliente, ordenados por la fecha en la que fueron creados.

Por tanto el resultado de la consulta es el siguiente:

Row	CpuTemp	Fecha
11	cadena3	Tue May 22 11:09:47 2018
12	cadena4	Tue May 22 11:09:53 2018
13	49.388 Grados Centigrados	Tue May 22 11:10:27 2018
14	48.85 Grados Centigrados	Tue May 22 11:10:29 2018
15	49.925 Grados Centigrados	Tue May 22 11:12:49 2018
16	49.388 Grados Centigrados	Tue May 22 11:12:51 2018
17	49.925 Grados Centigrados	Tue May 22 11:13:30 2018
18	50.464 Grados Centigrados	Tue May 22 11:13:32 2018

Figura 4-41. Visualización de los datos en Google.

BigQuery además ofrece la opción de exportar los datos a un fichero formato CSV o en formato JSON.

4.3.2.6 Costes

Costes de la plataforma Cloud IoT Core por mes y datos consumidos.

Monthly data volume	Price per MB	Registered devices	Minimum charge*
Up to 250 MB	\$0.00	Unlimited, within QPS maximums	1024 bytes
250 MB to 250 GB	\$0.0045	Unlimited, within QPS maximums	1024 bytes
250 GB to 5 TB	\$0.0020	Unlimited, within QPS maximums	1024 bytes
5 TB and above	\$0.00045	Unlimited, within QPS maximums	1024 bytes

Figura 4-42. Costes Cloud IoT Core.

4.3.3 Azure IoT Hub

Microsoft también aporta un producto para el desarrollo de proyectos IoT, Azure IoT Hub. El servicio tiene por objetivo facilitar la conexión y administración de millones de dispositivos y el tratamiento de grandes cantidades de datos generados por los mismos.

Soporta de forma nativa el protocolo MQTT versión 3.1.1, AMQP y HTTPS para las comunicaciones y añade la posibilidad de incorporar IoT Edge, un segundo módulo que proporciona inteligencia artificial a la nube pudiendo por ejemplo operar con los dispositivos sin conexión o conectividad intermitente. [23]

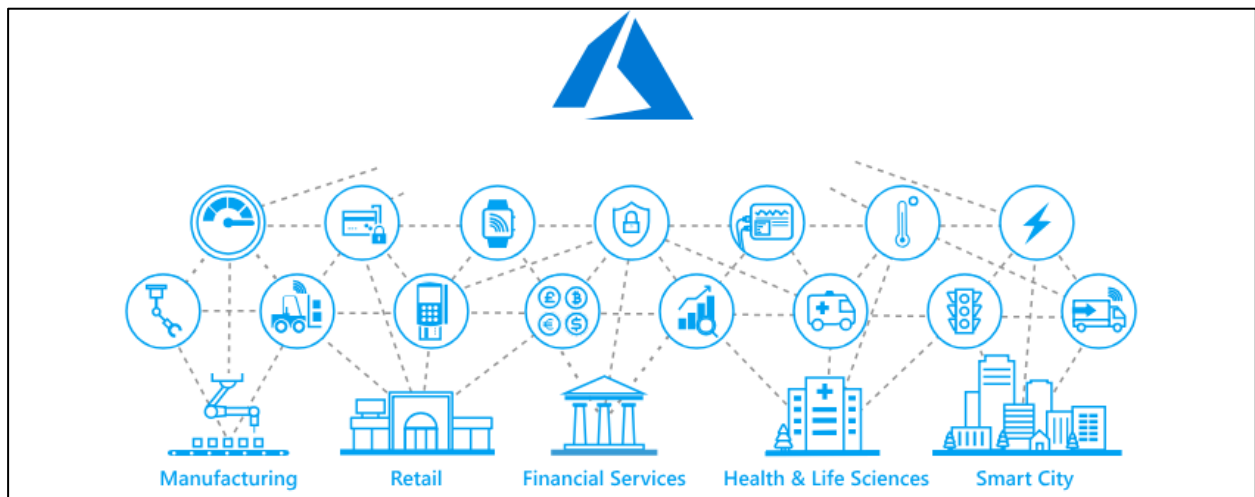


Figura 4-43. Presentación Azure.

Microsoft defiende su plataforma frente al resto asegurando que realiza un cumplimiento normativo más completo que la competencia y que fue el primero de los grandes proveedores de servicio en comprometerse contractualmente con la RPGD o Reglamento General de Protección de Datos. Cuenta con una autorización [FedRAMP](#) que la avala como la nube más confiable para las instituciones del gobierno de los Estados Unidos [24].

En el apartado de seguridad para las comunicaciones ofrece la utilización de certificados de entidad X.509 ya mencionados anteriormente y autenticación mediante Token. No admite comunicaciones no seguras, como el puerto mqtt 1883.

Algunas de las empresas que ya trabajan con la propuesta de Microsoft son ThyssenKrupp, Ecolab o Rockwell Automation entre otras.



Figura 4-44. Algunos clientes de Azure.

4.3.3.1 Cuenta en Azure Portal

Como el resto de plataformas Microsoft ofrece una cuenta gratuita limitada, durante una duración de 12 meses con una tasa máxima de 8000 mensajes al día y 0.5 KB por mensaje. Será necesario proporcionar una tarjeta de crédito para activar los servicios.

Aunque en mi experiencia personal, al cabo de un mes se desactivó la suscripción gratuita y se activó una suscripción por pago automáticamente. Sin embargo manteniendo el límite en las transmisiones especificado de 8000 mensajes y con un almacenamiento ligero de datos parece que no se realiza ningún cargo.

4.3.3.2 Registro de la Raspberry Pi

Antes de conectar un dispositivo es necesario crear el recurso IoT Hub en la plataforma. El proceso mantiene cierta similitud con las anteriores plataformas, se requiere definir un grupo de recursos, establecer la región para el despliegue del servicio y definir a qué suscripción pertenece.

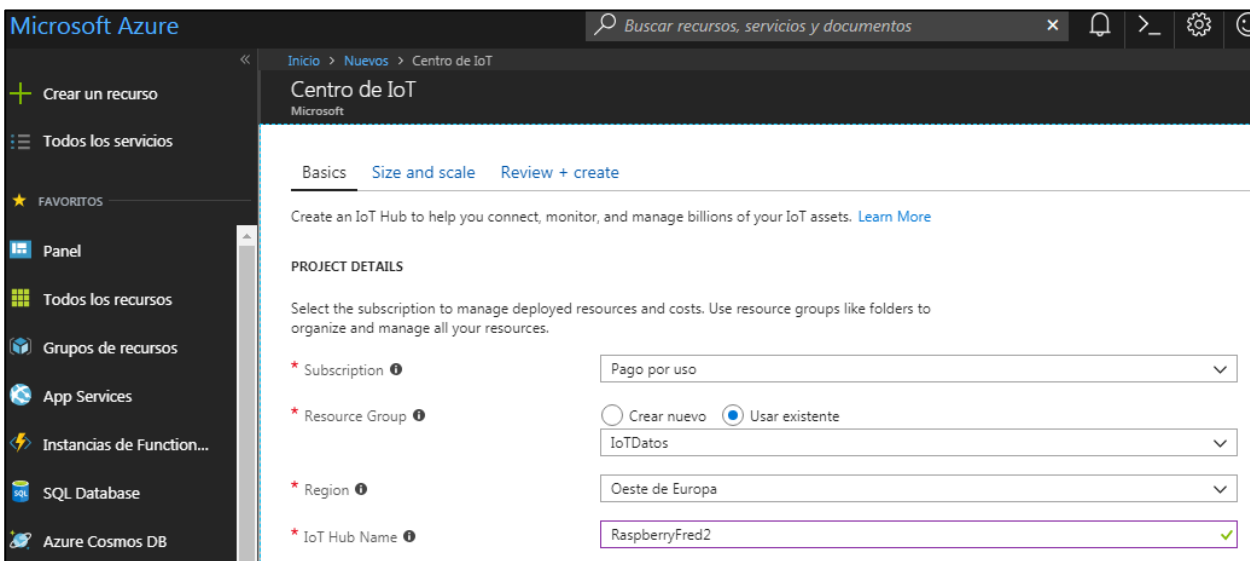


Figura 4-45. Recurso IoT Hub.

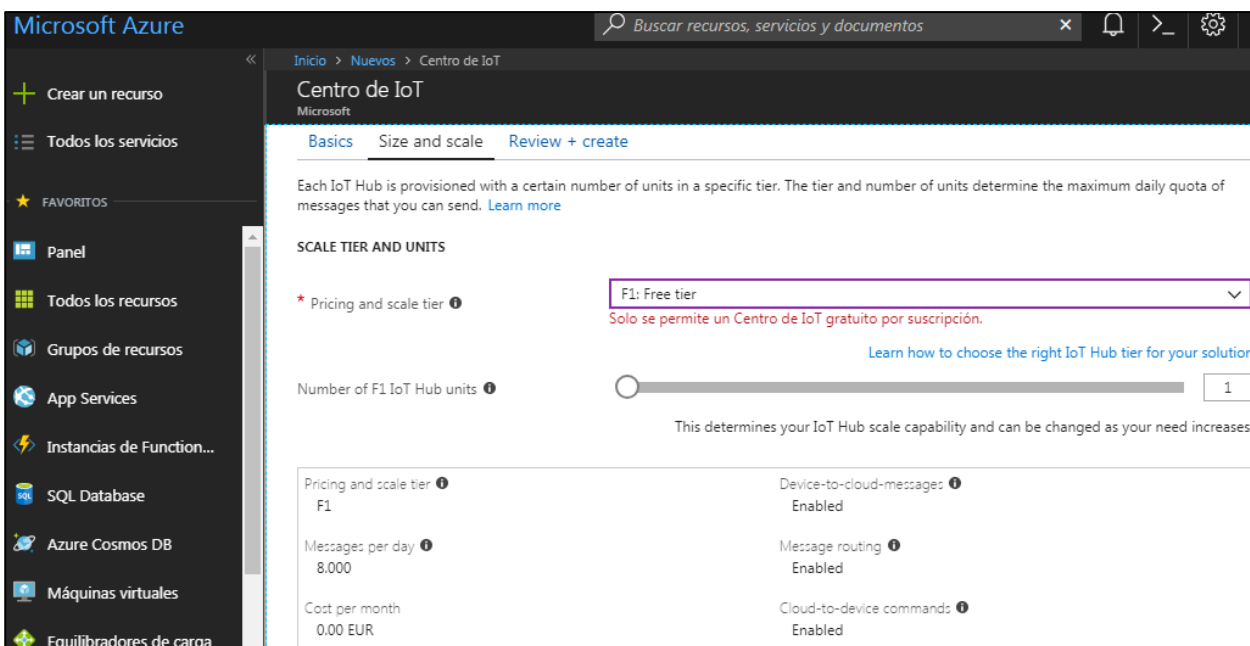


Figura 4-46. Suscripción para el recurso IoT Hub.

Creado el registro se continúa asociando el dispositivo al mismo, tarea que se realiza desde el apartado del registro recién creado. Destacar que se disponen tres opciones a la hora de agregar el dispositivo en cuanto a la seguridad:

- Autenticación mediante clave simétrica.
- Certificados X.509 autofirmados.
- Certificados X.509 firmados por una Autoridad Certificadora (CA).

Para el proyecto se utiliza clave simétrica seleccionando la opción de generar las claves automáticamente. La clave principal generada será necesaria a la hora de transmitir los datos desde el código cliente mqtt, para realizar la conexión.

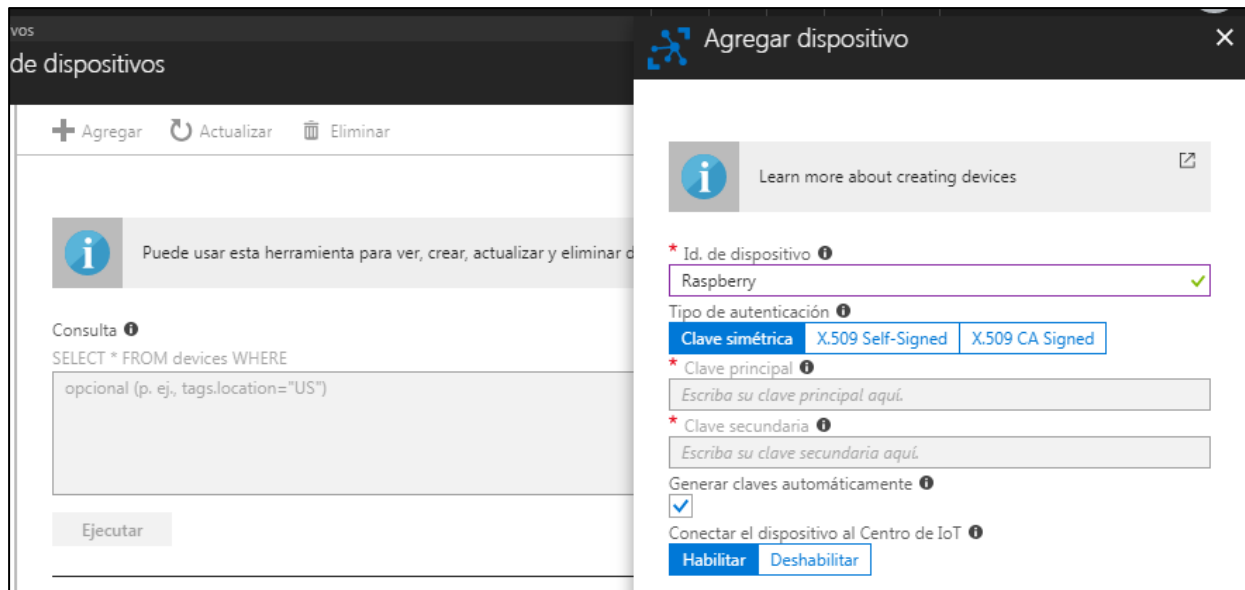


Figura 4-47. Creación del dispositivo y tipo de autenticación.

4.3.3.3 Cuenta de almacenamiento

En este punto estaría configurado el dispositivo y su conexión con la plataforma, para almacenar los datos transmitidos hay que crear un recurso de almacenamiento. De forma similar a la creación del recurso IoT Hub se especifica una región en la que desplegar el servicio y el grupo de recursos al que asociarla.

Entonces se procede a enrutar los mensajes desde el módulo IoT Hub hasta la cuenta de almacenamiento, para ello desde las opciones del registro nos desplazamos a la sección de “Mensajería” donde se encuentra “Puntos de conexión”. Desde ahí se crea un punto de conexión asignando un nombre y el recurso de almacenamiento recién creado, para el proyecto *datostemp/contenedor2*.

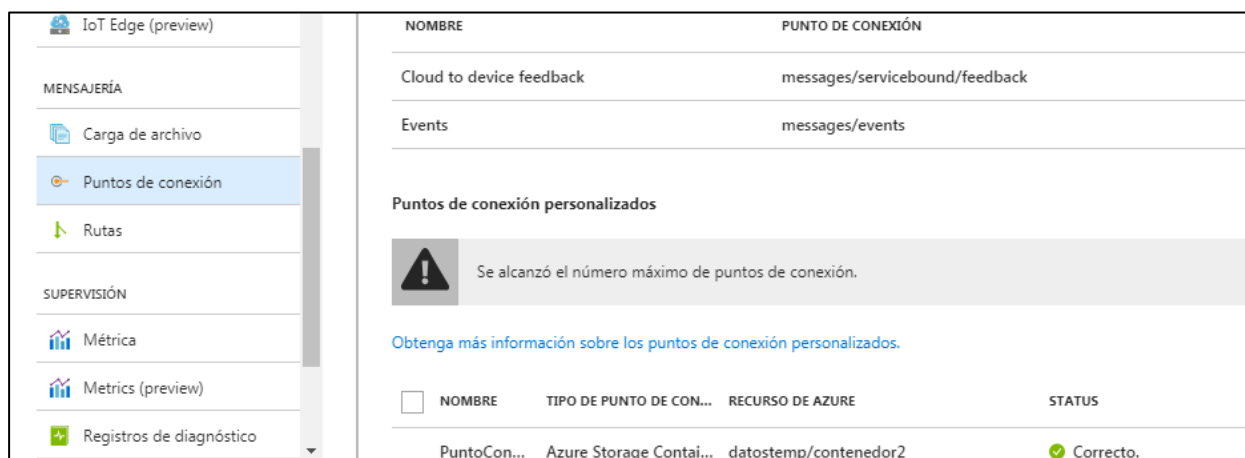


Figura 4-48. Punto de conexión.

Este punto de conexión se utiliza en la Ruta que se define a continuación, donde se establece como origen de los datos “Mensajes del dispositivo” y como punto de conexión el que se acaba de definir, como se muestra en la siguiente figura.

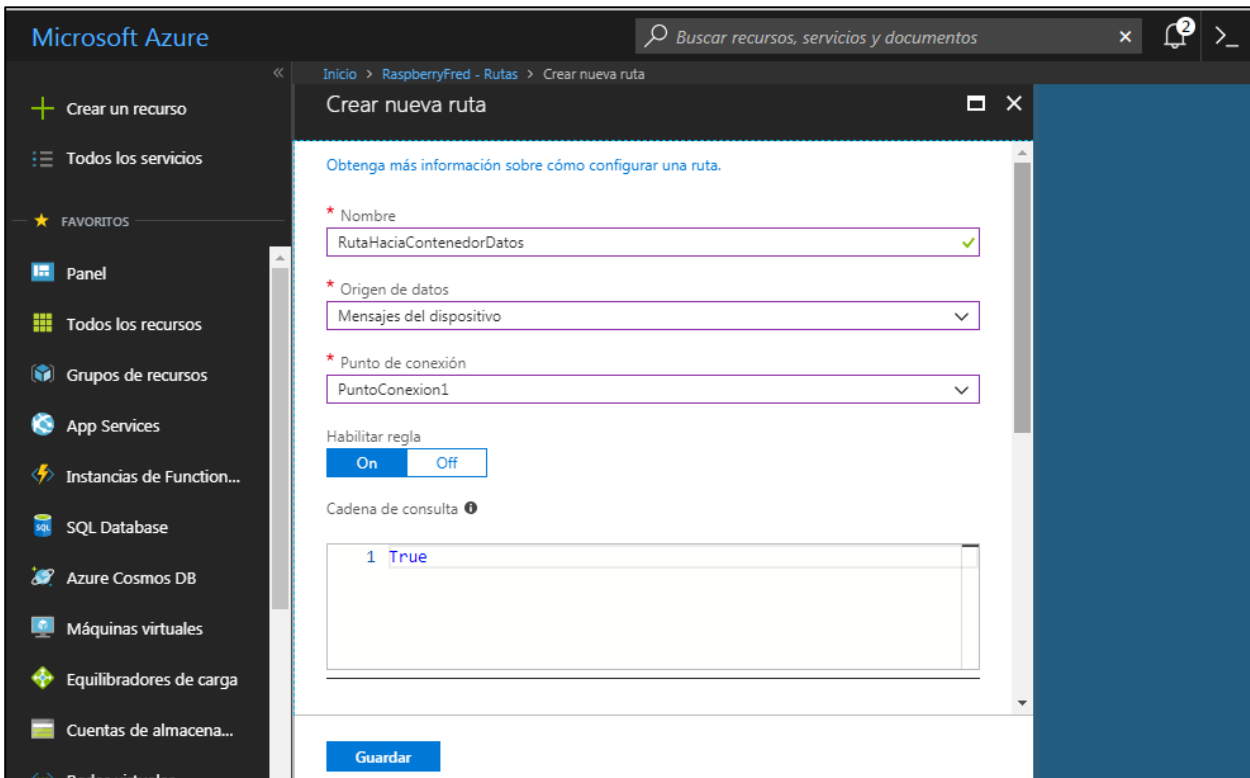


Figura 4-49. Ruta para los datos.

Así pues queda definida la ruta, mediante el punto de conexión, que se encargará de redirigir el tráfico de datos hacia el nuevo recurso de almacenamiento. Falta entonces revisar la configuración del propio dispositivo realizada mediante el cliente mqtt.

4.3.3.4 Cliente MQTT para Microsoft

Continuando con la constante de las plataformas desarrolladas hasta el momento, se utiliza de nuevo un cliente mqtt Paho Eclipse para mantener la máxima similitud posible entre las distintas plataformas en cuanto a la conexión y la transmisión de los datos. Aunque con una serie de diferencias que se comentarán a continuación.

Se comienza como viene siendo habitual con la definición de los atributos necesarios para identificar al dispositivo en la conexión, y definiendo el comportamiento de las funciones “connect”, “disconnect”, “log” y “error_str”. [25]

```
Root_cert = "digicert.cer" # Nombre del certificado Root a usar
device_id = "RaspberryP" # Identificador del dispositivo en la plataforma
Key = "3w0M2dKonKBotF92r4ZR6BTSQ54T0uV9JB8kH" # Clave para la conexión
iot_hub_name = "RaspberryFred" # Nombre del recurso IoT Hub creado desde la plataforma
connflag = False
```

Figura 4-50. Parámetros para conexión con Microsoft.

Se establece el fichero que contiene el certificado, el nombre del dispositivo registrado en la nube, la clave [generada desde la plataforma](#) al registrar el dispositivo y el nombre del recurso IoT Hub.

En cuanto a las funciones, se define un comportamiento similar al de otras plataformas, por ejemplo tras una conexión satisfactoria el valor de rc será 0 y la bandera connflag se actualizará a “True”, permitiendo así la publicación de los datos. Mientras que si se produce un error en la conexión por fallo en la autenticación se mostrará el código 5, tal como se definió en la [tabla de valores](#), y connflag se mantendrá con valor “False”.

La función “error_str” se encargaba de transformar los códigos de errores a formato cadena, para que sean legibles por el usuario.

```
def on_connect(client, userdata, flags, rc):
    print ("Conectado a la nube Azure de Microsoft con resultado: " + error_str(rc))

    # Si no hay errores en la conexión se actualiza la bandera connflag
    if rc == 0:
        global connflag
        connflag = True

def on_disconnect(client, userdata, rc):
    print ("Desconectado de la plataforma con resultado: " + error_str(rc) + "\n")
    global connflag
    connflag = False

# Muestra informacion sobre el intercambio de mensajes, para depurar
def on_log(client, userdata, level, buf):
    print(buf)

# Convierte los errores de paho a cadena
def error_str(rc):
    return '{}: {}'.format(rc, mqtt.error_string(rc))
```

Figura 4-51. Definición funciones para Microsoft.

Continuando con las funciones se define “generate_sas_token” que se necesita para crear el token que se utilizará como contraseña del usuario para la autenticación. [26]

```
##### Creacion del token para autenticar la conexión #####
def generate_sas_token(uri, key, policy_name, expiry):
    ttl = time.time() + expiry #time.time() devuelve los segundos desde las 12:00 am de Enero 1970
    sign_key = "%s\n%d" % ((quote_plus(uri)), int(ttl))
    print sign_key
    signature = b64encode(HMAC(b64decode(key), sign_key, sha256).digest())

    rawtoken = {
        'sr': uri,
        'sig': signature,
        'se': str(int(ttl))
    }

    if policy_name is not None:
        rawtoken['skn'] = policy_name

    return 'SharedAccessSignature ' + urlencode(rawtoken)

#####
```

Figura 4-52. Creación del token para Microsoft.

Definidas estas funciones se finaliza con la función main.

```
def main(subirDatosPropios, datos):

    # Inicializacion del cliente con protocolo mqtt version 3.1.1
    client = mqtt.Client(client_id=device_id, protocol=mqtt.MQTTv311)

    client.on_connect = on_connect
    client.on_disconnect = on_disconnect
    client.on_log = on_log
    client.error_str = error_str

    # Se genera el token permitiendo el acceso al dispositivo indicado como primer parametro,
    # se pasa tambien la clave y el tiempo en el que expirara el token.
    sas_token = generate_sas_token("RaspberryFred.azure-devices.net/RaspberryP", Key, None, 120)
    client.username_pw_set(username=iot_hub_name+".azure-devices.net/" + device_id, password=sas_token)

    #Configuracion tls
    client.tls_set(ca_certs=Root_cert, certfile=None, keyfile=None, cert_reqs=ssl.CERT_REQUIRED,
                  tls_version=ssl.PROTOCOL_TLSv1, ciphers=None)

    # Asegura que se utilicen los certificados
    client.tls_insecure_set(False)

    # Conexion con puerto seguro 8883
    client.connect(iot_hub_name+".azure-devices.net", port=8883)

    # Inicia un hilo en segundo plano para llamar a loop() que procesa los eventos de red
    client.loop_start()
```

Figura 4-53. Inicialización cliente, configuración tls y conexión.

Como con los clientes mqtt ya definidos para otras plataformas se inicializa el cliente, se establece nombre y contraseña del mismo con “username_pw_set”, con la excepción de que se utiliza el token generado como contraseña.

Se realiza la configuración tls mediante los certificados, la conexión mediante “connect” y se define una espera de dos segundos para facilitar la conexión con la plataforma antes de intentar transmitir los datos.

Finalmente en cuanto a la transmisión de los datos se vuelve a distinguir entre dos situaciones según la variable “subirDatosPropios”. Se toman los datos de temperatura o los proporcionados por el usuario y se publican, teniendo en cuenta que el tema de IoT Hub para publicar ha de ser *devices/deviceId/messages/event*. Tras el envío de los datos se produce la desconexión de la plataforma.

```
if subirDatosPropios == 0:
    #Pausa de 2 seg para dar tiempo al establecimiento de conexion
    sleep(2)

    if connflag == True:

        fichero = open("/sys/class/thermal/thermal_zone0/temp") # Se accede al fichero temp, donde se almacenan los valores de
        temp = fichero.read(5)
        temp = float(temp)/1000 # Se divide entre 1000 por el formato
        cadena = str(temp) + " Grados Centigrados" # Se forma la cadena a enviar

        fecha = time.strftime("%c") # Fecha de la consulta (%c) devuelve fecha y hora actual
        datosCadena = {'Datos':cadena, 'Fecha': fecha} # Datos a enviar
        datosJson = json.dumps(datosCadena) # En formato Json

        client.publish("devices/" + device_id + "/messages/events/", datosJson, qos=1)
        print("Datos publicados: " + cadena + " en la fecha " + fecha)
    else:
        print("No se pudo establecer conexion\n")

elif subirDatosPropios == 1:
    #Pausa de 2 seg para dar tiempo al establecimiento de conexion
    sleep(2)

    if connflag == True:

        fecha = time.strftime("%c")
        datosCadena = {'Datos':datos, 'Fecha':fecha} # En este caso los datos son los proporcionados por el usuario
        datosJson = json.dumps(datosCadena)

        client.publish("devices/" + device_id + "/messages/events/", datosJson, qos=1)
        print("Datos del usuario publicados: " + datos + " en la fecha " + fecha)
    else:
        print("No se pudo establecer conexion\n")
```

Figura 4-54. Selección de los datos a enviar.

Al publicar los datos se utiliza la calidad de servicio 1, que es la máxima soportada por la plataforma Azure de Microsoft. [27]

4.3.3.5 Visualización de los datos para Microsoft

Microsoft ofrece varias opciones para visualizar los datos transmitidos a Azure como utilizar Power Bi o Storage Explorer.

Power Bi es un conjunto de herramientas de análisis empresarial enfocado a la gestión de datos para su visualización y análisis, es un servicio de pago pero ofrece una suscripción gratuita válida para dos meses. [28]

Para hacer uso de la misma es necesaria una cuenta de correo profesional, no siendo válidas cuentas de Gmail o Hotmail, para la que utilizo el correo de la Universidad de Sevilla. Mencionar que si no se tiene activado el correo de la Universidad simplemente hay que acceder a <https://identidad.us.es/> y activarlo.

Tras realizar varios intentos de iniciar sesión en Power Bi para visualizar los datos, no hubo éxito a pesar de cumplir los requisitos. La alternativa es utilizar Microsoft Azure Storage Explorer para visualizar los datos transmitidos. Para ello partimos de la base de haber creado la cuenta de almacenamiento y la ruta para canalizar los datos transmitidos desde el dispositivo a la misma, realizado en pasos anteriores.

Este explorador es una aplicación independiente que permite trabajar con los datos de Azure Storage, la cuenta de almacenamiento creada antes, para visualizarlos por ejemplo. Se descarga entonces el software necesario desde la web de Microsoft <https://azure.microsoft.com/es-es/features/storage-explorer/>

Se asocia la cuenta de Azure con el explorador iniciando sesión con la misma, aparecerán entonces los contenedores definidos desde la plataforma con los datos transmitidos. El formato elegido para organizar los datos es año/mes/día/hora. Así pues para una fecha concreta encontramos todos los datos recibidos.

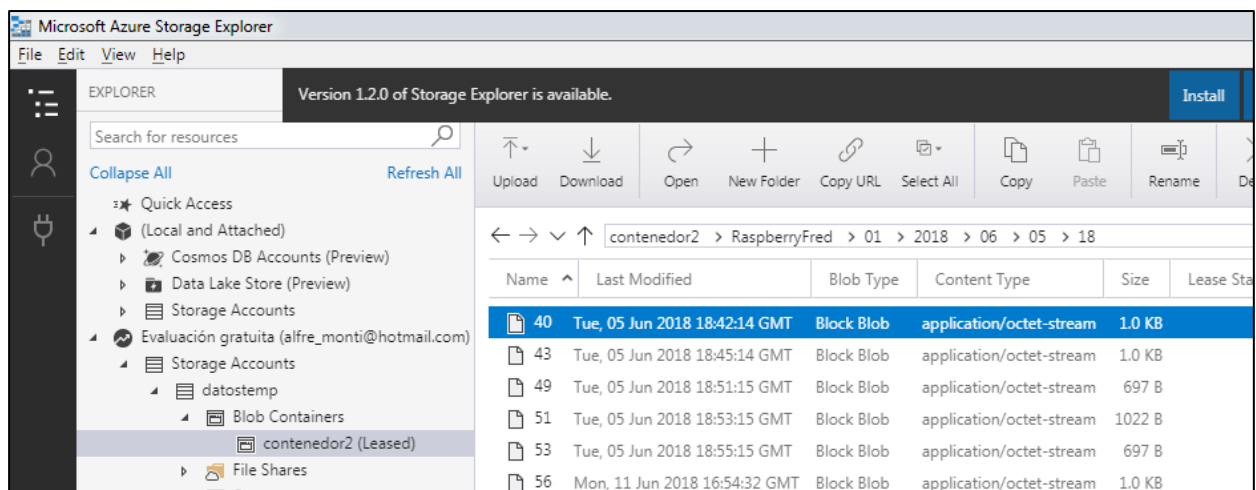


Figura 4-55. Visualización datos Azure.

En cada fichero se encuentran los datos transmitidos correspondientes al minuto de la hora en la que fueron recibidos, lo cual define el nombre del fichero. El problema de esta opción es que la plataforma añade parámetros a los datos transmitidos que dificultan su lectura.

4.3.3.6 Costes

Costes representados por mes, tamaño de los mensajes y cantidad de los mismos.

TIPO DE EDICIÓN	PRECIO POR UNIDAD (AL MES)	NÚMERO TOTAL DE MENSAJES POR DÍA Y POR UNIDAD	TAMAÑO DEL MEDIDOR DE MENSAJES
Gratis	Gratis	8.000	0,5 KB
S1	€21,09	400.000	4 KB
S2	€210,83	6.000.000	4 KB
S3	€2.108,25	300.000.000	4 KB

Figura 4-56. Costes para la plataforma Azure.

4.3.4 Alternativa a las grandes plataformas

Existen multitud de plataformas enfocadas al desarrollo de proyectos IoT en el mercado. Tras haber presentado las tres que actualmente son más importantes, implemento también Carriots para el desarrollo del proyecto.

Carriots es una plataforma española, diseñada para proyectos IoT, surgida tras la escisión de la división M2M de Wairbut, compañía de ingeniería y desarrollo en el campo de las TIC con más de 10 años de experiencia. Miguel Castillo es el CEO de la empresa. Antes de la fundación de Carriots, el éxito de la división M2M fue respaldado por contratos con grandes empresas como Telefónica o Vodafone.

El Ministerio de Economía y Competitividad del Gobierno de España le concedió el título de Pyme Innovadora, válido hasta 31/12/2018.

Esta plataforma ofrece de manera gratuita la gestión de hasta dos dispositivos y un total de 500 tramas al día, de forma indefinida. Soporta los protocolos HTTP y MQTT para establecer la conexión con los dispositivos y una base de datos para almacenar las tramas enviadas. [29]



Figura 4-57. Interfaz principal del panel de control de Carriots.

Un ejemplo de caso de éxito es la ciudad de Pozuelo de Alarcón, que implementa un proyecto de Smart City basado en la plataforma de Carriots para irrigación inteligente en 11 parques, 970 plazas de aparcamiento, gestión del alumbrado público y eficiencia energética gracias a la recolección de datos en más de 150 ubicaciones. [30]

Para adaptarlo a este proyecto se vuelve a hacer uso del cliente MQTT Eclipse Paho, con el que transmitir los datos a la plataforma.

4.3.4.1 Registro del dispositivo

Para registrar la Raspberry Pi se debe tener una cuenta creada previamente, cabe destacar que a diferencia de las anteriores plataformas, no es necesario proporcionar ninguna forma de pago para comenzar a utilizar el servicio.

Una vez iniciada la sesión desde el apartado Gestión de Dispositivos, se encuentra la opción de registrar un nuevo dispositivo. Se proporciona un nombre, una descripción y se activa la opción de utilizar MQTT.

The screenshot shows the Carriots web interface. On the left is a navigation menu with options: Proyectos, Servicios, Grupos, and Assets. The main content area displays the details for a device named 'Raspberry@Freddd.Freddd'. The details include:

- Nombre:** Raspberry
- Descripción:** Valores de la temperatura CPU
- Tipo:** Raspberry Pi
- Sensor:** Other
- Zona horaria:** Europe/Madrid
- Checksum:**
- Frecuencia de tramas de datos:** 1440 minutos
- Frecuencia de tramas de estado:** 1440 minutos
- Grupo:** defaultGroup@Freddd.Freddd
- Estado:** disconnected
- Activo:**
- Id. Developer:** Raspberry@Freddd.Freddd
- Mqtt:**

Figura 4-58. Registro dispositivo en Carriots.

Una vez completado el registro antes de proseguir con la configuración del código, rescatamos la ApiKey desde el apartado de configuración. Esta Apikey será necesaria para la autorización al enviar los datos.

The screenshot shows the 'Listado de Apikeys' page in the Carriots web interface. It features a table with the following columns: Apikey, Descripción, Usuario, Por Defecto, and Activo. The table contains three entries:

Apikey	Descripción	Usuario	Por Defecto	Activo
0655cf16f8c2d21794a8922b9bb03f93c0ea79658fa368dd45217b...	Automatic Apikey Read Only	Freddd	<input type="checkbox"/>	<input checked="" type="checkbox"/>
485602dff14f32631f1b29228872837c4fe24174a98f1ae6053bbc5...	Automatic Apikey Full	Freddd	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5cd2acb206debb9d6e7bd30e904333ee503c8448f570cd33c2fcd...	Automatic Apikey Stream On ly	Freddd	<input type="checkbox"/>	<input checked="" type="checkbox"/>

At the bottom of the table, it indicates 'Mostrando registros del 1 al 3 de un total de 3 registros'.

Figura 4-59. Apikey Carriots.

4.3.4.2 Cliente MQTT para Carriots

Para esta última plataforma también se utiliza el cliente mqtt de Eclipse, aunque en esta ocasión al utilizar de base el código que proporciona la plataforma, la estructura del programa presenta algunos cambios. [31]

Se empieza creando la clase CarriotsMqttClient, encargada de inicializar los parámetros necesarios para la conexión con la plataforma así como de publicar los datos. Para dicha publicación en lugar de utilizar la función habitual *Publish* se utiliza *Single*, cuyo objetivo es simplemente publicar los datos y desconectarse sin realizar más acciones.

Además se configura el topic en el que publicar que debe ser la Apikey seguida de “streams” o “status”, los dos topics permitidos por Carriots, y se revisa la configuración tls en caso de activarse.

```
import paho.mqtt.publish as publish
from json import dumps
from ssl import PROTOCOL_TLSv1
from time import sleep

class CarriotsMqttClient():
    host = 'mqtt.carriots.com'
    port = '1883'
    auth = {}
    topic = '%s/streams'
    tls = None

    def __init__(self, auth, tls=None):
        self.auth = auth
        self.topic = '%s/streams' % auth['username'] # Topic en el que se publican los datos
        if tls:
            self.tls = tls
            self.port = '8883'

    def publish(self, msg):
        try:
            publish.single(topic=self.topic, payload=msg, hostname=self.host, auth=self.auth, tls=self.tls, port=self.port)
        except Exception, ex:
            print ex
```

Figura 4-60. Clase CarriotsMqttClient().

A continuación se encuentra la función main que recibe los dos parámetros habituales para escoger los datos a transmitir. Se define la variable *auth* que contiene la Apikey utilizada como nombre de usuario y un campo de contraseña vacío.

Entonces se comprueba la opción elegida por el usuario, si desea subir los datos de temperatura se procede de forma similar a las anteriores plataformas. Se accede al fichero para obtener los datos y se publican, en este caso se realizan cuatro mediciones. La espera de 0.5 seg entre publicaciones se utiliza para dar tiempo al sensor a obtener nuevos valores.

```
def main(subirDatosPropios, datos):
    auth = {'username': '485602dff14f32631f1b29228872837c4fe24174a9[redacted]79', 'password':''}

    if subirDatosPropios == 0: # El cliente decide leer los datos de temperatura
        for i in range(1,5):
            fichero = open("/sys/class/thermal/thermal_zone0/temp") # Se abre el fichero temp, que almacena la temperatura de la cpu
            temp = fichero.read(5) # Se leen los 5 primeros caracteres
            temp = float(temp)/1000 # Se divide entre 1000 por cuestiones de formato
            cadena = str(temp) + " Grados Centigrados"
            mensaje = {'protocol': 'v2', 'device': 'Raspberry@Freddd.Freddd', 'at': 'now', 'data': cadena}
            client_mqtt = CarriotsMqttClient(auth=auth) # Se inicializa el Cliente MQTT con la apikey del usuario
            client_mqtt.publish(dumps(mensaje)) # Publica los datos
            print("Temperatura " + str(temp) + "°C publicada\n")
            sleep(0.5)
```

Figura 4-61. Datos de temperatura en Carriots.

Finalmente se encuentra la opción de subir los datos propios del usuario para lo que se configura el mensaje a publicar, se inicializa el cliente y se transmite el mensaje.

```

else:
    print("Subiendo datos a la nube...\n")
    datosRecibidos = datos
    mensaje = {'protocol': 'v2', 'device': 'Raspberry@Freddd.Freddd', 'at': 'now', 'data': datosRecibidos}
    client_mqtt = CarriotsMqttClient(auth=auth) # Inicializacion del cliente MQTT
    client_mqtt.publish(dumps(mensaje)) # Publicacion de los datos

    print("'" + datosRecibidos + "'" + " ha sido publicado\n")
if __name__ == '__main__':
    main(LeerDatos, datos)

```

Figura 4-62. Datos del usuario en Carriots.

En este caso como se ha utilizado la función *Single* para publicar en lugar de *Publish*, la desconexión se realiza de forma automática y por tanto no es necesario llamar a la función *disconnect* de Paho.

Esta plataforma soporta únicamente la calidad de servicio 0 o “at most once”. [32]

4.3.4.3 Visualización de los datos

En la parte izquierda de la interfaz de Carriots se encuentra el módulo “Datos” desde donde accedemos al subpartado “Tramas de Datos”. A diferencia de las anteriores plataformas, este paso resulta mucho más sencillo. No es necesario crear reglas para enrutar los datos desde un módulo de recepción a otro de gestión o análisis, ni configurar la base de datos. Aunque esto puede limitar una configuración más compleja facilita el uso a clientes con un perfil menos técnico.

Se muestran entonces todas las tramas enviadas desde la Raspberry Pi en este caso, ordenadas por la fecha de envío. Cada trama mantiene un registro del dispositivo que la envió y finalmente se encuentra el campo que contiene los datos de cada trama.

Así pues se muestra un ejemplo de visualización en la siguiente figura.

	Fecha	Dispositivo	Datos
<input type="checkbox"/>	2018/06/05 20:56:31	Raspberry@Freddd.Freddd	"Cadena Global"
<input type="checkbox"/>	2018/06/05 20:56:00	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:56:00	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:55:59	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:55:59	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:53:28	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:53:28	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:53:28	Raspberry@Freddd.Freddd	"52.078 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:53:27	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"
<input type="checkbox"/>	2018/06/05 20:51:29	Raspberry@Freddd.Freddd	"51.54 Grados Centigrados"

Mostrando registros del 1 al 10 de un total de 317 registros

Figura 4-63. Visualización de los datos desde Carriots.

Tras ejecutar el cliente haciendo uso de las dos opciones de las que dispone, se observan en el listado de los datos recibidos tanto los valores de temperatura de la CPU del dispositivo, como cadenas definidas por el propio usuario.

4.3.4.4 Costes

Precios representados por mes y dispositivo.

GRATIS	CORPORATE	LITE	NUBE PRIVADA BAJO DEMANDA
GRATIS (NO HACE FALTA TARJETA)	2 €* AL MES POR DISPOSITIVO	0,50 €* AL MES POR DISPOSITIVO	Contacte CON NOSOTROS
PARA PRUEBAS Y PROTOTIPOS	DISPOSITIVOS QUE ENVÍAN HASTA 1 MB POR DÍA ESTABLECIENDO MUCHAS CONEXIONES	MUCHOS DISPOSITIVOS QUE ENVÍAN POCO VOLUMEN DE DATOS. EJ. SMART METER, MANTENIMIENTO REMOTO	PARA CONEXIONES ILIMITADAS, ALMACENAMIENTO DE DATOS, USO PRIVADO Y USO PERSONALIZADO
Número mín. dispositivos 1	Número mín. dispositivos 11	Número mín. dispositivos 100	Número mín. dispositivos Contacte con nosotros
Número max. dispositivos 2	Número max. dispositivos ilimitado	Número max. dispositivos ilimitado	Número max. dispositivos ilimitado**
API KEYS 4	API KEYS 100	API KEYS 10	API KEYS ilimitado**
Max. tramas aceptadas 500 tramas por día 10 tramas por minuto	Max. tramas aceptadas 1500 x num. dispositivos por día 50 x num. dispositivos por minuto + INFO	Max. tramas aceptadas 25 x num. dispositivos por día 5 x num. dispositivos por minuto + INFO	Max. tramas aceptadas ilimitado**

Figura 4-64. Costes para la plataforma Carriots.

4.4 Comparativa de las plataformas

En esta sección se realiza una comparación entre las distintas plataformas, según las características principales que se han ido revelando a lo largo del documento.

Tabla 4-2. Comparativa de las características de las plataformas.

Plataformas	Amazon	Google	Microsoft	Carriots
Máxima QoS soportada	1	1	1	0
Métodos de encriptación	TLSv1.2	TLSv1.2	TLSv1.2	TLSv1
Métodos de autenticación	Certificados X.509 IAM (Identity and Access Management) users Amazon Cognito identities Identidades federadas	Clave pública/privada mediante JSON Web Token Cloud Identity and Access Management (IAM) roles	Clave simétrica Certificados X.509 autofirmados Certificados X.509 firmados por CA (Autoridad Certificadora) SAS token (firma de acceso compartido)	Clave pública/privada La propia Api key proporcionada por la plataforma Certificados de autoridades certificadoras (CA)
Versión de MQTT implementada	3.1.1	3.1.1	3.1.1	No especificado*
Protocolos soportados	MQTT HTTP MQTT sobre WebSockets	HTTP MQTT	MQTT AMQP HTTPS	HTTP MQTT
Cuota de mercado**	62%	12%	20%	----
Creación de la plataforma	2006	2011	2010	2011

La autenticación Identity and Access Management (IAM) consiste en un servicio web que facilita la gestión de identidades, para controlar quién está autenticado y de qué permisos dispone.

Amazon Cognito es un servicio de este tipo, propio de Amazon, que permite llevar a cabo un control de acceso e inicios de sesión de usuarios a gran escala. Al igual que el método de autenticación **Cloud identity**, en este caso propio de Google.

*Al inicializar el cliente mediante la función Client() de Eclipse Paho, si no se especifica la versión mqtt, por defecto toma la versión 3.1.

**Según analistas del [KeyBanc](#), banco de Estados. [33]

Se realiza esta vez una comparativa del coste de cada plataforma frente al resto en función de unos parámetros comunes, que son los mensajes al día permitidos y el tamaño de los mismos, todo en relación a un dispositivo y para la región de Europa.

Tabla 4-3. Comparativa de costes para las plataformas.

Plataformas	Mensajes/día permitidos	Tamaño de los mensajes	Coste mensual
Google	400.000	5 KB	8€ /dispositivo
Amazon	400.000	5 KB	11,95 € /dispositivo
Microsoft	400.000	4 KB	21,09 € /dispositivo
Carriots	1.500	10 KB	2€ / dispositivo**

Cálculos utilizados:

- Google: $400.000 \text{ Mensajes/día} * 0.005 \text{ MB/mensaje} * 0.0045\$/\text{MB} = 9 \$$ (aproximadamente 8 €).
- Amazon: Se utiliza la calculadora que proporciona Amazon, con los parámetros 400.000 mensajes/día y un tamaño de 5 KB por mensaje. Resultado 14\$, que son aproximadamente 11,95€.
- Microsoft: Datos obtenidos directamente de la [documentación de Azure](#).
- Carriots: Datos recogidos de la [documentación de Carriots](#). ** Carriots exige un mínimo de 11 dispositivos para la tarifa de 2€ / dispositivo. Lo que termina aumentando el coste hasta los 22€ mensuales.

5 SERVICIO EN CONJUNTO

La Física es el Sistema Operativo del Universo

- Steven R Garman -

Tras desarrollar cada parte del proyecto por separado, en este nuevo y último capítulo se define un nuevo servicio con el objetivo de aunar cada una de los módulos para conseguir una funcionalidad global con el proyecto en una sola aplicación, para obtener un funcionamiento más dinámico y sencillo de ejecutar enfocado con la perspectiva de un posible cliente final.

El objetivo por tanto de este nuevo código Python es crear un servicio desde el que poder hacer uso de cada una de las plataformas por separado, desde un mismo sitio, e incluso utilizar todas las plataformas a la vez a modo de disponer de un respaldo de los datos por ejemplo.

Esto ofrece al cliente la posibilidad de escoger la plataforma que esté más acorde con las necesidades de su proyecto así como de su capacidad financiera.

5.1 Desarrollo del código servicio

El código de cada plataforma se encuentra en un directorio distinto, para poder importarlos desde el módulo servicio hay que incluir en cada directorio un archivo vacío denominado “__init.py__”. Este archivo hará que el directorio pase a ser un paquete y por tanto quede disponible para ser importado.

También se importa *time* para poder utilizar *sleep* que permite realizar pausas en el programa.

```
from Carriots import carriots
from Amazon import amazon
from Google import google
from Microsoft import microsoft

from time import sleep
```

Figura 5-1. Importación de los paquetes.

Cuando el cliente ejecuta la aplicación se muestra una serie de mensajes que indican las posibilidades que se ofrecen, quedando a la espera de la respuesta por teclado del usuario.

Tras mostrar por pantalla las opciones se inicializa la variable *entrada* a cero, variable que recogerá la opción escogida por el usuario. De igual forma se inicializa la variable *leerDatos* a uno, así por defecto se leerían los datos del sensor y por tanto *datos* se inicializa como cadena vacía.

Como última observación de la figura mostrada a continuación, se encuentra el bucle *while* en el que se permanece mientras el usuario no introduzca el valor correspondiente con la salida.

```
def main():
    print("\nT e m p e r a t u r a   d e   l a   C P U\n")
    print("Elija un Servicio en la Nube para monitorizar la temperatura CPU de su dispositivo:\n")
    print("Pulse 1 para subir los datos a AWS Amazon\n"+
          "Pulse 2 para subir los datos a Carriots\n"+
          "Pulse 3 para subir los datos a Azure de Microsoft\n"+
          "Pulse 4 para subir los datos a Google IoT Core\n"+
          "Pulse 5 para subir los datos a todas las nubes mencionadas\n"+
          "Pulse 6 para visualizar de nuevo las opciones\n" +
          "\n7 para Salir\n")

    entrada = 0          #Recoge la opcion escogida por el usuario mediante entrada estandar "input()"
    leerDatos=1         #Variable para establecer si se desean leer los datos del sensor o enviar otros
    datos=""
    while entrada != 7:
        entrada = input("Opcion numero: ")
        print("\n")
```

Figura 5-2. Mensajes iniciales.

Una vez dentro del bucle while y con la opción seleccionada por el usuario, se compara ésta con las posibles entradas.

Si es una opción comprendida entre 1 y 4, asociadas a cada plataforma IoT, se solicita al cliente que especifique qué parte del servicio desea utilizar. Es decir si quiere realizar una lectura de los datos de temperatura de la CPU de la Raspberry, y subirlos a la nube seleccionada o por el contrario desea subir su propia cadena.

Si decide subir los datos del sensor de temperatura se llama a la función main del módulo que contiene la plataforma seleccionada, con las opciones elegidas. Por el contrario si decide enviar su propia cadena se solicita por teclado mediante “raw_input”, que permite introducir espacios, y se llama entonces a la función main con los parámetros establecidos.

Se muestra como representación de la estructura seguida para las cuatro primeras opciones, la implementación de las dos primeras.

```
while entrada != 7:
    entrada = input("Opcion numero: ")
    print("\n")

    if entrada == 1:
        subirDatosPropios = input("Desea subir sus propios datos a la nube?\nPulse 1 para Si --- Pulse 0 para No: ")

        if subirDatosPropios == 0:
            datos=""
            amazon.main(subirDatosPropios, datos)
            # Se subiran los datos del sensor
            # En este caso 'datos' se envia como cadena vacia
            # Metodo main de aws_iot_pub para subir datos a la nube de Amazon

        elif subirDatosPropios == 1:
            datos = raw_input("Introduzca la cadena a enviar:")
            amazon.main(subirDatosPropios, datos)
            # Se llama al metodo main de aws_iot_pub para subir datos del
            # usuario a la nube de Amazon

        else:
            print("Respuesta con formato incorrecto, seleccione de nuevo un servicio\n")
            print("\n")

    elif entrada == 2:
        subirDatosPropios = input("Desea subir sus propios datos a la nube?\nPulse 1 para Si --- Pulse 0 para No: ")

        if subirDatosPropios == 0:
            datos=""
            carriots.main(subirDatosPropios, datos)
            # Metodo main de carriotsMQTT para subir datos del sensor a la nube d

        elif subirDatosPropios == 1:
            datos = raw_input("Introduzca la cadena a enviar:")
            carriots.main(subirDatosPropios, datos)
            # Metodo main de carriotsMQTT para subir datos del usuario a la nube

        else :
            print("Respuesta con formato incorrecto, seleccione de nuevo un servicio\n")
            print("\n")
```

Figura 5-3. Selección de plataforma por el cliente.

Si el cliente introdujese una opción distinta a 0 o 1 se notificaría por pantalla que la respuesta ha sido errónea.

La opción 5 actúa de forma similar en cuanto a las dos opciones para la elección de los datos a transmitir, pero en esta ocasión en lugar de subir los datos a una plataforma se suben a todas. Entre llamadas a la función main de cada paquete se realiza una espera de un segundo gracias a *sleep*, para proporcionar tiempo suficiente a la transición de una plataforma a otra y evitar así que se superpongan los mensajes impresos por pantalla.

Así pues para el caso de transmisión de datos de temperatura:

```
elif entrada == 5:
    subirDatosPropios = input("Subir datos propios?\nPulse 1 para Si --- Pulse 0 para No: ")

    if subirDatosPropios == 0:
        datos = ""

        #Subiendo datos a la nube de Amazon
        print("-----AMAZON-----\n")
        amazon.main(subirDatosPropios, datos)
        sleep(1)

        #Subiendo datos a la nube de Google
        print("-----GOOGLE-----\n")
        google.main(subirDatosPropios, datos)
        sleep(1)

        #Subiendo datos a la nube de Microsoft
        print("-----MICROSOFT-----\n")
        microsoft.main(subirDatosPropios, datos)
        sleep(1)

        #Subiendo datos a la nubre de Carriots
        print("-----CARRIOTS-----\n")
        carriots.main(subirDatosPropios, datos)
        sleep(1)

        print("\n")
```

Figura 5-4. Transmisión datos temperatura a todas las plataformas.

Mientras que para la opción de subir los datos propios.

```
elif subirDatosPropios == 1:

    datos = raw_input("Introduzca la cadena a enviar:")

    #Subiendo datos a la nube de Amazon
    print("-----AMAZON-----\n")
    amazon.main(subirDatosPropios, datos)
    sleep(1)

    #Subiendo datos a la nubre de Carriots
    print("-----CARRIOTS-----\n")
    carriots.main(subirDatosPropios, datos)
    sleep(1)

    #Subiendo datos a la nube de Google
    print("-----GOOGLE-----\n")
    google.main(subirDatosPropios, datos)
    sleep(1)

    #Subiendo datos a la nube de Microsoft
    print("-----MICROSOFT-----\n")
    microsoft.main(subirDatosPropios, datos)
    sleep(1)

    print("\n")
else:
    print("Respuesta con formato incorrecto, seleccione de nuevo un servicio\n")
```

Figura 5-5. Transmisión datos propios a todas las plataformas.

Finalizando con el servicio se incluye por comodidad como opción 6 volver a imprimir todas las opciones disponibles y finalmente si el cliente introduce 0, se finalizará la ejecución.

```
elif entrada == 6:
    print("Pulse 1 para subir los datos a AWS Amazon\n"+
          "Pulse 2 para subir los datos a Carriots\n"+
          "Pulse 3 para subir los datos a Azure de Microsoft\n"+
          "Pulse 4 para subir los datos a Google IoT Core\n"+
          "Pulse 5 para subir los datos a todas las nubes mencionadas\n"+
          "Pulse 6 para visualizar de nuevo las opciones\n" +
          "\n0 para Salir\n")
elif entrada==7:
    print("Saliendo...")
else:
    print("Opcion seleccionada no valida!, seleccione un servicio de la lista\n")
```

Figura 5-6. Recordatorio de las opciones disponibles.

CONCLUSIONES

Cada día parece más claro que las plataformas IoT han llegado para quedarse, y las cifras de volumen de negocio que están generando lo confirman. Como se ha visto en el capítulo 4, se espera que estas plataformas generen un volumen de 3.000 millones de euros para 2021 frente a los 610 ya generados en 2015. Sin duda aportan un servicio cada vez más demandado y la competencia en este ámbito es tan amplia que en mi opinión favorecerá el continuo desarrollo de estos servicios.

La evolución que ha tenido el protocolo MQTT desde su creación en IBM, hasta convertirse en un estándar abierto de OASIS y finalmente en un estándar ISO, hace de este protocolo un sólido candidato para este ámbito y prueba de ello es el soporte que todas las plataformas ofrecen para el mismo.

Para concluir con este documento me gustaría destacar que, en cuanto a mi experiencia personal, ha sido un reto empezar en el campo *Internet of Things* pues no tenía experiencia y partía de cero. Sin embargo ha resultado muy gratificante ya que me ha servido para tener una buena perspectiva sobre el ámbito de trabajo de IoT y realmente creo que la experiencia adquirida puede resultarme muy útil en el futuro próximo, por ser afín al campo en el que estoy interesado.

El proyecto me ha prestado la oportunidad no sólo de conocer mejor este ámbito sino de explorar también la multitud de posibilidades que ofrecen los dispositivos de bajo coste como la Raspberry Pi, que unido a los sensores y las plataformas IoT forman un conjunto con una gran potencial para multitud de proyectos, que pueden aportar un valor realmente positivo para la sociedad, desde pequeños proyectos de Smart Cities hasta grandes proyectos de BigData.

Otro de los retos a los que me he enfrentado surgió tras documentarme sobre cada una de las plataformas, antes de comenzar con el desarrollo de los clientes MQTT, pues tuve que decidir en qué lenguaje de programación lo haría. Python resultaba el más extendido entre las documentaciones de las distintas plataformas y me permitiría mantener cierta homogeneidad entre todos los clientes, lo que hizo decidirme por este lenguaje a pesar de ser nuevo para mí.

Finalmente considero importante destacar que, según mi experiencia a lo largo de estos meses, es un campo que se está desarrollando muy rápido gracias al incremento constante de nuevos dispositivos conectados a Internet y esto implica que las plataformas están sometidas a cambios continuamente. Esto sugiere que se debe mantener una conexión constante con la información sobre nuevas actualizaciones, que pueden implicar algunos cambios en el proyecto. Para respaldar esta última afirmación proporciono en el Anexo C, algunos correos de notificación recibidos de las plataformas.

REFERENCIAS

- [1] Mqtt sobre OSI. Ermesh [Imagen] Disponible en: <http://www.ermesh.com/aprender-protocolo-mqtt-parte-1/>
- [2] Origen protocolo mqtt. IBM [En línea] Disponible en: <https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html>
- [3] ISO Standard [En línea] Disponible en: <https://www.iso.org/standard/69466.html>
- [4] Mqtt version 3.1.1. OASIS [En línea] Disponible en: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>
- [5] Differences-between-3.1.0-and-3.1.1. Github. [Tabla] Disponible en: <https://github.com/mqtt/mqtt.github.io/wiki/Differences-between-3.1.0-and-3.1.1>
- [6] Etcher [En línea] Disponible en: <https://etcher.io/>
- [7] Recomendación microSD [En línea] Disponible en: <https://www.raspberrypi.org/documentation/installation/sd-cards.md>
- [8] Paho Client. Eclipse [En línea] Disponible en: <https://www.eclipse.org/paho/clients/python/docs/>
- [9] CA cert. IBM [En línea] Disponible en: https://www.ibm.com/support/knowledgecenter/es/SS3JSW_5.2.0/com.ibm.help.security_523.doc/SI_ca_cert.html
- [10] Dennis Smith, Lydia Leong, Raj Bala. *Magic Quadrant for Cloud Infrastructure as a Service*. Gartner [Imagen] Disponible en: <https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519&st=sb&aliId=1154870580>
- [11] Redacción Byte. Plataformas IoT generarán 3.000 millones de euros en 2021. En: *Internet de las cosas*. 27 Octubre 2017 [En línea] Disponible en: <https://www.revistabyte.es/actualidad-byte/plataformas-iot-3-000-millone/>
- [12] Álvaro Cárdenas. Secmotic [Imagen] Disponible en: <https://secmotic.com/blog/plataforma-iot/>
- [13] IoT Core doc. Amazon [En línea] Disponible en: https://aws.amazon.com/es/iot-core/?nc2=h_iot
- [14] AWS developer guide. Amazon [En línea] Disponible en: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html>
- [15] QoS support. Amazon [En línea] Disponible en: <https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html>
- [16] IoT Core. Google [En línea] Disponible en: <https://cloud.google.com/iot-core/?hl=es>
- [17] IoT Core developer guide. Google [En línea] Disponible en: <https://cloud.google.com/iot/docs/quickstart?hl=es>
- [18] Mqtt example. Google [En línea] Disponible en: https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/iot/api-client/mqtt_example/cloudiot_mqtt_example.py
- [19] Quality of service. Google [En línea] Disponible en: https://cloud.google.com/iot/docs/how-tos/mqtt-bridge#quality_of_service_qos
- [20] BigQuery. Google [En línea] Disponible en: <https://cloud.google.com/bigquery/>
- [21] Cloud Pub/Sub. Google [En línea] Disponible en: <https://cloud.google.com/pubsub/docs/>
- [22] Dataflow. Google [En línea] Disponible en: <https://cloud.google.com/dataflow/>

- [23] About IoT Hub. Microsoft [En línea] Disponible en: <https://docs.microsoft.com/es-es/azure/iot-hub/about-iot-hub>
- [24] Azure vs AWS. Microsoft [En línea] Disponible en: <https://azure.microsoft.com/es-es/overview/azure-vs-aws/>
- [25] IoT Hub mqtt. Microsoft [En línea] Disponible en: <https://docs.microsoft.com/es-es/azure/iot-hub/iot-hub-mqtt-support>
- [26] Security token. Microsoft [En línea] Disponible en: <https://docs.microsoft.com/es-es/azure/iot-hub/iot-hub-devguide-security#security-tokens>
- [27] Quality of service. Microsoft [En línea] Disponible en: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support#using-the-mqtt-protocol-directly>
- [28] Power Bi. Microsoft [En línea] Disponible en: <https://docs.microsoft.com/es-es/power-bi/power-bi-overview>
- [29] Carriots [En línea] Disponible en: <https://www.carriots.com/nosotros/sobre-carriots>
- [30] Casos de éxito. Carriots [En línea] Disponible en: <https://www.carriots.com/casos-de-exito/pozuelo-smart-city>
- [31] Send stream. Carriots [En línea] Disponible en: https://www.carriots.com/tutorials/send_stream/mqtt
- [32] Calidad de servicio. Carriots [En línea] Disponible en: <https://www.carriots.com/documentation/mqtt>
- [33] Market share. CNBC [En línea] Disponible en: <https://www.cnbc.com/2018/01/12/amazon-lost-cloud-market-share-to-microsoft-in-the-fourth-quarter-keybanc.html>

GLOSARIO

DBaaS: Data Base as a Service	iii
ISO: International Organization of Standardization	1
IEC: International Electrotechnical Commission	1
MQTT: Message Queue Telemetry Transport	1
RFC: Request for Comments	2
UTF: Unicode Transformation Format	2
QoS: Quality of Service	3
CRL: Certificate Revocation List	16
OCSP: Online Certificate Status Protocol	16
TLS: Transport Layer Security	16
CPU: Central Processing Unit	17
M2M: Machine to Machine	22
HTTP: Hypertext Transport Protocol	23
UIT: Unión Internacional de las Telecomunicaciones	23
UIT-T: Sector de Normalización de las Telecomunicaciones de la UIT	23
JSON: JavaScript Object Notation	28
KB: Kilobyte	32
SQL: Structured Query Language	33
CSV: Comma Separated Values	42
AMQP: Advanced Message Queuing Protocol	43
HTTPS: Hypertext Transport Protocol Secure	43
TIC: Tecnologías de Información y Comunicación	50
CEO: Chief Executive Officer	50
IAM: Identity and Access Management	55

ANEXO A

En este anexo se exponen los pasos a seguir para ejecutar el programa. Al estar el servicio completo recogido en un mismo fichero Python la puesta en marcha del servicio es sencilla. El resto de operaciones que se pueden realizar están disponibles de forma dinámica, desde el programa en ejecución de forma simplificada a través del teclado.

Una vez descargado el directorio que contiene el código es importante no modificar la ubicación de los subdirectorios y ficheros. Cada servicio que permite conectarse a una plataforma está ubicado en un directorio con el nombre de la misma, y los certificados que permiten estas conexiones están en el directorio Certificados.

En el fichero servicio.py, perteneciente al directorio principal, se encuentra el código que permite conectarse con cualquiera de las plataformas desde un mismo sitio.

En primer lugar hay que abrir un terminal y desplazarse hasta el directorio principal en el que se encuentra el script servicio.py. Para ejecutar el script es necesario descargar Python en la versión con la que se ha desarrollado el proyecto que es la 2.7.13 disponible mediante el comando:

```
sudo apt-get install python2.7
```

También es necesario tener instalado el cliente Paho de Eclipse detallado en el [capítulo 3](#), disponible mediante el siguiente comando:

```
pip install paho-mqtt
```

Si el anterior comando resultase en error, utilizar el siguiente y volver a ejecutar el anterior:

```
sudo apt-get install python-pip
```

Por último antes de ejecutar el script servicio.py es necesario utilizar el siguiente comando para poder generar el JWT necesario en la plataforma de Google:

```
pip install PyJWT
```

Ahora sí, instalado todo lo anterior, ya es posible ejecutar el programa para lo que se utiliza el comando:

```
python servicio.py
```

La interfaz principal debe ser similar a la siguiente:

```
T e m p e r a t u r a   d e   l a   C P U
Elija un Servicio en la Nube para monitorizar la temperatura CPU de su dispositivo:
Pulse 1 para subir los datos a AWS Amazon
Pulse 2 para subir los datos a Carriots
Pulse 3 para subir los datos a Azure de Microsoft
Pulse 4 para subir los datos a Google IoT Core
Pulse 5 para subir los datos a todas las nubes mencionadas
Pulse 6 para visualizar de nuevo las opciones

7 para Salir
Opcion numero: █
```

Figura Anexo a-1. Interfaz principal del servicio.

ANEXO B

Durante el desarrollo de esta memoria se ha visto para cada plataforma, el resultado final de enviar los datos a la nube, es decir su visualización. Se incluye este anexo para plasmar el funcionamiento desde la parte del cliente, para lo que se proporcionan unas capturas que muestran la interacción con el mismo para cada plataforma.

- **Interacción cliente Amazon:**

Si el cliente decide subir los datos de temperatura a la plataforma de Amazon el resultado es el siguiente:

```

Temperatura de la CPU
Elija un Servicio en la Nube para monitorizar la temperatura CPU de su dispositivo:
Pulse 1 para subir los datos a AWS Amazon
Pulse 2 para subir los datos a Carriots
Pulse 3 para subir los datos a Azure de Microsoft
Pulse 4 para subir los datos a Google IoT Core
Pulse 5 para subir los datos a todas las nubes mencionadas
Pulse 6 para visualizar de nuevo las opciones
7 para Salir
Opcion numero: 1

Desea subir sus propios datos a la nube?
Pulse 1 para Si --- Pulse 0 para No: 0

Conectando a AWS Amazon Web Services
Resultado de la conexion: 0: No error.

Mensaje enviado: temperatura 47.236 Grados Centigrados

Resultado de la desconexion con la plataforma: 0: No error.
Desconectado

Opcion numero: █

```

Figura Anexo b-1. Interacción cliente Amazon.

Donde se va notificando sobre las fases de la conexión así como del resultado de la misma, se muestra el mensaje que se ha enviado y se notifica la desconexión con el resultado de ésta.

- **Interacción cliente Amazon en modo depuración.**

En esta situación se muestran los mensajes MQTT intercambiados con la plataforma, lo cual se ha utilizado para ir comprobando que el paso de mensajes era correcto. Para ello se hace uso de la función “on_log” utilizada en los clientes MQTT de Eclipse Paho.

```

Opcion numero: 1

Desea subir sus propios datos a la nube?
Pulse 1 para Si --- Pulse 0 para No: 0
Sending CONNECT (u0, p0, wr0, wq0, wf0, c1, k60) client_id=
Received CONNACK (0, 0)

Conectando a AWS Amazon Web Services
Resultado de la conexion: 0: No error.

Sending PUBLISH (d0, q1, r0, m1), 'raspberry/temperatura/cpu', ... (75 bytes)
Mensaje enviado: temperatura 47.774 Grados Centigrados

Sending DISCONNECT
Resultado de la desconexion con la plataforma: 0: No error.
Desconectado

Opcion numero: █

```

Figura Anexo b-2. Interacción cliente Amazon depuración.

Continuando con el resto de plataformas para una interacción con el cliente final:

- **Interacción cliente Carriots:**

```
Pulse 1 para subir los datos a AWS Amazon
Pulse 2 para subir los datos a Carriots
Pulse 3 para subir los datos a Azure de Microsoft
Pulse 4 para subir los datos a Google IoT Core
Pulse 5 para subir los datos a todas las nubes mencionadas
Pulse 6 para visualizar de nuevo las opciones

0 para Salir

Opcion numero: 2

Desea subir sus propios datos a la nube?
Pulse 1 para Si --- Pulse 0 para No: 0

Temperatura 47.236°C publicada
Temperatura 47.236°C publicada
Temperatura 47.236°C publicada
Temperatura 47.236°C publicada

Opcion numero: █
```

Figura Anexo b-3. Interacción cliente Carriots.

En este caso no se observa ningún tipo de comprobación sobre los mensajes publicados, lo que se debe al hecho de que la calidad de servicio máxima para esta plataforma es qos=0.

- **Interacción cliente Azure:**

```
Temperatura de la CPU
Elija un Servicio en la Nube para monitorizar la temperatura CPU de su dispositivo:
Pulse 1 para subir los datos a AWS Amazon
Pulse 2 para subir los datos a Carriots
Pulse 3 para subir los datos a Azure de Microsoft
Pulse 4 para subir los datos a Google IoT Core
Pulse 5 para subir los datos a todas las nubes mencionadas
Pulse 6 para visualizar de nuevo las opciones

7 para Salir

Opcion numero: 3

Desea subir sus propios datos a la nube?
Pulse 1 para Si --- Pulse 0 para No: 0

Conectado a la nube Azure de Microsoft con resultado: 0: No error.
Datos publicados: 46.16 Grados Centigrados en la fecha Tue Jul 10 08:40:26 2018
Desconectado de la plataforma con resultado: 0: No error.

Opcion numero: █
```

Figura Anexo b-4. Interacción cliente Azure.

- **Interacción cliente Google:**

```
Temperatura de la CPU
Elija un Servicio en la Nube para monitorizar la temperatura CPU de su dispositivo:
Pulse 1 para subir los datos a AWS Amazon
Pulse 2 para subir los datos a Carriots
Pulse 3 para subir los datos a Azure de Microsoft
Pulse 4 para subir los datos a Google IoT Core
Pulse 5 para subir los datos a todas las nubes mencionadas
Pulse 6 para visualizar de nuevo las opciones
7 para Salir
Opcion numero: 4
Desea subir sus propios datos a la nube?
Pulse 1 para Si --- Pulse 0 para No: 0
Creando el JWT usando el algoritmo RS256 y el fichero Certificados/rsa_private-Google.pem
Conectando a Google IoT Core
Resultado: 0: No error.
Datos '{"Fecha": "Tue Jul 10 08:44:57 2018", "Cputemp": "46.16 Grados Centigrados"}' publicados
Desconectando de la plataforma
Resultado: 0: No error.
Opcion numero: █
```

Figura Anexo b-5. Interacción cliente Google.

Como particularidad sobre las anteriores plataformas, antes de conectar con Google IoT Core se notifica la creación del JWT necesario para la conexión. Posteriormente se comunica el resultado de la conexión así como los datos publicados y la desconexión con la plataforma.

ANEXO C

En este anexo se muestran las notificaciones de las plataformas sobre cambios que podrían afectar el funcionamiento de la misma.

- Notificación de cambios en Amazon:

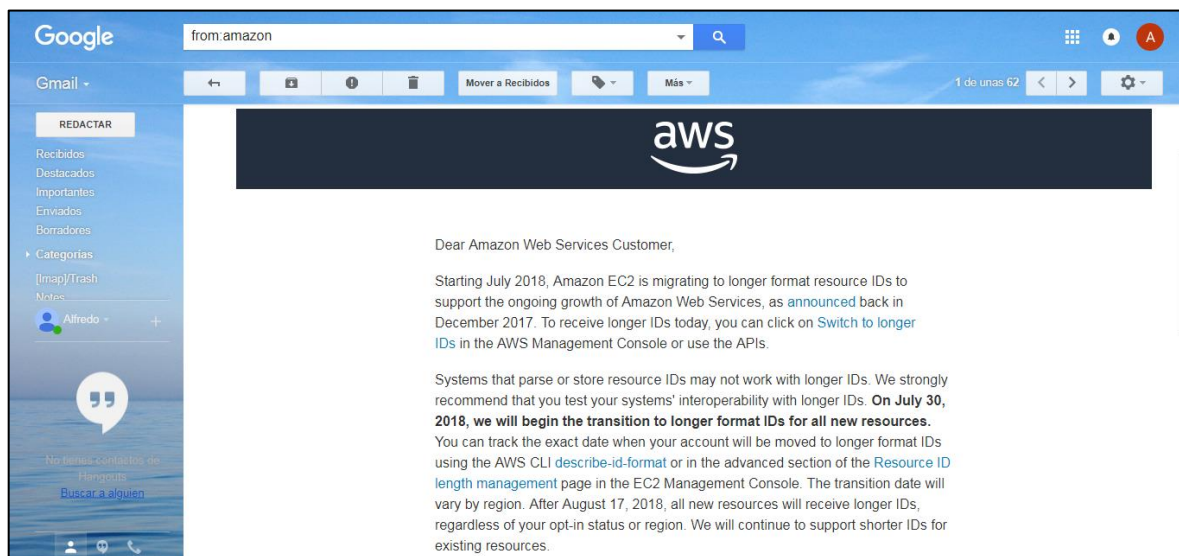


Figura Anexo c-1. Notificación Amazon.

- Notificación plataforma Carriots:

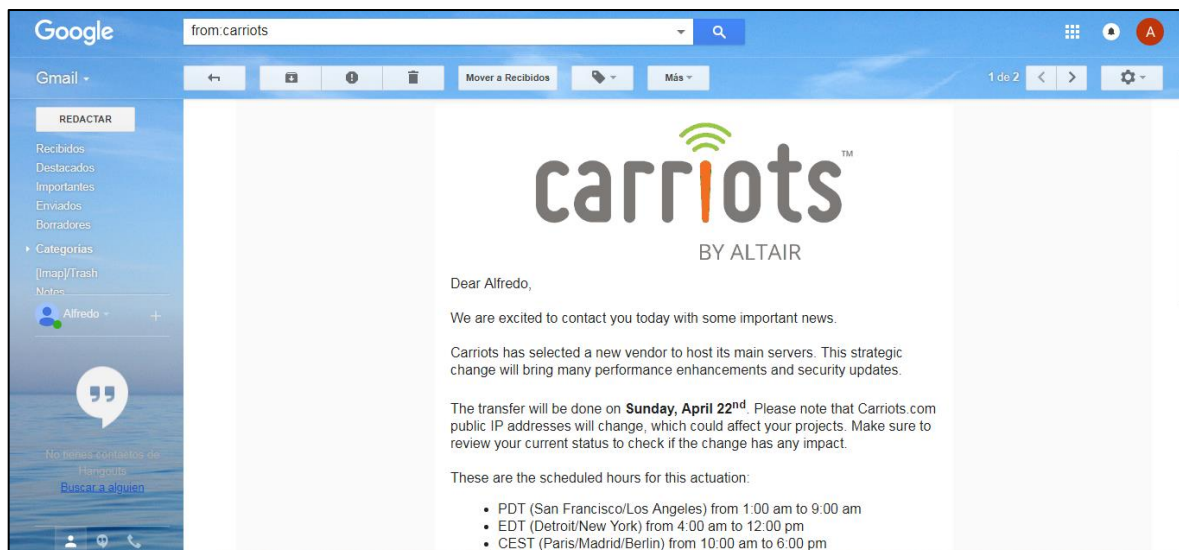


Figura Anexo c-2. Notificación Carriots.

- Notificación plataforma Google:

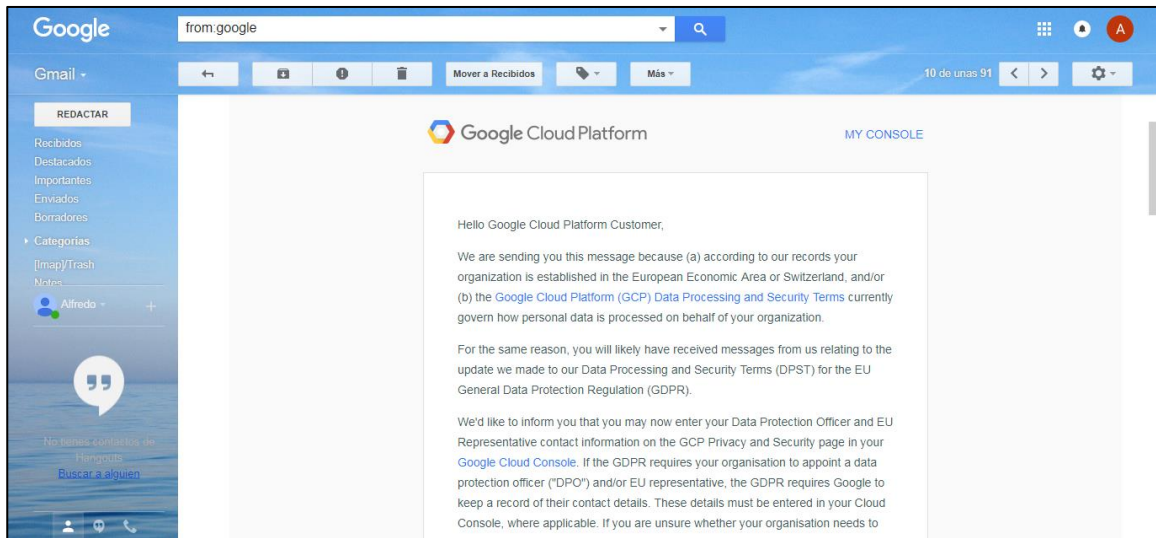


Figura Anexo c-3. Notificación Google.

ANEXO D

En este último anexo se adjunta todo el código desarrollado para el proyecto, empezando por el de las plataformas y finalizando con el código del servicio que aúna la funcionalidad de las mismas.

Código para la plataforma de Amazon:

```

/home/pi/Documents/Plataformas/PruebaConjunto/Amazon/amazon.py
Página 1 de 3 mar 10 jul 2018 10:10:09 CEST
1  #!/usr/bin/python
2
3  import paho.mqtt.client as mqtt
4  import ssl
5  import time
6  from time import sleep
7
8  import json
9
10 # Bandera estado de la conexion
11 connflag = False
12
13 # Cuando se recibe un CONNACK desde la plataforma
14 def on_connect(client, userdata, flags, rc):
15     print("\n")
16     # Se notifica que la conexion se ha realizado
17     print("Conectando a AWS Amazon Web Services")
18     print("Resultado de la conexion: " + error_str(rc) + "\n")
19
20     # Se comprueba la conexion ha tenido exito
21     if rc == 0:
22         # Bandera para notificar conexion
23         # establecida
24         global connflag
25         connflag = True
26
27 # Cuando el cliente se desconecta de la plataforma
28 def on_disconnect(client,userdata, rc):
29     # Un valor distinto de 0 indica desconexion inesperada
30     if rc != 0:
31         print("Desconexion inesperada\n")
32
33     print("Resultado de la desconexion con la plataforma: " + error_str(rc))
34     global connflag
35     connflag = False
36     print("Desconectado\n")
37
38 # Muestra informacion sobre el intercambio de mensajes, util solo para desarrollador
39 def on_log(client, userdata, level, buf):
40     print(buf)
41
42 # Convierte los codigos de error de paho a formato cadena para hacerlos legibles
43 # por el usuario
44 def error_str(rc):
45     return '{}: {}'.format(rc, mqtt.error_string(rc))
46
47 def main(subirDatosPropios, datos):
48     # Se inicializa el cliente mqtt version 3.1.1
49     mqttc =
50     mqtt.Client(protocol=mqtt.MQTTv311)
51
52     # Asignacion a on_connect
53     mqttc.on_connect = on_connect
54
55     # Asignacion a on_disconnect
56     mqttc.on_disconnect = on_disconnect
57
58     # Asignacion a on_log, util para depurar no para el cliente
59     mqttc.on_log = on_log
60
61     # Asignacion a error_str
62     mqttc.error_str = error_str
63
64     ##### Parametros para establecer conexion #####
65     # Direccion de la plataforma
66     awshost = "amgbu8oes223z.iot.eu-west-2.amazonaws.com"
67     awsport = 8883
68     clientId = "Raspberry"
69     # Puerto
70     # Id del cliente

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/Amazon/amazon.py
```

```
Página 2 de 3
```

```
mar 10 jul 2018 10:10:09 CEST
```

```

67     thingName = "Raspberry"                                # Nombre del objeto
68
69     # Ruta hasta el certificado Root-CA
70     caPath = "Certificados/root-CA.crt"
71
72     # Ruta hasta el certificado .cert.pem
73     certPath = "Certificados/Raspberry.cert.pem"
74
75     # Ruta hasta la clave privada .private.key
76     keyPath = "Certificados/Raspberry.private.key"
77
78     ### Configuracion tls ###
79     mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath,
80                  cert_reqs=ssl.CERT_REQUIRED,
81                  tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
82
83     # Para conectarse con aws
84     mqttc.connect(awshost, awsport, keepalive=60)
85
86     # Inicia un hilo en segundo plano para llamar a loop() que procesa los eventos
87     # de red
88     mqttc.loop_start()
89
90     # ----Se envian los datos de
91     # temperatura----#
92     if subirDatosPropios == 0:
93         # Pausa de dos seg para dar tiempo a la conexion
94         sleep(2)
95
96         if connflag == True:
97
98             # Se accede al fichero temp, donde se almacenan los valores de
99             # temperatura
100            fichero = open("/sys/class/thermal/thermal_zone0/temp")
101
102            temp = fichero.read(5)
103
104            # Se divide entre 1000 por el formato
105            temp = float(temp)/1000
106
107            # Se forma la cadena a enviar
108            cadena = str(temp) + " Grados Centigrados"
109
110            # Fecha de la consulta, %c para formato fecha y hora
111            fecha = time.strftime("%c")
112
113            # Datos a enviar
114            datosCadena = {'Datos':cadena, 'Fecha': fecha}
115
116            # Se convierte a formato Json para permitir el acceso a los datos desde
117            # la base de datos "dynamoDB" de Amazon
118            datosJson = json.dumps(datosCadena)
119
120            # Se publican los datos en la nube con qos = 1
121            # o calidad de servicio, indica que el mensaje se manda "al menos una
122            # vez"
123            mqttc.publish("raspberry/temperatura/cpu", datosJson, qos=1)
124
125            # Se imprime por pantalla el mensaje publicado
126            print("Mensaje enviado: temperatura " + cadena + "\n")
127
128        else:
129            print("No se pudo establecer la conexion\n")
130
131     # ----Se envian los datos pasados por el usuario----#
132     else:
133         # Pausa de seg seg para esperar la conexion
134         sleep(2)
135
136         if connflag == True:

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/Amazon/amazon.py
Página 3 de 3 mar 10 jul 2018 10:10:09 CEST
132
133     fecha = time.strftime("%c")
134
135     # En este caso "datos" contiene la cadena pasada por el usuario
136     datosCadena = {'Datos':datos, 'Fecha': fecha}
137     datosJson = json.dumps(datosCadena)
138
139     mqttc.publish("raspberrry/temperatura/cpu", datosJson, qos=1)
140
141     print("Mensaje '" + datos + "' se ha publicado en la nube de Amazon\n")
142
143     else:
144         print("No se ha conseguido establecer conexion\n")
145
146     # Desconexion con la plataforma
147     mqttc.disconnect()
148
149     # Espera de dos seg para desconexion
150     sleep(2)
151
152 if __name__ == '__main__':
153     main(subirDatosPropios, datos)
154
```

Código para Carriots:

```

/home/pi/Documents/Plataformas/PruebaConjunto/Carriots/carriots.py
Página 1 de 2 mar 10 jul 2018 10:04:54 CEST
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import paho.mqtt.publish as publish
5  from json import dumps
6  from ssl import PROTOCOL_TLSv1
7  from time import sleep
8
9  class CarriotsMqttClient():
10
11     ##### Parametros para la conexion #####
12     host = 'mqtt.carriots.com'
13     port = '1883'
14     auth = {}
15     topic = '%s/streams'
16     tls = None
17     ##### Fin parametros para conexion #####
18
19     def __init__(self, auth, tls=None):
20         self.auth = auth
21         self.topic = '%s/streams' % auth['username'] # Topic en el que se
22         # publican los datos
23         if tls: # Si se utiliza tls
24             self.tls = tls
25             self.port = '8883'
26
27     def publish(self, msg):
28         try:
29             publish.single(topic=self.topic, payload=msg, hostname=self.host,
30                 auth=self.auth, tls=self.tls, port=self.port)
31         except Exception, ex:
32             print ex
33
34     def main(subirDatosPropios, datos):
35
36         # Para autenticar al usuario
37         auth = {'username':
38             '485602dff14f32631f1b29228872837c4fe24174a98f1ae6053bbc5a6006cf79',
39             'password': ''}
40
41         # El cliente decide leer los datos de temperatura
42         if subirDatosPropios == 0:
43             for i in range(1,5):
44                 # Se abre el fichero temp, que almacena la temperatura de la cpu
45                 fichero = open("/sys/class/thermal/thermal_zone0/temp")
46
47                 # Se leen los 5 primeros caracteres
48                 temp = fichero.read(5)
49                 # Se divide entre 1000 por cuestiones de formato
50                 temp = float(temp)/1000
51                 cadena = str(temp) + " Grados Centigrados"
52
53                 mensaje = {'protocol': 'v2', 'device': 'Raspberry@Freddd.Freddd',
54                     'at': 'now', 'data': cadena}
55
56                 # Se inicializa el Cliente MQTT con la apikey del usuario
57                 client_mqtt = CarriotsMqttClient(auth=auth)
58
59                 # Publica los datos
60                 client_mqtt.publish(dumps(mensaje))
61
62                 print("\n" + "Temperatura " + str(temp) + "°C publicada")
63                 sleep(0.5)
64             else:
65                 print("Subiendo datos a la nube...\n")

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/Carriots/carriots.py
Página 2 de 2 mar 10 jul 2018 10:04:54 CEST
66     datosRecibidos = datos
67     mensaje = {'protocol': 'v2', 'device': 'Raspberry@Freddd.Freddd', 'at':
68               'now', 'data': datosRecibidos}
69     # Inicializacion del cliente MQTT
70     client_mqtt = CarriotsMqttClient(auth=auth)
71     # Publicacion de los datos
72     client_mqtt.publish(dumps(mensaje))
73
74     print(""" + datosRecibidos + """ + " ha sido publicado\n")
75
76
77
78 if __name__ == '__main__':
79     main(LeerDatos, datos)
80
```

Código Google:

```

/home/pi/Documents/Plataformas/PruebaConjunto/Google/google.py
Página 1 de 3 mar 10 jul 2018 10:20:44 CEST
1  #!/usr/bin/env python
2
3  import datetime
4  import ssl
5  import time
6  import json
7
8  from time import sleep
9
10 import jwt
11 import paho.mqtt.client as mqtt
12
13 ##### Variables para establecer la conexion con Google Cloud IoT Core #####
14
15 project_id = "iot-core-200510" # Id del proyecto
16 cloud_region = "europe-west1" # Define la region de los
servidores de la plataforma a usar
17 registry_id = "RaspbPi"
18 device_id = "RaspberryPi"
19
20 #---Certificados---#
21 private_key_file = 'Certificados/rsa_private-Google.pem'
22 algorithm = "RS256" # Algoritmo de encriptacion para
la creacion del Json Web Token
23 ca_certs = 'Certificados/rootsGoogle.pem'
24
25 mqtt_bridge_hostname = "mqtt.googleapis.com" # Direccion de la plataforma
26 mqtt_bridge_port = "8883" # Puerto MQTT sobre SSL
27
28 connflag = False # Bandera para la conexion
29
30 ##### Fin Variables establecer la conexion #####
31
32
33 # Entra cuando se establece la conexion con la plataforma (llega el mensaje CONNACK)
34 def on_connect(unused_client, unused_userdata, unused_flags, rc):
35     # Se notifica al usuario que se esta conectando
36     print("\n" + "Conectando a Google IoT Core")
37     # Resultado de la conexion
38     print("Resultado: " + error_str(rc) + "\n")
39
40     # Se comprueba si el resultado ha sido satisfactorio
41     if rc == 0:
42         # Bandera que notifica conexion
establecida
43         global connflag
44         connflag = True
45
46 # Se llama cuando el cliente se desconecta de la plataforma
47 def on_disconnect(unused_client, unused_userdata, rc):
48     # rc distinto de 0 implica desconexion inesperada
49     if rc != 0:
50         print("Desconexion inesperada")
51
52         print("Desconectando de la plataforma")
53         print("Resultado: " + error_str(rc) + "\n")
54         global connflag
55         # Actualizacion de la bandera de conexion
56         connflag = False
57
58
59 # Muestra informacion sobre el intercambio de mensajes, para depurar
60 def on_log(client, userdata, level, buf):
61     print(buf)
62
63 # Devuelve los codigos de errores de paho en formato cadena
64 def error_str(rc):
65     return '{}: {}'.format(rc, mqtt.error_string(rc))
66
67

```

```

/home/pi/Documents/Plataformas/PruebaConjunto/Google/google.py
Página 2 de 3 mar 10 jul 2018 10:20:44 CEST
68 #----- Creacion del Json Web Token para conseguir una comunicacion segura por la red -----#
69 def create_jwt(project_id, private_key_file, algorithm):
70
71     token = {
72         # Momento en el que se crea el token
73         'iat': datetime.datetime.utcnow(),
74         # Tiempo en el que expirara el token
75         'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
76         # Este campo debe ser siempre el id del proyecto
77         'aud': project_id
78     }
79
80     # Lee el fichero de clave privada
81     with open(private_key_file, 'r') as f:
82         private_key = f.read()
83
84     print("Creando el JWT usando el algoritmo {} y el fichero {}".format(
85         algorithm, private_key_file) + "\n")
86
87     return jwt.encode(token, private_key, algorithm=algorithm)
88 #----- Fin JWT -----#
89
90 def main(subirDatosPropios, datos):
91
92     # Se inicializa el cliente MQTT, Google IoT Core necesita el id de cliente con el siguiente formato
93     client = mqtt.Client(client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(
94         project_id, cloud_region, registry_id, device_id), protocol=mqtt.MQTTv311)
95
96     client.on_connect = on_connect
97     client.on_disconnect = on_disconnect
98     client.on_log = on_log
99     client.error_str = error_str
100
101     # En Google Cloud IoT Core, el parametro username se ignora
102     # El campo contraseña se utiliza para transmitir el Json Web Token (JWT)
103     client.username_pw_set(username='unused', password=create_jwt(project_id, private_key_file, algorithm))
104
105     # Configuracion tls para comunicacion segura por la red
106     client.tls_set(ca_certs=ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)
107
108     # Se realiza la conexion con Google Cloud IoT Core
109     client.connect(mqtt_bridge_hostname, mqtt_bridge_port)
110
111     # Procesa los eventos de red
112     client.loop_start()
113
114     # Tema en el que se publicaran los mensajes
115     sub_topic = 'state'
116     mqtt_topic = '/devices/{}/{}'.format(device_id, sub_topic)
117
118     # Se accede entonces al sensor para obtener datos
119     if subirDatosPropios == 0:
120         # Espera de dos seg para establecer la conexion
121         sleep(2)
122
123         if connflag == True:
124
125             # Se accede al fichero temp, donde se almacenan los valores de temperatura
126             fichero = open("/sys/class/thermal/thermal_zone0/temp")
127
128             # Se leen los 5 primeros caracteres
129             temp = fichero.read(5)
130
131             # Se divide entre 1000 por el formato
132             temp = float(temp)/1000

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/Google/google.py
```

```
Página 3 de 3
```

```
mar 10 jul 2018 10:20:44 CEST
```

```

132
133     # Se forma la cadena a enviar
134     payload = str(temp) + " Grados Centigrados"
135
136     # Registro de la fecha de la consulta, %c establece fecha y hora
137     fecha = time.strftime("%c")
138
139     # Se convierte a formato Json para permitir el almacenamiento en la
140     # base de datos de Google "BigQuery"
141     datosEnviar = {'Cputemp': payload, 'Fecha': fecha}
142     datosJson = json.dumps(datosEnviar)
143
144     # Publicacion de los datos en el Tema establecido y con calidad de
145     # servicio (qos) 1, se publica "al menos 1 vez"
146     client.publish(mqtt_topic, datosJson, qos=1)
147     print("Datos '" + datosJson + "' publicados\n")
148
149     else:
150         print("No se pudo establecer la conexion\n")
151
152     # Se envia a la nube la cadena especificada por el usuario
153     else:
154         # Pausa de dos seg para la conexion
155         sleep(2)
156
157         print("Publicando datos\n")
158
159         # Se comprueba que haya conexion
160         if connflag == True:
161             fecha = time.strftime("%c")
162
163             # Los datos ahora son los especificados por el usuario mediante
164             # teclado
165             payload = datos
166
167             # Se convierte a Json
168             datosEnviar = {'Cputemp': payload, 'Fecha':
169                             fecha}
170
171             datosJson = json.dumps(datosEnviar)
172
173             # Se publican los datos proporcionados por el
174             # usuario
175             client.publish(mqtt_topic, datosJson, qos=1)
176             print("Datos '" + datosJson + "' publicados\n")
177
178         else:
179             print("No se pudo establecer conexion\n")
180
181         # Desconexion con la plataforma
182         client.disconnect()
183
184         # Espera de dos seg para dar tiempo a la desconexion
185         sleep(2)
186
187 if __name__ == '__main__':
188     main(subirDatosPropios, datos)
189

```


Código Microsoft:

```

/home/pi/Documents/Plataformas/PruebaConjunto/Microsoft/microsoft.py
Página 1 de 3 mar 10 jul 2018 10:05:33 CEST
1  #!/usr/bin/env python
2
3  from paho.mqtt import client as mqtt
4  import ssl
5  import time
6  import json
7  from time import sleep
8
9  ##### Necesarios para generar el token #####
10 from base64 import b64encode, b64decode
11 from hashlib import sha256
12 from urllib import quote_plus, urlencode
13 from hmac import HMAC
14 #####
15
16 # Nombre del certificado Root a usar
17 Root_cert = "Certificados/digicert.cer"
18
19 # Identificador del dispositivo en la plataforma
20 device_id = "RaspberryP"
21
22 # Clave para la conexión
23 Key = "3w0M2dKonKBotF92r4ZR6BTSQ54T0uV9JB8kHfF9gkw="
24
25 # Nombre del recurso IoT Hub creado desde la plataforma
26 iot_hub_name = "RaspberryFred"
27
28 # Bandera estado de la conexión
29 connflag = False
30
31 def on_connect(client, userdata, flags, rc):
32     print ("\n" + "Conectado a la nube Azure de Microsoft con resultado: " +
33           error_str(rc) + "\n")
34
35     # Si no hay errores en la conexión se actualiza la bandera connflag
36     if rc == 0:
37         global connflag
38         connflag = True
39
40 def on_disconnect(client, userdata, rc):
41     print ("Desconectado de la plataforma con resultado: " + error_str(rc) + "\n")
42     global connflag
43     connflag = False
44
45 # Muestra información sobre el intercambio de mensajes, para depurar
46 def on_log(client, userdata, level, buf):
47     print(buf)
48
49 # Convierte los errores de paho a cadena
50 def error_str(rc):
51     return '{}: {}'.format(rc, mqtt.error_string(rc))
52
53 ##### Creación del token para autenticar la conexión #####
54 def generate_sas_token(uri, key, policy_name, expiry):
55     #time.time() devuelve los segundos desde las 12:00 am de Enero 1970
56     ttl = time.time() + expiry
57     sign_key = "%s\n%d" % ((quote_plus(uri)), int(ttl))
58     signature = b64encode(HMAC(b64decode(key), sign_key, sha256).digest())
59
60     rawtoken = {
61         'sr': uri,
62         'sig': signature,
63         'se': str(int(ttl))
64     }
65
66     if policy_name is not None:
67         rawtoken['skn'] = policy_name
68
69     return 'SharedAccessSignature ' + urlencode(rawtoken)

```

```

/home/pi/Documents/Plataformas/PruebaConjunto/Microsoft/microsoft.py
Página 2 de 3 mar 10 jul 2018 10:05:33 CEST
70 #####
71
72 def main(subirDatosPropios, datos):
73
74     # Inicializacion del cliente con protocolo mqtt version 3.1.1
75     client = mqtt.Client(client_id=device_id, protocol=mqtt.MQTTv311)
76
77     client.on_connect = on_connect
78     client.on_disconnect = on_disconnect
79     #client.on_log = on_log
80     client.error_str = error_str
81
82     # Se genera el token permitiendo el acceso al dispositivo indicado como primer
83     # parametro,
84     # se pasa tambien la clave y el tiempo en el que expirara el token.
85     sas_token = generate_sas_token("RaspberryFred.azure-devices.net/RaspberryP",
86     Key, None, 120)
87     client.username_pw_set(username=iot_hub_name+".azure-devices.net/" +
88     device_id, password=sas_token)
89
90     #Configuracion tls
91     client.tls_set(ca_certs=Root_cert, certfile=None, keyfile=None,
92     cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1, ciphers=None)
93
94     # Asegura que se utilicen los certificados
95     client.tls_insecure_set(False)
96
97     # Conexion con puerto seguro 8883
98     client.connect(iot_hub_name+".azure-devices.net", port=8883)
99
100     # Inicia un hilo en segundo plano para llamar a loop() que procesa los eventos
101     # de red
102     client.loop_start()
103
104     if subirDatosPropios == 0:
105         #Pausa de 2 seg para dar tiempo al establecimiento de conexion
106         sleep(2)
107
108         if connflag == True:
109
110             # Se accede al fichero temp, donde se almacenan los valores de
111             # temperatura
112             fichero = open("/sys/class/thermal/thermal_zone0/temp")
113
114             temp = fichero.read(5)
115
116             # Se divide entre 1000 por el formato
117             temp = float(temp)/1000
118
119             # Se forma la cadena a enviar
120             cadena = str(temp) + " Grados Centigrados"
121
122             # Fecha de la consulta (%c) devuelve fecha y hora actual
123             fecha = time.strftime("%c")
124
125             # Datos a enviar
126             datosCadena = {'Datos':cadena, 'Fecha': fecha}
127
128             # En formato Json
129             datosJson = json.dumps(datosCadena)
130
131             client.publish("devices/" + device_id + "/messages/events/",
132             datosJson, qos=1)
133             print("Datos publicados: " + cadena + " en la fecha " + fecha + "\n")
134
135         else:
136             print("No se pudo establecer conexion\n")
137
138     elif subirDatosPropios == 1:
139         #Pausa de 2 seg para dar tiempo al establecimiento de conexion

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/Microsoft/microsoft.py
Página 3 de 3 mar 10 jul 2018 10:05:33 CEST
```

```
133     sleep(2)
134
135     if connflag == True:
136
137         fecha = time.strftime("%c")
138
139         # En este caso los datos son los proporcionados por el usuario
140         datosCadena = {'Datos':datos, 'Fecha':fecha}
141         datosJson = json.dumps(datosCadena)
142
143         client.publish("devices/" + device_id + "/messages/events/",
144                       datosJson, qos=1)
145         print("Datos del usuario publicados: " + datos + " en la fecha " +
146               fecha + "\n")
147
148     else:
149         print("No se pudo establecer conexion\n")
150
151     client.disconnect()
152
153     # Espera de dos seg para dar tiempo a la desconexion
154     sleep(2)
155
156 if __name__ == "__main__":
157     main(subirDatosPropios, datos)
```

Código servicio.py:

```

/home/pi/Documents/Plataformas/PruebaConjunto/servicio.py
Página 1 de 3
mar 10 jul 2018 10:05:55 CEST
1  from Carriots import carriots
2  from Amazon import amazon
3  from Google import google
4  from Microsoft import microsoft
5
6  from time import sleep
7
8
9  def main():
10     print("\nTemperatura de la CPU\n")
11     print("Elija un Servicio en la Nube para monitorizar la temperatura CPU de su
12     dispositivo:\n")
13     print("Pulse 1 para subir los datos a AWS Amazon\n"+
14           "Pulse 2 para subir los datos a Carriots\n"+
15           "Pulse 3 para subir los datos a Azure de Microsoft\n"+
16           "Pulse 4 para subir los datos a Google IoT Core\n"+
17           "Pulse 5 para subir los datos a todas las nubes mencionadas\n"+
18           "Pulse 6 para visualizar de nuevo las opciones\n" +
19           "\n7 para Salir\n")
20
21     # Recoge la opcion escogida por el usuario mediante entrada estandar "input()"
22     entrada = 0
23
24     # Variable para establecer si se desean leer los datos del sensor o enviar
25     leerDatos=1
26     datos=""
27     while entrada != 7:
28         entrada = input("Opcion numero: ")
29         print("\n")
30
31         if entrada == 1:
32             subirDatosPropios = input("Desea subir sus propios datos a la
33             nube?\nPulse 1 para Si --- Pulse 0 para No: ")
34
35             # Se subiran los datos del sensor
36             if subirDatosPropios == 0:
37                 # En este caso 'datos' se envia como cadena vacia
38                 datos=""
39
40             # Metodo main de aws_iot_pub para subir datos a la nube de
41             Amazon
42             amazon.main(subirDatosPropios, datos)
43
44             # Se subira la cadena especificada por el usuario
45             elif subirDatosPropios == 1:
46                 datos = raw_input("Introduzca la cadena a enviar:")
47
48             # Se llama al metodo main de aws_iot_pub para subir datos del
49             usuario a la nube de Amazon
50             amazon.main(subirDatosPropios, datos)
51
52             else:
53                 print("Respuesta con formato incorrecto, seleccione de nuevo un
54                 servicio\n")
55
56             print("\n")
57
58         elif entrada == 2:
59             subirDatosPropios = input("Desea subir sus propios datos a la
60             nube?\nPulse 1 para Si --- Pulse 0 para No: ")
61
62             if subirDatosPropios == 0:
63                 datos=""
64
65             # Se llama al metodo main de carriotsMQTT para subir datos del
66             sensor a la nube de Carriots
67             carriots.main(subirDatosPropios, datos)
68
69             elif subirDatosPropios == 1:
70
71     - 1 -

```

/home/pi/Documents/Plataformas/PruebaConjunto/servicio.py

Página 2 de 3

mar 10 jul 2018 10:05:55 CEST

```

63         # raw_input para evitar introducir "" al usuario
64         datos = raw_input("Introduzca la cadena a enviar:")
65
66         # Se llama al metodo main de carriotsMQTT para subir datos del
67         usuario a la nube de Carriots
68         carriots.main(subirDatosPropios, datos)
69
70     else :
71         print("Respuesta con formato incorrecto, seleccione de nuevo un
72         servicio\n")
73
74     print("\n")
75
76 elif entrada == 3:
77     subirDatosPropios = input("Desea subir sus propios datos a la
78     nube?\nPulse 1 para Si --- Pulse 0 para No: ")
79
80     if subirDatosPropios == 0:
81         datos=""
82         microsoft.main(subirDatosPropios, datos)
83
84     elif subirDatosPropios == 1:
85         datos = raw_input("Introduzca la cadena a enviar:")
86         microsoft.main(subirDatosPropios, datos)
87
88     else:
89         print("Opcion seleccionada no valida, pruebe de nuevo\n")
90
91     print("\n")
92
93 elif entrada == 4:
94     subirDatosPropios = input("Desea subir sus propios datos a la
95     nube?\nPulse 1 para Si --- Pulse 0 para No: ")
96
97     if subirDatosPropios == 0:
98         datos = ""
99         google.main(subirDatosPropios, datos)
100
101     elif subirDatosPropios == 1:
102         # raw_input para evitar introducir "" al usuario
103         datos = raw_input("Introduzca la cadena a enviar:")
104         google.main(subirDatosPropios, datos)
105
106     else:
107         print("Respuesta con formato incorrecto, seleccione de nuevo un
108         servicio\n")
109
110     print("\n")
111
112 elif entrada == 5:
113     subirDatosPropios = input("Subir datos propios?\nPulse 1 para Si ---
114     Pulse 0 para No: ")
115
116     if subirDatosPropios == 0:
117         datos = ""
118
119         #Subiendo datos a la nube de Amazon
120         print("-----AMAZON-----\n")
121         amazon.main(subirDatosPropios, datos)
122         sleep(1)
123
124         #Subiendo datos a la nube de Google
125         print("-----GOOGLE-----\n")
126         google.main(subirDatosPropios, datos)
127         sleep(1)
128
129         #Subiendo datos a la nube de Microsoft
130         print("-----MICROSOFT-----\n")
131         microsoft.main(subirDatosPropios, datos)
132         sleep(1)

```

```
/home/pi/Documents/Plataformas/PruebaConjunto/servicio.py
```

```
Página 3 de 3
```

```
mar 10 jul 2018 10:05:55 CEST
```

```

127
128         #Subiendo datos a la nube de Carriots
129         print("-----CARRIOTS-----\n")
130         carriots.main(subirDatosPropios, datos)
131         sleep(1)
132
133         print("\n")
134
135     elif subirDatosPropios == 1:
136
137         datos = raw_input("Introduzca la cadena a enviar:")
138
139         #Subiendo datos a la nube de Amazon
140         print("-----AMAZON-----\n")
141         amazon.main(subirDatosPropios, datos)
142         sleep(1)
143
144         #Subiendo datos a la nube de Carriots
145         print("-----CARRIOTS-----\n")
146         carriots.main(subirDatosPropios, datos)
147         sleep(1)
148
149         #Subiendo datos a la nube de Google
150         print("-----GOOGLE-----\n")
151         google.main(subirDatosPropios, datos)
152         sleep(1)
153
154         #Subiendo datos a la nube de Microsoft
155         print("-----MICROSOFT-----\n")
156         microsoft.main(subirDatosPropios, datos)
157         sleep(1)
158
159         print("\n")
160
161     else:
162         print("Respuesta con formato incorrecto, seleccione de nuevo un
servicio\n")
163
164     elif entrada == 6:
165
166         print("Pulse 1 para subir los datos a AWS Amazon\n"+
"Pulse 2 para subir los datos a Carriots\n"+
"Pulse 3 para subir los datos a Azure de Microsoft\n"+
"Pulse 4 para subir los datos a Google IoT Core\n"+
"Pulse 5 para subir los datos a todas las nubes mencionadas\n"+
"Pulse 6 para visualizar de nuevo las opciones\n" +
"\n0 para Salir\n")
173
174     elif entrada==7:
175         print("Saliendo...")
176
177     else:
178         print("Opcion seleccionada no valida!, seleccione un servicio de la
lista\n")
179
180 if __name__ == '__main__':
181     main()
182
183
184
185

```